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

Électronique Numérique

Cours
Premier semestre

C. Belleudy, D.Gaffé

version 3.11

Chapitre 1

Introduction aux systèmes numériques

Plan

1 - Introduction

2 - Représentation des nombres dans les systèmes numériques

3 - Les bases de la conception d'un système numérique

4 - Quelques notions d'optimisation lors de la conception

2

Introduction (1): Historique de l'électronique

1820 : Découverte du courant électrique,

1870 : Invention du téléphone sur fil

1900 : Transmission de signaux par voie hertzienne :

Télégraphe sans fil, radar

1920 : Premier ordinateur utilisant des relais

1946 : Apparition du premier ordinateur à tubes à vides (ENIAC)

1950 : Construction des premiers transistors

1960 : Construction des premiers circuits intégrés

(quelques centaines de transistors sur un centimétre carré : SSI)

3

Introduction (2) : Histoire de l'électronique

Intégration plus importante du nombre de transistor sur la même surface :

1970: plusieurs dizaines de milliers de transistors (MSI, 10 micron).

1980: plusieurs centaines de milliers de transitors (LSI, 1 micron).

1990: plusieurs millions de transistors (VLSI, 0,35 micron).

2000: plusieurs dizaines de millions de transistors (VLSI, 0,12 micron).
(ex: Pentium 4 : 55 millions de transistors)

Quelques chiffres :

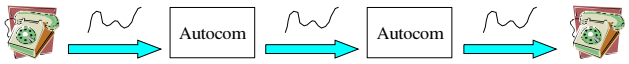
fabrication de :
- 15 milliards de circuits intégrés par an,
- 2 millions de transistors par seconde.

⇨ **Enjeu économique et militaire.**

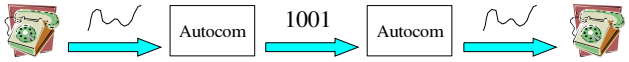
Vers le « tout numérique »

Un exemple : évolution technologique du téléphone

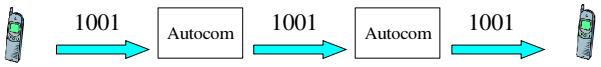
1900 : transmission analogique par fils, commutateurs électromécaniques.



1970 : numérisation du réseau entre centraux, la transmission reste analogique entre le poste de l'abonné et le central local.

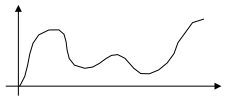


1995 : portables,GSM : transmission numérique de l'appelant à l'appelé.



Pourquoi cette évolution ?

Transmission analogique

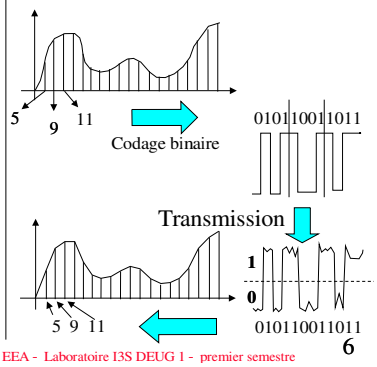


Ensemble continu de valeurs

Transmission

Trop sensible aux fluctuations et aux parasites.

Transmission numérique



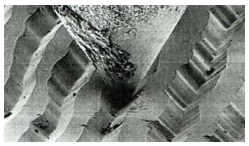
Pourquoi cette évolution (2) ?

Intérêt du codage binaire

- Puissants outils mathématiques (**algèbre de Boole**)
  - simplification de systèmes
  - équivalence de systèmes
  - correction d'un signal erroné
- Beaucoup de systèmes ont **deux** états physiques
  - {présence, absence} de courant
  - système {ouvert,fermé}
  - surface avec {creux,bosses}
  - aimantation {nord,sud}

Applications Audio

Enregistrement par gravure :



Disques 78, 45, 33 Tr (vinyle)

Enregistrement Analogique : microsillon dont l'épaisseur varie avec l'intensité du signal

- sensibles aux poussières et aux rayures

Compact disque

Enregistrement Numérique : alternance de creux et de plats

Le codage binaire : 1 - Représentation des entiers naturels

Binaire naturel :

Décimal -> Binaire naturel : méthode des divisions successives

20	2	
0	10	2
	0	5	2
		1	2	2
			0	1	2
				1	0

Sens de recopie

$(20)_{10} = (10100)_2$

Binaire naturel -> Décimal : multiplication par les puissances de deux croissantes

$(10100) = 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 0.2^0$

Le codage binaire : 1 - Représentation des entiers naturels

Décimal codé binaire naturel (DCBN) :

Chaque chiffre décimal est codé séparément en binaire :

Base 10	Base 2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

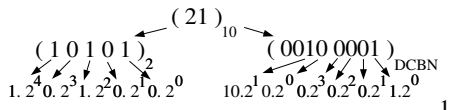
4 bits !

Exemple : $(124)_{10} = (\ 0001\ 0010\ 0100\ )_{DCBN}$

Avantage : facilité de conversion

Inconvénients :

- Toutes les combinaisons ne sont pas utilisées,
- **Code pondéré de façon non linéaire.**



Le codage binaire : 2 - Représentation des entiers relatifs

Le binaire signé :

Le bit de poids le plus fort représente le signe du nombre : **0** → **+**, **1** → **–**

Signe	valeur absolue
1 bit	**n-1 bits**

3	11				11
+ -3	011		**Complément à deux**		011
0	+111				+101
	010	???			**000**

Le codage binaire : 2 - Représentation des entiers relatifs

Décimal -> binaire signé :

1 - Codage du signe

$(-3)_{10}$ => bit de poids le plus fort égal à 1

2 - Codage en binaire naturel de la valeur absolue, substitution des 0 par des 1 et des 1 par des 0, ajout de 1

Ainsi $(3)_{10} = (11)_2$, 11=>00, 00+1= 01,
D'ou $(-3)_{10} = (1\ 01)_{BS}$

Binaire signé -> Décimal :

1 - Détermination du signe : (1 0 1) => nombre négatif

2 - Isoler la valeur absolue, substitution des 0 par des 1 et des 1 par des 0, ajout de 1. **01 =>10, ajout de 1 => 11, soit –3**

Démonstration de la règle du complément à deux

Remarque préliminaire 1 :

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

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

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

Remarque préliminaire 2 :

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

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

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

En particulier en binaire, nous pouvons écrire :

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

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

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

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

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

Pour X donné, tout Y qui vérifie cette équation sera solution. Intéressons-nous en particulier au Y qui vérifie :

$$X + Y = 2^n$$

Ce qui s'écrit également :

$$Y = 2^n - X$$

Par la remarque 2, nous pouvons donc affirmer que

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

et par la remarque 1 que

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

d'où

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

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

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

Finalement

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

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

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

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

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

– Nous pourrons gérer le bit de signe à part (ou non ...) lors de l'opération de complément à deux

– Les nombres que l'on qualifiera de "négatifs" correspondront à une translation de 2^{n-1} de nombres positifs privés de leur bit de poid fort ...

Le codage binaire : 3 - Notion de format

Notion de format :

un système numérique se trouve limité par le format des données qu'il est capable de traiter.

Etendue de l'échelle de numération (n bits) :

- Binaire naturel : $[0,2^n-1]$
- DCBN (n doit être un multiple de 4) : $[0,10^{n/4}-1]$
- Binaire signé ; $[-2^{n-1},+2^{n-1}-1]$

Exemple : pour les microprocesseurs 32 bits actuels

- Binaire naturel : [0,4 294 967 295]
- DCBN : [0,99 999 999],
- Binaire signé : [-2 147 483 648, +2 147 483 647].

15

Binaire signé : nombres caractéristiques

	n=2	n=8	n quelconque
0	00	00000000	0 ... 0 (n fois)
1	01	00000001	0 ... 01 (n-1 fois)
-1	11	11111111	1 ... 1 (n fois)
$2^{n-1}-1$	01	01111111	01 ... 1 (n-1 fois)
-2^{n-1}	10	10000000	10 ... 0 (n-1 fois)

Les bases de la conception numérique

Caméra CCD Audine

Comment concevoir un système ?

Comment modéliser le système ?

Comment réaliser le système ?

Exemple : Code de parité

1001 0111 0001 0111 ???

Solution : ajout d'un bit de parité

Principe : ajouter ''1" si le nombre de bits est impair sinon ajouter ''0"

➡ Nombre de bits à ''1" transmis toujours pair

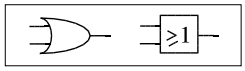
1 1001 0111 ➡ **1 0001 0111**

➡ Demande de retransmission

Les circuits logiques de base

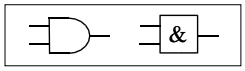
L'opérateur OU:

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



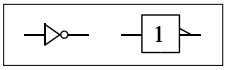
L'opérateur ET:

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



L'opérateur NON:

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



Modélisation d'un système numérique

Table de vérité :

Pour chaque combinaison des entrées, indiquer la valeur de la sortie

E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	0

Equation logique :

La sortie vaut 1 lorsque E1=0 et E0=1 ou lorsque E1=1 et E0=0

$$S = \overline{E1} \cdot E0 + E1 \cdot \overline{E0}$$

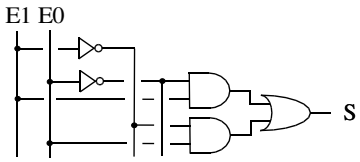
Modélisation d'un système numérique

Spécification : ajouter ''1'' si le nombre de bits est impair

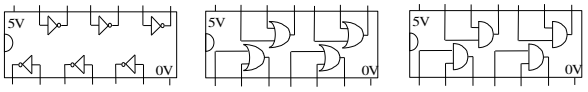
Table de vérité

Equation : $S = \overline{E1} \cdot E0 + E1 \cdot \overline{E0}$

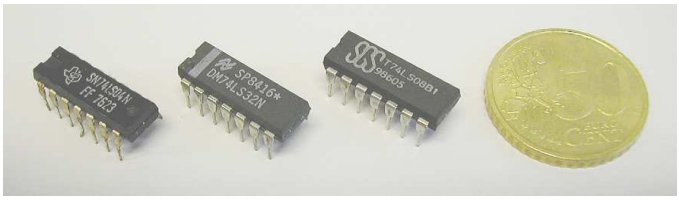
Logigramme :



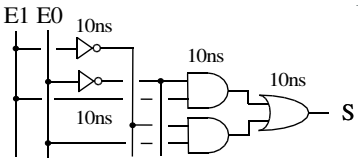
Exemple de circuits disponibles



7404 : 6 x NON **7432 : 4 x OU** **7408 : 4 x ET**



Trois critères : Coût, Vitesse, Encombrement



Vitesse: 1 / 30 ns = 33,3 Mhz

Coût: 3 cicuits à 0.5 euro (sous-utilisation)

Encombrement : 3 circuits de 2 par 0.5 cm, les connections représentent, en moyenne 5% à 10% de la taille des circuits

Comment réduire les trois critères ?

Les différentes étapes de la conception :

Modélisation par table de vérité

Pas d'optimisation possible

Modélisation par équation logique

Utilisation des propriétés de l'algèbre de boole pour réduire l'expression de l'équation logique

Choix des composants

Optimisation du choix des composants.

24

Chapitre 2

Simplification des systèmes numériques par algèbre de Boole

2.1 Notations introductives

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

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

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

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

2.2 Produits fondamentaux et forme canonique

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

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

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

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

Remarque :

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

Exemple : Considérons trois variables booléennes E_1, E_2, E_3

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

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

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

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

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

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

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

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

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

Propriété fondamentale :

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

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

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

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

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

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

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

2.3 Opérateur logique de base et algèbre de Boole

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

ET

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

OU

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

NON

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Par exemple $T1$ peut se démontrer ainsi :

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

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

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

Remarque 1 :

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

Remarque 2 :

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

2.4 Simplification algébrique

2.4.1 Définition et propriétés complémentaires

définition

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

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

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

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

Propriété fondamentale 2 :

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

Sur notre exemple,

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

2.4.2 Règles de simplification

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

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

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

peut se simplifier comme :

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

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

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

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

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

Propriété d'absoption généralisée 1 :

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

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

Propriété d'absorption généralisée 2 :

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

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

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

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

Propriété d'absoption généralisée 3 :

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

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

Propriété d'absorption généralisée 4 :

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

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

Remarque :

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

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

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

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

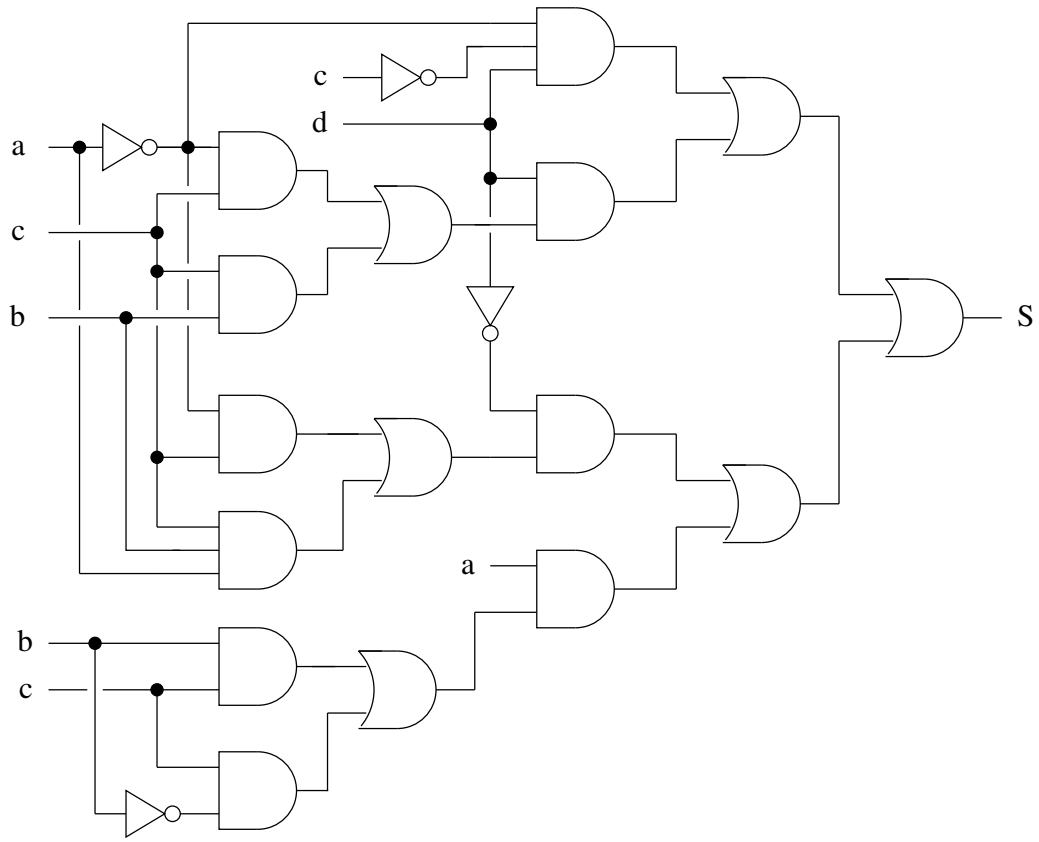
$\mathsf{C}_{x.y} = \{x.y.z, x.y.\overline{z}\}, \mathsf{C}_{y.z} = \{x.y.z, \overline{x}.y.z\}, \mathsf{C}_{\overline{x}.z} = \{\overline{x}.y.z, \overline{x}.\overline{y}.z\}$

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

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

2.5 Exemple complet

Considérons le schéma électronique suivant :



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

ainsi :

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

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

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

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

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

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

en effet :

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

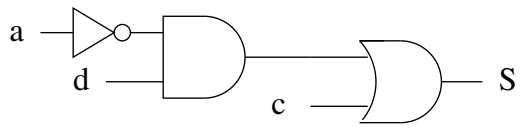
or $\mathbb{C}_{b.c.d} \subset (\mathbb{C}_{\overline{a}.c.d} \bigcup \mathbb{C}_{a.b.c})$ d'ou $S = \overline{a}.\overline{c}.d + \overline{a}.c.d + \overline{a}.c.\overline{d} + a.b.c + a.\overline{b}.c$

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

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

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

ce qui donne le schéma final suivant :



2.6 Fonctions booléennes non-simplifiables

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

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

Pour trois variables, nous obtenons :

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

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

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

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

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

x et y sont égaux :

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

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

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

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

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

Pour le cas à trois variables, nous trouvons :

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

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

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

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

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

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

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

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

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

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

Chapitre 3

Simplification par table de Karnaugh

3.1 Introduction

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

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

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

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

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

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

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

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

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

3.2 Table de simplification : première tentative

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

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

FIG. 3.1 – Première tentative

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

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

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

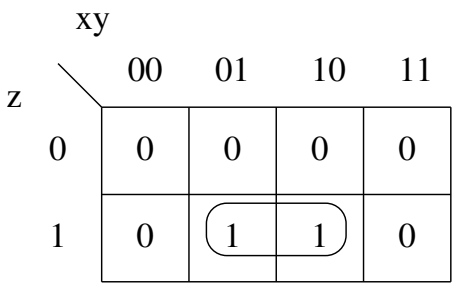


FIG. 3.2 – Mauvais regroupement

3.3 Table de simplification : seconde tentative

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

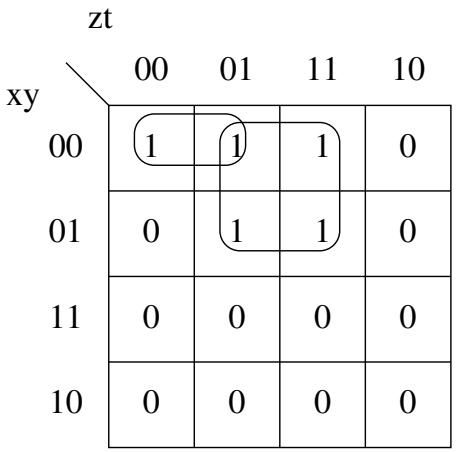


FIG. 3.3 – table de KARNAUGH associée à la table de vérité

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

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

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

3.3.1 Code de Gray cyclique

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

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

x	y
0	0
0	1
1	1
1	0

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

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

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

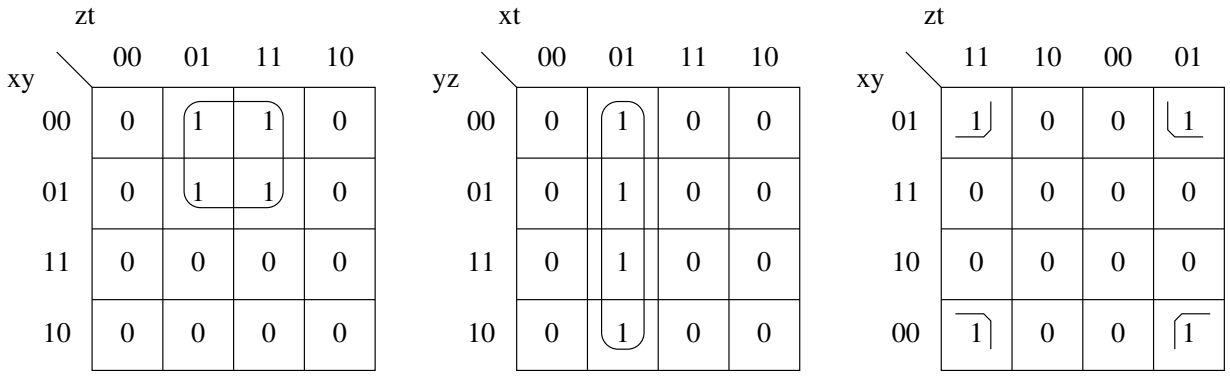
x	y	z
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Nota : Le code G_2 obéit déjà à cette règle !

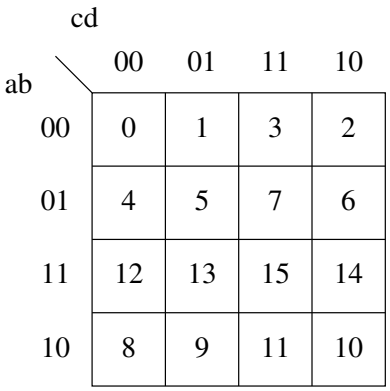
Cette propriété de construction spécifique du code de Gray lui a valu une autre dénomination : *code binaire réfléchi*. Nous parlerons donc dans la suite de ce cours, indifféremment de *code de Gray* ou de *code réfléchi*.

3.3.2 Propriétés structurelles des tables de Karnaugh

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

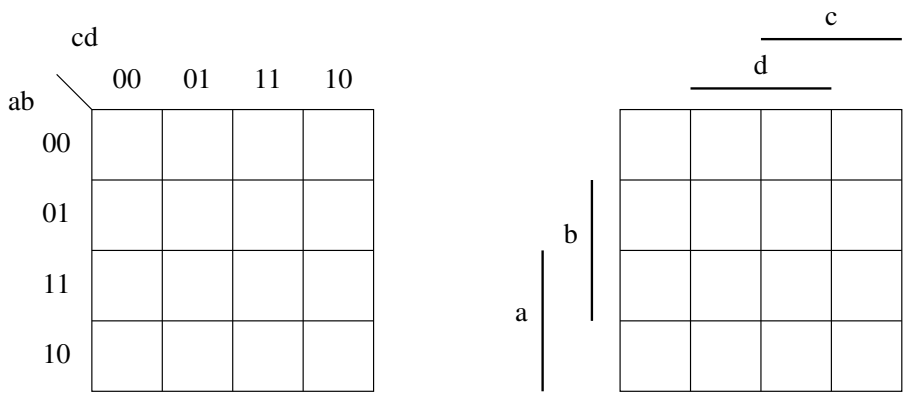


En pratique, le nombre important de karnaugh à contruire à partir de tables de vérité conduit souvent à conserver l'ordre des variables a, b, c, d des tables de vérité. Ceci donne systématiquement le codage suivant (ici pour 4 variables) :



3.3.3 Notation

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



(a) **(b)**

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

3.4 Expression graphique des règles de simplification

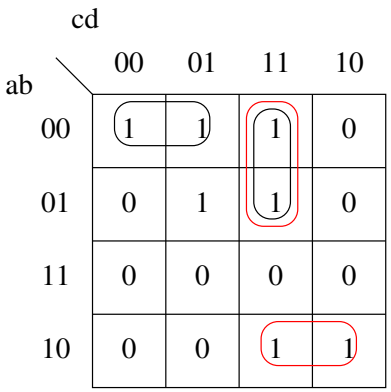
Nous allons voir dans ce chapitre que les tables de Karnaugh donnent une représentation graphique des règles algébriques.

3.4.1 Idempotence

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

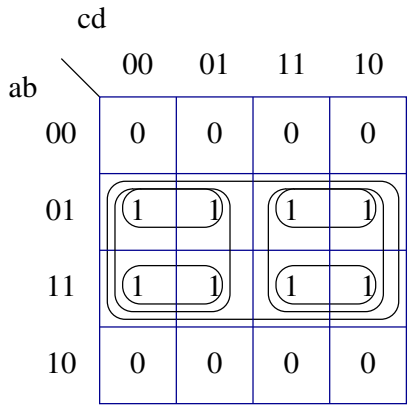
Ainsi $\overline{a}.\overline{b}.\overline{c} + \overline{a}.c.d + a.\overline{b}.c + \overline{a}.c.d = \overline{a}.\overline{b}.\overline{c} + \overline{a}.c.d + a.\overline{b}.c$

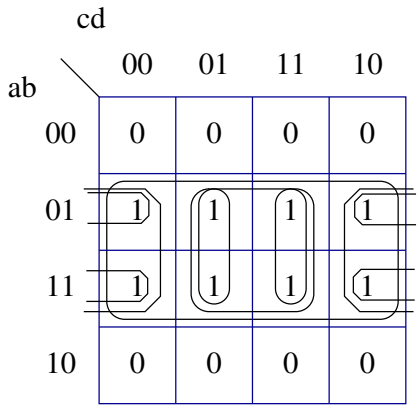
c'est à dire en Karnaugh :



3.4.2 Regroupement de produits

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





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

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

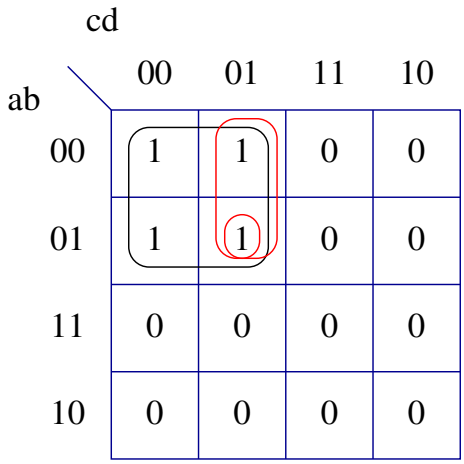
Comme les produits de même taille se regroupent toujours deux par deux, le regroupement global concernera 2^n produits fondamentaux.

3.4.3 Absorption de produits, absorption généralisée

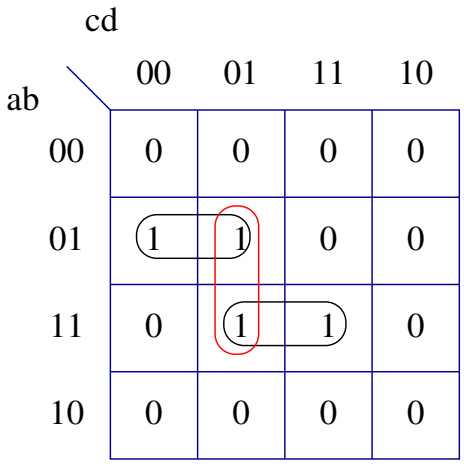
Les deux règles d'absoption peuvent se représenter sous forme de Karnaugh (ici nous avons pris un exemple moins simple que la rêgle de base pour montrer le phénomène récursif d'absoption dans Karnaugh).

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

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



absorption récursive



absorption généralisée

3.4.4 Impliquant premier, impliquant essentiel

Rappel du cours précédent :

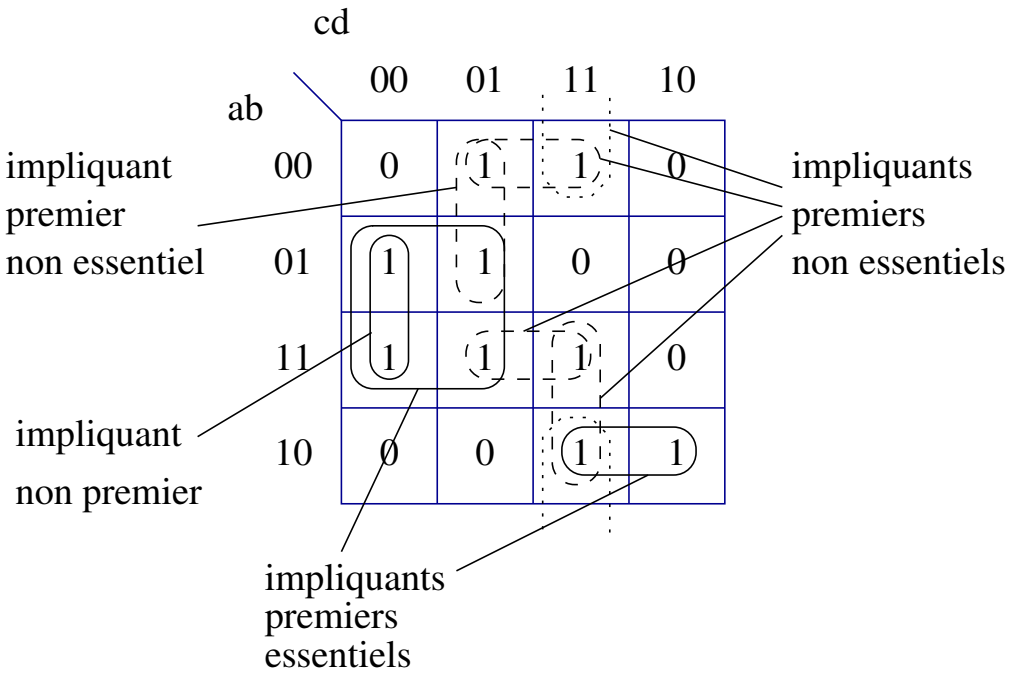
- On appelle impliquant de f, tout produit dont l'évaluation à 1 implique l'évaluation à 1 de f.
- On appelle impliquant premier de f, tout impliquant de f qui ne peux pas se simplifier avec un autre produit.

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

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

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

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



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

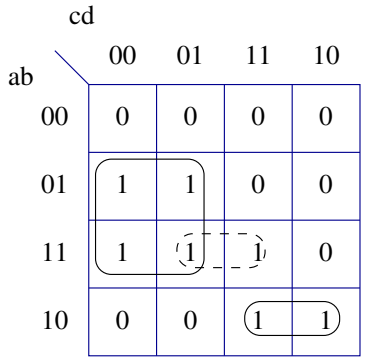
En conclusion, simplifier une fonction par Karnaugh, consiste à *couvrir* (au sens C_f) le plus de "1" par des impliquants premiers essentiels, puis à compléter par certains impliquants premiers non essentiels.

3.4.5 Formes équivalentes simplifiées d'une fonction

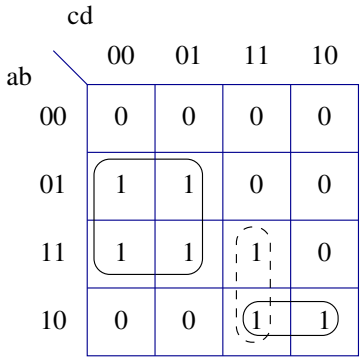
Les fonctions f et g sont simplifiées et équivalentes car elles ont la même couverture. Ceci se matérialise sur le Karnaugh par des regroupements équivalents. La différence des expressions vient du fait que les produits $a.b.d$, $a.c.d$ sont non essentiels et qu'un des deux est quand même indispensable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

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

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



f

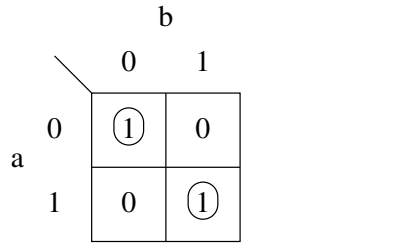


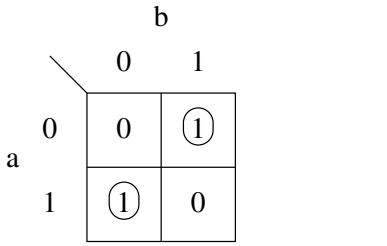
g

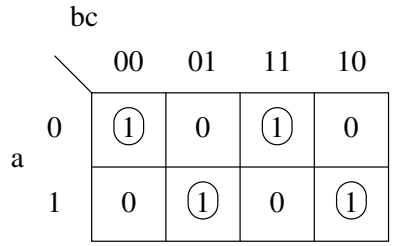
3.4.6 Formes caractéristiques

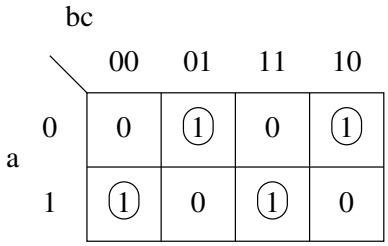
Nous avons vu au cours précédent qu'il existe deux fonctions particulières $F(x_1, ...x_n)$, $\overline{F(x_1, ..., x_n)}$ qui font appel à 2^{n-1} mintermes sur les 2^n possibles et qui sont non-simplifiables.

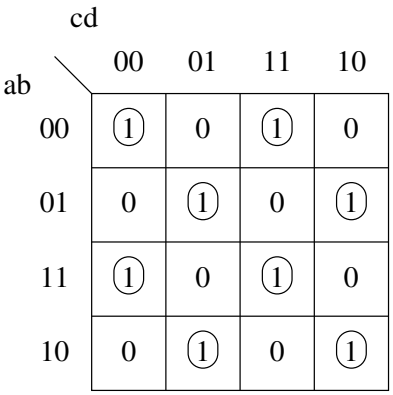
Ces deux fonctions prennent une forme très particulière avec Karnaugh : On les appelle "damiers".



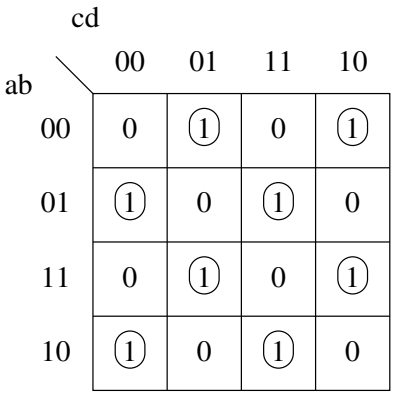








$\overline{a \oplus b \oplus c \oplus d}$

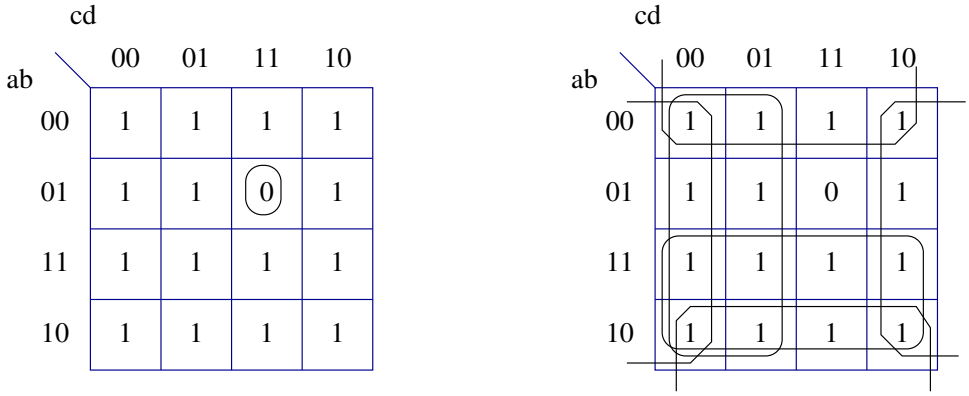


$a \oplus b \oplus c \oplus d$

3.4.7 Karnaugh à “0” minoritaires

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

L’exemple suivant montre une telle configuration :



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

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

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

3.5 Simplification des fonctions non complètement spécifiées

3.5.1 Définition d’une FNCS, notion d’indifférent

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

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

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

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

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

3.5.2 Simplification par algèbre de Boole

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

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

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

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

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

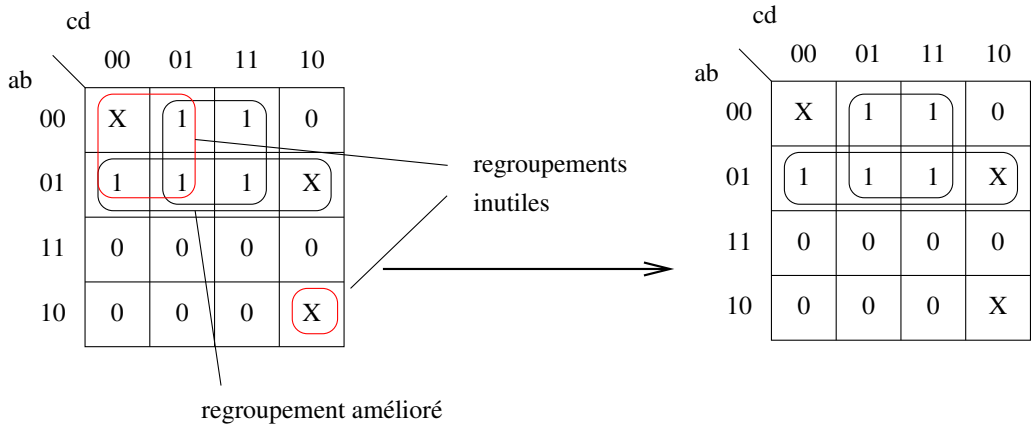
Cette méthode a cependant quelques inconvénients :

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

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

3.5.3 Simplification par Karnaugh

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



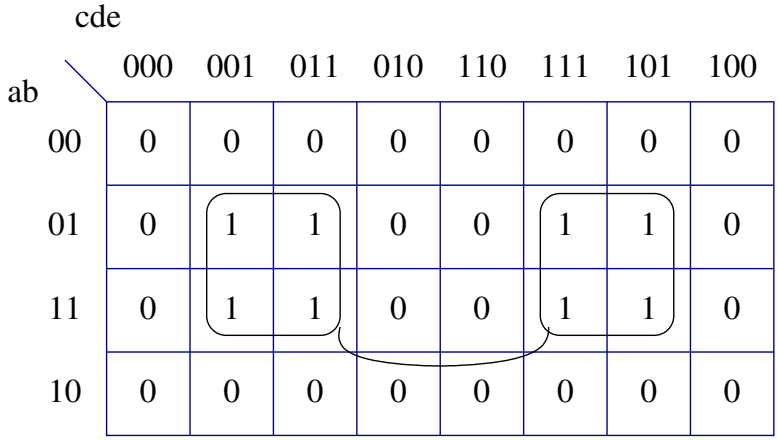
L'incidence des indifférents apparait de suite : tout les indifférents isolés doivent rester à "0", par contre d'autres produits comme "$\overline{a}\ b\ c\ \overline{d}$" vont effectivement améliorer la simplification.

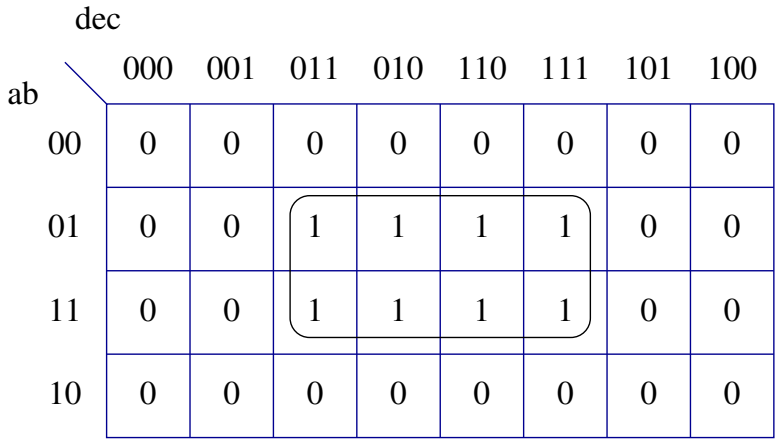
Remarque : Une erreur courante est d'ajouter de nouveaux regroupements pour couvrir certains indifférents. Même si ces regroupements sont grands, ils sont parfaitement inutiles (fonction déjà couverte) voire nuisible puisqu'ils recompliquent de nouveau la fonction ! En fait l'intégration d'un indiférent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

3.6 Limitation liée aux tables de Karnaugh

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

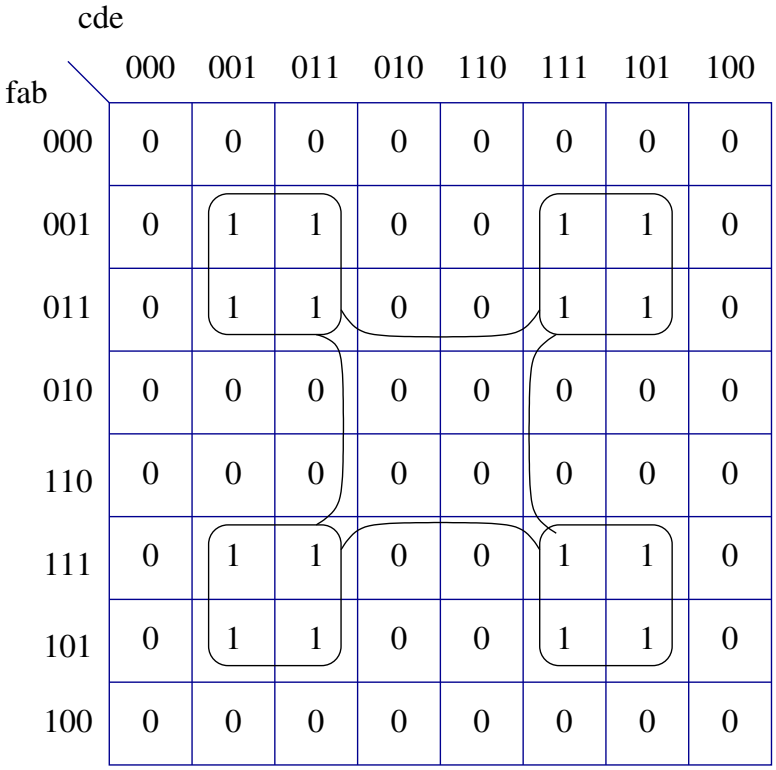
Considérons ainsi la fonction f_5 définie par : $f_5(a, b, c, d, e) = b\ e$
Voici deux facons équivalentes de représenter cette fonction par Karnaugh :





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

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



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

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

Chapitre 4

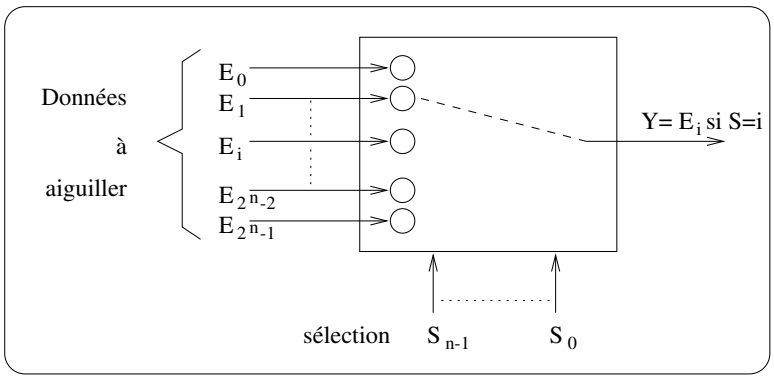
Aiguillages, générateurs de fonctions et de mintermes : Les Multiplexeurs

4.1 Introduction

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

4.2 La fonction de multiplexage

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



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

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

S_1S_0	Y
00	E_0
01	E_1
10	E_2
11	E_3

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

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

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

C'est à dire :

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

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

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

4.3 Générateur de fonction

4.3.1 Multiplexeurs possédant un nombre suffisant d'entrées

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

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

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

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

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

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

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

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

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

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

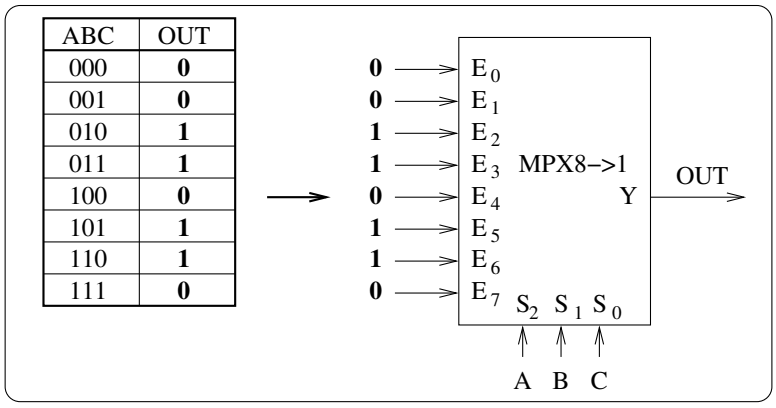


FIG. 4.1 – Exemple de cablâge de multiplexeur obtenu directement à partir de la table de vérité

4.3.2 Multiplexeurs ne possédant pas un nombre suffisant d'entrées

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

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

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

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

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

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

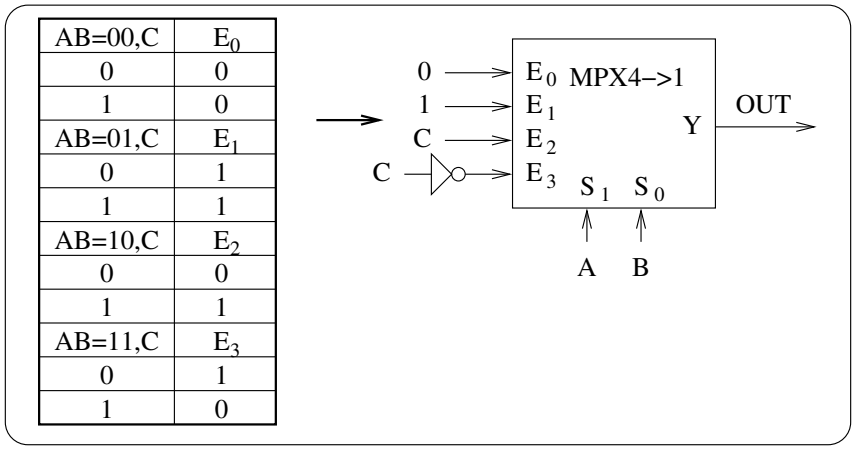


FIG. 4.2 – Synthèse indirecte à l'aide d'un multiplexeur 4 vers 1