CHAPITRE 7

ELEMENTS DE LOGIQUE COMBINATOIRE

CODAGE

1. Fonctions/Opérateurs logiques – Algèbre de Boole

1.1 Définitions

1.1.1 Variable logique et fonction logique

Une variable logique est une variable qui peut prendre **deux états** : "0" ou "1" ; "faux" ou "vrai".

Une fonction logique de variables logiques est une fonction dont les valeurs peuvent être les deux états "0" ou "1" ; "faux" ou "vrai".

1.1.2 Opérateurs logiques élémentaires

Les opérateurs logiques élémentaires servent à construire les fonctions logiques.
En algèbre classique on distingue quatre opérateurs de base : **+**, **-**, *****, **/** ; en algèbre de Boole ils sont au nombre de trois : **ET**, **OU**, **NON** ou encore « intersection », « union », « complément ».

Exemple : la fonction *f* est vraie **si** *a* est vrai **ou** si *b* et *c* sont vrais **ou** si *d* est faux s'écrit :

$$f = OU(a, ET(b,c), NON(d)) \text{ ou encore } f = a + bc + \overline{d}$$

1.1.3 Opérateurs logiques complets

On peut montrer qu'il est possible de synthétiser les trois opérateurs de base avec un seul type d'opérateur que l'on appelle alors « *opérateur complet* ».
Il existe deux opérateurs complets : le ET-NON ou **NAND** et le OU-NON ou **NOR**.

1.2 Ecriture des fonctions logiques

Une fonction logique, définie par son expression, peut être représentée de plusieurs manières.

1.2.1 Table de vérité

Les combinaisons de n variables logiques sont limitées à 2^n du fait que les variables ne peuvent prendre que deux valeurs. On peut ainsi représenter la fonction logique à l'aide d'un tableau faisant correspondre à chaque combinaison des variables la valeur de la fonction (0 ou 1) correspondante. On appelle cette représentation « **table de vérité** ».

Exemple : $f(a,b) = a + b$ signifie : $f = 1$ si $a = 1$ ou $b = 1$ ou $a = b = 1$ (voir table de vérité figure 7.1).

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

$f(0,0) = 0$
$f(0,1) = 1$
$f(1,0) = 1$
$f(1,1) = 1$

La table de vérité comporte 2^n lignes (une pour chaque combinaison des variables d'entrée)

Figure 7.1. Table de vérité.

1.2.2 Tableau de Karnaugh

Le tableau de karnaugh est une représentation de la table de vérité en deux dimensions. Il comprend 2^n cases.

Exemple : $f(a,b)=a+b$

$b\backslash a$	0	1
0	0	1
1	1	1

Au-delà de deux variables, le tableau devrait être un volume. Pratiquement on conserve évidemment une représentation à deux dimensions mais il faut alors respecter une **règle** qui est **l'adjacence de deux lignes ou colonnes**. Les combinaisons des variables respectent donc un **code de Gray**.

Exemple à 4 variables :

code de Gray ⇒

code de Gray ⇓

$cd\backslash ab$	00	01	11	10
00	0	1	0	1
01	1	1	1	1
11	0	1	0	1
10	1	1	1	1

Figure 7.2. Code de Gray.

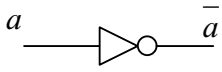
1.3 Fonctions élémentaires

1.3.1 Fonction/Opérateur Inverseur (NON)

Expression : Si *a* est vrai, $f(a)$ est faux : $f(a)=\overline{a}$.

Table de vérité :

a	$f(a)$
0	1
1	0

Symbole : 

1.3.2 Fonction/Opérateur AND (ET)

Expression : $f(a,b)$ est vraie si *a* est vrai et *b* est vrai : $f(a,b)=S=ab$.

Table de vérité :

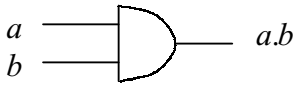
a	b	$S=ab$
0	0	0
0	1	0
1	0	0
1	1	1

Figure 7.3. Table de vérité du AND

Tableau de Karnaugh :

b\a	*0*	*1*
0	*0*	*0*
1	*0*	*1*

Symbole :



Utilisation en porte logique

$S = a$ si $b = 1$

$S = 0$ si $b = 0$

Le AND réalise donc une « porte » (ouverte si $b = 1$, fermée si $b = 0$).

1.3.3 Fonction/Opérateur OR (OU)

Expression : $f(a,b)$ est vraie si *a* est vrai ou *b* est vrai : $f(a,b) = S = a + b$.

Table de vérité :

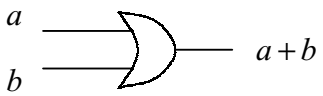
a	b	$S = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

Figure 7.4. Table de vérité du OR.

Tableau de Karnaugh :

b\a	*0*	*1*
0	*0*	*1*
1	*1*	*1*

Symbole :



Utilisation en porte logique

$S = a$ si $b = 0$

$S = 1$ si $b = 1$

Le OR réalise donc une « porte » (ouverte si $b = 0$, fermée si $b = 1$).

1.4 Propriétés des fonctions élémentaires

1.4.1 Elément neutre

OR : $a + 0 = a$ 0 = élément neutre pour le OR

AND : $a.1 = a$ 1 = élément neutre pour le AND

1.4.2 Commutativité

OR : $a + b = b + a$

AND : $a.b = b.a$

1.4.3 Associativité

OR : $(a+b)+c = a+(b+c) = a+b+c$

AND : $(a.b).c = a.(b.c) = a.b.c$

1.4.4 Distributivité double

$a.(b+c) = a.b + a.c$

1.4.5 Idempotence

OR : $a + a = a$

AND : a.a=a

1.4.6 Complément

$\overline{\overline{a}} = a$; $a + \overline{a} = 1$; $a.\overline{a} = 0$

1.5 Théorèmes fondamentaux

1.5.1 Théorème de De Morgan

Ce théorème s'écrit : $\overline{f}(a,b,...,+,.) = f(\overline{a},\overline{b},...,.,+)$

Le complément d'une fonction *f* de variables *a*, *b*, … s'obtient en remplaçant les variables par leur complément, et les opérateurs AND (**.**) par des OR (+) (et réciproquement).

Exemple : $f = ab + \overline{c}d + ad$ Ł $\overline{f} = (\overline{a}+\overline{b})(c+\overline{d})(\overline{a}+\overline{d})$.

On suppose que la priorité des opérateurs est la même qu'en algèbre classique, ce qui permet d'omettre certaines parenthèses.

1.5.2 Dualité

Une égalité demeure vraie si l'on permute les OR et AND et les 0 et 1.

Exemple : $a + \overline{a}b = a + b$ Ł $a.(\overline{a}+b) = ab$

$a + 1 = 1$ Ł $a.0 = 0$

1.6 Formes normales

Les formes normales sont des expressions particulières de fonctions logiques, sous formes de ET de OU ("produits de sommes") ou de OU de ET ("sommes de produits"). **Dans chaque terme apparaissent toutes les variables.**

1.6.1 Forme normale disjonctive (FND)

Théorème n° 1 de Shannon :
Toute fonction logique peut se décomposer en un OU de deux ET logiques :
$f(a,b,c,...) = a.f(1,b,c,...) + \overline{a}.f(0,b,c,...)$

En utilisant successivement ce théorème on arrive à **la forme normale disjonctive** :
$f(a,b,c,...) = a.b.f(1,1,c,...) + \overline{a}.b.f(0,1,c,...) + a.\overline{b}.f(1,0,c,...) + \overline{a}.\overline{b}.f(0,0,c,...)$

La fonction s'écrit donc comme un OU de toutes les combinaisons possibles de n variables pondérées par des 0 et 1 (**il y a donc 2^n termes**). Les termes pondérés par des 0 disparaissent et il

ne reste donc que les termes pondérés par des 1, d'où l'expression : **"développement de la fonction suivant les 1"**.

Exemple 1 : soit f(a,b,c) définie par le tableau de Karnaugh suivant, déterminer sa FND.

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

d'où $f(a,b,c) = \overline{a}\,\overline{b}\,\overline{c}.0 + \overline{a}b\overline{c}.0 + ab\overline{c}.0 + a\overline{b}\,\overline{c}.1 + \overline{a}\,\overline{b}c.1 + \overline{a}bc.1 + abc.0 + a\overline{b}c.1$

soit : $f(a,b,c) = a\overline{b}\,\overline{c} + \overline{a}\,\overline{b}c + \overline{a}bc + a\overline{b}c$

On note que ce résultat est obtenu directement à partir du tableau de Karnaugh (ou de la table de vérité) en écrivant que f est l'union (+) des combinaisons de a, b, c pour lesquelles f vaut 1.

Exemple 2 : $f(a,b,c) = \overline{a}\,\overline{b}\,\overline{c} + \overline{a}\,\overline{b}c + \overline{a}bc + a\overline{b}c$

Trouver le tableau de Karnaugh (ou la table de vérité) de f.

sous sa forme normale, f s'écrit :

$$f(a,b,c) = \overline{a}\,\overline{b}\,\overline{c}.f(0,0,0) + \overline{a}\,\overline{b}c.f(0,0,1) + \overline{a}b\overline{c}.f(0,1,0) + \overline{a}bc.f(0,1,1)$$
$$+ a\overline{b}\,\overline{c}.f(1,0,0) + a\overline{b}c.f(1,0,1) + ab\overline{c}.f(1,1,0) + abc.f(1,1,1)$$

avec $f(0,0,0)=1 \quad f(0,0,1)=1 \quad f(0,1,0)=0 \quad f(0,1,1)=1$
$f(1,0,0)=0 \quad f(1,0,1)=1 \quad f(1,1,0)=0 \quad f(1,1,1)=0$

d'où le tableau de Karnaugh :

c \ ab	00	01	11	10
0	1	0	0	0
1	1	1	0	1

On note que cela revient à mettre des 1 dans les cases définies par les combinaisons de variables présentes dans l'expression de f.

1.6.2 Forme normale conjonctive (FNC)

Théorème n° 2 de Shannon :
Toute fonction logique peut se décomposer en un ET de deux OU logiques :
$f(a,b,c,...) = (a + f(0,b,c,...)).(\overline{a} + f(1,b,c,...))$

Cette forme est la duale de la FND. Comme précédemment l'utilisation successive de ce théorème permet d'arriver à la forme normale conjonctive :

Exemple 1 :

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

1er terme : $a + b + c + f(0,0,0)$
2ème terme : $a + b + \overline{c} + f(0,0,1)$
3ème terme : $a + \overline{b} + c + f(0,1,0)$
….

8ème terme : $\bar{a}+\bar{b}+\bar{c}+f(1,1,1)$

$$f(a,b,c)=[(a+b+c)+0][(a+b+\bar{c})+1][(a+\bar{b}+c)+0][(a+\bar{b}+\bar{c})+1]$$
$$[(\bar{a}+b+c)+1][(\bar{a}+b+\bar{c})+1][(\bar{a}+\bar{b}+c)+0][(\bar{a}+\bar{b}+\bar{c})+0]$$

Les termes contenant 1 disparaissent :

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

En pratique, on cherche les cases à 0, puis on fait le ET des combinaisons de variables qui leur "correspondent" en prenant garde que la "correspondance" n'est pas la même que pour la FND.

Développement suivant les "0".

Exemple 2 :

c \ ab	00	01	11	10
0	1	0	1	0
1	0	1	0	0

Développement suivant les 0 :

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

Ne pas confondre les règles de développement suivant les "1" et les "0".

Exemple 3 : Passage de la FNC au tableau de Karnaugh

$f(a,b,c)=(a+b+c).(a+\bar{b}+c).(\bar{a}+\bar{b}+c).(\bar{a}+\bar{b}+\bar{c})$ s'écrit aussi :

$$f(a,b,c)=[(a+b+c)+f(0,0,0)][(a+b+\bar{c})+f(0,0,1)][(a+\bar{b}+c)+f(0,1,0)][(a+\bar{b}+\bar{c})+f(0,1,1)]$$
$$[(\bar{a}+b+c)+f(1,0,0)][(\bar{a}+b+\bar{c})+f(1,0,1)][(\bar{a}+\bar{b}+c)+f(1,1,0)][(\bar{a}+\bar{b}+\bar{c})+f(1,1,1)]$$

avec ici : $f(0,0,0)=0$ $f(0,0,1)=1$ $f(0,1,0)=0$ $f(0,1,1)=1$
$f(1,0,0)=1$ $f(1,0,1)=1$ $f(1,1,0)=0$ $f(1,1,1)=0$

d'où le tableau de Karnaugh :

c \ ab	00	01	11	10
0	0	0	0	1
1	1	1	0	1

1.7 Simplification de fonctions

La simplification d'une fonction consiste à obtenir son expression la plus compacte possible afin de minimiser le nombre d'opérateurs logiques nécessaires à sa réalisation.

On distingue deux méthodes de simplification :

méthode **algébrique** ;

méthode **graphique**.

1.7.1 Simplification graphique

1.7.1.1 Principe

Cette méthode repose sur l'utilisation des tableaux de Karnaugh. Elle consiste à mettre en évidence des associations du type :

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

On remarque que les regroupements ci-dessus correspondent aux cas où l'on a 2, 4, 8, (2^n en général) **cases adjacentes** sur le tableau de Karnaugh, qui sont simultanément égales à 1.

1.7.1.2 Adjacence des cases

Deux cases adjacentes sur le tableau de Karnaugh correspondent à des combinaisons différant d'un seul bit (ceci est dû l'utilisation du code de Gray). Ceci est valable à l'intérieur du tableau mais aussi sur ses bords : en passant du bord droit au bord gauche ou du haut au bas il y a adjacence. Ceci revient à dire que l'on peut considérer le tableau comme une sphère.

v **Règle pratique**

La règle consiste donc à «fusionner » ces 2^n cases adjacentes pour trouver l'expression résultante. On regarde la ou les variables qui change(nt) entre les cases fusionnées ; ces variables sont alors supprimées dans la nouvelle expression simplifiée.

v **Exemples**

Exemple 1 :

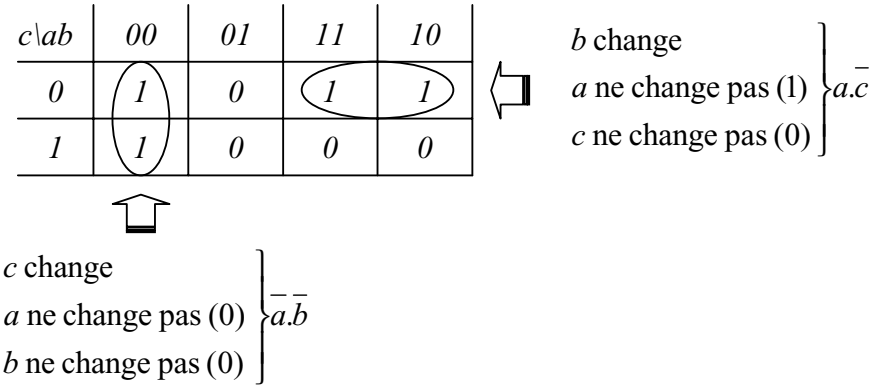


Figure 7.5. Principe de regroupement (exemple 1).

Donc : $f = \bar{a}\bar{b}c + \bar{a}\bar{b}c + a\bar{b}c + a\bar{b}\bar{c} = \bar{a}\bar{b} + a\bar{c}$

Exemple 2 :

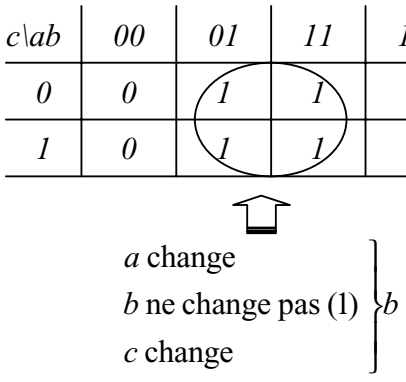


Figure 7.6. Principe de regroupement (exemple 2).

Donc : $f = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + abc + \bar{a}bc = b$

Exemple 3 :

cd\ab	*00*	*01*	*11*	*10*
00	*1*	*0*	*0*	*1*
01	*0*	*0*	*0*	*0*
11	*0*	*0*	*0*	*0*
10	*1*	*0*	*0*	*1*

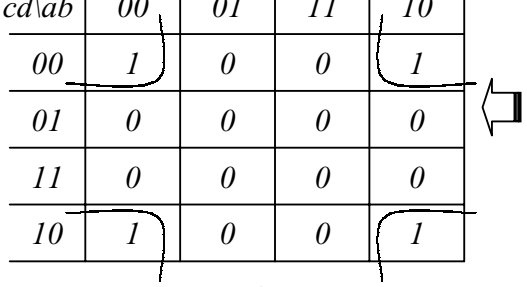


Figure 7.7. Principe de regroupement (exemple 3).

Donc : $f = \overline{a}bc\overline{d} + \overline{a}bcd + a\overline{b}c\overline{d} + a\overline{b}cd = \overline{b}\overline{d}$

v **Récapitulation des règles** :

1. On ne regroupe que 2^n cases (2,4,8,16,…).
2. les regroupements sont forcément des rectangles ou carrés (compte tenu des permutations pour les bords). Pas de « L », pas de croix, …
3. l'expression sera d'autant plus compacte que l'étendue des regroupements est grande. Pour un regroupement occupant la moitié du tableau il n'y a plus qu'une variable, pour le quart il reste deux variables, pour un regroupement de deux cases, il reste n-1 variables. A la limite un regroupement de tout le tableau fait disparaître toute variable ($f = 1$). D'une manière générale, un regroupement de 2^i cases conduit à supprimer i variables.
4. Il est inutile de regrouper des « 1 » qui ont tous déjà été regroupés par ailleurs. On parle alors de terme inclus.

1.7.2 Simplification algébrique

Cette technique de simplification repose sur l'utilisation des propriétés de l'algèbre de Boole et des théorèmes fondamentaux. Elle est moins systématique que la simplification graphique mais peut parfois donner des résultats rapidement.

v **Utilisation des identités particulières**

(1) $ab + a\overline{b} = a$ (4) $(a+b)(a+\overline{b}) = a$

(2) $ab + a = a$ (5) $a(a+b) = a$

(3) $a + \overline{a}b = a + b$ (6) $a(\overline{a}+b) = ab$

Exemple :

$f = abc + ab\overline{c} + a\overline{b}cd$

$f = \quad ab \quad + a\overline{b}cd = a(b + \overline{b}cd) = ab + acd$

v **Ajout de terme existant (utilisation de l'idempotence)**

Exemple : $f = abc + \overline{a}bc + a\overline{b}c + ab\overline{c}$

$f = abc + \overline{a}bc + abc + a\overline{b}c + abc + ab\overline{c}$

$f = \quad bc \quad + \quad ac \quad + \quad ab$

v **Suppression de terme inclus**

Exemple : $f = ab + \bar{b}c + ac$

$$f = ab + \bar{b}c + ac\left(b + \bar{b}\right) = ab + \bar{b}c + acb + ac\bar{b} = ab\left(1 + c\right) + \bar{b}c\left(1 + a\right) = ab + \bar{b}c$$

1.7.3 Combinaisons impossibles

Parfois des combinaisons particulières des valeurs des variables ne peuvent pas se produire, pour des raisons physiques ou technologiques. On utilise alors ces combinaisons pour simplifier la fonction.

Le principe consiste à dire que puisque la combinaison n'apparaît pas, on considère que si elle apparaissait, elle donnerait un 1 ou un 0, selon ce qui nous arrange pour la simplification (un ascenseur ne peut être à 2 étages au même instant).

Exemple 1 : on crée la fonction f(a,b,c,d) telle que f = 1 ssi 1 < N < 5 avec N codé en BCD.

On dispose de 4 bits pour coder les nombres de 0 à 9 ; or 4 bits permettent de coder jusqu'à $2^4 = 16$ valeurs :

N	a	b	c	d	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	impossible				1 ou 0
11	(ne				1 ou 0
12	peut				1 ou 0
13	pas				1 ou 0
14	se				1 ou 0
15	produire)				1 ou 0

Tableau de Karnaugh :

cd \ ab	00	01	11	10
0	0	1	X	0
01	0	0	X	0
11	1	0	X	X
10	1	0	X	X

Si on ne tient pas compte des états 10 à 15, f s'écrit : $f = \overline{a}\overline{b}c + \overline{a}b\overline{c}\overline{d}$

En considérant que les combinaisons impossibles sont affectées arbitrairement des valeurs 0 ou 1 on peut obtenir : $f = \overline{b}c + b\overline{c}\overline{d}$

Règle pratique : on remplit le cases avec le **symbole ϕ** qui montre que l'on peut choisir la valeur que l'on veut (0 ou 1) pour cette case.

1.8 Fonctions/Opérateurs complets

1.8.1 Fonction/Opérateur NAND

Expression : $f(a,b)$ est vraie si *a* et *b* est faux : $f(a,b) = S = \overline{a.b}$.

Table de vérité :

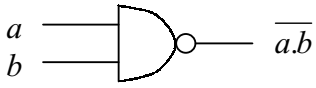
a	*b*	$S = \overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0

Figure 7.8. Table de vérité du NAND.

Tableau de Karnaugh :

b\a	*0*	*1*
0	*1*	*1*
1	*1*	*0*

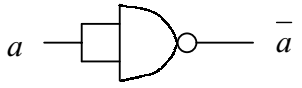
Symbole :



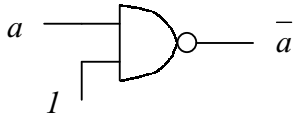
Synthèse de l'inverseur (NON) avec des NAND

$S = \overline{a.b}$;

si $a = b$ $\quad S = \overline{a.a} = \overline{a}$

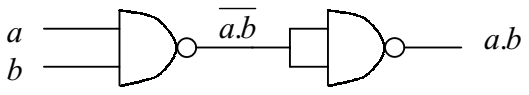


si $b = 1$ $\quad S = \overline{a.1} = \overline{a}$

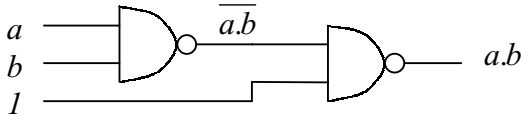


Synthèse du AND avec des NAND

$S = a.b = \left(\overline{\overline{a.b}}\right)$

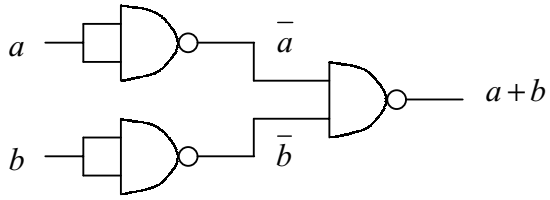


$S = \left(\overline{\overline{a.b}}\right)1 = \left(\overline{\overline{a.b}}\right)$



Synthèse du OR avec des NAND

$$S = \overline{\overline{a+b}} = \overline{\left(\overline{a}.\overline{b}\right)} = \overline{\left(\overline{a.a}\right)}\overline{\left(\overline{b.b}\right)}$$



1.8.2 Fonction/Opérateur NOR

Expression : $f(a,b)$ est vraie si *a* ou *b* est faux : $f(a,b) = S = \overline{a+b}$.

Table de vérité :

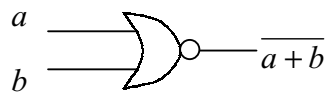
a	b	$S = \overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

Figure 7.9. Table de vérité du NOR.

Tableau de Karnaugh :

$b\backslash a$	*0*	*1*
0	*1*	*0*
1	*0*	*0*

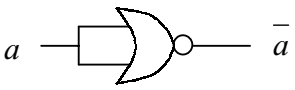
Symbole :



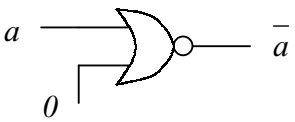
Synthèse de l'inverseur (NON) avec des NOR

$S = \overline{a+b}$;

si $a = b$ $\quad S = \overline{a+a} = \overline{a}$



si $b = 1$ $\quad S = \overline{a+0} = \overline{a}$



Synthèse du AND avec des NOR

$$S = a.b = \left(\overline{\overline{a.b}}\right) = \overline{\overline{a}+\overline{b}} = \left(\overline{\overline{a+a}}\right)+\left(\overline{\overline{b+b}}\right) = \left(\overline{\overline{a+0}}\right)+\left(\overline{\overline{b+0}}\right)$$

Synthèse du OR avec des NOR

$$S = a+b = \overline{\overline{a+b}} = \left(\overline{\overline{a+b}}\right) = \left(\overline{\overline{a+b}}\right)+0$$

1.8.3 Utilisation des NOR et NAND pour la synthèse d'une fonction quelconque

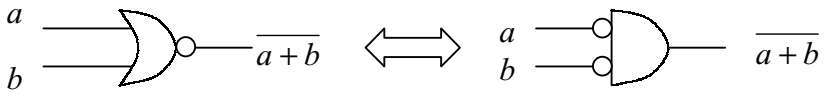
Les deux opérateurs NOR et NAND étant universels, ils peuvent être utilisés indifféremment pour synthétiser une même fonction. Pour les faire apparaître on peut procéder comme précédemment, mais également utiliser une règle simple.

Remarque : en pratique, le NAND est plus rapide que le NOR ; il est donc plus utilisé.

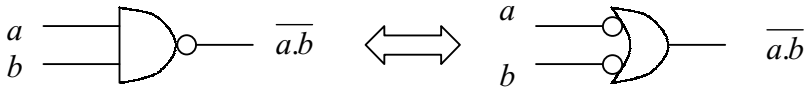
Règle de De Morgan graphique

On peut exprimer le théorème de De Morgan directement sur l'expression graphique de la fonction :

$\overline{a+b} = \bar{a}.\bar{b}$ se dessine :



$\overline{a.b} = \bar{a}+\bar{b}$ se dessine :



On peut déduire une **règle simple de synthèse graphique** :
« On peut remplacer directement un NOR (NAND) par un AND (OR) dont les entrées sont complémentées ».

1.9 Fonction/Opérateur XOR (OU – Exclusif)

1.9.1 XOR à deux entrées

Expression : $f(a,b)$ est vraie si *a* est vrai ou *b* est vrai, mais pas les deux: $f(a,b) = S = a \oplus b$.

Table de vérité :

a	b	$S = a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

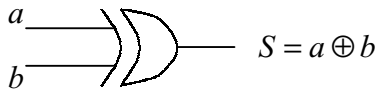
Figure 7.10. Table de vérité du XOR.

D'après la table de vérité, on obtient : $S = a \oplus b = a\bar{b} + \bar{a}b$

Tableau de Karnaugh :

b\a	*0*	*1*
0	*0*	*1*
1	*1*	*0*

Symbole :



1.9.2 XOR à plusieurs entrées

Définition

$XOR(a,b,c,...) = 1$ si le nombre de variables à 1 est impair.

Exemple à trois variables : $S = a \oplus b \oplus c$

c\ab	*00*	*01*	*11*	*10*
0	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*0*

Figure 7.11. XOR à plusieurs entrées.

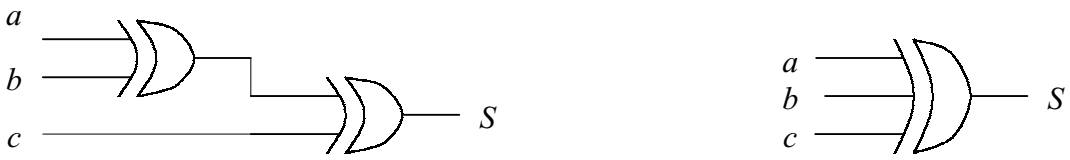
La représentation d'un XOR en tableau de Karnaugh est un damier avec un 0 en haut à gauche (car toutes les variables sont à 0).

Associativité

Il est facile de montrer que le XOR possède la propriété d'associativité : $a \oplus b \oplus c = a \oplus (b \oplus c) = (a \oplus b) \oplus c$.

$$a \oplus b \oplus c = \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc + a\bar{b}\bar{c} = \bar{a}(\bar{b}c + b\bar{c}) + a(bc + \bar{b}\bar{c}) = \bar{a}(b \oplus c) + a\overline{(b \oplus c)} = a \oplus (b \oplus c)$$

La symétrie permet d'effectuer toutes les permutations en a, b ou c :



1.9.3 Utilisation du XOR

Détection de non égalité de deux variables

$a \oplus b = 0$ si $a = b$; $a \oplus b = 1$ si $a \neq b$.

Détecteur d'imparité

$a \oplus b \oplus c = 0$ si le nombre de variables à 1 est pair

$a \oplus b \oplus c = 1$ si le nombre de variables à 1 est impair

Opérateur programmable

Le XOR est l'opérateur programmable le plus simple. Avec les AND, OR, NON, NOR et NAND, la relation entre sortie (f) et entrée (a,b,c,…) est figée. Avec un XOR on peut jouer sur cette relation de manière à la modifier en fonction d'une entrée qui joue le rôle de commande.

Principe :

a	p	S
0	0	0
0	1	1
1	0	1
1	1	0

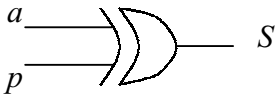
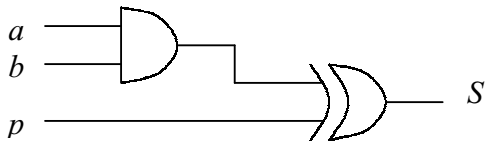


Figure 7.12. Porte logique.

$$\left.\begin{array}{l} S = 0 \text{ si } a = 0 \text{ et } p = 0 \\ S = 1 \text{ si } a = 1 \text{ et } p = 0 \end{array}\right\} S = a \text{ si } p = 0 \qquad \left.\begin{array}{l} S = 1 \text{ si } a = 0 \text{ et } p = 1 \\ S = 0 \text{ si } a = 1 \text{ et } p = 1 \end{array}\right\} S = \bar{a} \text{ si } p = 1$$

p est donc l'entrée de contrôle ; a peut être la sortie d'autres opérateurs.

Exemple :



$S = ab$ si $p = 0$; $S = \overline{ab}$ si $p = 1$

Additionneur élémentaire sur deux bits

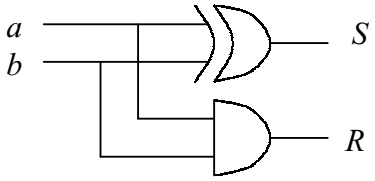
Les fonctions « somme (S) » et « retenue (R) » sont définies ainsi :

a	*b*	*R*	*S*
0	*0*	*0*	*0*
0	*1*	*0*	*1*
1	*0*	*0*	*1*
1	*1*	*1*	*0*

Figure 7.13. Table de vérité de l'additionneur élémentaire 2 bits.

On constate que : $S = a \oplus b$
$R = ab$

Ce circuit additionneur est donc très facile à réaliser avec un XOR et un AND :



2. Codeurs, décodeurs, transcodeurs

2.1 Généralités sur le codage

Le codage est une opération permettant de transmettre une information de la meilleure façon possible :

la plus simple ;
la plus rapide ;
la plus sécurisée (confifentialité) ;

… selon ce cas.

Chaque langue est par exemple un code permettant à des hommes et des femmes de dialoguer et d'échanger des informations.

2.2 Définitions

2.2.1 Codeur

Un codeur fait correspondre à l'activation d'une entrée particulière une combinaison de bits en sortie (le circuit a donc une entrée et *N* sortie).
Le codage de 2^n combinaisons en entrée nécessite *n* bits en sortie.

Exemple : panneau lumineux indicateur de trajet dans le métro ; la sélection d'une destination particulière entraîne l'allumage d'une série de lampes qui balise l'itinéraire.

2.2.2 Décodeur

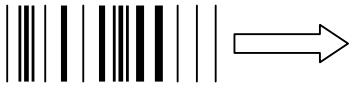
Un décodeur réalise la fonction inverse, il active une sortie particulière lorsqu'on lui présente une combinaison donnée de bits en entrée (le circuit a donc *N* entrée et une sortie).
n bits en entrée fournissent 2^n combinaisons possibles en sortie.

Exemple : clavier de porte automatique : le déverrouillage de la porte s'effectue lorsqu'on présente au clavier la bonne combinaison.

2.2.3 Transcodeur

Un transcodeur est un circuit qui permet de passer d'un code à un autre (*N* entrées et *N* sorties).

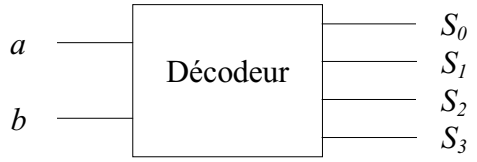
Exemple : lecteur de code barre : la combinaison des barres correspond à une succession de nombres décimaux.



2.3 Décodeur

2.3.1 Exemple : décodeur binaire 1 parmi 4.

Le « numéro » de la sortie activée correspond au nombre binaire de deux bits *a*, *b* présenté en entrée du circuit.



Pour réaliser un décodeur, il suffit d'écrire la table de vérité des différentes sorties du décodeur. L'expression des sortie S_i se déduit alors simplement de la table de vérité (Figure 7.14).

$S_0 = \bar{a}\bar{b}$
$S_1 = \bar{a}b$
$S_2 = a\bar{b}$
$S_3 = ab$

a	*b*	*S0*	*S1*	*S2*	*S3*
0	*0*	*1*	*0*	*0*	*0*
0	*1*	*0*	*1*	*0*	*0*
1	*0*	*0*	*0*	*1*	*0*
1	*1*	*0*	*0*	*0*	*1*

Figure 7.14. Décodeur 1 parmi 4.

2.3.2 Association de décodeurs, décodeur 1 parmi 2^n

Lorsque le nombre d'entrées est grand, le décodeur peut devenir complexe. En pratique, on utilise souvent plusieurs décodeurs plus petits (les circuits commerciaux sont forcément limités en taille et complexité, et ils ne peuvent pas toujours répondre exactement aux besoins spécifiques).

Exemple : décodeur binaire 1 parmi 8 (3 bits) à réaliser avec des décodeurs 1 parmi 4 (2 bits).
Si l'on ajoute une porte à la sortie d'un décodeur, on peut activer (valider) celui-ci ou bien forcer ses sorties à 0 quelles que soient les niveaux sur les entrées. On appelle cette entrée la **validation**.

Si $Val = 0 \Rightarrow S_i = 0 \; \forall a,b$

Si $Val = 1 \Rightarrow$ fonctionnement en décodeur normal

$S_0 = \bar{a}\bar{b}Val$

$S_1 = \bar{a}b.Val$

$S_2 = a\bar{b}Val$

$S_3 = ab.Val$

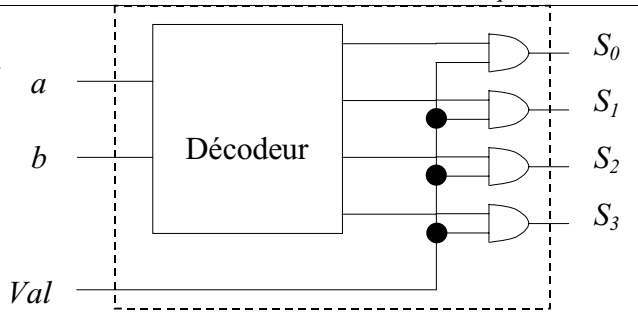


Figure 7.15. Décodeur avec entrée de validation.

Le décodeur incluant les portes de validation est appelé "**décodeur complet**" ou "**décodeur avec entré de validation**".
Pour réaliser le décodeur 1 parmi 8, il suffit alors d'utiliser deux décodeurs 1 parmi 4 complets.

a	*b*	*c*	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
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*

Figure 7.16. Table de vérité du décodeur 1 parmi 8.

Les entrées de validation sont utilisées pour sélectiionner un décodeur parmi 2 à l'aide de la variable *a* comme le montre la figure 7.17.

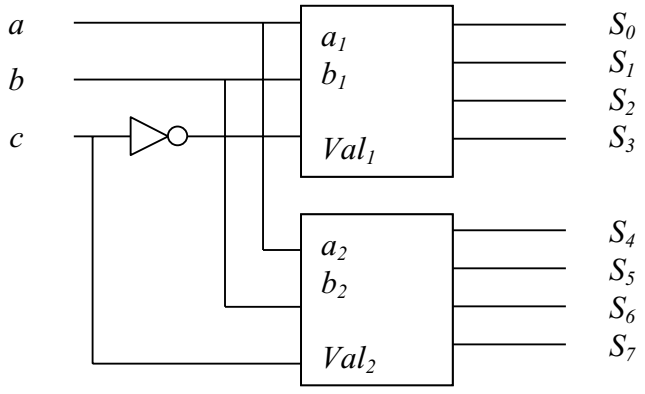


Figure 7.17. Décodeur 1 parmi 8 réalisé avec deux décodeurs 1 parmi 4.

Généralisation : décodeur 1 parmi 2^n

On peut généraliser le résultat précédent en « fabriquant » le signal Val_i à l'aide d'un décodeur.

2.3.3 Codeur

v Synthèse complète du codeur

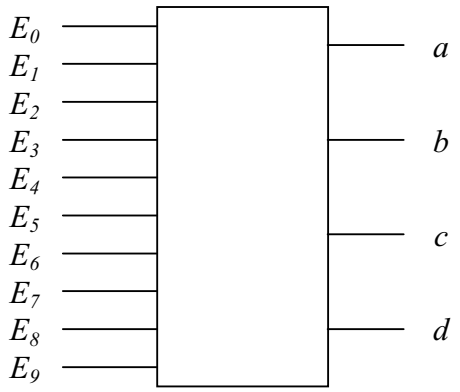
Chacune des variables de sortie est réalisée en fonction des combinaisons des variables d'entrée. Un codeur à *n* sorties aura au maximum 2^n entrées. Pour envisager toutes les combinaisons des variables d'entrée, il faudrait donc *n* tableaux de Karnaugh à 2^{2^n} cases.

Par exemple, 4 variables de sortie pour un codeur binaire permettent de coder $2^4 = 16$ variables d'entrée. Pour synthétiser chaque sortie il faudrait un tableau de Karnaugh à 16 variables, soit $2^{16} = 65536$ cases (ou une table de vérité de 65536 lignes).

v Synthèse simplifiée du codeur

Pour un fonctionnement classique, il ne peut y avoir qu'une seule entrée à 1 à la fois. On a donc 2^n combinaisons possibles des variables d'entrée au lieu de 2^{2^n}.

Exemple : codeur BCD. BCD signifie « Binary Coded Decimal » soit « Décimal Codé en Binaire ». Dans ce code, les chiffres décimaux (0 à 9) sont codés en binaire sur 4 bits.
On utilise une table de vérité condensée qui utilise le fait qu'il ne peut y avoir qu'une seule entrée à 1 à la fois (ceci revient à placer des Φ dans toutes les cases pour lesquelles il n'y a qu'une seule entrée à 1). La table de vérité condensée est donnée sur la figure 7.18.



entrée à 1	*a*	*b*	*c*	*d*
E_0	0	0	0	0
E_1	0	0	0	1
E_2	0	0	1	0
E_3	0	0	1	1
E_4	0	1	0	0
E_5	0	1	0	1
E_6	0	1	1	0
E_7	0	1	1	1
E_8	1	0	0	0
E_9	1	0	0	1

Figure 7.18. Table de vérité condensée du codeur BCD.

On obtient aisément les expressions des variables de sortie *a*,*b*,*c*,*d* :
$a = E_8 + E_9$; $b = E_4 + E_5 + E_6 + E_7$; $c = E_2 + E_3 + E_6 + E_7$; $d = E_1 + E_3 + E_5 + E_7 + E_9$.
La réalisation du codeur est donnée sur la figure 7.19.

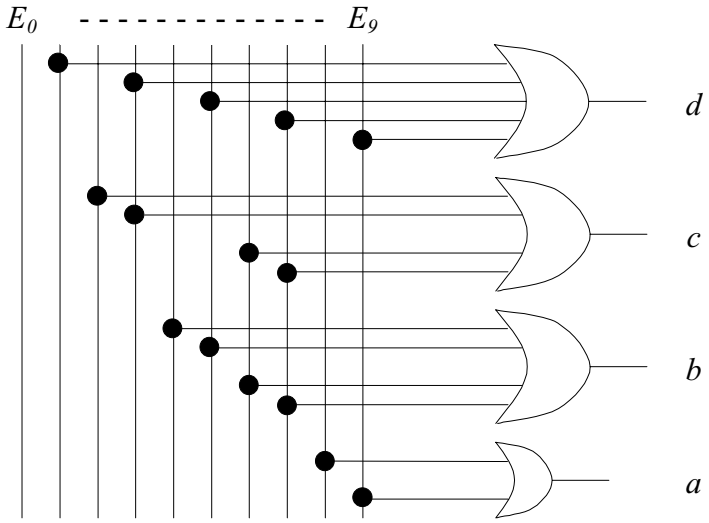


Figure 7.19. Codeur BCD.

Ce codeur est très simple mais il présente des inconvénients :

- si deux entrées sont activées simultanément la sortie peut prendre une valeur mal définie ;
- la sortie code « 0 » si aucune entrée n'est activée (E_0 n'est pas connecté).

Pour répondre au premier inconvénient, on peut ajouter des contraintes à la synthèse simplifiée. Ceci donne lieu par exemple au **codeur prioritaire**. Dans ce codeur on remplace un certain nombre de Φ par des 0 ou des 1 en fonction de critères particuliers, comme par exemple la priorité d'une entrée par rapport à une autre.

2.3.4 transcodeur

Le transcodeur sert à passer d'un code à un autre. On calcule chaque sortie S_j en fonction des combinaisons des entrées E_i (voir figure 7.20).

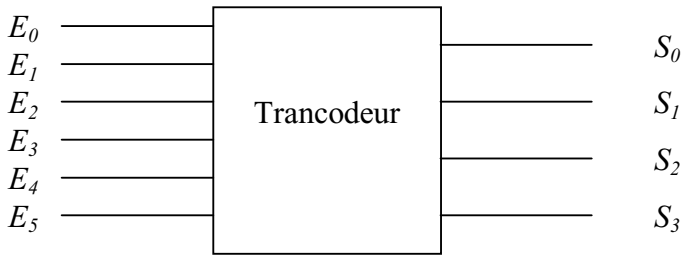


Figure 7.20. Exemple de transcodeur.

Exemple 1 : transcodeur à deux bits : binaire naturel Ł code de Gray.
La table de vérité est donnée sur la figure 7.21.

a	*b*	*x*	*y*
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Figure 7.21. Table de vérité transcodeur 2 bits, binaire Ł code de Gray.

On déduit : $x = a$ et $y = a\bar{b} + \bar{a}b$.

Exemple 2 : décodeur BCD Ł 7 segments.
Souvent on affiche les nombres sur des Diodes ElectroLuminescentes (DEL) à 7 segments qui forments le chiffre à afficher (voir figure 7.22). Les sorties *a* à *g* s'expriment alors en fonction des entrées A à D (code BCD).

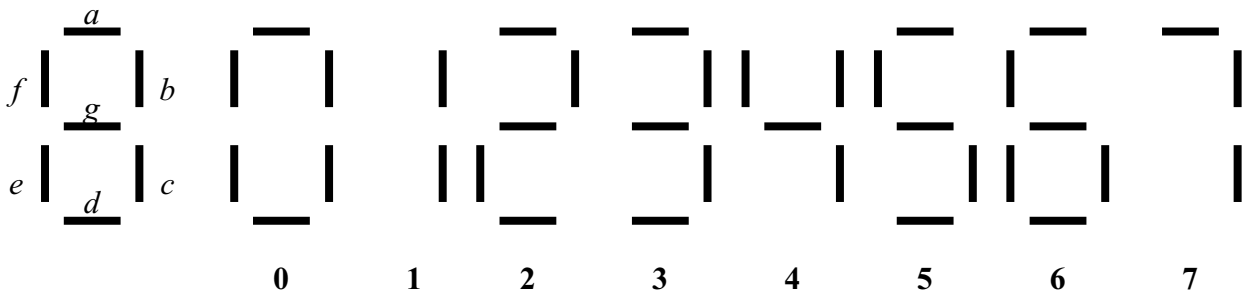


Figure 7.22. Afficheur 7 segments.