

Université Alger 1
Faculté des Sciences
Département Mathématiques et Informatique

Codification et représentation de l'information

Mahseur Mohammed



2016/2017

Objectifs du cours

Le cours de Codage de l'Information est un cours proposé aux étudiants en première année de licence ou technicien en informatique, électronique, électrotechnique,...

Il a pour but d'initier les étudiants aux différentes problématiques du codage de l'information :

1. représentation des données : les nombres entiers ou non, les caractères, les images, ...
2. connaissance générale des codes.
3. Une vision générale sur les circuits logiques...

Introduction

Quelle que soit la nature de l'information traitée par un ordinateur (image, son, texte, vidéo), elle est toujours représentée sous la forme d'un ensemble de nombres écrits en base 2, par exemple 01001011.

Le terme bit (b minuscule dans les notations) signifie « **binary digit** », c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique. Il est possible de représenter physiquement cette information binaire par un phénomène physique (un signal électrique, magnétique...), qui, au-delà d'un certain seuil, correspond à la valeur 1.

Le codage de l'information permet d'établir une correspondance sans ambiguïté entre une représentation (dite externe) d'une information et une autre représentation (dite interne : sous forme binaire) de la même information, suivant un ensemble de règles précises.

I Chapitre 1 : Codification et représentation des nombres

I.1 Les bases de numérotation

Soit I un ensemble d'informations, soit $A = \{a_1, a_2, \dots, a_b\}$ un ensemble fini de symboles appelé **alphabet**. Les a_i sont appelés **caractères** de A . un ensemble ordonné de caractères est appelé **mot**. La base du codage est le cardinal de l'ensemble A . pour un codage de longueur fixe n est une base b on peut coder b^n mots.

Plusieurs bases sont possibles : 10 la plus utilisée par les êtres humains, 2 et 8 et 16 sont utilisées par les machines.

I.1.1 Décimal

Ce système est inspiré des dix doigts de l'être humain, basé sur dix symboles, de 0 à 9 (base 10), avec des unités supérieures (dizaine, centaine, etc.)

I.1.2 Binaire

Puisque l'algèbre booléenne est à la base de l'électronique numérique. Deux symboles suffisent : 0 et 1 (base 2).

I.1.3 Hexadécimal

Du fait de sa simplicité d'utilisation et de représentation pour les mots machines (il est bien plus simple d'utilisation que le binaire). Il faut alors six symboles supplémentaires (base 16) : A (qui représente le 10), B (11), C (12), D (13), E (14) et F (15)

I.1.4 Octal

Les nombres sont représentés sous forme de combinaison de chiffres parmi les suivants : 0, 1, 2, 3, 4, 5, 6, 7 (base 8)

I.1.5 Changement de bases (transcodage)

I.1.5.1 Conversion décimale → base b

Convertir un nombre décimal $(N)_{10}$ vers la base b se fait par la conversion de la partie entière à part et la conversion de la partie décimale à part :

La partie entière est convertie par divisions euclidiennes successives sur b jusqu'obtention d'un résultat de la division égale à zéro.

Le résultat de la conversion est obtenu par la composition des restes de la division

La partie décimale est convertie par multiplications successives de la partie décimale par la base b, jusqu'obtention d'une partie fractionnaire nulle ou bien atteindre la précision désirée, le résultat de la conversion est obtenu par la composition de la partie entière des résultats.

Exemple 1 : $(143,526)_{10} = (?)_2$ avec une précision de sept chiffres après la virgule

$$\begin{array}{r}
 143 \quad \left| \begin{array}{l} 2 \\ \hline 71 \\ \hline 1 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 35 \\ \hline 1 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 17 \\ \hline 1 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 8 \\ \hline 0 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 4 \\ \hline 0 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 2 \\ \hline 0 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 1 \\ \hline 1 \end{array} \right| \left| \begin{array}{l} 2 \\ \hline 0 \end{array} \right|
 \end{array}$$

Donc $(143)_{10} = (10001111)_2$

$0,526 \times 2 = \underline{1},052$

$0,052 \times 2 = \underline{0},104$

$0,104 \times 2 = \underline{0},208$

$0,208 \times 2 = \underline{0},416$

$0,416 \times 2 = \underline{0},832$

$0,832 \times 2 = \underline{1},664$

$0,664 \times 2 = \underline{1},328$

Donc $(0,526)_{10} = (0,1000011)_2$

Donc $(143,526)_{10} = (10001111,1000011)_2$

Exemple 2 : $(297,358)_{10} = (?)_8$ avec une précision de cinq chiffres après la virgule

$$\begin{array}{r}
 297 \quad \left| \begin{array}{l} 8 \\ \hline 37 \\ \hline 5 \end{array} \right| \left| \begin{array}{l} 8 \\ \hline 4 \\ \hline 4 \end{array} \right| \left| \begin{array}{l} 8 \\ \hline 0 \end{array} \right|
 \end{array}$$

Donc $(297)_{10} = (451)_8$

$0,358 \times 8 = \underline{2},864$

$0,864 \times 8 = \underline{6},912$

$0,912 \times 8 = \underline{7},296$

$0,296 \times 8 = \underline{2},368$

$0,368 \times 8 = \underline{2},944$

Donc $(0,358)_{10} = (0,26722)_8$

D'où $(297,358)_{10} = (451,26722)_8$

- *Soustraction*

0-0=0 ; 1-0=1 ; 1-1=0 ; 0-1 =1 avec '1' emprunté

Exemple :

$$\begin{array}{r} + \quad 10110 \\ \quad 01100 \\ \hline = \quad 01010 \end{array}$$

- *Multiplication*

0*0=0 ; 1*0=0*1=0 ; 1*1=1

Exemple :

$$\begin{array}{r} * \quad 10110 \\ \quad 101 \\ \hline = \quad 10110 \\ + \quad 00000 \cdot \\ \quad 10110 \cdot \cdot \\ \hline = \quad 1101110 \end{array}$$

- *Division*

1/1=1 ; 0/1=0 ;

Exemple

$$\begin{array}{r} 10010 \quad | \quad 10 \\ 10 \quad | \quad 01001 \\ 00 \\ 001 \\ 0010 \\ 0 \end{array}$$

I.2.1.2 Entiers signés

Ce sont des nombres possédant un signe + ou -, Il existe 3 méthodes pour les représenter :

I.2.1.2.a Signe et valeur absolue (SVA)

Si un nombre est représenté sur n bits, alors la valeur absolue du nombre est codée sur (n-1) bits et le signe est codé sur le bit du poids fort, par convention 0 représente un nombre positif et 1 un nombre négatif tels que :

- L'intervalle des nombres représentables sur n bits en SVA est : $[-(2^{n-1}-1), + (2^{n-1}-1)]$
- Le nombre de représentations possibles sur n bits est 2^n .

Cette représentation paraît simple mais elle souffre de problème de complication des opérations arithmétiques.

I.2.1.2.b Complément à 1 (C1)

Dans cette représentation : le premier bit est réservé pour le signe et si le nombre est positif alors il garde son format, sinon (il est négatif) alors chaque bit est inversé (0 devient 1 et 1 devient 0).

Tels que :

- L'intervalle des nombres représentables sur n bits en C1 est : $[-(2^{n-1}-1), + (2^{n-1}-1)]$

- Le nombre de représentations possibles sur n bits est 2^n .

Exemple : $(-5)_{10} \rightarrow (1101)_2$ en SVA $\rightarrow (1010)_2$ en C1

• *Addition (soustraction) en C1*

Elle se base sur le principe suivant :

- Si aucune retenue n'est générée par le bit de signe, le résultat est correct, et il est représenté en C1
- Sinon, elle sera enlevée et additionnée au résultat de l'opération, celui-ci est représenté en C1

exemple 1 : $+13-4 = +13+(-4) = (01101)_{SVA} + (10100)_{SVA} = (01101)_{C1} + (11011)_{C1} = (\underline{1}01000)_{C1}$

on remarque une retenue générée par le bit de signe donc elle sera ajoutée : $(\underline{1}01000+1)_{C1} = (01001)_{C1}$

C'est un nombre positif donc sa forme en C1 = sa forme naturelle = $(+1001)_2 = (+9)_{10}$

Exemple 2 : $-13+4 = (11101)_{SVA} + (00100)_{SVA} = (10010)_{C1} + (00100)_{C1} = (10110)_{C1}$

On remarque qu'il n'y a pas de retenue générée par le bit de signe et le résultat est un nombre négatif en C1 pour le transformer en forme naturelle il faut remplacer le 1 (bit de signe) par le signe (-) puis inverser le reste des bits, $(10110)_{C1} = (-1001)_2 = (-9)_{10}$

I.2.1.2.c Complément à 2 (C2)

Dans cette représentation : le premier bit est réservé pour le signe, et si le nombre est positif alors il garde son format, sinon (il est négatif) il est transformé en C1 puis ajouté a 1.

Le nombre de représentations possibles sur n bits est 2^n . Ainsi que l'intervalle des nombres représentables sur n bits en C2 est : $[-(2^{n-1}), + (2^{n-1}-1)]$

Exemple : $(-5)_{10} \rightarrow (1101)_2$ en SVA $\rightarrow (1010)_2$ en C1 $\rightarrow (1011)_2$ en C2

• *Addition en complément à 2*

Elle se base sur le principe suivant :

- S'il y a une retenue générée par le bit de signe, elle est ignorée et le résultat est en C2
- Sinon le résultat est correct et il est représenté en C2.

Exemple 1 : $+13-4 = +13+(-4) = (01101)_{SVA} + (10100)_{SVA} = (01101)_{C2} + (11100)_{C2} = (\underline{1}01001)_{C2}$

on remarque une retenue générée par le bit de signe donc il sera négligée : $(\underline{1}01001)_{C2} = (01001)_{C2}$

C'est un nombre positif donc sa forme en C2 = sa forme naturelle = $(+1001)_2 = (+9)_{10}$

Exemple 2 : $-13+4 = (11101)_{SVA} + (00100)_{SVA} = (10011)_{C2} + (00100)_{C2} = (10111)_{C2}$

On remarque qu'il n'y a pas de retenue générée par le bit de signe et le résultat est un nombre négatif en C2 pour le transformer en forme naturelle il faut le compléter en C2 car $C2(C2(n))=n$, donc $(10111)_{C2} = (11001)_2 = (-9)_{10}$

I.2.2 Représentation des nombres réels

Les formats de représentations des nombres réels sont :

I.2.2.1 Format virgule fixe

Utilisé par les premières machines, possède une partie 'entière' et une partie 'décimale' séparées par une virgule. La position de la virgule est fixe d'où le nom.

Exemple : 54,25(10) ; 10,001(2) ; A1,F0B(16)

I.2.2.2 Format virgule flottante

Utilisé actuellement sur machine, défini par : $\pm m \cdot b^e$

- un signe + ou -
- une mantisse m (en virgule fixe)
- un exposant e (un entier relative)
- une base b (2,8,10,16,...)

Exemple : $(3,25)_{10} = (32,5 \times 10^{-1})_{10} = (325 \times 10^{-2})_{10} = (0,325 \times 10^1)_{10}$

La conversion de 3,25 en binaire donne

$$\begin{array}{r|l}
 3 & 2 \\
 \hline
 1 & 1 \\
 & \underline{1} \\
 & 0
 \end{array}$$

0,25x2=0,5

0,5x2=1,0

Donc $(3,25)_{10}=(11,01)_2$ (en virgule fixe)

$= (1,101x2^1)_2 = (0,1101x2^2)_2 = (110,1x2^{-1})_2$ (en virgule flottante)

Plusieurs manières de représenter un nombre réel en virgule flottante (différentes valeurs de m et e)

Donc il faut une **normalisation**

I.2.2.3 Codage en Virgule Flottante Normalisé

$X \pm 1, M \cdot 2^{Eb}$

SM	Eb	M
----	----	---

1bit **p** bits **q** bits

SM : signe de la mantisse, il est codé sur 1 bit ayant le poids fort : le signe - : bit 1 ; le signe + : bit 0

Eb : Exposant biaisé, il est placé avant la mantisse pour simplifier la comparaison ; codé sur **p** bits et biaisé pour assurer d'être positif (ajout de $2^{p-1}-1$)

M : Mantisse normalisée: la virgule est placée après le bit à 1 ayant le poids fort; elle est codée sur **q** bits

Exemple : $11,01 = 1,101 \times 2^{-1}$; donc M =101

I.2.2.4 Le standard IEEE 754 (1985)

(IEEE, que l'on peut prononcer « i 3 e » : *Institute of Electrical and Electronics Engineers*)

Cette norme propose la codification suivante :

I.2.2.4.a Simple précision sur 32 bits :

Signe	Eb	M
1bit	8 bits	23 bits

- 1 bit de signe de la mantisse
- 8 bits pour l'exposant
- 23 bits pour la mantisse

I.2.2.4.b Double précision sur 64 bits :

SM	Eb	M
1bit	11 bits	52 bits

- 1 bit de signe de la mantisse
- 11 bits pour l'exposant
- 52 bits pour la mantisse

Exemple1 : $(35,5)_{10} = (?)_{IEE\ 754\ simple\ précision}$

$(35,5)_{10} = (100011,1)_2$ (virgule fixe) $= (1,000111x2^5)_2$ (virgule flottante)

$Eb = E + 2^{p-1} - 1 = 5 + 2^{8-1} - 1 = 5 + 127 = 132 = (10000100)_2$

$1, M = 1,000111 \rightarrow M = 000111000...$

SM=0 ; c'est un nombre positif

0	10000100	000111000000000000000000
1bit	8 bits	23 bits

Exemple2 : $(01000000111100000000000000000000)_{IEE\ 754\ simple\ précision} = (X)_{10}$

SM=0 donc c'est nombre positif

$Eb = 10000001 = 129$

Et on a : $E_b = E + 2^{8-1} - 1 = E + 127 \rightarrow E = E_b - 127 = 129 - 127 = 2$

$M = 111000000000000000000000 = 111$

Donc $X = (1,111 \times 2^2)$ en virgule flottante = $(111,1)$ en virgule fixe = $(7,5)_{10}$

- *Addition*

Elle se fait en trois étapes :

- Dénormaliser les deux nombres afin d'avoir le même exposant (le plus élevé)
- Additionner les mantisses
- Normaliser le résultat

Exemple : $7 - 2,25 = 4,75$

$(+7)_{10} = (0 \ 1000001 \ 110000000000000000000000)_{IEE754}$

$(-2,25)_{10} = (1 \ 1000000 \ 001000000000000000000000)_{IEE754}$

- Dénormalisation :

$(+7)_{10} = +1,11 \times 2^2$

$(-2,25)_{10} = -1,001 \times 2^1 = 0,1001 \times 2^2$

- Addition des mantisse $(+1,11) + (-0,1001) = (+1,0011)_2$
- Normalisation

Signe positif $\rightarrow SM = 0$

Mantisse = $1,0011 \rightarrow M = 0011$

$E = 2 \rightarrow E_b = 2 + 127 = 129 = (1000001)_2$

Donc les résultat est $(0 \ 1000001 \ 001100000000000000000000)_{IEE754}$

- *Multiplication*

Elle se fait en quatre étapes :

- Dénormaliser les deux nombres (exposants naturels)
- Additionner les exposants naturels
- Multiplier les mantisses
- Normaliser le résultat

Exemple

$7 \times 9,5 = ?$

$(7)_{10} = (111)_2 = (0 \ 1000001 \ 110000000000000000000000)_{IEE754}$

$(9,5)_{10} = (1001,1)_2 = (0 \ 1000010 \ 001100000000000000000000)_{IEE754}$

- Dénormalisation :

$(0 \ 1000001 \ 110000000000000000000000)_{IEE754} = 1,11 \times 2^2$

$(0 \ 1000010 \ 001100000000000000000000)_{IEE754} = 1,0011 \times 2^3$

- Addition des exposants : $E = 2 + 3 = 5$
- Multiplication des mantisses : $1,11 \times 1,0011 = 10,00101$
- Normalisation

$10,00101 \times 2^5 = 1,000101 \times 2^6$

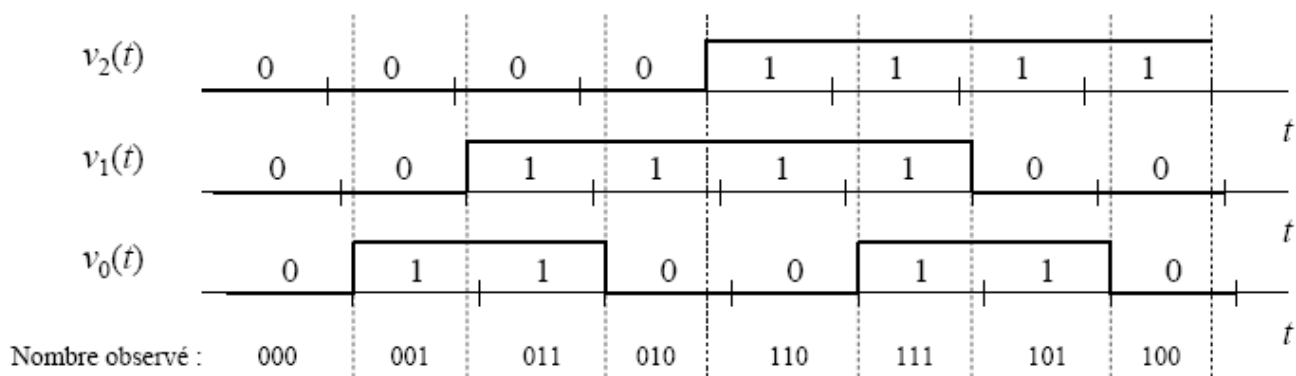
$SM = 0$; $E = 6 \rightarrow E_b = 6 + 127 = 133 = (1000101)_2$; $M = 000101$

Donc les résultat est $(0 \ 1000101 \ 000101000000000000000000)_{IEE754}$

Décimal	Binaire	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000
.	.	.
.	.	.

Le code de Gray règle le problème de compte que nous avons vu précédemment, comme l'illustre la figure suivante :

(CAS RÉEL)



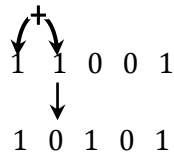
On construit le code de Gray en recopiant les bits de façon symétrique (effet miroir) et en procédant de façon itérative jusqu'au bit désiré.

0	0	0	0	00
1	0	1	0	01
	1	1	0	11
	1	0	1	10
	1	1	1	11
	1	0	1	01
	1	0	0	00
Itération 1	Itération 2		Itération 3	

II.2.1 Conversion Binaire \rightarrow Gray

Pour obtenir le code de Gray à partir du binaire pur, on conserve le premier bit du binaire pur, c'est-à-dire le premier bit du binaire pur reste le premier bit du binaire réfléchi(Gray). On additionne le premier bit du binaire pur au deuxième bit du binaire pur pour obtenir le deuxième bit du binaire réfléchi. On additionne ensuite le deuxième bit du binaire pur au troisième bit du binaire pur pour obtenir le troisième bit du binaire réfléchi et ainsi de suite. Dans toutes ces additions, on considère toujours $0+0=0$, $0+1=1$ et $1+1=0$.

Exemple : $(11001)_2 = (?)_{Gray}$



II.2.2 Conversion Gray \rightarrow Binaire

Pour obtenir le code de binaire à partir du code Gray, on conserve le premier bit du Gray, c'est-à-dire le premier bit du Gray reste le premier bit du binaire. On additionne le premier bit du binaire au deuxième bit du Gray pur pour obtenir le deuxième bit du binaire, On additionne ensuite le deuxième bit du binaire au troisième bit du Gray pur pour obtenir le troisième bit du binaire et ainsi de suite.

En général : si $(b_n b_{n-1} \dots b_1 b_0)_2 = (g_n g_{n-1} \dots g_1 g_0)_{GRAY}$ alors $b_n = g_n$; $g_i = b_{i+1} + b_i$ et $b_i = b_{i+1} + g_i$ (les retenues sont négligées)

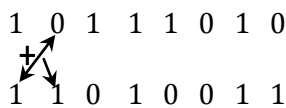
Ces trois règles sont suffisantes pour la conversion Binaire \rightarrow Gray et Gray \rightarrow Binaire

Exemple $(10111010)_{Gray} = (?)_2$

$$b_7 = g_7 = 1; \quad b_6 = b_7 + g_6 = 1 + 0 = 1; \quad b_5 = b_6 + g_5 = 1 + 0 = 1; \quad b_4 = b_5 + g_4 = 1 + 1 = 0; \quad b_3 = b_4 + g_3 = 0 + 1 = 1;$$

$$b_2 = b_3 + g_2 = 1 + 0 = 1; \quad b_1 = b_2 + g_1 = 1 + 1 = 0; \quad b_0 = b_1 + g_0 = 0 + 0 = 0$$

Donc $(10111010)_{Gray} = (10101100)_2$



II.3 Le code ASCII

(American Standard Code for Information Interchange)Code Américain Standard pour l'Echange d'Informations.Le jeu de caractères codés ASCII (American Standard Code for Information

Interchange) ou code américain normalisé pour l'échange d'informations, initié par la compagnie américaine *Bellen* 1961, c'est la norme de codage de caractères alphanumériques en informatique la plus connue, la plus ancienne et la plus largement compatible. Le code ASCII (on prononce généralement « aski ») est un code sur 7 bits (valeurs 0 à 127), il permet de définir :

- des codes de contrôle non imprimables (0 à 31) : indicateurs de saut de ligne, de fin de texte, codes de contrôle de périphériques, ...
- des caractères imprimables universels : lettres minuscules (65 à 90) et majuscules (97 à 122), chiffres, symboles,...

Binaire					b6	0	0	0	0	1	1	1	1
					b5	0	0	1	1	0	0	1	1
Hexadécimal					b4	0	1	0	1	0	1	0	1
						0	1	2	3	4	5	6	7
b3	b2	b1	b0	Décimal		0	16	32	48	64	80	96	112
0	0	0	0	0	+0	NUL (NUL)	TC7 (DEL)	SP	0	@	P	-	p
0	0	0	1	1	+1	TC1 (SOH)	DC1	!	1	A	Q	a	q
0	0	1	0	2	+2	TC2 (STX)	DC2	..	2	B	R	b	r
0	0	1	1	3	+3	TC3 (ETX)	DC3	#	3	C	S	c	s
0	1	0	0	4	+4	TC4 (EOT)	DC4	\$	4	D	T	d	t
0	1	0	1	5	+5	TC5 (ENC)	TC8 (NAK)	%	5	E	U	e	u
0	1	1	0	6	+6	TC6 (ACK)	TC9 (SYN)	&	6	F	V	f	v
0	1	1	1	7	+7	BEL	TC 10 (ETB)	'	7	G	W	g	w
1	0	0	0	8	+8	FE0 (BS)	CAN	(8	H	X	h	x
1	0	0	1	9	+9	FE1 (HT)	EM)	9	I	Y	i	y
1	0	1	0	A	+10	FE2 (LF)	SUB	*	:	J	Z	j	z
1	0	1	1	B	+11	FE3 (VT)	ESC	+	;	K	[k	é
1	1	0	0	C	+12	FE4 (FF)	IS4 (FS)	.	<	L	\	l	ù
1	1	0	1	D	+13	FE5 (CR)	IS3 (GS)	-	=	M]	m	è
1	1	1	0	E	+14	SO	IS2 (RS)	.	>	N	^	n	-
1	1	1	1	F	+15	SI	IS1 (US)	/	?	O	_	o	DEL

Exemple : la signification de cette suite binaire: 100110110010010110010011000001100010110110 est M12016

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...). Cette norme s'appelle ISO-8859.

II.3.1 La norme IOS-8859

À la fin des années 1980, sont définis les codages ISO-8859 qui sont diverses extensions du codage ASCII permettant de coder en plus des lettres latines accentuées et d'introduire (en partie) certains autres alphabets (arabe, cyrillique, grec, hébreu).

Tous ces codages codent les caractères avec des mots binaires de 8 bits.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ù	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	Ṭ	226	E2	Γ
131	83	â	163	A3	ú	195	C3	ṭ	227	E3	Π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	ã	166	A6	ª	198	C6	‡	230	E6	μ
135	87	ç	167	A7	º	199	C7	‡	231	E7	ι
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	φ
137	89	ë	169	A9	ƒ	201	C9	Ṛ	233	E9	θ
138	8A	è	170	AA	ƒ	202	CA	Ṛ	234	EA	Ω
139	8B	ÿ	171	AB	½	203	CB	Ṛ	235	EB	δ
140	8C	î	172	AC	¼	204	CC	‡	236	EC	∞
141	8D	ì	173	AD	ı	205	CD	=	237	ED	∞
142	8E	Ă	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Ą	175	AF	»	207	CF	Ł	239	EF	∩
144	90	É	176	B0	⋯	208	DO	Ł	240	FO	≡
145	91	æ	177	B1	⋮	209	D1	Ṛ	241	F1	±
146	92	Æ	178	B2	⋮	210	D2	Ṛ	242	F2	≥
147	93	ó	179	B3		211	D3	Ł	243	F3	≤
148	94	ö	180	B4	†	212	D4	Ł	244	F4	[
149	95	ò	181	B5	‡	213	D5	Ṛ	245	F5]
150	96	û	182	B6	‡	214	D6	Ṛ	246	F6	÷
151	97	ù	183	B7	π	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	ƒ	216	D8	‡	248	F8	°
153	99	Ö	185	B9	‡	217	D9	ƒ	249	F9	•
154	9A	Û	186	BA		218	DA	ƒ	250	FA	·
155	9B	ø	187	BB	η	219	DB	■	251	FB	√
156	9C	£	188	BC	∫	220	DC	■	252	FC	∂
157	9D	¥	189	BD	∫	221	DD	■	253	FD	ε
158	9E	ℳ	190	BE	∫	222	DE	■	254	FE	■
159	9F	f	191	BF	ƒ	223	DF	■	255	FF	□

II.4 L'Unicode

Dans les années 1990, le projet Unicode de codage de tous les alphabets est né. Unicode est une famille de codages, dont UTF-8 est un membre.

C'est une autre norme que l'ASCII, qui présente l'avantage de proposer une version unifiée des différents encodages de caractères complétant l'ASCII mais aussi de permettre l'encodage de caractères autres que ceux de l'alphabet latin. Unicode définit des dizaines de milliers de codes, mais les 128 premiers restent compatibles avec ASCII. Dans le codage UTF-8, chaque point de code est codé sur une suite d'un à quatre octets. Il a été conçu pour être compatible avec certains logiciels originellement prévus pour traiter des caractères d'un seul octet.

II.4.1 Le codage UTF-8

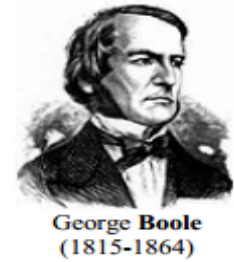
C'est un codage de longueur variable, Certains caractères sont codés sur un seul octet, ce sont les 128 caractères du codage ASCII. Le bit de poids fort des codes de ces caractères est toujours un 0. Les autres caractères peuvent être codés sur 2, 3 ou 4 octets. Les bits de poids fort du premier octet codant l'un de ces caractères indique toujours le nombre d'octets codant. Si le caractère est codé sur 2 octets, les trois premiers bits du premier octet sont 110. S'il est codé sur 3 octets, alors les quatre premiers bits du premier octet sont 1110. Et enfin s'il est codé sur 4 octets les cinq premiers bits du premier octet sont 11110. Les deux bits de poids fort des octets qui suivent le premier octet codant sont toujours 10. Voir la table 2.1 pour un résumé.

Nbre octets codant	Format du code
1	0xxxxxxx
2	110xxxxx 10xxxxxx
3	1110xxxx 10xxxxxx 10xxxxxx
4	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

III Algèbre de Boole

III.1 Introduction

L'industrie des circuits électroniques de toutes machines numériques est basée sur la conception des fonctions logiques, ces dernières réalisent des opérations de base (addition, négation, multiplication, comparaison, incrémentation,...). Une fonction logique est représentée par une équation mathématique en respectant une algèbre nommée Algèbre de Boole développée par le mathématicien anglais George Boole (1815-1864).



III.2 Terminologie

III.2.1 Logique combinatoire

La valeur de sortie d'une fonction dépend uniquement des valeurs des variables d'entrées et ne dépend pas des états antérieurs de la fonction (pas de mémorisation).

III.2.2 Variable logique

Grandeur représentée par un symbole, pouvant prendre deux valeurs logiques distinctes.

III.2.3 Etat logique

Valeur d'une variable logique, représentée par les chiffres « 0 » ou « 1 » ou les lettres « L » ou « H ». (H=High; L=Low).

III.2.4 Opérateur logique

Il existe trois opérateurs de base : Non, Ou, Et.

III.2.5 Porte logique

Un circuit électronique élémentaire permettant de réaliser la fonction d'un opérateur logique

III.3 Opérateurs de base

III.3.1 La négation (non)

Opérateur unaire (appliqué sur une variable) qui inverse la valeur d'une variable (0 devient 1 et 1 devient 0) ; il est représenté par une barre au-dessus de la variable. exemple: \bar{A} , il est défini comme suit :



E	S
0	1
1	0

III.3.2 La disjonction (ou)

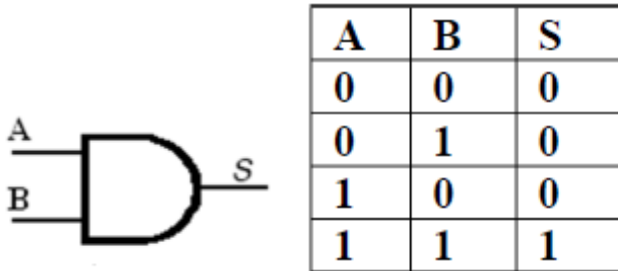
Opérateur binaire (appliqué sur deux variables) qui fait la somme logique entre deux variables, il donne 1 si au moins une des variables en entrées est en état 1, il est représenté par +, exemple $A+B$, il est défini comme suit :



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

III.3.3 La conjonction (et)

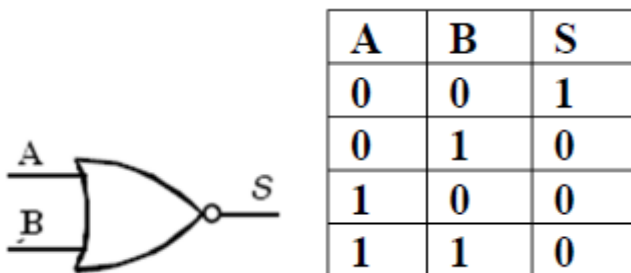
Opérateur binaire qui fait le produit logique entre deux variables, il retourne 1 si et seulement si les deux variables en entrées sont à l'État 1, il est représenté par un point, exemple $A.B$, il est défini comme suit :



III.4 Opérateurs composés

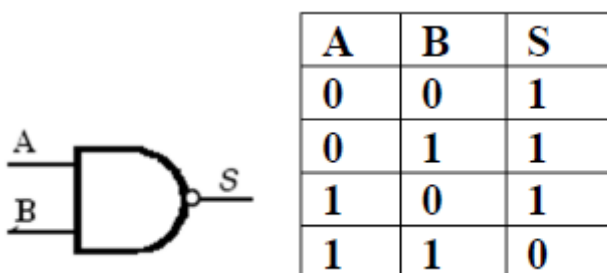
III.4.1 Opérateur NOR (Non Ou)

Opérateur binaire qui fait la négation de l'opérateur OU, il retourne 1 si toutes les variables en entrées sont à 0, il est représenté par une flèche vers le bas, exemple $\overline{A + B} = A \downarrow B$, il est défini comme suit :



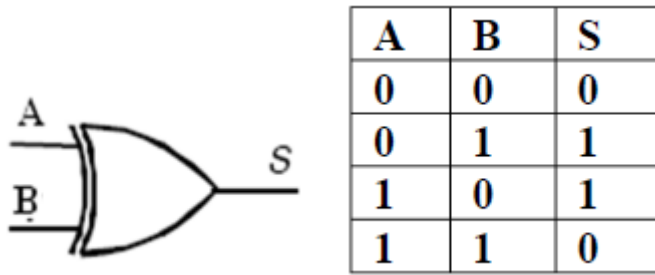
III.4.2 Opérateur NAND (Non Et)

Opérateur binaire qui fait la négation de l'opérateur ET, il retourne 1 si au moins une variable d'entrées égale à 0, il est représenté par une flèche vers le haut, exemple $\overline{A.B} = A \uparrow B$, il est défini comme suit :



III.4.3 Opérateur XOR (OU Exclusif)

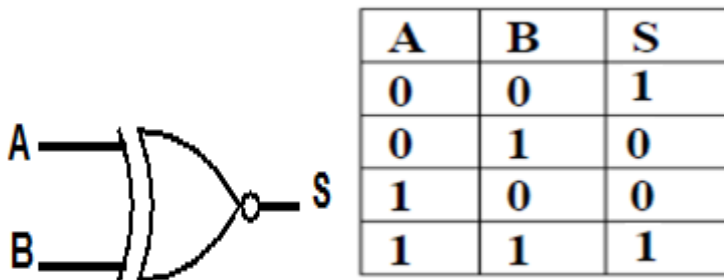
Opérateur binaire qui vérifie si les deux variables en entrées sont différentes, il retourne 1 si et seulement si une variable d'entrées égale à 1 et l'autre égale à 0, il est représenté par un plus encerclé, Exemple $A \oplus B = A.\bar{B} + \bar{A}.B$, il est défini comme suit :



Remarque $A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$

III.4.4 Opérateur XNOR

Opérateur binaire qui vérifie si les deux variables en entrées sont différentes, il retourne 1 si et seulement si les deux variables ont le même état, il est défini comme suit : $\overline{A \oplus B} = AB + \overline{A}\overline{B}$



III.5 Evaluation des expressions booléennes

III.5.1 Postulats

Disjonction (+)	Conjonction (.)
$1+1=1$	$1.1=1$
$1+0=0+1=1$	$1.0=0.1=0$
$0+0=0$	$0.0=0$
$\overline{\overline{1}} = 0$	$\overline{\overline{0}} = 1$

III.5.2 Axiomes

propriété	Disjonction (+)	Conjonction (.)
Commutativité	$A+B=B+A$	$A.B=B.A$
Associativité	$A+(B+C)=(A+B)+C$	$A.(B.C)=(A.B).C$
Distributivité	$A+(B.C)=(A+B).(A+C)$	$A.(B+C)=A.B+A.C$
Elément neutre	$A+0=A$	$A.1=A$
Elément absorbant	$A+1=1$	$A.0=0$
Complémentation	$A + \overline{A} = 1$	$A.\overline{A} = 0$
Idempotence	$A+A=A$	$A.A=A$
Involution	$\overline{\overline{A}} = A$	$\overline{\overline{\overline{A}}} = \overline{A}$

III.5.3 Théorèmes

III.5.3.1 Théorème de DE Morgan

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

III.5.3.2 Théorème d'inclusion

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

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



Augustus de Morgan
(1806-1871)

III.5.3.3 Théorème d'allégement

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

III.5.3.4 Théorème d'absorption

$$A+A.B=A ; A.(A+B)=A$$

III.5.3.5 Théorème de dualité

Chaque axiome et chaque postulat possède un équivalent dual, où les éléments 0 sont remplacés par des 1, les 1 par des 0, les (·) par des (+) et vice versa. Aussi, tout théorème de l'algèbre de Boole a son équivalent dual.

$$\text{Exemple : } A+A.B=A \Leftrightarrow A.(A+B)=A ; A+1=1 \Leftrightarrow A.0=0$$

III.5.3.6 Réflexion de Michaud

$$\text{Si } A=B \oplus C \text{ alors } B=A \oplus C$$

III.6 Représentation des fonctions Booléennes

III.6.1 Définitions

- On appelle fonction logique un groupes de variables reliées entre-elles par des opérateurs logiques.
- On appelle Minterme de n variables un produit logique de ces variables.
- On appelle Maxterme de n variables une somme logique de ces variables.
Avec n variables on peut construire 2^n Mintermes et 2^n Maxtermes
Exemple : $n=2$
Les mintermes sont : $A.B ; A.\bar{B} ; \bar{A}.B ; \bar{A}.\bar{B}$
Les maxtermes sont : $A + B ; A + \bar{B} ; \bar{A} + B ; \bar{A} + \bar{B}$
- On appelle forme canonique d'une fonction la forme où chaque terme (minterme ou maxterme) comporte toutes les variables de la fonction
Exemple : $f_1(A, B, C) = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C$ (Une forme canonique)
 $f_2(A, B, C) = \bar{A}.B.C + A.B + A$ (Ce n'est pas une forme canonique)
- On appelle forme canonique disjonctive (première forme canonique) d'une fonction, une fonction canonique composée d'une somme de mintermes.
Exemple : $f_1(A, B, C) = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C$
- On appelle forme canonique conjonctive (deuxième forme canonique) d'une fonction, une fonction canonique composée d'un produit de maxtermes.
Exemple : $f_2(A, B, C) = (\bar{A} + B + C).(A + \bar{B} + C).(A + B + \bar{C})$.
- Pour transformer une fonction non canonique disjonctive vers une fonction canonique disjonctive on multiplie chaque minterme de la fonction par la somme de la variable qui manque et son complément
Exemple :
$$f(A, B, C) = A.\bar{B} + B.\bar{C} + C = A.\bar{B}.(C + \bar{C}) + B.\bar{C}.(A + \bar{A}) + C.(A + \bar{A}).(B + \bar{B})$$
$$= A.\bar{B}.C + A.\bar{B}.\bar{C} + B.\bar{C}.A + B.\bar{C}.\bar{A} + C.A.B + C.A.\bar{B} + C.\bar{A}.B + C.\bar{A}.\bar{B}$$
- On appelle forme décimale de la forme canonique disjonctive la somme des nombres décimaux équivalents aux mintermes de la fonction tels que \bar{A} est représenté par 0 et A est représenté par 1, le minterme $\bar{A}\bar{B}\bar{C}$ est représenté par $(010)_2 = (2)_{10}$
Exemple : $f_1(A, B, C) = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C = \sum(3,5,6,7)$
- On appelle forme décimale de la forme canonique conjonctive le produit des nombres décimaux équivalents aux maxtermes de la fonction tels que \bar{A} est représenté par 1 et A est représenté par 0, le maxterme $(\bar{A} + B + \bar{C})$ est représenté par $(101)_2 = (5)_{10}$
Exemple : $f_2(A, B, C) = (A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C) = \prod(1,2,4)$

III.6.2 .Table de vérité

C'est une table qui représente les différentes valeurs d'une fonction selon les différentes combinaisons de ses variables d'entrée, exemple : soit la fonction suivante MAJORITE (A,B,C) qui retourne 1 si la majorité des entrées sont à l'état 1, sa table de vérité est la suivante :

A	B	C	MAJORITE(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

← $\bar{A}.B.C$

← $A.\bar{B}.C$

← $A.B.\bar{C}$

← $A.B.C$

III.6.3 Extraction d'une fonction logique à partir de la table de vérité

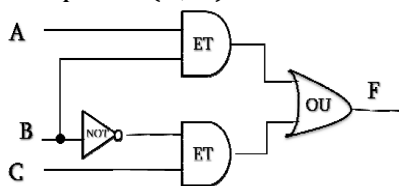
- Pour obtenir l'expression logique sous la première forme canonique (disjonctive), on établit l'équation logique de la fonction en sommant les mintermes pour lesquels la sortie vaut 1. D'après la table de vérité on peut déduire la formule de la fonction :
 $MAJORITE(A, B, C) = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C.$
- Pour obtenir l'expression logique sous la deuxième forme canonique (conjonctive), on établit l'équation logique de la fonction \bar{f} en sommant les mintermes pour lesquels la sortie vaut 0, puis on calcul $\bar{\bar{f}}$, Exemple :

$$\begin{aligned} \overline{MAJORITE(A, B, C)} &= \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} + A.\bar{B}.\bar{C} \\ \overline{MAJORITE(A, B, C)} &= \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} + A.\bar{B}.\bar{C} \\ MAJORITE(A, B, C) &= \overline{\bar{A}.\bar{B}.\bar{C}.\bar{A}.\bar{B}.C.\bar{A}.B.\bar{C}.A.\bar{B}.\bar{C}} \\ &= (A + B + C).(A + B + \bar{C}).(A + \bar{B} + C).(\bar{A} + B + C) \end{aligned}$$

III.6.4 Logigramme

C'est la représentation symbolique normalisée d'une fonction à l'aide des portes logiques (dans notre cours, la norme ANSI est utilisée : American National Standards Institute)

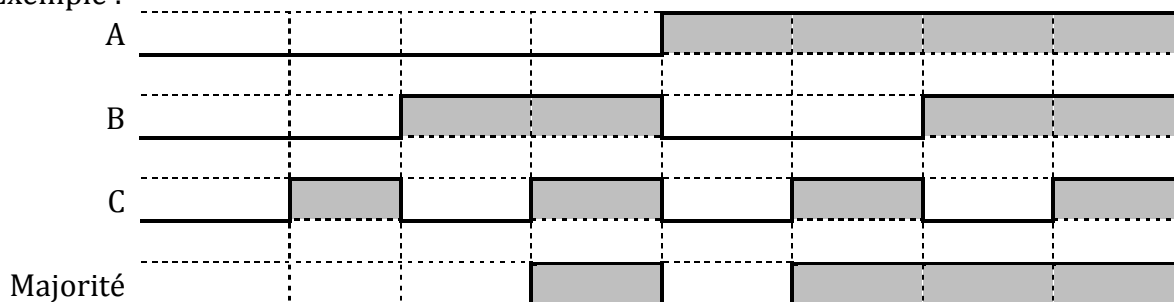
Exemple : $F(A, B) = A.B + \bar{B}.C$



III.6.5 Chronogramme

C'est le graphe représentant l'évolution des variables d'entrée et de sortie au cours du temps, tels que l'état supérieur représente le '1' et l'état inférieur représente le '0'.

Exemple :



III.7 Simplification des fonctions

La simplification d'une fonction consiste à l'écrire avec le minimum de termes et les plus simples possibles afin de la réaliser avec moins d'éléments électroniques (portes logiques) sans changer sa valeur, deux méthodes : simplification algébrique et simplification graphique (tableau de **Karnaugh**).

III.7.1 Simplification algébrique

Il n'y a pas une démarche bien précise à suivre, il faut appliquer des simplifications, des factorisations, des distributions,...en utilisant les postulats et les axiomes et les théorèmes de l'algèbre de Boole afin d'éliminer ou de minimiser des termes, exemple :

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

III.7.2 Simplification graphique : Tableau de Karnaugh

C'est une méthode inventée par **Maurice Karnaugh** en 1954 et qui sert à simplifier des équations logiques ou à trouver l'équation logique correspondant à une table de vérité. Elle fonctionne très bien avec 2 ou 3 ou 4 variables, beaucoup moins bien avec 5 ou 6 variables, et plus du tout au-delà !

Les cases à l'extrémité de la table représentent les différentes combinaisons des variables d'entrées, elles sont rangées de manière adjacente (le bit à droite d'une case égale au bit à gauche de la case suivante exemple **(00,01,11,10)**).

Les cases à l'intérieur de la table représentent les différentes valeurs de la fonction logique.



Maurice Karnaugh
(1924-)

III.7.2.1 Adjacence des cases internes

Deux cases adjacentes dans le tableau de **Karnaugh** correspondent à des combinaisons différant d'un seul bit. 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.

Exemple : les cases adjacentes de la case x sont les cases grises

AB CD	00	01	11	10
00				
01			x	
11				
10				

AB CD	00	01	11	10
00				
01				x
11				
10				

AB CD	00	01	11	10
00				
01				
11				
10		x		

AB CD	00	01	11	10
00				x
01				
11				
10				

III.7.2.2 Regroupement des cases adjacentes

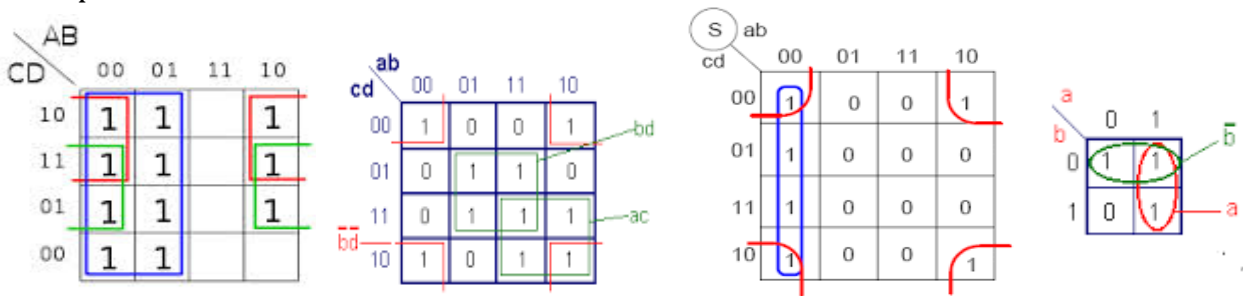
Après remplissage du tableau de **Karnaugh** à partir de la table de vérité, tel que chaque case représente une ligne de la table de vérité, on procède au regroupement des cases adjacentes contenant l'état '1' selon les règles suivantes :

- Un regroupement doit prendre la forme d'un carré ou d'un rectangle composé d'un nombre de cases en puissance de deux (1 ;2 ;4 ;8 ;16).
- Il faut prendre le minimum de groupes composés de maximum des cases.
- Une case à l'état '1' doit figurer au moins dans un regroupement, comme elle peut exister dans plusieurs regroupements.
- Arrêter le regroupement si toutes les cases à '1' sont regroupées.

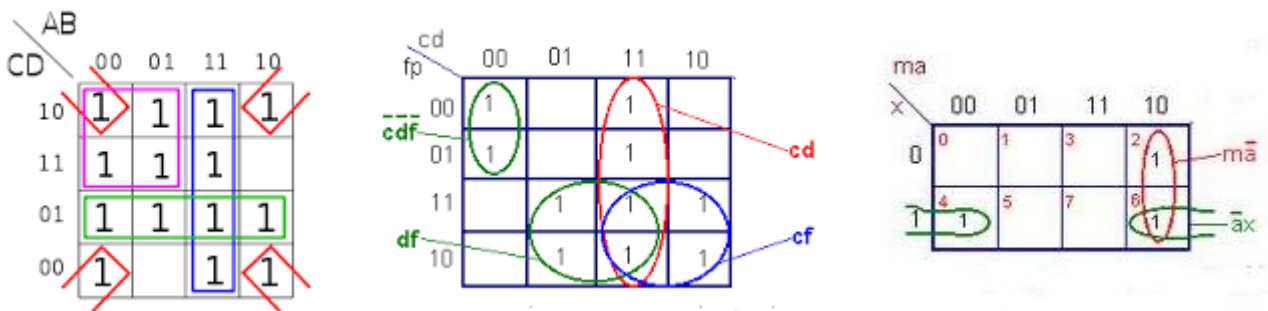
III.7.2.3 Extraction de la fonction simplifiée

La fonction est représentée généralement en somme de produits, où chaque regroupement représente un minterme composé de produit des variables d'entrées, une variable qui change d'état dans le regroupement doit être éliminée du minterme.

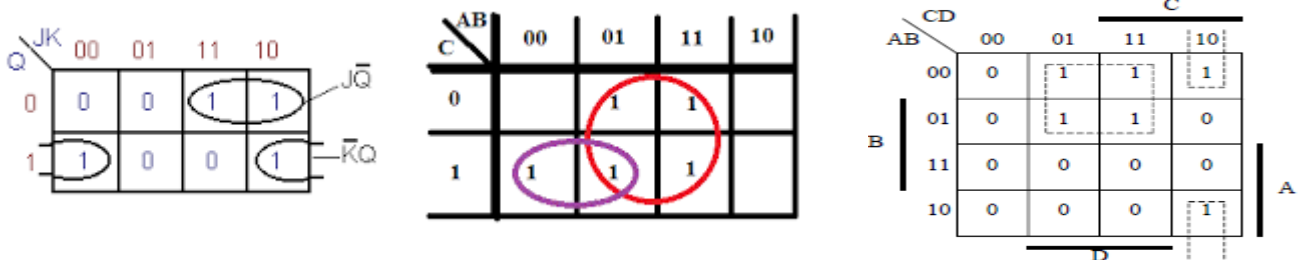
Exemples :



$$f = \bar{A} + \bar{B}.C + \bar{B}.D \quad f = b.d + \bar{b}.\bar{d} + a.c.S = \bar{a}.\bar{b} + \bar{b}.\bar{d}f = \bar{b} + a$$



$$f = A.B + \bar{C}.D + \bar{A}.C + \bar{B}.\bar{D} \quad f = c.d + d.f + c.f + \bar{c}.\bar{d}.\bar{f}f = m.\bar{a} + \bar{a}.x$$



$$f = J.\bar{Q} + \bar{K}.Q \quad f = B + \bar{A}.Cf = \bar{A}.D + \bar{B}.C.\bar{D}$$

III.7.2.4 Cas de cinq variables

Le tableau de Karnaugh ne représente que quatre variables au maximum, pour simplifier une fonction $f(a,b,c,d,e)$ de cinq variables il faut suivre les étapes suivantes :

- Écrire la fonction f dans la première forme canonique
- Transformer f en deux fonctions : $f(a,b,c,d,e) = e.f(a,b,c,d,1) + \bar{e}.f(a,b,c,d,0)$
- Simplifier les deux fonctions $f(a,b,c,d,1)$ et $f(a,b,c,d,0)$ qui contiennent quatre variables par le tableau de Karnaugh
- Réécrire f en fonction de ces deux fonctions simplifiées

Exemple : $f(a,b,c,d,e) = a.b.d + \bar{a}.b.d.e + a.b.\bar{c}.\bar{d}.\bar{e} + a.b.c.\bar{d}.\bar{e}$

- Écriture en forme canonique :

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

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

- Transformation de f en deux fonctions

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

- Simplifications de chaque fonction :

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

$b.d$

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

$a.b$

- Donc $f(a, b, c, d, e) = b.d.e + a.b.\bar{e}$

III.7.2.5 Cas de six variables

Pour une fonction de six variables $S(a,b,c,d,e,f)$ il suffit de suivre les mêmes étapes qu'avec cinq variables sauf que la fonction est transformée de la manière suivantes :

$$\begin{aligned}
 S(a, b, c, d, e, f) &= e.f.S(a, b, c, d, 1, 1) + e.\bar{f}.S(a, b, c, d, 1, 0) + \bar{e}.f.S(a, b, c, d, 0, 1) \\
 &+ \bar{e}.\bar{f}.S(a, b, c, d, 0, 0)
 \end{aligned}$$

Table des matières

Objectifs du cours	1
Introduction.....	1
I Chapitre1 : Codification et représentation des nombres.....	1
I.1 Les bases de numérotation	1
I.1.1 Décimal	1
I.1.2 Binaire.....	1
I.1.3 Hexadécimal	1
I.1.4 Octal.....	1
I.1.5 Changement de bases (transcodage)	2
I.2 Représentation interne des données :.....	3
I.2.1 Représentation des Entiers.....	3
I.2.2 Représentation des nombres réels.....	5
II Chapitre 2 : Codification et représentation α - Numérique	8
II.1 Le code BCD (Binary Coded Decimal)	8
II.2 LE CODE GRAY OU CODE BINAIRE REFLECHI	9
II.2.1 Conversion Binaire \rightarrow Gray	11
II.2.2 Conversion Gray \rightarrow Binaire	11
II.3 Le code ASCII.....	11
II.3.1 La norme IOS-8859	12
II.4 L'Unicode	14
II.4.1 Le codage UTF-8	14
III Algèbre de Boole	6
III.1 Introduction.....	6
III.2 Terminologie.....	6
III.2.1 Logique combinatoire.....	6
III.2.2 Variable logique	6
III.2.3 Etat logique.....	6
III.2.4 Opérateur logique	6
III.2.5 Porte logique	6
III.3 Opérateurs de base	6
III.3.1 La négation (non).....	6
III.3.2 La disjonction (ou)	6
III.3.3 La conjonction (et).....	7
III.4 Opérateurs composés.....	7
III.4.1 Opérateur NOR (Non Ou)	7
III.4.2 Opérateur NAND (Non Et)	7
III.4.3 Opérateur XOR (OU Exclusif)	7
III.4.4 Opérateur XNOR	8
III.5 Evaluation des expressions booléennes	8
III.5.1 Postulats	8
III.5.2 Axiomes	8
III.5.3 Théorèmes.....	8
III.6 Représentation des fonctions Booléennes.....	9
III.6.1 Définitions.....	9
III.6.2 .Table de vérité	10
III.6.3 Extraction d'une fonction logique à partir de la table de vérité	10
III.6.4 Logigramme	10
III.6.5 Chronogramme.....	10
III.7 Simplification des fonctions.....	11
III.7.1 Simplification algébrique	11
III.7.2 Simplification graphique : Tableau de Karnaugh.....	11