Safwan El Assad, Philippe Guyonnet

Électronique numérique

Fondements et applications

Incluant 57 exercices avec solutions détaillées,
5 travaux pratiques et un mini-projet en VHDL
avec éléments concis de solutions

ellipses

Partie 1 : Cours

Chapitre 1. Logique combinatoire

1.1. Représentation des nombres et conversion

1.1.1. Numération binaire

Un nombre N exprimé en base b donnée, $(N)_b$ est écrit sous forme d'une juxtaposition de symboles (chiffres) appartenant à la base b en question.

$$(N)_b = (a_n\ a_{n-1} \cdots a_i\ \cdots a_1\ a_0\ , a_{-1}\ a_{-2}\ \cdots a_{-m})_b \quad (1.1)$$

La valeur numérique de ce nombre est obtenue en exprimant sous forme polynomiale le nombre $(N)_b$, soit :

$$\begin{aligned}(N)_b = a_n b^n + a_{n-1}b^{n-1} + \cdots + a_i b^i + \cdots + a_1 b^1 + a_0 b^0, + a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots \\ + a_{-m}b^{-m} \\ = \sum_{k=-m}^{n} a_k b^k \end{aligned} \quad (1.2)$$

Où : a_k : symbole (chiffre) de rang k, appartenant à la base b

b^k : poids du symbole (chiffre) a_k

k : rang du symbole (chiffre) a_k Les bases utilisées en pratique sont :

base 2 (binaire), $b = 2$, $a_k \in [0, 1]$,

base 8 (octale), $b = 8$, $a_k \in [0, 1, \cdots, 6, 7]$,

base 10 (décimale), $b = 10$, $a_k \in [0, 1, \cdots, 8, 9]$,

base 16 (hexadécimale), $b = 16$, $a_k \in [0, 1, \cdots, 8, 9, A, B, C, D, E, F]$, avec $A = 10, \cdots, F = 15$.

1.1.2. Changement de base de numération

1.1.2.1. De la base b vers la base 10

Dans ce cas, il suffit d'exprimer le nombre sous sa forme polynomiale et de faire le calcul

Exemples :

$$\begin{aligned}(101,11)_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0, 1 \times 2^{-1} + 1 \times 2^{-2} = 4 + 0 + 1, 0{,}5 + 0{,}25 \\ &= (5{,}75)_{10} \\ (257,24)_8 &= 2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0, 2 \times 8^{-1} + 4 \times 8^{-2} \\ &= 128 + 40 + 7, 0{,}25 + 0{,}0625 = (175{,}3125)_{10} \\ (2A9F, A)_{16} &= 2 \times 16^3 + 10 \times 16^2 + 9 \times 16^1 + 15 \times 16^0, 10 \times 16^{-1} \\ &= 8192 + 2560 + 144 + 15, 0{,}625 = (10911{,}625)_{10}\end{aligned}$$

1.1.2.2. De la base 10 vers la base b

Pour convertir un nombre écrit en base 10, $(N)_{10}$ en un nombre écrit en base b donné, $(Nx)_b$, il faut trouver les valeurs des différents symboles a_k de la base b.

$$(Nx)_b = (a_n\ a_{n-1}\ \cdots\cdots a_0, a_{-1}\ a_{-2}\ \cdots\cdots a_{-m})_b$$
$$= (a_n\ a_{n-1}\ \cdots\cdots a_0) + (0, a_{-1}\ a_{-2}\ \cdots\cdots a_{-m}\ )$$
$$= PE(Nx)_b + PF(Nx)_b \qquad (1.3)$$

Où : $PE(Nx)_b$ est la partie entière du nombre et $PF(Nx)_b$ est la partie fractionnaire du nombre.

Pour la partie entière du nombre, deux méthodes peuvent être utilisées pour déterminer les valeurs des différents symboles a_k, qui sont :

- divisions successives par la base b ;
- soustractions successives de la plus grande puissance entière de la base b.

Pour la partie fractionnaire du nombre une seule méthode est disponible pour déterminer les valeurs des différents symboles a_k :

- multiplications successives par la base b.

1.1.2.2.1. Détermination de la partie entière par la méthode divisions successives par la base b : $PE(N)_{10} \Longrightarrow PE(Nx)_b$

Écrivons la partie entière $PE(N)_b$ sous forme polynomiale :

$PE(N)_b = a_n b^n + a_{n-1}b^{n-1} + \cdots\cdots + a_1 b^1 + a_0$

Faisons les divisions successives par la base b :

$PE(N)_b/b = \underbrace{[a_n b^{n-1} + a_{n-1}b^{n-2} + \cdots\cdots + a_1]}_{quotien=q_1} + \underbrace{a_0}_{reste}$: la première division par b détermine le symbole a_0

$q_1/b = \underbrace{[a_n b^{n-2} + a_{n-1}b^{n-3} + \cdots\cdots + a_2]}_{quotien=q2} + \underbrace{a_1}_{reste}$: la deuxième division par b détermine le symbole a_1

Et ainsi de suite.

Après $(n-1)$ divisions, on aboutit à :

$q_{(n-1)}/b = \underbrace{a_n}_{q_n} + \underbrace{a_{n-1}}_{reste}$

Exemple : $(75)_{10} = (?\cdots?)_2$

Par divisons successives on obtient :

75	2					
–74	37	2				
1 a_0	–36	18	2			
	1	–18	9	2		
		0	–8	4	2	
			1	–4	2	2
				0	–2	1 a_6
					0	

D'où :

$(75)_{10} = (1\ 0\ 0\ 1\ 0\ 1\ 1\ )_2$

a_6 a_0

1.1.2.2.2. Détermination de la partie entière par la méthode soustractions successives de la plus grande puissance entière de la base

$b : PE(N)_{10} \Longrightarrow PE(Nx)_b$

On cherche i tel que : $b^i \leq PE(N)_{10} < b^{i+1}$

Puis on soustrait b^i au nombre de départ : $PE_1(N)_{10} = PE(N)_{10} - b^i$

On répète ces deux opérations avec $PE_1(N)_{10}$, et ainsi de suite jusqu'à ce que $PE_k(N)_{10} < b$

Appliquons la méthode sur l'exemple précédent $(75)_{10} = (? \cdots ?)_2$

i	6	5	4	3	2	1	0	
b^i	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	64	32	16	8	4	2	1	
	1 a_6	0 a_5	0 a_4	1 a_3	0 a_2	1 a_1	1 a_0	
								75 –64 ← 1×2^6
								11 –8 ← 1×2^3
								3 –2 ← 1×2^1
								1 –1 ← 1×2^0
								0

On trouve forcément le même résultat obtenu par la méthode de divisions successives.

1.1.2.2.3. Détermination de la partie fractionnaire : $PF(N)_{10} \Longrightarrow PF(Nx)_b$

La multiplication successive par la base b permet d'avoir les différents symboles $a_{-i}\ i = 1, 2, \cdots$

Écrivons la partie fractionnaire $PF(N)_b$ sous forme polynomiale :

$$PF(N)_b = a_{-1}b^{-1} + a_{-2}b^{-2} + a_{-3}b^{-3} \cdots \cdots + a_{-m}b^{-m}$$

Faisons les multiplications successives par la base b .

$PF(N)_b \times b = \underbrace{a_{-1}}_{Partie\ entière} + \underbrace{a_{-2}b^{-1} + a_{-3}b^{-2} \cdots + a_{-m}b^{-(m-1)}}_{PF_1 : Nouvelle\ partie\ fractionnaire}$: première étape détermine le symbole a_{-1}

$PF_1 \times b = \underbrace{a_{-2}}_{Partie\ entière} + \underbrace{a_{-3}b^{-1} + a_{-4}b^{-2} \cdots + a_{-m}b^{-(m-2)}}_{PF_2 : Nouvelle\ partie\ fractionnaire}$: deuxième étape détermine le symbole a_{-2}

On répète ce procédé jusqu'à obtenir une « partie fractionnaire nulle » ou jusqu'au rang nécessaire pour satisfaire une précision donnée. Attention, une suite fractionnaire limitée dans une base n'est pas nécessairement limitée dans une autre.

Exemple : $(0{,}27)_{10} = (?\cdots?)_2$

$0{,}27 \times 2 = 0{,}54$ a_{-1}
$0{,}54 \times 2 = 1{,}08$
$0{,}08 \times 2 = 0{,}16$
$0{,}16 \times 2 = 0{,}32$
$0{,}32 \times 2 = 0{,}64$
$0{,}64 \times 2 = 1{,}28$
$0{,}28 \times 2 = 0{,}56$ a_{-7}
⋮

La suite fractionnaire n'est pas encore terminée. Elle peut être illimitée, périodique ou non.

Si on arrête comme ici le procédé à 7 chiffres après la virgule, on fait une erreur maximale de 2^{-7}, cas extrême d'une suite illimitée de 1 à partir du symbole a_{-8}

D'où :

$$(0{,}27)_{10} = \begin{pmatrix} 0,\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\ a_{-1} \qquad\qquad a_{-7} \end{pmatrix}_2$$

Rappelons que les trois méthodes de conversion ci-dessus s'appliquent quelle que soit la base b utilisée. Les exemples donnés ci-dessus ont été appliqués sur la base binaire $b = 2$

1.1.2.3. Passage de la base 2^n vers la base 2

En électronique numérique, en particulier en micro-informatique, on utilise essentiellement les bases $2^3 = 8$ (octal) et 2^4 (hexadécimal).

Le passage d'une de ces bases à la base 2 consiste simplement à traduire les chiffres octaux (ou hexadécimaux) en leurs équivalents binaires.

Chaque chiffre octal (respectivement hexadécimal) s'écrit en binaire avec 3 (respectivement 4) bits.

Exemples : $(3A9)_{16} = (?\cdots?)_2$ et $(264)_8 = (?\cdots?)_2$

$\begin{cases} 3 & A & 9 \\ 0011 & 1010 & 1001 \end{cases}$ et $\begin{cases} 2 & 6 & 4 \\ 010 & 110 & 100 \end{cases}$

D'où : $(3A9)_{16} = (001110101001)_2$ et $(264)_8 = (010110100)_2$

1.1.2.4. Passage de la base 2 vers la base 2^n

Pour passer de la base 2 à la base 8 (respectivement 16), il suffit de grouper les chiffres binaires (bits) par 3 (respectivement 4) et convertir ces sous-ensembles en leur équivalent octal (respectivement hexadécimal).

Exemples :

$(1011101,011\,01)_2 = (?\cdots?)_8$ et $(010001101101001,0110001101)_2 = (?\cdots?)_{16}$

$\begin{cases} (001\ 011\ 101,011\ 010)_2 \\ (\ 1\quad 3\quad 5,3\quad 2)_8 \end{cases}$ et $\begin{cases} (0010\ 0011\ 0110\ 1001,0110\ 0011\ 0100)_2 \\ (\ 2\quad 3\quad 6\quad 9,6\quad 3\quad 4)_{16} \end{cases}$

D'où :

$(1011101,011\,01)_2 = (135,32)_8$

et $(010001101101001,0110001101)_2 = (2369,634)_{16}$

1.1.2.5. Passage d'une base i à une base j

Les cas où i et j valent 2 et 10 ont été traités dans les paragraphes précédents.

Dans le cas général, deux situations peuvent se présenter :

- i et j sont des puissances de 2 alors on passe par la base 2 :

$(N)_i = (N')_2 = (N'')_j$

- i et j ne sont pas des puissances de 2 alors on passe par la base 10 :

$(N)_i = (N')_{10} = (N'')_j$

1.1.3. Norme IEEE 754 pour représenter des nombres à virgule flottante, cas format simple précision sur 32 bits

Un nombre simple précision sur 32 bits (1 bit de signe, 8 bits d'exposant et 23 bits de mantisse) représenté en base binaire IEEE, s'écrit (les 3 autres formats de la norme IEEE 754 sont traités de la même manière que ci-dessous):

$$(N)_{2IEEE} = \underbrace{s}\ \underbrace{E\cdots\cdots\cdots E}\ \underbrace{m\cdots\cdots\cdots\cdots\cdots\cdots\cdots m} \qquad (1.4)$$

↑	↑	↑
1 bit de signe	Exposant E sur 8 bits	Mantisse m sur 23 bits

Si $s = 1$, le nombre est négatif, autrement $(s = 0)$ le nombre est positif.

La valeur numérique du nombre peut se mettre sous la forme suivante :

$$(Nx)_{10} = V = (-1)^s \times 2^{(E-127)} \times \underbrace{(1,m)_2}$$

Équivalent décimal

Où dans le terme $\underbrace{(1,m)_2}$ le 1 est implicite. Soit encore :

$$(Nx)_{10} = V = (-1)^s \times 2^e \times \underbrace{(1,m)_2}_{\text{Équivalent décimal}} \qquad (1.5)$$

avec : $e = E - 127$ qui est l'exposant après décalage de 127

Exemples : $(-101{,}5)_{10} = (\,?\,)_{16IEEE}$ et $(47800180)_{16IEEE} = (\,?\,)_{10}$

Pour $(-101{,}5)_{10} = (\,?\,)_{16IEEE}$, on convertit d'abord $(-101{,}5)_{10}$ en base binaire $(\,?\,)_2$, puis on met le résultat sous la forme donnée par la relation (1.5), qui permet d'identifier tous les paramètres de la forme $(\,?\,)_{2IEEE}$. On écrit le résultat sous la forme $(\,?\,)_{2IEEE}$, donnée par la relation (1.4), ensuite on passe à la forme demandée $(\,?\,)_{16IEEE}$.

$(-101{,}5)_{10} = (-1100101{,}1\,)_2 = (-1{,}1001011\,)_2 \times 2^6$

Écrivons le résultat obtenu sous la forme donnée par la relation (1.5) puis identifions s, e et m puis déduisons E.

$$(-1{,}1001011\,)_2 \times 2^6 = (-1)^1 \times 2^6 \times (1,m)_2 \Rightarrow \begin{cases} s = 1 \\ e = 6 \\ m = 1001011\underbrace{0\cdots\cdots0}_{16\ z\acute{e}ros} \end{cases}$$

$E = e + 127 = 6 + 127 = 133 = (10000101)_2$

D'où, la forme $(\,?\,)_{2IEEE}$, puis $(\,?\,)_{16IEEE}$

$$(-101{,}5)_{10} = (N)_{2IEEE} = \Big(\underbrace{1}_{s}\ \underbrace{10000101}_{E}\ \underbrace{10010110000000000000000}_{m}\Big)_{2IEEE}$$

$$(-101{,}5)_{10} = (N)_{2IEEE} = \Big(\underline{1100}\ \underline{0010}\ \underline{1100}\ \underline{1011}\ \underline{0000}\ \underline{0000}\ \underline{0000}\ \underline{0000}\Big)_{2IEEE}$$
$$= (C2CB0000)_{16IEEE}$$

Pour $(47800180)_{16IEEE} = (\,?\,)_{10}$, on convertit d'abord $(47800180)_{16IEEE}$ en $(\,?\,)_2$, puis on met le résultat sous la forme $(\,?\,)_{2IEEE}$, qui permet d'identifier tous les paramètres de la forme donnée par la relation (1.5). On écrit le résultat sous la forme (1.5), ensuite on passe à la forme demandée $(\,?\,)_{10}$

$$(47800180)_{16IEEE} = (\,0100\ 0111\ 1000\ 0000\ 0000\ 0001\ 1000\ 0000)_2 =$$
$$\Big(\underbrace{0}_{s}\ \underbrace{100\ 01111}_{E}\ \underbrace{000\ 0000\ 0000\ 0001\ 1000\ 0000}_{m}\Big)_{2IEEE}$$

$$\Rightarrow \begin{cases} s = 0 \\ E = (100\ 01111)_2 = (143)_{10} \end{cases} \Rightarrow e = E - 127 = 16$$

D'où :

$$(47800180)_{16IEEE} = V = (-1)^0 \times 2^{16} \times \underbrace{\Big(1,\underline{000\ 0000\ 0000\ 0001\ 1000\ 0000}\Big)_2}_{\text{Équivalent décimal}}$$

$$V = (-1)^0 \times 2^{16}(1 + 1 \times 2^{-15} + 1 \times 2^{-16}) = 2^{16} + 2^1 + 2^0 = 65536 + 2 + 1 = 65539$$

Ou aussi par décalage de la virgule de 16 positions vers la droite, on obtient :

$$V = (1000\ 0000\ 0000\ 0001\ 1)_2 = (1 \times 2^{16} + 1 \times 2^1 + 1 \times 2^0)_{10} = 65539$$

1.2. Arithmétique binaire

Les 4 opérations sur les nombres binaires non signés sont :

Addition :

$0+0=0$	
$0+1=1$	
$1+0=1$	
$1+1=0$	et la retenue $=1$

Soustraction :

$0-0=0$	
$0-1=1$	et la retenue $=1$
$1-0=1$	
$1-1=0$	

Pour ces deux opérations « addition et soustraction », les retenues doivent être propagées, comme en décimal.

Multiplication :

$0\times0=0$
$0\times1=0$
$1\times0=0$
$1\times1=1$

Division :

$0\div0$	non défini
$0\div1=0$	
$1\div0$	non défini
$1\div1=1$	

Les opérations de multiplication et de division se font de la même façon qu'en décimal.

Exemple : $10101\times1001=?$

1 0 1 0 1 × 1 0 0 1	21 × 9
1 0 1 0 1 0 0 0 0 0 + 0 0 0 0 0 1 0 1 0 1	
1 0 1 1 1 1 0 1	189

Exemple : $10110100{,}1 \div 100110 = ?$

$1\,0\,1\,1\,0\,1\,0\,0{,}1\,0$ $\div$ $-\,1\,0\,0\,1\,1\,0$	$1\,0\,0\,1\,1\,0$	$180{,}5 \div 38$
$0\,0\,0\,1\,1\,1\,0\,0\,1$ $-\quad 1\,0\,0\,1\,1\,0$	$1\,0\,0{,}1\,1$	$= 4{,}75$
$0\,1\,0\,0\,1\,1\,0$ $-\quad 1\,0\,0\,1\,1\,0$		
$0\,0\,0\,0\,0\,0$		

1.3. Fonctions logiques binaires de base

Nous donnons ci-après la table de vérité, l'interprétation électrique et le masque binaire utilisé dans les microprocesseurs, de trois fonctions logiques de deux variables : ET (AND), OU (OR) et $OU_{ex}(XOR)$.
Ensuite, nous introduisons la fonction logique NON (NOT) et les opérations de décalage sur les mots binaires.

Fonction *ET* (*AND*)

La table de vérité de la fonction logique ET de deux variables est donnée par le tableau 1.1.

x_1	x_0		$z = x_1 \times x_0$
0	0		0
0	1		0
1	0		0
1	1		1

Tableau 1.1. Table de vérité de la fonction *ET*

L'interprétation électrique de cette fonction ET de deux variables est donnée par la figure 1.1.

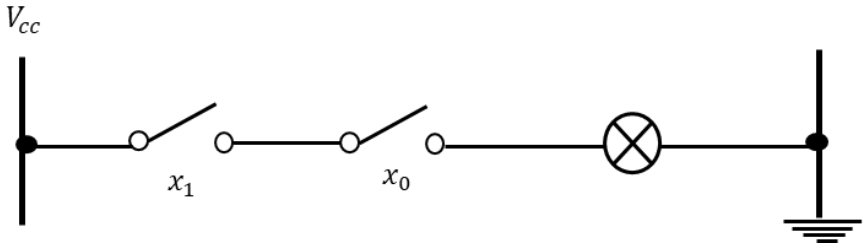


Figure 1.1. Interprétation électrique de la fonction ET de deux variables

Si l'un des interrupteurs x_1, ou x_0 est ouvert, la lampe est éteinte, elle s'allume lorsque les deux interrupteurs x_1 et x_0 sont fermés.

Pour forcer sélectivement des bits à zéro dans un mot binaire, il suffit d'utiliser un mot binaire masque ET, comme montré sur la figure 1.2, sur un mot de huit bits.

x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0 1 0 1 0 1 1 1 1	← Mot binaire ← Mot binaire masque ET
x_7 0 x_5 0 x_3 x_2 x_1 x_0	← Résultat

Figure 1.2. Mot binaire masque ET

Fonction OU (OR)

La table de vérité de la fonction logique OU de deux variables est donnée par le tableau 1.2.

x_1	x_0		$z = x_1 + x_0$
0	0		0
0	1		1
1	0		1
1	1		1

Tableau 1.2. Table de vérité de la fonction OU

L'interprétation électrique de la fonction OU de deux variables est donnée par la figure 1.3.

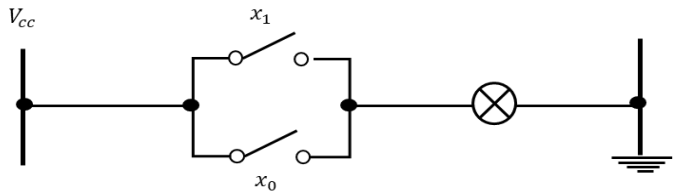


Figure 1.3. Interprétation électrique de la fonction OU de deux variables

Si les deux interrupteurs x_1 et x_0 sont ouverts, la lampe est éteinte, elle s'allume lorsqu'un des deux interrupteurs x_1 ou x_0 est fermé.

Pour forcer sélectivement des bits à un dans un mot binaire, il suffit d'utiliser un mot binaire masque OU, comme montré sur la figure 1.4, sur un mot de huit bits.

x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0 0 0 1 0 0 1 0 0	← Mot binaire ← Mot binaire masque OU
x_7 x_6 1 x_4 x_3 1 x_1 x_0	← Résultat

Figure 1.4. Mot binaire masque OU

Fonction OU_{ex}(XOR)

La table de vérité de la fonction logique OU_{ex} de deux variables (fonction différence) est donnée par le tableau 1.3.

x_1	x_0		$z = x_1 \oplus x_0$
0	0		0
0	1		1
1	0		1
1	1		0

Tableau 1.3. Table de vérité de la fonction OU_{ex}

L'interprétation électrique de la fonction OU_{ex} de deux variables est donnée par la figure 1.5.

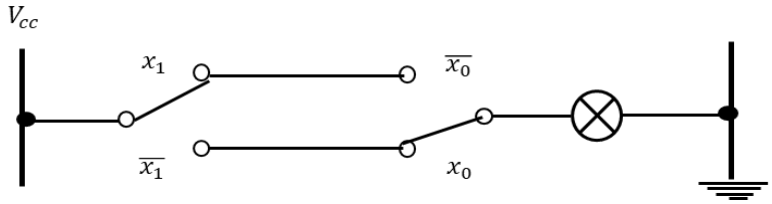


Figure 1.5. Interprétation électrique de la fonction OU_{ex} de deux variables

Si les deux interrupteurs x_1 et x_0 ont la même position fermée (x_1, x_0), ou ouvert $(\bar{x}_1, \bar{x}_0)$, la lampe est éteinte, elle s'allume lorsque les deux interrupteurs x_1 et x_0 ont des positions différentes $(\bar{x}_1, x_0)$, ou $(x_1, \bar{x}_0)$. Pour inverser sélectivement des bits dans un mot binaire, il suffit d'utiliser un mot binaire masque OU_{ex}, comme montré sur la figure 1.6, sur un mot de huit bits.

$x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0$ $0\ 1\ 0\ 0\ 0\ 1\ 0\ 0$	← Mot binaire ← Mot binaire masque OU_{ex}
$x_7\ \bar{x}_6\ x_5\ x_4\ x_3\ \bar{x}_2\ x_1\ x_0$	← Résultat

Figure 1.6. Mot binaire masque OU_{ex}

Fonction NON (NOT)

La fonction NON d'une variable binaire est définie par : $NON(x) = \bar{x}$

Décalage binaire

Nous introduisons ci-dessous quatre opérations de décalage binaire sur un mot de huit bits, soit : $(N)_2 = (0\,0\,1\,0\,1\,1\,1\,0)_2 = (46)_{10}$

- **Décalage ouvert à droite** : $introduction\,0 \rightarrow 0\,0\,1\,0\,1\,1\,1\,0 \rightarrow 0\,perdu \Longrightarrow (0\,0\,0\,1\,0\,1\,1\,1)_2 = (23)_{10}$

  Ce type de décalage a pour résultat de diviser le nombre binaire $(N)_2$ par la base $b = 2$.

  Si l'équivalent décimal du nombre $(N)_2$ est impair : $(N)_2 = (47)_{10}$ par exemple, alors le résultat de ce décalage sera $((N)_2 - 1)/2$.

- **Décalage ouvert à gauche** : $perdu\ 0 \leftarrow 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \leftarrow 0\ introduction \Longrightarrow (0\ 1\ 0\ 1\ 1\ 1\ 0\ 0)_2 = (92)_{10}$. Ce type de décalage a pour résultat de multiplier le nombre binaire $(N)_2$ par la base $b = 2$.
- **Décalage cyclique à droite** : $0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \implies 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1$: le bit LSB sortant « 0 » est réinjecté de l'autre côté en tant que bit MSB entrant.
- **Décalage cyclique à gauche** : $0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \implies 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0$: le bit MSB sortant « 0 » est réinjecté de l'autre côté en tant que bit LSB entrant.

1.4. Représentation des nombres binaires signés (entiers relatifs)

1.4.1. Représentation sous forme module plus signe

Dans le cas d'une écriture sur n bits, nous avons :

$$(N)_2 = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots\cdots + a_1 2^1 + a_0 2^0 \quad (1.6)$$

Où :

a_{n-1} (le bit de poids fort) est le bit de signe : $s = 0$, si le nombre est positif, et $s = 1$, si le nombre est négatif.

$a_{n-2}\ \cdots\cdots\ a_1\ a_0$: représente le module exprimé en binaire naturel.

La plage de valeurs décimales codables sur n bits est :

$$-(2^{n-1} - 1) \leq (N)_2 \leq (2^{n-1} - 1) \quad (1.7)$$

Exemple : $n = 4$, d'où : $-(2^3 - 1) \leq (N)_2 \leq (2^3 - 1) \implies -7 \leq N \leq 7$

Pour cet exemple, le tableau 1.4, montre le résultat des valeurs décimales codables.

Nombres positifs				Nombres négatifs		
s	$Module$	EqD		s	$Module$	EqD
0	0 0 0	$+0$		1	0 0 0	-0
0	0 0 1	$+1$		1	0 0 1	-1
0	0 1 0	$+2$		1	0 1 0	-2
0	0 1 1	$+3$		1	0 1 1	-3
0	1 0 0	$+4$		1	1 0 0	-4
0	1 0 1	$+5$		1	1 0 1	-5
0	1 1 0	$+6$		1	1 1 0	-6
0	1 1 1	$+7$		1	1 1 1	-7

Tableau 1.4. Valeurs décimales représentables en binaire signé sur $n = 4$ bits en format module plus signe

L'avantage de cette représentation en module plus signe est sa simplicité, son inconvénient est double :

- la valeur 0 a deux représentations différentes $+0 \rightarrow (0\ 0\ 0\ 0)$ et $-0 \rightarrow (1\ 0\ 0\ 0)$;
- le calcul arithmétique est impossible.

Exemple :

$(+1) + (-1) = 1\ 0\ 1\ 0 = -2$?

$(+2) + (-2) = 1\ 1\ 0\ 0 = -4$?
$(+3) + (-3) = 1\ 1\ 1\ 0 = -6$?

À cause de cette faiblesse, cette représentation n'est pas utilisée par les machines informatiques (calculateurs, ordinateurs, microprocesseurs, etc.).

1.4.2. Représentation sous forme de complément à deux $C2(N)_2$

Par définition, le complément vrai d'un nombre $(N)_b$ exprimé par n symboles (chiffres) dans la base b est :

$$CV(N)_b = b^n - (N)_b = -(N)_b \tag{1.8}$$

En effet, $b^n = 0$, car il est hors format sur n bits

$(a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots\cdots + a_1b^1 + a_0b^0)$, (interprétation calculateur).

En base binaire $b = 2$, la relation (1.8) s'écrit :

$$C2(N)_2 = 2^n - (N)_2 = -(N)_2 \tag{1.9}$$

En pratique, le calcul du complément à deux d'un nombre est effectué plus simplement que la relation (1.9), en utilisant le complément à « 1 » d'un nombre binaire en base $b = 2$, $C1(N)_2$. En effet, nous montrons ci-dessous le résultat suivant :

$$C2(N)_2 = C1(N)_2 + 1 = -(N)_2 \tag{1.10}$$

Le complément à « 1 » d'un nombre binaire, $C1(N)_2$ est obtenu en complémentant tous les bits constituant le nombre binaire, donc : $C1(N)_2 = (\bar{N})_2$.

L'addition $(N)_2 + (\bar{N})_2$ donne le résultat suivant :

$(N)_2$	→	$a_{n-1}\ a_{n-2}\ \cdots\cdots\ a_0$
$+\ (\bar{N})_2$	→	$\bar{a}_{n-1}\ \bar{a}_{n-2}\ \cdots\cdots\ \bar{a}_0$
$2^n - 1 = -1$	=	$1\quad 1\quad \cdots\cdots 1$

D'où : $(N)_2 + C1(N)_2 = 2^n - 1 = -1 \implies -(N)_2 = C1(N)_2 + 1 = C2(N)_2$

La plage de valeurs décimales codables sur n bits en $C2(N)_2$ est :

$$-2^{n-1} \leq (N)_2 \leq 2^{n-1} - 1 \tag{1.11}$$

Cette plage est dissymétrique à cause du zéro qui n'apparaît que parmi les nombres positifs (codage unique du zéro)

Exemple : $n = 4$, d'où : $-2^3 \leq (N)_2 \leq 2^3 - 1 \implies -8 \leq N \leq 7$

Pour cet exemple, le tableau 1.5, montre le résultat des valeurs décimales codables.

Nombres positifs				$C1(N)_2$		Nombres négatifs $C2(N)_2 = C1(N)_2 + 1 = -(N)_2$		
s		EqD				*s*		EqD
0	**0 0 0**	**+0**		1 1 1 1		**0**	**0 0 0**	**+0**
0	0 0 1	+1		1 1 1 0		1	1 1 1	−1
0	0 1 0	+2		1 1 0 1		1	1 1 0	−2
0	0 1 1	+3		1 1 0 0		1	1 0 1	−3
0	1 0 0	+4		1 0 1 1		1	1 0 0	−4

0	1 0 1	+5		1 0 1 0		1	0 1 1	−5
0	1 1 0	+6		1 0 0 1		1	0 1 0	−6
0	1 1 1	+7		1 0 0 0		1	0 0 1	−7

Tableau 1.5. Valeurs décimales représentables en binaire signé sur $n = 4$ bits en format complément à deux

On note que le code 1000 n'apparaît pas. C'est normal puisque ce codage correspond à la valeur -8 dont l'opposé $+8$ ne peut s'écrire sur 4 bits en format $C2(N)_2$.

Le code 1000 est cependant tout à fait valide. On peut s'en convaincre en effectuant une opération (sans débordement, faute de quoi le résultat ne pourrait être que faux) qui l'utilise, comme dans l'exemple ci-dessous :

1 0 0 0	$-(8)$
0 1 0 1	$+(5)$
1 1 0 1	$= -(3)$ en $C2(N)_2$

L'avantage de la représentation des nombres entiers en complément à deux par rapport à la représentation module plus signe est :

- le zéro n'a qu'une seule représentation « **0 0 0 0** » quel que soit le signe du zéro ;
- le calcul arithmétique se fait correctement. Si on fait l'addition de $+3$ avec -3 le résultat est 0 :

3	0 0 1 1
+	
−3	1 1 0 1
carry in →	1 1 1
carry out → 1	0 0 0 0 = 0 OK

Remarque importante : dans les machines informatiques, l'opération de soustraction de deux nombres binaires est remplacée par l'opération d'addition du premier nombre avec le complément à deux du second nombre :

$$N = N_1 - N_2 = N_1 + C2(N_2) \tag{1.12}$$

Règle : Le résultat de l'addition de deux nombres, complément à deux de signes opposés est toujours correct. Par contre le résultat de l'addition de deux nombres, complément à deux de même signe, peut être correct ou non, cela dépend si le résultat présente un débordement (overflow) ou non.

Pour savoir si le résultat est correct ou non, il suffit d'opérer un XOR logique entre le bit de retenue sur le bit de signe « *carry in into the bit of sign* » et le bit retenu sortant du bit de signe « *carry out going out of the bit sign* ». Le résultat V du XOR indique s'il y débordement ou non, comme montré ci-dessous.

$V = XOR(carry\ in, carry\ out\ )$

$$\Longrightarrow \text{Si} \begin{cases} V = 1 & \Longrightarrow KO, \text{ débordement } (overflow) \\ V = 0 & \Longrightarrow OK, \text{ pas de débordement } (no\ overflow) \end{cases} \quad (1.13)$$

Exemple :
Si on fait l'addition de -6 avec -5 le résultat est -11, et on aura un débordement

-5	1 0 1 1
$+$	
-6	1 0 1 0
$carry\ in \rightarrow$	0 1
$carry\ out \rightarrow 1$	$0\ 1\ 0\ 1 = 5$ KO car $V = 1$

Si on fait l'addition de -4 avec -3 le résultat est -7, et on n'aura pas un débordement

-4	1 1 0 0
$+$	
-3	1 1 0 1
$carry\ in \rightarrow$	1
$carry\ out \rightarrow 1$	$1\ 0\ 0\ 1 = -7$ OK car $V = 0$

Ici, le *carry out* est ignoré car le résultat est correct.

1.5. Codage de l'information

Définition générale : Le codage de l'information est une opération de correspondance biunivoque entre les éléments de deux ensembles, comme montre la figure 1.7.

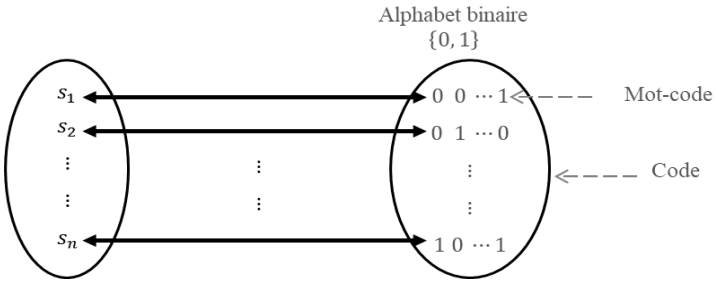


Figure 1.7. Codage de l'information

Le but de cette opération dépend du domaine d'utilisation :

En télécommunications, nous avons :

- le codage de source pour canaux sans perturbations permet d'adapter le débit binaire de la source à la capacité du canal de transmission ;
- le codage de canal pour canaux à perturbations sert d'une part à se protéger contre les erreurs de transmission par l'introduction d'une redondance voulue, et d'autre part à faire correspondre un signal électrique (à deux niveaux 0 et 5 V par exemple) à chacun des bits (symboles) d'information ;
- le codage peut être aussi utilisé dans une optique de chiffrement de l'information.

En électronique numérique, l'opération de codage sert à réduire le nombre des variables à traiter. Par exemple, avec 8 bits seulement on peut coder l'ensemble des touches du clavier d'un ordinateur.

1.5.1. Codes couramment utilisés en électronique numérique

Nous donnons ci-après les différents codes couramment utilisés en électronique numérique.

1.5.1.1. Codes décimaux utilisés pour la représentation des chiffres du système décimal

Pour représenter les chiffres décimaux de zéro à neuf, il suffit d'utiliser un mot binaire de quatre bits. Trois codes sont généralement utilisés : le code DCB (Décimal Codé Binaire), (BCD : *Binary Coded Decimal*), le code DCB+3, (DCB excess3) et le code Aïken. Ce sont des codes pondérés, comme montré par la table de vérité du tableau 1.6.

$(N)_{10}$	DCB	DCB+3	Aïken
	8 4 2 1	8 4 2 1	2 4 2 1
0	0 0 0 0	0 0 1 1	0 0 0 0
1	0 0 0 1	0 1 0 0	0 0 0 1
2	0 0 1 0	0 1 0 1	0 0 1 0
3	0 0 1 1	0 1 1 0	0 0 1 1
4	0 1 0 0	0 1 1 1	0 1 0 0
5	0 1 0 1	1 0 0 0	1 0 1 1
6	0 1 1 0	1 0 0 1	1 1 0 0
7	0 1 1 1	1 0 1 0	1 1 0 1
8	1 0 0 0	1 0 1 1	1 1 1 0
9	1 0 0 1	1 1 0 0	1 1 1 1

Tableau 1.6. Codes décimaux généralement utilisés

Les codes DCB+3 et Aïken sont des codes auto-complémentés, cela veut dire :

- l'axe médian est un axe de symétrie, donc de part et d'autre de cet axe les mots-codes sont complémentés ;
- le complément à 1 d'un mot-code représente le complément à 9 dans l'ensemble source.

Exemple : en DCB+3, le $C1(6) = C1(1001) = 0110 = 3 = C9(6)$.

1.5.1.2. Code de Gray (code à distance unité) ou code binaire réfléchi

Le code de Gray est un code continu (les combinaisons consécutives sont adjacentes) et strict (la première combinaison et la dernière sont aussi adjacentes). La figure 1.8 montre comment construire un code de Gray à $(n+1)$ variables binaires à partir de code d'un Gray à n variables binaires.

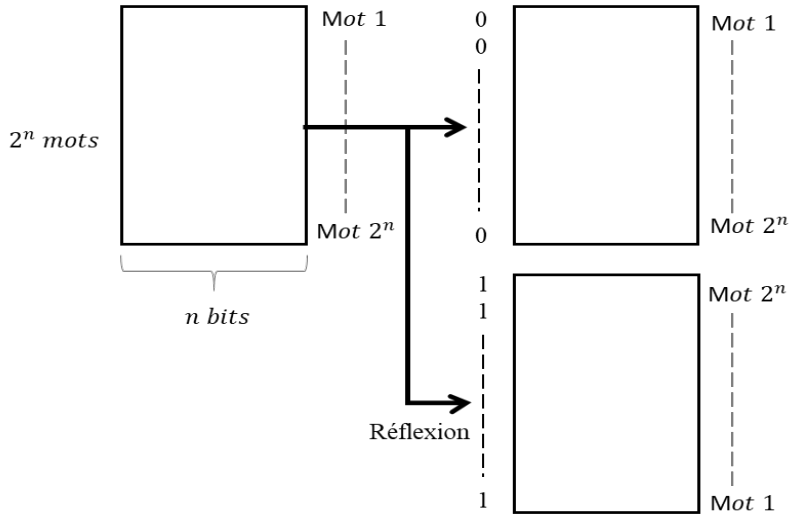


Figure 1.8. Construction du code de Gray à $(n + 1)$ variables binaires à partir d'un code de Gray à n variables binaires.

Sur la figure 1.9, nous montrons la construction du code de Gray à 2, 3 et 4 variables binaires par étapes successives à partir du code élémentaire à une variable binaire. Ainsi nous pouvons construire un code de Gray pour un nombre de variables binaires donné.

0	0 0	0 0 0	0 0 0 0
1	0 1	0 0 1	0 0 0 1
	------	0 1 1	0 0 1 1
	1 1	0 1 0	0 0 1 0
	1 0	------	0 1 1 0
		1 1 0	0 1 1 1
		1 1 1	0 1 0 1
		1 0 1	0 1 0 0
		1 0 0	---------
			1 1 0 0
			1 1 0 1
			1 1 1 1
			1 1 1 0
			1 0 1 0
			1 0 1 1
			1 0 0 1
			1 0 0 0

Figure 1.9. Construction du code de Gray à 2, 3 et 4 variables binaires par étapes successives

Remarque importante : intérêt de l'adjacence des codes

Avec un code non adjacent, le passage d'une combinaison à la suivante peut entraîner la commutation de plusieurs bits (sorties d'un compteur binaire par exemple). Par exemple, le passage de la valeur 7 à la valeur 8 à la sortie d'un compteur 4 bits, entraîne la commutation de tous les bits car on passe de la combinaison 0111 à la combinaison 1000.

La commutation des bits n'est pas instantanée, d'où possibilité de passage par des états intermédiaires :
$0111 \rightarrow 0110 \rightarrow 0100 \rightarrow 0000 \rightarrow 1000$.

Ce phénomène porte le nom d'aléas de commutation (rien à voir avec un « parasite »).

La séquence ci-dessus est donnée à titre d'exemple, mais d'autres sont tout aussi envisageables, par exemple : $0111 \rightarrow 0101 \rightarrow 1101 \rightarrow 1100 \rightarrow 1000$.

Conséquences éventuelles

- suivant l'instant d'observation, le résultat change ;
- si un circuit, placé en aval, décode la valeur 0110 (ou 0100, ou 0000) il peut se produire une détection erronée de cette valeur de manière transitoire mais potentiellement source de dysfonctionnement.

Avec le code Gray, ce problème disparaît naturellement puisque seulement 1 bit change d'état entre 2 combinaisons successives.

1.5.1.3. Code ASCII : *American Standard Code for Information Interchange*

A l'origine, ce code utilisant 7 bits associe une valeur numérique à chacun des caractères (latins) alphanumériques affichables et/ou imprimables. Une version étendue sur 8 bits est apparue ensuite afin de coder les caractères accentués et bon nombre de caractères dits « spéciaux » (caractères de contrôle, de mise en forme, etc.).

Le code ASCII est utilisé dans les réseaux informatiques assurant les connexions entre des ordinateurs et des organes périphériques (claviers, écrans, imprimantes, etc.). Le tableau 1.7, donne la table du code ASCII 7 bits.

b_7 b_6 b_5 → / Bits					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Tableau 1.7. Table du code ASCII 7 bits

Exemples :

- le code de la lettre K est $(4B)_{16}$ soit $(1001011)_2$
- le code $(07)_{16} = (0000111)_2 =$ BEL permet d'émettre un son (*bell* en anglais)

1.6. Fonctions logiques – Algèbre de Boole (1850)

1.6.1. Variable logique ou booléenne

Une variable logique ou booléenne ne peut prendre que deux états : vrai/faux ; état haut/état bas ; niveau 1/niveau 0 $\Leftrightarrow$ V_{cc} ou V_{dd} / Gnd ou V_{ss}.

Avec n variables logiques indépendantes : $x_{n-1}\, x_{n-2} \cdots \cdots x_0$, nous avons : 2^n combinaisons différentes et 2^{2^n} fonctions logiques différentes. Le tableau 1.8 donne un exemple de résultat pour n égal à 1, 2, 3 et 4.

Nombre de variables logiques : n	1	2	3	4
Nombre de combinaisons différentes : 2^n	2	4	8	16
Nombre de fonctions logiques différentes : 2^{2^n}	4	16	256	65 536

Tableau 1.8. Nombre de variables logiques, nombre de combinaisons et nombre de fonctions logiques différentes

Nous donnons dans le tableau 1.9 toutes les fonctions logiques de deux variables $F_{i=0,\cdots,15}(x_1, x_0)$ (telles que la valeur décimale de l'indice i exprime la valeur de F_i en binaire naturel).

$x_1\ x_0$		F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0 0		0	1	0	1	0	1	0	1
0 1		0	0	1	1	0	0	1	1
1 0		0	0	0	0	1	1	1	1
1 1		0	0	0	0	0	0	0	0
		0	NOR	$x_1 < x_0$	$\bar{x}_1$	$x_1 > x_0$	$\bar{x}_0$	XOR	$NAND$

$x_1\ x_0$		F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0 0		0	1	0	1	0	1	0	1
0 1		0	0	1	1	0	0	1	1
1 0		0	0	0	0	1	1	1	1
1 1		1	1	1	1	1	1	1	1
		AND	$\overline{XOR}$	x_0	$x_1 \leq x_0$	x_1	$x_1 \geq x_0$	OR	1

Tableau 1.9. Fonctions logiques de deux variables connues

Nous observons qu'avec seulement deux variables logiques nous obtenons toutes les fonctions logiques connues. Les fonctions grisées $F_0, F_3, F_5, F_{10}, F_{12}$ et F_{15} ne sont pas utiles.

Les fonctions $F_8, F_9, \cdots, F_{15}$ sont obtenues par négation des fonctions $F_7, F_6, \cdots, F_0$ respectivement.

Notons qu'avec les fonctions logiques AND (ET), OR (OU) et NOT (NON) nous pouvons réaliser toutes les autres fonctions logiques. De même, avec seulement des $NAND$ ou avec seulement des NOR.

1.6.2. Normes CEI (internationale) et MIL (américaine) pour la représentation des portes logiques de deux variables

Nous donnons dans la figure 1.10 les deux normes CEI (internationale) et MIL (américaine) pour la représentation des portes logiques de deux variables binaires.

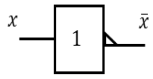
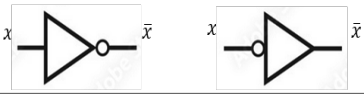
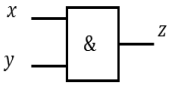
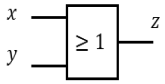
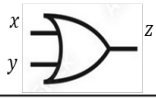
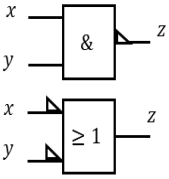
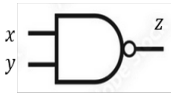
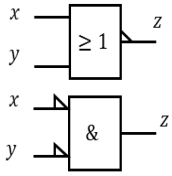
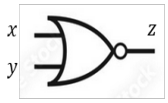
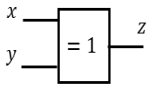
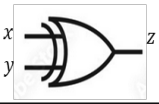
Fonctions logiques	Norme CEI (internationale)	Norme MIL (américaine)	Circuits intégrés TTL
$\bar{x}$ NOT			74*LS*04
$z = x.y$ $z = x \times y$ AND			74*LS*08
$z = x + y$ OR			74*LS*32
$z = \overline{x \times y}$ $NAND$ $z = \bar{x} + \bar{y}$			74*LS*00
$z = \overline{x + y}$ NOR $z = \bar{x} \times \bar{y}$			74*LS*02 74*LS*28
$z = x \oplus y$ XOR			74*LS*86 74*LS*136

Figure 1.10. Normes CEI (internationale) et MIL (américaine) pour la représentation des portes logiques

1.6.3. Algèbre de Boole

Algèbre de deux états : $\begin{cases} 1 & \text{vrai} \\ 0 & \text{faux} \end{cases}$

1.6.3.1. Lois fondamentales

Nous donnons ci-dessous les lois fondamentales utilisées dans l'algèbre de Boole :

a) Commutativité : $\begin{cases} \times \Longrightarrow x \times y = y \times x \\ + \Longrightarrow x + y = y + x \end{cases}$

b) Idempotence : $\begin{cases} \times \Longrightarrow x \times x = x \\ + \Longrightarrow x + x = x \end{cases}$

c) Élément neutre : $\begin{cases} \times \Longrightarrow x \times 1 = x \\ + \Longrightarrow x + 0 = x \end{cases}$

d) Élément invariant : $\begin{cases} \times \Longrightarrow x \times 0 = 0 \\ + \Longrightarrow x + 1 = 1 \end{cases}$

e) Complémentation : $\begin{cases} \times \Longrightarrow x \times \bar{x} = 0 \\ + \Longrightarrow x + \bar{x} = 1 \end{cases}$

f) Associativité : $\begin{cases} \times \Longrightarrow x \times y \times z = x \times (y \times z) = (x \times y) \times z \\ + \Longrightarrow x + y + z = x + (y + z) = (x + y) + z \end{cases}$

g) Distributivité : $\begin{cases} \times/+ \Longrightarrow x \times (y + z) = x \times y + x \times z \\ +/\times \Longrightarrow x + (y \times z) = (x + y) \times (x + z) \end{cases}$

h) Réversibilité de NON : $\bar{\bar{x}} = x$.

1.6.3.2. Relations de base

Ci-après, nous donnons les relations de base de l'algèbre de Boole utilisées pour la simplification des variables logiques.

a) $x \times y + x \times \bar{y} = x$

b) $(x + y) \times (x + \bar{y}) = x$

c) $x + x \times y = x$

d) $x \times (x + y) = x$

e) $x + \bar{x} \times y = x + y$

f) $x \times (\bar{x} + y) = x \times y$

g) $x \times y + \bar{x} \times z + y \times z = x \times y + \bar{x} \times z$

h) $(x + y) \times (\bar{x} + z) \times (y + z) = (x + y) \times (\bar{x} + z)$

i) $x \times y + x \times \bar{y} \times z = x \times y + x \times z$

j) $(x + y) \times (x + \bar{y} + z) = (x + y) \times (x + z)$.

1.6.3.3. Théorèmes de De Morgan et de Shannon

Nous donnons ci-dessous les deux théorèmes de De Morgan (détermination du complément d'une expression logique) et le théorème de Shannon qui généralise les théorèmes de De Morgan.

Premier théorème de De Morgan :

Le complément des produits de variables logiques est égal à la somme des compléments de chacune des variables logiques.

$$\overline{x_0 \times x_1 \times \cdots\cdots \times x_{n-1}} = \bar{x}_0 + \bar{x}_1 + \cdots\cdots + \bar{x}_{n-1} \tag{1.14}$$

Deuxième théorème de De Morgan :

$$\overline{x_0 + x_1 + \cdots\cdots + x_{n-1}} = \bar{x}_0 \times \bar{x}_1 \times \cdots\cdots \times \bar{x}_{n-1} \tag{1.15}$$

Le complément des sommes de variables logiques est égal au produit des compléments de chacune des variables logiques.

Théorème de Shannon :

$$\overline{F(x, y, z, +, \times)} = F(\bar{x}, \bar{y}, \bar{z}, \times, +) \tag{1.16}$$

Le complément d'une expression logique quelconque s'obtient en complémentant les variables et en permutant les opérateurs $\times$ et $+$

Exemple : complémentez la fonction : $F = (x + y) \times (x + z) + (\bar{x} + y)$

En utilisant les théorèmes de De Morgan, nous avons par étape :

$$\bar{F} = \overline{(x + y) \times (x + z)} \times \overline{(\bar{x} + y)} = \left(\overline{(x + y)} + \overline{(x + z)}\right) \times (x \times \bar{y})$$
$$= \left((\bar{x} \times \bar{y}) + (\bar{x} \times \bar{z})\right) \times (x \times \bar{y})$$

En utilisant le théorème de Shannon, nous avons directement :

$\bar{F} = \left((\bar{x} \times \bar{y}) + (\bar{x} \times \bar{z})\right) \times (x \times \bar{y})$.

1.7. Représentation et simplification des fonctions logiques combinatoires

Une fonction logique est dite combinatoire lorsque l'état des sorties à l'instant t est uniquement défini par l'état des entrées à l'instant $t - \Delta t$: $S(t) = F[E(t - \Delta t)]$, quelque soit l'instant.

Nous verrons dans le chapitre 3, que dans le cas des systèmes séquentiels synchrones, l'état des sorties dépend de l'état des entrées mais aussi de l'état présent du système mémorisé par $Q(t)$.

1.7.1. Représentation d'une fonction logique

Une fonction logique se présente de plusieurs façons : écriture algébrique, table de vérité, table de Karnaugh, formes canoniques des expressions algébriques et logigramme.

1.7.1.1. Écriture algébrique

Exemple, soit la fonction : $F(c, b, a) = \bar{c}\,\bar{b}\,a + \bar{c}\,b\,\bar{a} + c\,\bar{b}\,\bar{a} + c\,b\,a$.

L'équation est ici écrite sous la première forme canonique : somme de termes (OU logiques) de produits de variables (ET logiques). Les termes produits de variables logiques portent le nom de termes-produits (*product-terms* ou *p-terms* en anglais).

1.7.1.2. Table de vérité

Dans le tableau 1.10, nous donnons la table de vérité de la fonction $F(c, b, a)$ définie plus haut.

Équivalent Décimal (EqD)	Entrée				Sortie
	c	b	a		$F(c, b, a)$
0	0	0	0		0
1	0	0	1		1
2	0	1	0		1
3	0	1	1		0
4	1	0	0		1
5	1	0	1		0
6	1	1	0		0
7	1	1	1		1

Tableau 1.10. Table de vérité de la fonction $F(c, b, a)$

La table de vérité d'une fonction $F(n \text{ variables})$ est formée de $n + 1$ colonnes et 2^n lignes. Pour $n > 5$, il est difficile d'exploiter visuellement la table de vérité. Une table à double entrées aussi carrée que possible est dans ce cas préférable, c'est la table de Karnaugh.

1.7.1.3. Table de Karnaugh

La table de Karnaugh contient les mêmes données que la table de vérité. Le principe est de construire une table aussi carrée que possible, et de coder les variables logiques des lignes et des colonnes en code de Gray. Ceci engendre les propriétés de l'adjacence et de la symétrie cylindrique des combinaisons. L'intérêt de cette disposition est que, dès lors, les regroupements de termes logiques adjacents peuvent être détectés visuellement ; en observant dans la table, la position des « 1 » pour la fonction F ou la position des « 0 » pour la fonction $\bar{F}$. L'identification des termes logiques adjacents permet ensuite de procéder très facilement à des simplifications logiques comme cela sera indiqué au paragraphe 1.7.2. Dans le cas de la fonction $F(c, b, a)$, la table de Karnaugh correspondante est donnée par le tableau 1.11.

$c \setminus b\ a$	0 0	0 1	1 1	1 0
0	0 0	1 1	0 3	1 2

1	1 4	0 5	1 7	0 6

Tableau 1.11. Table de Karnaugh de la fonction $F(c, b, a)$

Dans chaque case, en plus de la valeur de la fonction $F(c, b, a)$, « 1 » ou « 0 », nous avons mis son numéro en équivalent décimal, cette numérotation est facultative et non nécessaire.

1.7.1.4. Formes canoniques

Nous donnons dans le tableau 1.12, les quatre formes canoniques de l'écriture algébrique d'une fonction logique.

	Nom de la forme	Expression algébrique	Combinaisons utiles	Forme des variables d'entrée	Résultat
1ère forme	AND	$\sum\times$	$F = 1$	Normales	F
2ème forme	OR	$\prod+$	$F = 0$	Complémentées	F
3ème forme	$NAND$	Déduite de la 1ère forme canonique (par le théorème de De Morgan)			
4ème forme	NOR	Déduite de la 2ème forme canonique (par le théorème de De Morgan)			

Tableau 1.12. Formes canoniques de l'écriture algébrique d'une fonction logique

Appliquons l'écriture de ces quatre formes canoniques sur l'exemple de la fonction $F(c, b, a) = \bar{c}\,\bar{b}\,a + \bar{c}\,b\,\bar{a} + c\,\bar{b}\,\bar{a} + c\,b\,a$, en utilisant sa table de vérité.

1ère forme canonique : somme de produits (appelés min-termes) ⟹ des portes AND reliées à une porte OU

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

2ème forme canonique : produit de sommes (appelées max-termes) ⟹ des portes OR reliées à une porte AND

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

3ème forme : double complémentation de la 1ère forme ⟹ utilisation de portes $NAND$

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

4ème forme canonique : double complémentation de la 2ème forme ⟹ utilisation de portes NOR

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

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

1.7.1.5. Logigramme

Soit la fonction logique suivante : $F(d, c,\ b,\ a) = d\ \bar{c} + b\ a$.

Le logigramme de cette fonction logique est représenté par la figure 1.11, utilisant des circuits *SSI* (*Small Scale Integration* : le nombre des transistors est inférieur à 10).

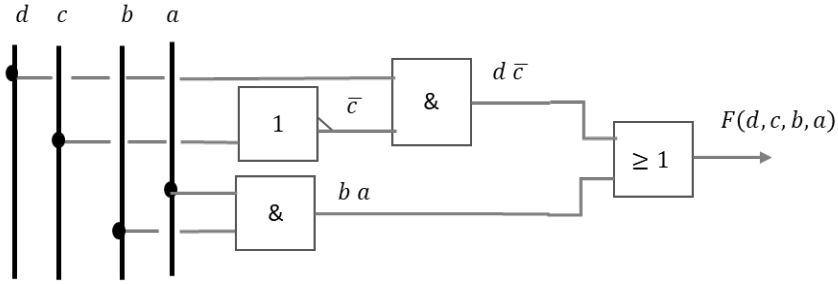


Figure 1.11. Logigramme de la fonction : $F(d, c,\ b,\ a) = d\ \bar{c} + b\ a$

1.7.2. Simplification (ou réduction) des équations logiques

L'objectif est d'obtenir un nombre minimal de termes logiques et de variables logiques dans chaque terme. Pour cela on peut utiliser :

- la méthode algébrique ;
- la méthode graphique lorsque le nombre de variables logiques de la fonction est ≤ 6 : utilise la table de Karnaugh ;
- les méthodes programmables lorsque le nombre de variables logiques est quelconque : utilisent un moyen de calcul informatique ;

Dans le cadre de ce cours nous utiliserons essentiellement la méthode de simplification par la table de Karnaugh. La méthode algébrique est utilisée pour des cas de figures simples.

1.7.2.1. Simplification par la méthode algébrique

Exemple : simplifiez la fonction algébrique suivante :

$$F(c,b,a) = (b+a) \times \left(c + \bar{b}\right) \times (c + \bar{a})$$

$$F(c,b,a) = \left(c\ b + \underbrace{b\ \bar{b}}_{0} + c\ a + \bar{b}\ a\right) \times (c + \bar{a})$$

$$= \underbrace{c\ c}_{c}\ b\ + c\ b\ \bar{a} + \underbrace{c\ c}_{c}\ a\ + c\ \underbrace{a\ \bar{a}}_{0} + c\ \bar{b}\ a + \bar{b}\ \underbrace{a\ \bar{a}}_{0}$$

$$F(c,b,a) = c\ b + c\ b\ \bar{a} + c\ a + \ c\ \bar{b}\ a = c\ b(1+\bar{a}) + c\ a\left(1+\bar{b}\right) = c\ b + c\ a.$$

1.7.2.2. Simplification par la table de Karnaugh

Rappelons que la numérotation des lignes et des colonnes est faite en code de Gray, d'où adjacence des combinaisons qui suivent et donc possibilité de simplification. De plus, les

lignes et les colonnes extrêmes de la table de Karnaugh sont adjacentes. On parle de symétrie cylindrique.

Un groupement de 2^k cellules adjacentes contenant la même valeur logique (« 1 » pour la simplification d'une fonction logique F, « 0 » pour la simplification fonction logique $\bar{F}$) correspond à une somme de 2^k termes logiques qui se réduisent à un seul dans lequel k variables logiques n'apparaissent pas (voir illustrations plus loin).

L'exploitation du tableau consiste à mettre en évidence les plus gros groupements de 2^k (en pratique : 2, 4, 8, 16, etc.) cellules adjacentes portant la même valeur logique.

Remarque : une même cellule peut participer à plusieurs groupements (être utilisée plusieurs fois) car cela revient à ajouter plusieurs fois le même terme-produit dans l'équation finale, ce qui ne modifie pas le résultat. En effet, en logique booléenne, $a + a + \cdots a = a$.

Exemple 1 : soit la fonction $F(d, c, b, a)$ définie par sa table de vérité donnée par le tableau 1.13.

$d\ c \setminus b\ a$	0 0	0 1	1 1	1 0
0 0	1	0	1	1
0 1	1	0	1	0
1 1	X	X	X	X
1 0	1	1	X	X

Tableau 1.13. Table de vérité de la fonction $F(d, c, b, a)$

Nous utilisons ici la première forme canonique pour la simplification de la fonction $F(d, c, b, a)$, les cases $F = \text{X}$ (indifférent), peuvent être utilisées pour $\text{X} = 1$, ou $\text{X} = 0$. Ici, nous avons pris $\text{X} = 1$.

Après simplifications nous obtenons :

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

Si on groupe les zéros au lieu des uns dans la table de Karnaugh, nous aurons la fonction $\bar{F}(d, c, b, a)$:

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

Exemple 2 : soit la fonction $F(e, d, c, b, a)$ définie par sa table de vérité donnée par le tableau 1.14.

$e\ d \setminus c\ b\ a$	0 0 0	0 0 1	0 1 1	0 1 0	1 1 0	1 1 1	1 0 1	1 0 0
0 0	1	1	0	0	0	0	0	0
0 1	0	1	1	0	0	1	1	0
1 1	0	1	1	1	1	1	0	0
1 0	1	1	0	0	0	0	1	1

Tableau 1.14. Table de vérité de la fonction $F(e,d,c,b,a)$

Après simplifications nous obtenons :
$F(e,d,c,b,a) = e\ d\ b + d\ \bar{c}\ a + \bar{d}\ \bar{c}\ \bar{b} + e\ \bar{d}\ \bar{b} + \bar{e}\ d\ a$.

L'exploitation d'un tel tableau n'est pas simple car les lignes/colonnes adjacentes sur le plan logique ne sont pas forcément voisines dans le tableau. Pour contourner cette difficulté, on peut construire deux tableaux à quatre variables d'entrées, chacun correspondant à chacun des deux états logiques de la cinquième entrée (e.g. $e = 1$, et $e = 0$). Cette stratégie a toutefois elle aussi ses inconvénients (elle est lourde).

1.8. Opérateurs combinatoires standards

Dans ce paragraphe nous présentons les principales fonctions combinatoires qui donnaient lieu par le passé à la fabrication de circuits (opérateurs) spécifiques, dans diverses technologies (voir chapitre 2). Ces opérateurs appartiennent à la catégorie des circuits *MSI (Medium Scale Integration* : le nombre des transistors est compris entre 10 et 100). Nous abordons les opérateurs suivants :

- les circuits d'aiguillage : multiplexeurs / démultiplexeurs ;
- les circuits codage/transcodage : codeurs / décodeurs ; transcodeurs ;
- les circuits arithmétiques : comparateurs ; additionneurs/soustracteurs ; unité arithmétique et logique (traités en travaux dirigés TD et travaux pratiques TP).

1.8.1. Multiplexeurs / Démultiplexeurs

1.8.1.1. Multiplexeur $2^n \rightarrow 1$

Un multiplexeur $2^n \rightarrow 1$ est un circuit qui possède : $\begin{cases} 2^n \text{ entrées d'information} \\ n \text{ entrées commande ou adresse} \\ 1 \text{ sortie } s \end{cases}$

Son symbole logique est donné par la figure 1.12.

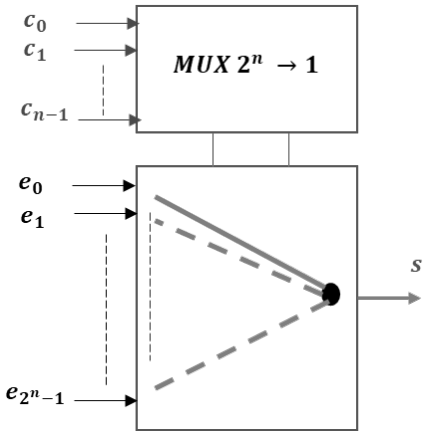


Figure 1.12. Symbole logique du multiplexeur $2^n \rightarrow 1$

Fonctionnement : selon la valeur des entrées adresse, par exemple k : $c_{n-1}\, c_{n-2} \cdots c_0 = k$, le multiplexeur sélectionne l'entrée e_k parmi l'ensemble des entrées et l'aiguille vers la sortie s.

La sortie s est donnée par l'équation suivante :

$$s = \sum_{i=0}^{2^n-1} [c_1 c_0 = i]\, e_i \tag{1.17}$$

Prenons le cas du multiplexeur $2^2 \rightarrow 1$, son symbole logique est donné par la figure 1.13.

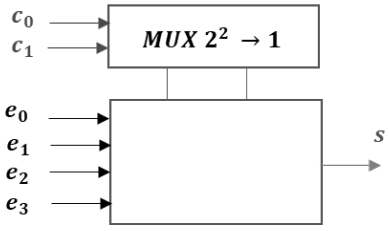


Figure 1.13. Symbole logique du multiplexeur $2^2 \rightarrow 1$

L'équation de sa sortie s s'écrit :

$$s = \sum_{i=0}^{3} [c_1 c_0 = i]\, e_i = [\bar{c}_1 \bar{c}_0\, e_0] + [\bar{c}_1 c_0\, e_1] + [c_1 \bar{c}_0\, e_2] + [c_1 c_0] e_3$$

Et sa table de vérité est donnée par le tableau 1.15.

c_1	c_0		s
0	0		e_0
0	1		e_1

1	0		e_2
1	1		e_3

Tableau 1.15. Table de vérité du multiplexeur $2^2 \rightarrow 1$

1.8.1.2. Démultiplexeur $\mathbf{1 \rightarrow 2^n}$

Un démultiplexeur $1 \rightarrow 2^n$ est un circuit qui possède :

$$\begin{cases} 1 \text{ entrée d'information} \\ n \text{ entrées commande ou adresse} \\ 2^n \text{ sorties} \end{cases}$$

Son symbole logique est donné par la figure 1.14.

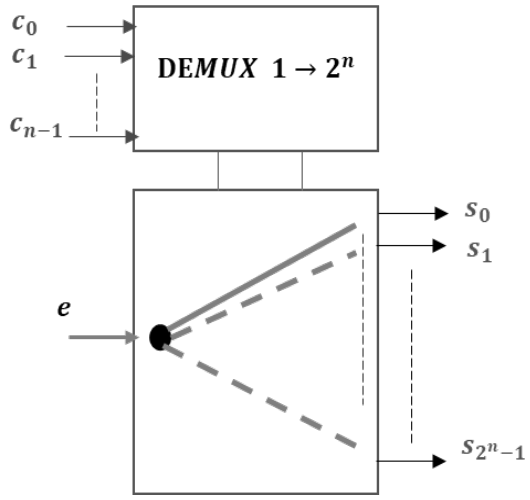


Figure 1.14. Symbole logique du démultiplexeur $1 \rightarrow 2^n$

Fonctionnement : selon la valeur des entrées adresse, par exemple k : $c_{n-1}\, c_{n-2} \cdots c_0 = k$, le démultiplexeur aiguille l'unique entrée e vers la sortie s_k choisie parmi 2^n sorties.

Le démultiplexeur réalise l'opération inverse du multiplexeur. Il fonctionne comme un décodeur binaire.

1.8.1.3. Applications des multiplexeurs / démultiplexeurs

Le multiplexeur est un opérateur à usage multiple. En effet, en plus de la fonctionnalité d'aiguillage de l'information, il peut également être utilisé en tant que générateur de fonctions (implantation de n'importe quelle fonction combinatoire). Aussi, le multiplexeur est un élément essentiel pour la réalisation de conversion parallèle-série d'un mot-binaire n-bit, en vue de sa transmission. Après sa transmission série, un démultiplexeur est nécessaire pour réaliser la conversion série-parallèle du mot-binaire. La conversion parallèle-série-parallèle est traitée dans le chapitre 8, avec les travaux pratiques.

1.8.1.3.1. Générateur de fonctions

Théorème de Shannon : toute fonction logique de deux variables $F(b,\ a)$ peut s'écrire sous la forme :

$F(b,\ a) = \bar{b}\ \bar{a}\ F(0,\ 0) + \bar{b}\ a\ F(0,\ 1) + b\ \bar{a}\ F(1,\ 0) + b\ a\ F(1,\ 1).$

Nous savons aussi que la sortie s d'un multiplexeur $2^2\ \rightarrow 1$ s'écrit :

$s = [\bar{c}_1\bar{c}_0\ e_0] + [\bar{c}_1 c_0\ e_1] + [c_1\bar{c}_0\ e_2] + [c_1 c_0]e_3$

Par identification, la fonction $F(b,\ a)$ est directement implantable par ce multiplexeur, pour cela il suffit d'associer les éléments suivants, entre la fonction $F(b,\ a)$ et le multiplexeur :

$s\ \leftrightarrow F(b,\ a)$; $\ c_1c_0\ \leftrightarrow b\ a$; $\ e_i\ \leftrightarrow F(b, a = i)$;

La figure 1.15, montre l'implantation de la fonction $F(b,\ a)$ par un multiplexeur $2^2\ \rightarrow 1$

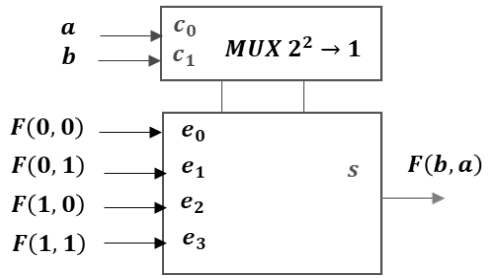


Figure 1.15. Implantation de la fonction $F(b,\ a)$ par un multiplexeur $2^2\ \rightarrow 1$

Généralisation du théorème de Shannon :

Avec un multiplexeur $2^n\ \rightarrow 1$, il est possible de de réaliser toute fonction combinatoire de $(n+1)$ variables d'entrées. Dans ce cas, les entrées sélectionnées du multiplexeur prennent les valeurs « 0 », « 1 » et celle de la variable d'entrée de poids faible (LSB) ou de son complément.

Exemple : Implantation de la fonction logique $F(d, c,\ b,\ a)$, spécifiée par la table de vérité, donnée par le tableau 1.16, par un multiplexeur $2^3\ \rightarrow 1$.

Eq D	d	c	b	a	F		
0	0	0	0	0	1	$\overline{a}$	e_0
1	0	0	0	1	0		
2	0	0	1	0	0	0	e_1
3	0	0	1	1	0		
4	0	1	0	0	1	1	e_2
5	0	1	0	1	1		
6	0	1	1	0	0	0	e_3
7	0	1	1	1	0		
8	1	0	0	0	1	1	e_4
9	1	0	0	1	1		
10	1	0	1	0	1	1	e_5
11	1	0	1	1	1		
12	1	1	0	0	1	1	e_6
13	1	1	0	1	1		
14	1	1	1	0	0	0	e_7
15	1	1	1	1	0		

Tableau 1.16. Table de vérité de la fonction logique $F(d, c, b, a)$

L'implantation de la table de vérité du tableau 1.16 par un multiplexeur $2^3 \rightarrow 1$ est donnée par la figure 1.16.

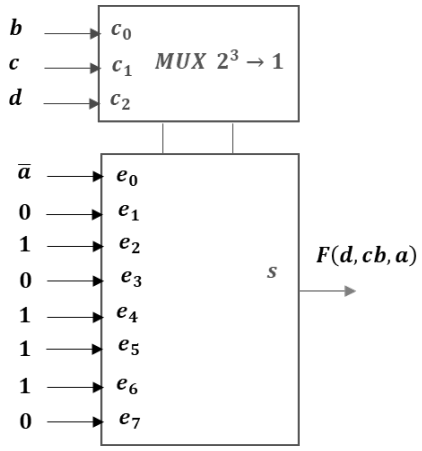


Figure 1.16. Implantation de la table de vérité du tableau 1.16 par un multiplexeur $2^3 \rightarrow 1$

1.8.1.3.2. Conversion parallèle-série

Sur la figure 1.17, nous donnons un exemple d'utilisation d'un multiplexeur $2^3 \rightarrow 1$ dans la réalisation de la conversion parallèle-série d'un mot-binaire de huit bits.

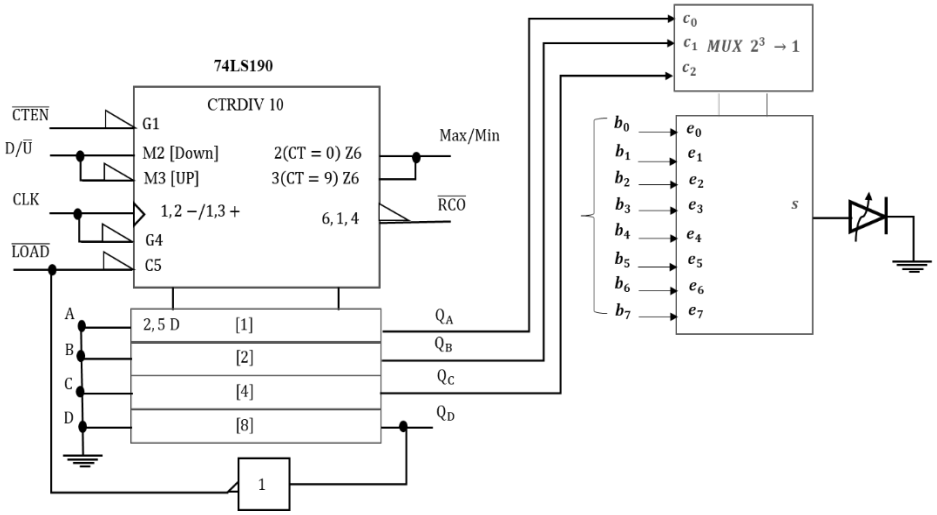


Figure 1.17. Utilisation d'un multiplexeur $2^3 \rightarrow 1$ dans la réalisation de conversion parallèle-série d'un mot-binaire de huit bits.

Le compteur « 74LS190 » est câblé pour compter de « 0 » à « 7 » (compteur modulo 8), afin de fournir les signaux de commande au multiplexeur. Aux entrées du multiplexeur on affecte les huit bits du mot-binaire qui sortent en série bit par bit, le bit b_0 en tête (sur la sortie s du multiplexeur) au rythme de l'horloge CLK, comme le montrent les chronogrammes de la figure 1.18.

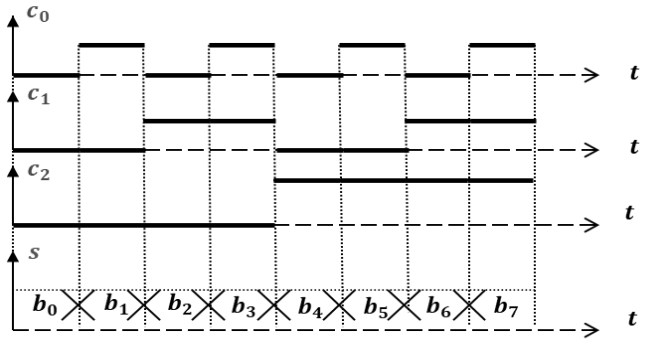


Figure 1.18. Chronogrammes des signaux de commande et de la sortie s

1.8.2. Codeurs ou encodeurs / décodeurs - transcodeurs

1.8.2.1. Encodeur binaire de priorité $2^n \rightarrow n$

Un encodeur binaire de priorité $2^n \rightarrow n$ est un circuit qui possède :

$$\begin{cases} 2^n \text{ entrées d'information} \\ n \text{ sorties} \end{cases}$$

Par définition un encodeur binaire génère en sortie un code binaire équivalent au numéro décimal d'une entrée activée. Si plusieurs entrées sont actives simultanément, on impose par construction une priorité, et le codeur code l'entrée la plus prioritaire.

Pour appréhender cette définition, nous prenons l'exemple de l'encodeur prioritaire « 74LS348 ». Son symbole logique est illustré par la figure 1.19, et sa table de vérité est donnée par le tableau 1.17.

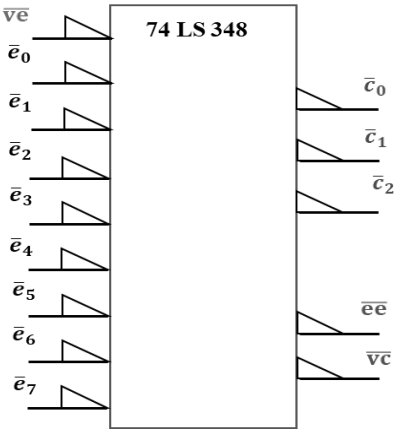


Figure 1.19. Symbole logique de l'encodeur prioritaire « 74LS348 »

Entrées									Sorties				
ve	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	c_2	c_1	c_0	ee	vc
1	X	X	X	X	X	X	X	X	Z	Z	Z	1	1
0	1	1	1	1	1	1	1	1	Z	Z	Z	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

Tableau 1.17. Table de vérité de l'encodeur prioritaire « 74LS348 »

La table de vérité du tableau 1.17 répond parfaitement à la définition de l'encodeur binaire de priorité, « 74LS348 » $2^3 \rightarrow 3$. Par exemple, lorsque l'entrée e_7 est active niveau bas, quels que soient les états des autres entrées, les trois sorties $c_2\ c_1\ c_0$ sont actives niveau bas.

En plus de ses entrées et sorties principales, l'encodeur « 74LS348 », dispose d'une entrée « ve » trois états pour la validation des entrées, d'une sortie « ee » indiquant les états des entrées et une autre sortie « vc » pour la validation du cascadage. Les équations des sorties peuvent être exprimées directement à partir de la table de vérité. À titre illustratif, l'équation de la sortie c_2 est :

$c_2 = \overline{\text{ve}}\,[\bar{e}_7 + e_7\,\bar{e}_6 + e_7\,e_6\,\bar{e}_5 + e_7\,e_6\,e_5\,\bar{e}_4] = \overline{\text{ve}}\,[\bar{e}_7 + \bar{e}_6 + \bar{e}_5 + \bar{e}_4]$.

Rappelons que l'intérêt de l'encodage ici est de permettre la réduction des variables logiques à traiter, par exemple les 128 symboles du clavier d'un micro-ordinateur sont codés avec 7 bits seulement.

1.8.2.2. Décodeur binaire $\boldsymbol{n \rightarrow 2^n}$

Un décodeur binaire $n \rightarrow 2^n$ est un circuit qui possède : $\begin{cases} n\ entrées \\ 2^n\ sorties \end{cases}$

Par définition, un décodeur binaire fait correspondre à une combinaison codée en binaire naturel sur n bits présente en entrée une sortie active et une seule parmi 2^n sorties.

Pour bien illustrer cette définition, nous prenons l'exemple du décodeur binaire-décimal « 74LS138 ». Son symbole logique est représenté par la figure 1.20, et sa table de vérité est donnée par le tableau 1.18.

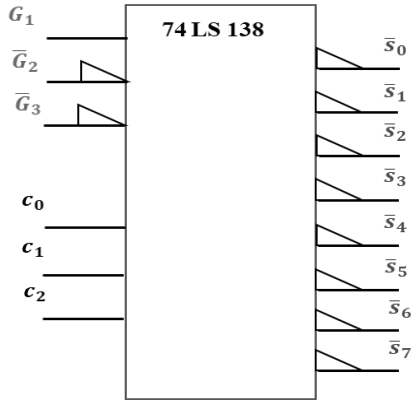


Figure 1.20. Symbole logique du décodeur binaire-décimal « 74LS138 »

Validation		**Entrées (code binaire)**			**Sorties**							
G_1	$\overline{G} = \overline{G}_2 + \overline{G}_3$	c_2	c_1	c_0	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
X	1	X	X	X	1	1	1	1	1	1	1	1
0	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

Tableau 1.18. Table de vérité du décodeur binaire-décimal « 74LS138 »

La table de vérité du tableau 1.18 illustre la définition du décodeur binaire-décimal « 74LS138 » $3 \rightarrow 2^3$. En effet, par exemple, lorsque l'équivalent décimal des entrées $c_2\ c_1\ c_0$ est égal à zéro, seule la sortie s_0 sera active au niveau bas.

En plus de ses entrées principales le décodeur binaire-décimal « 74LS138 », dispose de trois entrées, G_1, G_2 et G_3 pour la validation du circuit.

L'équation des différentes sorties est donnée par :

$\bar{s}_k = G_1(\bar{G}_2 + \bar{G}_3)[c_2c_1c_0 = k],\ k = 0,\ 1,\ \cdots, 7.$

Une utilisation typique des décodeurs binaires est le décodage d'adresses de circuits, comme c'est montré sur l'exemple qui suit.

Exemple d'application : décodage d'adresses de circuits mémoires.

L'utilisation de plusieurs boîtiers mémoires de 1024 mots pose le problème de sélection de ces boîtiers. Avec un μP de 16 bits, on peut avoir $2^{16} = 65536$ lignes d'adresses, soit 64 boîtiers mémoires de 1024 mots chacun.

La figure 1.21 montre la structure d'un boîtier mémoire de 1024 mots de 8-bit chacun. La structure intègre un décodeur binaire interne $10 \rightarrow 2^{10}$ pour adresser les 1024 mots du boîtier mémoire.

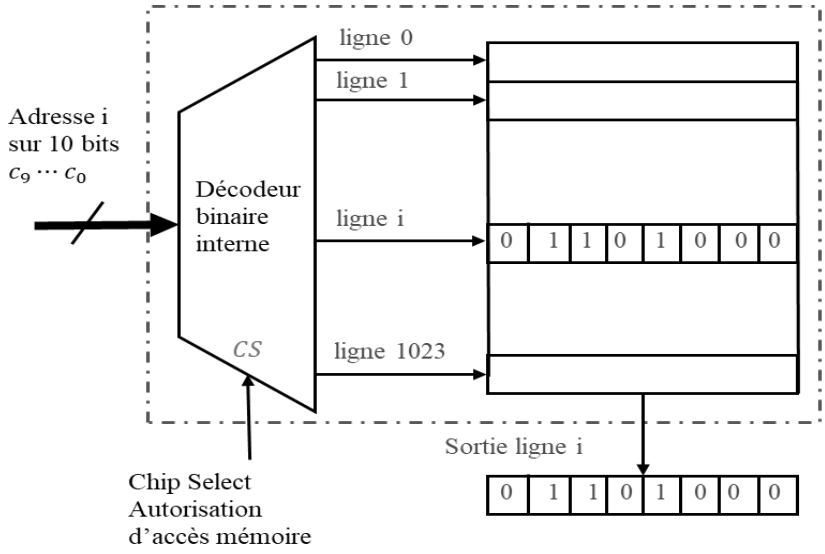


Figure 1.21. Structure d'un boîtier mémoire de 1024 mots de 8-bit chacun

Sur la figure 1.22, nous montrons comment adresser 64 boîtiers mémoires de 1024 mots.

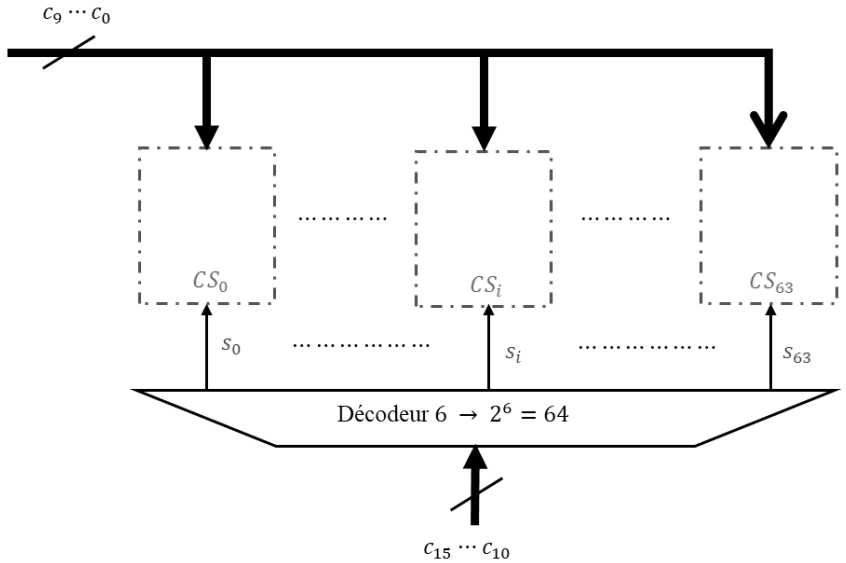


Figure 1.22. Adressage de 64 boîtiers mémoires de 1024 mots chacun

Si par exemple nous sélectionnons le botier N° 3, nous adressons alors tous les mots (octets) compris entre l'adresse $(0C00)_H$ et l'adresse $(0FFF)_H$ comme montré par l'équation suivante :

$(c_{15} \cdots c_{10})_2 = (3)_{10} \implies (0C00)_H \leq (000011c_9 \cdots c_0)_2 \leq (0FFF)_H$

1.8.2.3. Transcodeurs

Un transcodeur est un circuit permettant de passer du nombre N écrit avec le code C_1 au même nombre N écrit avec le code C_2. Plus généralement, un transcodeur permet de calculer $Y = f(X)$.

En pratique, il existe un nombre important de transcodeurs que l'on trouve en circuits intégrés, nous citons à titre d'exemple les transcodeurs suivants :

- le convertisseur décimal – binaire SN 147 ;
- le convertisseur binaire – décimal SN 45 ;
- le convertisseur excédant 3 - décimal SN 43 ;
- le convertisseur Gray excédant 3 - décimal SN 44 ;
- le transcoder DCB – affichage sur 7 segments 74LS47.

Par la suite, à titre d'exemples, nous développons le transcodeur $Y = \log_2 X$ et nous analysons succinctement le transcoder DCB – affichage sur 7 segments, circuit intégré « 74LS47 ». Ce dernier fait l'objet d'une expérimentation dans le chapitre 5, avec les travaux pratiques.

1.8.2.3.1. Transcodeur $Y = \log_2 X$

Cahier des charges : Réalisez un transcodeur $Y = \log_2 X$, pour les valeurs décimales de X allant de 0 à 15, et pour une erreur maximale admise sur la partie fractionnaire de Y de 2^{-4} .

Solution :

Ce transcodeur est traité en détail dans l'exercice 14, de la partie 2A sur les exercices. Nous donnons ci-après l'essentiel pour comprendre.

Sachant que : $\log_a X = \log_e X / \log_e a$, la valeur maximale de Y est $Y = \log_2 15 = 3,907$, donc les formats en binaire de X et Y sont : $X = x_3x_2x_1x_0 \rightarrow Y = y_1y_0, y_{-1}y_{-2}y_{-3}y_{-4}$.

Prenant l'exemple de $Y = \log_2 3 = 1,585$, la partie entière est codée sur « 0 1 » et le codage binaire de la partie fractionnaire est obtenu comme suit (voir exercice 1) :

$0,585 \times 2 = 1,17 \rightarrow y_{-1} = 1$
$0,17 \times 2 = 0,34 \rightarrow y_{-2} = 0$
$0,34 \times 2 = 0,68 \rightarrow y_{-3} = 0$
$0,68 \times 2 = 1,36 \rightarrow y_{-4} = 1$

L'erreur maximale est de 2^{-4} (cas extrême d'une suite illimitée de « 1 » à partir de y_{-5}), donc on arrête la conversion au symbole y_{-4}

D'où la table de vérité (non complètement développée) est donnée par le tableau 1.19.

EqD de X	x_3	x_2	x_1	x_0	EqD de Y	y_1	y_0 ,	y_{-1}	y_{-2}	y_{-3}	y_{-4}
0	0	0	0	0	∞	-	- ,	-	-	-	-
1	0	0	0	1	0	0	0 ,	0	0	0	0
2	0	0	1	0	1	0	1 ,	0	0	0	0
3	0	0	1	1	1,585	0	1 ,	1	0	0	1
-	-	-	-	-	-	-	- ,	-	-	-	-
-	-	-	-	-	-	-	- ,	-	-	-	-
-	-	-	-	-	-	-	- ,	-	-	-	-
15	1	1	1	1	3,907	1	1 ,	1	1	1	0

Tableau 1.19. Table de vérité du transcodeur $Y = \log_2 X$

1.8.2.3.2. Transcodeur DCB – Affichage sur 7 segments

Un transcodeur DCB – affichage sur 7 segments est un circuit qui possède : $\begin{cases} 4 \text{ entrées } DCB \\ 7 \text{ sorties} \end{cases}$

Rappelons qu'en code DCB (Décimal Codé Binaire) (BCD : *Binary Coded Decimal*), les chiffres décimaux de zéro à neuf, sont codés par les 10 premières combinaisons d'un mot binaire de quatre bits, notés ici : e_3, e_2, e_1, e_0.

Les sept sorties notées : a, b, c, d, e, f, g sont destinées à commander un afficheur 7 segments, dont les noms sont disposés (par convention) comme le montre la figure 1.23.

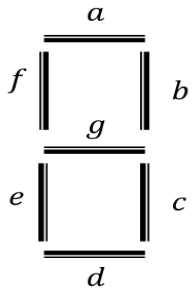


Figure 1.23. Afficheur 7 segments

La table de vérité, en logique positive, du transcodeur DCB – Affichage de 7 segments est donnée par le tableau 1.20.

EqD	Entrée					Sortie							Affichage
	e_3	e_2	e_1	e_0		g	f	e	d	c	b	a	
0	0	0	0	0		0	1	1	1	1	1	1	
1	0	0	0	1		0	0	0	0	1	1	0	
2	0	0	1	0		1	0	1	1	0	1	1	

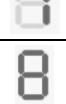
3	0	0	1	1		1	0	0	1	1	1	1	
4	0	1	0	0		1	1	0	0	1	1	0	
5	0	1	0	1		1	1	0	1	1	0	1	
6	0	1	1	0		1	1	1	1	1	0	1	
7	0	1	1	1		0	0	0	0	1	1	1	
8	1	0	0	0		1	1	1	1	1	1	1	
9	1	0	0	1		1	1	0	1	1	1	1	
10	1	0	1	0		1	1	1	1	0	0	1	
11	1	0	1	1		1	1	1	1	0	0	1	
12	1	1	0	0		1	1	1	1	0	0	1	
13	1	1	0	1		1	1	1	1	0	0	1	
14	1	1	1	0		1	1	1	1	0	0	1	
15	1	1	1	1		1	1	1	1	0	0	1	

Tableau 1.20. Table de vérité du transcodeur DCB – affichage de 7 segments

La lettre « E » est affichée (codée) ici, pour les combinaisons 10 à 15 non autorisées des entrées.

La table de vérité du transcoder DCB – affichage sur 7 segments du circuit intégré « 74LS47 » est différente pour les combinaisons 10 à 15 non autorisées (voir datasheet du circuit « 74LS47 » sur internet ou en salle de TP). On observe aussi une différence au niveau de l'écriture de 6 et du 9. Le symbole logique du circuit intégré « 74LS47 » est donné par la figure 1.24.

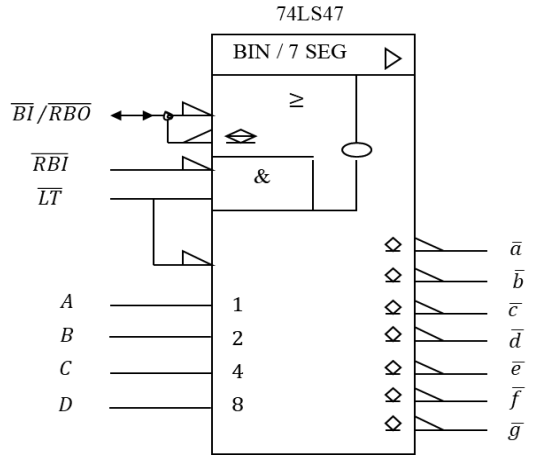


Figure 1.24. Symbole logique du transcoder DCB – affichage de 7 segments « 74LS47 »

En plus de ses entrées principales A, B, C, D, le circuit « 74LS47 » possède 2 entrées $\overline{LT}$, $\overline{RBI}$ et une entrée-sortie $\overline{BI}/\overline{RBO}$ supplémentaires, dont le rôle de chacun est :

- l'entrée $\overline{LT}$ « *Lamp Test* », qui est active au niveau logique bas, permet d'allumer toutes les LED et donc de vérifier leur bon fonctionnement ;
- l'entrée $\overline{RBI}$ « *Ripple Blanking Input* », qui est active au niveau logique bas, permet de valider ou de ne pas valider l'affichage du zéro. Cette fonctionnalité est utilisée lorsque plusieurs afficheurs sont branchés ensembles. On peut ainsi choisir d'éclairer ou non les digits de poids fort lorsque l'on affiche un nombre de rang inférieur au maximum utilisé ;

Exemple : on a deux afficheurs et on veut visualiser le chiffre quatre.

Si $\overline{RBI}$ vaut « 1 », alors sur les deux afficheurs, on verra : 0 4

Si $\overline{RBI}$ vaut « 0 », alors le digit de gauche sera éteint et le digit de droite affichera la valeur quatre. On verra :

4

- l'entrée/sortie $\overline{BI}/\overline{RBO}$ peut-être configurée en entrée « *Blanking Input* » ou en sortie « *Ripple Blanking output* ». Si on applique un niveau logique « 0 » sur la broche (vu comme une entrée, $\overline{BI}$), quelques que soient les valeurs des autres entrées, on force tous les segments à s'éteindre. Quand la broche est vue comme une sortie $\overline{RBO}$, elle vaut toujours « 1 », sauf lorsque $\overline{LT}$ vaut « 1 » et $\overline{RBI}$, D, C, B et A sont égaux à « 0 », alors elle vaut « 0 ».

Rôle de l'entrée/sortie $\overline{BI}/\overline{RBO}$

On a vu que $\overline{RBI}$ permet d'éteindre les « 0 » d'un digit. C'est intéressant si les digits à éteindre sont à gauche du digit à allumer, et gênant par ailleurs.

Exemple : si on travaille sur 3 digits :

0 0 4 ⇒ 4 *OK*

5 0 4 ⇒ 5 4 pose problème

Pour résoudre ce problème, on utilise le montage donné par la figure 1.25.

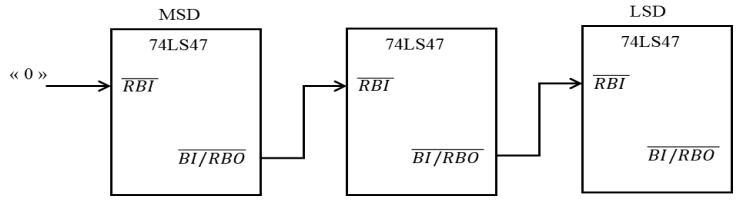


Figure 1.25. Montage de couplage de plusieurs décodeurs/afficheurs pour n'éteindre les digits à zéro que si les poids supérieurs sont également à zéro

En effet, si $\overline{RBI}$ du digit le plus fort (MSD) vaut « 0 », on aura :

$$\overline{BI}/\overline{RBO} = \begin{cases} 0 & si\ \ D,C,B,A = 0 \\ 1 & sinon \end{cases}$$

Si on branche la sortie $\overline{BI}/\overline{RBO}$ sur l'entrée $\overline{RBI}$ du digit de poids inférieur, on n'aura l'extinction de ce digit que si le poids supérieur était à zéro, avec $\overline{RBI}$ à zéro, et si le présent digit est à zéro.

1.9. Conception d'une fonction combinatoire quelconque

La démarche à suivre pour la conception d'une fonction combinatoire quelconque est schématisée par la figure 1.26.

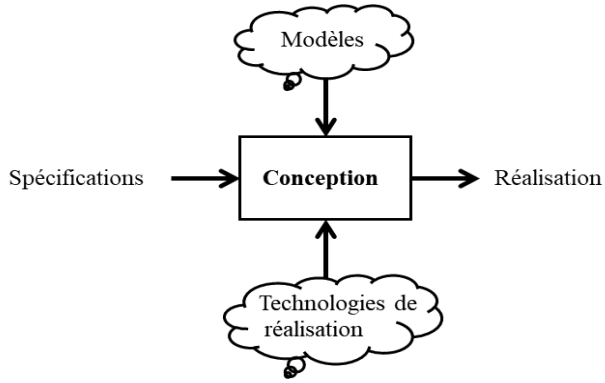


Figure 1.26. Démarche de conception d'une fonction combinatoire

1.9.1. Spécifications des fonctions combinatoires

Les spécifications sont d'essences fonctionnelles, opératoires et technologiques :

a) Fonctionnelles

Les spécifications fonctionnelles s'effectuent par :

- équations ; table de vérité ; table de Karnaugh ;
- description opératoire : modèle mathématique ; algorithmes ; description *HDL*.

Exemple : $S = A + B$.