



Charlotte Langlais
Enseignante-chercheure
Département MEE
Tél : 02.29.00.15.34
E-mail : charlotte.langlais@imt-atlantique.fr

Module FIP ELP111 - Fonctions électronique logiques « Recueil des classes inversées et BE2 »

UV FIP ELP 100

Fiches séances

Année 2025-2026

Responsable module : Fabrice Seguin - Charlotte Langlais



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Table des matières

PCD1 - Arithmétique binaire	5
PCD2 - Algèbre de Boole	27
PCD3 - Simplification des fonctions logiques	41
PCD4 - Circuit combinatoire	61
PCD5 - Additionneur binaire	95
BE2 - Distributeur de boissons chaudes	117



ELP111

Fonctions électroniques logiques et analogiques

Fiche séance

PCD1 ARITHMETIQUE BINAIRE

PCD1 ARITHMETIQUE BINAIRE	1
1. LES OBJECTIFS D'APPRENTISSAGE	2
2. LES CONCEPTS	3
2.1 REPRESENTATION POLYNOMIALE D'UN NOMBRE, EXPRESSION D'UN NOMBRE DANS UNE BASE	3
2.2 REPRESENTATION D'UN NOMBRE POSITIF ET CONVERSION ENTRE BASES	3
2.2.1 Base b vers base 10	4
2.2.2 Base 2 vers base 8 ou 16	4
2.2.3 Base 10 vers base b	4
2.2.3.1. Nombres entiers positifs (2 méthodes disponibles)	4
2.2.3.2. Nombres réels positifs (ou fractionnaires)	6
2.2.3.3. Maintien de la résolution pour les nombres réels positifs	7
2.3 REPRESENTATION BINAIRE DES NOMBRES SIGNES (COMPLEMENT A 2)	7
2.3.1 Représentation en complément à 2	7
2.3.2 Extension d'un nombre codé en CA2	9
2.4 ADDITION ET SOUSTRACTION BINAIRE	9
2.5 REPRESENTATION BINAIRE DES NOMBRES REELS	11
2.5.1 Codage en virgule fixe	11
2.5.2 Codage en virgule flottante	11
2.6 REPRESENTATION DES CARACTERES ALPHANUMERIQUES	13
3. LES EXERCICES D'APPLICATION	14
3.7 LES NOMBRES ENTIERS, REPRESENTATION ET CONVERSION DE BASES	14
3.8 LES NOMBRES ENTIERS SIGNES, COMPLEMENT A 2	14
3.8.1 Représentation binaire en CA2	14
3.8.2 Conversion d'un nombre en CA2 en sa valeur décimale	14
3.9 LES NOMBRES REELS	14
3.9.1 Conversion de bases	14
3.9.2 Représentation de nombres en virgule flottante dans les processeurs (norme IEEE 754)	15
4. POUR ALLER PLUS LOIN	16
REPRESENTATION SIGNE+AMPLITUDE (OU MODULE)	16
REPRESENTATION BINAIRE DECALEE	16
CLASSIFICATION DES CODES BINAIRES	17
Codes pondérés	17
Le code binaire pur et ses dérivés (octal, hexadécimal)	17
Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal)	17
Codes non pondérés	17
Code excédent 3 ou excess 3	18
Code binaire réfléchi ou code de Gray	18
Codes redondants	20

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD1 Arithmétique binaire

1. LES OBJECTIFS D'APPRENTISSAGE

OA1 Représenter un nombre dans n'importe quelle base arithmétique.

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

2.1 REPRESENTATION POLYNOMIALE D'UN NOMBRE, EXPRESSION D'UN NOMBRE DANS UNE BASE

De manière générale tout nombre N exprimé dans une base b peut se décomposer sous la forme polynomiale suivante :

$$N_{(b)} = S(a_nb^n + a_{n-1}b^{n-1} + \dots + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + \dots + a_{-m}b^{-m}) \quad (\text{équation 1})$$

avec

- S est le **signe** du nombre
- a_i est le **symbole de rang i** , $a_i \in \mathbb{N}$ et $0 \leq a_i < b$
- a_n est le **symbole de poids le plus fort** (MSB : Most Significant Bit si $b = 2$), et a_{-m} est le **symbole de poids le plus faible** (LSB : Least Significant Bit si $b = 2$)

Le nombre $N_{(b)}$ s'exprime en numérotation de position par $S a_n a_{n-1} \dots a_0, a_{-1} \dots a_{-m}$. Les symboles $a_n a_{n-1} \dots a_0$ et $a_{-1} \dots a_{-m}$ représentent respectivement la **partie entière** et la **partie fractionnaire** de N .

On appelle **dynamique** ou **amplitude de codage** d'une représentation la différence entre le plus grand nombre et le plus petit nombre représentables. On appelle **résolution** ou **précision** d'une représentation la différence entre deux nombres consécutifs. A titre d'exemple pour une représentation décimale d'entiers positifs sur 5 chiffres, la dynamique est égale à 99999 et la résolution est égale à 1.

2.2 REPRESENTATION D'UN NOMBRE POSITIF ET CONVERSION ENTRE BASES

Les bases de numération les plus utilisées sont la base **décimale** ($b = 10$), la base **binaire** ($b = 2$), et les bases dérivées de la base binaire : base **octale** ($b = 8$) et base **hexadécimale** ($b = 16$).

La numération binaire utilise les 2 bits 0 et 1, la numération octale utilise 8 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, et la numération hexadécimale utilise 16 symboles : 0, 1, 2, ..., 9, A, B, C, D, E, F (les symboles A à F ont pour équivalents décimaux les nombres 10 à 15).

Le système binaire et ses dérivés sont ceux utilisés pour le codage des informations dans les systèmes numériques. La base 16 est couramment utilisée car elle peut être considérée comme une écriture condensée de l'écriture binaire et, par conséquent, sa conversion vers le binaire est particulièrement aisée.

Les conversions les plus utilisées sont les suivantes

- base b vers base 10
- base 10 vers base b
- base 2 vers base 2^n (8 ou 16)
- base 2^n (8 ou 16) vers base 2

2.2.1 Base b vers base 10

Pour convertir un nombre d'une base b vers la base décimale, on utilise la **méthode dite des additions** qui consiste à utiliser la représentation du nombre sous forme polynomiale (équation 1).

Exemple 1 : conversion du nombre binaire entier $N_{(2)} = 1101\ 0011_{(2)}$ en base 10.

$$N = 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = 128 + 64 + 16 + 2 + 1 = 211_{(10)}$$

Exemple 2 : conversion du nombre binaire fractionnaire $N_{(2)} = 110011,1001_{(2)}$ en base 10

$$N = 1.2^5 + 1.2^4 + 1.2^1 + 1.2^0 + 1.2^{-1} + 1.2^{-4} = 51,5625_{(10)}$$

Exemple 3 : conversion du nombre octal entier $N_{(8)} = 4513_{(8)}$ en base 10

$$N = 4.8^3 + 5.8^2 + 1.8^1 + 3.8^0 = 2379_{(10)}$$

Exemple 4 : conversion du nombre hexadécimal fractionnaire $N_{(16)} = 1B20,8_{(16)}$ en base 10

$$N = 1.16^3 + 11.16^2 + 2.16^1 + 8.16^{-1} = 6944,5_{(10)}$$

N.B. : La méthode des additions requiert la connaissance des puissances successives de la base de départ.

2.2.2 Base 2 vers base 8 ou 16

Il s'agit d'utiliser la représentation du nombre sous sa forme polynomiale (équation 1), et à factoriser par des puissances de 8 ou de 16.

Exemple 1 : conversion du nombre entier $N_{(2)} = 1101\ 0011$ en base 8

$$N_{(2)} = 1101\ 0011 = 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 1.2^0$$

$$N_{(2)} = 8^2.(1.2^1 + 1.2^0) + 8^1.(0.2^2 + 1.2^1 + 0.2^0) + 8^0.(0.2^2 + 1.2^1 + 1.2^0)$$

$$N_{(2)} = 3.8^2 + 2.8^1 + 3.8^0 = 323_{(8)}$$

Visuellement, cela peut se représenter par des regroupements de bits de taille 3 :

$$N_{(2)} = \underbrace{1101}_{3}\ \underbrace{001}_{2}\ \underbrace{1}_{3}$$

2.2.3 Base 10 vers base b

2.2.3.1. Nombres entiers positifs (2 méthodes disponibles)

Il existe deux méthodes pour convertir un nombre entier positif :

- Méthode des divisions successives.
- Méthode des soustractions successives : on travaille directement à partir des puissances de la base d'arrivée. Cette méthode est rapide à appliquer en base 2 où les puissances de 2 sont connues facilement.

(a) Méthode des divisions successives

Pour effectuer une conversion d'un entier décimal dans une autre base on applique la **méthode des divisions successives** : on effectue des divisions successives du nombre par cette base, les restes successifs forment alors le nombre converti.

A titre d'exemple, dans le cas d'une conversion d'un nombre décimal en son équivalent binaire, on réalisera des divisions successives par 2. Les restes de ces divisions formeront le nombre converti dans la base 2.

Exemple 1 : conversion de $N_{(10)} = 52$ en base 2

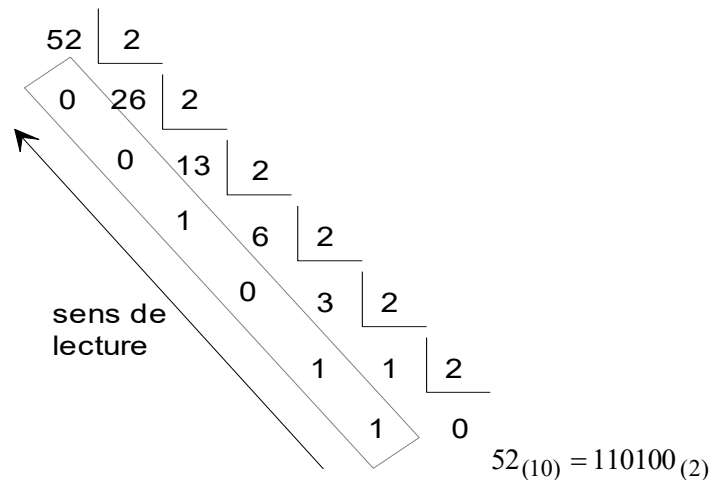


figure 1.1 : conversion de $52_{(10)}$ en base 2 par divisions successives par 2

Exemple 2 : conversion de $N_{(10)} = 90$ en base 8

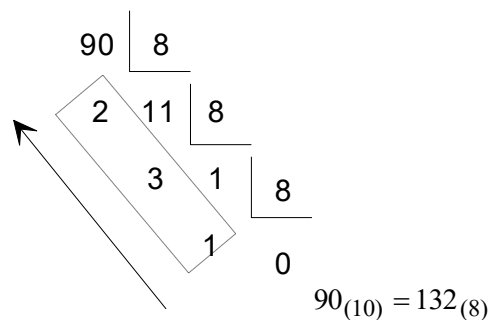


figure 1.2 : conversion de $90_{(10)}$ en base 8 par divisions successives par 8

Chaque division revient à opérer un décalage à droite d'une position et permet ainsi d'isoler un bit dans la partie fractionnaire.

(b) Méthodes des soustractions successives

Si les puissances successives de la base d'arrivée sont connues, on peut également, plutôt que d'utiliser la méthode précédente, effectuer la transformation par **soustractions successives** de ces puissances. Cette méthode est illustrée sur les deux exemples traités précédemment.

Exemple 1 : conversion de $N_{(10)} = 52$ en base 2

$32(=2^5) \leq N < 64(=2^6)$, on peut donc retrancher 32 à N : $N = 32 + 20$,
 $16(=2^4) \leq 20 < 32(=2^5)$, on peut retrancher 16 : $N = 32 + 16 + 4$,
 4 est une puissance de 2, l'itération est donc terminée. On en déduit :
 $N = 2^5 + 2^4 + 2^2 = 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 0.2^0 = 110100_{(2)}$.

Exemple 2 : conversion de $N_{(10)} = 90$ en base 8

$64(=8^2) \leq N < 512(=8^3)$, on retranche 64 à N : $N = 64 + 26$,
 $8 \leq 26 < 64(=8^2)$, on retranche 8 : $N = 64 + 8 + 18$,
 on peut de nouveau retrancher 2 fois 8 à 18, on obtient alors : $N = 64 + 3.8 + 2$. Puisque 2 est inférieur à 8, l'itération est terminée, d'où $N = 1.8^2 + 3.8^1 + 2.8^0 = 132_{(8)}$.

2.2.3.2. Nombres réels positifs (ou fractionnaires)

Pour convertir un nombre fractionnaire de la base 10 vers une autre base, il faut procéder en deux étapes. La partie entière du nombre est convertie comme indiqué précédemment ; la partie fractionnaire du nombre est convertie par **multiplications successives** : on multiplie successivement la partie fractionnaire par la base cible, en retenant les parties entières qui apparaissent au fur et à mesure.

Exemple 1 : conversion de $N_{(10)} = 12,925$ en base 2

- partie entière : $12_{(10)} = 1100_{(2)}$
- partie fractionnaire :

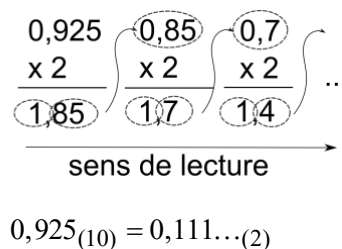


figure 1.3 : conversion de $0,925_{(10)}$ en base 2 par multiplications successives

Finalement $12,925_{(10)} = 1100,111\dots_{(2)}$

Exemple 2 : conversion de $N_{(10)} = 0,45$ en base 8

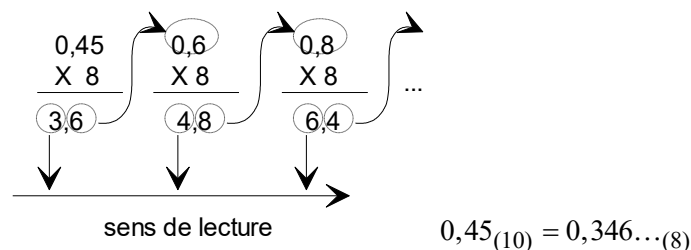


figure 1.4 : conversion de $0,45_{(10)}$ en base 8 par multiplications successives

Il est visible sur les deux exemples précédents que la conversion peut ne pas se terminer et que l'on obtient, en s'arrêtant à un nombre fini de positions une approximation de la représentation du nombre.

2.2.3.3. *Maintien de la résolution pour les nombres réels positifs*

Soit $(a_n a_{n-1} \dots a_0, a_{-1} \dots a_{-m})_{(10)}$ et $(a_p a_{p-1} \dots a_0, a_{-1} \dots a_{-k})_{(b)}$ les numérotations de position d'un même nombre N exprimé respectivement en base 10 et en base b . La résolution d'une base b est définie comme étant la différence entre 2 nombres consécutifs dans cette base. Elle est donc égale à b^{-k} .

Ainsi, la résolution est conservée lors du passage de la base 10 à la base b si et seulement si la résolution du nombre transformé en base b est inférieur ou égal à la résolution de ce nombre en base 10. $b^{-k} \leq 10^{-m}$, c'est-à-dire si $k \log b \geq m \log 10$, soit

$$k \geq m \frac{\log 10}{\log b}$$

Exemple 1 : pour conserver la résolution lors du passage de $0,925_{(10)}$ en base 2, il faut garder $k \geq 3 \frac{\log 10}{\log 2} \approx 9,97$, soit 10 bits après la virgule.

Exemple 2 : pour conserver la résolution lors de la conversion de $0,45_{(10)}$ en base 8, il faut garder $k \geq 2 \frac{\log 10}{\log 8} \approx 2,2$, soit 3 bits après la virgule.

2.3 REPRESENTATION BINAIRE DES NOMBRES SIGNES (COMPLEMENT A 2)

Les systèmes numériques doivent être capables de traiter des nombres positifs et négatifs. L'utilisation d'une représentation signée suppose l'utilisation d'un **format** (nombre de bits) fixé au préalable.

2.3.1 Représentation en complément à 2

Le **complément à 2** est le mode de représentation le plus utilisé en arithmétique binaire et donc dans les ordinateurs pour coder les nombres entiers.

Dans cette représentation, les **nombres positifs** se représentent par leur valeur binaire naturelle. Par exemple +6 est représenté par 0000 0110 sur un format de 8 bits.

La représentation **des nombres négatifs** s'obtient comme suit :

- On part de la représentation binaire naturelle de l'opposé arithmétique du nombre à coder (nombre positif),
- On calcule son **complément à 1** (CA1) ou **complément restreint**. Celui-ci est obtenu en inversant tous ses bits,
- On en déduit son **complément à 2** (CA2) ou **complément vrai** en ajoutant 1 au niveau du LSB, c'est-à-dire en réalisant l'addition binaire du complément à 1 et de 1. L'addition binaire est décrite succinctement à la section 2.4.

Exemple : représentation de -5 en CA2 sur un format de 8 bits

- Représentation binaire naturelle de +5 : $5 = 0000\ 0101$,

- CA1 de +5 : $\bar{5} = 1111\ 1010$,
- CA2 de +5 : $-5 = 1111\ 1011$.

On identifie le CA2 d'un nombre à son opposé arithmétique, ainsi $CA2(A) = -A$.

En effet, soit $A = a_{n-1} \dots a_1 a_0$, alors $\bar{A} = \bar{a}_{n-1} \dots \bar{a}_1 \bar{a}_0$, et donc $A + \bar{A} = 11 \dots 11$, soit $A + \bar{A} = 2^n - 1$, et $-A = \bar{A} + 1 = CA2(A)$. L'addition de 2 nombres sur un format fixé de n bits étant toujours réalisée modulo 2^n (cf. section 1.4).

La représentation en complément à 2 présente les caractéristiques suivantes :

- Le principe d'obtention de l'opposé d'un nombre négatif est le même que celui permettant d'obtenir l'opposé d'un nombre positif,
- Le nombre 0 a une représentation unique,
- Un format sur n bits permet de coder en CA2 les nombres N vérifiant

$$-2^{n-1} \leq N \leq +2^{n-1} - 1$$

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$\bar{N}_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	0000	1111	0000	0
1	0001	1110	1111	-1
2	0010	1101	1110	-2
3	0011	1100	1101	-3
4	0100	1011	1100	-4
5	0101	1010	1011	-5
6	0110	1001	1010	-6
7	0111	1000	1001	-7
			1000	-8

tableau 1 : représentation en complément à 2 sur 4 bits

On peut ainsi représenter des nombres compris entre -4 et +3 sur un format de 4 bits, entre -16 et +15 sur un format de 5 bits, entre -32 et +31 sur un format de 6 bits, entre -64 et +63 sur un format de 7 bits, etc.

Remarques :

- Le bit de poids fort (MSB) est représentatif du bit de signe, mais il est traité comme les autres bits dans les opérations arithmétiques : si MSB = 0 le nombre est positif, si MSB = 1 le nombre est négatif.
- Le nombre $+2^{n-1}$ n'est pas représenté. En effet, dans le cas où $n = 4$, le calcul du CA2 de 1000 donne $CA2(-8) = -(-8) = CA2(1000) = 0111 + 1 = 1000 = (-8)_{(10)}$. Ce qui est arithmétiquement incorrect car 0 est le seul entier à être son propre opposé. On a donc choisi de supprimer la représentation du nombre $+2^{n-1}$, un MSB à 1 étant représentatif d'un nombre négatif.
- La représentation des nombres négatifs peut être recalculée rapidement en observant que les nombres négatifs entre 1000 (-8) et 1111 (-1) correspondent à une translation du bloc des nombres codés en binaire naturel de 1000 (8) à 1111 (15) de 16 positions vers le bas. C'est la conséquence de l'addition modulo 2^n .

La représentation sur 4 bits (en CA2 signé et en binaire naturel non signé) est expliquée sur la figure 1.5.

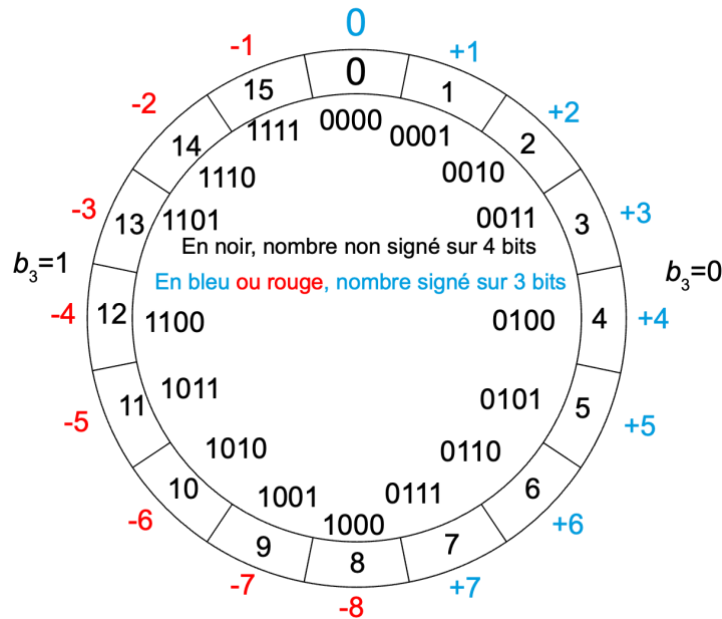


figure 1.5 : Représentation d'un nombre signé et non signé sur 4 bits.

2.3.2 Extension d'un nombre codé en CA2

L'extension d'un nombre codé sur n bits à un format sur $n+k$ bits est réalisée comme suit :

- Si le nombre est positif, on complète les k bits de poids forts par des 0. Par exemple,

$$\underbrace{0110}_{\substack{6_{(10)} \text{ codé} \\ \text{sur 4 bits}}}_{(CA2)} \xrightarrow{6 \text{ bits}} \underline{000110}_{(CA2)}$$

- Si le nombre est négatif, on complète les k bits de poids forts avec des 1. Par exemple,

$$\underbrace{1010}_{\substack{-6_{(10)} \text{ codé} \\ \text{sur 4 bits}}}_{(CA2)} \xrightarrow{6 \text{ bits}} \underline{111010}_{(CA2)}$$

2.4 ADDITION ET SOUSTRACTION BINAIRE

Dans toutes les bases, le principe de l'**addition** est similaire à celui de l'addition décimale : on additionne symbole par symbole en partant des poids faibles, et en propageant éventuellement une retenue. Des exemples sont donnés en fin de section.

Si le nombre de bits est fixé à n , alors on réalise une addition modulo 2^n (puisque'on ne peut pas aller au-delà de $2^n - 1$).

Par exemple, $A + (-A) = 2^n = 0 \bmod 2^n$

En effet, $A + (-A) = A + CA2(A) = A + \bar{A} + 1 = \underbrace{(1 \dots 1)}_{n \text{ bits}} + 1 = 10 \dots 0 = 2^n = 0 \bmod 2^n$ car 2^n n'est pas représentable sur n bits.

Exemple sur 3 bits : $+2 + (-2) = 010 + 101 + 1 = 111 + 1 = 1000 = 000 \bmod 2^3$

La **soustraction**, en arithmétique binaire, est le plus souvent appliquée sur des nombres signés. Dans ce cas, cette opération se ramène dans tous les cas à une addition.

Si le format des nombres est fixe, le résultat de l'addition peut donner lieu à un **dépassement de capacité**.

Dans le cas où les nombres **sont codés en binaire naturel**, le résultat de l'addition de 2 nombres binaires codés sur n bits peut dépasser la plus grande valeur codable sur n bits, qui est $2^n - 1$ en binaire naturel.

Dans le cas où les nombres **sont codés en complément à 2**, l'addition de 2 nombres exprimés sur n bits fournit toujours un résultat correct, sauf dans le cas où le résultat n'est pas représentable sur les n bits. Il y a alors **dépassement de capacité** lorsque les deux opérandes sont de même signe et que le résultat est de signe opposé.

Dans le registre d'état d'un ordinateur, deux indicateurs viennent renseigner le programmeur (ou le système d'exploitation) sur la **validité des résultats obtenus** : la **retenue** (carry C) et le **débordement** (overflow OVF). L'indicateur C signale la présence d'une retenue au niveau des poids forts; l'indicateur OVF indique que le résultat de l'opération n'est pas représentable dans le système du complément à 2 sur le format défini au départ.

Nous allons illustrer le positionnement de la retenue et du débordement par quatre exemples pour des nombres signés, codés en CA2 sur 8 bits :

$\begin{array}{r} + \quad 0000 \ 0110 \ (+6) \\ + \quad 0000 \ 0100 \ (+4) \\ \hline 0000 \ 1010 \ (+10) \\ \text{OVF}=0 \\ \text{C}=0 \\ \text{résultat correct} \end{array}$	$\begin{array}{r} + \quad 0111 \ 1111 \ (+127) \\ + \quad 0000 \ 0001 \ (+1) \\ \hline 1000 \ 0000 \ (-128) \\ \text{OVF}=1 \\ \text{C}=0 \\ \text{résultat incorrect} \end{array}$	$\begin{array}{r} \quad 0000 \ 0100 \ (+4) \\ + \quad 1111 \ 1110 \ (-2) \\ \hline 1 \ 0000 \ 0010 \ (+2) \\ \text{OVF}=0 \\ \text{C}=1 \text{ ignoré} \\ \text{résultat correct} \end{array}$	$\begin{array}{r} \quad 0000 \ 0100 \ (+4) \\ + \quad 1111 \ 1100 \ (-4) \\ \hline 1 \ 0000 \ 0000 \ (0) \\ \text{OVF}=0 \\ \text{C}=1 \text{ ignoré} \\ \text{résultat correct} \end{array}$
--	---	--	---

Exemple 1 : le résultat (+10) est codable sur 8 bits, le résultat est correct.

Exemple 2 : le résultat (+128) n'est pas codable sur 8 bits. Du fait de l'addition modulo 2^n , on obtient -128, le résultat est incorrect et OVF est positionné à 1 pour indiquer au système d'exploitation que le résultat est incorrect.

Exemple 3 : le résultat (+2) est codable sur 8 bits, le résultat est correct.

Exemple 4 : le résultat (0) est codable sur 8 bits, le résultat est correct.

Remarques :

- La retenue sortante (C) est toujours ignorée lorsque l'on manipule des nombres signés en CA2 et que le format des nombres est fixé
- La sortie OVF indique si le résultat est correct ou non ; si les 2 nombres sont positifs (resp. négatifs) et le résultat négatif (resp. positif) alors OVF = 1.
- Pour que le résultat d'une opération sur n bits soit correct dans la méthode du complément à 2, il faut que les retenues de rang n et de rang $n+1$ soient identiques (cf. Séance sur l'addition binaire).

$$N = (-1)^S \times 2^{C-127} \times 1, M \quad (\text{équation 3})$$

où la valeur de l'exposant e est donnée par $e = C - 127$. Ainsi, C peut s'exprimer en binaire naturel en s'affranchissant du problème de son signe. C est toujours positif. Le 1 précédent la mantisse n'est pas représenté en machine. On l'appelle le bit caché.

Cette représentation présente des singularités

- $C = '0..0'$ et $M = '0..0'$: nombre zéro,
- $C \neq '0..0'$ et $C \neq '1..1'$, float normalisé (ou normal)
 $N = (-1)^S \times 2^{C-127} \times 1, M$
- $C = '0..0'$ et $M \neq '0..0'$, float dénormalisé (très petit)
 $N = (-1)^S \times 2^{-127} \times 0, M$
- $C = '1..1'$ (255) nombre infini ($+\infty$ si $S = 0$, $-\infty$ si $S = 1$) si $M = '0..0'$, **NaN** (not a number) sinon.
 $N = (-1)^S \times 2^{128} \times 1, 0 = \pm\infty$

Exemple 1 : traduire le nombre décimal $1_{(10)}$ dans le format précédemment présenté.

Il faut le mettre sous la forme normalisée de l'équation 3, soit

$$1 = (-1)^0 \times 2^{127-127} \times 1,00 \dots 0$$

d'où $S = 0$, $C = 127$, et $M = 000\dots0$.

On en déduit la représentation en virgule flottante :

31	30	23	22	0
0	0111 1111	000000	0000	
signe	carac = +127	mantisse = 0		

soit $3F800000_{(16)}$.

Exemple 2 : traduire le nombre décimal $N = +4,75_{(10)}$ dans le format précédemment présenté.

On commence par convertir en binaire $\rightarrow N = 100,11$ sur 5 bits.

Puis on décale la virgule jusqu'à obtenir un nombre en 1,.. $\rightarrow N = 1,0011 \times 2^2$

Afin de conserver le nombre N , un décalage de la virgule de 2 positions à gauche doit être compensé par une multiplication du nombre par 2^2 .

On peut ainsi récupérer la mantisse $M = 00110..0$ sur 23 bits. Puis l'exposant $e = C - 127 = 2$ soit $C = 129 = 10000001$.

On en déduit la représentation en virgule flottante :

0	10000001	00110.....0
---	----------	-------------

La représentation en virgule flottante autorise une dynamique plus grande qu'en virgule fixe. Dans le format précédent, on peut coder des nombres dont la valeur absolue va de 10^{-38} à

10^{38} , et le standard d'écriture en virgule flottante double précision (sur 64 bits) permet de représenter des nombres atteignant 10^{300} .

2.6 REPRESENTATION DES CARACTERES ALPHANUMERIQUES

Certains codes peuvent avoir une signification non numérique. Le plus connu d'entre eux est le code ASCII (American Standard Code for Information Interchange), qui est utilisé pour représenter les caractères alphanumériques. Dans ce code, 128 combinaisons (lettres, chiffres, signes de ponctuation, caractères de contrôle, etc.) sont codées à l'aide de 7 bits de 00000000 à 01111111. Les transmissions asynchrones entre machines s'effectuant souvent sur un format de 8 bits, le dernier bit est alors utilisé pour contrôler la parité du message. Ainsi, sur les 8 bits de l'octet, celui de poids fort est placé à zéro.

La plus grande limitation de l'ASCII est donc le nombre de caractères codables. Un système de codage plus performant a donc été proposé : UCS (*Universal Character Set*), également appelé Unicode. Ce code permet de représenter n'importe quel caractère de n'importe quel système d'écriture de langue par un identifiant numérique et un nom unique, et ce de manière unifiée quelle que soit la machine utilisée. Une fois le code défini, il faut en déduire le formatage, c'est à dire son association à un code binaire, c'est l'objectif de UTF-8 (UCS transformation Format 8 bits). Le tableau 2 présente trois exemples.

Caractère	Id. Unicode	Codage binaire UTF-8
A	65	0 1000001
é	233	11 000011 10 101001
€	8364	111 00010 10 000010 10 101100

tableau 2 : Exemples de format UTF-8.

Pour le caractère A, le bit de poids fort à '0' indique que le caractère n'est représenté que par un unique octet (c'est de l'ASCII). Pour le caractère é, l'entête 110 indique que le caractère est constitué d'un train de 2 octets. Pour un train de 3 octets et 4 octets, nous avons l'entête 1110 et 11110, respectivement. Enfin, un entête 10 indique que le l'octet fait partie d'un train d'octets.

3. LES EXERCICES D'APPLICATION

3.7 LES NOMBRES ENTIERS, REPRESENTATION ET CONVERSION DE BASES

Effectuer les conversions suivantes:

a) $1030_{(10)} = \underline{\hspace{2cm}}_2$

b) $1030_{(10)} = \underline{\hspace{2cm}}_{(8)}$ (à déduire du a))

c) $1030_{(10)} = \underline{\hspace{2cm}}_{(16)}$ (à déduire du a))

Expliquer pour chaque conversion la méthode utilisée.

3.8 LES NOMBRES ENTIERS SIGNES, COMPLEMENT A 2

3.8.1 Représentation binaire en CA2

Donner la représentation en Complément à 2 (CA2) des nombres décimaux suivants:

+ 34 + 4,75 - 57 - 23,75

3.8.2 Conversion d'un nombre en CA2 en sa valeur décimale

Calculer la valeur décimale des nombres suivants codés en CA2:

0 1 0 1 1 1 0 1 1 1 1, 1 1 0 0 0, 0 1
0 1 0 0 1 0 0 0 1 0 0 1

3.9 LES NOMBRES REELS

3.9.1 Conversion de bases

Effectuer les conversions suivantes:

a) $ABCD,1F_{(16)} = \underline{\hspace{2cm}}_{(8)}$

b) $ABCD,1F_{(16)} = \underline{\hspace{2cm}}_{(10)}$

c) $31,35 = \underline{\hspace{2cm}}_{(2)}$ avec 8 bits en tout

d) $34,703125_{(10)} = \underline{\hspace{2cm}}_{(2)}$ sans approximation

Expliquer pour chaque conversion la méthode utilisée.

3.9.2 Représentation de nombres en virgule flottante dans les processeurs (norme IEEE 754)

Donner la représentation en virgule flottante simple précision (sur 32 bits) du nombre décimal:

+ 35, 5703125

4. POUR ALLER PLUS LOIN

REPRESENTATION SIGNE+AMPLITUDE (OU MODULE)

Il s'agit d'une représentation parfois utilisée car plus simple que celle du CA2, mais qui est moins bien adaptée aux opérations arithmétiques.

Dans cette représentation, le bit de poids le plus fort représente le signe (MSB = 0 => nombre positif, MSB = 1 => nombre négatif), et les autres bits la valeur absolue du nombre.

Ainsi, un format de n bits permet de coder les nombres compris entre $-(2^{n-1} - 1)$ et $2^{n-1} - 1$.

Dans cette représentation, le zéro possède deux notations possibles.

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	0000	1000	0
1	0001	1001	-1
2	0010	1010	-2
3	0011	1011	-3
4	0100	1100	-4
5	0101	1101	-5
6	0110	1110	-6
7	0111	1111	-7

tableau 3 : représentation "module + signe" sur 4 bits

N. B. Extension d'un nombre en représentation "module + signe"

L'extension d'un nombre codé sur n bits à un format sur $n+k$ bits consiste à décaler le bit de signe à la position du MSB et à compléter les autres positions par des 0, que le nombre soit positif ou négatif. Par exemple

$$\begin{array}{ccc}
 \underbrace{0110}_{\substack{6_{(10)} \text{ codé} \\ \text{sur 4 bits}}} \xrightarrow{6 \text{ bits}} \underbrace{000110}_{\substack{S \\ 6_{(10)} \text{ codé} \\ \text{sur 4 bits}}} & \text{, et} & \underbrace{1110}_{\substack{-6_{(10)} \text{ codé} \\ \text{sur 4 bits}}} \xrightarrow{6 \text{ bits}} \underbrace{100110}_{\substack{S \\ -6_{(10)} \text{ codé} \\ \text{sur 4 bits}}}
 \end{array}$$

REPRESENTATION BINAIRE DECALEE

Cette représentation peut être déduite du complément à 2 par une simple inversion du bit de signe (MSB = 0 => nombre négatif, MSB = 1 => nombre positif). Cette représentation est commode pour la conversion numérique/analogique car la valeur maximale positive est codée par tous les bits à 1 et la valeur minimale négative par tous les bits à 0.

Par exemple, pour $n = 4$,

$N_{(10)}$	$N_{(2)}$	$-N_{(2)}$	$-N_{(10)}$
0	1000	1000	0
1	1001	0111	-1
2	1010	0110	-2
3	1011	0101	-3

4	1100	0100	-4
5	1101	0011	-5
6	1110	0010	-6
7	1111	0001	-7
		0000	-8

tableau 4 : représentation binaire décalée sur 4 bits

CLASSIFICATION DES CODES BINAIRES

Le champ d'application des systèmes numériques est très étendu. Lorsque l'application ne nécessite pas de calculs arithmétiques, les codages précédents sont inutiles ou peu adaptés. On utilise alors des codages possédant d'autres propriétés. On emploie ainsi dans certains systèmes des codes permettant d'éviter des états transitoires parasites lors de la saisie de données, ou de visualiser facilement des chiffres ou des lettres, ou bien encore de détecter des erreurs et/ou de les corriger dans un résultat susceptible d'être erroné. Nous présentons ci-après quelques codes fréquemment utilisés.

L'ensemble des codes binaires peuvent être regroupés en deux classes : les **codes pondérés** et les **codes non pondérés**.

Codes pondérés

Un code est dit pondéré si la position de chaque symbole dans chaque mot correspond à un poids fixé : par exemple 1, 10, 100, 1000 ... pour la numération décimale, et 1, 2, 4, 8, ... pour la numération binaire. Les codes pondérés ont, en général, des propriétés intéressantes du point de vue arithmétique.

Le code binaire pur et ses dérivés (octal, hexadécimal)

Ce sont les codes utilisés en arithmétique binaire et qui ont été étudiés dans la première partie de ce chapitre.

Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal)

Ce code est utilisé dans de nombreux systèmes d'affichage, de comptage ou même les calculatrices de poche. Dans le code BCD chaque chiffre d'un nombre décimal (de $0_{(10)}$ à $9_{(10)}$) est codé à l'aide de 4 bits (de $0000_{(2)}$ à $1001_{(2)}$). Ainsi le code BCD n'utilise que 10 **mots de codes** de 4 bits. Par exemple la représentation du nombre $1995_{(10)}$ est : $(0001\ 1001\ 1001\ 0101)_{(BCD)}$. Il est possible d'effectuer des opérations arithmétiques en BCD, mais celles-ci sont plus complexes qu'en binaire classique. Ce code est pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100, ...

Codes non pondérés

Dans le cas des codes non pondérés, il n'y a pas de poids affecté à chaque position des symboles. On convient simplement d'un tableau de correspondance entre les objets à coder et une représentation binaire. De tels codes peuvent néanmoins parfois posséder des propriétés arithmétiques intéressantes, comme le code **excédent 3**.

Code excédent 3 ou excess 3

Le code excédent 3 utilise, tout comme le code BCD, 10 mots de codes, auxquels on fait correspondre les 10 chiffres décimaux.

$N_{(10)}$	$N_{(XS3)}$
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

tableau 5 : code excédent 3

Il est obtenu en décalant le code binaire de trois lignes vers le haut. Ce code peut être intéressant pour effectuer des soustractions car le complément à 1 de la représentation binaire d'un chiffre correspond au complément à 9 de ce chiffre. Ainsi, toute opération de soustraction se ramène à une addition.

Par exemple $5_{(XS3)} = 1000$, son complément à 1 est obtenu par inversion des bits, $\bar{5}_{(XS3)} = 0111 = 4_{(XS3)}$, le résultat en excédent 3 est le nombre 4, qui est le complément à 9 de 5 en décimal. Ainsi, pour faire une soustraction, il suffit d'ajouter le complément à 1 du nombre à retrancher, puis 1. Par exemple, $(7-5)_{(XS3)} = 7_{(XS3)} + \bar{5}_{(XS3)} + 1_{(XS3)} = 1010 + 0111 + 0100 = 0101 = 2_{(XS3)}$.

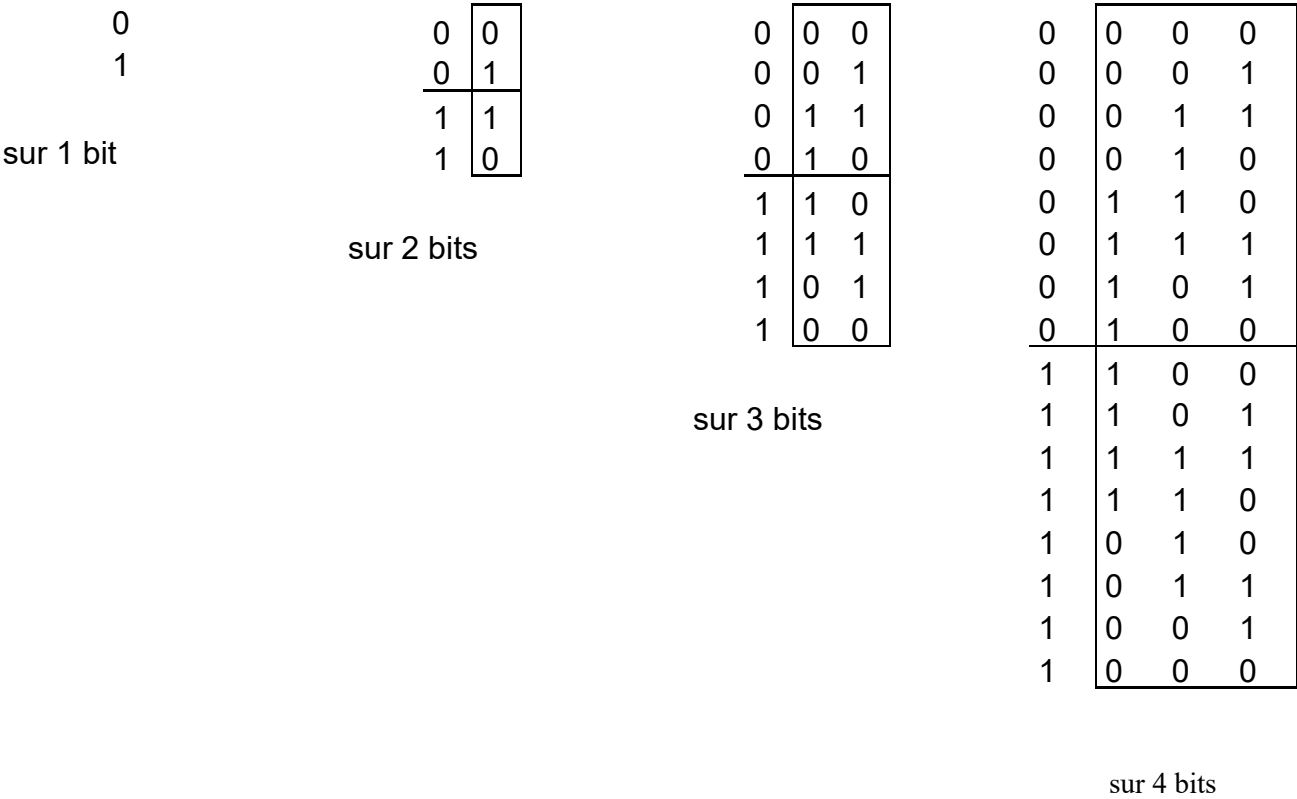
Le code excédent 3 ne présente pas d'intérêt en addition.

Code binaire réfléchi ou code de Gray

Ce code numérique n'étant pas pondéré, il est peu employé pour les opérations arithmétiques. Il est, par contre, utilisé pour le codage des déplacements angulaires, linéaires ou pour la réalisation des tableaux de Karnaugh (cf. chapitre « Propriétés des variables et fonctions logiques »). La propriété principale de ce code est que **deux mots successifs du code ne diffèrent que par un élément binaire**. Ceci permet, d'une part d'éviter la génération d'aléas (états parasites) au passage de deux combinaisons successives, et d'autre part de tirer parti de cette adjacence du codage pour simplifier les fonctions.

L'appellation "binaire réfléchi" provient de sa technique de construction : on peut construire un code de Gray sur n bits à partir d'un code de Gray sur $n-1$ bits en procédant comme suit : on copie les mots du code de départ, précédés d'un 0, suivis des mots du même code, pris dans **l'ordre inverse** et précédés d'un 1. Ceci permet de construire un code de Gray de n'importe quel format (cf. figure 1.7).

figure 1.7 : Construction du code de Gray sur 1, 2, 3, et 4 bits



Codes redondants

Il existe un ensemble de codes conçus pour pouvoir détecter, voire corriger des erreurs dans des messages binaires. Leur principe repose sur l'insertion de données redondantes dans l'information initiale. Leur étude approfondie relève du domaine des communications numériques, et ne sera pas traité dans ce cours. Nous citerons cependant quelques exemples simples de codes redondants, les codes p parmi n et les codes de contrôle de parité.

(a) Code p parmi n

Ce code est constitué de $C_n^p = \frac{n!}{p!(n-p)!}$ mots de code. Chaque mot de code est codé sur n bits et contient exactement p "1" et $(n - p)$ "0". Par exemple, le code 2 parmi 5 (tableau 6) est constitué de 10 mots de codes et permet de coder les chiffres décimaux.

$N_{(10)}$	$N_{(2 \text{ parmi } 5)}$
0	0 0 0 1 1
1	1 1 0 0 0
2	1 0 1 0 0
3	0 1 1 0 0
4	1 0 0 1 0
5	0 1 0 1 0
6	0 0 1 1 0
7	1 0 0 0 1
8	0 1 0 0 1
9	0 0 1 0 1

tableau 6 : code 2 parmi 5

L'utilisation de ce code permet, à la réception d'une information, de vérifier par comptage du nombre de 1 si une erreur s'est introduite dans l'information transmise. Dans le cas où plus d'une erreur s'est glissée dans un mot de code, la détection n'est pas assurée dans tous les cas. Ce code ne permet pas non plus de trouver la place de l'erreur, donc de la corriger. D'autre part, le décodage des combinaisons est particulièrement simple, car il ne porte que sur 2 bits par combinaison.

(b) Code contrôle de parité

Le codage d'un mot de n bits par contrôle de parité consiste à y adjoindre un $(n+1)^{\text{ème}}$ bit dont le rôle est de rendre systématiquement pair ou impair le nombre total de 1 contenus dans l'information codée. Si une erreur se glisse dans l'information, le nombre de 1 devient impair et l'erreur est détectée. Ce code ne permet pas non plus de corriger les erreurs.

 IMT Atlantique Bretagne - Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD2 – ALGEBRE DE BOOLE

PCD2 – ALGEBRE DE BOOLE.....	1
1. LES OBJECTIFS D'APPRENTISSAGE	2
2. LES CONCEPTS	3
2.1. INTRODUCTION.....	3
2.2. PROPRIETES DE L'ALGEBRE DE BOOLE.....	3
2.2.1. Définitions.....	3
2.2.2. Table de vérité d'une fonction logique	3
2.2.3. Les fonctions logiques élémentaires	3
2.3. REPRESENTATION DES FONCTIONS LOGIQUES	9
2.3.1. Formes algébriques disjonctives, conjonctives, canoniques	9
2.3.2. Représentations de référence d'une fonction logique	10
2.3.3. Critères de choix d'une représentation	10
3. EXERCICES D'APPLICATION	12
3.1. EXERCICE 1 :TABLE DE VERITE ET FORMES CANONIQUES	12
3.2. EXERCICE 2 : THEOREME DE <i>DE MORGAN</i>	12
3.3. EXERCICE 3 : FORME CANONIQUE (1)	12
3.4. EXERCICE 4 : FORME CANONIQUE (2).....	12
4. POUR ALLER PLUS LOIN.....	13
4.1.1. Opérateurs complets.....	13

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD2 – Algèbre de Boole

1. LES OBJECTIFS D'APPRENTISSAGE

OA2 Représenter une fonction logique décrite par une proposition logique sous la forme d'une table de vérité, ou sous ses formes canoniques.

OA3 Appliquer des propriétés et des théorèmes tirés de l'algèbre de Boole.

OA4 Choisir la forme conjonctive ou disjonctive d'une fonction logique pour minimiser les ressources matérielles nécessaires à l'implantation matérielle de la fonction.

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

2.1. INTRODUCTION

Le fonctionnement des systèmes numériques repose sur la manipulation de variables et fonctions dont les valeurs sont représentées par des grandeurs physiques dites **binaires** car ne pouvant prendre que deux valeurs (généralement notées **0** et **1**). La structure mathématique permettant de formaliser les opérations de manipulation de ces grandeurs binaires est dite **algèbre de commutation** ou plus communément **algèbre de Boole**. Nous nous intéressons dans ce chapitre aux bases et aux propriétés fondamentales de l'algèbre de Boole indispensables à la compréhension du fonctionnement des systèmes numériques.

2.2. PROPRIETES DE L'ALGEBRE DE BOOLE

2.2.1. Définitions

Dans l'algèbre de commutation, une variable ne peut prendre que 0 ou 1 comme valeur possible. Une telle variable est dite **variable logique**, **variable binaire**, ou **variable booléenne**. De même, une fonction de n variables logiques ne peut prendre comme valeur que 0 ou 1. Elle est dite **fonction logique**, **fonction binaire**, ou **fonction booléenne**.

2.2.2. Table de vérité d'une fonction logique

C'est une table donnant l'état logique de la fonction pour chacune des combinaisons des états de ses variables. Une fonction de n variables est représentée par une table de vérité à $n+1$ colonnes et au plus 2^n lignes. Le tableau 2.1 donne la forme générale d'une fonction de deux variables logiques.

A	B	F(A,B)
0	0	F(0,0)
0	1	F(0,1)
1	0	F(1,0)
1	1	F(1,1)

tableau 2.1 : forme générale de la table de vérité d'une fonction de deux variables logiques

2.2.3. Les fonctions logiques élémentaires

Trois fonctions suffisent pour définir une algèbre de Boole : la **complémentation**, le **produit logique**, et l'**addition logique**.

2.2.3.1. La fonction de complémentation ou fonction NON

Le complément de la variable A se note $\bar{A}$ (lire « A barre » ou « non A »). $\bar{A}$ vaut 1 (respectivement 0) si et seulement si A vaut 0 (respectivement 1). On parle encore de fonction d'**inversion logique**. Le tableau 2.2 donne la table de vérité de la fonction de complémentation. Les symboles usuellement utilisés pour représenter graphiquement l'opérateur correspondant, appelé **inverseur**, sont ceux de la figure 2.1.



0	1
1	0

tableau 2.2 : table de vérité de la fonction NON

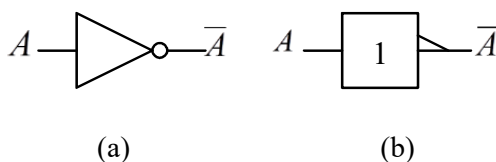


figure 2.1 : symboles logiques d'un inverseur

(a) notation usuelle (ancienne notation US)

(b) notation normalisée IEEE (ancienne notation européenne)

2.2.3.2. La fonction produit logique ou fonction ET

Le produit logique de 2 variables se note $A.B$, AB , ou bien encore $A \wedge B$ (lire « A et B »). $A.B$ vaut 1 si et seulement si A et B valent 1. Le tableau 2.3 donne la table de vérité de la fonction ET, et la figure 2.2 les symboles logiques de l'opérateur associé.

A	B	$A.B$
0	0	0
0	1	0
1	0	0
1	1	1

tableau 2.3 : table de vérité de la fonction ET

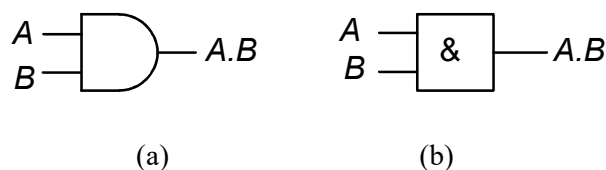


figure 2.2 : symboles logiques de l'opérateur ET.

(a) notation usuelle

(b) notation normalisée IEEE

2.2.3.3. La fonction addition logique ou fonction OU

L'addition logique de 2 variables se note $A+B$ ou $A \vee B$ (lire « A ou B »). $A+B$ vaut 0 si et seulement si A et B valent 0. Le tableau 2.4 donne la table de vérité de la fonction OU, et la figure 2.3 les symboles logiques de l'opérateur associé.

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

tableau 2.4 : table de vérité de la fonction OU

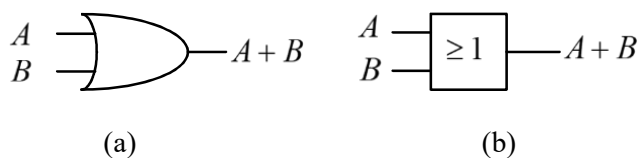


figure 2.3 : symboles logiques de l'opérateur OU.

(a) notation usuelle

(b) notation normalisée IEEE

2.2.3.4. Propriétés des fonctions NON, ET, et OU

- Commutativité des fonctions ET et OU :

$$AB = BA$$
$$A + B = B + A$$

- Associativité des fonctions ET et OU :

$$A(BC) = (AB)C = ABC$$
$$A + (B + C) = (A + B) + C = A + B + C$$

- Éléments neutres pour les fonctions ET et OU

$$A.1 = 1.A = A$$
$$A + 0 = 0 + A = A$$

- Éléments absorbants pour les fonctions ET et OU

$$A.0 = 0.A = 0$$
$$A + 1 = 1 + A = 1$$

- Propriété d'idempotence des fonctions ET et OU

$$A.A = A$$
$$A + A = A$$

- Propriétés de l'inversion logique

$$\overline{\overline{A}} = A$$
$$\overline{\overline{A}}.A = 0$$
$$\overline{\overline{A}} + A = 1$$

- Distributivité de ET par rapport à OU

$$A(B + C) = AB + AC$$
$$(A + B)C = AC + BC$$

- Distributivité de OU par rapport à ET

$$A + BC = (A + B)(A + C)$$
$$AB + C = (A + C)(B + C)$$

- Autres relations utiles se déduisant des précédentes (relations de simplification)

$$A + A.B = A.1 + A.B = A.(1 + B) = A$$
$$A.(A + B) = (A + 0).(A + B) = A + (0.B) = A$$
$$A + \overline{A}.B = (A + \overline{A}).(A + B) = 1.(A + B) = A + B$$
$$A.(\overline{A} + B) = A.\overline{A} + A.B = A.B$$

N.B : Si certaines égalités ne vous paraissent pas triviales, c'est qu'elles font vraisemblablement appel à la propriété de distributivité du OU par rapport au ET.

- *Théorème de De Morgan*

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

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

Ce théorème se généralise à un nombre quelconque de variables :

$$\overline{\sum_i X_i} = \prod_i \overline{X_i}$$

$$\overline{\prod_i X_i} = \sum_i \overline{X_i}$$

N. B. On notera que l'analogie entre l'addition logique (resp. produit logique) et l'addition (resp. multiplication) de l'arithmétique classique se limite à un nombre très restreint de propriétés.

2.2.3.5. Opérateurs secondaires

Dans les circuits logiques, on utilise également des opérateurs qui sont des combinaisons des fonctions ET, OU, et NON.

1. La fonction NON ET ou NAND : $\overline{A . B}$

La table de vérité de la fonction NON ET se déduit immédiatement de celle de la fonction ET par inversion du résultat (tableau 2.5).

A	B	$\overline{A . B}$
0	0	1
0	1	1
1	0	1
1	1	0

tableau 2.5 : table de vérité de la fonction NON ET

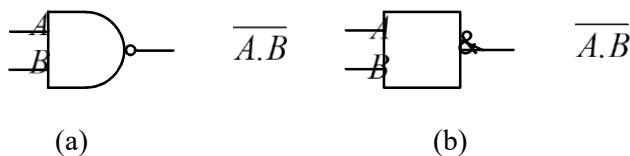


figure 2.4 : symboles logiques de l'opérateur NON ET.

(a) notation usuelle

(b) notation normalisée IEEE

1. La fonction NON OU ou NOR : $\overline{A+B}$

La table de vérité de la fonction NON OU se déduit immédiatement de celle de la fonction OU par inversion du résultat (tableau 2.6).

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

tableau 2.6 : table de vérité de la fonction NON OU

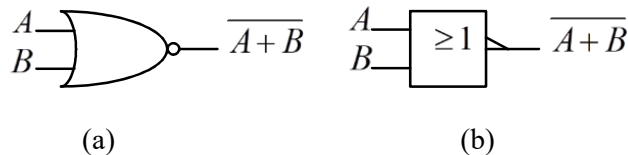


figure 2.5 : symboles logiques de l'opérateur NON OU, (a) notation usuelle, (b) notation normalisée IEEE

2. Quelques propriétés des fonctions NON ET et NON OU

Les propriétés des fonctions NON ET et NON OU se déduisent des propriétés des fonctions élémentaires NON, ET, et OU.

$$\begin{aligned}
 \overline{AB} &= \overline{BA} & \overline{A+B} &= \overline{B+A} \\
 \overline{(AB)C} &= \overline{A(BC)} = \overline{ABC} & \text{mais } \overline{\overline{ABC}} &\neq \overline{\overline{ABC}} \\
 \overline{(A+B)+C} &= \overline{A+(B+C)} = \overline{A+B+C} & \text{mais } \overline{\overline{A+B+C}} &\neq \overline{\overline{A+B+C}} \\
 \overline{A.1} &= \overline{A} & \overline{A+1} &= 0 \\
 \overline{A.0} &= 1 & \overline{A+0} &= \overline{A} \\
 \overline{A.A} &= \overline{A} & \overline{A+A} &= \overline{A} \\
 \overline{\overline{A}.A} &= 1 & \overline{\overline{A}+A} &= 0
 \end{aligned}$$

3. La fonction OU exclusif (abrégé OUEX ou XOR) : $A \oplus B = A.\overline{B} + \overline{A}.B$

4.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

tableau 2.7 : table de vérité de la fonction OU exclusif

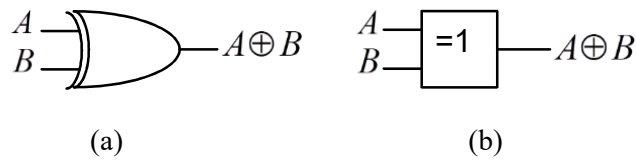


figure 2.6 : symboles logiques de l'opérateur OU exclusif.

(a) notation usuelle

(b) notation normalisée IEEE

- **Propriétés de la fonction OU exclusif**

$$A \oplus B = B \oplus A \quad (\text{commutativité})$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C = A \oplus B \oplus C \quad (\text{associativité})$$

$$A \oplus 0 = A \quad A \oplus 1 = \bar{A}$$

$$A \oplus A = 0 \quad A \oplus \bar{A} = 1$$

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

- **Utilisations courantes de la fonction OU exclusif**

⇒ Détection de deux éléments binaires différents,

$$A \oplus B = 1 \Leftrightarrow A \neq B$$

⇒ Détection d'un nombre de variables impair,

$$A \oplus B \oplus C \oplus \dots = 1 \Leftrightarrow (A, B, C, \dots) \text{ contient un nombre impair de } 1$$

⇒ Somme modulo 2 de deux éléments binaires.

5. La fonction ET inclusif (abrégié XNOR) : $A \oplus B = \overline{A \oplus B} = A \cdot B + \bar{A} \cdot \bar{B}$

A	B	A ⊕ B
0	0	1
0	1	0
1	0	0
1	1	1

tableau 2.8 : table de vérité de la fonction ET inclusif

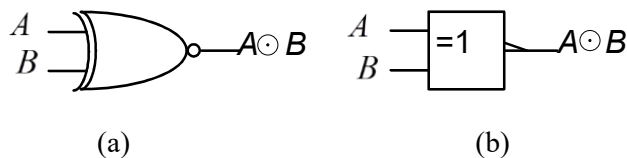


figure 2.7 : symboles logiques de l'opérateur ET inclusif.

(a) notation usuelle

(b) notation normalisée IEEE

- **Propriétés de la fonction ET inclusif**

Les propriétés du ET inclusif se déduisent aisément des propriétés de la fonction OU exclusif en remarquant que

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

- *Utilisations courantes de l'opérateur ET inclusif*

⇒ Détection de deux éléments binaires égaux,

$$\overline{A \oplus B} = 1 \Leftrightarrow A = B$$

⇒ Détection d'un nombre de variables pair,

$$\overline{A \oplus B \oplus C \oplus \dots} = 1 \Leftrightarrow (A, B, C, \dots) \text{ contient un nombre pair de 1}$$

2.3. REPRESENTATION DES FONCTIONS LOGIQUES

2.3.1. Formes algébriques disjonctives, conjonctives, canoniques

Considérons la table de vérité de la fonction booléenne F de 3 variables A, B, et C, définie par le tableau 2.9.

A	B	C	numéro de la combinaison	F(A,B,C)	$\overline{F(A,B,C)}$
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	2	0	1
0	1	1	3	0	1
1	0	0	4	1	0
1	0	1	5	1	0
1	1	0	6	1	0
1	1	1	7	0	1

tableau 2.9 : table de vérité d'une fonction booléenne F de 3 variables

On peut extraire une expression de F en exprimant les combinaisons des variables A, B, et C pour lesquelles F est égale à 1 : F vaut 1 pour les combinaisons 0, 1, 4, 5, et 6, c'est-à-dire si $\overline{A}\overline{B}\overline{C}=1$, $\overline{A}\overline{B}C=1$, $A\overline{B}\overline{C}=1$, $A\overline{B}C=1$, ou $AB\overline{C}=1$. La fonction F peut donc s'écrire sous la forme :

$$F(A,B,C) = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C}$$

Dans cet exemple, si $A=B=C=0$, alors $F=1$ car $\overline{A}\overline{B}\overline{C}=1$. Toutes les autres combinaisons de variables d'entrée sont égales à 0.

L'expression obtenue est une somme logique de produits logiques, il s'agit d'une **forme algébrique disjonctive**. Les produits logiques font intervenir toutes les variables, sous leur forme directe ou complémentée. Ces produits élémentaires sont appelés **mintermes**. Pour n variables logiques, il existe 2^n mintermes différents, chaque minterme étant égal à 1 pour une seule combinaison des n variables. La représentation d'une fonction sous la forme d'une somme de mintermes est dite **forme canonique disjonctive** ou **première forme canonique**.

On peut extraire une seconde expression de F en exprimant les combinaisons des variables A , B , et C pour lesquelles F est égale à 0. F vaut 0 pour les combinaisons 2, 3, et 7, ce qui peut encore s'écrire :

$$F(A, B, C) = (A + \bar{B} + C). (A + \bar{B} + \bar{C}). (\bar{A} + \bar{B} + \bar{C})$$

En effet, si $B = 1$ et $A = C = 0$ (ce qui correspond à la combinaison 2), alors $A + \bar{B} + C = 0$ et donc $F = 0$. Idem pour les combinaisons 3 et 7.

On peut également commencer par exprimer $\bar{F} = \bar{A}.B.\bar{C} + \bar{A}.B.C + A.B.C$. En effet, $F = 0$ si $B = 1$ et $A = C = 0$. Puis appliquer le théorème de De Morgan sur cette expression pour obtenir F .

Comme pour la forme disjonctive, toutes les autres combinaisons sont, dans ce cas, égales à un. Cette nouvelle expression a une forme duale de la précédente. C'est un produit logique de sommes logiques, il s'agit d'une **forme algébrique conjonctive**. Les sommes logiques composant le produit font intervenir toutes les variables, sous leur forme directe ou complémentée. Elles sont appelées **maxtermes**. Pour n variables logiques, il existe 2^n maxtermes différents, chaque maxterme étant égal à 0 pour une seule combinaison des n variables. La représentation d'une fonction sous la forme d'un produit de maxtermes est dite **forme canonique conjonctive** ou **seconde forme canonique**.

2.3.2. Représentations de référence d'une fonction logique

Parmi les différentes représentations des fonctions logiques étudiées dans ce chapitre, trois d'entre elles peuvent être considérées comme des représentations de référence en raison de leur unicité :

- la table de vérité,
- les deux formes canoniques.

En effet, deux fonctions logiques sont égales si et seulement si leurs tables de vérité ou leurs formes canoniques sont identiques.

2.3.3. Critères de choix d'une représentation

L'un des deux types de représentation, forme disjonctive ou conjonctive, peut être préférable à l'autre si des contraintes sont imposées sur la réalisation matérielle des fonctions. En particulier, dans le cas de l'utilisation de **circuits logiques** réalisant les fonctions logiques élémentaires, le type de circuits disponibles peut favoriser une des deux formes.

Ainsi, la forme disjonctive est bien adaptée à une réalisation à base d'opérateurs NON ET. En effet, soit F une fonction de 4 variables écrite sous la forme disjonctive suivante (non canonique dans le cas traité, puisque les produits ne sont pas des mintermes) :

$$F(A, B, C, D) = A.B + \bar{C}D$$

Pour réaliser cette fonction à l'aide d'opérateurs NON ET et d'inverseurs, il faut dans un premier temps transformer la fonction pour l'écrire sous la forme d'une combinaison de fonctions élémentaires NON ET et d'inversion (application du théorème de De Morgan):

$$F(A, B, C, D) = A.B + \bar{C}D = \overline{\overline{A.B + \bar{C}D}} = \overline{\overline{A.B} . \overline{\bar{C}D}} = \overline{\overline{A.B} . C.D}$$

Il reste ensuite à assembler le nombre adéquat d'opérateurs élémentaires pour réaliser F . La figure 2.8 montre un schéma de réalisation (ou **logigramme**) de F utilisant 3

opérateurs NON ET et 1 inverseur. Si l'opérateur d'inversion n'est pas disponible, il peut lui-même être réalisé à l'aide d'un opérateur NON ET, cf. §4.1.1.

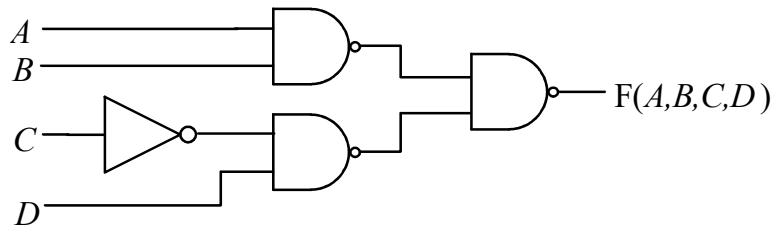


figure 2.8: logigramme de la fonction F à base d'opérateurs NON ET et d'un inverseur

De même, la forme conjonctive est bien adaptée à une réalisation à base d'opérateurs NON OU. En effet, soit une fonction G de 4 variables écrite sous une forme conjonctive :

$$G(A,B,C,D) = (\overline{A} + \overline{B}).(C + D) = \overline{\overline{\overline{\overline{\overline{A} + \overline{B}}}}}.(C + D) = \overline{\overline{\overline{A} + \overline{B} + C + D}}$$

La fonction G peut être réalisée à l'aide de 3 opérateurs NON OU et 2 inverseurs (figure 2.9).

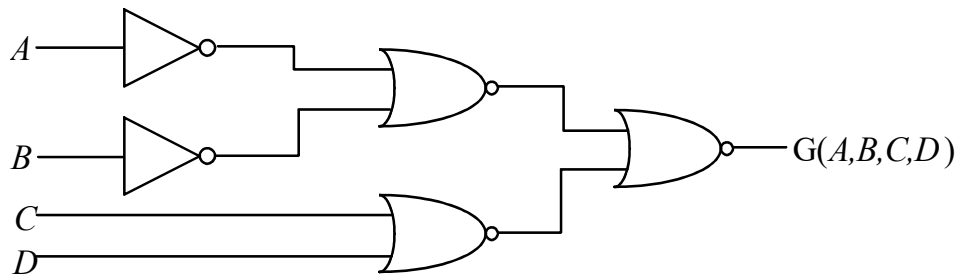


figure 2.9 : logigramme de la fonction G à base d'opérateurs NON OU et d'inverseurs

Lorsqu'aucune contrainte extérieure n'impose l'une des représentations, la forme disjonctive est traditionnellement plus utilisée que la forme conjonctive, en raison de l'analogie de notation entre les opérations logiques et arithmétiques.

Lors de la mise en œuvre d'une fonction logique dans un circuit, deux types de contraintes peuvent être prises en compte : optimiser la **vitesse** du circuit (c.-à-d. obtenir une fréquence maximale de fonctionnement la plus grande possible) ou bien optimiser sa **complexité** (c.-à-d. obtenir un encombrement sur silicium minimal). Dans le cas où la contrainte de **complexité** est la plus forte, il faut utiliser le minimum de matériel. Il est, pour cela, nécessaire de représenter la fonction à réaliser sous une forme simplifiée, c'est-à-dire utilisant un nombre minimal d'opérateurs. Le problème de la simplification des fonctions logiques est traité à la séance suivante.

3. EXERCICES D'APPLICATION

3.1. EXERCICE 1 : TABLE DE VERITE ET FORMES CANONIQUES

Établir les tables de vérité des fonctions suivantes, puis les écrire sous les deux formes canoniques :

$$F_1 = XY + YZ + XZ$$

$$F_3 = (X + Y)(\bar{X} + Y + Z)$$

À faire à la maison pour s'entraîner

$$F_2 = X + Y.Z + \bar{Y}.\bar{Z}.T$$

$$F_4 = (\bar{X} + \bar{Y} + Z)(X + \bar{Y} + Z)(X + \bar{Y} + \bar{Z})(X + Y + \bar{Z})(X + Y + Z)$$

3.2. EXERCICE 2 : THEOREME DE DE MORGAN

Complémenter les fonctions suivantes (sans simplifier) :

$$F_1 = \bar{X}\bar{Y} + XY + \bar{X}Y$$

$$F_2 = X.(\bar{Y}.\bar{Z} + Y.Z) + \bar{X}.Y.\bar{Z} + \bar{X}.\bar{Y}.Z$$

À faire à la maison pour s'entraîner

$$F_3 = X\bar{Y} + Z\bar{T} + \bar{X}\bar{Y} + \bar{Z}\bar{T}$$

3.3. EXERCICE 3 : FORME CANONIQUE (1)

Écrire sous la première forme canonique les fonctions définies par les propositions suivantes :

$$f_1(a, b, c) = 1 \quad \text{si et seulement si aucune des variables } A, B, C \text{ ne prend la valeur 1}$$

$$f_2(a, b, c) = 1 \quad \text{si et seulement si au plus une des variables } A, B, C \text{ prend la valeur 0}$$

Mettre les fonctions sous la seconde forme canonique.

À faire à la maison pour s'entraîner

$$f_3(a, b, c) = 1 \quad \text{si et seulement si exactement une des variables } A, B, C \text{ prend la valeur 1}$$

Mettre les fonctions sous la seconde forme canonique.

3.4. EXERCICE 4 : FORME CANONIQUE (2)

Écrire sous la seconde forme canonique la fonction définie par la proposition suivante:

$$g(a, b, c) = 0 \quad \text{si et seulement si aucune des variables } A, B, C \text{ ne prend la valeur 1.}$$

Mettre cette fonction sous la première forme canonique.

4. POUR ALLER PLUS LOIN

4.1.1. Opérateurs complets

Un opérateur logique est dit complet s'il permet de réaliser les trois fonctions de base de l'algèbre de Boole et, par conséquent, toutes les fonctions logiques. **Par exemple, l'opérateur NON ET est complet. Il en est de même pour l'opérateur NON OU.**

En effet,

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

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

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A} \cdot \overline{B}}$$

De même,

$$\overline{A} = \overline{A + A}$$

$$A \cdot B = \overline{\overline{A \cdot B}} = \overline{\overline{A + B}} = \overline{\overline{A + A + B + B}}$$

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

En revanche, les opérateurs OU exclusif et ET inclusif, ne sont pas complets.

 IMT Atlantique Bretagne - Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance
PCD3 – SIMPLIFICATION DES FONCTIONS LOGIQUES	

PCD3 – SIMPLIFICATION DES FONCTIONS LOGIQUES.....	1
1. LES OBJECTIFS D'APPRENTISSAGE	2
2. LES CONCEPTS	3
2.1. POURQUOI SIMPLIFIER LES FONCTIONS LOGIQUES ?	3
2.2. SIMPLIFICATION PAR TABLEAU DE KARNAUGH.....	3
2.2.1. Introduction	3
2.2.2. Adjacence logique.....	3
2.2.3. Construction d'un diagramme de Karnaugh.....	4
2.2.4. Principe de la simplification.....	7
2.3. CONCLUSION	14
2.4. REFERENCES	15
3. EXERCICES D'APPLICATION	16
3.1. SIMPLIFICATION DES FONCTIONS A 3 ENTREES.....	16
3.2. SIMPLIFICATION DES FONCTIONS A 4 ENTREES.....	16
3.3. SIMPLIFICATION ALGEBRIQUE A PARTIR DES PROPRIETES DES FONCTIONS LOGIQUES.....	16
4. POUR ALLER PLUS LOIN.....	17
4.1. SIMPLIFICATION ALGEBRIQUE	17
4.1.1. Exemple 1 : simplification de $F_1 = BC + AC + AB + B$	17
4.1.2. Exemple 2 : simplification de $F_2 = (A + \overline{B})(\overline{A}\overline{B} + C)C$	17
4.1.3. Exemple 3 : simplification de $F_3 = \overline{A}B\overline{C} + AB\overline{C} + ABC + \overline{A}BC$	17
4.1.4. Exemple 4 : simplification de $F_4 = \overline{A}B + AC + BC$	17
4.1.5. Exemple 5 : simplification de $F_5 = (\overline{A} + B)(A + C)(B + C)$	17
4.1.6. Exemple 6 : simplification de $F_6 = (\overline{A}\overline{B} + \overline{A}B).(AB + \overline{A}\overline{B})$	18
4.1.7. Conclusion	18

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD3 – Simplification des fonctions logiques

1. LES OBJECTIFS D'APPRENTISSAGE

OA5 Simplifier une fonction logique à 3, 4 ou 5 entrées à l'aide d'un tableau de Karnaugh.

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

2.1. POURQUOI SIMPLIFIER LES FONCTIONS LOGIQUES ?

Simplifier une fonction logique consiste à rechercher une expression de cette fonction conduisant à la réalisation d'un circuit de coût minimal. Il faut cependant noter que la minimisation à tout prix du nombre d'opérateurs n'est pas toujours le but recherché. Dans le cas de fonctions complexes, des contraintes de vitesse ou de testabilité peuvent aller à l'encontre d'une minimalisation des expressions. On peut, par exemple, être amené à augmenter la complexité des opérateurs d'un circuit pour accroître sa vitesse de fonctionnement, ou à limiter la simplification d'une fonction pour extraire du circuit des variables logiques internes.

2.2. SIMPLIFICATION PAR TABLEAU DE KARNAUGH

2.2.1. Introduction

Le diagramme ou tableau de Karnaugh est un **outil graphique** qui permet de **simplifier de façon méthodique** une fonction logique. Bien que les diagrammes de Karnaugh soient applicables en théorie à des fonctions ayant un nombre quelconque de variables, ils ne sont en pratique utilisables « à la main » que pour un nombre de variables inférieur ou égal à 6.

Dans la section 2.2.2, nous partons de la définition de l'adjacence logique pour expliquer pourquoi l'adjacence géométrique de deux termes dans le tableau de Karnaugh conduit à la simplification d'une variable d'entrée.

Dans la section 2.2.3, nous détaillons la construction du diagramme (ou tableau) de Karnaugh selon le nombre de variables d'entrée de la fonction.

Dans la section 2.2.4, nous décrivons le principe de la simplification géométrique à partir de regroupements et de la lecture des variables communes à chaque regroupement.

2.2.2. Adjacence logique

Deux termes sont dits **logiquement adjacents** s'ils ne diffèrent que par une variable. Par exemple, ABC et $\overline{A}BC$ sont deux termes produits adjacents, et $\overline{A} + \overline{B} + \overline{C} + D$ et $\overline{A} + B + \overline{C} + D$ sont deux termes sommes adjacents.

La somme de deux produits adjacents et le produit de deux sommes adjacentes peuvent être simplifiés par mise en facteur, en raison des propriétés de distributivité réciproque des opérateurs ET et OU. En effet,

$$AB + A\overline{B} = A(B + \overline{B}) = A \text{ (distributivité de ET par rapport à OU),}$$

et

$$(A + B)(A + \overline{B}) = A + B\overline{B} = A \text{ (distributivité de OU par rapport à ET).}$$

Ainsi, à partir de la somme (resp. le produit) de 2 combinaisons de 2 variables, nous obtenons une seule variable. Une variable a donc été simplifiée, ce qui conduit à une fonction logique moins complexe. La diminution de la complexité est recherchée si l'on souhaite minimiser le coût d'un circuit.

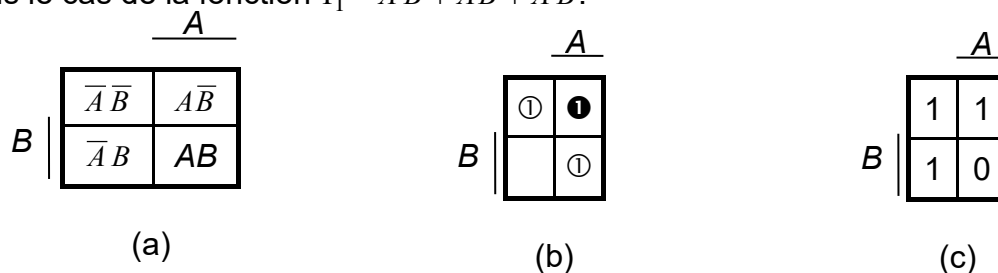
Un tableau de Karnaugh est une **table de vérité disposée de telle sorte que tous les termes logiquement adjacents soient également géométriquement adjacents**, afin de mettre visuellement en évidence les simplifications possibles.

2.2.3. Construction d'un diagramme de Karnaugh

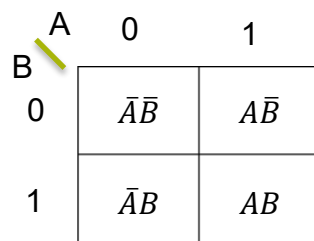
Dans un diagramme de Karnaugh, la correspondance entre adjacence logique et adjacence géométrique est due au codage des combinaisons de variables : deux combinaisons voisines ne varient que par un seul bit (codage de Gray). Chaque case du tableau représente un minterme, et pour une fonction de n variables, chaque case est adjacente à n autres cases, représentant les n mintermes adjacents.

Lors du remplissage du diagramme, la valeur logique 1 est inscrite dans les cases correspondant aux mintermes présents dans l'expression de la fonction, puis le tableau est complété par des 0. Les 0 peuvent être omis pour alléger l'écriture.

2.2.3.1. *Fonction de 2 variables*



(c) remplissage pour $F_1 = \overline{A} \overline{B} + A \overline{B} + \overline{A} B$



2.2.3.2. Fonction de 3 variables

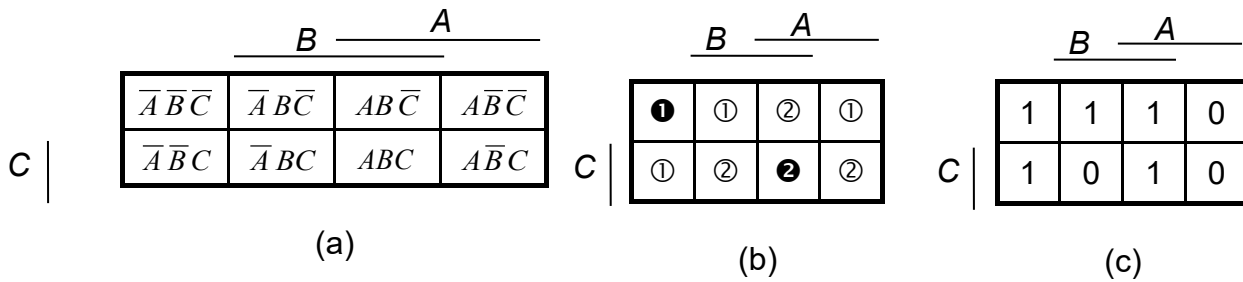


figure 2.3 : diagramme de Karnaugh à 3 variables

(a) position des mintermes

(b) termes adjacents à $\overline{A}\overline{B}\overline{C}$ (symbole ❶) et à ABC (symbole ❽)

(c) remplissage pour $F_2 = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$

Pour le terme ABC , l'adjacence géométrique est évidente. En revanche, pour retrouver l'adjacence géométrique dans le cas de $\overline{A}\overline{B}\overline{C}$, il faut remarquer que les cases aux deux extrémités de la première ligne sont adjacentes. Ceci est mis en évidence en représentant le tableau sous forme cylindrique comme le montre la figure 2.4.

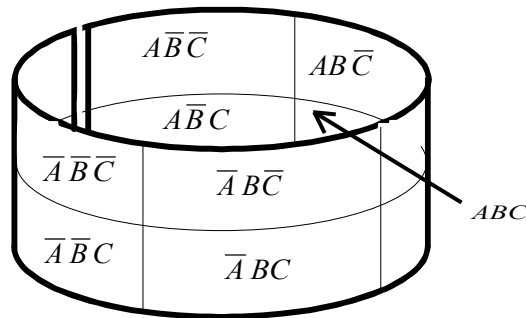


figure 2.4 : représentation cylindrique d'un tableau de Karnaugh à trois variables

Plus généralement, deux cases situées aux extrémités d'une même ligne ou d'une même colonne sont adjacentes. Ceci est dû au code de Gray qui est un code cyclique.

2.2.3.3. Fonction de 4 variables

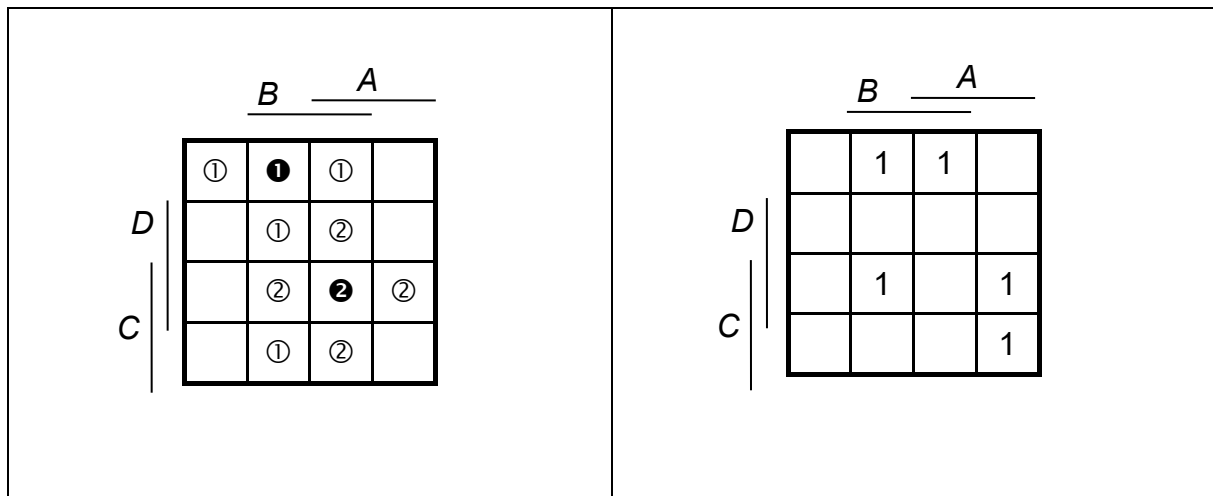


figure 2.5 : diagramme de Karnaugh à 4 variables

- (a) termes adjacents à $\bar{A}B\bar{C}\bar{D}$ (symbole ❶) et à $ABCD$ (symbole ❷)
 (b) remplissage pour $F_3 = \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + A\bar{B}CD$

Les adjacences sur les bords d'un tableau à quatre variables sont mises en évidence par les deux représentations de la figure 2.6.

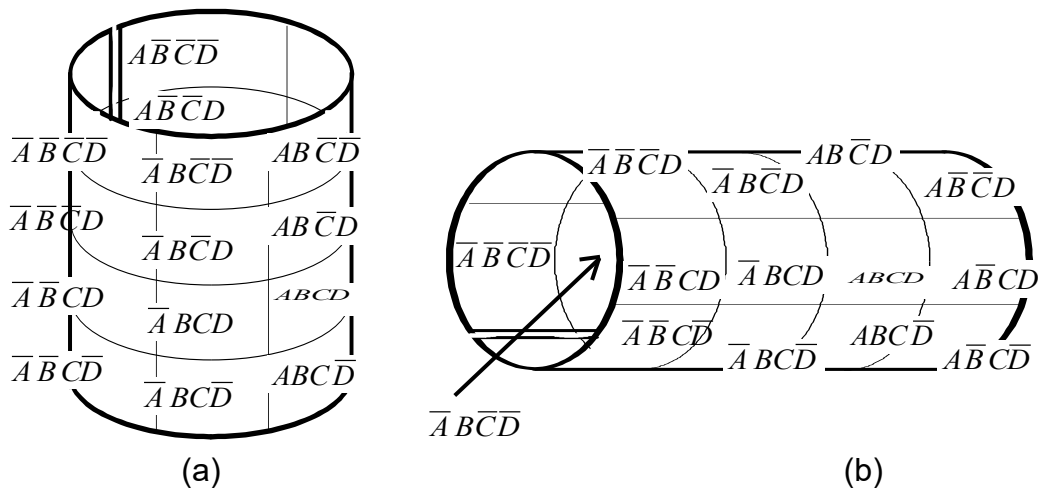


figure 2.6 : représentations cylindriques d'un diagramme de Karnaugh à quatre variables

- (a) mise en évidence des adjacences entre lignes
 (b) mise en évidence des adjacences entre colonnes

2.2.3.4. Fonctions de 5 et 6 variables

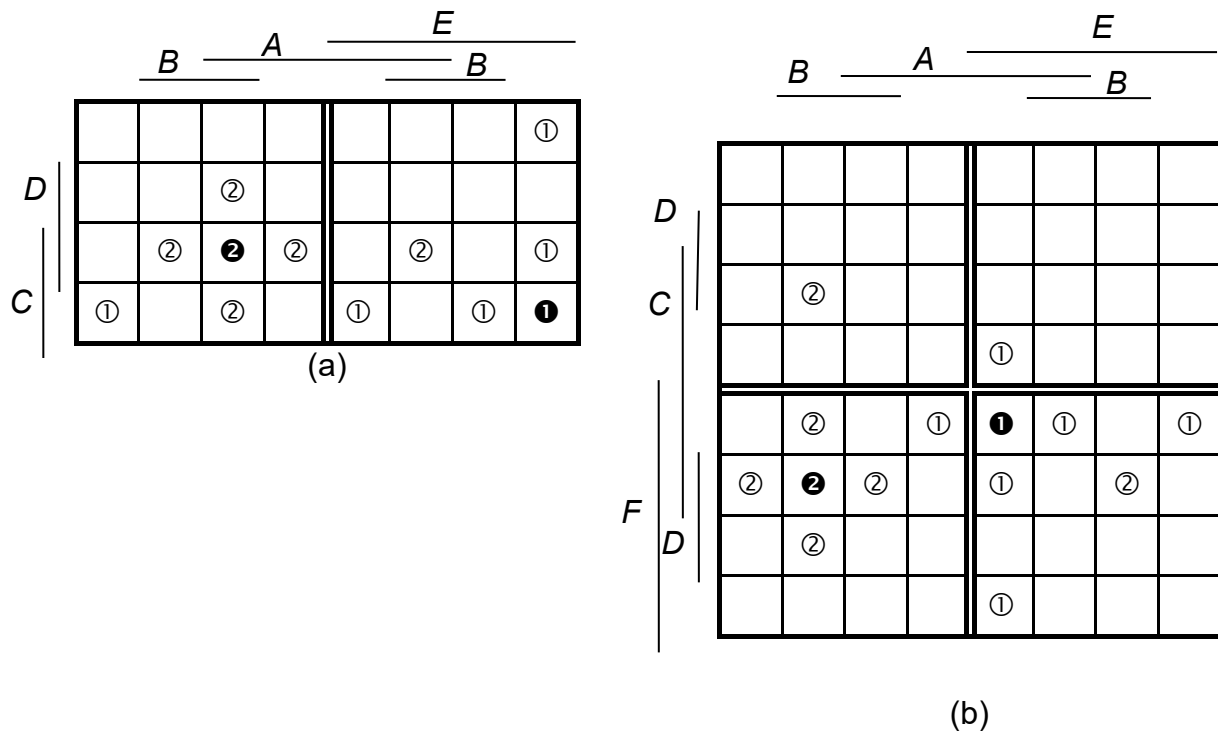


figure 2.7 : forme générale d'un diagramme de Karnaugh à 5 variables (a) et à 6 variables (b)

(a) termes adjacents à $\overline{A}\overline{B}C\overline{D}E$ (symbole ❶) et à $ABCD\overline{E}$ (symbole ❷)

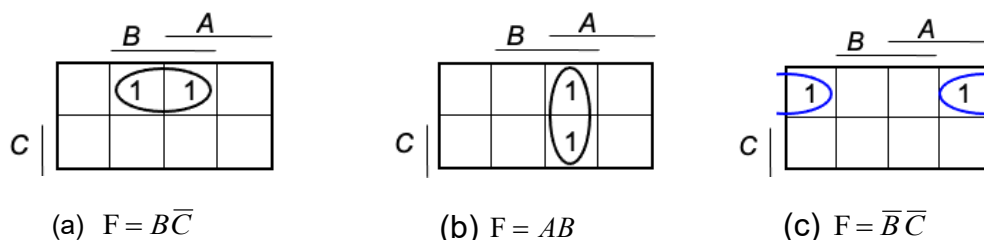
(b) termes adjacents à $\overline{A}\overline{B}CDEF$ (symbole ❶) et à $\overline{A}BCD\overline{E}F$ (symbole ❷)

On notera qu'à partir de 5 variables, le repérage des termes adjacents devient beaucoup plus délicat, et qu'une représentation similaire à celle de la figure 2.6 est irréalisable. La limite de cette méthode, utilisée « à la main », est donc liée au problème de visualisation des adjacences.

2.2.4. Principe de la simplification

On repère les cases adjacentes contenant un 1 et on les regroupe par paquets de 2^n . Un regroupement par 2^n correspond à la simplification par n variables.

- A titre d'exemple, pour une fonction de 3 variables (cf. figure 2.8)



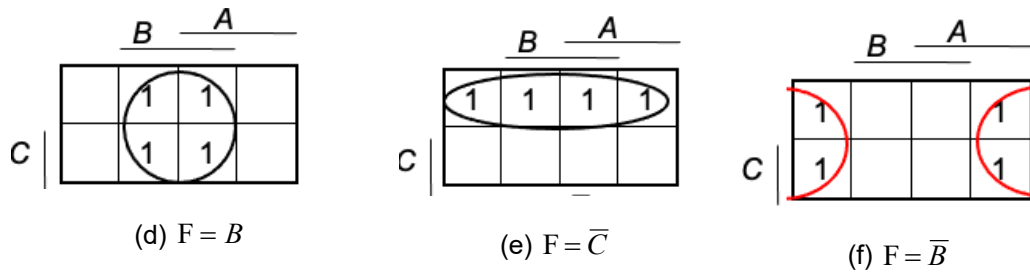


figure 2.8 :

- 1 case correspond à un minterme donc à un produit des 3 variables,
- 2 cases groupées représentent un produit de 2 variables : il y a simplification par la variable intervenant à la fois sous forme directe et sous forme complémentée. Trois exemples sont présentés dans les tableaux (a), (b), et (c) de la figure 2.8. Ainsi, pour le diagramme (a), $F = \bar{A}B\bar{C} + A\bar{B}\bar{C} = (\bar{A} + A)B\bar{C} = B\bar{C}$.
- 4 cases groupées représentent un « produit » de 1 variable, comme l'illustrent les diagrammes (d), (e), et (f). Par exemple, le diagramme (d) donne

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC = \bar{A}B(\bar{C} + C) + AB(\bar{C} + C) = \bar{A}B + AB = (\bar{A} + A)B = B.$$
Toutes les variables intervenant à la fois sous forme directe et sous forme complémentée sont éliminées.
- 8 cases regroupées conduisent alors naturellement à $F = 1$.

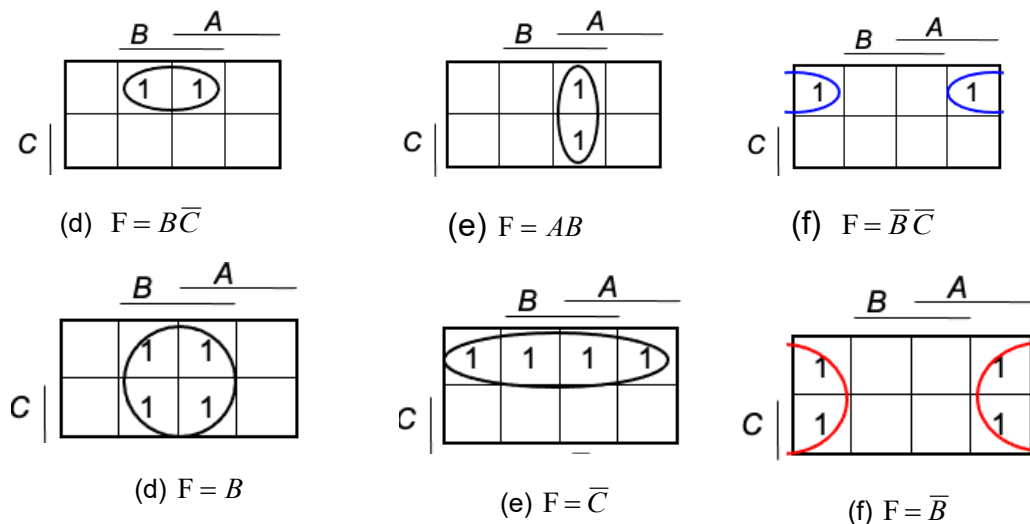


figure 2.8 : exemples de regroupements dans des diagrammes de Karnaugh à 3 variables

Dans le cas plus général où plusieurs regroupements sont possibles, il faut remarquer qu'une case peut être utilisée plusieurs fois, en raison de la propriété d'idempotence de la fonction OU : $A + A + \dots = A$. Considérons les exemples de la figure 2.9 dans le cas de fonctions de 4 variables :

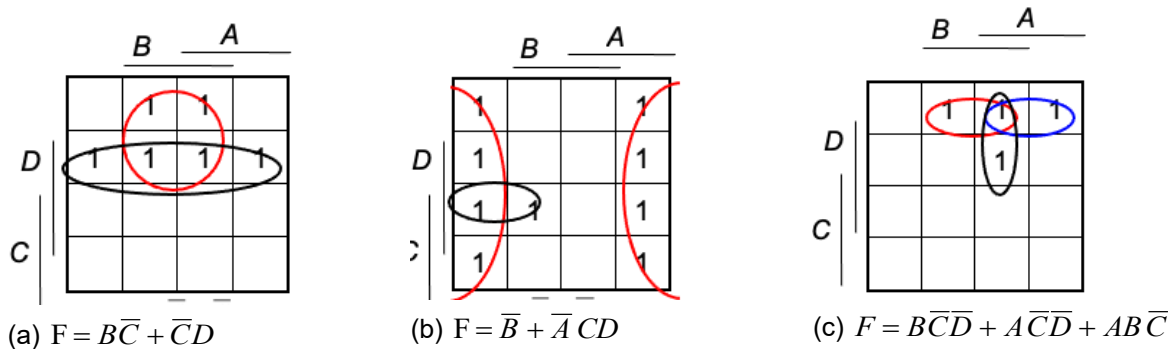


figure 2.9 : exemples de regroupements multiples dans des diagrammes de Karnaugh de 4 variables

2.2.4.1. *Technique à appliquer sur un diagramme de Karnaugh quelconque*

Pour obtenir une expression simplifiée minimale, il faut inclure tous les 1 du tableau dans des groupements de taille 2^n en respectant les principes suivants :

- Essayer de minimiser le nombre de groupements afin de minimiser le nombre de termes dans l'expression de la fonction. Il est alors préférable de rechercher les groupements en commençant par les cases qui ne peuvent se grouper que d'une seule façon. Ceci permet d'utiliser chaque 1 un minimum de fois.
- Vérifier que toutes les cases d'un groupe partagent le même nombre d'adjacences avec leurs congénères du groupe (soit n adjacences pour un groupe de 2^n cases).
- Les groupements de 1 doivent être les plus grands possibles (minimisation du nombre de variables).

2.2.4.2. Exemples

- **Exemple 1** : Simplification de

$$F_1 = \overline{A} \overline{B} \overline{C} \overline{D} + AB \overline{C} \overline{D} + \overline{A} \overline{B} \overline{C} D + \overline{A} B \overline{C} D + AB \overline{C} D + A \overline{B} \overline{C} D + \overline{A} \overline{B} C D + ABCD + A \overline{B} C D + \overline{A} B C \overline{D}.$$

(a) diagramme de Karnaugh de F_1

(b) identification des groupes de taille 4

(c) identification des groupes de taille 2

(d) identification des cases isolées

figure 2.10 : simplification de la fonction F_1

Après regroupement des 1 suivant les règles définies précédemment (figure 2.10), on obtient la forme simplifiée suivante : $F_1 = \overline{C}D + AD + \overline{B}D + \overline{A} \overline{B} \overline{C} + AB \overline{C} + \overline{A} B C \overline{D}$.

• **Exemple 2** : Simplification de

$$F_2 = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}BCD + ABCD + \overline{A}\overline{B}C\overline{D}.$$

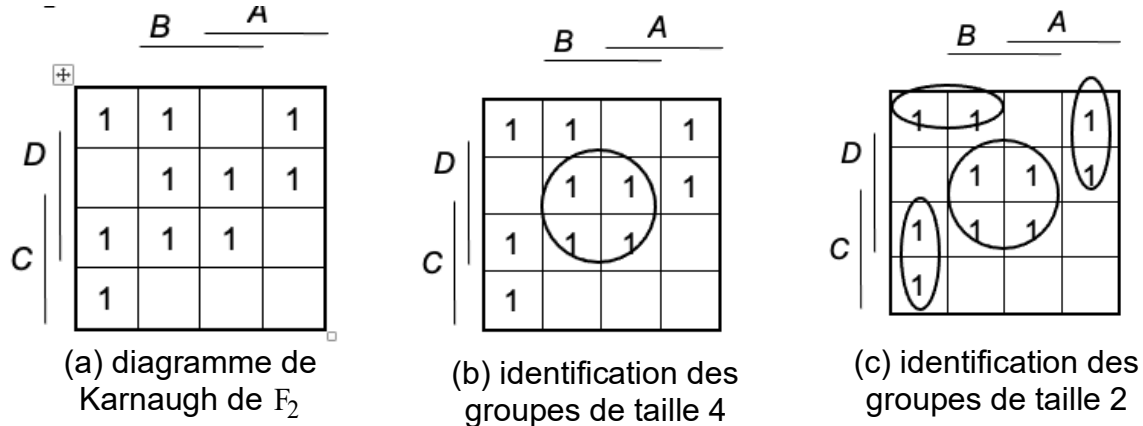


figure 2.11 : simplification de la fonction F_2

Les regroupements ont été effectués de façon à minimiser le nombre de groupes. On obtient alors $F_2 = BD + \overline{A}\overline{C}\overline{D} + A\overline{B}\overline{C} + \overline{A}\overline{B}C$.

- **Exemple 3** : On va traiter ici un exemple de simplification à partir de la représentation de la fonction sous la seconde forme canonique. On place alors dans le tableau les 0 correspondant aux maxtermes intervenant dans l'expression de la fonction : $F_3 = (\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(A + \overline{B} + C)$.

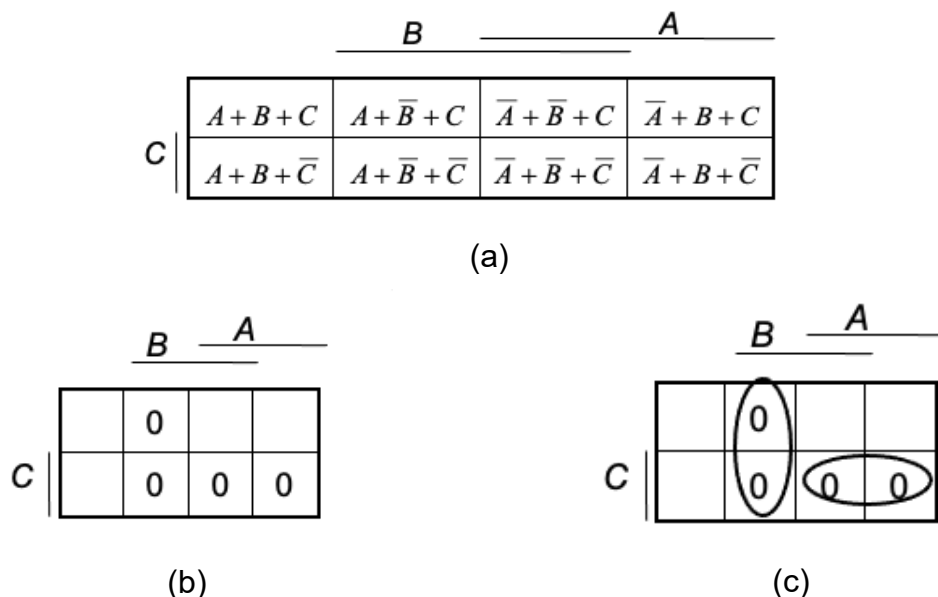


figure 2.12 : simplification de F_3 (représentée sous la deuxième forme canonique)

- (a) position des maxtermes dans un diagramme de Karnaugh à 3 variables
- (b) diagramme de la fonction F_3 : placement des 0
- (c) résultat du regroupement des 0

On obtient, après simplification : $F_3 = (\bar{A} + \bar{C})(A + \bar{B})$.

La lecture du résultat de regroupement des 0 est telle qu'une variable d'entrée égale à 1 correspond à cette variable complémentée dans l'expression finale. Ainsi dans l'exemple de la $F_3 = (\bar{A} + \bar{C})(A + \bar{B})$, la 1^{ère} combinaison correspond à A=1 et C=1. Dans la 2^{de} combinaison, A=0 et B=1.

Une autre façon de faire consiste à d'abord obtenir la fonction simplifiée $\bar{F}$, en considérant les 0 comme des 1 dans le tableau de Karnaugh, et en en faisant la lecture correspondante (comme si on cherchait la 1^{ère} forme canonique). Puis de complémenter la fonction simplifiée à l'aide du théorème de De Morgan, pour obtenir F.

Ainsi on lit $\bar{F}_3 = (\bar{A}.B) + (A.C)$ puis on complémente et on applique le théorème de De Morgan :

$$F_3 = \overline{(\bar{A}.B) + (A.C)} = (A + \bar{B}).(\bar{A} + \bar{C})$$

Remarque : Le résultat de la simplification peut ne pas être unique. Par exemple, soit $F_4 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + ABC + A\bar{B}\bar{C} + A\bar{B}C$. La figure 2.13 donne deux résultats de simplification de même complexité.

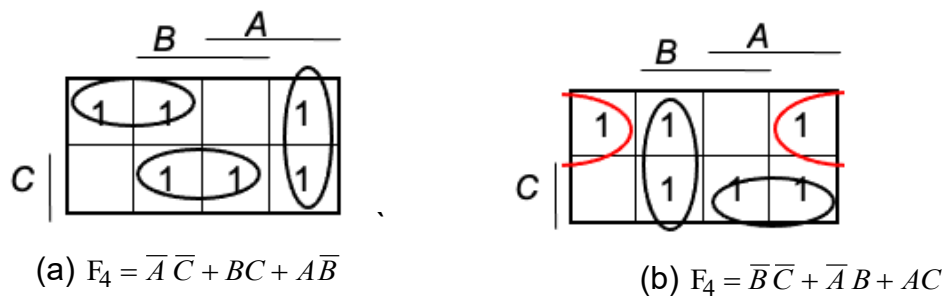


figure 2.13 : simplification de F_4

2.2.4.3. Cas des fonctions incomplètement spécifiées

Une fonction est dite **incomplètement spécifiée** si, pour certaines combinaisons des variables, elle prend indifféremment la valeur 0 ou la valeur 1, ou bien si l'occurrence de ces combinaisons n'est pas prévue dans la définition de la fonction. Dans ce cas, on a l'habitude d'inscrire un "X" ou un "-" dans les cases associées à ces combinaisons. On parle alors d'**états indifférents** ou **indéterminés** pour désigner les cases du diagramme correspondantes. Ces états peuvent être utilisés partiellement ou totalement pour simplifier la fonction. Les règles suivantes doivent alors être respectées :

- Effectuer d'abord les regroupements sans tenir compte des cases X ou -,
- Utiliser ensuite les cases X ou - pour réunir les groupes préexistants ou augmenter leur taille,

- Ne jamais utiliser une case X ou - pour créer un nouveau groupe, ceci irait à l'encontre du principe de minimisation du nombre de termes.
- **Exemple** : Soit la fonction F_5 définie par le diagramme de la figure 2.14(a) (forme $\Sigma\Pi$).

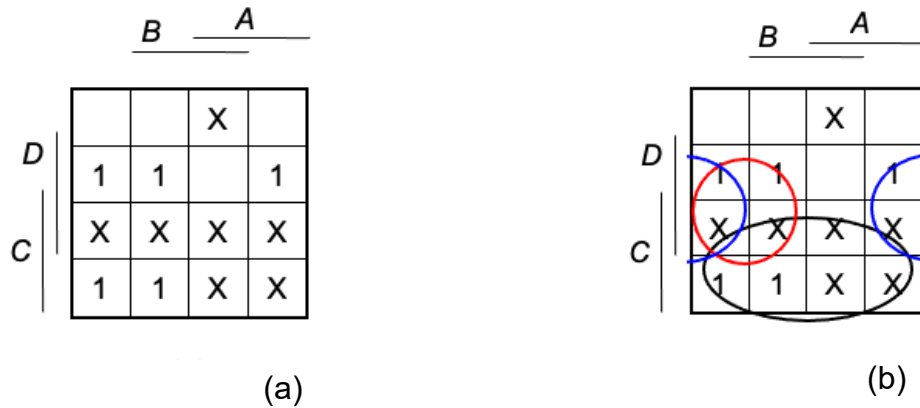
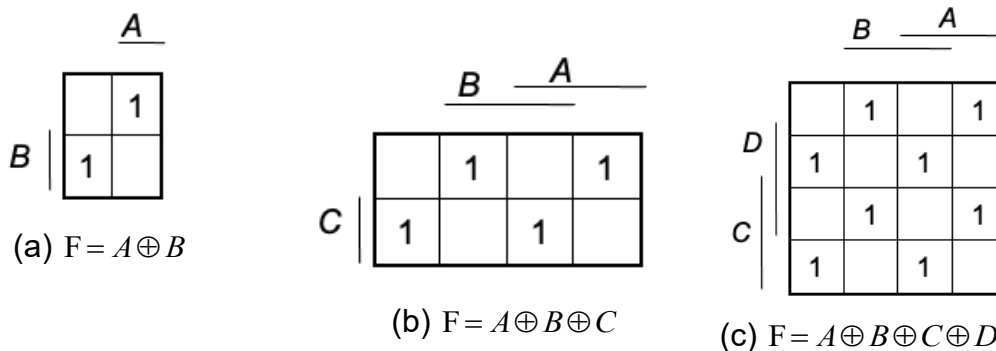


figure 2.14 : exemple de fonction incomplètement spécifiée

La présence d'états indifférents permet d'optimiser les regroupements comme indiqué sur la figure 2.14(b). Ainsi, pour l'écriture de F_5 sous forme simplifiée, les états indéterminés insérés dans les groupements sont considérés comme des 1 et les autres comme des 0. On obtient alors $F_5 = C + \bar{A}D + \bar{B}D$.

2.2.4.4. Les cas du OU exclusif et du ET inclusif

L'expression booléenne $A \oplus B \oplus C \oplus \dots$ vaut 1 si elle contient un nombre impair de 1. Si l'on complète le diagramme de Karnaugh correspondant, on observe que, en raison du codage de Gray, le tableau a l'aspect d'un damier (Figure 2.15).



complexité des circuits (plusieurs centaines de milliers d'opérateurs élémentaires par circuit en technologie CMOS).

2.4. REFERENCES

- [Dan96] J. D. Daniels, *Digital design from zero to one*, John Wiley & Sons, 1996.
- [GR87a] M. Gindre et D. Roux, *Electronique numérique : logique combinatoire et technologie, cours et exercices*, McGraw-Hill, Paris, 1987.
- [LV86] J.-C. Lafont et J.-P. Vabre, *Cours et problèmes d'électronique numérique*, Ellipses, 1986.
- [Sas93] T. Sasao, *Logic synthesis and optimization*, Kluwer Academic Publishers, 1993.
- [TP94] J.-L. Danger, C. Degois, A. Galisson, J. Leroux, D. Roux, *Composants et fonctions de l'électronique numérique, cours*, polycopié Télécom Paris, 1994.

3. EXERCICES D'APPLICATION

3.1. SIMPLIFICATION DES FONCTIONS A 3 ENTREES

Simplifier, par la méthode des diagrammes de Karnaugh, les fonctions booléennes suivantes :

1. $F(A,B,C) = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$
2. $F(A,B,C) = \overline{A}\overline{B}\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}$
3. $F(A,B,C) = \overline{A}\overline{B}C + A\overline{B}\overline{C} + AB\overline{C}$, sachant que la valeur de F pour les états $\overline{A}BC$ et ABC est indifférente.
4. $F(A,B,C) = (A+B+C).(A+\overline{B}+\overline{C}).(\overline{A}+\overline{B}+C).(\overline{A}+\overline{B}+\overline{C}).(\overline{A}+B+C)$
Utiliser les zéros du tableau de Karnaugh et donner le résultat sous forme conjonctive.

3.2. SIMPLIFICATION DES FONCTIONS A 4 ENTREES

Simplifier, par la méthode des diagrammes de Karnaugh, les fonctions booléennes suivantes

1. À faire à la maison :

$$F(A,B,C,D) = \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + AB\overline{C}\overline{D} + \overline{A}BCD + A\overline{B}CD + \overline{A}\overline{B}C\overline{D} + ABC\overline{D}$$

2. $F(A,B,C,D) = (A+B+C+D)(A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})(A+\overline{B}+\overline{C}+D)(\overline{A}+B+\overline{C}+D)$

Donner le résultat sous les deux formes algébriques, conjonctive et disjonctive.

3. À faire à la maison :

$$F(A,B,C,D) = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D}, \text{ sachant que quatre combinaisons de variables sont impossibles : } AB\overline{C}D, ABCD, \overline{A}\overline{B}C\overline{D} \text{ et } \overline{A}\overline{B}C\overline{D}.$$

4. $F(A,B,C,D)$ prend la valeur 1 pour les combinaisons suivantes des variables booléennes A, B, C, et D : $\overline{A}\overline{B}\overline{C}\overline{D}, \overline{A}B\overline{C}\overline{D}, \overline{A}B\overline{C}D, \overline{A}BC\overline{D}, \overline{A}BCD, \overline{A}\overline{B}C\overline{D}, \overline{A}\overline{B}CD$. La valeur de F peut être quelconque pour les combinaisons $\overline{A}\overline{B}C\overline{D}, \overline{A}BCD, A\overline{B}CD, \overline{A}\overline{B}CD$, et $AB\overline{C}\overline{D}$.

3.3. SIMPLIFICATION ALGEBRIQUE A PARTIR DES PROPRIETES DES FONCTIONS LOGIQUES

Simplifier algébriquement les fonctions suivantes :

1. $F_1 = (X+Y)(\overline{X}+Y)$
2. À faire à la maison $F_2 = \overline{X}\overline{Y} + XY + \overline{X}Y$

4. POUR ALLER PLUS LOIN

4.1. SIMPLIFICATION ALGEBRIQUE

Elle consiste à appliquer les propriétés de l'algèbre de Boole (cf. PC4) aux expressions algébriques des fonctions logiques. Il s'agit principalement de « recettes » dont l'application demande un peu d'entraînement. Nous allons traiter quelques exemples qui permettront de passer en revue la plupart des astuces utilisées pour mener à bien les simplifications.

Dans les exemples suivants, la technique consiste à regrouper judicieusement les termes puis à simplifier en utilisant les relations de simplification vues à la PC4.

4.1.1. Exemple 1 : simplification de $F_1 = BC + AC + AB + B$.

$AB + B = B$, donc $F_1 = BC + AC + B$,
d'où $\boxed{F_1 = AC + B}$, car $BC + B = B$.

4.1.2. Exemple 2 : simplification de $F_2 = (A + \bar{B})(A\bar{B} + C)C$.

$(X + C)C = C$, d'où $\boxed{F_2 = (A + \bar{B})C = AC + \bar{B}C}$.

4.1.3. Exemple 3 : simplification de $F_3 = \bar{A}B\bar{C} + AB\bar{C} + ABC + \bar{A}BC$

Il suffit de remarquer que $AX + \bar{A}X = (A + \bar{A})X = X$, et l'expression devient $\boxed{F_3 = B\bar{C} + BC = B}$.

Dans d'autres cas, il faut « compliquer » l'expression en utilisant les propriétés d'idempotence du ET et du OU, ou les propriétés de l'inversion, pour éliminer des termes superflus.

4.1.4. Exemple 4 : simplification de $F_4 = \bar{A}B + AC + BC$.

L'expression reste inchangée si le troisième terme est multiplié par 1 :

$$F_4 = \bar{A}B + AC + (A + \bar{A})BC.$$

En utilisant la distributivité de ET par rapport à OU, on obtient

$$\begin{aligned} F_4 &= \bar{A}B + AC + ABC + \bar{A}BC \\ &= \bar{A}B(1 + C) + AC(1 + B) \end{aligned}$$

d'où $\boxed{F_4 = \bar{A}B + AC}$.

4.1.5. Exemple 5 : simplification de $F_5 = (\bar{A} + B)(A + C)(B + C)$.

On ne change pas F_5 en ajoutant 0 à l'un des termes :

$$F_5 = (\bar{A} + B)(A + C)(B + C + \bar{A}A).$$

En utilisant ensuite la distributivité de OU par rapport à ET, on obtient

$$F_5 = (\bar{A} + B)(A + C)(\bar{A} + B + C)(A + C + B) \\ = (\bar{A} + B + 0.C)(A + C + 0.B)$$

d'où $\boxed{F_5 = (\bar{A} + B)(A + C)}$.

Il peut également être utile de savoir reconnaître le OU exclusif et son complément !

4.1.6. Exemple 6 : simplification de $F_6 = (A\bar{B} + \bar{A}B).(AB + \bar{A}\bar{B}).$

On remarque que $F_6 = (A \oplus B).(\overline{A \oplus B})$, d'où $\boxed{F_6 = 0}$.

4.1.7. Conclusion

Les méthodes algébriques de simplification présentent un inconvénient majeur : elles ne sont pas systématiques, et leur efficacité dépend donc largement du savoir-faire de la personne qui les applique. Elles ne peuvent, par conséquent, être utilisées que ponctuellement sur des cas simples. Nous ne les utiliserons que ponctuellement dans les exercices.

 IMT Atlantique Bretagne-Pays de la Loire Ecole Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD4 – CIRCUITS COMBINATOIRES

Sommaire

PCD4 – CIRCUITS COMBINATOIRES	1
1. LES OBJECTIFS D'APPRENTISSAGE	2
2. LES CONCEPTS	3
2.1 QU'EST-CE QU'UNE FONCTION COMBINATOIRE ET UN CIRCUIT COMBINATOIRE ?.....	3
2.2 AVANT-PROPOS : CLASSIFICATION DES CODES BINAIRES	3
2.2.1 Codes pondérés	3
2.2.2 Codes non pondérés	4
2.2.3 Codes alphanumériques	5
2.3 LES OPERATEURS DE TRANSCODAGE.....	5
2.3.1 Définition	5
2.3.2 Les codeurs.....	5
2.3.3 Les décodeurs	8
2.3.4 Les transcodeurs.....	13
2.4 LES OPERATEURS D'AIGUILLAGE	15
2.4.1 Les multiplexeurs.....	15
2.4.2 Les démultiplexeurs.....	18
2.4.3 Exemples de démultiplexeurs	18
2.4.4 Applications des démultiplexeurs.....	19
2.5 LES OPERATEURS DE COMPARAISON	20
2.5.1 Définition	20
2.5.2 Exemple de comparateurs	21
3. LES EXERCICES D'APPLICATION	23
3.1 MUX A TOUT FAIRE	23
3.2 DECODEUR BINAIRE / DEMULTIPLEXEUR.....	23
3.3 COMPAREUR DE MOTS BINAIRES	23
3.3.1 Compareur élémentaire.....	24
3.3.2 Extension à 2 mots de 2 bits	24
3.3.3 Généralisation (à faire à la maison)	25
3.4 FONCTION LOGIQUE COMBINATOIRE ET ASSURANCE (A FAIRE A LA MAISON)	25
4. POUR ALLER PLUS LOIN.....	26
4.1 AUTRES CODES BINAIRES.....	26
4.1.1 Code excédent 3 ou excess 3.....	26
4.1.2 Codes redondants	26
4.2 DECODEURS DE GRANDE CAPACITE (NB D'ENTREES SUPERIEUR A 3).....	27
4.2.1 Matrice de décodage	27
4.2.2 Association de décodeurs	28
1.1.1. Décodage à plusieurs niveaux.....	29
4.3 ASSOCIATION DE MULTIPLEXEURS.....	30

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD4 – Circuits combinatoires

1. LES OBJECTIFS D'APPRENTISSAGE

OA6 À partir d'une description d'une fonction à réaliser, synthétiser le circuit combinatoire répondant au problème à partir de fonctions logiques combinatoires élémentaires.

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

2.1 QU'EST-CE QU'UNE FONCTION COMBINATOIRE ET UN CIRCUIT COMBINATOIRE ?

On appelle **opérateur** ou **fonction combinatoire** un opérateur logique dont les sorties ne dépendent, à chaque instant, que de l'état de ses entrées. A chaque configuration des entrées correspond une configuration des sorties et une seule. Un circuit numérique réalisant une fonction combinatoire est un **circuit combinatoire**.

Outre les opérateurs élémentaires cités au chapitre 2, on distingue comme opérateurs combinatoires standard :

- les opérateurs de transcodage,
- les opérateurs d'aiguillage,
- les opérateurs de comparaison,
- les opérateurs arithmétiques.

Avertissement : L'étude de ces différents opérateurs est illustrée par des exemples de circuits standard issus de catalogues de fabricants de circuits intégrés. Les références complètes de ces circuits contiennent, outre un numéro caractéristique de la fonctionnalité du circuit, des informations complémentaires inutiles dans le cadre de ce chapitre (fabricant, technologie de fabrication, produit commercial ou militaire, etc.). Nous nous limiterons donc, ici, à fournir une référence générique n'indiquant que la fonctionnalité du circuit. A titre d'exemple, « *xx85* » représente tous les circuits dont la référence se termine par 85, numéro désignant un comparateur de deux mots de 4 bits. Les fiches techniques (**data sheets**) des circuits cités dans ce chapitre peuvent être consultées dans des catalogues tels que [Mot], [TI], ou [NS], par exemple.

2.2 AVANT-PROPOS : CLASSIFICATION DES CODES BINAIRES

Le champ d'application des systèmes numériques est très étendu. Lorsque l'application ne nécessite pas de calculs arithmétiques, les codages présentés à la PC3 sont inutiles ou peu adaptés. On utilise alors des codages possédant d'autres propriétés. On emploie ainsi dans certains systèmes des codes permettant d'éviter des états transitoires parasites lors de la saisie de données, ou de visualiser facilement des chiffres ou des lettres, ou bien encore de détecter des erreurs et/ou de les corriger dans un résultat susceptible d'être erroné. Nous présentons ci-après quelques codes fréquemment utilisés.

L'ensemble des codes binaires peuvent être regroupés en deux classes : les **codes pondérés** et les **codes non pondérés**.

2.2.1 Codes pondérés

Un code est dit **pondéré** si la position de chaque symbole dans chaque mot correspond à un poids fixé : par exemple 1, 10, 100, 1000 ... pour la numération décimale, et 1, 2, 4, 8, ... pour la numération binaire. Les codes pondérés ont, en général, des propriétés intéressantes du point de vue arithmétique.

2.2.1.1. *Le code binaire pur et ses dérivés (octal, hexadécimal)*

Ce sont les codes utilisés en arithmétique binaire et qui ont été étudiés dans la PC3.

2.2.1.2. *Le code DCB (Décimal Codé Binaire) ou BCD (Binary-Coded Decimal)*

Ce code est utilisé dans de nombreux systèmes **d'affichage**, de **comptage** ou même les **calculatrices** de poche. Dans le code BCD chaque chiffre d'un nombre décimal (de $0_{(10)}$ à $9_{(10)}$) est codé à l'aide de 4 bits (de $0000_{(2)}$ à $1001_{(2)}$). Ainsi le code BCD n'utilise que 10 **mots de codes** de 4 bits. Par exemple la représentation du nombre $1995_{(10)}$ est : $(0001\ 1001\ 1001\ 0101)_{(BCD)}$. Il est possible d'effectuer des opérations arithmétiques en BCD, mais celles-ci sont plus complexes qu'en binaire classique. Ce code est pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100, ...

2.2.2 Codes non pondérés

Dans le cas des codes non pondérés, il n'y a pas de poids affecté à chaque position des symboles. On convient simplement d'un tableau de correspondance entre les objets à coder et une représentation binaire. De tels codes peuvent néanmoins parfois posséder des propriétés arithmétiques intéressantes, comme le code **excédent 3**.

2.2.2.1. *Code binaire réfléchi ou code de Gray*

Ce code numérique n'étant pas pondéré, il est peu employé pour les opérations arithmétiques. Il est, par contre, utilisé pour le codage des déplacements angulaires, linéaires ou pour la réalisation des tableaux de Karnaugh (cf. chapitre « Propriétés des variables et fonctions logiques »). La propriété principale de ce code est que **deux mots successifs du code ne diffèrent que par un élément binaire**. Ceci permet, d'une part d'éviter la génération d'aléas (états parasites) au passage de deux combinaisons successives, et d'autre part de tirer parti de cette adjacence du codage pour simplifier les fonctions.

L'appellation "binaire réfléchi" provient de sa technique de construction : on peut construire un code de Gray sur n bits à partir d'un code de Gray sur $n-1$ bits en procédant comme suit : on copie les mots du code de départ, précédés d'un 0, suivis des mots du même code, pris dans **l'ordre inverse** et précédés d'un 1. Ceci permet de construire un code de Gray de n'importe quel format (cf. **Error! Reference source not found.**).

Une seule entrée du codeur doit normalement être activée à la fois. Dans le cas où le code en sortie est le code binaire pur, le circuit correspondant possède N entrées et n sorties, avec $2^{n-1} < N \leq 2^n$.

2.3.2.1. Exemples de codeurs

- **Exemple 1 : codeur décimal vers binaire (10 entrées vers 4 sorties)**

Ce codeur reçoit un chiffre décimal sur une des dix entrées et génère l'équivalent binaire sur les sorties $A0$ à $A3$. Une seule entrée doit être active à la fois.

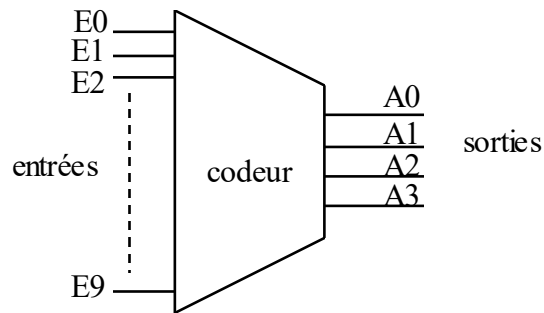


figure 4.2 : codeur décimal vers binaire

On trouvera ci-après (tableau 4.1) la table de vérité simplifiée et les équations de fonctionnement de ce type de codeur.

Entrée activée (= 1)	Sorties			
	$A3$	$A2$	$A1$	$A0$
$E0$	0	0	0	0
$E1$	0	0	0	1
$E2$	0	0	1	0
$E3$	0	0	1	1
$E4$	0	1	0	0
$E5$	0	1	0	1
$E6$	0	1	1	0
$E7$	0	1	1	1
$E8$	1	0	0	0
$E9$	1	0	0	1

tableau 4.1 : table de vérité réduite du codeur 10 vers 4

Equations logiques :

$$A0 = E1 + E3 + E5 + E7 + E9$$

$$A1 = E2 + E3 + E6 + E7$$

$$A2 = E4 + E5 + E6 + E7$$

$$A3 = E8 + E9$$

Cet opérateur, qui n'existe pas sous forme de circuit intégré standard, peut facilement être réalisé à partir de portes élémentaires.

- **Exemple 2 : codeur binaire 8 vers 3**

Ce codeur reçoit une information codée sur une de ses huit entrées et génère l'équivalent binaire sur les sorties $A0$ à $A2$. Une seule entrée doit être active à la fois.

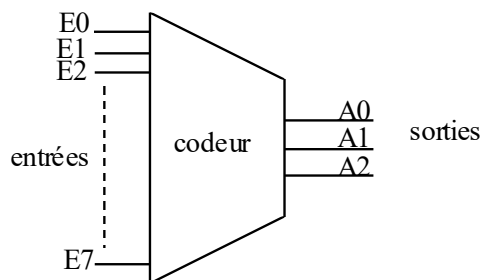


figure 4.3 : codeur 8 vers 3

Entrée activée (= 1)	Sorties		
	$A2$	$A1$	$A0$
$E0$	0	0	0
$E1$	0	0	1
$E2$	0	1	0
$E3$	0	1	1
$E4$	1	0	0
$E5$	1	0	1
$E6$	1	1	0
$E7$	1	1	1

tableau 4.2 : table de vérité réduite du codeur 8 vers 3

Equations logiques :

$$A0 = E1 + E3 + E5 + E7$$

$$A1 = E2 + E3 + E6 + E7$$

$$A2 = E4 + E5 + E6 + E7$$

Dans les deux exemples précédents il n'y a en théorie qu'une seule entrée active parmi N . Si, dans une application, plusieurs entrées peuvent être actives simultanément il faut utiliser un codeur **prioritaire**.

2.3.2.2. Codeur prioritaire

Ce type de codeur fixe un ordre de priorité entre les entrées. Dans le cas d'un encodage en binaire pur, le codeur prioritaire donne en général la priorité à l'entrée de poids le plus élevé. Par exemple, si les entrées 2, 8 et 9 sont activées simultanément, le codage de sortie correspondra à l'entrée 9. Ce circuit permet de détecter le rang du premier bit positionné à 1 dans un mot d'entrée.

- **Exemple : encodeur prioritaire 4 vers 2**

L'opérateur de la figure 4.4 est un encodeur prioritaire possédant 4 entrées ; l'entrée $E3$ correspond à l'entrée la plus prioritaire, et l'entrée $E0$ correspond à l'entrée la moins prioritaire

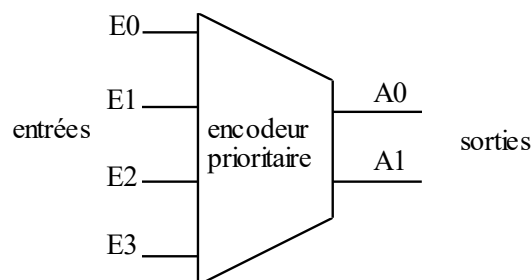


figure 4.4 : encodeur prioritaire 4 vers 2

Sa table de vérité et ses équations de fonctionnement sont présentées ci-après (tableau 4.3).

$E3$	$E2$	$E1$	$E0$	$A1$	$A0$
1	X	X	X	1	1
0	1	X	X	1	0
0	0	1	X	0	1
0	0	0	1	0	0

tableau 4.3: table de vérité de l'encodeur prioritaire 4 vers 2

Après simplification, nous obtenons les équations suivantes :

$$A0 = E3 + \overline{E2} E1$$

$$A1 = E3 + E2$$

2.3.3 Les décodeurs

Un décodeur est un opérateur à n entrées et N sorties avec $N \leq 2^n$. Une sortie du décodeur est activée lorsqu'une configuration particulière du code est affichée en entrée. Suivant le nombre de sorties, le décodeur peut décoder toutes les configurations possibles du code en entrée ou une partie seulement. Un décodeur à n entrées, permettant de décoder toutes les configurations du code binaire pur sur n bits, possède 2^n sorties. Une seule sortie est activée à la fois. En plus des n entrées, certains décodeurs possèdent une ou plusieurs entrées de validation. Par exemple, pour une validation active au niveau bas, si l'entrée de validation V vaut 0, alors le décodage est autorisé ; dans le cas contraire ($V = 1$), les sorties sont inhibées et

restent inactives. Ces entrées de validation permettent de grouper plusieurs décodeurs afin d'augmenter l'amplitude de décodage (voir section 4.2).

2.3.3.1. Les principaux types de décodeurs

a. Les décodeurs binaires

Ce type d'opérateur permet d'associer une sortie parmi 2^n avec une information d'entrée codée sur n bits.

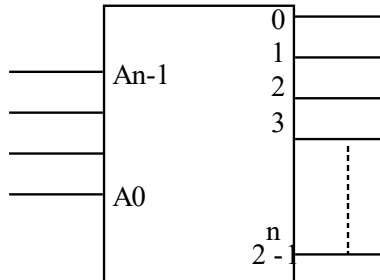


figure 4.5 : décodeur binaire « n vers 2^n » ou « 1 parmi 2^n »

• Exemple 1 : décodeur 2 vers 4

Le tableau 4.4 présente le fonctionnement d'un décodeur binaire 2 vers 4 ou 1 parmi 4.

$E1$	$E0$	$S0$	$S1$	$S2$	$S3$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

tableau 4.4 : table de vérité d'un décodeur binaire 2 vers 4

Dans cet exemple, on a choisi d'associer la valeur "0" lorsque la sortie est décodée (sorties actives à 0). Les équations de fonctionnement de ce circuit sont les suivantes :

$$\overline{S0} = \overline{E1} \cdot \overline{E0}$$

$$\overline{S1} = \overline{E1} \cdot E0$$

$$\overline{S2} = E1 \cdot \overline{E0}$$

$$\overline{S3} = E1 \cdot E0$$

La réalisation du logigramme de ce composant à l'aide d'opérateurs logiques élémentaires ne présente pas de difficulté particulière.

• Exemple 2 : décodeur 3 vers 8

Contrairement à l'exemple précédent, on associe la valeur logique "1" lorsque le rang de la sortie active correspond à la valeur binaire présentée en entrée (sorties actives à 1). La table de vérité (tableau 4.5) comporte 8 sorties qui correspondent aux mintermes des variables d'entrée.

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

tableau 4.5 : table de vérité du décodeur 3 vers 8

Les sorties du décodeur sont données par les relations suivantes :

$$S0 = \overline{C} \overline{B} \overline{A}$$

$$S1 = \overline{C} \overline{B} A$$

...

$$S7 = C B A$$

Une réalisation possible du décodeur 3 vers 8 est immédiate :

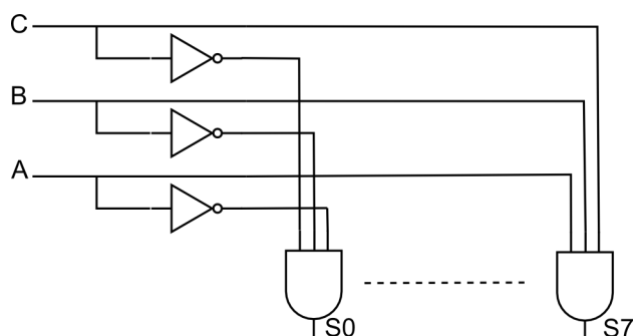


figure 4.6 : réalisation d'un décodeur 3 vers 8

b. Le décodeur BCD

Le code BCD (Binary-Coded Decimal, cf. chapitre 1, section 2.5.1.2) est utilisé pour coder les dix chiffres décimaux. Ce code à 4 bits laisse donc inutilisées 6 combinaisons sur les 16 possibles. Lorsqu'une combinaison, comprise entre 0 et 9, est appliquée sur les entrées $ABCD$ (A est le bit de poids fort), la sortie correspondante est validée. Les sorties restent au repos (niveau 1 par exemple) dans le cas où une combinaison comprise entre 10 et 15 est appliquée sur les entrées.

Table de vérité :

A	B	C	D	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

tableau 4.6 : table de vérité du décodeur BCD

Représentation symbolique usuelle :

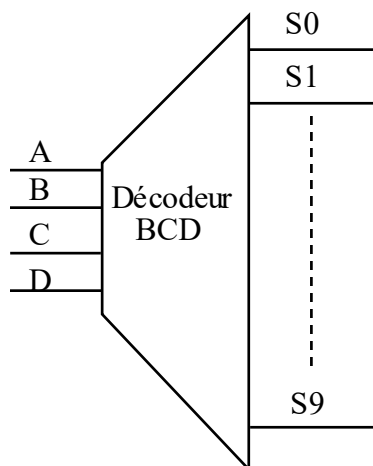


figure 4.7 : entrées et sorties du décodeur BCD

2.3.3.2. Applications des décodeurs

Les applications des décodeurs sont nombreuses. Nous avons choisi de les illustrer à travers trois exemples.

- **Exemple 1 : décodage d'adresse d'une mémoire ou sélection de composants dans une carte microprocesseur**

Dans une carte mère, qui possède en général un microprocesseur et plusieurs composants annexes (RAM, ROM, REPRON, boîtiers d'entrées-sorties, etc.), la

fonction de décodage est essentielle afin de ne valider qu'un seul boîtier à la fois. Les signaux qui servent au décodage, et donc à la sélection des boîtiers sont issus du bus d'adresse et du bus de contrôle du microprocesseur. La figure 4.8 montre un synoptique simplifié d'une telle application des décodeurs.

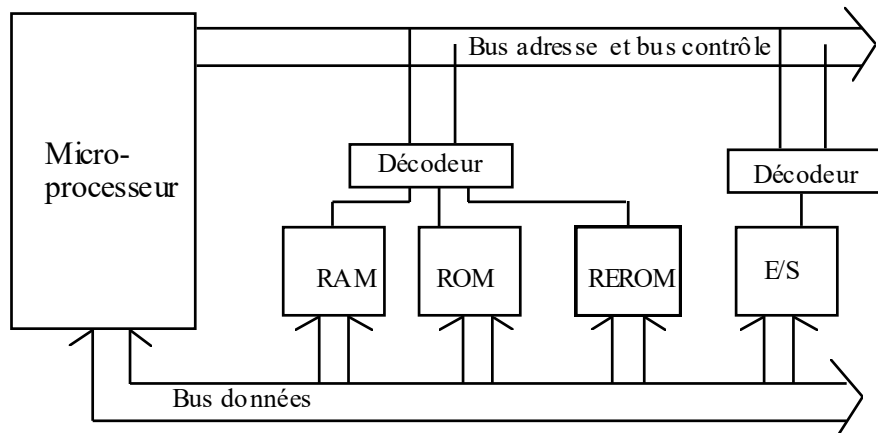


figure 4.8 : sélection d'un composant par décodeur

- **Exemple 2 : générateur de fonction**

Nous avons vu dans la section a que les sorties S_i d'un décodeur 3 vers 8 étaient régies par les équations suivantes :

$$S0 = \overline{C} \overline{B} \overline{A}$$

$$S1 = \overline{C} \overline{B} A$$

...

$$S7 = C B A$$

On remarque que chaque sortie S_i représente un des mintermes des variables A , B , et C . En réunissant ces mintermes on peut donc calculer n'importe quelle fonction de trois variables à l'aide d'un décodeur 3 vers 8 associé à des portes OU. Il est cependant à noter que l'utilisation de décodeurs en générateur de fonctions est en pratique peu fréquente.

- **Exemple 3 : démultiplexage**

Le décodeur est parfois utilisé pour **démultiplexer** des informations. Dans ce type d'application, son rôle consiste à aiguiller une information présente sur une entrée vers une sortie choisie parmi les N disponibles.

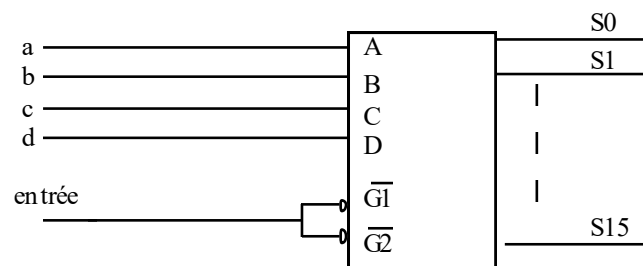


figure 4.9 : décodeur XX 154 utilisé pour démultiplexer

Dans l'exemple de la figure 4.9, les informations multiplexées sont injectées dans les entrées de validation ($\overline{G1}$ et $\overline{G2}$) du décodeur 4 vers 16. Les entrées A , B , C , et D possèdent ici un rôle d'entrées d'adressage. Elles servent à aiguiller l'information présente sur $\overline{G1}$ et $\overline{G2}$ vers une des 16 sorties S_i .

N. B. Les sorties du décodeur $xx154$ sont actives au niveau bas (à 0).

2.3.4 Les transcodeurs

Ces opérateurs permettent de convertir un nombre écrit dans un code C1 vers un code C2.

2.3.4.1. Exemple 1 : le transcodeur BCD/7 segments

Le transcodeur BCD/7 segments permet de commander un afficheur alphanumérique possédant 7 segments (des diodes électroluminescentes, par exemple). Cet afficheur permet la visualisation des chiffres 0 à 9 codés en binaire naturel sur 4 bits D , C , B , et A , où A représente le bit de poids le plus faible.

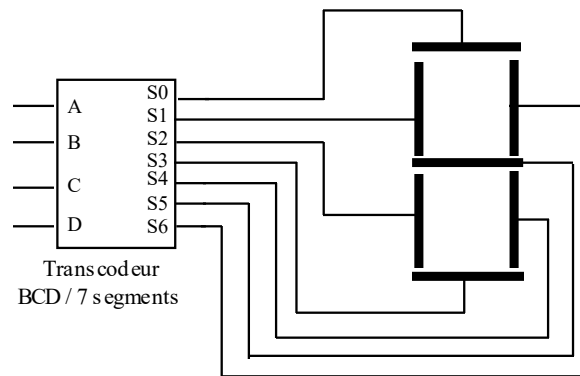


figure 4.10 : commande d'un afficheur par transcodeur

Les équations logiques du transcodeur de la figure 4.10 s'établissent sans difficultés à l'aide de 7 tableaux de Karnaugh :

$$\begin{aligned}
 S0 &= \overline{A}.\overline{C} + A.C + B + D \\
 S1 &= \overline{A}.C + \overline{A}.\overline{B} + \overline{B}.C + D \\
 S2 &= \overline{A}.\overline{C} + \overline{A}.B \\
 S3 &= A.\overline{B}.C + \overline{A}.\overline{C} + \overline{A}.B + B.\overline{C} + D \\
 S4 &= A + \overline{B} + C \\
 S5 &= \overline{B}.C + \overline{A}.B + B.\overline{C} + D \\
 S6 &= A.B + \overline{C}
 \end{aligned}$$

Ces transcodeurs existent sous la forme de circuits standard proposés par les fabricants sous les références $xx46$ à $xx49$. Ces circuits possèdent des entrées de contrôle supplémentaires : extinction de l'affichage des segments, allumage de tous les segments, etc. D'autres types de transcodeurs sont capables de commander des afficheurs alphanumériques. Les fonctions de trancodages sont parfois intégrées dans les afficheurs.

2.3.4.2. **Exemple 2 : les convertisseurs Gray/binaire et binaire/Gray**

Le code de Gray ou code binaire réfléchi (cf. chapitre 1, section 2.5.2.2) est largement utilisé dans les systèmes numériques, notamment dans les capteurs de position (pour coder des positions angulaires, par exemple). En effet, un seul bit change entre deux positions successives, et les risques d'informations parasites lors des transitions sont éliminés.

Les équations logiques d'un convertisseur de code Gray/binaire ou binaire/Gray s'établissent sans problème à l'aide de la méthode de Karnaugh à partir de la table de vérité suivante :

Décimal	Binaire				Gray			
	B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

tableau 4.7 : passage du code binaire au code Gray sur 4 bits

Pour des mots codés sur n bits, on obtient :

- **Convertisseur Gray/binaire**

$$B_n = G_n$$

et

$$B_{n-i} = G_n \oplus G_{n-1} \oplus \dots \oplus G_{n-i} \text{ pour } i = 1, \dots, n-1$$

- **Convertisseur binaire/Gray**

$$G_i = B_i \oplus B_{i+1} \text{ pour } i = 1, \dots, n-1$$

et

$$G_n = B_n$$

2.4 LES OPERATEURS D'AIGUILLAGE

On distingue deux classes d'opérateurs d'aiguillage : les **multiplexeurs** et les **démultiplexeurs**.

2.4.1 Les multiplexeurs

Ce sont des opérateurs à $N = 2^n$ entrées d'information ou de données, $E_0, E_1, \dots, E_{N-2}, E_{N-1}$, n entrées de sélection ou d'adressage $A_{n-1}, A_{n-2}, \dots, A_0$, et une sortie S . L'affichage d'une adresse permet de sélectionner une entrée de données parmi N , pour l'aiguiller vers la sortie S .

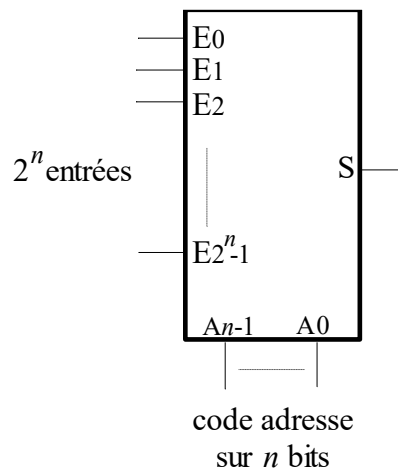


figure 4.11 : multiplexeur 2^n vers 1 ou $2^n : 1$

La fonction logique réalisée par le multiplexeur est régie par l'équation suivante :

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

où m_i est le $i^{\text{ème}}$ minterme des variables logiques $A_{n-1}, A_{n-2}, \dots, A_0$. Par exemple, $m_0 = \overline{A_{n-1}} \cdot \overline{A_{n-2}} \cdots \overline{A_1} \cdot \overline{A_0}$, $m_1 = \overline{A_{n-1}} \cdot \overline{A_{n-2}} \cdots \overline{A_1} \cdot A_0$, ...

La représentation symbolique usuelle et l'équation de sortie d'un multiplexeur 4 vers 1 sont donnés par la figure 4.12.

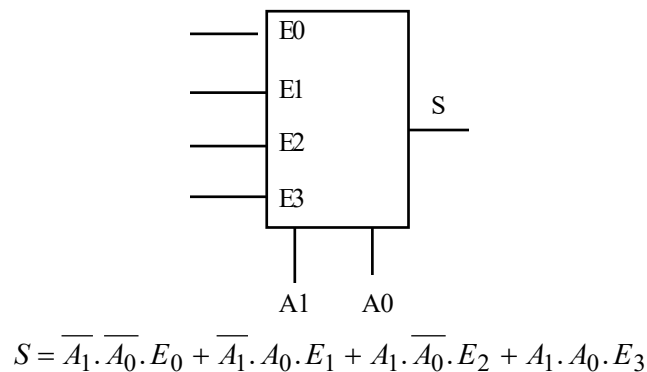


figure 4.12 : multiplexeur 4 vers 1

En général, les opérateurs de multiplexage disposent d'une entrée supplémentaire de validation V qui permet de valider ou d'inhiber la sortie S . Cette entrée est souvent utilisée afin d'augmenter les capacités de multiplexage à partir d'une fonctionnalité de base. Cet aspect est détaillé dans la section suivante.

2.4.1.1. Exemples de multiplexeurs

La figure 4.13 présente la réalisation d'un multiplexeur 2 vers 1 à l'aide de deux interrupteurs CMOS et d'un inverseur élémentaire. La structure inverseur CMOS a été présentée dans la partie Fonctions électroniques analogiques d'ELP111. Elle sera revue dans le cours C4.

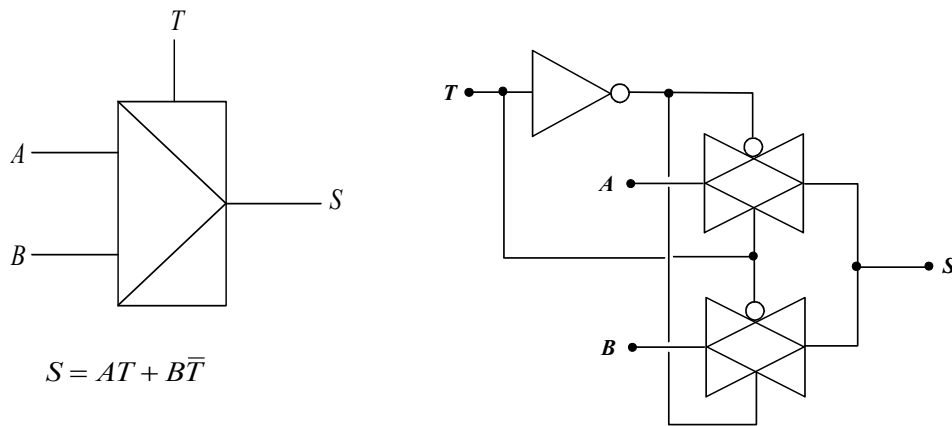


figure 4.13 : représentation usuelle et réalisation en logique CMOS d'un multiplexeur 2 vers 1

2.4.1.2. Multiplexage de mots

Certains types de multiplexeurs travaillent non pas sur des bits mais sur des mots de x bits. Les x bits qui forment le mot désigné par l'adresse sont traités simultanément et aiguillés vers les x bits de la sortie.

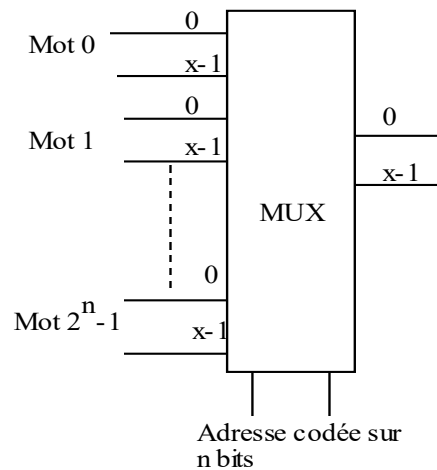


figure 4.14 : multiplexeur de 2^n mots de x bits

2.4.1.3. Applications des multiplexeurs

- **Conversion parallèle/série**

La fonction première du multiplexeur est d'aiguiller des informations provenant de N canaux vers un canal unique. Ce dispositif permet donc de convertir une information présente en parallèle sur les entrées d'information en une information de type série, si toutes les combinaisons d'adresses sont énumérées sur les entrées de sélection.

- **Génération de fonctions**

Les multiplexeurs peuvent également, tout comme les décodeurs, être utilisés pour la génération de fonctions logiques. En effet, toute fonction logique peut se décomposer sous la forme d'une somme de mintermes (cf. chapitre 2, section 3.1) :

$$f(x_{n-1}, \dots, x_1, x_0) = \sum_{i=0}^{2^n-1} d_i \cdot m_i$$

où m_i est le $i^{\text{ème}}$ minterme des variables logiques $x_{n-1}, \dots, x_1, x_0$ et $d_i = 0$ ou 1 .

Cette expression peut être rapprochée de l'équation de fonctionnement du multiplexeur. En effet, les mintermes m_i peuvent être identifiés avec les entrées d'adresse d'un multiplexeur. Les coefficients d_i sont identifiés avec les entrées d'information.

L'exemple du tableau 4.8 et de la figure 4.15 illustre la réalisation d'une fonction de deux variables à l'aide d'un multiplexeur 4 vers 1. Le passage de la table vérité au montage est immédiat.

A	B	F(A,B)
0	0	1
0	1	0
1	0	1
1	1	1

tableau 4.8 : table de vérité de la fonction F à réaliser

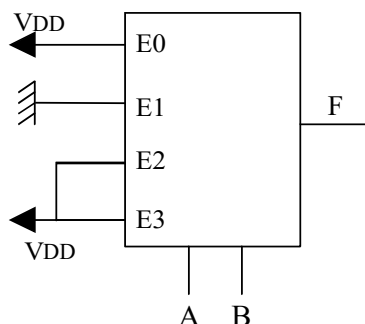


figure 4.15 : réalisation de la fonction F à l'aide d'un MUX 4 : 1

La réalisation de fonctions logiques à l'aide de multiplexeurs est en pratique utilisée dans certaines familles de circuits programmables (cf. ELP213 et SIT212). Par exemple, dans les FPGA (Field Programmable Gate Arrays) ACTEL, toutes les fonctions combinatoires sont réalisées à base de multiplexeurs.

2.4.2 Les démultiplexeurs

Ces opérateurs, comme leur nom l'indique, réalisent la fonction inverse des multiplexeurs. Ils possèdent une entrée d'information ou de données, n entrées de sélection (ou de commande), et $N = 2^n$ sorties. L'affichage d'une adresse sur les entrées de sélection permet de sélectionner la sortie vers laquelle l'entrée sera aiguillée. Le démultiplexeur peut, tout comme le multiplexeur, être doté d'une entrée de validation des sorties.

2.4.3 Exemples de démultiplexeurs

En pratique, les fabricants de circuits intégrés standard proposent dans leurs catalogues des circuits pouvant être utilisés en tant que décodeurs ou en tant que multiplexeurs (cf. références *xx* 137, *xx* 138, *xx* 139). A titre d'exemple, la table de vérité du tableau 4.9 et le schéma de la figure 4.16 illustrent le fonctionnement d'un décodeur/démultiplexeur 1 vers 4. Les circuits intégrés *xx* 139 comportent deux décodeurs/démultiplexeurs de ce type.

Entrées			Sorties			
Validation	Sélection					
$\bar{G}$	$S1$	$S0$	$Y0$	$Y1$	$Y2$	$Y3$
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

tableau 4.9 : table de vérité d'un décodeur/démultiplexeur 1 vers 4 du circuit de référence *xx* 139

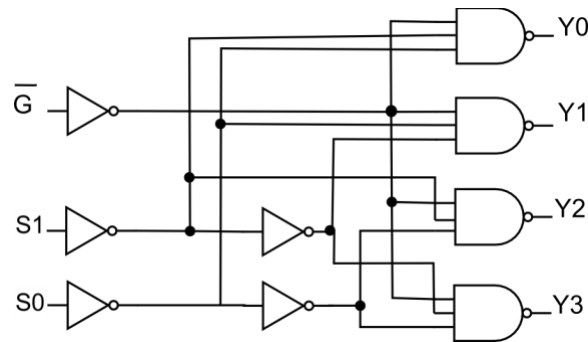


figure 4.16 : structure interne d'un décodeur/démultiplexeur du circuit *XX 139*

Lorsque ce circuit est utilisé **en décodage**, l'entrée $\bar{G}$, active à 0, est utilisée pour valider ou non les sorties. Dans ce mode de fonctionnement, les sorties sont également **actives à la valeur logique 0**.

En mode **démultiplexage**, $S1$ et $S0$ sont les entrées d'adresse et $\bar{G}$ joue le rôle d'entrée d'information. La valeur 0 de $\bar{G}$ est donc démultiplexée sur les sorties Y en fonction de la valeur de la sélection $S1S0$.

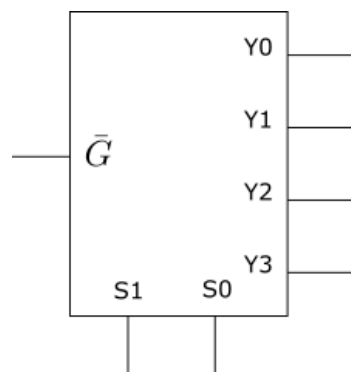


figure 4.17 : vue externe d'un décodeur/démultiplexeur du circuit *XX 139*

2.4.4 Applications des démultiplexeurs

Une des applications les plus classiques du démultiplexeur est la conversion d'une information présente en série en une information de type parallèle.

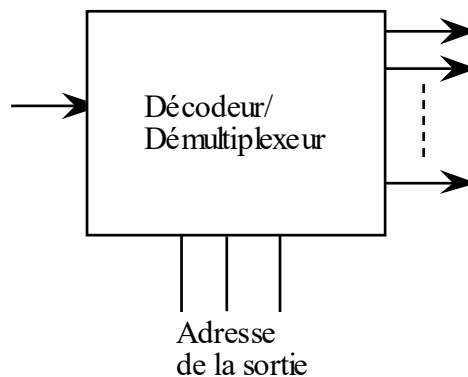


figure 4.18 : conversion série/parallèle

2.5 LES OPERATEURS DE COMPARAISON

2.5.1 Définition

On désigne par **opérateur de comparaison** ou **comparateur** un opérateur capable de détecter l'égalité ou l'inégalité de 2 nombres (comparateur simple) et éventuellement d'indiquer le plus grand ou le plus petit (comparateur complet). Le tableau 4.10 donne le résultat de la comparaison de 2 éléments binaires.

A	B	$A = B$	$A > B$	$A < B$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

tableau 4.10 : résultat de la comparaison de 2 bits A et B

Les équations logiques correspondantes sont les suivantes :

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

$$(A > B) = A \bar{B}$$

$$(A < B) = \bar{A} B$$

Dans le cas plus général de nombres binaires, deux nombres A et B , $A = A_{n-1} \dots A_i \dots A_0$ et $B = B_{n-1} \dots B_i \dots B_0$, sont égaux si tous les bits de même rang, A_i et B_i , sont égaux. L'équation de fonctionnement du circuit comparateur simple est alors la suivante :

$$S = \overline{A_0 \oplus B_0} \dots \overline{A_i \oplus B_i} \dots \overline{A_{n-1} \oplus B_{n-1}}$$

$$S = \prod_{i=0}^{n-1} \overline{A_i \oplus B_i}$$

soit encore :

$$S = \overline{(A_0 \oplus B_0) + (A_i \oplus B_i) + (A_{n-1} \oplus B_{n-1})}$$

$$S = \sum_{i=0}^{n-1} \overline{A_i \oplus B_i}$$

Cette détection peut être réalisée à l'aide d'opérateurs OU exclusif et d'une fonction NOR (figure 4.19).

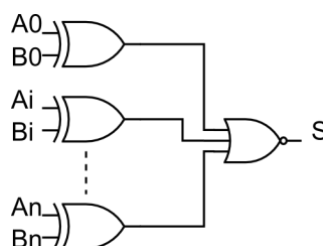


figure 4.19 : comparaison simple de 2 nombres A et B

Dans le cas d'une comparaison complète de deux nombres binaires A et B , les sorties $A < B$ et $A > B$ sont obtenues en effectuant une comparaison complète des bits de même rang, A_i et B_i , de façon prioritaire à partir des bits de poids forts (cf. tableau 4.11). Par exemple, si $A_n > B_n$ alors $A > B$, et si $A_n < B_n$ alors $A < B$. Mais si $A_n = B_n$, il est nécessaire de comparer les bits de rang $n-1$ pour décider, et ainsi de suite ...

2.5.2 Exemple de comparateurs

Le circuit standard de référence $xx85$ (figure 4.20) compare 2 mots de 4 bits A et B . Outre les entrées de données recevant les deux mots à comparer, il possède également trois entrées $A = B_{in}$, $A < B_{in}$, et $A > B_{in}$, permettant de cascader les comparateurs pour pouvoir comparer des nombres de plus de 4 bits. Si le comparateur est utilisé seul, les entrées $A = B_{in}$, $A < B_{in}$, et $A > B_{in}$ doivent être connectées respectivement à 1, 0, et 0. Le tableau 4.11 donne la table de vérité condensée de ce circuit.

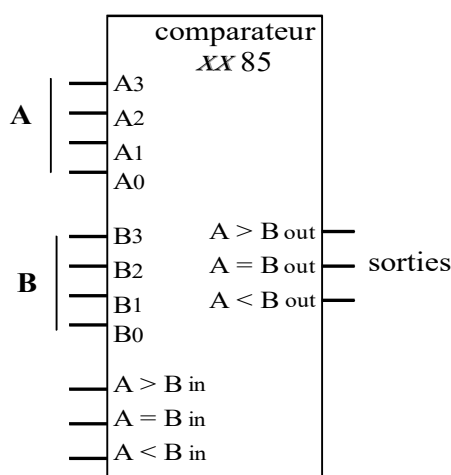


figure 4.20 : comparateur complet 4 bits $xx85$

Entrées de données				Entrées de mise en cascade			Sorties		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A > B_{in}$	$A = B_{in}$	$A < B_{in}$	$A > B_{out}$	$A = B_{out}$	$A < B_{out}$
$A_3 > B_3$	X	X	X	X	X	X	1	0	0
$A_3 < B_3$	X	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0

tableau 4.11 : table de vérité du comparateur 4 bits XX 85

A titre d'exemple, pour comparer deux nombres de 8 bits, il suffit de relier les sorties $A > B_{out}$, $A = B_{out}$, et $A < B_{out}$ du comparateur gérant les 4 bits de poids faibles aux entrées $A < B_{in}$, $A = B_{in}$, et $A > B_{in}$ du comparateur gérant les bits de poids forts. Dans ce cas, les valeurs logiques 010 sont appliquées sur les entrées $A < B_{in}$, $A = B_{in}$, et $A > B_{in}$ du premier comparateur.

3. LES EXERCICES D'APPLICATION

3.1 MUX A TOUT FAIRE

Soit un multiplexeur 8 vers 1. Nous souhaitons réaliser les 3 fonctions logiques combinatoires suivantes. Donner le schéma du circuit pour chacune d'entre elles, à partir du symbole du MUX 8 vers 1.

$$f_1 = \bar{A}.\bar{B}.\bar{C} + \bar{A}.B.C$$

$$f_2 = A.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + A.B.C \text{ (à la maison)}$$

$$f_3 = \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + A.B.\bar{C} + A.B.C \text{ (à la maison)}$$

3.2 DECODEUR BINAIRE / DEMULTIPLEXEUR

Soit un **décodeur binaire** à 2 entrées A_0 et A_1 et à 4 sorties S_0 , S_1 , S_2 , et S_3 tel qu'une seule sortie soit active à la fois et que le rang de la sortie active correspond à la valeur binaire présentée en entrée (ex : $A_0=A_1=0 \Rightarrow S_0=1$).

1. Etablir la table de vérité de ce décodeur.
2. Réaliser ce décodeur à l'aide d'un circuit démultiplexeur (actif à 1), dont le symbole est donné ci-dessous.

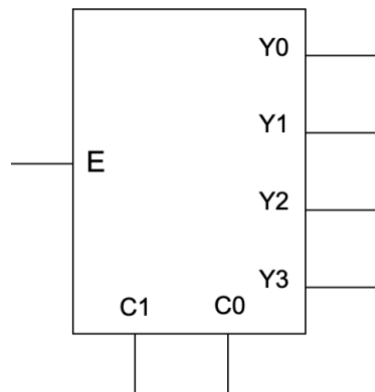


Figure. Symbole du démultiplexeur (actif à 1). 2 entrées de commande C1 et C0 et 4 sorties de Y0 à Y3. 1 entrée de donnée E.

3.3 COMPAREUR DE MOTS BINAIRES

On souhaite réaliser un **comparateur** de deux mots de n bits (figure 1)

$$A = (A_n \cdots A_i \cdots A_1) \text{ et } B = (B_n \cdots B_i \cdots B_1)$$

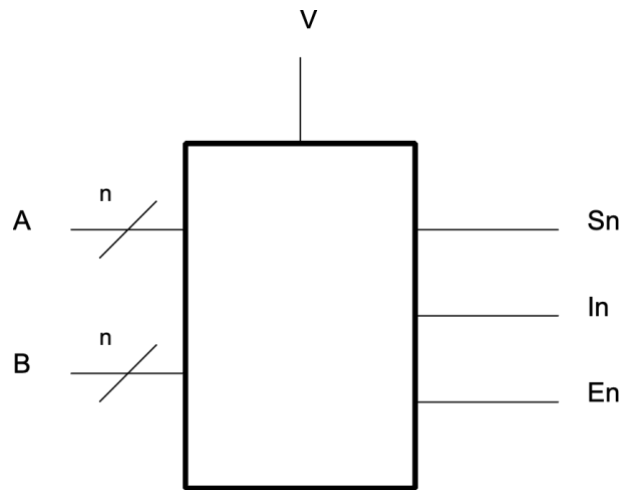


Figure. Symbole du comparateur.

Fonctionnement du comparateur :

- Si l'entrée de validation (V) vaut 0, alors les trois sorties (Egal : E_n), (Supérieur : S_n) et (Inférieur : I_n) sont égales à 0.
- Si l'entrée de validation (V) vaut 1, alors :
 - $S_n = 1$ ssi $A > B$
 - $I_n = 1$ ssi $A < B$
 - $E_n = 1$ ssi $A = B$

3.3.1 Comparateur élémentaire

Dans un premier temps, on souhaite réaliser un comparateur élémentaire de deux mots de 1 bit ($n = 1$).

- Etablir les équations des sorties S_1 , I_1 , et E_1 en fonction des entrées A_1 , B_1 , et V .
- Dessiner le schéma de ce comparateur à partir d'opérateurs élémentaires (INV, NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

3.3.2 Extension à 2 mots de 2 bits

On souhaite maintenant étendre l'amplitude du comparateur à deux mots de 2 bits.

1. Etablir les équations de S_2 , I_2 , et E_2 en fonction des bits A_2 , A_1 , B_2 , B_1 , et de l'entrée de validation V
2. Concevoir le schéma de ce comparateur en associant des comparateurs élémentaires et un minimum d'opérateurs (NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

3.3.3 Généralisation (à faire à la maison)

A partir des équations trouvées précédemment, établir les relations de récurrence ci-dessous :

$$S_n = f(S_{n-1}, A_n, B_n, V)$$

$$I_n = g(I_{n-1}, A_n, B_n, V)$$

$$E_n = h(E_{n-1}, A_n, B_n, V)$$

3.4 FONCTION LOGIQUE COMBINATOIRE ET ASSURANCE (A FAIRE A LA MAISON)

Les conditions de délivrance de la police d'assurance n° 15 sont les suivantes :

- avoir souscrit à la police n° 10, être du sexe féminin et mariée,

ou

- n'avoir pas souscrit à la police n° 10, être du sexe masculin et marié,

ou

- avoir souscrit à la police n° 10, être marié(e) et âgé(e) de moins de 25 ans,

ou

- être marié(e) et avoir plus de 25 ans,

ou

- être du sexe féminin et âgée de moins de 25 ans.

1. Donner la table de vérité de la fonction logique f telle que f=1 si la police d'assurance n°15 est délivrée.
2. Donner la fonction logique en utilisant la méthode de simplification de Karnaugh.
3. Donner le circuit à base d'opérateurs logiques combinatoires (OR, AND)

4. POUR ALLER PLUS LOIN

4.1 AUTRES CODES BINAIRES

4.1.1 Code excédent 3 ou excess 3

Le code excédent 3 est un **code non pondéré** qui utilise, tout comme le code BCD, 10 mots de codes, auxquels on fait correspondre les 10 chiffres décimaux.

$N_{(10)}$	$N_{(XS3)}$
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

tableau 12 : code excédent 3

Il est obtenu en décalant le code binaire de trois lignes vers le haut. Ce code peut être intéressant pour effectuer des soustractions car le complément à 1 de la représentation binaire d'un chiffre correspond au complément à 9 de ce chiffre. Ainsi, toute opération de soustraction se ramène à une addition.

Par exemple $5_{(XS3)} = 1000$, son complément à 1 est obtenu par inversion des bits, $\bar{5}_{(XS3)} = 0111 = 4_{(XS3)}$, le résultat en excédent 3 est le nombre 4, qui est le complément à 9 de 5 en décimal. Ainsi, pour faire une soustraction, il suffit d'ajouter le complément à 1 du nombre à retrancher, puis 1. Par exemple, $(7 - 5)_{(XS3)} = 7_{(XS3)} + \bar{5}_{(XS3)} + 1_{(XS3)} = 1010 + 0111 + 0100 = 0101 = 2_{(XS3)}$.

Le code excédent 3 ne présente pas d'intérêt en addition.

4.1.2 Codes redondants

Il existe un ensemble de codes conçus pour pouvoir **détecter**, voire **corriger des erreurs dans des messages binaires**. Leur principe repose sur l'insertion de données redondantes dans l'information initiale. Leur étude approfondie relève du **domaine des communications numériques**, et ne sera pas traité dans ce cours. Nous citerons cependant quelques exemples simples de codes redondants, les codes p parmi n et les codes de contrôle de parité.

4.1.2.1. Code p parmi n

Ce code est constitué de $C_n^p = \frac{n!}{p!(n-p)!}$ mots de code. Chaque mot de code est codé sur n bits et contient exactement p "1" et $(n - p)$ "0". Par exemple, le code 2 parmi 5

(**Error! Reference source not found.**) est constitué de 10 mots de codes et permet de coder les chiffres décimaux.

$N_{(10)}$	$N_{(2 \text{ parmi } 5)}$
0	0 0 0 1 1
1	1 1 0 0 0
2	1 0 1 0 0
3	0 1 1 0 0
4	1 0 0 1 0
5	0 1 0 1 0
6	0 0 1 1 0
7	1 0 0 0 1
8	0 1 0 0 1
9	0 0 1 0 1

tableau 13 : code 2 parmi 5

L'utilisation de ce code permet, à la réception d'une information, de vérifier par comptage du nombre de 1 si une erreur s'est introduite dans l'information transmise. Dans le cas où plus d'une erreur s'est glissée dans un mot de code, la détection n'est pas assurée dans tous les cas. Ce code ne permet pas non plus de trouver la place de l'erreur, donc de la corriger. D'autre part, le décodage des combinaisons est particulièrement simple, car il ne porte que sur 2 bits par combinaison.

4.1.2.2. Code contrôle de parité

Le codage d'un mot de n bits par contrôle de parité consiste à y adjoindre un $(n+1)^{\text{ème}}$ bit dont le rôle est de rendre systématiquement pair ou impair le nombre total de 1 contenus dans l'information codée. Si une erreur se glisse dans l'information, le nombre de 1 devient impair et l'erreur est détectée. Ce code ne permet pas non plus de corriger les erreurs.

4.2 DECODEURS DE GRANDE CAPACITE (NB D'ENTREES SUPERIEUR A 3)

4.2.1 Matrice de décodage

Dès que l'on souhaite réaliser un décodeur de plus de 3 bits en entrée, il est conseillé d'utiliser une structure en matrice (figure 4.21) pour obtenir un opérateur de complexité matérielle minimale.

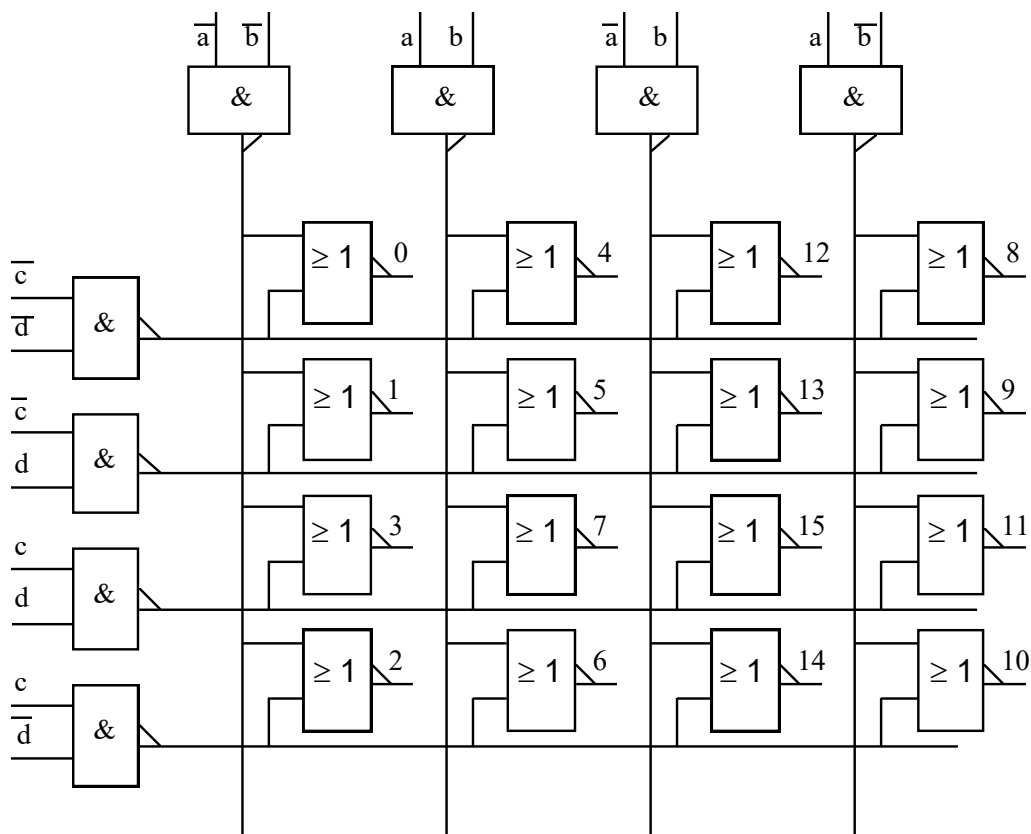


figure 4.21 : matrice de décodage de 4 bits

Dans la matrice de la figure 4.21, chaque case du tableau de Karnaugh ou chaque ligne de la table de vérité correspond à une structure matérielle composée de 2 portes NAND et d'une porte NOR. L'utilisation multiple des mintermes des variables c et d , ainsi que des variables a et b , permet réduire la complexité matérielle par rapport à un circuit décodant les seize sorties indépendamment les unes des autres. On notera également que tous les chemins entre entrées et sorties traversent des couches logiques identiques.

4.2.2 Association de décodeurs

Il est possible d'associer plusieurs décodeurs afin d'augmenter les capacités d'un système. La figure 4.22 montre comment deux décodeurs 1 parmi 16 (référence *xx* 154 des catalogues de circuits standard) sont associés pour former un décodeur de 1 parmi 32 (fonctionnalité qui n'existe pas chez les fabricants de circuits standard, mais qui peut être directement réalisée dans un circuit spécifique). Outre les entrées A , B , C , et D , le circuit *xx* 154 possède deux entrées de validation $\overline{G1}$ et $\overline{G2}$. La notation barrée ainsi que les ronds sur ces entrées signifient qu'elles sont actives au niveau logique 0 : les sorties du décodeur ne sont validées que lorsque $\overline{G1} = \overline{G2} = 0$.

Si l'entrée a vaut 0, les entrées $\overline{G1}$ et $\overline{G2}$ du décodeur **1** sont actives, celles du décodeur **2** inactives. Le décodeur **1** est validé, et le décodeur **2** inhibé. Les sorties $S0$ à $S15$ sont alors représentatives des entrées b , c , d , et e . Si l'entrée a vaut 1, seul le décodeur **2** est validé et les entrées b , c , d , et e sont décodées sur les sorties $S16$ à $S31$.

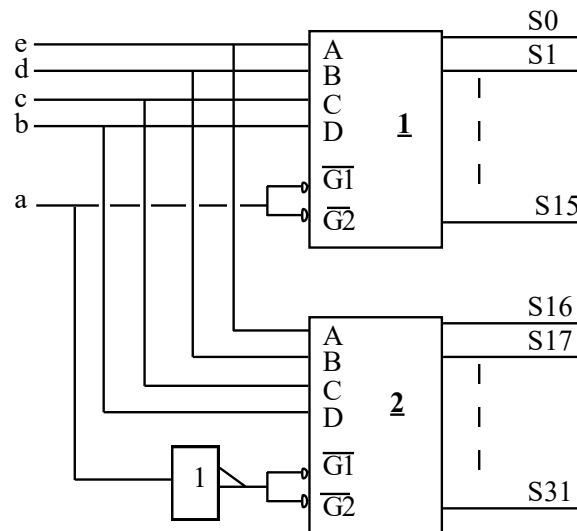


figure 4.22 : réalisation d'un décodeur 1 parmi 32 à partir de 2 décodeurs 1 parmi 16

1.1.1. Décodage à plusieurs niveaux

Afin de réaliser des circuits décodeurs de grande capacité, il est possible d'associer des décodeurs de base sur plusieurs niveaux.

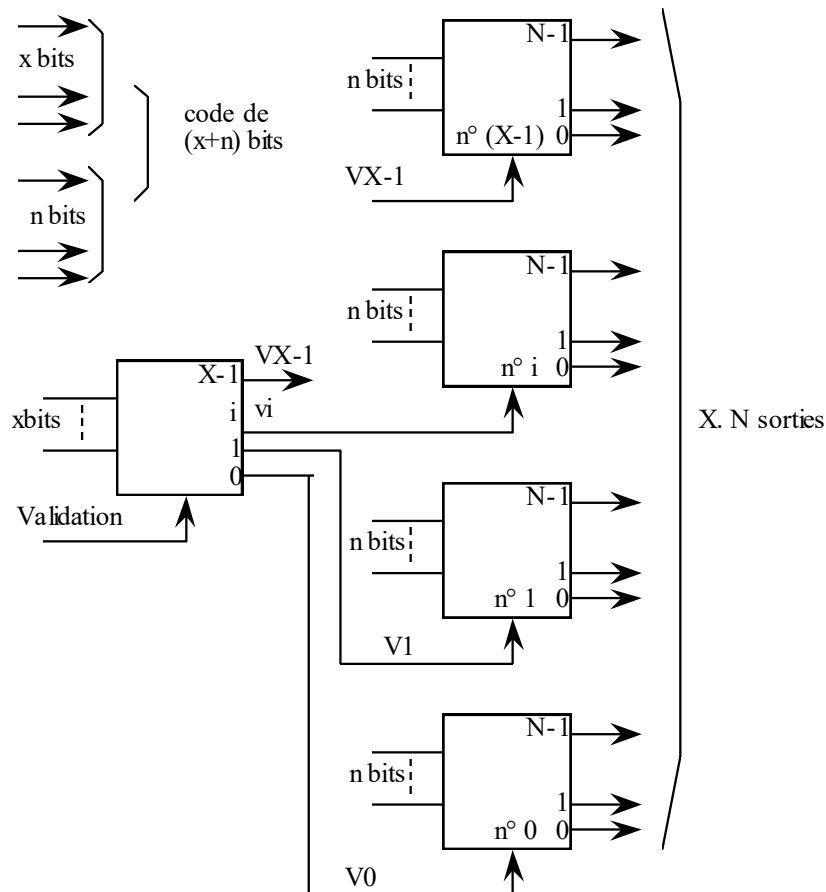


figure 4.23 : décodage de 1 parmi $X.N$ sur deux niveaux

Dans l'exemple de la figure 4.23, le dispositif, qui possède deux niveaux de décodeurs, accepte $x + n$ entrées et délivre $X.N$ sorties, avec $X = 2^x$ et $N = 2^n$. Le premier niveau

est constitué d'un décodeur de type 1 parmi X . Celui-ci reçoit x bits en entrées et fournit sur une des sorties $V0$ à $VX - 1$ un signal de sélection qui est utilisé pour valider un des X décodeurs du second niveau. Ces X décodeurs reçoivent en entrée les n bits restants et possèdent chacun N sorties. Le signal *Validation* appliqué sur le décodeur de premier niveau permet d'autoriser ou non le décodage.

4.3 ASSOCIATION DE MULTIPLEXEURS

Dans l'exemple de la figure 4.24, deux multiplexeurs élémentaires 8 vers 1 avec entrées de validation sont associés afin d'obtenir un dispositif capable d'aiguiller 1 entrée parmi 16 vers la sortie. On remarque que le fil d'adresse A_3 joue un rôle particulier. Si $A_3 = 0$, le multiplexeur 1 est validé, et le multiplexeur 2, qui correspond aux 8 bits de poids forts, est inhibé (sa sortie est forcée à 0). Les adresses A_0 à A_2 permettent donc de choisir une des 8 entrées du premier multiplexeur. Lorsque le signal A_3 vaut 1, le premier multiplexeur est inhibé (sortie à 0) et le second validé. Une porte OU permet de recueillir la sortie du multiplexeur validé.

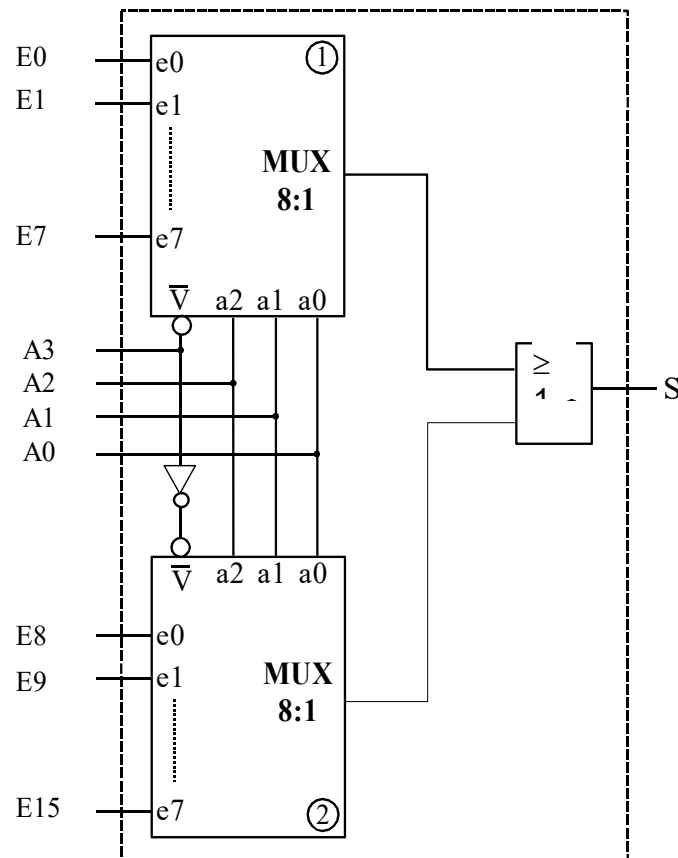


figure 4.24 : réalisation d'un multiplexeur 16 vers 1 à partir de deux multiplexeurs 8 vers 1

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance
PCD5 - ADDITIONNEUR BINAIRE	

Sommaire

PCD5 - ADDITIONNEUR BINAIRE.....	1
1. LES OBJECTIFS D'APPRENTISSAGE	2
2. LES CONCEPTS	3
2.1 OPERATIONS ARITHMETIQUES.....	3
2.1.1 Addition et soustraction.....	3
2.1.2 Multiplication et division.....	5
2.2 LES OPERATEURS ARITHMETIQUES.....	6
2.2.1 Les additionneurs	6
2.2.2 Les soustracteurs	9
2.2.3 Les multiplieurs et diviseurs.....	9
2.2.4 Les unités arithmétiques et logiques.....	9
2.3 BIBLIOGRAPHIE.....	9
3. LES EXERCICES D'APPLICATION	10
3.1 OPERATION ARITHMETIQUE : L'ADDITION, ALGORITHME ET CIRCUIT	10
3.1.1 Algorithme « scolaire »	10
3.1.2 Circuit Additionneur binaire complet	10
3.2 OPERATION ARITHMETIQUE : LA SOUSTRACTION, ALGORITHME ET CIRCUIT (A FAIRE A LA MAISON) ..	11
3.2.1 Algorithme « scolaire », quelques exemples	11
3.2.1.1 Circuit Additionneur/soustracteur.....	12
3.3 DEPASSEMENT DE CAPACITE : OVERFLOW (A FAIRE A LA MAISON).....	12
4. POUR ALLER PLUS LOIN.....	13
4.1 CAS GENERAL DE LA MULTIPLICATION.....	13
4.2 ADDITIONNEUR A RETENUE ANTICIPEE	14
4.3 LES MULTIPLIEURS ET DIVISEURS (OPERATEURS ARITHMETIQUES).....	16
4.4 EXEMPLE D'UNE UNITE ARITHMETIQUE ET LOGIQUE DU MARCHE : LE CIRCUIT INTEGRE DE REFERENCE	
xx 382 18	

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance PCD5 – Additionneur binaire

1. LES OBJECTIFS D'APPRENTISSAGE

OA7 Expliquer la différence entre un algorithme et une architecture matérielle

OA8 Produire l'architecture à retenue propagée de l'additionneur binaire

Enjeu : Notion de complexité dans un processeur matériel de traitement de l'information.

2. LES CONCEPTS

La section 2.1 traite des opérations arithmétiques effectuées sur les nombres binaires signés ou non signés : addition et soustraction, multiplication et division.

La section 2.2 traite des opérateurs arithmétiques, c'est-à-dire des circuits combinatoires permettant la mise en œuvre matérielle de ces opérations.

Ces opérateurs constituent les circuits élémentaires des unités arithmétiques et logiques qui constituent le cœur des micro-processeurs actuels.

Dans cette séance, nous partons de la fonctionnalité que l'on souhaite obtenir, l'addition, puis à partir de l'algorithme de calcul dit « scolaire » (addition posée en colonne), nous en déduisons une architecture matérielle. Cette architecture matérielle pourra être ensuite mise en œuvre dans un circuit programmable type FPGA (*Field Programmable Gate Array* ; voir module SIT212) ou bien dans un circuit intégré, de type ASIC (*Application Specific Integrated Circuit*) par exemple.

2.1 OPERATIONS ARITHMETIQUES

2.1.1 Addition et soustraction

Addition

Le principe de l'addition est, dans toutes les bases, similaire à celui de l'addition décimale : on additionne symbole par symbole en partant des poids faibles, et en propageant éventuellement une retenue.

Si le format des nombres est fixe (en binaire naturel ou en complément à 2), le résultat de l'addition peut donner lieu à un dépassement de capacité (*overflow*, en anglais). Cependant, selon le codage des nombres binaires, ce « problème » est géré différemment.

Dans le cas des nombres codés en binaire naturel, le résultat de l'addition de 2 nombres binaires codés sur n bits peut dépasser la plus grande valeur codable sur n bits (égale à $2^n - 1$ en binaire naturel). Dans ce cas, il suffit de prendre en compte la retenue sortante de l'addition, qui devient le bit de poids fort du résultat. Cela revient à prendre un bit supplémentaire sur le résultat (soit $n+1$ bits), pour obtenir le résultat correct.

Dans le cas où les nombres sont codés en complément à 2, ce n'est pas aussi simple.

Le résultat de l'addition de 2 nombres exprimés sur n bits sur le même format fournit toujours un résultat correct, sauf dans le cas où le résultat n'est pas représentable sur n bits. **Il y a alors dépassement de capacité. On observe que, dans ce cas, les deux opérandes sont de même signe et le résultat est de signe opposé (ce qui est incorrect).**

Dans le registre d'état d'un ordinateur, deux indicateurs (ou *flags*, en anglais) viennent renseigner le programmeur (ou le système d'exploitation) sur la validité des résultats obtenus : la retenue sortante (*carry C*) et le débordement (*overflow OVF*).

L'indicateur *C* signale la présence d'une retenue au niveau des poids forts; l'indicateur *OVF* indique que le résultat de l'opération n'est pas représentable **dans le système du complément à 2 sur le format défini au départ** (c'est-à-dire le nombre de bits fixé des opérandes). Ces 2 indicateurs sont directement générés par le circuit arithmétique présent dans le micro-processeur.

⚠ le circuit arithmétique ne connaît pas le format des nombres qui lui sont donnés. Il calcule toujours : le résultat sur n bits, et les 2 indicateurs C et OVF .

Nous allons illustrer le positionnement de la retenue et du débordement par quatre exemples pour des nombres binaires signés (dans le format CA2) :

+ 0000 0110 (+6)	+ 0111 1111 (+127)	0000 0100 (+4)	0000 0100 (+4)
+ 0000 0100 (+4)	+ 0000 0001 (+1)	+ 1111 1110 (-2)	+ 1111 1100 (-4)
0000 1010 (+10)	1000 0000 (-128)	1 0000 0010 (+2)	1 0000 0000 (0)
OVF=0	OVF=1	OVF=0	OVF=0
C=0	C=0	C=1 ignoré	C=1 ignoré
résultat correct	résultat incorrect	résultat correct	résultat correct

Dans le 1^{er} exemple, +10 est représentable sur 8 bits, le résultat est donc correct. Dans le 2^{ème} exemple, +128 n'est pas représentable sur 8 bits, on obtient ainsi -128 en représentation CA2. Le résultat de l'addition de 2 nombres positifs est un nombre négatif et l'indicateur OVR est positionné à 1.

Pour que le résultat d'une opération sur n bits soit correct dans la représentation du complément à 2, il faut que les retenues de rang n et de rang $n+1$ soient identiques.

Reprenons maintenant les mêmes exemples mais en binaire naturel :

Dans les 1^{er} et 2^e exemples, le résultat en binaire naturel sur 8 bits est correct, la retenue est égale à 0. Dans le 3^e exemple, on ajoute 4 et 126 en binaire naturel, on obtient 2 sur 8 bits (0000 0010). Le résultat est incorrect sur 8 bits. Cela est dû au dépassement de capacité puisque 130 n'est pas représentable sur 8 bits. Cependant, si maintenant on prend en compte la retenue, on obtient bien 130 (1 0000 0010), mais cette fois-ci sur 9 bits.

Soustraction

La soustraction, en arithmétique binaire, est appliquée sur des nombres signés. Dans ce cas, cette opération se ramène dans tous les cas à une addition. Ainsi, si l'on souhaite réaliser l'opération $A - B$, on effectue tout d'abord le complément à 2 de B , $CA2(B) = -B$ puis on réalise l'addition $A + (-B)$.

Pour plus d'explications sur le complément à 2, consulter la PC3.

Illustration

La figure 2.1 illustre le principe du débordement (OVR) lors d'opérations arithmétiques en CA2.

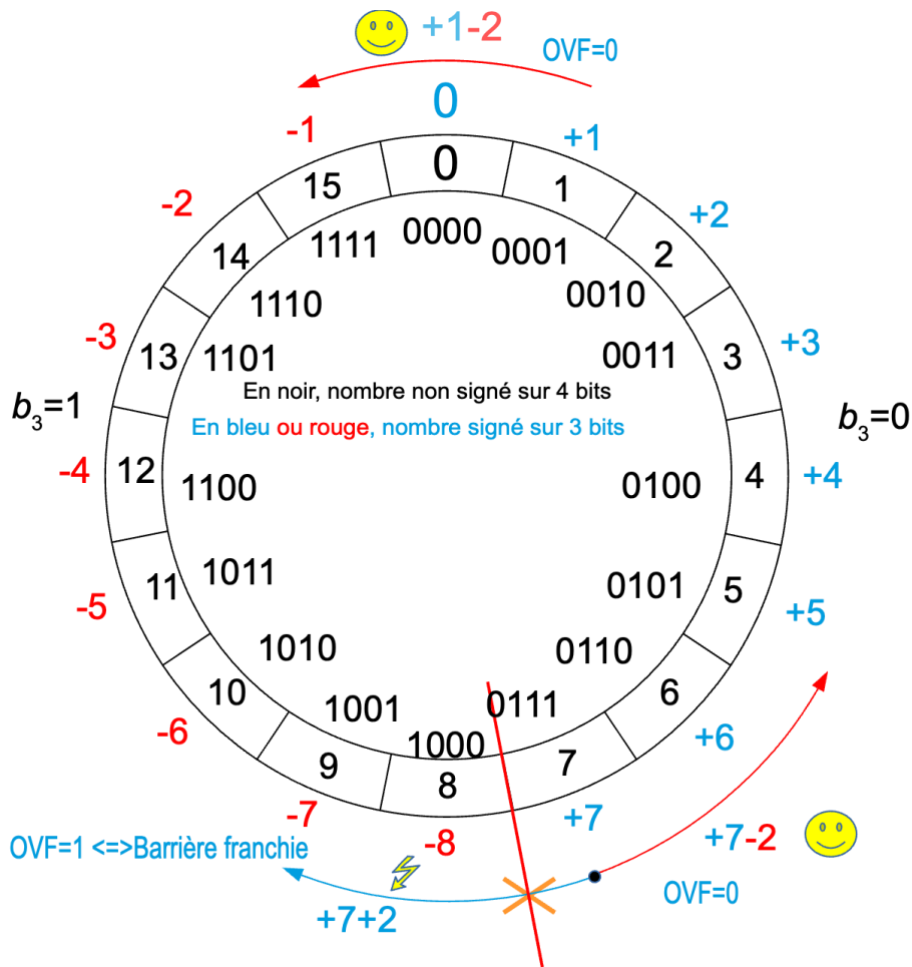


figure 2.1 : illustration de l'addition et soustraction en CA2 sur 4 bits. Principe de débordement (OVR).

Extension du bit de signe en CA2

Si on augmente le format d'un nombre positif en rajoutant un ou plusieurs "0" à gauche du nombre, on ne change pas la valeur du nombre :

$$0111 = 00111 = 000000111 = +7$$

Même chose pour les nombres négatifs où on ne change pas la valeur du nombre en ajoutant un ou plusieurs 1.

Donc, si on veut additionner deux nombres codés sur 4 bits (compris entre -8 et +7), on peut coder ces nombres sur 5 bits par une extension du bit de signe et on est sûr que le résultat sera correct sur 5 bits.

2.1.2 Multiplication et division

Les opérations de multiplication et de division sont plus complexes que l'addition, et ne sont traitées que partiellement dans ce chapitre.

Multiplication et division par 2^k

Le nombre 2 occupant une place privilégiée en numération binaire (tout comme le nombre 10 en numération décimale), les cas de la multiplication et de la division par 2^k peuvent être traités à part.

- Multiplication par 2^k

Multiplier un nombre binaire par 2^k consiste à décaler la virgule de k positions vers la droite, ou à ajouter k "0" au niveau des bits de poids faible dans le cas des entiers. Par exemple, soit $N = 18_{(10)} = 10010_{(2)}$. La multiplication de N par 2 donne $N \cdot 2_{(10)} = 36_{(10)} = 100100_{(2)}$, et sa multiplication par 4, $N \cdot 4_{(10)} = 72_{(10)} = 1001000_{(2)}$. Le mécanisme est le même que celui appliqué lors de la multiplication d'un nombre décimal par 10^k .

- Division par 2^k

Le mécanisme est inverse de celui de la multiplication. Diviser un nombre binaire par 2^k consiste à décaler la virgule de k positions vers la gauche, ou à enlever les k bits de poids faible dans le cas d'une division entière. Par exemple, soit $N = 37_{(10)} = 100101_{(2)}$. La division entière de N par 4 donne $N / 4_{(10)} = 9_{(10)} = 1001_{(2)}$, tandis que la même division considérée sur des réels donne $N / 4_{(10)} = 9,25_{(10)} = 1001,01_{(2)}$.

2.2 LES OPERATEURS ARITHMETIQUES

2.2.1 Les additionneurs

Dans la section 2.1, nous avons détaillé l'algorithme, très simple, qui permet d'additionner deux nombres binaires. Dans cette section, nous présentons les circuits combinatoires, appelés additionneurs, qui mettent en œuvre matériellement la fonction d'addition. L'étude détaillée de tous les algorithmes de calcul arithmétique sort du cadre de ce cours.

La structure des additionneurs a été intensément étudiée depuis les premières recherches sur l'architecture des calculateurs. En effet, la conception d'additionneurs rapides est importante pour effectuer des additions, et donc des multiplications et des divisions, de la manière la plus efficace possible.

Ainsi, après avoir posé le principe de base de l'addition de 2 chiffres binaires dans la section 2.1, deux architectures d'additionneurs de nombres binaires sont présentées : l'additionneur à **retenue propagée** et l'additionneur à **retenue anticipée**. Une architecture correspond à une instanciation matérielle d'un algorithme, c'est-à-dire à un agencement spécifique de ressources matérielles, typiquement présenté sous la forme d'un logigramme dans le cas d'un circuit combinatoire. Les deux architectures présentées dans cette section correspondent à deux compromis vitesse/encombrement différents.

Addition de deux bits

Le **demi-additionneur binaire** prend en entrée 2 éléments binaires A_k et B_k , et délivre en sortie leur somme S_k et une retenue C_k suivant la table de vérité du tableau 2.1.

A_k	B_k	S_k	C_k
-------	-------	-------	-------

0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

tableau 2.1 : table de vérité du demi-additionneur binaire

Le comportement du circuit est régi par les équations $S_k = A_k \oplus B_k$ et $C_k = A_k . B_k$. L'opérateur d'addition arithmétique binaire est donc le OU exclusif.

Le schéma de la figure 2.2 représente la structure du demi-additionneur binaire.

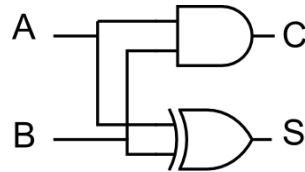


figure 2.2 : structure du demi-additionneur binaire

Pour additionner deux nombres binaires à plusieurs bits, une technique consiste à additionner successivement les bits de même poids avec la retenue de l'addition précédente. L'opérateur de base, appelé **additionneur complet**, doit alors prendre en compte une retenue entrante C_{k-1} .

A_k	B_k	retenue entrante C_{k-1}	somme S_k	retenue sortante C_k
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

tableau 2.2 : table de vérité de l'additionneur complet

Les équations logiques correspondantes peuvent s'écrire :

$$S_k = A_k \oplus B_k \oplus C_{k-1}$$

$$C_k = A_k B_k + (A_k + B_k) C_{k-1} = A_k B_k + (A_k \oplus B_k) C_{k-1}$$

L'additionneur complet peut, par exemple, être réalisé à partir de deux demi-additionneurs et d'un opérateur OU (figure 2.3).

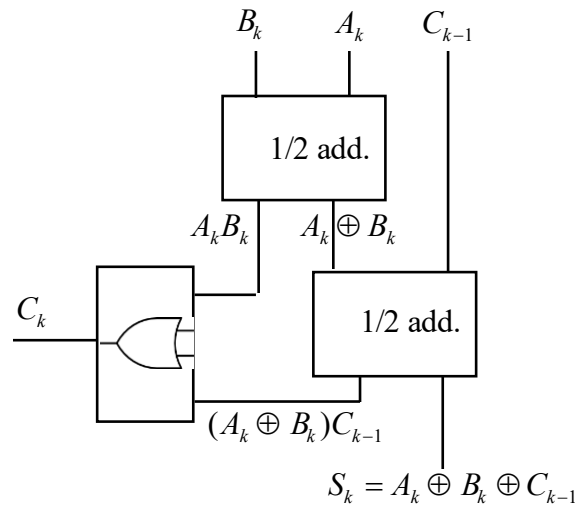


figure 2.3 : une réalisation de l'additionneur complet

Addition de deux nombres binaires

Additionneur à retenue propagée (ripple carry adder)

La solution la plus naturelle pour additionner deux nombres binaires A et B de n bits s'écrivant en numération de position $A_{n-1}A_{n-2}\dots A_k\dots A_1A_0$ et $B_{n-1}B_{n-2}\dots B_k\dots B_1B_0$ consiste à associer en cascade n étages d'additionneurs complets. Le résultat de l'addition est constitué de $n+1$ bits, soit n bits de somme $S_{n-1}S_{n-2}\dots S_k\dots S_1S_0$ et la retenue finale C_{n-1} . Chaque additionneur complet de rang k calcule la somme S_k et la retenue C_k nécessaire à l'étage suivant. La figure 2.4 présente la structure d'un additionneur de deux mots de n bits.

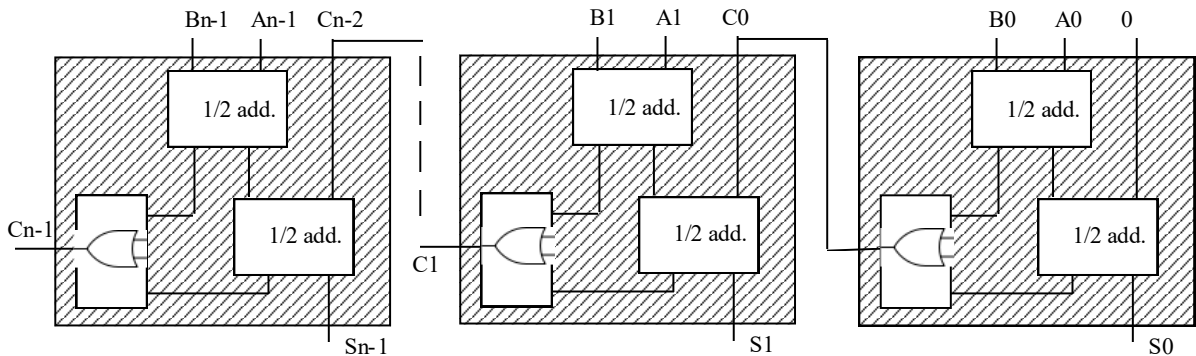


figure 2.4 : additionneur à retenue propagée ou série

Cette solution est intéressante d'un point de vue matériel car répétitive. Cependant, le résultat de l'opération n'est disponible en sortie de l'additionneur que lorsque toutes les retenues, du poids le plus faible au poids le plus fort, ont été calculées (d'où le nom d'additionneur à **retenue propagée**). Le temps de calcul du résultat est donc directement proportionnel à la taille des nombres manipulés et donc au nombre d'étages du dispositif, soit n . La technique de calcul anticipé de la retenue permet de remédier à ce défaut.

Additionneur à retenue anticipée (carry look-ahead adder)

Voir la section 4.

2.2.2 Les soustracteurs

Pour la soustraction $A - B$, on peut adopter la même approche que pour l'addition. On commence par définir l'opérateur binaire de base et on l'utilise pour réaliser des soustractions de nombres binaires. En pratique, se pose le problème de la représentation des nombres signés dans le cas où $B > A$. Pour résoudre ce problème, on convient d'une représentation des nombres négatifs adaptée et la soustraction est alors ramenée à une addition. La représentation généralement utilisée est celle du complément vrai ou complément à 2.

2.2.3 Les multiplieurs et diviseurs

Voir la section 4.

2.2.4 Les unités arithmétiques et logiques

Les fabricants de circuits intégrés proposent de nombreux composants (sous forme de circuits standard ou bien de macrofonctions) capables d'effectuer un ensemble d'opérations arithmétiques (addition, soustraction, ...) et logiques (ET, OU, OU exclusif, ...). L'opération à effectuer est déterminée par des entrées de commande ou de sélection. Ces opérateurs, appelés UAL (Unité Arithmétique et Logique) ou ALU (Arithmetic and Logic Unit), sont présents dans de nombreux dispositifs de calcul comme les microprocesseurs. Un exemple du marché est donné à la section 4.1.

2.3 BIBLIOGRAPHIE

- [BH90] J.-M. Bernard et J. Hugon, *Pratique des circuits logiques*, Collection technique et scientifique des télécommunications, Eyrolles, 1990.
- [Mot] <http://www.motorola.com/>
- [Mul89] J.-M. Muller, *Arithmétique des ordinateurs, opérateurs et fonctions élémentaires*, Etudes et recherches en informatique, Masson, 1989.
- [NS] <http://www.national.com/>
- [TI] <http://www.ti.com/>
- [Aum96] M. Aumiaux, *Logique arithmétique et techniques synchrones*, 1^{ère} partie : *Arithmétique binaire et décimale*, Enseignement de l'électronique, Masson, 1996.

3. LES EXERCICES D'APPLICATION

Dans cette séance, nous partons de la fonctionnalité que l'on souhaite obtenir, l'addition, puis à partir de l'algorithme de calcul dit « scolaire » (addition posée en colonne), nous en déduisons une architecture matérielle. Cette architecture matérielle pourra être ensuite mise en œuvre dans un circuit programmable type FPGA (*Field Programmable Gate Array* ; voir module SIT212) ou bien dans un circuit intégré, de type ASIC (*Application Specific Integrated Circuit*) par exemple.

A une fonction donnée peut correspondre plusieurs algorithmes. Et à chaque algorithme peut correspondre plusieurs architectures, et donc plusieurs circuits.

3.1 OPERATION ARITHMETIQUE : L'ADDITION, ALGORITHME ET CIRCUIT

3.1.1 Algorithme « scolaire »

Le bit de retenue *Carry* est généralement utilisé lorsque les opérandes sont **en binaire naturel**. Ce bit supplémentaire permet ainsi d'obtenir un résultat toujours correct sur $n+1$, si n est la taille des mots binaires en entrée.

Le bit de débordement (OVF, *overflow*) est utilisé lorsque les opérandes sont **en CA2**, c'est à dire nombres signés. Il indique que le résultat de l'addition est incorrect (OVF=1).

L'Unité Arithmétique et Logique (UAL) n'a pas connaissance du codage des opérandes. Elle calcule donc les 3 sorties : résultat, carry et OVR indépendamment du codage.

Compléter le tableau suivant. Vous calculez OVF en supposant que A et B sont en CA2.

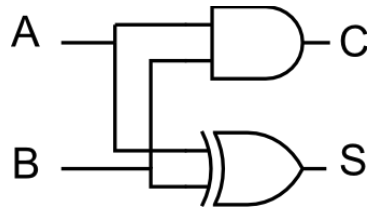
A A3 A2 A1 A0	B B3 B2 B1 B0	A+B S3 S2 S1 S0	Carry	OVF
1100	0010			
1010	1001			
0010	0111			
0011	1111			

3.1.2 Circuit Additionneur binaire complet

Le demi-additionneur binaire est un circuit qui prend en entrée 2 nombres de 1 bit (A_k et B_k) et génère leur somme (S_k) et une retenue (C_k).

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

Le schéma ci-après représente la structure d'un "demi-additionneur binaire" :



A partir des informations ci-dessus, on souhaite réaliser un additionneur complet qui prenne en compte une retenue (C_{k-1}) générée à partir d'un étage précédent.

Table de vérité

- Etablir la table de vérité donnant la somme S_k et la retenue sortante C_k en fonction de A_k , B_k et C_{k-1} .
- Donner la 1^{ère} forme canonique pour chacune de ces fonctions.

On souhaite réaliser un additionneur complet **à partir de deux demi-additionneurs et d'un opérateur combinatoire élémentaire g** . L'objectif est donc de faire apparaître les fonctions somme (AND) et retenue sortante (XOR) des $\frac{1}{2}$ additionneurs dans les fonctions de l'additionneur complet.

Obtention des fonctions logiques à partir de AND et XOR

- Pour la somme S_k , tracer le tableau de Karnaugh correspondant et simplifier la fonction.
- Pour la retenue sortante C_k , simplifier algébriquement à partir de la 1^{ère} forme canonique et à l'aide des propriétés logiques (cf. PC3), pour faire apparaître les opérateurs AND et XOR.

Schéma de l'additionneur complet

- Identifier l'opérateur combinatoire élémentaire manquant g , puis
- Donner le *schéma d'architecture* de l'additionneur complet à partir de deux demi-additionneurs et de l'opérateur combinatoire élémentaire g .

Schéma de l'additionneur de mots de n bits (bonus)

En utilisant le module de l'additionneur complet, donnez le schéma d'architecture, d'un additionneur de mots de n bits capable d'effectuer l'opération $S = A + B$, (+ représente ici l'opérateur d'addition arithmétique), avec $A = (A_{n-1}, \dots, A_k, \dots, A_1, A_0)$, $B = (B_{n-1}, \dots, B_k, \dots, B_1, B_0)$, et $S = (S_{n-1}, \dots, S_k, \dots, S_1, S_0)$. La retenue finale est désignée par C_{n-1} .

3.2 OPERATION ARITHMETIQUE : LA SOUSTRACTION, ALGORITHME ET CIRCUIT (A FAIRE A LA MAISON)

3.2.1 Algorithme « scolaire », quelques exemples

Compléter le tableau suivant (A et B sont signés, représentés en CA2) :

+A A3 A2 A1 A0	+B B3 B2 B1 B0	-B	+A+(-B) S3 S2 S1 S0	OVF
1100	0010			

1010	1001			
0010	0111			
0011	1111			

1.1.1. Circuit Additionneur/soustracteur

On souhaite transformer le montage précédent en un additionneur / soustracteur. On rappelle que dans la représentation en complément à 2, $A - B = A + (-B) = A + \bar{B} + 1$.

Proposer le schéma d'un additionneur / soustracteur capable de manipuler des nombres de 4 bits codés en complément à 2.

Cet additionneur / soustracteur possèdera une entrée de commande *SOUSTRAC* qui sera utilisée comme suit :

- *SOUSTRAC* = 0, fonctionnement en additionneur
- *SOUSTRAC* = 1, fonctionnement en soustracteur

Pour cela, vous ré-utiliserez le schéma proposé à la question 3.2.4, auquel vous ajouterez des portes logiques et vous connecterez astucieusement les entrées.

Astuces :

- Commencez par trouver la fonction qui permet d'obtenir $\bar{B}$ ou B en fonction de la valeur de *SOUSTRAC*.
- Puis connectez les entrées astucieusement pour réaliser +1 ou +0 en fonction de la valeur de *SOUSTRAC*.

3.3 DEPASSEMENT DE CAPACITE : OVERFLOW (A FAIRE A LA MAISON)

Le montage précédent ne traite pas les problèmes de débordement de capacité. Le bit de sortie *OVF* (*OVF* = overflow) sera positionné à "1" lorsqu'il y aura débordement de capacité en arithmétique signée (CA2).

- A partir de la table de vérité de l'additionneur complet donnée dans l'énoncé, donner la table de vérité de C_{n-1} , S_{n-1} et *OVF* à partir de A_{n-1} , B_{n-1} et C_{n-2} .

C_{n-2}	A_{n-1}	B_{n-1} $1 \oplus \text{SOUSTRAC}$	S_{n-1}	C_{n-1}	<i>OVF</i>

- En remarquant que *OVF* peut s'écrire uniquement en fonction des retenues C_{n-1} et C_{n-2} , donner l'équation logique du bit de sortie *OVF* en fonction des retenues C_{n-1} et C_{n-2} . Ainsi, dans la table de vérité de *OVF*, vous ne considèrerez que 2 entrées C_{n-1} et C_{n-2} .

4. POUR ALLER PLUS LOIN

Dans cette section, nous détaillons le cas de la multiplication et de son opérateur arithmétique, le multiplieur. Nous distinguons donc la fonction, de l'algorithme et de sa réalisation matérielle (architecture). Puis nous présentons un exemple d'unité arithmétique et logique sous forme de circuit discret.

4.1 CAS GENERAL DE LA MULTIPLICATION

La multiplication de deux nombres binaires de n bits fournit un résultat sur $2n$ bits. Lorsque les nombres ne sont pas signés, le principe est le même qu'en décimal et fait intervenir des produits partiels de bits, des additions et des décalages. A titre d'exemple, la multiplication de 2 nombres de 4 bits non signés, A et B , se décompose comme suit :

				A_3	A_2	A_1	A_0	Multiplicande A
			$\times$	B_3	B_2	B_1	B_0	Multiplieur B
				A_3B_0	A_2B_0	A_1B_0	A_0B_0	Produit partiel $A \times B_0$
				A_3B_1	A_2B_1	A_1B_1	A_0B_1	$A \times B_1$ décalé de 1 rang
				A_3B_2	A_2B_2	A_1B_2	A_0B_2	$A \times B_2$ décalé de 2 rangs
				A_3B_3	A_2B_3	A_1B_3	A_0B_3	$A \times B_3$ décalé de 3 rangs
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	Produit $P = A \times B$

La multiplication de A par B se déroule en trois étapes :

- Multiplication de A par chacun des symboles de B : en base 2, la multiplication de A par le symbole B_i revient à faire un ET logique entre chaque symbole de A et B_i ,
- Décalage de $A \times B_i$ de i rangs vers la gauche,
- Addition des résultats de l'étape précédente.

Lorsque les nombres à multiplier sont signés, le principe de l'opération devient plus complexe et fait appel à des algorithmes non traités dans ce cours (cf. par exemple [Aum96]). Néanmoins, dans le cas d'une représentation en complément à 2, l'algorithme de multiplication précédent est applicable, moyennant une légère modification, si le multiplieur est positif :

- Si le multiplicande est positif, pas de changement,
- Si le multiplicande est négatif, tous les produits partiels sont négatifs ou nuls. Il faut étendre les produits partiels non nuls à $2n$ bits en rajoutant des 1 sur les bits de poids forts.

Exemple :

1 0 1 1 (-5)

				×	0	0	1	1	(+3)
1	1	1	1	1	0	1	1		
1	1	1	1	0	1	1			
0	0	0	0	0	0				
0	0	0	0	0					
1	1	1	1	0	0	0	1		(-15)

Dans les systèmes numériques, la division binaire n'est pas réalisée suivant la méthode utilisée en décimal [Aum96].

4.2 ADDITIONNEUR A RETENUE ANTICIPEE

Additionneur à retenue anticipée (carry look-ahead adder)

Le principe de l'additionneur à retenue anticipée consiste à calculer toutes les retenues en parallèle.

L'équation logique de la retenue sortante de l'additionneur complet établie en section 2.2.1 peut s'écrire :

$$C_k = A_k B_k + (A_k \oplus B_k)C_{k-1} = G_k + P_k C_{k-1}$$

Le tableau 4.3 permet d'analyser l'obtention de la retenue sortante à partir des termes P_k et G_k .

n° de ligne	A_k	B_k	C_{k-1}	C_k	P_k	G_k	
1	0	0	0	0	0	0	Retenue sortante à 0
2	0	0	1	0	0	0	
3	0	1	0	0	1	0	Retenue entrante propagée vers la sortie
4	0	1	1	1	1	0	
5	1	0	0	0	1	0	
6	1	0	1	1	1	0	
7	1	1	0	1	0	1	Retenue sortante à 1
8	1	1	1	1	0	1	

tableau 4.3 : table de vérité de la retenue pour un additionneur complet

On observe que :

- Lignes 1 et 2 : la retenue sortante C_k est à 0,
- Lignes 3 à 6 : la retenue entrante C_{k-1} est propagée vers la sortie, $C_k = C_{k-1}$. Ces 4 lignes de la table de vérité sont caractérisées par la relation $P_k = A_k \oplus B_k = 1$. P_k est ici appelé terme de propagation de retenue.
- Lignes 7 à 8 : une retenue sortante à 1 est délivrée en sortie. Ces 2 lignes sont caractérisées par la relation $G_k = A_k \cdot B_k = 1$. G_k est appelé terme de génération de retenue. C'est encore l'expression de la retenue R_k en sortie d'un demi-additionneur (figure 2.2).

Dans le cas de l'addition de 2 nombres A et B de n bits, le bit de retenue de rang k est donc donné par la relation $C_k = G_k + C_{k-1}P_k$.

Le principe des additionneurs à retenue anticipée consiste à calculer :

- les couples (P_k, G_k) à l'aide de demi-additionneurs, pour $0 \leq k \leq n-1$,
- les retenues C_k , pour $0 \leq k \leq n-1$, avec $C_k = G_k + C_{k-1}P_k$, directement à partir des différents termes de propagation et de génération, et de la retenue entrante du premier étage C_{-1} : $C_0 = G_0 + C_{-1}P_0$,
 $C_1 = G_1 + C_0P_1 = G_1 + G_0P_1 + C_{-1}P_0P_1$, etc.
- les bits de somme $S_k = P_k \oplus C_{k-1}$, pour $0 \leq k \leq n-1$.

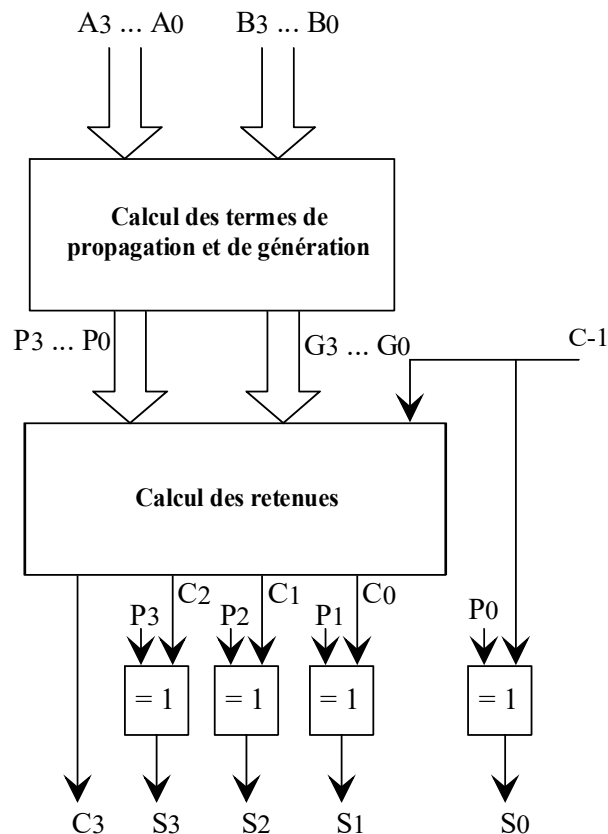


figure 4.5 : structure d'un additionneur de mots de 4 bits à retenue anticipée

Cette solution architecturale mène à des circuits plus rapides que les additionneurs à retenue propagée, car le nombre de blocs logiques traversés pour le calcul des sorties de l'additionneur est indépendant de la taille des mots en entrée. Néanmoins, ce type d'additionneur est plus encombrant que l'additionneur à retenue propagée, la logique de génération des retenues étant plus complexe. Les constructeurs proposent des circuits connus sous le nom de CLU (*Carry Look-ahead Unit*) qui assurent le calcul des termes précédents (ex : référence *xx* 182).

Les solutions étudiées ne permettent pas de traiter le cas de l'addition de deux nombres de signes différents. Ce problème est lié à celui de la soustraction qui est traitée dans le paragraphe suivant.

4.3 LES MULTIPLIEURS ET DIVISEURS (OPERATEURS ARITHMETIQUES)

Dans les années 70-80, les opérations de multiplication et de division étaient effectuées de manière algorithmique dans les processeurs. Ces opérateurs sont maintenant intégrés sous forme matérielle pour des raisons de rapidité.

Le principe de la multiplication en base 2 a été présenté à la section précédente. Il existe deux types de solutions pour la réalisation des opérateurs correspondants.

- Solution combinatoire : réseau de portes ET et d'additionneurs binaires permettant de réaliser simultanément les opérations élémentaires de la multiplication. La figure 4.6 donne une réalisation possible d'un multiplieur combinatoire de deux nombres A et B de 4 bits. Cette structure transcrit directement sous forme de circuit la réalisation « à la main » du calcul.

- Solution séquentielle : cette solution sort du cadre de ce chapitre car non combinatoire. La multiplication est réalisée en plusieurs étapes faisant intervenir des opérations successives d'addition et de décalage (cf. chapitre 5) réutilisant les mêmes opérateurs. L'implantation de cette solution est beaucoup moins encombrante que la précédente, mais présente des performances de vitesse dégradées, car elle nécessite plusieurs étapes de calcul.

La réalisation de diviseurs est en fait plus délicate que celle des multiplieurs mais les divisions sont en pratique beaucoup moins courantes dans les systèmes numériques que les additions/soustractions et les multiplications. L'approche la plus simple, qui consiste à implanter la division sous forme d'additions/soustractions et de décalages, conduit à des performances médiocres en termes de vitesse de calcul et n'est guère utilisée.

Nous ne détaillerons pas davantage les familles de circuits arithmétiques. De nombreuses architectures existent : pipe-line, parallèle, série-parallèle, cellulaires, etc. Leur étude, qui sort du cadre de ce cours, est détaillée dans [Mul89].

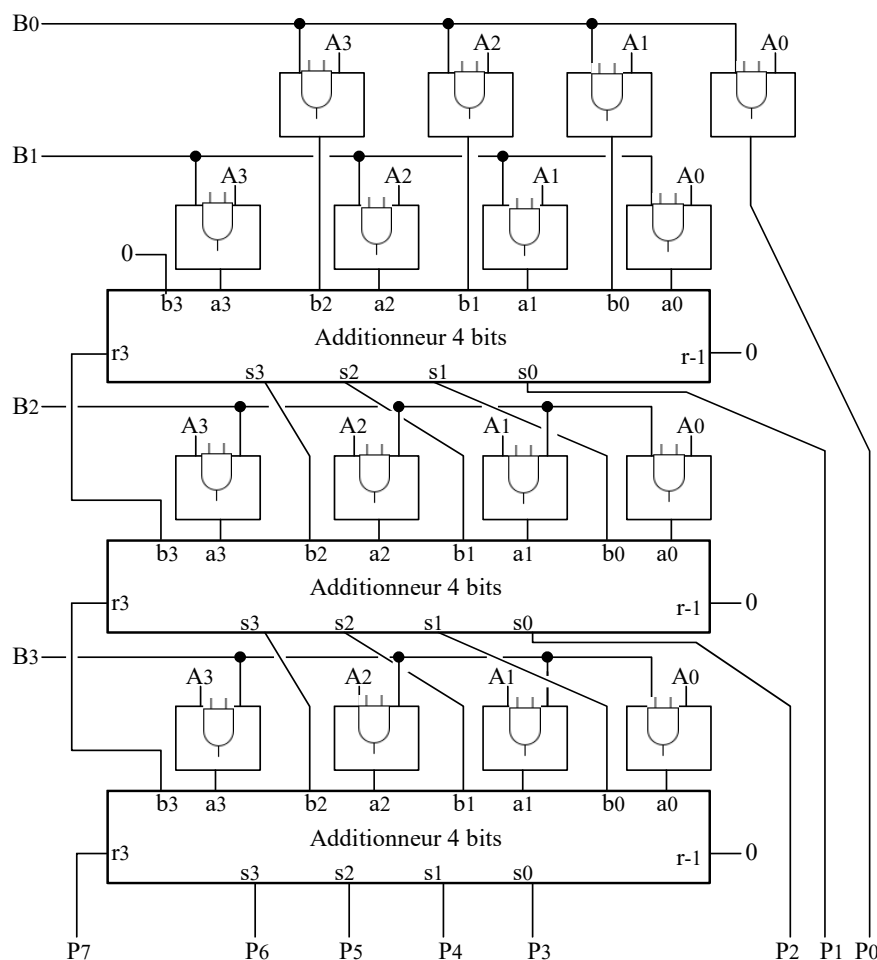


figure 4.6 : réalisation d'un multiplieur combinatoire de 2 mots de 4 bits

4.4 EXEMPLE D'UNE UNITE ARITHMETIQUE ET LOGIQUE DU MARCHE : LE CIRCUIT INTEGRE DE REFERENCE XX 382

L'UAL référencée *XX 382* est un circuit qui est constitué d'environ 80 portes logiques élémentaires. A l'aide des entrées de sélection S_0 à S_2 , l'utilisateur peut choisir une opération parmi les huit disponibles. Cette unité accepte en entrée des mots de 4 bits.

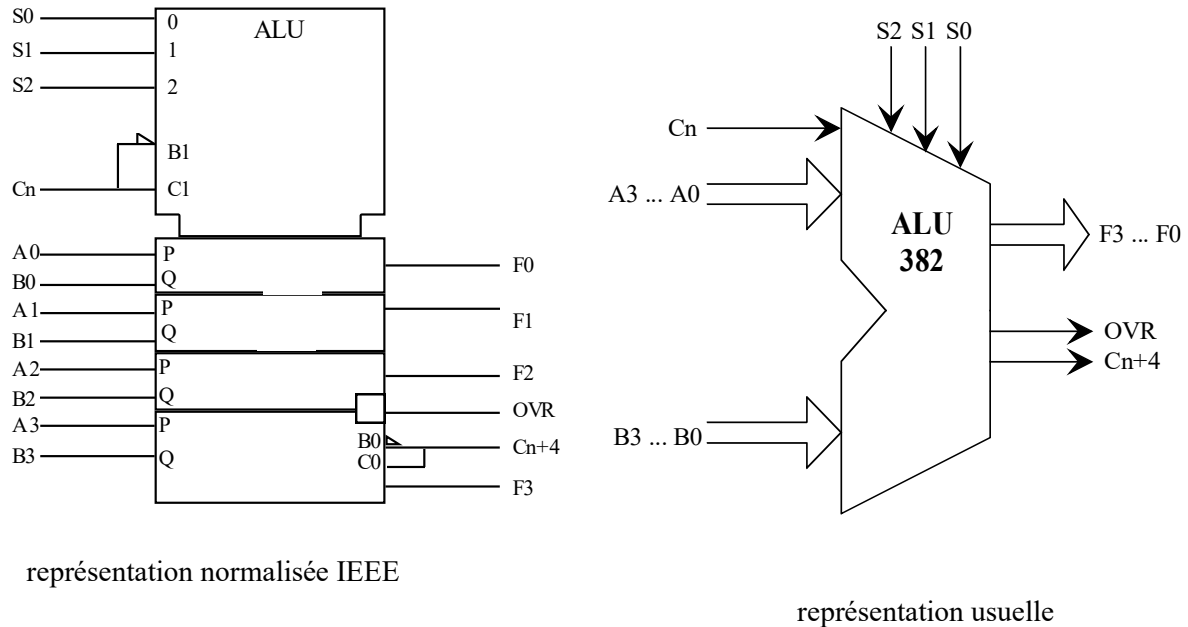


figure 4.7 : unité arithmétique et logique *XX 382*

Le tableau 4.4 présente une table de vérité partielle de l'UAL de référence *XX 382*. Seuls les cas où A et B sont égaux à 0000 ou 1111 sont traités à titre d'exemple.

Pour les modes arithmétiques, « A plus B », « A moins B », et « B moins A », les nombres A et B sont codés en complément à 2. Dans les trois cas, l'opérateur utilisé est un additionneur dont la retenue entrante est C_n .

- $S_2 S_1 S_0 = 011$, « A plus B » :

Les nombres A et B sont directement appliqués en entrée de l'additionneur. Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est en fait $F = A + B + 1$, sinon $F = A + B$ (+ désigne ici l'addition).

- $S_2 S_1 S_0 = 001$, « B moins A » :

Le complément à 1 de A , $\bar{A}$, est calculé et additionné à B . Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = B + \bar{A} + 1 = B - A$, sinon $F = B + \bar{A} = B - A - 1$ (− désigne ici la soustraction).

- $S_2 S_1 S_0 = 010$, « A moins B » :

Le complément à 1 de B , $\bar{B}$, est calculé et additionné à A . Si la retenue entrante est active ($C_n = 1$), l'opération effectuée est $F = A + \bar{B} + 1 = A - B$, sinon $F = A + \bar{B} = A - B - 1$.

Dans ces trois modes arithmétiques, la sortie C_{n+4} est activée ($C_{n+4} = 1$) lorsque l'addition génère une retenue. La sortie *OVF* indique un dépassement de capacité de l'additionneur. Cette situation n'est pas considérée dans le tableau 4.4. A titre d'exemple, dans le mode « *A plus B* », la sortie *OVF* est activée si $A = B = 0111$.

Entrées de sélection $S_2 S_1 S_0$	Opération arithmétique/logique	C_n 1	$A =$ $A_3 A_2 A_1 A_0$	$B =$ $B_3 B_2 B_1 B_0$	$F =$ $F_3 F_2 F_1 F_0$	OVF	C_{n+4}
000	Mise à 0 (<i>Clear</i>)	X	XXXX	XXXX	0000	—	—
001	<i>B moins A</i>	0	0000	0000	1111	0	0
			0000	1111	1110	0	1
			1111	0000	0000	0	0
			1111	1111	1111	0	0
		1	0000	0000	0000	0	1
			0000	1111	1111	0	1
			1111	0000	0001	0	0
			1111	1111	0000	0	1
010	<i>A moins B</i>	0	0000	0000	1111	0	0
			0000	1111	0000	0	0
			1111	0000	1110	0	1
			1111	1111	1111	0	0
		1	0000	0000	0000	0	1
			0000	1111	0001	0	0
			1111	0000	1111	0	1
			1111	1111	0000	0	1
011	<i>A plus B</i>	0	0000	0000	0000	0	0
			0000	1111	1111	0	0
			1111	0000	1111	0	0
			1111	1111	1110	0	1
		1	0000	0000	0001	0	0
			0000	1111	0000	0	1
			1111	0000	0000	0	1
			1111	1111	1111	0	1
100	$A \oplus B$	X	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 1111 0000	—	—
101	<i>A OU B</i>	X	0000 0000 1111 1111	0000 1111 0000 1111	0000 1111 1111 1111	—	—
110	<i>A ET B</i>	X	0000 0000	0000 1111	0000 0000	—	—

¹ C_n : entrée de retenue pour l'addition et entrée de retenue inversée pour la soustraction

			1111 1111	0000 1111	0000 1111		
111	Mise à 1 (<i>Preset</i>)	X	XXXX	XXXX	1111	—	—

tableau 4.4 : table de fonctionnement partielle de l'UAL *xx* 382

Cette UAL permet également d'effectuer 3 opérations logiques sur les mots *A* et *B* : OU exclusif, OU et ET. Ces trois opérations s'effectuent bit à bit. En outre, les sorties *F3F2F1F0* peuvent être forcées à 0 ou à 1. Les sorties C_{n+4} et *OVF* ne sont, *a priori*, utilisées que dans les modes arithmétiques, leurs valeurs ne sont donc pas données dans les modes logiques car sans intérêt.

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance BE2 - SYNTHÈSE D'UN CIRCUIT CMOS A PARTIR D'UN PROBLÈME DONNÉ. - LE DISTRIBUTEUR DE BOISSONS CHAUDES

Sommaire

BE2 - SYNTHÈSE D'UN CIRCUIT CMOS A PARTIR D'UN PROBLÈME DONNÉ. - LE DISTRIBUTEUR DE BOISSONS CHAUDES.....	1
1. CONTEXTE	2
2. OBJECTIF.....	2
3. OBSERVATION DES FONCTIONNALITÉS	2
4. ÉVALUATION DES SOLUTIONS	2

 IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom	ELP111
	Fonctions électroniques logiques et analogiques
	Fiche séance (classe inversée) BE2 - Synthèse d'un circuit CMOS à partir d'un problème donné. Le distributeur de boissons chaudes

1. CONTEXTE

Nous sommes une société de production de distributeur de boissons chaudes pour le marché des entreprises. Dans le contexte d'amélioration de notre gamme de distributeurs, nous souhaitons étudier les distributeurs des sociétés concurrentes (*reverse engineering*). Pour cela, nous avons à notre disposition le distributeur en question, et en fonction des actions réalisées sur la machine, nous observons son comportement.

2. OBJECTIF

L'objectif est d'obtenir le schéma électrique, à partir d'opérateurs CMOS, du circuit permettant la distribution des boissons chaudes.

Plusieurs équipes réalisent cette action de reverse engineering et le directeur de notre société sélectionnera l'équipe ayant produit le circuit avec l'encombrement et les temps de propagation les plus faibles.

3. OBSERVATION DES FONCTIONNALITES

Le distributeur de boissons chaudes permet de distribuer du café ou du thé, avec ou sans lait, ou du lait seul.

Trois boutons permettent de commander le distributeur : « café », « thé », « lait ». Pour obtenir l'une de ces boissons seule, il suffit d'appuyer sur le bouton correspondant. Pour obtenir une boisson avec lait, il faut appuyer en même temps sur le bouton correspondant à la boisson choisie et sur le bouton « lait ».

De plus, le distributeur ne fonctionne que si un jeton a préalablement été introduit dans la fente de l'appareil. La fausse manœuvre « appui simultané sur « café » et « thé » après introduction du jeton » provoque la restitution du jeton. Le lait étant gratuit, le jeton est également restitué si du lait seul est choisi.

On notera que la fonction de restitution du jeton peut indifféremment être active ou non lorsque aucun jeton n'est introduit dans l'appareil.

Enfin, si un jeton est introduit mais qu'aucun autre bouton n'est activé, le jeton est conservé par la machine (en attente d'une commande).

4. EVALUATION DES SOLUTIONS

Chaque équipe évaluera la solution d'une équipe concurrente à partir de la fiche d'évaluation donnée en annexe.

Le score obtenu permettra de sélectionner la ou les solutions gagnantes.

ANNEXE

 IMT Atlantique Bretagne-Pays de la Loire Ecole Mines-Télécom	FIP ELP111 Compétence : Mettre en œuvre la méthode de synthèse d'un circuit CMOS à partir d'un problème donné.
--	---

Evaluateur // Date / Élève ou Groupe :

Situation d'évaluation : Reverse engineering du distributeur de boissons chaudes

Critères	-	=	+
Obtenir la table de vérité répondant au problème	1 ligne erronée ou plus	Aucune ligne erronée	
Simplifier les fonctions logiques	Nombre minimal d'opérateurs et de variables non atteints (cf. correction pour la référence) et méthode utilisée incorrecte	Nombre minimal d'opérateurs et de variables partiellement atteint, au plus 1 erreur de regroupement, méthode correcte sinon	Nombre minimal d'opérateurs et de variables atteints grâce à une méthode correcte et sur une table de vérité correcte
Transformer les fonctions simplifiées pour obtenir un nombre minimal de transistors	Méthode utilisée incorrecte, solution incorrecte	Méthode utilisée correcte mais sur des fonctions simplifiées incorrectes	Méthode utilisée correcte sur des fonctions simplifiées correctes
Donner le circuit électrique CMOS de réponse au problème	Erreur(s) dans la traduction des fonctions transformées en circuit CMOS	Circuit correct sur des fonctions transformées incorrectes. Nb minimal de transistors non atteint (cf. correction pour référence)	Circuit correct sur des fonctions transformées correctes. Nb minimal de transistors atteint. (cf. correction pour référence)
Calculer les temps de propagation	Erreur(s) dans le calcul des temps de propagation sur un schéma électrique incorrect.	Méthode de calcul correcte sur un circuit incorrect.	Méthode de calcul correcte sur un circuit correct.
La compétence est validée s'il y a 4 critères au niveau '=' ou '+'. Score = nombre de '+'.			

Observations :

ANNEXE