Fondements mathématiques de l'INformatique (INF121)

Numération & Circuits logiques (INF 1212)

21 septembre 2023

Année académique 2023-2024

Table des matières

Présentation

Le cours vise à :

- Traiter en détails les différents systèmes de numération : systèmes décimal, binaire, octal et hexadécimal ainsi que les méthodes de conversion entre les systèmes de numération.
- Traiter les opérations arithmétiques sur les nombres dans différentes représentations de nombres possibles.
- Etudier plusieurs codes numériques tels que les codes DCB, GRAY et ASCII
- Analyser, synthétiser et construire des circuits numériques

Chapitre 1

Systèmes de numération et codage

1.1 Introduction

Dans un ordinateur, toute l'information est sous forme de bits qui sont regroupés en octets. Il faut donc qu'il y ait un codage de cette information. Ce codage dépend bien sûr du type des données. Cette partie décrit les codages les plus utilisés pour les types de base, c'est-à-dire les entiers, les nombres flottants et les caractères.

1.2 Système de numération

Soit un nombre N et C_i un ensemble de caractères, on dit que le nombre N est représenté dans la base B s'il existe la relation suivante entre N, C_i et B :

$$N = C_{n-1}.B^{n-1} + C_{n-2}.B^{n-2} + \ldots + C_1.B^1 + C_0.B^0$$

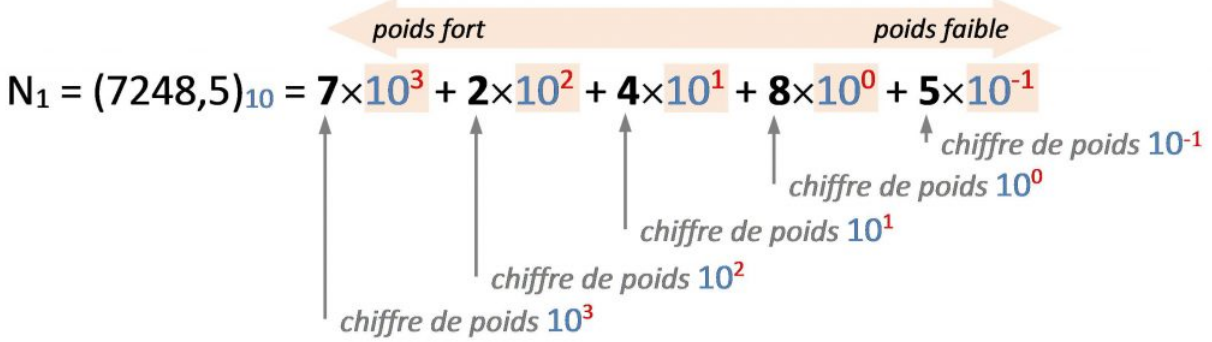
$B^{n-1}, B^{n-2}, \ldots, B_1, B_0$ étant les puissances de la base B, ils désignent les poids du nombre N.
C^{n-1} (resp. C^0) étant associé à la plus grande puissance, est dit **de poids fort** (resp. **de poids faible**).

Ainsi, un système de numération est un ensemble de symboles et de règles permettant la représentation de n'importe quel élément d'un ensemble de nombres données.
Exemple 1.

- Le système décimal permet d'écrire un nombre à l'aide de dix chiffres 0 à 9. Tout nombre est représenté par une combinaison de ces dix symboles. Ainsi : B=10, $C_i \in \{0, 1, 2, \ldots, 9\}$
- Dans le système binaire : B=2, $C_i \in \{0, 1\}$
- Dans le système hexadécimal : B=16, $C_i \in \{0, 1, 2, \ldots, 9, AB, C, D, E, F\}$

1.2.1 Notions de base

La base d'un système de numération est la référence qui permet l'écriture d'un nombre. Dans le cas du système décimal, la base est 10.
Exemple 2. $(7248) = 7.10^3 + 2.10^2 + 4.10^1 + 8.10^0$



En général, un nombre A peut être exprimé dans une base B par n chiffres a_i :

$$(A)_B = a_{n-1}a_{n-2}\dots a_0$$

Remarque 1. La notation $(A)_B$ signifie que A est exprimé dans la base B

Il se passe qu'en informatique, les nombres ne peuvent pas avoir une taille arbitrairement grande. Ils ont donc toujours une taille

fixée. Déterminons la plage de nombres que l'on peut écrire en base b avec des nombres de taille n : il y a n places possibles pouvant contenir chacune un chiffre entre 0 et b - 1, soit b^n nombres différents. Sur n chiffres, on écrit donc les nombres compris entre 0 et $b^n - 1$.

1.2.2 Le système décimal

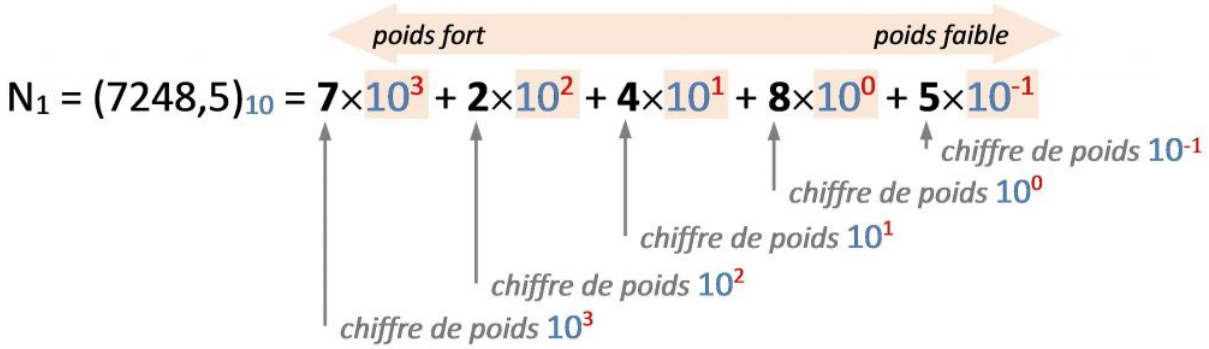
C'est le système que nous utilisons dans la vie courante. Il est basé sur les dix chiffres 0 à 9.
Si nous employons le système décimal avec aisance dans tous nos calculs, il n'en va pas de même pour les machines électroniques dans lesquelles ce système reste difficile à mettre en oeuvre pour plusieurs raisons d'ordre technique. C'est un autre système, plus adapté, qui est implanté.
Le problème de conversion se pose alors du moment que la machine et l'homme parlent deux langages différents : La machine ne peut accepter un nombre décimal et l'homme de son côté reste incapable d'interpréter un résultat fourni par la machine dans sa base sans faire appel à une gymnastique lourde d'esprit.
Par conséquent, les règles de passage entre la base décimale et n'importe quelle base B ont été définies :

Passage d'une base B à la base décimale

Soit A un nombre exprimé dans une base B : $(A)_{10} = a_{n-1}a_{n-2}\dots a_0$ avec $0 \leq a_i \leq B-1$
La valeur décimale du nombre A est calculée par :

$$(A)_{10} = a_{n-1}10^{n-1} + \dots + a_2 10^2 + a_1 10^1 + b_0 = \sum_{i=0}^{n-1}(a_i * 10^i)$$

Exemple 3. $(7248) = 7.10^3 + 2.10^2 + 4.10^1 + 8.10^0$



Passage de la base décimale à une base B

Soit A un nombre exprimé dans le système décimal.
L'équivalent de A dans une base B est calculé en divisant A par B, puis le quotient obtenu par B et ainsi de suite jusqu'à ce que le quotient soit nul. Les restes de divisions successives lus du bas vers le haut représentent le nombre A dans la base B.

1.2.3 Les systèmes binaire, hexadécimal et octal

Le système binaire

C'est la base 2. Elle est utilisée dans tous les systèmes électroniques où les deux états VRAI et FAUX sont facilement réalisables (interrupteur fermé ou ouvert, transistor bloqué ou saturé, lampe allumée ou éteinte, ...). Cette base comporte uniquement les deux symboles 0 et 1. Ce sont les chiffres binaires appelés aussi **bits** (**Bi**nary digi**t**) :

$$(A)_2 = a_{n-1}a_{n-2}\dots a_0 \text{ avec } a_i \in [0,1]$$

Vu l'importance de système, nous allons consacrer plusieurs paragraphes à l'étude de certains des ses aspects, notamment le calcul arithmétique et les nombres négatifs.
Remarque 2.

- Un regroupement de 4 bits est appelé **quartet**
- Un regroupement de 8 bits est appelé **octet**

Traditionnellement ou par convention et en contradiction avec le système international d'unités, les unités dérivées de l'octet telles que le kilo-octet (Ko), le mega-octet (Mo), le Giga-octet (Go), le Tera-octet (To), le Peta-octet (Po), l'exa-octet (Eo), le zetta-octet (Zo) ou encore le yotta-octet (Yo) sont utilisées pour représenter les valeurs ci-dessous en puissance de 2.

- 1 Ko = 2^{10} octets
- 1 Mo = 2^{10} Ko
- 1 Go = 2^{10} Mo
- 1 To = 2^{10} Go
- 1 Po = 2^{10} To
- 1 Eo = 2^{10} Po
- 1 Zo = 2^{10} Eo
- 1 Yo = 2^{10} Zo
- ...

Depuis la normalisation survenue en 1998, préfixes kilo, méga, giga, téra, etc, correspondent aux mêmes multiplicateurs que dans tous les autres domaines, soit des puissances de 10. On a donc

- 1 Ko = 10^3 octets
- 1 Mo = 10^3 Ko
- 1 Go = 10^3 Mo
- 1 To = 10^3 Go
- 1 Po = 10^3 To
- 1 Eo = 10^3 Po
- 1 Zo = 10^3 Eo
- 1 Yo = 10^3 Zo
- ...

En revanche, les mesures traditionnellement utilisées ont changé d'appelation pour devenir Kilo binaire (Kibi) ou Kilo internet (Kio), Mega binaire (Mebi) ou Mega internet (Mio), Giga binaire (Gibi) ou Giga internet (Gio), Tera binaire (Tebi) ou Tera internet (Tio), ... On a donc 1 Kibi = 1 Kio = 2^{10} octets, 1 Mebi = 1 Mio = 2^{10} Kibi, ...

Par ailleurs, cette distinction est d'ailleurs utilisée depuis longtemps par les fabricants de supports de stockage qui utilisent les préfixes en puissances de 10 contrairement aux concepteurs de Systèmes d'exploitation qui jusqu'à présent, mesurent à l'aide du kilo binaire (préfixe de 2^{10}).
Ainsi, un disque dur de 100 Go (100×10^9 octets) contient le même nombre (arrondi) d'octets qu'un disque de 93,13 gibioctets ($93,13 \times 2^{30}$ octets).

Le système hexadécimal

Le système hexadécimal comporte 16 symboles : les dix chiffres 0 à 9 et les six lettres A, B, C, D, E et F. Ce système est très répandu pour la simple raison qu'il permet de représenter les nombres binaires d'une manière plus compacte.

Exemple 4. Soit en base hexadécimale : $(A)_{16} = 3CB$.
En décimal : $(A)_{16} = 3.16^2 + 12.16^1 + 11.16^0 = 971$
Remarque 3.

- Pour passer de la base 16 à la base 2, on exprime chaque chiffre hexadécimal en binaire sur 4 bits.
- Pour faire la transformation inverse (Base 2 vers 16), il suffit de prendre les chiffres binaires 4 par 4 en partant de la droite et convertir chaque quartet en hexadécimal.

Décimal	Binaire	Hexa	Décimal	Binaire	Hexa
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

TABLE 1.1 – Table de correspondance Décimal / Binaire / Hexadécimal

Exemple 5.

$(N)_2 = \underbrace{0011}_{3}\underbrace{1110}_{E}$ donc $(N)_{16} = 3E$

Remarque 4. Vu que le passage Base 2 / Base 16 est direct et que la représentation hexadécimale est plus compacte, on préfère très souvent écrire les nombres binaires en hexadécimal.

Le système octal

Il contient 8 symboles : 0 à 7. Il est moins fréquemment utilisé dans la représentation des nombres binaires. Le passage du système binaire au système octal se fait par groupement de 3 bits.

Exemple 6.

$(A)_2 = \underbrace{\_1}_{1}\underbrace{100}_{4}\underbrace{110}_{6}$ donc $(A)_8 = 146$

$(A)_8 = 253$ d'où $(A)_2 = 100101011$

Quelques propriétés sur la numération

Soit $x = b_n b_{n-1} \dots b_2 b_1 b_0$ écrit dans la base β.

- Si $b_n \neq 0$, on note $n + 1 = N_\beta(x)$ le nombre de chiffres nécessaires pour exprimer x dans la base β.
- Estimons $N_\beta(x)$

Théorème 1. $N_\beta(x)$ est le plus petit entier strictement supérieur à $Log_\beta(x)$

Exemple 7. $\beta = 2$ et $x = (1503)_{10}$
$ln(1503) = 7.32$ et $ln(2) = 0.693 \Rightarrow Log_2(1503) = \frac{7.32}{0.693} = 10.6$
$\Rightarrow N_2(1503) = 11.$
Il faut donc 11 bits pour représenter $(1503)_{10}$ en binaire.
En effet, ce nombre s'écrit, en base 2 : 10111011111
Remarque 5. Le rapport $\frac{N_\beta(x)}{N'_\beta(x)}$ tend vers $\frac{Log_\beta((x))}{Log'_\beta((x))}$ quand x tend vers l'infini

Exemple 8. Pour écrire un grand nombre en base 2, il faut environ 3.32 fois plus de digits qu'en base 10 car $\frac{Log_{10}(x)}{Log_2(x)} = 3.32\dots$

1.2.4 Représentation des entiers relatifs

Il existe 3 modes de représentation des nombres signés en binaire :

Représentation en valeur absolue et bit de signe

On utilise un caractère binaire pour représenter le signe, le bit de poids fort. Conventionnellement, on attribue la valeur 0 au signe (+) et la valeur 1 au signe (?). Ici, tout nombre positif est représenté comme en binaire pur (commence par 0). En revanche, tout nombre négatif a son bit de poids fort à 1 et sa valeur absolue représentée en binaire pur sur les bits restants.

Exemple 9.

$+12 \rightarrow 01100$
$-12 \rightarrow 11100$
↑
bit de signe

Remarque 6.

- Pour une mémoire à n bits (n>1), tous les entiers naturels de l'intervalle $[-(2^n-1),(2^n-1)]$ seront représentés.
- Ce système de représentation n'est pas couramment employé car il n'est pas pratique dans les opérations arithmétiques.
- De plus, 0 a deux représentations à savoir 1000 et 0000
- Le zéro possède deux représentations +0 ($0\ldots0$) et -0 ($1\ldots1$) ce qui conduit à des difficultés au niveau des opérations arithmétiques.
- Pour les opérations arithmétiques il nous faut deux circuits : l'un pour l'addition et le deuxième pour la soustraction

Exemple 10. Opérations arithmétiques (avec n=4)

5	0101	5	0101	Résultats faux : 0 retourné
+3	+0011	−3	+1011	⇓
8	1000	2	0000	Complication des opérations arithmétiques

Représentation par complément à 1

Comme dans le cas précédent, le bit de poids fort permet de représenter le signe du nombre.

- Tout nombre positif (resp. négatif) s'écrit 0 (resp. 1) suivi de la représentation non signée (resp. complémentée) du nombre.
- L'addition est aisée : on additionne les nombres comme s'il s'agissait de représentations non signées, et si l'opération conduit à une retenue en position n (bit de poids fort), l'ajouter à la somme calculée.
- Le nombre 0 a deux représentations : $000\ldots000$ et $111\ldots111$

Exemple 11.

$+7 \rightarrow 0111$
$-7 \rightarrow 1000$

Remarque 7.

- Ici aussi, le principal inconvénient est que le zéro possède deux représentations +0 ($0\ldots0$)et -0 ($1\ldots1$) ce qui conduit à des difficultés au niveau des opérations arithmétiques.
- Pour les opérations arithmétiques il nous faut deux circuits : l'un pour l'addition et le deuxième pour la soustraction
- Pour n bits (n>1), tous les entiers naturels de l'intervalle $[-(2^{n-1}-1),(2^{n-1}-1)]$ seront représentés. $\mathbf{C_1(N)+N=2^n-1}$

Exemple 12. Opérations arithmétiques (avec n=4)
Soit $N = 1010$
$\Rightarrow N' = C_1(N) = 2^4 - 1 - N = (16-1)_{10} - (1010)_2$
$\Rightarrow N' = (15)_{10} - (1010)_2 = (1111)_2 - (1010)_2 = 0101$
Sur n=8 bits, $106 - 2 = 01101010 + 10000001 = 01101011 + 1 = 01101100$

Représentation par complément à 2

Le complément à 2 s'obtient en ajoutant 1 au complément à 1.
Exemple 13.
+7 $\rightarrow$ en binaire naturel sur 4 bits : 0111 $\rightarrow$ le complément à 1 : 1000
auquel on ajoute 1 : 1000 + 1 = 1001 (l'équivalent de (-7) en code complément à 2 sur 4 bits)
Remarque 8.

- Le code complément à 2 est le plus utilisé car il est adapté aux opérations arithmétiques.
- Le bit de poids fort indique le signe du nombre : 1 $\rightarrow$ (-) et 0 $\rightarrow$ (+)
- On doit toujours spécifier le nombre de bits lorsqu'on travaille en complément à 2. Sur n bits, on peut représenter les nombres allant de -2^{n-1} à $2^{n-1} - 1$. Sur 4 bits par exemple, on ne peut représenter que les nombres compris entre -8 et +7 :
- Tout nombre N positif (resp. négatif) s'écrit 0 (resp. 1) suivi de la représentation non signée (resp. complémentée à 1) de N+1.
- L'addition est aisée : on additionne les nombres comme s'il s'agissait de représentations non signées, et si l'opération conduit à une retenue en n, supprimer ladite retenue.
- Le nombre 0 a une seule représentation : $000\ldots000$ Pour n bits (n>1), tous les entiers naturels de $[-2^n, 2^{n-1} - 1]$ seront représentés. On note que $\mathbf{X + 2^n = X}$ et $\mathbf{C_2(N) = 1 + C_1(N)}$
- En cas d'addition de :
  - 2 nombres de signes opposés
    - le résultat est représentable avec le nombre de bits fixés, pas de dépassement de capacité
    - s'il y a une retenue, on l'oublie.
    - On lit directement le résultat codé en complément à 2
  - 2 nombres de même signe :
    - Il y a dépassement de capacité si la retenue est distincte du dernier bit de report (i.e. celui sur le bit de signe)
    - s'il y a une retenue, on l'oublie.
    - On lit directement le résultat codé en complément à 2
- Pour les autres opérations arithmétiques, le codage en complément à deux est souvent choisi en pratique.

Exemple 14.

1.2.5 Le codage DCB (Décimal codé binaire) ou BCD (Binary Coded Decimal)

Il s'agit d'une représentation surtout utilisée dans les premiers temps de l'informatique. La représentation BCD d'un entier n est obtenue de la manière suivante. L'entier n est écrit en décimal puis chaque chiffre décimal entre 0 et 9 est ensuite codé sur 4 bits. Il faut donc (k+1)/2 octets pour coder un nombre ayant k chiffres décimaux. Le seul intérêt de cette représentation est de faciliter l'affichage en base 10 de l'entier. Certains processeurs possédaient quelques instructions spécifiques pour traiter, en particulier pour l'addition, les entiers codés en BCD.

Remarque 9.

ncrémenter Pour compter en BCD, on compte en binaire, et on détecte la combinaison 1010 qui suit 1001=9 et on ajoute 6, ce qui corrige le digit et ajoute 1 au chiffre de poids plus fort.

écrémenter Pour décompter en BCD, il suffit de surveiller les digits valant 0xF résultants de la soustraction.

dditionnner L'addition binaire doit être suivie d'un test chiffre par chiffre en commencant par les poids faibles. Si le résultat pour le digit est supérieur à 9, il faut ajouter 6 avant de tester le digit suivant.

Soustraire La soustraction binaire doit être suivie du test chiffre par chiffre en commencant par les poids faibles. Si le résultat pour le digit est supérieur à 9, il faut soustraire 6 avant de tester le digit suivant. Si le résultat est négatif .. ?

D en binaire Le plus rapide est d'avoir une table des poids binaire des bits des chiffres BCD et d'additionner les poids des bits à 1.

Exemple 15.

$18_{10} + 8_{10}$ = 0001 1000 + 0000 1000

```
0001  1000
0000  1000
-------------
0001 10000   On ajoute 6 car c'est > 9
    ┌ 0110
0001 10110   Le cinquième bit est passé comme retenue au digit de gauche
-------------
0010 0110    = 26₁₀
```

FIGURE 1.1 – Addition en BCD

$20_{10} - 8_{10}$ = 0010 0000 - 0000 1000

```
 0010  0000
-0000  1000
 ---------------
 0001  1000   Emprunt, on soustrait 6
        110
---------------
 0001  0010   = 12₁₀
```

FIGURE 1.2 – Soustraction en BCD

1.2.6 Représentation des nombres réels

Comme pour les autres types d'information, pour représenter un nombre réel, on dispose d'un nombre fini de bits et on peut imaginer différentes façons de représenter un réel avec cet ensemble de bits. Les deux méthodes de représentation les plus connues sont la représentation en virgule fixe et la représentation en virgule flottante. C'est cette dernière qui est la plus utilisée.

Représentation en virgule fixe

Il s'agit de la représentation habituelle comme on le fait sur papier sauf pour le signe et la virgule. Pour le signe, on réserve un bit, en général le bit de gauche. La virgule quant à elle n'est pas représentée, on travaille avec une virgule implicite située à une place bien déterminée.

Exemple 16.

- Représentation sur 9 bits
- Bit de gauche réservé au signe (0 = positif, 1 = négatif)
- Les autres bits représentent la valeur absolue
- La virgule est placée à gauche du 4ème bit à partir de la droite

| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | $= + (1100,0111)_2 = (12,4375)_{10}$

Rappelons qu'une méthode systématique pour convertir un nombre réel de décimal à binaire consiste à convertir la partie entière par divisions successives et la partie fractionnelle par multiplications successives, on arrête la multiplication quand on a un résultat nul ou quand on estime qu'on a suffisamment de chiffre après la virgule.

0.4375 x 2 = 0.875 → 0
0.875 x 2 = 1.75 → 1
0.75 x 2 = 1 .5 → 1
0.5 x 2 = 1 .0 → 1 ⇨ 0.4375 = 0,01

Comme avantage de cette représentation on peut citer le fait que les opérations aruthmétiques soient plus simples, ce qui induit des processeurs moins chers. De surcroît, ces opérations sont plus faciles à coder dans la machine.
Cependant, cette représentation n'est pas très utilisée car elle a une faible précision à cause des chiffres perdus à droite de la virgule. On peut vérifier qu'avec la représentation de l'exemple précèdent, 011000111 représente aussi bien le nombre +12,4375 que le nombre +12,4999.
De même, les deux nombres +0.5 et 0.56 seront tous les deux représentés par 000001000. Un autre problème de cette représentation est qu'elle ne permet pas de représenter les nombres très grands.

Pour effectuer une opération d'addition en virgule fixe, il faut dans cet ordre :

1. Aligner les virgules.
2. Choisir le nombre de bits à utiliser pour représenter les deux nombres : le nombre maximal de bits des parties entière et fractionnaire des deux opérandes.
3. Procéder à l'addition qui se fait bit fait par bit avec propagation des retenues
4. Pour éviter tout débordement, la somme nécessite un bit de plus que les opérandes

Par ailleurs,

- La représentation de nombres binaires signés à virgule fixe suit la représentation de nombres entiers signés en complément à deux.
- L'extension du signe se fait de la même façon que pour les nombres entiers.
- La soustraction de nombres binaires signés à virgule fixe se fait comme pour les nombres entiers

Enfin,

- La multiplication se fait comme pour les nombres entiers.
- On fait d'abord la multiplication sans tenir compte des virgules, puis à la fin on ajuste la virgule à la bonne place.

Exemple 17.
On observe que, si on enlève les virgules, la multiplication revient à faire
$7 \times 5 = 35$. Le produit doit être décalé de 3 bits vers la droite, soit une division par 8. On a bien $35 \div 8 = 4,375$.

Ci-dessous, erreur signalée en rouge : +1,125 ne peut pas être représenté avec seulement 1 bit pour la partie entière en notation signée

3,25	11,01
+ 13,125	1101,001
3,25	00011,010
+ 13,125	01101,001
16,375	10000,011

3,0625	11,0001
+ 13,125	1101,001
3,0625	00011,0001
+ 13,125	01101,0010
16,1875	10000,0011

1,75	01,11
× 2,5	010,1
	0111
	00000
	011100
	0000000
	0100011
4,375	0100,011

3,25	11,01
+ 13,125	1101,001
3,25	00011,010
+ 13,125	01101,001
16,375	10000,011

3,0625	11,0001
+ 13,125	1101,001
3,0625	00011,0001
+ 13,125	01101,0010
16,1875	10000,0011

1,5	001,1
inv(1,5)	110,0
+ 1×2^{-1}	000,1
-1,5	110,1
	= -4+2+0,5

-4,1875	1011,1101
inv(-4,1875)	0100,0010
+ 1×2^{-4}	0000,0001
4,1875	0100,0011

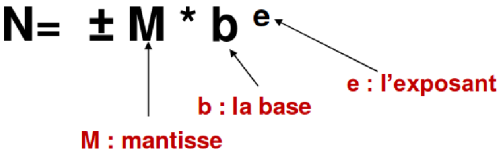
1,125	1,001
inv(1,125)?	0,110
+ 1/8	0,001
0,875	0,111

1,125	01,001
inv(1,125)	10,110
+ 1/8	00,001
-1,125	10,111
	= -2+7/8

Représentation en virgule flottante

Les scientifiques ont depuis longtemps appris à s'accommoder avec les problème de la virgule fixe en utilisant la représentation scientifique dans laquelle un nombre est représenté par le produit d'une mantisse et d'une puissance de 10 (décimal) ou d'une puissance de 2 (binaire). La représentation des nombres sur un ordinateur selon cette méthode est désignée par représentation en virgule flottante.

$R = -4{,}463876 \ 10^{23}$ (décimal)
$R = +5 \ 10^{-56}$ (décimal)
$R = -1{,}1001110 \ 2^{110011}$ (binaire)
$R = +1{,}11110 \ 2^{-10011}$ (binaire)



Dans les ordinateurs, la représentation se fait évidement en binaire, il faut maintenant se mettre d'accord sur la répartition du nombre de bits dont en dispose pour représenter :

- Le signe du nombre
- la mantisse
- l'exposant signé

Pour pouvoir échanger des documents, il est indispensable que tout le monde utilise la même représentation. Depuis les années 70, il existe un standard pour la représentation des flottants.
Aujourd'hui la plupart des ordinateurs utilisent ce standard. C'est la représentation IEEE 754.
Un nombre flottant est codé par 3 nombres représentés sur la figure ci-dessous.

On distingue plusieurs variantes dont les plus utilisée sont :

	taille	**signe**	**e**	**m**	**valeur**
Simple précision	32 bits	1	8	23	$-1^s m 2^{e-127}$
Double précision	64 bits	1	11	52	$-1^s m 2^{e-1023}$

Dans cette représentation et quelle que soit la précision (format) :

- La mantisse est de taille fixée et normalisée sous la forme $\pm 1, m \times 2^e$

Signe	Exposant	Mantisse

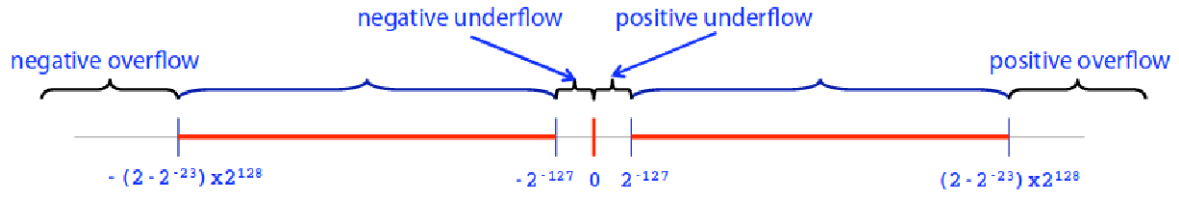
- On fait flotter la virgule en faisant varier e
- Le 1 précédant la virgule n'est pas codé en machine (bit caché)
- Exposant est en représentation biaisée. Ici, $Biais = 2^{n-1} - 1$

Aussi, il faut noter que cette norme ne représente pas que les nombres normalisés. Ainsi peut-on représenter le zéro, les infinis et d'autres nombres dénormalisés conformément aux règles édictées par le tableau suivant :

Exposant	**Mantisse**	**Valeur**
0	± 0	0
0	Non 0	$\pm 2^{-126} * 0.M$
$[-126, 127]$	Tout	$\pm 2^{E+127} * 1.M$
± 128	± 0	$\pm \infty$
± 128	Non 0	Conditions spéciales (NaN)

Quelques particularités :

- Le rang des nombres couverts en précision simple est :



- Il y a toujours un compromis entre le rang et la précision : si on augmente le nombre de bits de l'exposant, on étend le rang de nombres couverts. Mais, comme il y a toujours un nombre fixe de nombres que l'on peut représenter, la précision diminue. La seule façon d'augmenter le rang et la précision est d'augmenter le nombre total de bits du format
- Dans les deux précisions (simple et double) il y a quelques valeurs particulières :
  - deux valeurs pour 0 : les deux signes, avec exposant=mantisse=0
  - infini positif : signe positif, exposant=1…1, mantisse=0
  - infini négatif : signe négatif, exposant=1…1, mantisse=0
  - NaN (not a number) : signe indifférent, exposant$= 1 \ldots 1$, mantisse$\neq 0$

Exemple 18. On veut représenter le nombre $-(6,625)_{10}$ en IEEE 754 simple précision.

- Binaire (valeur absolue) : $(6)_{10} = (110)_2$ et $(0,625)_{10} = (0,101)_2$
  D'où $-(6,625)_{10} = (110,101)_2$
- Normalisation de la mantisse : $1,10101 \times 2^2$
- Occupation des 24 bits de mantisse
  $1,\underbrace{10101000000000000000000}$
- Décalage de l'exposant (En simple précision, Biais = 127) donc
  exposant $= (2 + 127)_{10} = (10000001)_2$
- Signe = 1 car négatif

Au final on a donc la représentation suivante :

1	10000001	10101000000000000000000

Considérons deux nombres réels N_1 et N_2 tel que $N_1 = M_1 * b^{e_1}$ et $N_2 = M_2 * b_{e_2}$

Addition : Pour effectuer la calcul de $N_1 + N_2$, deux cas peuvent se présenter :

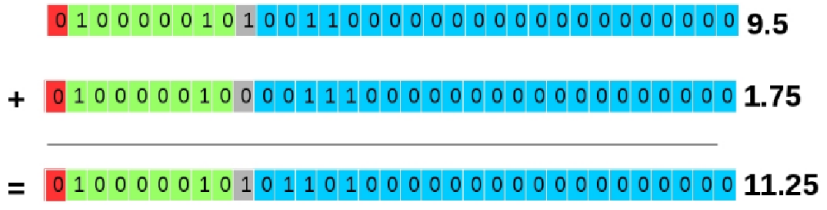
- Si $e_1 = e_2$ alors $N_3 = (M_1 + M_2)b^{e_1}$
- Si $e_1 \neq e_2$ alors il faut :
  1. élever au plus grand exposant (aligner)
  2. faire l'addition des mantisses
  3. normaliser la mantisse du résultat.

multiplication Pour effectuer la multiplication de deux nombres flottants :

- multiplier les mantisses ;
- additionner les exposants ;
- si nécessaire, soustraire le biais à partir du résultat

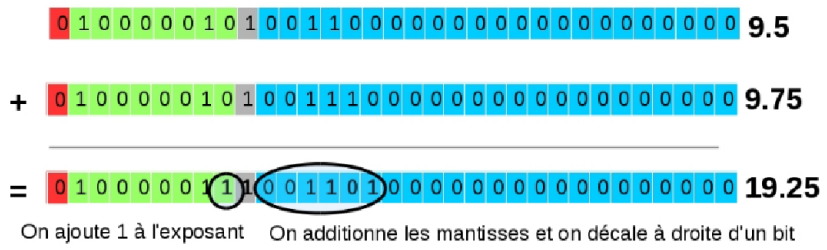
Exemple 19. Dans chacun des cas de cet exemple, considérer que le bit caché (couleur grisée) n'apparaît pas dans la représentation finale. On le présente ici juste pour une meilleure illustration.

Cas 1 : On veut effectuer l'opération suivante : $9,5 + 1,75$
L'addition des bits cachés(/implicites) ne pose pas de problème ici (1+0=1),on obtient un nombre avec un bit implicite.



Ceci fonctionne car il y a eu un décalage de mantisse pour l'un des opérandes.

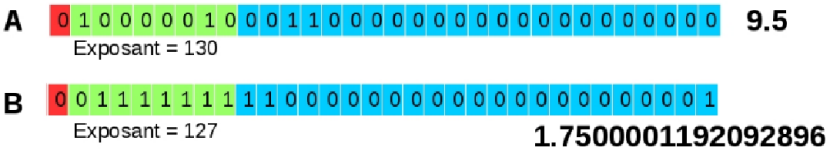
Cas 2 : On veut effectuer l'opération suivante : $9,5 + 9,75$



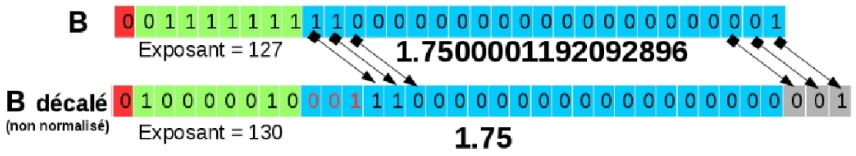
- On ajoute 1 à l'exposant
- On additionne les mantisses et on décale vers la droite d'un bit

On peut remarquer qu'en décalant d'un bit vers la droite, on perd le dernier bit (perte de précision)

Cas 3 : On veut effectuer l'opération suivante : $9,5 + 1$: 7500001192092896 qui illustre une perte de précision
Les exposants n'étant pas identiques, on aligne d'abord la mantisse de B

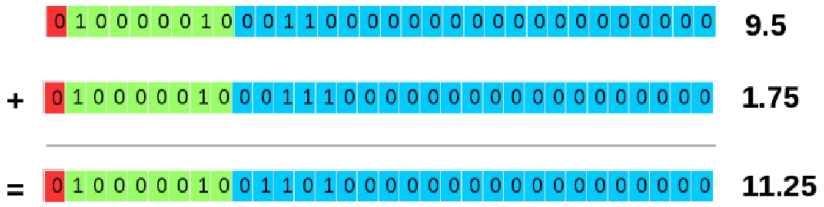


1. Aligner la mantisse de plus faible exposant dont l'illustration est donnée dans la figure ci-dessous
   On peut noter que :



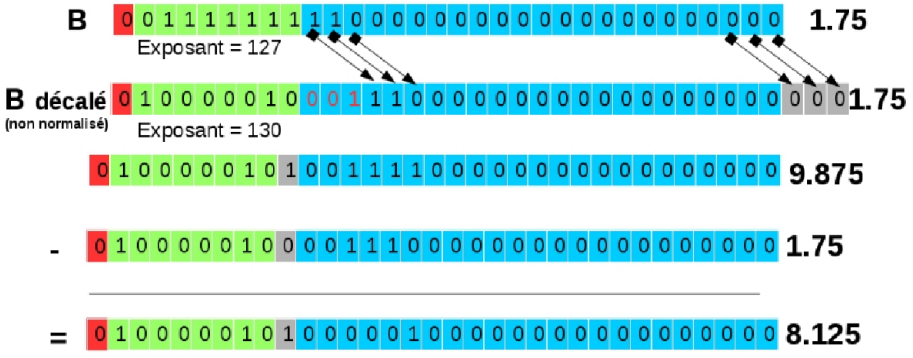
- Lors du décalage, il ne faut pas oublier de réintroduire le 1 (bit implicite)
- Après avoir réintroduit le 1, on introduit des 0
- Les bits qui "sortent" de la mantisse lors du décalage sont perdus : on perd donc de l'information

2. On somme les nouvelles mantisses, ce qui donne Il peut s'avérer utile de remarquer qu'ici :

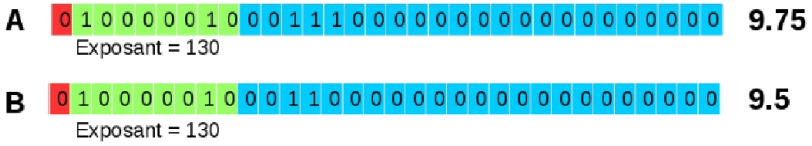


- On a perdu en précision
- Mais 11.25 est le **meilleur représentant** pour 11.2500001192092896
- L'addition respecte bien l'arrondi correct

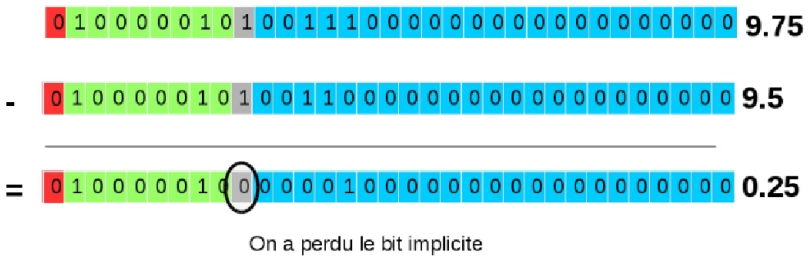
Cas 4 : Perte de précision en soustraction : $9.875 - 1.75$
[Cas 5 :] Perte de précision en soustraction : $9.875 - 9.5$



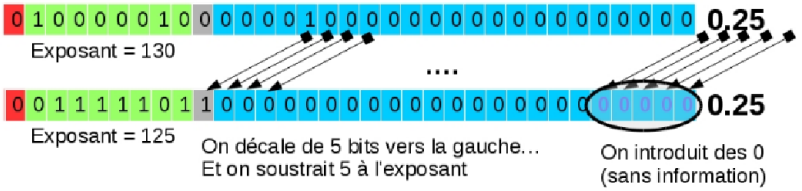
Les exposant sont identiques, on procède donc à la soustraction et on obtient :



Ayant donc perdu le bit implicite, il faut renormaliser le résultat. Ce qu'illustre la figure suivante : Noter qu'ici :



- Le résultat respecte l'arrondi correct
  C'est le plus proche représentant du vrai résultat
- **Mais** on a introduit des 0 **arbitrairement** (manque d'information)
- Aucun bit de la mantisse n'est **significatif**



1.3 Codage des caractères

1.3.1 Définition

Un caractère est un symbole informatique destiné à être affiché à l'écran ou imprimé sur un support quelconque.

L'homme utilise le plus souvent des informations sous forme de texte, de chiffres ou de symboles, c'est ce qu'on appelle en informatique des caractères (A,B,C,. . . ,1,2, 3,. . . ,(,@,&,. . .). Cependant, la mémoire de l'ordinateur ainsi que les supports d'information ne peuvent supporter que les valeurs 0 et 1. Il a donc fallu trouver un codage qui permettrait de représenter n'importe quel caractère sous forme d'un nombre binaire afin de faciliter son stockage et sa transmission. D'où l'existence des tables de conversion.

1.3.2 Code ASCII

Le code ASCII, mis pour American Standard Code for Information Interchange, était parmi les premiers codes utilisés pour ce propos. C'est un code universel qui fait correspondre à chaque caractère un code sur 7 ou 8 bits.
Dans la suite, vous trouverez une table détaillée du code ASCII. Dans sa première version, le code ASCII était codé sur 7 bits. Cette taille n'étant pas suffisante pour représenter tous les symboles, il a été étendu à 8 bits.

PDF : fr en v • d • m	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
002	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

FIGURE 1.3 – ASCII étendue : sur 8 bits, caractères accentués compris

1.3.3 Codage d'une chaîne de caractères

Une chaîne de caractères est stockée sous forme d ?une suite de codes ASCII représentant les caractères qui la constituent. Ces codes sont terminés par un indicateur de fin (en général, le caractère NUL (code ASCII : 0))
Exemple 20. En langage C, le mot "Bonjour" sera stocké dans la mémoire de l'ordinateur sous forme de 7 octets suivis du caractère NUL que sont : 42, 6F, 6E, 6A, 6F, 75, 72, 00

1.3.4 ISO 8859-1

Bien qu'étendu par la suite à un code sur 8 bits, beaucoup de caractères utilisés dans certaines langues ne sont pas représentés. L'organisme de normalisation OSI a donc défini différents standards plus adaptés aux besoins des différentes langues occidentales. Le français utilise le plus souvent ISO 8859-1, aussi nommé latin1, ou ISO 8859-15 (latin9).
Pour ce codage, on oublie le bit de parité, ce qui permet de multiplier le nombre de caractères par deux.

1.3.5 Norme Unicode - UTF8 depuis 1991

Enfin, Unicode créé en 1991 a pour vocation de rassembler tous ces codes afin d'avoir un code commun pour toutes les langues. Il contient plus d'un million de caractères.

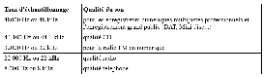


FIGURE 1.4 – NORME ISO 8859-1

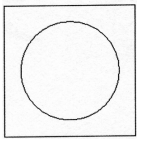
- Unicode est le standard d'échange de textes dans différentes langues le plus couramment utilisé aujourd'hui.
- Environs 110000 caractères.
- Compatible avec la norme ISO-8859-1 mais permet de coder des caractères non latins : caractères arabes, chinois, japonais, cyrilliques...
- C'est le codage UTF8, le plus répandu, qui permet son implémentation sur ordinateur. Dans ce codage, un caractère est représenté sur 1, 2, 3 ou 4 octets.

Pour permettre le décodage, le codage respecte une forme bien particulière :

1.4 Codage des images

1.4.1 Introduction

Pour représenter l'image ci-contre, nous avons vu deux méthodes :



- La première est de dire : "cette image est un cercle de centre Ω et de rayon R = 4". C'est la représentation dite vectorielle, ou symbolique, de l'image.
- La deuxième est de superposer à l'image un quadrillage. Chacune des cases est un pixel. On indique alors pour chaque carreau la couleur du pixel : « blanc blanc noir etc... ". Pour gagner de la place, on codera sur un bit blanc et noir, respectivement par 0 et 1. Plus le quadrillage est fin, et plus la description de l'image sera précise ; à partir de quelques millions de pixels l' ?il ne fait plus la différence. Cette représentation est dite matricielle, ou "bitmap". L'idée est donc

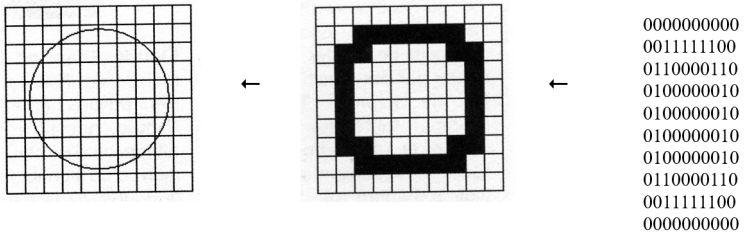
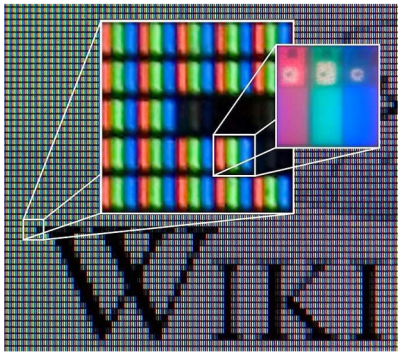


FIGURE 1.5 – Codage bitmap sur 100 pixels (Grille 10 × 10)

de convenir d'un standard pour tous : ici on décrit les pixels de haut en bas et de la droite vers la gauche, et non pas, par exemple, en spirale en partant du centre. Cette dernière méthode n'est pas si idiote qu'elle en a l'air : elle évite à une machine de dessin automatique de faire des grands trajets, et donc gagne du temps. De même il a été convenu que blanc est codé par 0 et noir par 1.
On peut généraliser ces principes aux images en niveaux de gris ou en couleurs.
Sur un écran d'ordinateur, le principe est le même :



Cette image est un agrandissement d'une partie de l'écran, composé d'une grille de pixels. Chaque pixel est composé de trois couleurs. Le premier zoom est un groupe de pixels. Trois d'entre eux sont presque éteints, ils correspondent au haut du "W". Le deuxième zoom montre un pixel unique. Le mélange des trois couleurs primaires, suivant leur intensité respective, donne de nombreuses couleurs.

1.4.2 Formats d'images matricielles sans compression

Le suffixe du fichier permet d'identifier le type d'objet codé dans le fichier, par exemple .txt pour du texte, .py pour un programme Python, .odf pour un document OpenOffice, etc...

i) Les images en noir et blanc peuvent être codés dans le format PBM (Portable Bit Map, suffixe .pbm).
Un fichier pbm ASCII se compose comme suit :

- Un nombre magique (P1)
- Un caractère d'espacement (espace, tabulation, nouvelle ligne)
- Largeur de l'image (codée en caractères ASCII)
- Un caractère d'espacement
- Hauteur de l'image (codée en caractères ASCII)
- Un caractère d'espacement ? Données ASCII de l'image :
- L'image est codée ligne par ligne en partant du haut
- Chaque ligne est codée de gauche à droite
- Un pixel noir est codé par un caractère 1, un pixel blanc est codé par un caractère 0
- Les caractères d'espacement à l'intérieur de cette section sont ignorés
- Aucune ligne ne doit dépasser 70 caractères.
- un zéro final.
- Toutes les lignes commençant par # sont ignorées.

Un exemple du codage de la lettre "J" :

```
P1
# exemple de la lettre "J", format de police 6 points !
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

ii) De la même manière il existe des formats PGM et PPM pour les niveaux de gris et la couleur.
Remarquez que la taille des fichiers devient vite très grande :

- La taille du fichier pour une image de $250 \times 167 = 41750$ pixels en PBM est de 8 Ko.
- En PGM, si l'on code sur $65536 = 2^{16}$ niveaux de gris, il faut deux octets pour co-der chaque pixel : on passe à 45 Ko (ce qui fait bien plus que $8 \times 2 \ldots$)
- En PPM, chaque couleur est codée par 3 octets (un pour le rouge, un pour le vert et le dernier pour le bleu), soit 127 Ko (même remarque : c'est plus que $8 \times 3 \ldots$). Remarquez que les trois couleurs de base ne sont pas les mêmes suivant que l'on imprime (ou dessine) en synthèse soustractive, ou bien que l'on affiche à l'écran. Dans le premier cas (feuille blanche), la synthèse des couleurs est soustractive ; les couleurs de base sont cyan, magenta et jaune. Dans le deuxième cas (écran noir), la synthèse est additive ; le couleurs de base sont rouge, vert, bleu (jouez avec les réglages sur votre écran de télévision).
- D'autres formats sans compression existent. Par exemple, sous le format BMP (windows bitmap) une image en 1200×1600 (1200 pixels de large par 1600 pixels de haut), en 24 bits (16,8 millions de couleurs) aura une taille de $(1200 \times 1600 \times 24)$ bits soit 5,76 Mo (5,5 Mio ; 1 Mio = 2m20 octets). C'est-à-dire sur un dvd de 4,7 Go, il tient 816 images, soit ... 33 secondes de film à 24 images par seconde.

1.4.3 Formats d'images matricielles avec compression

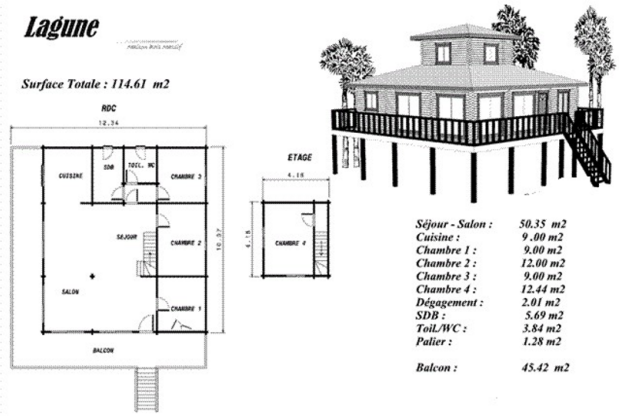
Les exemples précédents montrent tout l'intérêt de la compression de données. Certains formats, comme le PNG (portable network graphic) compressent l'image sans perte d'information : lors de la conversion de l'image du format BMP en PNG, l'image prend moins de place mais est de même qualité à l'ouverture. D'autres, comme le JPEG (joint photographic experts group), sont des formats avec perte d'information, un des paramètres permettant de régler cette perte. Lors du passage de format, la qualité de l'image est plus ou moins dégradée, et elle peut être alors bien moins lourde.

Sur l'image ci-contre, le taux de compression en jpeg augmente de la gauche vers la droite. On peut alors diviser le poids de l'image par 25.

1.4.4 Formats d'images vectorielles

Le format matriciel ne permet pas de satisfaire tous les besoins. Les plans d'architectes, les dessins techniques ont besoin de coder des informations très précises pour les besoins de l'utilisateur. Des logiciels comme OpenOffice Draw, ou Inkscape (libres tous les deux) permettent de faire du dessin vectoriel. Au lieu de préciser pixel par pixel la composition de l'image, le fichier contient la liste des éléments géométriques constitutifs de l'image (cercles, rectangles, segments, courbes de Bézier qui permettent de dessiner des tracés complexes, ?) ainsi que leurs caractéristiques (par exemple les coordonnées du centre d'un cercle, son rayon, ainsi que sa couleur de remplissage). Il est alors nécessaire de transformer l'image en format matriciel pour pouvoir l'afficher à l'écran. Suivant le type de travail à effectuer sur l'image, on choisira soit le format matriciel soit le format vectoriel. On peut augmenter la résolution d'une image vectorielle sans perte de qualité avec le même poids, contrairement à une image matricielle. Mais on ne peut pas dire qu'une image vectorielle nécessite moins de mémoire en général. PDF, Postscript (pour les imprimantes), SVG et Adobe Flash sont des formats vectoriels.

Redimensionnement d'une image :
à gauche en vectoriel, à droite en matriciel



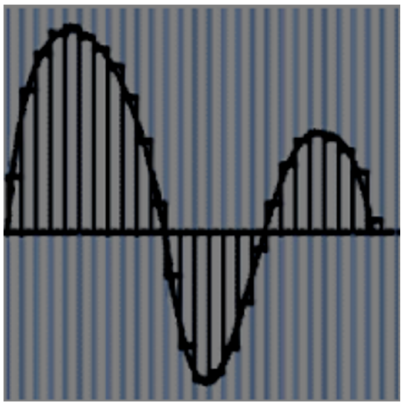
Un plan d'architecte réalisé en vectoriel, et le rendu en 3d

1.5 Codage du son

1.5.1 Le monde réel et le monde virtuel

Les différents types d'appareil

Les phénomènes qui nous entourent sont quasiment tous continus, c'est - à - dire que lorsque ces phénomènes sont quantifiables, ils passent d'une valeur à une autre sans discontinuité. Ainsi, lorsque l'on désire reproduire les valeurs du phénomène, il s'agit de l'enregistrer sur un support, afin de pouvoir l'interpréter pour reproduire le phénomène original de la façon la plus exacte possible.



Lorsque le support physique peut prendre des valeurs continues, on parle d'enregistrement analogique. Par exemple, une cassette vidéo, une cassette audio ou un disque vinyle sont des supports analogiques. Par contre, lorsque le signal ne peut prendre que des valeurs bien dé finies, en nombre limité, on parle alors de signal numérique.
La représentation d'un signal analogique est donc une courbe, tandis qu'un signal numérique pourra être visualisé par un histogramme. De cette façon, il est évident qu'un signal numérique est beaucoup plus facile à reproduire qu'un signal analogique (la copie d'une cassette audio provoque des pertes. . .).

Lorsque le support physique peut prendre des valeurs continues, on parle d'enregistrement analogique. Par exemple, une cassette vidéo, une cassette audio ou un disque vinyle sont des supports analogiques. Par contre, lorsque le signal ne peut prendre que des valeurs bien dé finies, en nombre limité, on parle alors de signal numérique. La représentation d'un signal analogique est donc une courbe, tandis qu'un signal numérique pourra être visualisé par un histogramme. De cette façon, il est évident qu'un signal numérique est beaucoup plus facile à reproduire qu'un signal analogique (la copie d'une cassette audio provoque des pertes. . .).

La numérisation

La transformation d'un signal analogique en signal numérique est appelée numérisation. La numérisation comporte deux activités parallèles : l'échantillonnage (en anglais sampling) et la quantification.
L'échantillonnage consiste à prélever périodiquement des échantillons d'un signal analogique. La quantification consiste à affecter une valeur numérique à chaque échantill on prélevé.
La qualité du signal numérique dépendra de deux facteurs :

- la fréquence d'échantillonnage (appelée taux d'échantillonnage) : plus celle - ci est grande (c'est - à - dire que les échantillons sont relevés à de petits intervalles de temps) plus le s ignal numérique sera fidèle à l'original ;
- le nombre de bits sur lequel on code les valeurs (appelé résolution) : il s'agit en fait du nombre de valeurs différentes qu'un échantillon peut prendre. Plus celui - ci est grand, meilleure est la qualité.

Ainsi, grâce à la numérisation on peut garantir la qualité d'un signal, ou bien la réduire volontairement pour :

- diminuer le coût de stockage
- diminuer le coût de la numérisation
- diminuer les temps de traitement
- tenir compte du nombre de valeurs nécessaires selon l'application
- tenir compte des limitations matérielles

Convertisseurs analogique vers numérique

Un convertisseur analogique numérique (CAN) est un appareil permettant de transformer en valeurs numériques un phénomène variant dans le temps. Lorsque les valeurs numériques peuvent être stockées sous forme binaire (donc par un ordinateur), on parle de données multimédia.
Un ordinateur dit "multimédia" est une machine capable de numériser des documents (papier, audio, vidéo. . .). Les principaux périphéri ques comportant des convertisseurs analogique vers numérique sont :

- la souris, l'écran tactile et tout mécanisme de pointage (tablette graphique, trackball, touchpad, . . .)
- les scanners
- les cartes de capture sonore (la quasi - totalité des cartes son)
- les cartes d'acquisition vidéo
- les lecteurs (optiques comme le lecteur de CD - ROM, magnétiques comme le disque dur)
- les modems (à la réception)

Convertisseurs numérique vers analogique

Les convertisseurs numérique / analogique permettent de restituer un signa l numérique en signal analogique. En effet, si une donnée numérique est plus facile à stocker et à manipuler, il faut tout de même pouvoir l'exploiter. A quoi servirait un son numérique si l'on ne pouvait pas l'entendre. . .
Ainsi, sur un ordinateur multimédia, on trouve des convertisseurs numérique vers analogique pour la plupart des sorties :

- sorties audio des cartes - sons
- synthétiseur musical
- imprimante
- écran
- modem (à l'émission)
- graveurs CD et DVD

1.5.2 Le son

Définition

Le son est une vibration de l'air, c'est-à-dire une suite de surpressions et de dépressions de l'air par rapport à une moyenne, qui est la pression atmosphérique. D'ailleurs, pour s'en convaincre, il suffit de placer un objet bruyant (un réveil par exemple) dans une cloche à vide pour s'apercevoir que l'objet initialement bruyant n'émet plus un seul son dès qu'il n'est plus entouré d'air !
La façon la plus simple de reproduire un son actuellement est de faire vibrer un objet. De cette façon, un violon émet un son lorsque l'archet fait vibrer ses cordes, un piano émet une note lorsque l'on frappe une touche, car un marteau vient frapper une corde et la fait vibrer.
Pour reproduire des sons, on utilise généralement des haut - parleurs. Il s'agit en fait d'une membrane reliée à un électroaimant, qui, suivant les sollicitations d'un courant électrique va aller en avant et en arrière très rapidement, ce qui provoque une vibration de l'air situé devant lui, c'est-à-dire du son !
La numérisation des sons utilise le codage PCM Pulse Coded Modulation , initialement utilisé pour coder la voix dans le réseau téléphonique.

L'échantillonnage des sons

Pour pouvoir représenter un son sur un ordinateur, il faut arriver à le convertir en valeurs numériques, car celui - ci ne sait travailler que sur ce type de valeurs. Il s'agit donc de relever des petits échantillons de son (ce qui revient à relever des différences de pression) à des intervalles de temps précis. On appelle cette action l'échantillonnage ou la numérisation du son.
L'intervalle de temps entre deux échantillons est appelé taux d'échantillonnage. Etant donné que, pour arriver à restituer un son qui semble continu à l'oreille, il faut des échantillons tous les 10 000 èmes de seconde, il est plus pratique de raisonner sur le nombre d'échantillons par seconde, exprimés en Hertz (Hz). On parlera donc tout aussi bien de taux d'échantillonnage que de fréquence d'échantillonnage.
Il existe un certain nombre de fréquences d'échantillonnage normalisées :

Taux d'échantillonnage	**Qualité du son**
48000 Hz ou 48 kHz	pour les enregistreurs numériques multipistes professionnels et l'enregistrement grand public (DAT, Mini disc . . .)
44 100 Hz ou 44,1 kHz	qualité CD
32000 Hz ou 32 kHz	pour la radio FM en numérique
22 000 Hz ou 22 kHz	qualité radio
8 000 Hz ou 8 kHz	qualité téléphone

La valeur du taux d'échantillonnage, pour un CD audio par exemple, n'est pas arbitraire, elle découle en réalité du théorème de Shannon. La fréquence d'échantillonnage doit être suffisamment grande, afin de préserver la forme du signal. Le théorème de Nyquist - Shannon stipule que la fréquence d'échantillonnage doit être égale ou supérieure au double de la fréquence maximale contenue dans ce signal. Notre orei lle perçoit les sons environ jusqu'à 22 000 Hz, il faut donc une fréquence d'échantillonnage au moins de l'ordre de 44 000 Hz pour obtenir une qualité satisfaisante.

Mémoire requise pour stocker un son

Lorsque l'on travaille avec un ordinateur, les informations en cours de traitement sont stockées dans la mémoire vive. Lorsque vous le souhaitez, vous les enregistrez sur le disque dur, ou sur un autre support, sous la forme d'un fichier.
Il faut donc avoir conscience de la capacité de votre ordinateur, pour comprendre ses limites.
Il est simple de calculer la taille d'une séquence sonore non compressée. En effet, en connaissant le nombre de bits sur lequel est codé un échantillon, on connaît la taille de celui - ci (la taille d'un échantillon est le nombre de bits...). Le nombre d'octets correspondant est facile à calculer, car 8 bits = 1 octet. Ainsi, si le codage de chaque échantillon se fait sur 16 bits, chaque échantillon occupera 2 octets.
Le taux d'échantillonnage permet de connaître le poids de chaque seconde d'enregistrement :
Si on travaille à un taux de 44,1 kHz (sur 16 bits) , on aura 44100 échantillons $\rightarrow$ 2 octets = 88200 octets, soit 88 ko.
Il faut alors connaître la durée de la séquence sonore pour connaître le poids d'une voie : 10 secondes (à 44,1kHz, sur 16 bits) = 10 x 88 ko = 880 ko
Enfin, il faut connaître le nombre de voies. Si le son est stéréo, il faut enregistrer deux voies, soit 1660 ko, pour 10 secondes d'enregistrement audio numérique (à 44,1kHz, sur 16 bits).
La taille T en octets d'une séquence sonore est ainsi égale à :

$$\textbf{T = Taux déchantillonnage} \times \textbf{Nombre de bits} \div 8 \times \textbf{Nombre de secondes} \times \textbf{Nombre de voies}$$

Dans cette qualité d'enregistrement, un cédé audio (700 Mo) peut stocker théoriquement un peu moins de 70 minutes de musique.

1.5.3 Les formats audio numériques

Les sons numériques peuvent être encodés dans de nombreux format, certains permettant de les compresser, d'autre non. Sans être exhaustif, on distingue ainsi les formats :

WAV : qui est un conteneur, généralement utilisé pour deux sous formats :

- Le WAV PCM, format brut sans compression et donc très lourd.
- Le format RIFF - WAV, contenant généralement un fichier mp3 ou WAV, avec matadonnées, mais très peu utilisé

.

MPEG, MP3 : : destiné à à l'encodage vidéo. Il existe plusieurs versions de MPEG, dont le MPEG-1, MPEG-2 et le MPEG-4.
Le MPEG-1 est subdivis é en plusieurs couches (layers in english). Chacune des couches permet un travail complémentaire des autres pour encoder des vidéos. La couche 3 permet entre autre l'encodage de la bande son de la vidéo. On parle donc de MPEG-1 couche 3, le fameux MP3. Le taux de compression du codec MP3 par rapport au format brut WAV PCM est de l'ordre de 80 % en moyenne sans perte de qualité audible.

Ogg Vorbis : est un conteneur, format de fichier capable de contenir divers données

WMA Ce format constitue la variante Microsoft de la norme MPEG. La qualité des fichiers WMA étant très semblable à celle des MP3, Microsoft a dû recourir à des astuces plutôt douteuses pour promouvoir son format propriétaire. En effet, Windows Media Player encode les MP3 avec une qualité inférieure aux fichiers WMA afin de faire croire à l'utilisateur que la qualité est meilleure avec son format.

AAC : AAC (Audio Advanced Coding) est un format de compression de donné es audio développé par l'institut Fraunhofer en partenariat avec AT&T, Nokia, Sony et Dolby. AAC est un codec audio basé sur la norme Mpeg4, d'où son surnom MP4. Ce nouveau standard serait vu comme le successeur du célèbre MP3.

1.6 Codes détecteurs/correcteurs d'erreurs

Lorsque deux personnes communiquent entre elles, il arrive souvent que le message reçu diffère de celui émis ; que ce soit à cause de fautes d'orthographes, d'une diction trop rapide ou bien de bruits perturbateurs. Toutefois, il n'est pas toujours nécessaire d'obtenir l'intégralité du message pour en deviner le sens. Ainsi, même si on mélange l'ordre des lettres dans un mot tout en laissant les lettres aux extrémités à leur place, on puet qnaud mmêe cmopnredre la prshae.

On constate que le transfert d'informations numériques est très important dans de nombreux domaines, par exemple pour les sondes spatiales qui reçoivent des signaux parcourant une énorme distance. Il est donc primordial de s'assurer que la perte d'informations soit minimale : c'est là qu'interviennent les codes correcteurs.

L'objectif est de rajouter une couche d'informations au message initial qui ne rajoute pas de sens mais qui permettront de détecter les erreurs et de les corriger. Bien évidemment, l'ajout d'informations a un prix : l'efficacité d'un code sera donc déterminée par la quantité d'informations ajoutée et par sa capacité à conserver l'information d'un message. Nous allons nous pencher sur les principaux codes correcteurs binaires et voir comment se déroule le codage et la correction d'erreurs, notamment pour les codes cycliques.

Un code correcteur est une technique de codage basée sur la redondance. Elle est destinée à corriger les erreurs de transmission d'un message sur une voie de communication peu fiable.
La théorie des codes correcteurs ne se limite pas qu'aux communications classiques (radio, câble coaxial, fibre optique, etc.) mais également aux supports de stockage comme les disques compacts, la mémoire RAM et d'autres applications où l'intégrité des données est importante. Comment détecter et/ou corriger des erreurs ?

Comment détecter et/ou corriger des erreurs ?

On peut transmettre un nombre soit en chiffres, soit en lettres :

1. On envoie "0324614103". S'il y a des erreurs de transmission, par exemple si je reçois "0323614203", je ne peux pas les détecter.
2. On envoie "zéro trente-deux quatre cent soixante et un quarante et un zéro trois". S'il y a des erreurs de transmission, par exemple si je reçois "zérb trente-deu quate cent soixante en un quaranhe et on zéro tros", je suis capable de corriger les erreurs et de retrouver le bon numéro.

Dans le premier cas, l'information est la plus concise possible. Dans le deuxième cas au contraire, le message contient plus d'informations que nécessaire. C'est cette redondance qui permet la détection et la correction d'erreurs.

Pourquoi ces codes ?

- Des canaux de transmission imparfaits entraînant des erreurs lors des échanges de données.
- La probabilité d'erreur sur une ligne téléphonique est de 10^{-7} (cela peut même atteindre 10^{-4}). Avec un taux d'erreur de 10^{-6} et une connexion à 1 Mo/s, en moyenne 8 bits erronés sont transmis chaque seconde. . .

Principe général

- Chaque suite de bits à transmettre est augmentée par une autre suite de bits dite "de redondance" ou "de contrôle".
- Pour chaque suite de k bits transmise, on ajoute r bits. On dit alors que l'on utilise un code C(n, k) avec n = k + r .
- À la réception, les bits ajoutés permettent d'effectuer des contrôles.

1.6.1 La distance de mingming

La distance de Hamming doit son nom à Richard Hamming. Elle est décrite dans un article fondateur pour la théorie des codes. Elle est utilisée en télécommunication pour compter le nombre de bits altérés dans la transmission d'un message d'une longueur donnée.

Exemple :

La distance de Hamming entre 1011101 et 1001001 est 2 car deux bits sont différents.
Il est souhaitable d'avoir une certaine distance entre les mots envoyés, afin de détecter s'il y a eu une erreur de transmission. Par exemple, si l'on a trois messages à transmettre de trois bits, il vaut mieux choisir les codages qui sont à distance 2 les uns des autres, par exemple 000, 110 et 101. En effet, si un seul bit est altéré, on recevra un message impossible. Par contre, en utilisant 000, 001 et 010, un bit altéré pourrait passer inaperçu.

1.6.2 Somme de contrôle (Checksum)

La somme de contrôle (en anglais "checksum") est un cas particulier de contrôle par redondance. Elle permet de détecter les erreurs, mais pas forcément de les corriger. Le principe est d'ajouter aux données des éléments dépendant de ces dernières et simples à calculer. Cette redondance accompagne les données lors d'une transmission ou bien lors du stockage sur un support quelconque. Plus tard, il est possible de réaliser la même opération sur les données et de comparer le résultat à la somme de contrôle originale, et ainsi conclure sur la corruption potentielle du message.

Bit de parité

Transmettons sept bits auxquels viendra s'ajouter un bit de parité. On peut définir le bit de parité comme étant égal à zéro si la somme des autres bits est paire et à un dans le cas contraire. Si la somme des bits est impaire, c'est qu'il y a eu une erreur de transmission.

Exemple :

1010001 (7 bits) devient 10100011 (8 bits)
Cette approche permet de détecter un nombre impairs d'erreurs, mais un nombre pair d'erreurs passera inaperçu.

1.6.3 Le code ISBN

L'ISBN (International Standard Book Number) est un numéro international qui permet d'identifier, de manière unique, chaque livre publié. Il est destiné à simplifier la gestion informatique des livres dans les bibliothèques, librairies, etc.

ISBN-10: 2-1234-5680-2
ISBN-13: 978-2-1234-5680-3

Le numéro ISBN-10 se compose de quatre segments, trois segments de longueur variable et un segment de longueur fixe, la longueur totale de l'ISBN comprend dix chiffres (le 1er janvier 2007, la longueur a été étendue à 13 chiffres en ajoutant un groupe initial de 3 chiffres).

Si les quatre segments d'un ancien code ISBN à 10 chiffres sont notés A - B - C - D :

- A identifie un groupe de codes pour un pays, une zone géographique ou une zone de langue.
- B identifie l'éditeur de la publication.
- C correspond au numéro d'ordre de l'ouvrage chez l'éditeur.
- D est un chiffre-clé calculé à partir des chiffres précédents et qui permet de vérifier qu'il n'y a pas d'erreurs. Outre les chiffres de 0 à 9, cette clé de contrôle peut prendre la valeur X, qui représente le nombre 10.

Calcul du chiffre-clé d'un numéro ISBN-10

- On attribue une pondération à chaque position (de 10 à 2 en allant en sens décroissant) et on fait la somme des produits ainsi obtenus.
- On conserve le reste de la division euclidienne de ce nombre par 11. La clé s'obtient en retranchant ce nombre à 11. Si le reste de la division euclidienne est 0, la clé de contrôle n'est pas 11 (11 - 0 = 11) mais 0.
- De même, si le reste de la division euclidienne est 1, la clé de contrôle n'est pas 10 mais la lettre X.

Le nombre 11 étant premier, une erreur portant sur un chiffre entraînera automatiquement une incohérence du code de contrôle. La vérification du code de contrôle peut se faire en effectuant le même calcul sur le code ISBN complet, en appliquant la pondération 1 au dixième chiffre de la clé de contrôle (si ce chiffre-clé est X, on lui attribue la valeur 10) : la somme pondérée doit alors être un multiple de 11.

Exemple :

Pour le numéro ISBN (à 9 chiffres) 2-940043-41, quelle est la clé de contrôle ?
La somme des produits est **179**, dont le reste de la division euclidienne par 11 est **3**.
La clé de contrôle est donc 11 - 3 = **8**. L'ISBN au complet est : 2-940043-41-**8**.
La vérification de la clé complète à 10 chiffres donne la somme pondérée 179 + 8 = 187, qui est bien un multiple de 11.

1.6.4 Code de Hamming

Un code de Hamming permet la **détection** et la **correction** automatique d'une erreur si elle ne porte que sur un bit du message. Un code de Hamming est parfait, ce qui signifie que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.

Structure d'un code de Hamming

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

- Les m bits du message à transmettre et les n bits de contrôle de parité.
- Longueur totale : $2^n - 1$
- Longueur du message : $m = (2^n - 1) - n$
- On parle de code $x - y$: x est la longueur totale du code (n+m) et y la longueur du message (m)
- Les bits de contrôle de parité Ci sont en position 2^i pour i = 0, 1, 2, ...
- Les bits du message D_j occupe le reste du message.

Exemples de code de Hamming

- Un mot de code 7-4 a un coefficient d'efficacité de 4/7 = 57%
- Un mot de code 15-11 a un coefficient d'efficacité de 11/15 = 73%
- Un mot de code 31-26 a un coefficient d'efficacité de 26/31 = 83 %

Détection d'erreur dans un code de Hamming

	7	6	5	4	3	2	1
7	6	5	4	3	2	1	
1	0	1		0			
C_0	1	0	1	0	1	0	1

Si les bits de contrôle de parité C_2, C_1, C_0 ont tous la bonne valeur, il n'
a pas d'erreur ; sinon la valeur des bits de contrôle indique la position d
l'erreur entre 1 et 7.
Le code de Hamming présenté ici ne permet de retrouver et corrige
qu'une erreur.
Pour savoir quels bits sont vérifiés par chacun des bits de contrôle d
parité, on peut construire le tableau ci-après.

- La première ligne indique la position des bits.
- Dans chaque colonne, on écrit le numéro de la position verticalement en binaire.
- Les 1 indiquent alors quels bits sont vérifiés. Ainsi :
  - C_2 vérifie les bits 4, 5, 6, 7. La somme de ces bits doit être paire.
  - C_1 vérifie les bits 2, 3, 6, 7. La somme de ces bits doit être paire.
  - C_0 vérifie les bits 1, 3, 5, 7. La somme de ces bits doit être paire.

Exemples d'un code de Hamming 7-4

On souhaite envoyer le message 1010.
Complétons le mot de Hamming correspondant :

7	6	5	4	3	2	1
1	0	1		0		

C_0 vaut 0 pour pouvoir rendre pair 1+1+0 (les bits d'indices 7, 5, 3).
C_1 vaut 1 pour pouvoir rendre pair 1+0+0 (les bits d'indices 7, 6, 3).
C_2 vaut 0 pour pouvoir rendre pair 1+0+1 (les bits d'indices 7, 6, 5).

7	6	5	4	3	2	1
1	0	1	**0**	0	**1**	**0**

Imaginons que l'on reçoive le mot 0010010 (le bit de poids fort a été altéré).
C_0 a la mauvaise valeur, car 0+1+0+0 est impair, donc il y a une erreur en position 7, 5, 3 ou 1.
C_1 a la mauvaise valeur, car 0+0+0+1 est impair, donc il y a une erreur en position 7, 6, 3 ou 2.
C_2 a la mauvaise valeur, car 0+0+1+0 est impair, donc il y a une erreur en position 7, 6, 5 ou 4.
Écrivons le nombre binaire $C_2C_1C_0$ où Ci vaut 0 si le bit de contrôle Ci a la bonne valeur et 1 sinon. On obtient ici 111, ce qui correspond à 7 en binaire. Le bit erroné est le numéro 7. Que se passe-t-il si c'est un des bits de contrôle qui est altéré ?
Imaginons que l'on ait reçu 1010011 (cette fois-ci, c'est le bit de poids faible qui a été altéré). C_0 a la mauvaise valeur, car

1+1+0+1 est impair. Il y a une erreur en position 7, 5, 3 ou 1.
C_1 a la bonne valeur, car 1+0+0+1 est pair. Il n'y a pas d'erreur en position 7, 6, 3 et 2
C_2 a la bonne valeur, car 1+0+1+0 est pair. Il n'y a pas d'erreur en position 7, 6, 5 et 4.
Ici, $C_2C_1C_0$ vaut 001. Le bit erroné est donc le numéro 1.

Chapitre 2

Algèbre de Boole et fonctions logiques

Introduction

Le binaire permet de représenter facilement l'état logique d'un système technique ou de ses entrées-sorties. C'est une logique à deux états :

- Un interrupteur est ouvert ou non ouvert (fermé)
- Une lampe est allumée ou non allumée (éteinte)
- Une tension est élevée ou pas élevée (faible)
- Une pression est présente ou pas présente (absente).

L'algèbre binaire encore appelée algèbre de Boole est un outil mathématique résultant des travaux de Georges BOOLE (1815-1864), mathématicien autodidacte anglais qui voulait faire un lien entre la logique (étude de la validité du raisonnement) et la représentation symbolique utilisée en mathématique.
Il a écrit deux ouvrages sur le sujet :

- Mathematical Analysis of Logic (1847)
- An Investigation of the Laws of Thought (1854)

Ces travaux n'ont pas connu d'intérêt particulier auprès de la communauté mathématique et scientifique de son époque, mis à part chez les logiciens.
C'est 70 ans plus tard que les travaux de Boole gagnent l'intérêt de tous, lorsque Claude Shannon fait le lien entre l'algèbre de Boole et la conception des circuits. Claude Shannon, dans des travaux publiés en 1938, montre que l'algèbre de Boole peut être utilisée pour optimiser les circuits. Cette nouvelle avenue de recherche va ouvrir la voie à l'ère numérique.
En utilisant l'algèbre de Boole avec le système binaire, on peut concevoir des circuits capables d'effectuer des opérations arithmétiques et logiques. L'algèbre de Boole repose sur des axiomes, des postulats et des théorèmes qu'il faut connaître par coeur !

2.1 Opérations logiques et représentation graphique

2.1.1 Définitions

Variable logique

Une variable logique, noté x est une grandeur qui ne peut prendre que deux valeurs appartenant à $\mathcal{B} = \{0, 1\}$, ("VRAI" ou "FAUX ") représentant ainsi **l'état logique** d'un système bistable générateur de la variable physique.

Les deux états d'une variable booléenne peuvent être associés à deux niveaux de tension : V(0) et V(1) pour les états 0 et 1 respectivement. On distingue les logiques positive et négative selon que $V(1) > V(0)$ ou $V(1) < V(0)$. La table nous donne un résumé de la signification logique des niveaux physiques : Dans la suite de ce cours, nous adopterons la logique positive.

Niveau	Logique positive	Logique négative
H	1	0
L	0	1

TABLE 2.1 – Logique positive vs Logique négative

Fonction logique

Une **fonction booléenne** est une fonction qui a pour arguments n variables logiques $\{x_1, \cdots, x_n\}$ et qui renvoie (comme chacune de ses variables) une valeur booléenne. C'est une expression de variables et d'opérateurs logiques. Ainsi peut-on la définir comme suit :

$$f : \mathcal{B} \times \mathcal{B} \times \cdots \times \mathcal{B} \to \mathcal{B}$$

$$\{x_1, \cdots, x_n\} \longmapsto \{0, 1\}$$

Une fonction logique a donc des variables en entrée et dés variables en sortie.

On appellera **variables d'entrée** celles sur lesquelles on peut agir directement. Ce sont des variables logiques indépendantes.
A l'opposée, les **variables de sortie** sont celles contenant l'état de la fonction après l'évaluation des opérateurs logiques sur les variables d'entrée.

Question : Combien existe-t-il de fonctions booléennes différentes à 1 argument ?
Réponse : 4, comme le montre le tableau 2.2 dans lequel f1 et f4 sont des constantes, f2 représente l'identité et f3 la négation.

x	f1	f2	f3	f4
0	0	0	1	1
1	0	1	0	1

TABLE 2.2 – Les fonctions booléennes à un argument existant

Question : Combien existe-t-il de fonctions booléennes différentes à 2 arguments ? **Réponse** : 16
Plus généralement : combien existe-t-il de fonctions booléennes différentes à n arguments ? **Réponse** :2^{2^n}.

Table de vérité

C'est un tableau qui donne l'état de la sortie en fonction des différentes combinaisons d'états de ses variables d'entrée. Chacune des combinaisons des variables d'entrée est écrite sur une ligne différente.
Si n définit le nombre de variables d'entrée, la table de vérité comportera 2^n combinaisons différentes.

Chronogramme

Un chronogramme est une représentation graphique qui permet de visualiser, en fonction du temps, l'état de la sortie correspondant aux différentes combinaisons d'états logiques des entrées.

Logigramme

Un logigramme est la traduction graphique d'une équation logique utilisant les symboles normalisés des opérateurs.

2.1.2 Opérateurs logiques élémentaires (de base)

Les opérateurs logiques élémentaires servent à construire les fonctions logiques. A chaque opérateur logique élémentaire, correspond une représentation graphique appelée **porte logique**.
En algèbre classique on distingue quatre opérateurs de base : +, -, *, /.
En algèbre de Boole, on définit trois opérateurs logiques effectuant sur les variables les trois opérations fondamentales : d'inversion ou complément ($\neg$ appelé **NON**), d'addition ou union ($\vee$ appelé **OU**), de multiplication ou d'intersection($\wedge$ appelé

ET).

En technologie électronique :

- les variables logiques sont généralement des signaux " bi-tension ",
- les opérateurs logiques sont des circuits électroniques appelés "portes logiques".

Opérateur OU (inclusif)

L'opération OU (OR), encore appelée addition logique, a au moins deux entrées. La sortie d'une fonction OU est dans l'état 1 si au moins une de ses entrées est dans l'état 1.
La fonction OU, notée +, est représentée par le symbole indiqué sur la figure 2.1 et est définie par la table 2.3 :

A	B	S = A+B
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 2.3 – Table de vérité "OU"

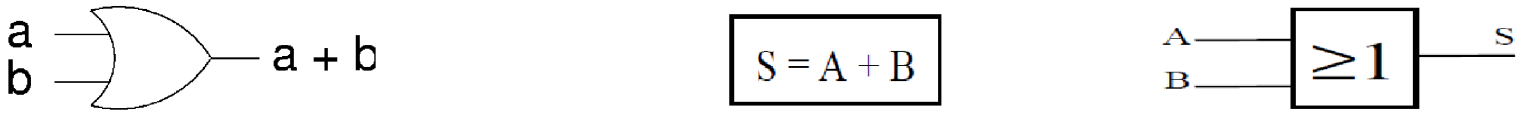


FIGURE 2.1 – Symboles du OU logique

Opérateur ET

L'opération ET (AND), encore dénommée produit logique ou intersection, a au moins deux entrées. La sortie d'une fonction AND est dans l'état 1 si et seulement si toutes ses entrées sont dans l'état 1. La fonction ET, notée "**.**", est représentée par le symbole indiqué sur la figure 2.2 et est définie par la table 2.4 :

A	B	S = A.B
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 2.4 – Table de vérité "ET"

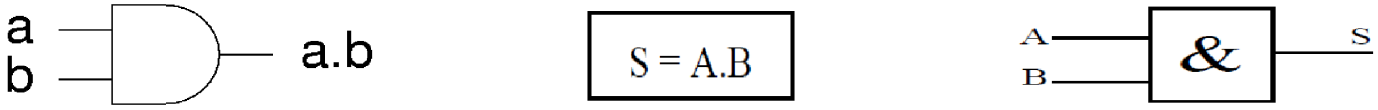


FIGURE 2.2 – Symbole du ET logique

Opérateur NON

L'opération NON (NOT) a une seule entrée et une seule sortie. La sortie d'une fonction NON prend l'état 1 si et seulement si son entrée est dans l'état 0. La négation logique est symbolisée par un petit cercle dessiné à l'endroit où une ligne en entrée ou en sortie rejoint un symbole logique, comme par exemple sur la figure 2.3. La table 2.5 donne la table de vérité correspondante.

A	Y = $\bar{A}$
0	1
1	0

TABLE 2.5 – Table de vérité "NON"

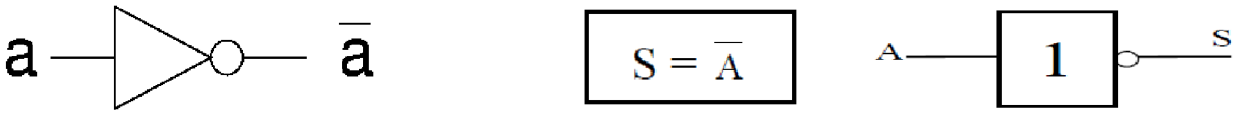


FIGURE 2.3 – Symboles du NON logique

2.1.3 Opérateurs logiques complets

On montre qu'il est possible de synthétiser les trois opérateurs de base avec un seul type d'opérateur que l'on appelle alors "**opérateur complet**" ou "**opérateur universel**", auxquels on associe aussi des portes logiques. il en existe deux :

- le NON-ET ou **NAND**
- le NON-OU ou **NOR**.

Opérateurs universels "NON ET" et "NON OU"

Une porte NON ET (NAND : NOT AND) est constituée par un inverseur à la sortie d'une porte ET (fig 2.4 et fig 2.5) et une négation à la sortie d'une porte OU constitue une fonction NON OU (NOR : NOT OR) symbolisée sur les figures fig 2.6 et fig 2.7.
Leurs tables de vérité respectives correspondent à celle de la table 2.6 :

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

TABLE 2.6 – Table de vérité "NAND" et "NOR"

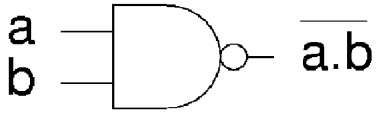


FIGURE 2.4 – Symbole du NAND

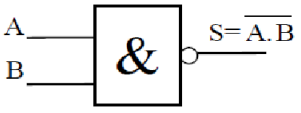


FIGURE 2.5 – Autre représentation

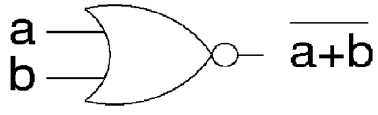


FIGURE 2.6 – Symbole du NOR

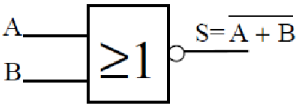


FIGURE 2.7 – Autre représentation

Aux opérateurs précédents (NON, OU , ET, NAND et NOR), on adjoint souvent un autre opérateur appelé **OU exclusif** ou **XOR**.

Opérateur OU exclusif

La sortie d'une fonction OU exclusif (XOR) a deux entrées est dans l'état 1 si une entrée et seulement une est dans l'état 1. C'est dire que le complément de la fonction OU exclusif est la fonction identité (a = b).

La représentation symbolique d'une fonction XOR (notée $\oplus$) est donnée sur la figure 2.8 et sa table de vérité correspond à la table 2.7 :
A ces opérateurs logiques de base, correspondent des circuits réalisant la fonction XOR à partir de portes OR et AND.

A	**B**	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 2.7 – Table de vérité "OU exclusif"

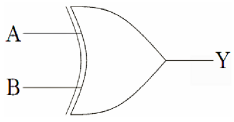


FIGURE 2.8 – XOR anglosaxon

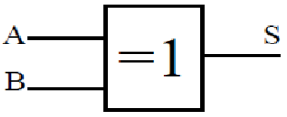


FIGURE 2.9 – XOR français

Grâce aux lois de De Morgan il est possible de réaliser des systèmes logiques avec uniquement des portes NAND ou NOR. Ainsi :

$$A \wedge B = \neg(\neg A \vee \neg B) \text{ et } A \vee B = \neg(\neg A \wedge \neg B)$$

qui peut encore s'écrire :

$$A \bullet B = \overline{\overline{A} + \overline{B}} \text{ et } A + B = \overline{\overline{A} \bullet \overline{B}}$$

D'autre part :

- XOR est associatif
- $s = a \oplus b \oplus c \cdots$ vaut 1 si le nombre de variable à 1 est impair.
- $s = \overline{a \oplus b} = \overline{a} \oplus b = a \oplus \overline{b} = aXNORb$
- $XNOR = \overline{XOR}$ vaut 1 si $a = b$
- $a \oplus 1 = \overline{a}\ a \oplus 0 = a$

Par ailleurs, le XOR a les propriétés suivantes :

- $a \oplus c = b \oplus c \Leftrightarrow a = b$
- $a \oplus x = b \Leftrightarrow x = a \oplus b$

2.2 Axiomes de l'algèbre de Boole

Les règles présentées ci-après (Cf. tableaux 2.8 et 2.9)sont utilisées pour simplifier les expressions logiques et, de ce fait, les circuits logiques dont la schématisation est appelée **logigramme**).

2.2.1 Propriétés portant sur une variable

Du tableau 2.8, on déduit que l'élément neutre pour l'opérateur OU (resp. ET) est 0 (resp. 1) tandis que l'élément absorbant pour l'opérateur OU (resp. ET) est 1 (resp. 0).

2.2.2 Propriétés portant sur plusieurs variables

Le tableau 2.9 présente les différents axiomes applicables quand on est en présence de plus d'une variable.

Règle	Opérateur OU	Opérateur ET
Involution	$\overline{\overline{A}} = A$	$\overline{\overline{A}} = A$
Idempotence	A + A = A	$A.A = A$
Complémentarité	$A + \overline{A} = 1$	$A.\overline{A} = 0$
Elément neutre	$A + 0 = A$	$A.1 = A$
Elément absorbant	$A + 1 = 1$	$A.0 = 0$

TABLE 2.8 – Axiomes de l'algèbre de Boole sur une variable

Règle	Opérateur OU	Opérateur ET
Commutativité	$A + B = B + A$	$A.B = B.A$
Associativité	$(A + B) + C = A + (B + C)$ = A + B + C	$(A.B).C = A.(B.C)$ = A.B.C
Distributivité	ET par rapport à OU	$A.(B + C) = (A.B) + (A.C)$
	OU par rapport à ET	$A + (B.C) = (A + B).(A + C)$
Absorption	$A + \overline{A}.B = A + B$	$A.(A + B) = A$
	$A.B + \overline{A}.C + B.C = A.B + \overline{A}.C$	$A.(\overline{A} + B) = A.B$
Absorption	$A + \overline{A}.B = A + B$	$A.(A + B) = A$
	$A.B + \overline{A}.C + B.C = A.B + \overline{A}.C$	$A.(\overline{A} + B) = A.B$

TABLE 2.9 – Axiomes de l'algèbre de Boole sur plusieurs variables

2.3 Théorèmes fondamentaux

2.3.1 Théorèmes de De Morgan

Ce théorème est une application du théorème de Shanon qui s'écrit :

$$\overline{f(a, b, \cdots, +, \bullet)} = f(\overline{a}, \overline{b}, \cdots, \bullet, +)$$

Autrement dit, le complément d'une fonction f de variables $a, b, \cdots$ s'obtient en remplaçant les variables par leur complément, et les opérateurs ET (.) par des OU (+) (et réciproquement). **Exemple 1.**

Si $f = ac + \overline{c}d + cd$, alors son complément n'est autre que $\overline{f} = (\overline{c} + \overline{c}).(c + \overline{d}).(\overline{c} + \overline{d})$

Le théorème de DE MORGAN permet ainsi de créer des liens entre les formes complémentées et des formes plus simples et peut s'énoncer comme suit :

Le complément d'une somme logique est égal au produit du complément de chacun des termes. De même, le complément d'un produit logique est égal à la somme du complément de chacun des termes.

Ce qui se traduit par les formules suivantes, appelées théorème de De Morgan :

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

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

Le théorème de DE MORGAN permet de simplifier les équations booléennes. Cependant,il ne permet pas forcément de traiter rapidement des cas compliqués. On est donc contraint d'utiliser une autre méthode dont la puissance est bien supérieure, le tableau de KARNAUGH, que nous verrons un peu plus loin.

2.3.2 Dualité

Une expression reste vraie si on intervertit les 1 par des 0 et les ET par des OU à la fois. **Exemple 2.**

Si $a + b = 1$, alors $\overline{a + b} = (\overline{a}.\overline{b}) = 0$

On pourrait aussi dire "Je suis riche si je suis bien payé et que je ne dépense pas tout mon argent = Je suis pauvre si je ne suis pas bien payé ou que je dépense tout mon argent".

2.4 Formes normales d'une fonction logique

Quand on parle de fonction logique, on parle souvent de la forme correspondant à l'expression logique. 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").
Si de plus, toutes les variables apparaissent dans chaque terme, on obtient une **forme normale canonique**.

Il existe deux formes normales canoniques et on montre qu'elles sont équivalentes :

- La forme normale disjonctive
- La forme normale conjonctive

En général, on choisit la forme normale qui donne le résultat le plus simple (peu de 0 dans le cas de la 2ème forme ou peu de 1 pour la 1ère). Si la fonction n'est pas sous forme normale i.e. une des variables (au moins) ne figure pas dans un des termes, la fonction est sous une forme simplifiée.

2.4.1 Définition

Soit f une fonction logique de n variables.

Minterm

On appelle **Minterm** tout produit des variables de la fonction booléenne ou de leur complément (négation). **Exemple 3.**
Pour n = 4 variables $\{a, b, c, d\}$,

- $m = a.b.c.d$ est un minterm
- $m = \overline{a}.\overline{b}.c.d$ est un autre minterm
- $m = a.\overline{b}.c$ n'est pas un minterm

Maxterm

On appelle **Maxterm** toute somme des variables de la fonction booléenne ou de leur complément. **Exemple 4.**
Pour n = 4 variables $\{a, b, c, d\}$,

- $m = a + b + c + d$ est un maxterm
- $m = \overline{a} + \overline{b} + c + d$ est un autre maxterm
- $m = a + \overline{b} + c$ n'est pas un maxterm

2.4.2 Forme normale disjonctive

Théorème $N^0 1$ de Shanon :
Toute fonction logique peut se décomposer en une somme de produits 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 une somme 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".
Pratiquement, la forme normale disjonctive s'obtient en faisant la somme des minterms là où la fonction vaut 1.

2.4.3 Forme normale conjonctive

Théorème $N^0 2$ de Shanon :
Toute fonction logique peut se décomposer en un produit de sommes logiques :

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

Cette forme est la duale de la Forme normale disjonctive.
Comme précédemment l'utilisation successive de ce théorème permet d'arriver à la forme normale conjonctive : En pratique, on cherche les cases à 0, puis on fait le produit des combinaisons de variables qui leur "correspondent" en prenant garde que la "correspondance" n'est pas la même que pour la forme normale disjonctive, d'où l'expression : "Développement suivant les 0". On fait donc le produit des maxterms là où la fonction vaut 0.

2.4.4 Passage aux formes canoniques

Pour passer à une forme normale canonique, on part de la fonction et on la transforme pour faire apparaître des minterms/maxterms complets. On s'appuie pour cela sur les propriétés de l'algèbre de Boole, et notamment des invariants : $x.\overline{x} = 0$ et $x + \overline{x} = 1$ **Exemple 5.**
Soit $f(a,b,c) = ab + \overline{b}c + a\overline{c}$ une fonction dont on désire obtenir les formes normales disjonctive et conjonctive.

Forme normale disjonctive canonique :
Le premier minterm est ab, il lui manque la variable c ; on a :
$ab = ab(c + \overline{c})$ car $c + \overline{c} = 1$. On fait la même chose pour les deux autres minterms. D'où :

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

Forme normale conjonctive canonique :
En passant par $\overline{\overline{x}} = x$, après développement, on obtient :
$\overline{f(a,b,c)} = \overline{a}b + \overline{a}bc + \overline{a}\overline{c} + \overline{a}\overline{b}\overline{c}$. Reste à transformer les mintermes à 2 variables :
$\overline{a}b + \overline{a}\overline{c} = \overline{a}b(c+\overline{c}) + \overline{a}\overline{c}(b+\overline{b})$.
Au final, $\overline{f(a,b,c)} = \overline{a}bc + \overline{a}\overline{b}\overline{c} + \overline{a}b\overline{c}$ et par conséquent :

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

2.5 Simplification des fonctions logiques

On peut combiner plusieurs fonctions logiques pour réaliser des fonctions plus complexes.
Il y a plusieurs expressions logiques différentes (dans leur formulation) pouvant être égales (logiquement) entre elles. Or, quand on réalise un circuit électronique, il est souhaitable d'utiliser l'expression la plus simple, puisque c'est celle qui utilisera le moins de transistors.
Ainsi, la simplification d'une fonction consiste à obtenir son expression la plus compacte possible afin de minimiser le nombre d'opérateurs logiques (portes) nécessaires à sa réalisation.
Nous montrons brièvement dans cette section deux méthodes de simplification. L'une des méthodes est algébrique : elle n'utilise que des manipulations de symboles. L'autre méthode utilise un support graphique appelé table de Karnaugh et permet de trouver la forme simplifiée des expressions à trois ou quatre voir cinq variables.

Remarque 2.5.1. *L'algèbre de Boole observe la priorité des opérations avec par ordre décroissant de priorité :*

- *la fonction NON,*
- *la fonction ET,*
- *la fonction OU.*

2.5.1 Simplification algébrique des fonctions logiques

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. On peut se servir des règles de simplification suivantes :

1. Chaque fois qu'on rencontre deux minterms adjacents (c'est-à-dire n'ayant qu'une seule variable qui change), on conserve l'intersection commune.

2. Chaque fois qu'on rencontre deux maxterms adjacents (c'est-à-dire n'ayant qu'une seule variable qui change), on conserve la réunion commune. **Exemple 6.**

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

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

3. On peut ajouter un terme déjà existant à une expression logique car il n'y a pas de coefficient en algèbre de Boole.

4. On ne change pas le résultat en multipliant l'un des termes par 1 ou en ajoutant 0.

Exemple 7.

On se donne $f = ab + \overline{b}c + ac$ que nous voulons simplifier (réduire). On a :

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

Exemple 8.

On se donne $f = \overline{x}yz + x\overline{y}z + xy\overline{z} + xyz$ On a :

$$F = (\overline{x}yz + xyz) + (x\overline{y}z + xyz) + (xy\overline{z} + xyz) = yz(\overline{x} + x) + xz(\overline{y} + y) + xy(\overline{z} + z) = xy + yz + zx$$

Exemple 9.

On se donne $f(a, b, c) = abc + ab\overline{c} + a\overline{b}c + \overline{a}\overline{b}c + a\overline{b}\overline{c}$

En factorisant, on obtient :
$f(a, b, c) = a(b(c + \overline{c}) + \overline{b}(c + \overline{c})) + \overline{a}\overline{b}c = a + \overline{a}\overline{b}c = a + \overline{b}c$ car $x + \overline{x}y = x + y$. Finalement,

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

La simplification algébrique est toujours possible mais la démarche qui elle, est intuitive, dépend de l'habileté et de l'expérience.

2.5.2 Simplification graphique des fonctions logiques

Cette méthode repose sur l'utilisation des tables de Karnaugh. C'est une table de 2^n cases où n représente le nombre de variables. Sur les lignes et colonnes, on place l'état des variables d'entrée codées en binaire. Dans chacune des cases, on place l'état de la sortie pour les combinaisons d'entrée correspondantes. La construction des tables de Karnaugh exploite le codage de l'information et la notion d'adjacence. Son principe consiste à mettre en évidence sur un graphique les minterms (ou maxterms) adjacents, puis de transformer les adjacences logiques en adjacences "géométriques".

Adjacence des cases

Deux cases adjacentes sur le tableau de Karnaugh correspondent à des combinaisons différentes 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.

Règle pratique

On procède en 3 phases :

1. Transcrire la fonction dans un tableau codé,

2. Recherche des adjacents

3. Simplification des équations des groupements effectués

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. Elle met en évidence des associations du type des items 1. et 2. ci-dessus. On remarque que ces regroupements 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. **Exemple 10.**
On se propose de simplifier, à l'aide de la table de Karnaugh, les fonctions booléennes respectives suivantes :

$$F1 = \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + ab\overline{c} + a\overline{b}\overline{c}$$

$$F2 = \overline{a}b\overline{c} + ab\overline{c} + abc + \overline{a}bc \text{ et}$$

$$F3 = \overline{a}\overline{b}\overline{c}\overline{d} + \overline{a}\overline{b}c\overline{d} + a\overline{b}c\overline{d} + a\overline{b}\overline{c}\overline{d}$$

On obtient les tables de Karnaugh des fonctions F1, F2 et F3 représentées aux figures 2.10, 2.11 et 2.10 respectivement.

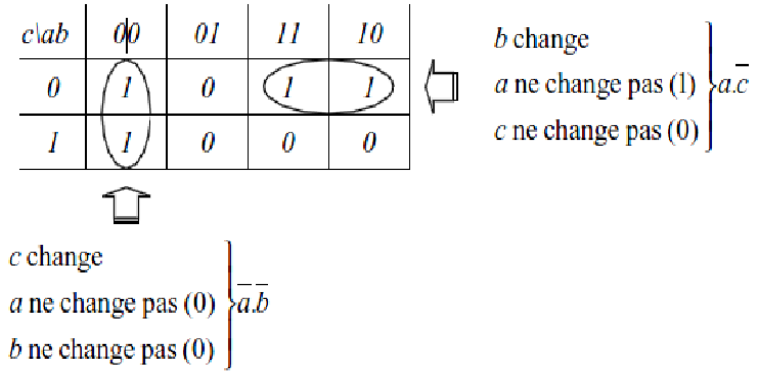


FIGURE 2.10 – Principe du regroupement (Exemple 1)

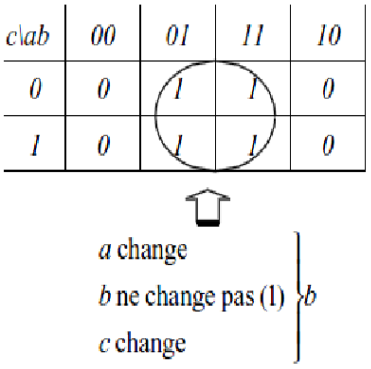


FIGURE 2.11 – Exemple 2

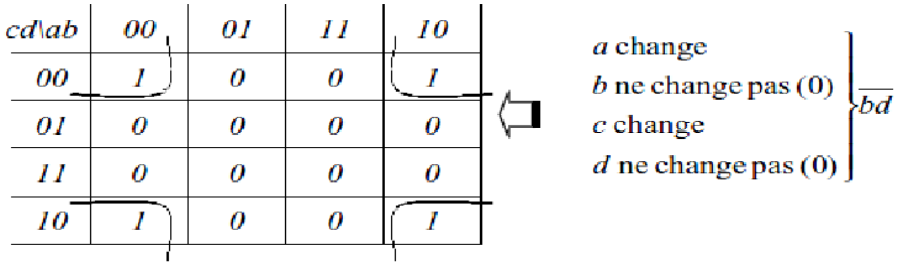


FIGURE 2.12 – Principe du regroupement (Exemple 3)

Récapitulatif des règles

1. On représente un tableau à 2 dimensions, chaque dimension concernant une ou 2 variables.
2. Le passage d'une colonne à une colonne adjacente ou d'une ligne à une ligne adjacente modifie la valeur d'une seule variable.
3. Le tableau se referme sur lui-même : la colonne la plus à gauche est voisine de la colonne la plus à droite, idem pour les lignes du haut et du bas.
4. Pour les 2 colonnes (2 lignes) extrêmes, là aussi, une seule variable doit changer de valeur entre ces 2 colonnes (lignes)
5. Une case du tableau contient une valeur booléenne, déterminée à partir de la table de vérité et des valeurs des variables
6. On ne regroupe que des blocs rectangulaires de 2^i cases (2,4,8,16,...) où $1 \leq i \leq n$ et $n =$ nombre de variables sachant que :
   - Tous les bits à 1 (resp à 0 pour la forme normale conjonctive) du tableau doivent être englobés dans au moins un bloc (un bloc à une taille de $1, 2, 4, 8 \ldots$ bits)

- Un bit à 1 (resp à 0 pour la forme normale conjonctive) peut appartenir à plusieurs blocs
- On doit créer les blocs les plus gros possibles

7. A chaque bloc correspond un terme formé comme suit :
   - Pour le bloc, si une une variable prend les valeurs 0 et 1, on ne la prend pas en compte
   - On ne conserve que les variables qui ne varient pas. Si une variable a reste à 1 (resp à 0 pour la forme normale conjonctive), on la note a ; si elle reste à 0 (resp à 1 pour la forme normale conjonctive), on la note $\overline{a}$
   - Le terme logique du bloc correspond au produit (resp à la somme pour la forme normale conjonctive) de ces variables qui ne changent pas
8. 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.
9. Il faut utiliser au moins une fois chaque 1 (resp chaque 0 pour la forme normale conjonctive), le résultat est donné par la réunion (resp le produit pour la forme normale conjonctive) logique de chaque groupement.
10. Il est inutile de regrouper des "1" qui ont tous déjà été regroupés par ailleurs si cela n'améliore pas les regroupements. On parle alors souvent de terme inclus.

2.5.3 Simplification des fonctions logiques en pratique

En pratique, la simplification des expressions logiques est un problème difficile à traiter, mais dans lequel les ordinateurs sont en général bien plus efficaces que nous. Des applications logicielles existent pour faire ce travail.
Il faut cependant acquérir une certaine familiarité avec les simplifications algébriques ordinaires. Les tables de Karnaugh, présentent de plus l'avantage de faire de bonnes questions d'examen.

2.5.4 Combinaison 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 11.**
On désire écrire 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 :
On obtient la table de vérité illustrée à la figure 2.13 et la table de Karnaugh associée (Cf. Figure 2.14) Si on ne tient pas compte des états 10 à 15, f s'écrit : $F1 = \overline{a}\overline{b}c + \overline{a}b\overline{c}d$ En considérant que les combinaisons impossibles sont affectées arbitrairement des valeurs 0 ou 1, on peut obtenir : $F1 = \overline{b}c + b\overline{c}d$.
Règle pratique : on remplit les cases avec le symbole X qui montre que l'on peut choisir la valeur que l'on veut (0 ou 1) pour cette case.

2.5.5 Simplification par la méthode de Quine-Mc Cluskey

L'algorithme se déroule en deux étapes. Pour faciliter la mise en œuvre de la méthode, la fonction logique doit être exprimée soit sous forme tabulaire (table de vérité, tableau de Karnaugh), soit sous la forme suivante :
$f(A_0, \cdots, A_{N-1}) = R(N_0, \cdots, N_{k-1}) + \Phi(M_0, \cdots, M_{p-1})$ où $N_0, \cdots, N_{k-1}$ sont k valeurs (exprimées en binaire ou plus souvent en décimales) correspondant aux k cas où $f(N_0, \cdots, N_{k-1})$, avec le nombre $(A_{N-1} \cdots A_0) = N_k$, vaut 1 et $(M_0, \cdots, M_{p-1})$ sont p valeurs correspondant aux p cas où $f(A_0, \cdots, A_{N-1})$, avec le nombre $(A_{N-1} \cdots A_0) = M_p$ est indéterminé. **Exemple 12.**
Si l'on veut exprimer de la sorte une porte NAND, on obtiendrait : $f_{NAND}(a, b) = R(0, 1, 2)$.

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

FIGURE 2.13 – Table de vérité

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

FIGURE 2.14 – Table de Karnaugh

Méthode de Quine

La première étape de l'algorithme qui n'est autre que la méthode de Quine, correspond à la recherche de l'ensemble des termes générateurs. Un terme est générateur s'il ne peut être combiné avec aucun autre terme pour donner un terme plus simple. **Exemple 13.**
Par exemple, dans le cas de la porte NAND, la fonction vaut 1 lorsque (ab) vaut 00, 01 ou 10. Le terme 00 n'est pas un terme générateur car combiné avec 01, il donne naissance à un terme plus simple 0x. En revanche, 0x ne peut être combiné avec un autre terme, c'est donc un terme générateur. De la même manière x0 est un autre terme générateur.

Pour trouver les termes générateur, on utilise les valeurs $(N_0, \cdots, N_{k-1})$ et $(M_0, \cdots, M_{p-1})$ car, comme dans un tableau de Karnaugh, les termes indéterminés peuvent conduire à des simplifications.
La méthode de Quine consiste, en partant de la décomposition disjonctive canonique de f, à utiliser systématiquement la formule de simplification $xy + \overline{x}y = y$ où x est un littéral et y un monôme.

Considérons l'exemple suivant : $f(a,b,c,d) = ab + bc + a\overline{c} + \overline{a}cd + \overline{a}\overline{b}\overline{d} + \overline{a}\overline{b}c$
La décomposition sous forme normale disjonctive canonique de f est :

$$\begin{aligned} f(a,b,c,d) &= abcd + abc\overline{d} + ab\overline{c}d + ab\overline{c}\overline{d} + a\overline{b}cd + a\overline{b}c\overline{d} + \overline{a}bcd + \overline{a}bc\overline{d} + \overline{a}\overline{b}cd + \overline{a}\overline{b}c\overline{d} + \overline{a}\overline{b}\overline{c}\overline{d} \\ &= 1111 + 1110 + 1101 + 1100 + 1001 + 1000 + 0111 + 0110 + 0011 + 0010 + 0000 \end{aligned}$$

Etape 0 : On classe les mintermes de la décomposition canonique selon le nombre de "1" de leur écriture : classe sans "1", classe avec un seul "1", etc... On obtient le tableau 2.10 suivant :

Etape 1 : On additionne chaque minterm de la classe j avec chaque minterm de la classe $j+1$. Lorsqu'il est possible d'utiliser la formule $xy + \overline{x}y = y$, et d'éliminer ainsi une variable, on consigne le résultat dans la colonne Etape 1. Le symbole "x" remplace la variable éliminée. On repère à l'aide du symbole "1" dans la colonne repère les deux minterms concernés afin de montrer qu'ils disparaissent de la forme $\sum\prod$ (Somme de produits). On obtient le tableau 2.11 ci-dessous :

Etape 2 : On procède comme à l'étape précédente avec les monômes des classes 0, 1, 2, etc. On inscrit dans la colonne Etape 2, les nouveaux monômes obtenus. Si un monôme a déjà été obtenu, on ne le réinscrit pas, par contre on marque avec des "1", les monômes dont il est issu. D'où le tableau 2.12 ci-dessous :

Etape 3 (et suivantes) : On réitère le processus jusqu'à ce qu'il n'y ait plus de simplifications. Dans l'exemple, il n'y a pas d'étape 4 et on obtient le tableau 2.13 ci-dessous :

Classe	Etape 0	Repère
0	0000	
1	0010 1000	
2	0011 0110 1001 1100	
3	0111 1101 1110	
4	1111	

TABLE 2.10 – Tableau à l'étape 0 de la méthode de Quine

Classes	Etape 0	Repère	Etape 1	Repère
0	0000	1	00x0 x000	
1	0010 1000	1 1	001x 0x10 100x 1x00	
2	0011 0110 1001 1100	1 1 1 1	0x11 011x x110 1x01 110x 11x0	
3	0111 1101 1110	1 1 1	x111 11x1 111x	
4	1111	1		

TABLE 2.11 – Tableau à l'étape 1 de la méthode de Quine

A la fin, les monômes qui ont pour repère 0 sont les **termes générateurs** encore appelés **implicants premiers** de f. Ici les implicants premiers de f sont : $00x0$, $x000$, $0x1x$, $1x0x$, $x11x$ et $11xx$; soit encore :

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

Méthode de Mc Cluskey

La deuxième étape de l'algorithme qui n'est rien d'autre que la méthode de Mc Cluskey, correspond à l'élimination, parmi les termes générateurs, des termes redondants. Pour celà, on identifie les termes générateurs à partir desquels pour chaque $(n_0, \cdots, n_{k-1})$ peut-être écrit et on élimine ceux qui sont en trop.

Exemple 14.

Par exemple, dans le cas de la porte NAND, le terme 00 et 01 peuvent être écrit avec le terme générateur 0x mais celui-ci n'est pas utilisable pour écrire 10. L'utilisation du second générateur x0 est indispensable. Il n'y a pas de terme redondant.

Pour cette étape de simplification, on ne cherche à coder que les les valeurs $(N_0, \cdots, N_{k-1})$. Les valeurs indéterminés nous sont ici inutiles.

C'est dire qu'à la fin de l'applicatiion de la méthode de Quine, la forme simplifiée de f peut ne pas être optimale car il y subsiste des redondances. On construit alors la grille de Mc Cluskey de la figure 2.15.

Les barres verticales représentent les minterms de la décomposition disjonctive de f et les barres horizontales ses implicants premiers. Les minterms construits à partir des implicants premiers sont marqués d'une croix. Lorsqu'il n'y a qu'une croix sur une ligne verticale, elle est entourée. Cela signifie que les implicants premiers correspondants doivent figurer impérativement dans l'expression simplifiée de f. On a donc : $f(a,b,c,d) = a\overline{c} + \overline{a}c + \cdots$

Classes	Etape 0	Repère	Etape 1	Repère	Etape 2	Repère
0	0000	1	00x0	0		
			x000	0		
1	0010	1	001x	1	0x1x	
	1000	1	0x10	1	1x0x	
			100x	1		
			1x00	1		
2	0011	1	0x11	1	x11x	
	0110	1	011x	1	11xx	
	1001	1	x110	1		
	1100	1	1x01	1		
			110x	1		
			11x0	1		
3	0111	1	x111	1		
	1101	1	11x1	1		
	1110	1	111x	1		
4	1111	1				

TABLE 2.12 – Tableau à l'étape 2 de la méthode de Quine

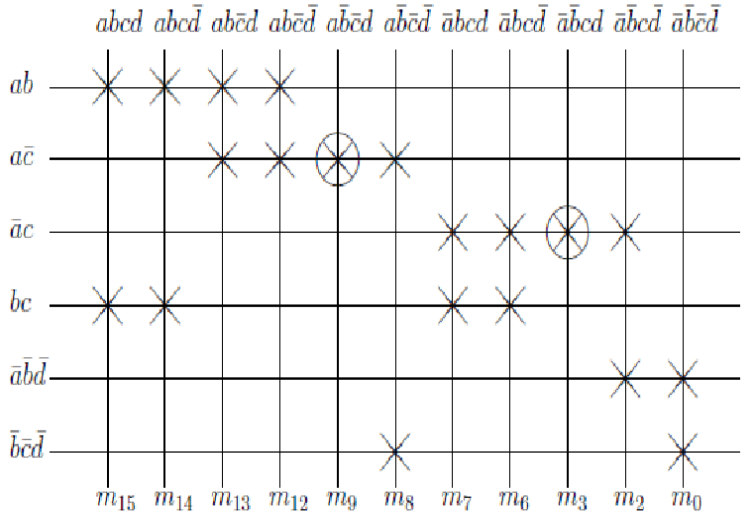


FIGURE 2.15 – Grille de Mc Cluskey

On remarque ensuite qu'avec $a\bar{c}$ et $\bar{a}c$, on peut construire tous les minterms de f sauf $abcd$, $abc\bar{d}$, $\bar{a}\bar{b}\bar{c}\bar{d}$.
Pour les couvrir, on a le choix entre : $ab + \bar{a}\bar{b}\bar{d}$, $ab + \bar{b}\bar{c}\bar{d}$, $bc + \bar{a}\bar{b}\bar{d}$ ou $bc + \bar{b}\bar{c}\bar{d}$.
Ce qui donne les quatre formes simplifiées minimales de f :

$$\begin{aligned} f(a,b,c,d) &= a\bar{c} + \bar{a}c + ab + \bar{a}\bar{b}\bar{d} \\ f(a,b,c,d) &= a\bar{c} + \bar{a}c + ab + \bar{b}\bar{c}\bar{d} \\ f(a,b,c,d) &= a\bar{c} + \bar{a}c + bc + \bar{a}\bar{b}\bar{d} \\ f(a,b,c,d) &= a\bar{c} + \bar{a}c + bc + \bar{b}\bar{c}\bar{d} \end{aligned}$$

Remarque 2.5.2. *Si aucune croix n'est entourée, on choisit judicieusement un certain nombre d'implicants premiers de manière à englober le plus de minterms possibles.*

2.6 Réalisation et décodage d'un logigramme

2.6.1 Réalisation d'un logigramme

Un logigramme est la traduction graphique d'une équation logique utilisant les symboles normalisés des opérateurs. Pour le réaliser, on construit les différentes portes intervenant dans la fonction logique à réaliser. Ensuite, on relie ces différentes

Classes	Etape 0	Repère	Etape 1	Repère	Etape 2	Repère
0	0000	1	00x0	0		
			x000	0		
1	0010	1	001x	1	0x1x	0
	1000	1	0x10	1	1x0x	0
			100x	1		
			1x00	1		
2	0011	1	0x11	1	x11x	0
	0110	1	011x	1	11xx	0
	1001	1	x110	1		
	1100	1	1x01	1		
			110x	1		
			11x0	1		
3	0111	1	x111	1		
	1101	1	11x1	1		
	1110	1	111x	1		
4	1111	1				

TABLE 2.13 – Tableau à l'étape 3 de la méthode de Quine

portes en veillant à ce que les entrées et les sorties correspondent. **Exemple 15.**
La figure 2.16 présente l'obtention d'une porte XOR à partir des portes classiques ET et OU.

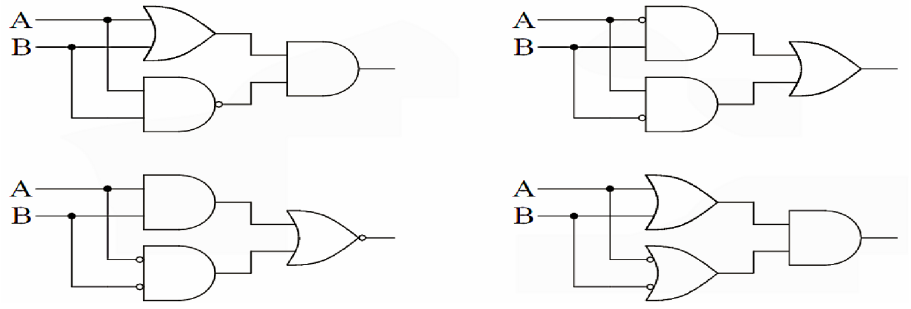


FIGURE 2.16 – Obtention de la porte XOR à partir des portes ET et OU

Exemple 16.
La figure 2.17 montre, par exemple, comment les portes NOT, OR et AND peuvent être obtenues à partir de portes NOR(↓) uniquement.

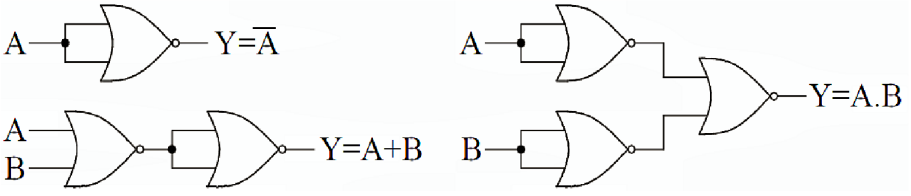


FIGURE 2.17 – Obtention des portes classiques à partir de la porte NOR

Exemple 17.
De façon analogue, il est possible d'obtenir les portes classiques à partir de la porte NAND(↑) exclusivement.

De manière générale, on a :

$$a+b=(a\downarrow b)\downarrow(a\downarrow b);\quad ab=(a\downarrow a)\downarrow(b\downarrow b);\quad \overline{a}=(a\downarrow a)\text{ et}$$

$$ab=(a\uparrow b)\uparrow(a\uparrow b);\quad a+b=(a\uparrow a)\uparrow(b\uparrow b);\quad \overline{a}=(a\uparrow a)$$

2.6.2 Décodage d'un logigramme

Décoder un logigramme consiste à rechercher l'équation de la sortie en fonction des variables d'entrée. La méthode de décodage la plus simple consiste à établir les équations logiques de la sortie de chaque opérateur, en partant des entrées vers la sortie. Une illustration est faite à la figure 2.19.

Il est également possible de procéder au décodage indirect en établissant la table de vérité correspondant au logigramme.

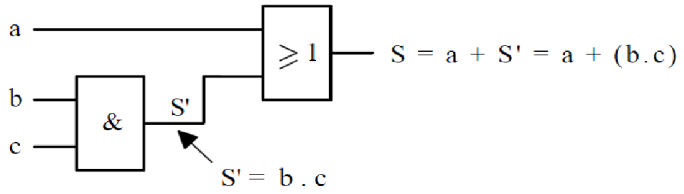


FIGURE 2.18 – Décodage d'une fonction

On utilise la table de vérité des opérateurs constituant le logigramme.
Ainsi, pour le même logigramme ci-dessus, on obtiendrait alors la table suivante (Cf figure 2.20).

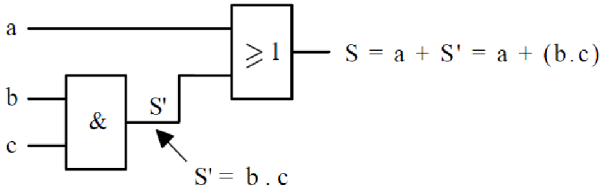


FIGURE 2.19 – Décodage direct

c	b	a	S
0	0	0	0
0	0	1	**1**
0	1	0	0
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

FIGURE 2.20 – Décodage indirect

Chapitre 3

Circuits logiques

Un circuit logique est un dispositif électronique constitué d'un ensemble de portes logiques reliées entre elles correspondant à une expression algébrique permettant ainsi la réalisation d'une fonction logique.
En réalité, chaque variable de la fonction est matérialisée par un conducteur et sa valeur sera définie à partir de sa tension. Dans un circuit logique, on a donc des entrées et des sorties :

- Les entrées sont les conducteurs qui vont permettre de présenter les opérandes.
- Les sorties sont les conducteurs qui vont permettre de consulter le résultat.

Exemple 18.
Si on considère le circuit logique : $f(x_1, x_2, x_3, \cdots, x_n) \rightarrow (y_1, y_2, y_3, ..., y_m)$, $x_1, x_2, x_3, ..., x_n$ seront appelés entrées du circuits tandis que $y_1, y_2, y_3, \cdots, y_m$ en sont les sorties.
les conducteurs sont très souvent appelés des broches ; il en existe 4 types :

- entrées " normales " : forçées par l'utilisateur
- sorties "normales" : forcées par le circuit
- sorties 3 états : (peuvent être laissées libres dans certains cas) on les utilise quand on travaille avec les bus, circuit à validation.
- les collecteurs ouverts : libres ou forcés à 0

L'assemblage des circuits ne peut se faire au hasard ; il faut respecter un minimum de règles ou principes :

- la compatibilité des tensions entre les entrées et les sorties
- à un instant donné, on ne peut forcer une entrée que d'une seule façon

3.1 Les circuits combinatoires

Un circuit combinatoire est un circuit logique dont l'état des sorties ne dépend que des valeurs assignées aux variables d'entrée. N'ayant pas de mémoire interne, les sorties d'un tel circuit sont alors déterminées de façon univoque. Les résultats donnés sont souvent représentés sur un chronogramme statique mais en réalité, il y a toujours un moment de latence entre le moment où on envoie quelque chose en entrée et le moment où on récupère un résultat en sortie ; cet intervalle de temps est connu et s'appelle le **"délai de propagation"** , noté tp ou δt.

On peut faire un assemblage de circuits combinatoires, en mettant bout à bout plusieurs circuits combinatoires, et en évitant les boucles (une entrée d'un circuit étant une de ses sorties ou une sortie d'un circuit "postérieur").
Quoiqu'il en soit, la construction d'un circuit combinatoire passe par les différentes étapes suivantes :

1. Spécification de la fonction logique : cela consiste à identifier les entrées et les sorties de la fonction.
2. Construction la table de vérité.
3. Identification de la fonction à partir de la table de vérité.

4. Optimisation (simplification) de la fonction.

5. Tracé du logigramme.

Exemple 19.
On se propose de construire le logigramme de la fonction logique

$$f(x, y, z) = (x \Rightarrow y) \oplus (y \Rightarrow z) \oplus (z \Rightarrow x)$$

On a donc trois entrées que sont x, y et z ; A la fin, nous aurons une seule sortie f, comme le montre la table de vérité 3.1.
On obtient donc

x	**y**	**z**	$x \Rightarrow y$	$y \Rightarrow z$	$z \Rightarrow x$	$(x \Rightarrow y) \oplus (y \Rightarrow z)$	$f(x, y, z)$
0	0	0	1	1	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	1	1	0
0	1	1	1	1	0	0	0
1	0	0	0	1	1	1	0
1	0	1	0	1	1	1	0
1	1	0	1	0	1	1	0
1	1	1	1	1	1	0	1

TABLE 3.1 – Table de la fonction f

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

Ensuite, les regroupements doivent être réalisés de telle sorte qu'un maximum de valeurs 1 soient englobées dans un minimum de regroupements. Autrement dit, il faut réaliser les plus gros regroupements possibles, et en plus petit nombre possible.
De là, on déduit le logigramme :

3.1.1 Le décodeur

Un décodeur comprend n entrées et 2^n sorties dont l'une est toujours fixée à 1. La sortie activée correspondant à la configuration binaire du mot formé par les n entrées. Un tel circuit sert entre autres, à sélectionner des adresses de la mémoire. Autrement dit : Un décodeur permet de sélectionner une sortie S_i $0 \leq i \leq 2^n - 1$ avec $i = (E_n \dots E_0)_{10}$
Exemple 20.
Application au décodeur 3-8
Soient $E_2 = 1$; $E_1 = 1$; $E_0 = 0$, c'est-à-dire 110. L'entier codé par $E_2E_1E_0$ est 6, par conséquent la sortie S_6 est sélectionnée, elle vaut 1.

Application au décodeur 2-4
Ce décodeurs possède deux entrées E_1 et E_0, quatre sorties S_3 ; S_2 ; S_1 ; S_0 telles que la sortie dont le numéro d'indice de la sortie est codé par la configuration binaire des entrées E_1E_0 est mise à 1. On a la table de vérité 3.2 Les équations

E_1	E_0	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

TABLE 3.2 – Table de vérité pour un décodeur 2-4

booléennes des sorties en fonction des entrées nous donnent :

$$S_0 = \overline{E_1E_0}, S_1 = \overline{E_1}E_0,\ S_2 = E_1\overline{E_0},\ S_3 = E_1E_0.$$

Et le logigramme correspondant nous conduit à la figure 3.1 ci-dessous : Un tel décodeur est utilisé pour une mémoire de 4 mots.

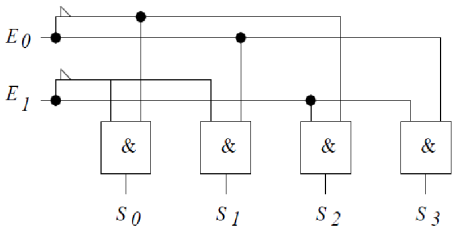


FIGURE 3.1 – Le décodeur 2-4

3.1.2 Le comparateur

Un comparateur est un circuit logique qui a 2n entrées et 1 sortie, les 2n entrées formant deux mots de n bits chacun : A et B. La sortie vaut 1 si et seulement si le mot A=B. Ainsi, 2 mots sont égaux si tous les bits de même rang sont égaux. Par conséquent :

- $A_i = B_i \Rightarrow A_i + B_i = 0$
- $A_i \neq B_i \Rightarrow A_i + B_i = 1$

On utilise ces deux relations pour construire le comparateur. Ainsi :

- $A = B \Rightarrow \ \forall i,\ A_i = B_i \Rightarrow \sum A_i + B_i = 0.$
- $A = B \Rightarrow \ NON(\sum A_i + B_i = 0).$

Exercice 1.
Réalisez un tel comparateur avec n portes "ou exclusif" et n-1 portes "et" Un comparateur 3 bits nous donnerait par exemple la figure ci-dessous

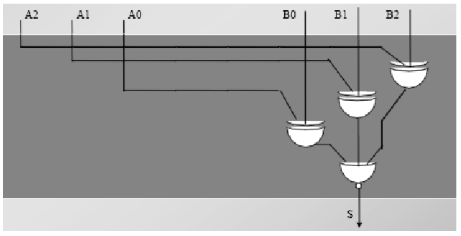


FIGURE 3.2 – Un comparateur 3 bits

3.1.3 Le décaleur

Un décaleur est un circuit logique formé de (n+1) entrées $D_1, \ldots, D_n, C$ et de n sorties $S_1, \ldots, S_n$ et opère un décalage de 1 bit sur les entrées $D_1, \ldots, D_n$. Si C=1, il s'agit d'un décalage à droite et si C=0, d'un décalage à gauche.
Exercice 2.
Construire le logigramme du décaleur à 4 bits d'entrée

3.1.4 L'additionneur

Pour réaliser des additions de 2 mots de 16 bits, on utilise 16 additionneurs à 1 bit, reliés pour gérer la propagation éventuelle de retenues. Ces additionneurs sont eux-mêmes formées à l'aide de demi-additionneurs dont la table de vérité et le logigramme correspondant sont donnés ci-dessous (Cf. Table 3.3 et Figure 3.3). Afin de permettre une liaison de plusieurs additionneurs en série, un additionneur doit avoir une retenue en entrée R_e en plus de la retenue en sortie R_s. Sa table de vérité est donc la suivante (cf. 3.4) : On peut vérifier que le logigramme de la figure 3.4 suivant réalise correctement l'additionneur. En effet, on a :

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

TABLE 3.3 – Table de vérité du demi-additionneur

A	**B**	R_e	**S**	R_s
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

TABLE 3.4 – Table de l'additionneur

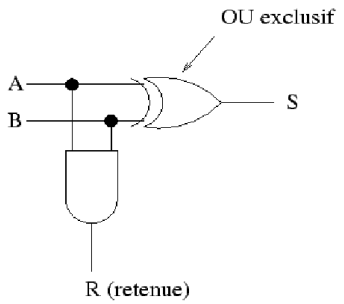


FIGURE 3.3 – le demi-additionneur

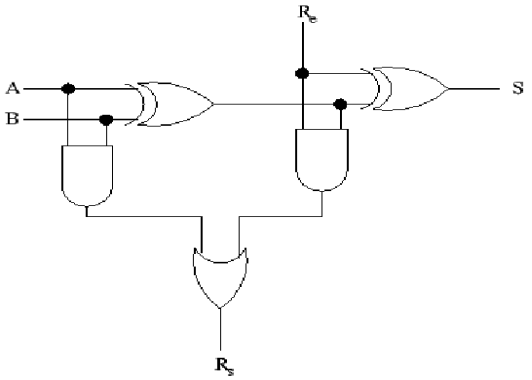


FIGURE 3.4 – L'additionneur

$$\begin{aligned} S &= \overline{AB}R_e + \overline{A}B\overline{R_e} + \overline{AB}\overline{R_e} + ABR_e \\ &= R_e(\overline{AB} + AB) + \overline{R_e}(\overline{A}B + A\overline{B}) \\ &= R_e(\overline{A \oplus B}) + R_e(\overline{A \oplus B}) \\ &= R_e \oplus (\overline{A \oplus B}) \end{aligned} \quad \text{et} \quad \begin{aligned} R_S &= \overline{A}BR_e + A\overline{B}R_e + AB\overline{R_e} + ABR_e \\ &= R_e(\overline{A}B + A\overline{B}) + AB(\overline{R_e} + R_e) \\ &= R_e(A \oplus B) + AB \end{aligned}$$

En résumé, la réalisation d'un circuit combinatoire passe par deux phases :

- Recherche d'une solution correcte (analyse)
- Simplification de la solution trouvée (-> circuit plus efficace).

3.2 Les circuits séquentiels

Dans les circuits de logique séquentielle nous devons tenir compte de l'état du système. Ainsi les sorties dépendent des entrées mais également de l'état du système. Celui-ci dépend aussi des entrées. Si nous notons Q l'état d'un système séquentiel, X ses entrées et Y ses sorties, nous avons de manière générale :

$$\begin{cases} Q &= f(X,Q) \\ Y &= g(X,Q) \end{cases} \tag{3.1}$$

La logique séquentielle permet de réaliser des circuits dont le comportement est variable avec le temps. L'état d'un système constitue une mémoire du passé.

Lorsque les changements d'état des divers composants d'un circuit séquentiel se produisent à des instants qui dépendent des

temps de réponse des autres composants et des temps de propagation des signaux on parle de logique séquentielle asynchrone. Cependant les retards peuvent ne pas être identiques pour toutes les variables binaires et conduire à certains aléas. Ceux-ci peuvent être évités en synchronisant la séquence des diverses opérations logiques sur les signaux périodiques provenant d'une horloge. La logique séquentielle est alors dite synchrone : tous les changements d'état sont synchronisés sur un signal de contrôle.

Nous commençons notre étude par celle des bascules, éléments de base des circuits séquentiels. Puis nous étudierons les registres et les compteurs.

3.2.1 Les bascules

Une bascule (flip-flop) a pour rôle de mémoriser une information élémentaire. C'est une mémoire à 1 bit. Une bascule possède deux sorties complémentaires Q et $\bar{Q}$. La mémorisation fait appel à un verrou (latch) ou système de blocage, dont le principe de rétro-action peut être représenté conformément à la figure 3.5) :

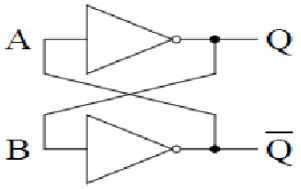


FIGURE 3.5 – Principe de rétro-action d'un verrou

Nous pouvons vérifier :

$$\begin{cases} (Q=1) \Rightarrow (B=1) \Rightarrow (\bar{Q}=0) \Rightarrow (A=0) \Rightarrow (Q=1) \\ (Q=0) \Rightarrow (B=0) \Rightarrow (\bar{Q}=1) \Rightarrow (A=1) \Rightarrow (Q=0) \end{cases} \tag{3.2}$$

Une bascule ne peut donc être que dans deux états :

- "1" $(Q=1, \bar{Q}=0)$ et
- "0" $(Q=0, \bar{Q}=1)$.

Les interconnexions du verrou interdisent les deux autres combinaisons à savoir : $Q=\bar{Q}=1$ ou $Q=\bar{Q}=0$.
Ce type de circuit, qui n'a que deux états stables possibles, est encore appelé circuit bistable.

Un verrou permet de conserver un état, il nous faut maintenant savoir comment charger cet état.

a/ Les bascules R-S

Les verrous les plus fréquemment rencontrés sont réalisés avec deux portes NOR (Cf. Figure 3.6) ou NAND (Cf. Figure 3.7).
Considérons dans un premier temps le circuit construit à l'aide de portes NOR, celle de la figure 3.6.

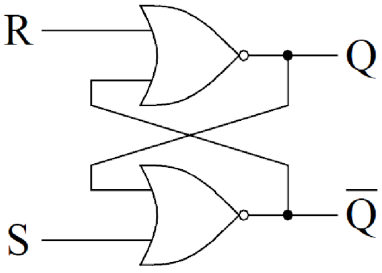


FIGURE 3.6 – Bascule R-S (portes NOR)

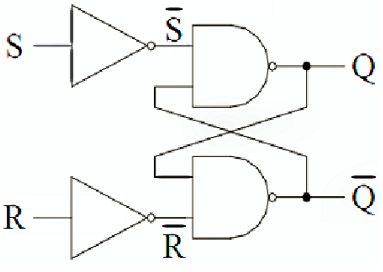


FIGURE 3.7 – Bascule R-S (portes NAND)

La table 3.5 donne la table de vérité correspondante.

Si on applique S = 1 et R = 0 ou S = 0 et R = 1 on impose l'état de la sortie Q respectivement à 1 ou à 0, la sortie Q prenant la valeur complémentaire. Cet état se maintient lorsque les deux entrées retournent à 0. La configuration S = R = 1 est à proscrire car ici elle conduit à Q = Q = 0 , ce qui est inconsistant logiquement avec notre définition. Mais surtout, lorsque R et S reviennent à 0, l'état Q = Q étant incompatible avec les interconnexions, l'une de ces deux sorties va reprendre l'état 1, mais il est impossible de 59 prédire laquelle : la configuration S = R = 1 conduit à une indétermination de l'état des sorties et est donc inutilisable. La représentation d'une bascule RS est donnée sur la figure 3.8.
A la première ligne, les sorties restent inchangées. A la deuxième ligne, on a un reset, i-e une remise à Zéro : RAZ. A la

S	**R**	**Q**	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	0	0

TABLE 3.5 – Table de vérité de la bascule R-S

troisième ligne, on a un Set, i-e une remise à Un : RAU. La dernière ligne est à proscrire.

Considérons maintenant la bascule réalisée avec des portes NAND(Cf. Figure 3.7).
Il lui correspond la table de vérité 3.6.
L'utilisation des deux inverseurs sur les lignes d'entrée nous permet de retrouver une table de vérité comparable à celle de

S	**R**	$\bar{\mathbf{S}}$	$\bar{\mathbf{R}}$	**Q**	$\bar{\mathbf{Q}}$
0	0	1	1	Q	$\bar{Q}$
0	1	1	0	0	1
1	0	0	1	1	0
1	1	0	0	1	1

TABLE 3.6 – Table de vérité de la bascule R-S avec des portes NAND

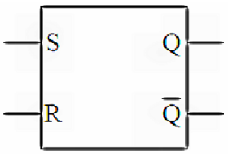


FIGURE 3.8 – Représentation d'une bascule R-S

la bascule RS précédente.
On se rend compte que dans le cas où $R = S = 1$, les deux sorties Q et $\bar{Q}$ sont toutes les deux à 0 dans la table 3.5 ou à 1 dans la table 3.6. Cet état est instable, et pour cette raison, cette situation ne doit pas être autorisée. La table 3.7 donne la table finale de la bascule RS dans laquelle la sortie Q_{n+1} à l'instant n+1 est déterminée en fonction de son état à l'instant n.

b/ Les bascules J-K

La bascule J-K permet de lever l'ambiguïté qui existe dans la table 3.7. Ceci peut être obtenu en asservissant les entrées R et S aux sorties Q et $\bar{Q}$ selon le schéma logique indiqué sur la figure 3.9.
Pour les signaux R et S, on a :

$$\begin{cases} S = J.\bar{Q} \\ R = K.Q \end{cases} \tag{3.3}$$

Ce qui nous permet de construire la table de vérité de la bascule J-K.
On constate que la combinaison $R = S = 1$ n'est jamais rencontrée.
De plus, la table obtenue peut se résumer sous la forme présentée à la table 3.9.

La figure 3.10 explicite le diagramme logique d'une bascule J-K. Les entrées asynchrones (car à utiliser en absence de

$\mathbf{S}_n$	$\mathbf{R}_n$	$\mathbf{Q}_{n+1}$
0	0	Q_n
0	1	0
1	0	1
1	1	indéfini

TABLE 3.7 – Table de vérité résumé de la bascule RS

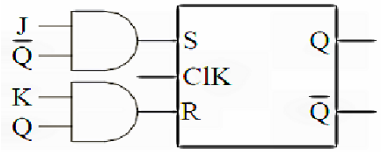


FIGURE 3.9 – Réalisation d'une bascule J-K

signal d'horloge) à savoir Pr (pour Preset) et Cr (pour Clear) permettent d'assigner l'état initial de la bascule, par exemple à la mise sous tension pour éviter tout aléa. En fonctionnement normal ces deux entrées doivent être maintenues à 1. La figure

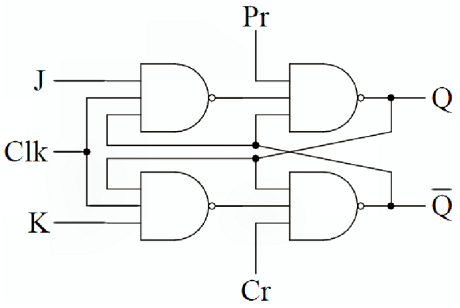


FIGURE 3.10 – Une bascule J-K

3.11 donne la représentation symbolique d'une bascule J-K avec les entrées Preset et Clear. A partir de la table 3.9 nous pouvons construire la table de transition de la bascule J-K. La table 3.10 donne les etats dans lesquels doivent se trouver les entrees J et K pour obtenir chacune des quatre transitions possibles de la sortie Q. Une croix indique que l'état de l'entrée considérée est indifferent : 0 ou 1. Par exemple, pour obtenir la transition $0 \rightarrow 1$ de la sortie Q il faut que l'entrée J soit dans l'état 1, quelque soit l'état de l'entrée K. En effet, nous pouvons avoir J = K = 1 qui inverse l'état de la bascule ou J = 1 et K = 0 qui charge 1 dans la bascule. Comme les deux entrées ne sont jamais spécifiées simultanément il est possible de choisir pour simplifier l'égalité des deux entrées :

$$J = K$$

On utilise parfois l'expression logique donnant $\mathbf{Q_{n+1}}$ en fonction de J_n, K_n et Q_n. Pour cela nous pouvons par exemple construire le tableau de Karnaugh à partir de la table de vérité (table 3.8) de la bascule J-K d'où nous tirons l'équation caractéristique qui exprime l'état futur en fonction de l'état présent et des entrées :

$$Q_{n+1} = J_n\overline{Q_n} + \overline{K_n}Q_n$$

c/ Les bascules D

Une bascule D (Delay) est obtenue à partir d'une bascule J-K en envoyant simultanément une donnée sur l'entrée J et son inverse sur l'entrée K : A partir de la table 3.9 on peut donc écrire écrire :

$$\begin{cases} (D_n = 1) & \Rightarrow & (J_n = 1, K_n = 0) & (Q_{n+1} = 1) \\ (D_n = 0) & \Rightarrow & (J_n = 0, K_n = 1) & (Q_{n+1} = 0) \end{cases} \tag{3.4}$$

Ce qui peut se résumer par $Q_{n+1} = D_n$. Ainsi l'état de la bascule Q pendant l'intervalle n+1 est égal à la valeur de l'entrée D pendant l'intervalle n. Une bascule D agit comme une unité à retard pour laquelle la sortie suit l'entrée avec un cycle de retard. Sa représentation symbolique est donnée par la figure 3.13.

J	K	Q_n	$\overline{Q_n}$	S	R	$\overline{Q_{n+1}}$
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	1	0	1
1	1	1	0	0	1	0

TABLE 3.8 – Table de vérité de la bascule JK

J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

TABLE 3.9 – Table de vérité résumé de la bascule JK

3.2.2 d/ Les bascules T

3.2.3 Registre de mémorisation

En réunissant plusieurs bascules sur un même signal d'horloge, on peut fabriquer un circuit qui constitue un registre, d'où la possibilité de construire des mémoires [1]. Le problème est de minimiser le nombre de broches.
La figure 3.14 présente un exemple de mémoire de quatre mots de 3 bits chacun, $M_0, ..., M_3$ dans un boîtier à 14 broches, chaque mot de 3 bits étant formé à partir de 3 bascules [2].

Sur cette figure, on a :

- 8 entrées :
  - 3 entrées de données I0, I1, I2 ;
  - 2 entrées d'adresses A0, A1 (4 mots) ;
  - 3 entrées de commande :
  - CS : sélection du boîtier (chip select) si CS=1 ;
  - RD : lecture/écriture (1/0) ;
  - OE : activation des sorties (output enable).
- 3 sorties de données D0, D1, D2.

Par ailleurs : Les lectures et les écritures sont réalisées de la manière suivante :

- si RD=1, le mot lu à l'adresse indiquée par A0 A1 est positionné en sortie ;
- si RD=0, le mot d'entrée est chargé dans celui sélectionné par A0 A1.

Remarque 3.2.1. *A_0, A_1 et les entrées des quatre mots forment un décodeur.*

Sur les lignes de sortie, on utilise des circuits qui jouent le rôle d'interrupteurs [3] et comportent 3 états : 0, 1, et un état de haute impédance qui déconnecte le circuit lorsque la commande vaut 0. Circuit 3 états :
Si les 3 signaux RD, CS, OE sont à 1, les sorties sont activées, sinon elles sont déconnectées par les circuits trois états.

1. Les mémoires actuelles ne sont plus fabriquées à l'aide de bascules, et stockent plusieurs bits d'information par transistor. Néanmoins, l'explication à base de bascules reste simple et compréhensible.
2. Les bascules utilisées sont des bascules D. L'entrée d'horloge est ici nommée CK
3. On les appelle des "latch".

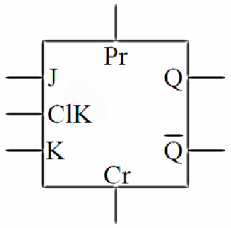


FIGURE 3.11 – Représentation d'une bascule J-K

Q_n	Q_{n+1}	J_n	K_n
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

TABLE 3.10 – Table de vérité résumé alternative de la bascule JK

Ce schéma se généralise avec un nombre de mots égal à 2^n en utilisant n lignes d'adresses $A_0, \cdots, A_{n-1}$. La taille d'un mot correspond au nombre de bascules, au nombre d'entrées et au nombre de sorties.

La structure décrite ci-dessus concerne les mémoires statiques. Pour les mémoires dynamiques, la structure interne n'utilise pas des bascules mais des condensateurs [4]. La capacité est plus grande, mais la charge électrique baisse avec le temps, d'où la nécessité de rafraîchir. Il existe différentes variantes intermédiaires entre les mémoires dynamiques [5] et les mémoires à lecture seulement (ROM, "read only memory"), programmables une seule fois à la fabrication [6].

3.2.4 Registre à décalage

3.2.5 Compteurs

Un compteur est un ensemble de n bascules interconnectées par des portes logiques. Ils peuvent donc mémoriser des mots de n bits. Au rythme d'une horloge ils peuvent décrire une séquence déterminée c'est-à-dire occuper une suite d'états binaires. Il ne peut y avoir au maximum que 2^n combinaisons. Ces états restent stables et accessibles entre les impulsions d'horloge. Le nombre total N des combinaisons successives est appelé le modulo du compteur. On a $N \leq 2^n$. Si $N < 2^n$, un certain nombre d'états ne sont jamais utilisés.

Les compteurs binaires peuvent etre classés en deux catégories : - les compteurs asynchrones ; - les compteurs synchrones. De plus on distingue les compteurs réversibles appelés compteurs-décompteurs.

3.2.6 Compteurs asynchrones

3.2.7 Compteurs synchrones

3.3 Application aux circuits séquentiels synchrones

Nous nous limiterons ici aux circuits séquentiels synchrones (le même signal d'horloge est envoyé à toutes les bascules du circuit).

Un circuit séquentiel est un circuit ayant des éléments de mémoire (bascules) et dans lequel, par conséquent, les sorties

4. En fait des transistors à effet de champ, dans lesquels on piège des électrons (ou des trous, c'est-à-dire des absences d'électrons), ce qui revient effectivement à les voir comme des condensateurs de très faible capacité.

5. et aujourd'hui les mémoires statiques-dynamiques (SDRAM), les innombrables variantes de mémoire vidéo, etc.

6. mais il existe aussi les PROM ("programmable ROM"), qui sont programmables une fois après fabrication, par destruction de fusibles, les EPROM ("erasable PROM"), que l'on peut effacer en totalité typiquement par une exposition prolongée à des rayons ultraviolets, puis reprogrammer, les EEPROM ("electrically erasable PROM"), que l'on peut effacer en totalité par une impulsion électrique, puis reprogrammer, les mémoires FLASH, qui sont intermédiaires entre les EEPROM et les mémoires vives (RAM, "random access memory") que nous avons décrites dans cette section. On peut lire et écrire dans une FLASH à volonté, mais le temps d'accès est plus long que pour les RAM. Ceci permet d'utiliser les mémoires FLASH comme des substituts rapides à des disques durs.

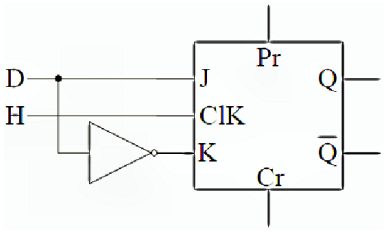


FIGURE 3.12 – Réalisation d'une bascule D

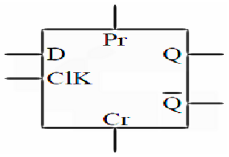


FIGURE 3.13 – Représentation d'une bascule D

dépendent non seulement des entrées présentes, mais aussi des entrées antérieures. L'ensemble des états (0 ou 1) de chacune des bascules du système constitue un état du système. On aura donc 2n états possibles dans un circuit comprenant n bascules.

Exemple : compteur par 2 sans entrée réalisé avec une bascule D

Ce circuit comprend une seule bascule, il n'a donc que deux états. On veut qu'il change d'état à chaque coup d'horloge.
On se sert du fait que pour une bascule D, $Q^+ = D$, ce qui signifie que la valeur de la sortie après le prochain coup d'horloge sera égale à la valeur actuelle de D.
La figure 3.15 présente le diagramme de transiton de ce compteur. On en déduit que $D = \overline{Q}$ et on a le circuit représenté à la figure 3.16.

Exemple : compteur par 2 avec entrée

On peut rendre ce circuit plus versatile en lui ajoutant une entrée T qui arrête le compteur quand elle est 0 et l'active quand elle est 1. La table de transition devient celle représentée à la figure 3.17. La figure 3.18 décrit les transitions obtenues. Le nombre voisin de chaque flèche représente l'entrée T. Il ne nous reste qu'à exprimer D en fonction de T et Q comme le montrent la figure 3.19, ce qui conduit à l'obtention du circuit représenté à la figure 3.20.

Synthèse d'un compteur synchrone par 5

Nous allons concevoir un compteur synchrone par 5 qui suit l'ordre binaire naturel et qui sera réalisé au moyen de bascules D. Comme il y a plus de 4 états, nous aurons besoin de 3 bascules. Sa table de vérité sera celle de la figure On obtient finalement le circuit de la figure ci-dessous : On remarque que comme il y a trois bascules, il y a en principe 8 états possibles. Nous en utilisons 5. Les trois autres sont possibles mais ne sons pas définis ici. En pratique, il faudrait prendre garde que ces états mènent à l'un des 5 utilisés. Autrement, si on arrive à un de ces états, par exemple, au moment de la mise en tension du circuit, ce dernier serait paralysé.

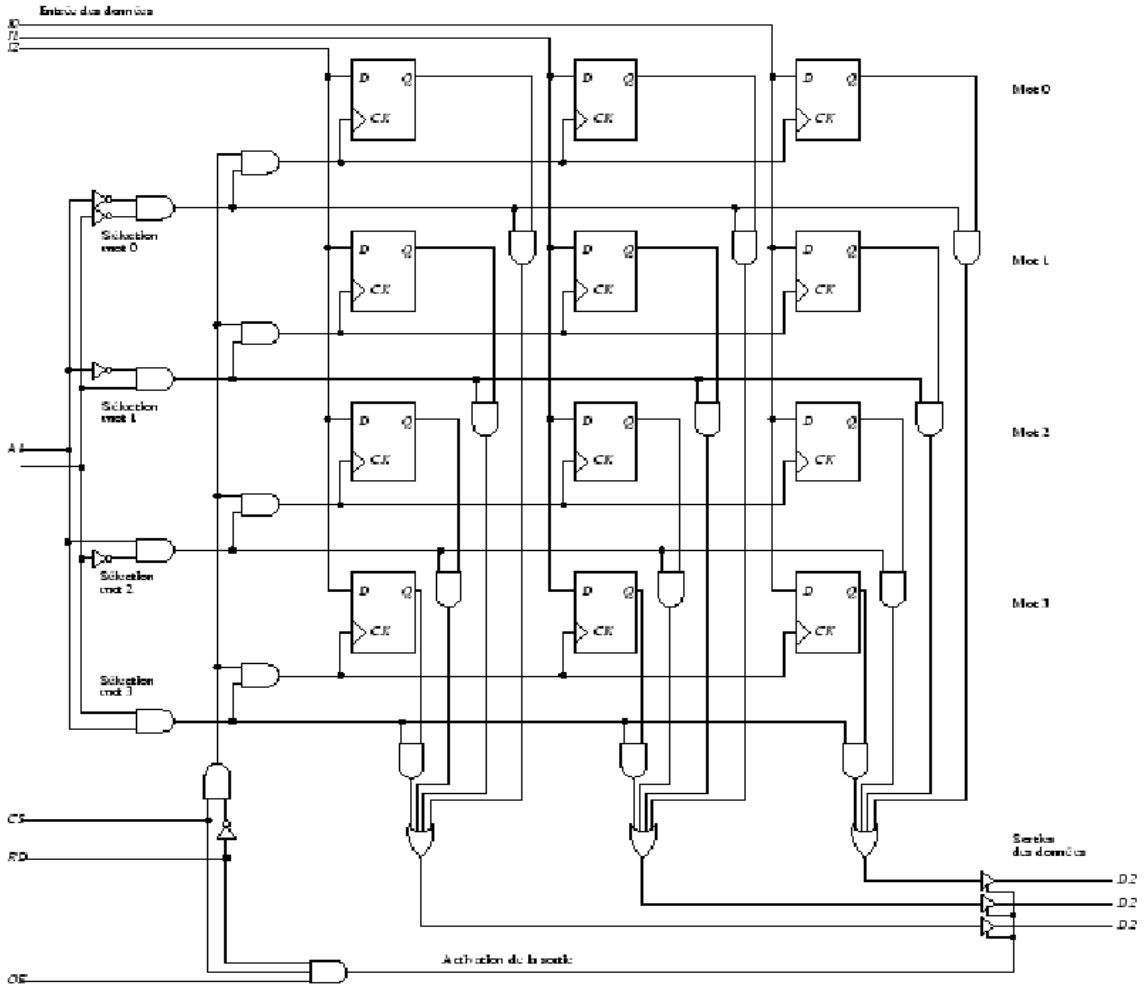


FIGURE 3.14 – Exemple de mémoire

FIGURE 3.15 – Diagramme de transitions

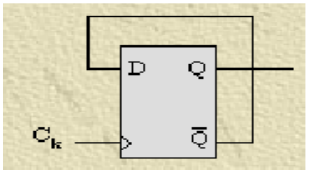


FIGURE 3.16 – Circuit synchrone assoc

Entrée présente	Etat présent	Etat futur	
T	Q	Q^+	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

FIGURE 3.17 – Table de transitions

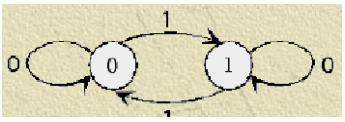


FIGURE 3.18 – Circuit synchrone associé

T \ Q	0	1
0	0	1
1	1	0

D

Donc $D = \overline{T}Q + T\overline{Q}$
$= T \oplus Q$

FIGURE 3.19 – Table de transitions

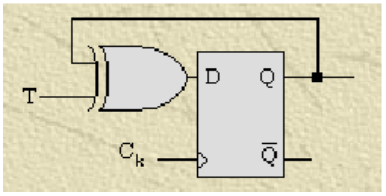


FIGURE 3.20 – Circuit synchrone assoc

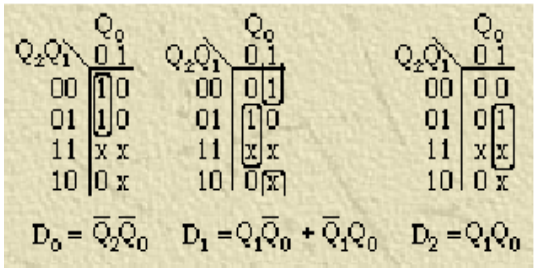


FIGURE 3.21 – Table

État présent $Q_2\,Q_1\,Q_0$	État futur $Q_2\,Q_1\,Q_0$	Entrées présentes $D_2\,D_1\,D_0$
0 0 0	0 0 1	0 0 1
0 0 1	0 1 0	0 1 0
0 1 0	0 1 1	0 1 1
0 1 1	1 0 0	1 0 0
1 0 0	0 0 0	0 0 0

FIGURE 3.22 – Diagramme

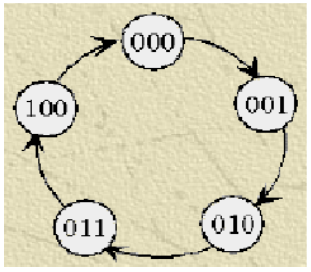


FIGURE 3.23 – Circuit synchrone associé

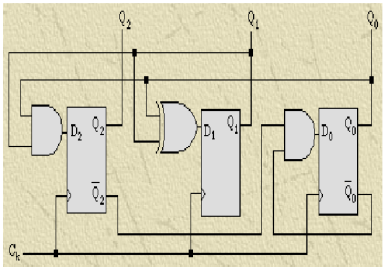


FIGURE 3.24 – Circuit synchrone assoc

Table des figures

Liste des tableaux