Université de Tunis
ISEFC

Cours AU1 :

SYSTEMES LOGIQUES COMBINATOIRES Automatique de Base

Octobre 2002

1. Champ d'application de l'automatisme

2. Systèmes de numérations

la base 2

la base 16 (hexadécimal)

Décimal Codé en Binaire (DCB ou BCD en anglais)

Binaire réfléchi (code GRAY)

3. Les fonctions logique

Définition
Les fonctions logiques de Bases :
Propriétés :

a) **Identités logiques remarquables :**
b) **Propriétés de l'Algèbre logique :**
c) **Théorèmes de DEMORGAN :**

Les Opérateurs Logiques Universels :

a-La fonction ET-NON :
b- la fonction OU-NON :

3-1 Logique des prédicats
3-2 Algèbre de BOOLE

3.2.1 axiomes

3.2.2 Théorèmes

3.2.3 Décomposition en NAND - NOR

3.2.4 Fonctions booléennes à n variables

3.2.5 Passage ET/OU en NAND

3.2.6 Applications de l'algèbre de BOOLE

3.2.6.1 logique des prédicats

3.2.6.2 Circuits électriques

3.2.7 Aléas technologiques

3.2.8 Electronique (portes)

4. Minimisation des fonctions logiques

4.1. Méthode algébrique de simplification
4.2. Méthode graphique de Karnaugh
4.3. Méthode tabulaire de Quine-MC Cluskey

5. Fonctions combinatoires usuelles et principaux types de circuits MSI

5.1. Comparaison entre information

5.1.1. Principe

5.1.2. Extension pour n éléments binaires

5.1.3. Exemple de circuits MSI : 7485

5.2. décodage binaire

5.2.1. Principe

5.2.2. Exemple de circuits MSI : 74 138

5.3. décodage BCD-7 segments

5.3.1. Principe

5.3.2. Exemple de circuits MSI :

5.4. Multiplexage de données

5.4.1. Principe

5.4.2. Exemple de circuits MSI :74151

5.5. l'Unité Arithmétique et Logique

5.5.1. Principe

5.5.2. Exemple de circuits MSI :74181

5.5.3. Réalisation d'un additionneur et d'un soustracteur:

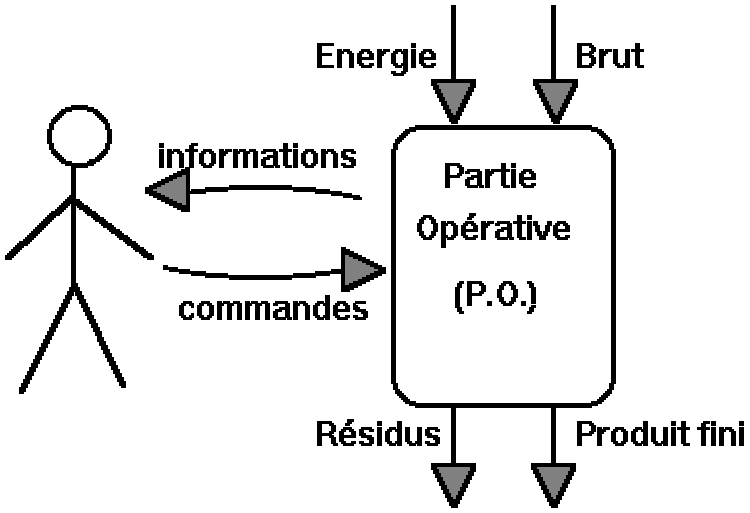
FONCTIONS LOGIQUES COMBINATOIRES
Automatique de Base

1. Champ d'application de l'automatisme - vocabulaire

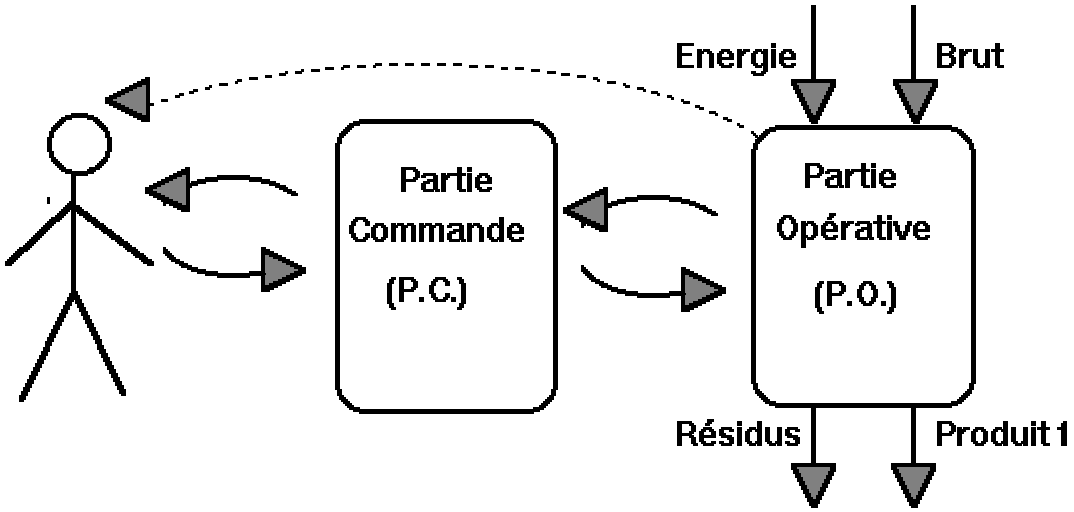
L'automatisme consiste en l'étude de la commande de systèmes industriels.

La première amélioration des conditions de travail a été de remplacer l'énergie humaine fournie par l'ouvrier par une machine (P.O.)

L'opérateur commande la machine, et regarde le résultat obtenu. Il adapte ses commandes en fonction du déroulement du processus.



L'automatisme débute lorsque l'on intercale entre l'opérateur et la P.O. une P.C. qui prend certaines décisions (gestion automatique des cas les plus simples et les plus courants).



La **P.C.** lit les informations sur la **P.O.** par l'intermédiaire de capteurs, et commande les actionneurs de la P.O. Le but est de prendre en compte par la P.C. tout ce qui est répétitif et simple, en laissant à l'opérateur les tâches nobles de réflexion. La P.C. doit nécessairement "tout savoir" : toute information ou commande, même non traitée par elle, devrait passer par elle. Il reste néanmoins quelques phénomènes difficiles à mesurer, ou dont la mesure coûte trop cher par rapport à la probabilité qu'ils se produisent, ou non prévus. Pour cela, l'opérateur- contrôleur reste nécessaire.

Nous nous identifierons toujours à la partie commande. Nous appellerons donc entrées les commandes de l'opérateur ainsi que les informations reçues des capteurs. Nous appellerons sorties les commandes envoyées à la P.O. ainsi que les informations transmises à l'opérateur.

On peut "classifier" les différents cas. La première distinction qui a été faite a été de séparer le Tout Ou Rien (allumé ou non, appuyé ou non, ouvert ou fermé... représenté par 0 ou 1) de l'analogique (grandeurs représentées par une valeur réelle, comme l'électronique par exemple). Désormais, le numérique (gestion de l'analogique par une combinaison de composants ToR, donc regroupement de zéros et de uns pour former des valeurs), tend à englober tous les cas, en premier lieu par son coût plus faible, en second lieu par les possibilités de programmation donc d'évolution.

Un autre distinction peut se faire entre le combinatoire (les sorties dépendent uniquement de l'état actuel des entrées) et le séquentiel (les sorties dépendent des entrées et de l'historique, c'est à dire de ce qui s'est passé auparavant). Le séquentiel en numérique est souvent appelé automatique, en analogique plutôt asservissement.

2. Systèmes de numérations

la base 2

On peut représenter des nombres par une combinaison de zéros et de uns. Chaque chiffre binaire (BInary digIT) est appelé BIT. 8 bits forment un octet (BYTE). Mais plusieurs codifications sont envisageables. La plus utilisée est le binaire (ou binaire naturel en cas d'ambiguïté).

passage binaire (indice b) -> décimal (indice d) :

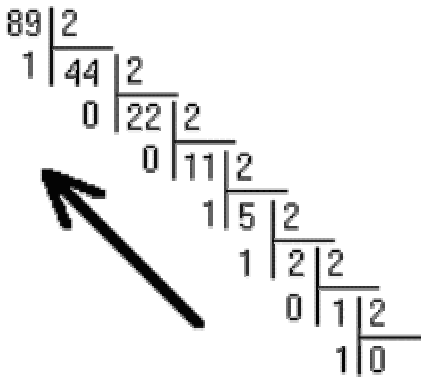
1011001_b représente :

$$1x2^6 +0x2^5 +1x2^4 +1x2^3 +0x2^2 +0x2^1 +1x2^0$$

$$= 1x64 +0x32 +1x16 +1x8 +0x4 +0x2 +1x1$$

$$= 89_d$$

à l'inverse, pour transformer 89_d en binaire, on peut utiliser la méthode des divisions successives par 2 : on divise successivement par 2 jusqu'à un résultat de 0, les restes successifs (de bas en haut) forment le nombre binaire.



De tête, je ferai : 89 = **1**x64 reste 25 donc **0**x32, **1**x16 reste 9, **1**x8 reste 1 donc **0**x4, **0**x2, **1**x1.

la base 16 (hexadécimal)

On utilise les chiffres 0 à 9 puis les lettres A à F. $3A5_h$ vaut $3x16^2 + 10x16^1 + 5x16^0 = 3x256 + 160 + 5 = 933_d$. On passe de l'hexa au décimal par divisions successives par 16. Transformer de l'hexadécimal en binaire est enfantin : il suffit de remplacer chaque chiffre par sa valeur binaire sur quatre bits : $3A5_h = 0011\ 1010\ 0101_b$ (on peut vérifier que ça vaut 933_d). En effet, 001110100101_b

$= 0.2^{11} +0.2^{10} +1.2^9 +1.2^8 +1.2^7 +0.2^6 +1.2^5 +0.2^4 +0.2^3 +1.2^2 +0.2^1 +1.2^0$

$= (0.2^3 +0.2^2 +1.2^1 +1.2^0)2^8 +(1.2^3 +0.2^2 +1.2^1 +0.2^0)2^4 +0.2^3 +1.2^2 +0.2^1 +1.2^0$

$= 0011_b x2^8 +1010_b x2^4 +0101_b x2^0$

$= 3_d x16^2+10_d x16+5_d$

$=3A5_h$

Contrairement à ce que beaucoup de gens croient, aucune machine ne compte en hexadécimal. Elles travaillent toutes en binaire, et ne se servent de l'hexa que

pour dialoguer avec nous (nous nous trompons trop souvent dans de longues listes de 0 et 1).

Décimal Codé en Binaire (DCB ou BCD en anglais)

Si vous achetez un voltmètre numérique (69F en supermarché), la valeur mesurée est transmise à l'afficheur en numérique. Mais elle est auparavant transformée en décimal, chaque chiffre décimal est transmis à un afficheur en binaire naturel (sur 4 bits). C'est le BCD : la juxtaposition des valeurs binaires (sur quatre bits) des chiffres décimaux. Donc 583_d se notera 0101 1000 0011_{bcd}. Cette codification pose deux problèmes principaux :

- un certain nombre de combinaisons ne sont pas utilisées (celles qui correspondent à A à F en hexa). Sur 8 bits on représente les nombres de 0 à 99 au lieu de 0 à 255 en binaire. Sur 16 bits, on se limite à 9999 au lieu de 65535...
- les calculs sont compliqués : rien que pour faire un programme d'incrémentation (ajouter 1), il faut ajouter 1 aux 4 bits de droite, si on obtient 1010 (10_d), on les remplace par 0000 et on ajoute 1 aux quatre bits suivants (dizaines), sans oublier de vérifier si l'on ne passe pas à la centaine suivante...

Dès qu'il y a des calculs à effectuer, les systèmes numériques traduisent les nombres BCD en binaire dès leur acquisition, les résultats seront transformés en BCD au moment de leur sortie. On peut remarquer que pour transformer un nombre binaire en décimal (bcd), l'ordinateur est obligé de faire des divisions successives par 1010 (10 en binaire)

Binaire réfléchi (code GRAY)

On désire qu'en passant d'un nombre à son suivant (+1) ou précédent (-1), on n'aie qu'un seul bit qui change. On désire de plus que les zéro rajoutés à gauche d'un nombre ne soient pas significatifs. Sur deux bits, on utilisera les codes 00,

01, 11 puis 10. Sur 3 bits, on gardera les mêmes premiers codes (précédés d'un zéro). La combinaison suivante débutera donc obligatoirement par 1, donc les deux autres bits ne peuvent pas changer. On continuera à prendre les mêmes codes, en ordre inverse, débutant par 1 : 110, 111, 101 et 100. En passant à 4 bits, on précède ces 8 cas d'un 0, les 8 suivants étant les mêmes, dans l'ordre inverse, précédés d'un 1. Ce codage est utilisé dans les cas où des valeurs ne peuvent varier que par incrémentation ou décrémentation : si l'on voit que plus d'un bit a changé entre deux valeurs, c'est qu'il y a eu un problème (en général le nombre a changé trop vite, le système n'a pas eu le temps de lire toutes les valeurs). Il faut par contre passer en binaire naturel pour tout autre calcul que l'incrémentation.

Un exemple est le capteur de position angulaire . Un capteur incrémental comptant des impulsions est utilisé par exemple sur les robots. C'est un disque, entaillé d'encoches régulièrement espacées, passant devant un capteur optique. Certaines impulsions trop rapprochées peuvent être "oubliées" en cas de choc par exemple, et donc occasionner un mauvais réglage. A l'initialisation et en cas de problème, on doit ramener toutes les articulations en position de repos, puis mettre les compteurs à 0, avant de pouvoir utiliser le robot. Un capteur absolu quand à lui donne toujours le position exacte (bien qu'il y ait souvent une démultiplication, le mouvement total fait plus d'un tour mais aucun choc ne fera sauter le capteur d'exactement un tour). On utilise un code binaire réfléchi car un autre codage nécessiterait, pour passer d'une valeur à la suivante, une modification simultanée de plusieurs bits

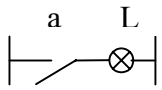
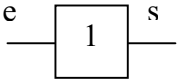
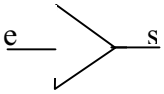
3. Les fonctions logique

***Définition* :** Un système est dit combinatoire lorsque les états des fonctions de sorties sont parfaitement définis par les combinaisons des variables d'entrées.
Les circuits combinatoires sont des circuits caractérisés par le fait qu'à chacune des combinaisons de leurs variables d'entrées ne correspond qu'une et une seule combinaison de leurs variables de soties.

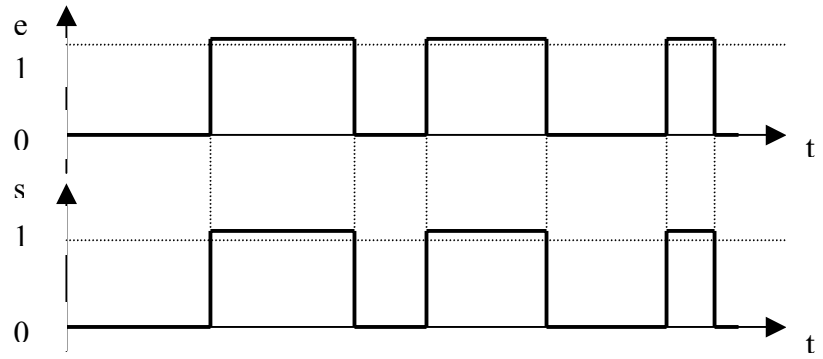
Les fonctions logiques de Bases :

a) La fonction oui :

*la sortie est à l'état 1 si, et seulement si, l'entrée est à l'état 1.

SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		
 La lampe L est à l'état 1 si	 **1** : Symbole distinctif de l'opérateur OUI.		e | s 0 | 0 1 | 1 Egalité d'état	S = e Se lit : l 'état de s est égal à l'état de e.

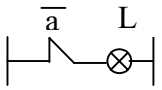
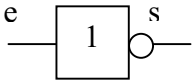
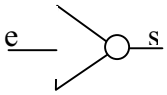
Le chronogramme ci-contre met en évidence que l'état de la sortie s est identique à l'état de l'entrée e.



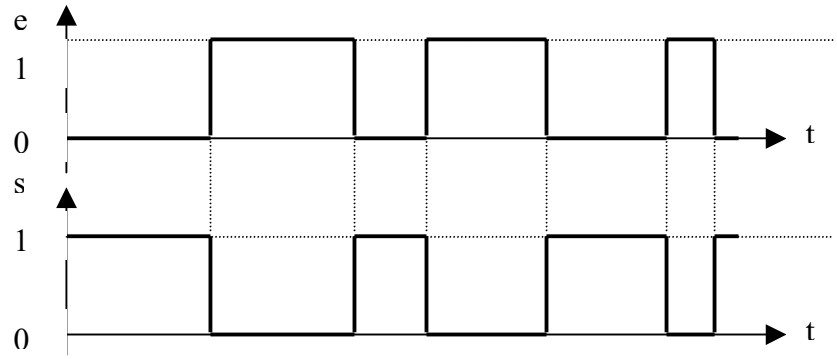
b) La fonction Non :

*la sortie est à l'état 1 si, et seulement si, l'entrée est à l'état 0.

SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		

 La lampe L est à l'état 1 ~~si~~	 1 et le cercle indiquent	 Le cercle indique	<table><tr><td>e</td><td>s</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> Complém entari-té	$S = \overline{e}$ Se lit : l 'état de s est égal à l'état de $\overline{e}$. (e barre ou non e).

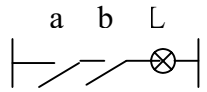
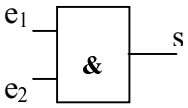
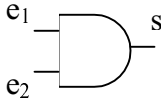
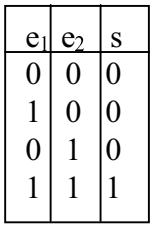
Le chronogramme ci-contre met en évidence que l'état de la sortie s est identique à l'état de l'entrée e.



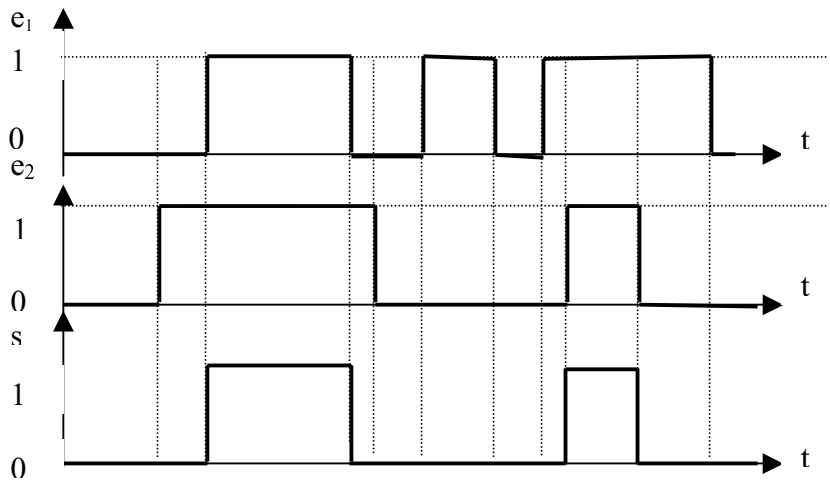
c) La fonction ET :

*la sortie est à l'état 1 si, et seulement si, toutes les entrées sont à l'état 1.

SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		

 La lampe L est à l'état 1 si et	 **&** : Symbole distinctif de l'opérateur ET.	 Se désigne AND	 Seule la dernière combinai	$S = e_1 . e_2$ Se lit : l 'état de s est égal à l'état de e_1 et à l'état de e_2 .

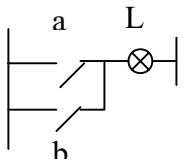
Le chronogramme ci-contre met en évidence que la sortie s est à l'état 1 si, et seulement si, les deux entrées e_1 et e_2 sont à l'état 1.



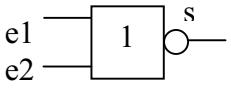
d) La fonction OU :

* La sortie est à l'état 1 si, et seulement si, une ou plusieurs entrées sont à l'état 1.

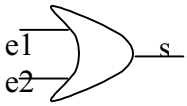
SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		



La lampe L est à l'état 1 si



≥1 :Symbole distinctif



Se désigne OR

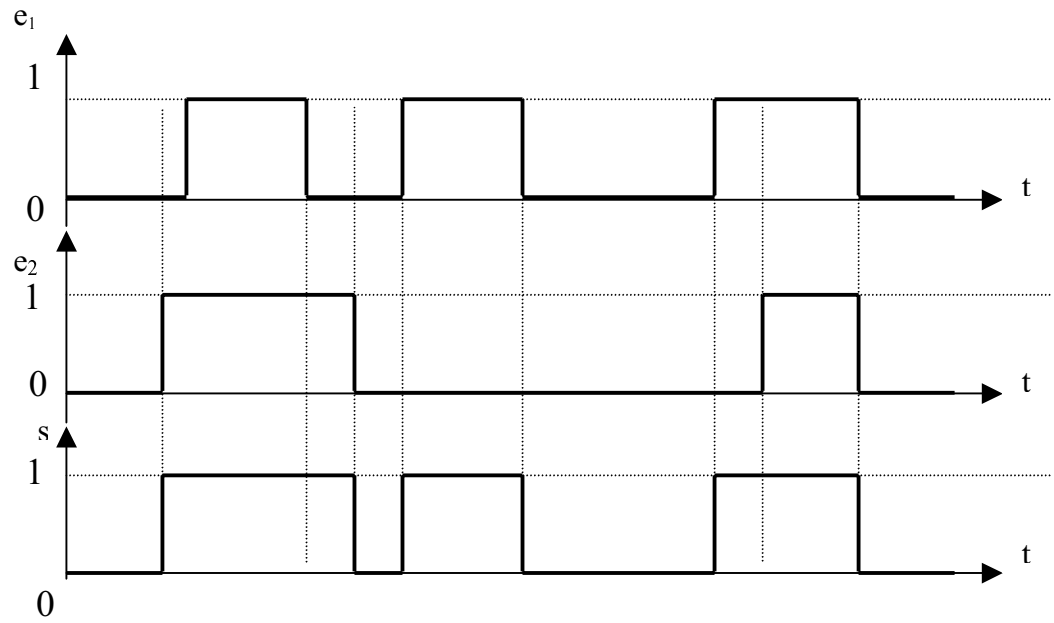
e_1	e_2	s
0	0	0
0	1	1
1	0	1
1	1	1

Les trois derniers combinai

$S = e_1 + e_2$

Se lit : l'état de s est égal à l'état de e1 ou l'état de e2. le (+) désigne la somme logique.

Le chronogramme ci-contre mit en évidence que la sortie s est à l'état 1 si, et seulement si l'entrée e1 ou l'entrée e2 sont à l'état 1.



Propriétés :

c) <u>Identités logiques remarquables :</u>

Elles permettent de simplifier les équations logiques :

- Fonction ET :
  - $e \cdot 1 = e$
  - $e \cdot 0 = 0$
  - $e \cdot \overline{e} = 0$
  - $e \cdot e = e$
- Fonction OU :

- $e + 1 = 1$
- $e + 0 = e$
- $e + \overline{e} = 1$
- $e + e = e$

d) Propriétés de l'Algèbre logique :

- Commutativité

$S = a \cdot b = b \cdot a$

$S = a + b = b + a$

- Associativité

$S = a \cdot (b + c) = (a \cdot b) \cdot c$

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

- Distributivité

$S = a \cdot (b+c) = (a.b) + (a.c)$

$S = a + b.c = (a + b) \cdot (a + c)$

c) Théorèmes de DEMORGAN :

- Le complément d'un produit logique est égal à la somme logiques des facteurs complémentés de ce produit.

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$

- Le complément d'une somme logique est égal au produit logique des termes complémentés de cette somme .

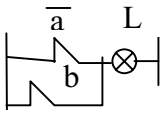
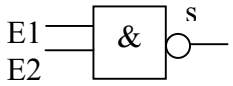
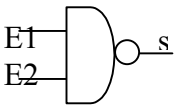
$$\overline{a + b} = \overline{a} \cdot \overline{b}$$

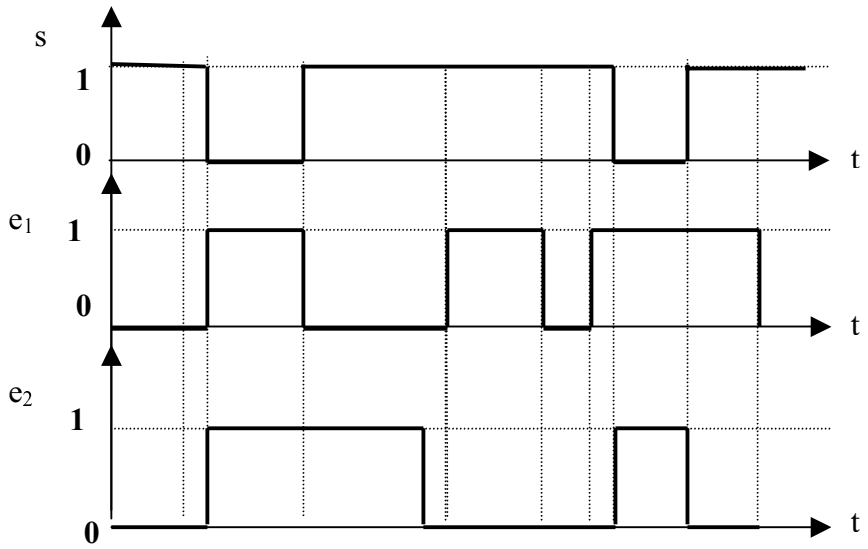
Les Opérateurs Logiques Universels :

Dans l'application du théorème de DEMORGAN,il est nécessaire de considérer les deux fonctions NON – ET et NON – OU.
A fin de réaliser une fonction logique, il faut avoir simultanément à la disposition des opérateurs ET, OU et NON. Au contraire en employant les opérateurs NON-Et et NON- OU , on peut réaliser toute fonction logique avec un seul type de ces deux derniers.
Pour ces deux opérateurs aussi on spécifie la définition, la table de vérité , les représentations symboliques, analytique, graphique et les propriétés .

a) La fonction ET-NON :

* La sortie est à l'état 0 si, et seulement si, toutes les entrées sont à l'état 1.

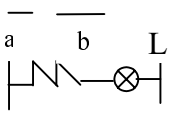
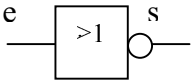
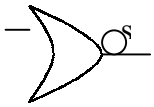
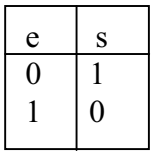
SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		
 La lampe L est à l'état 0 si	 1 et le cercle indiquent	 Le cercle indique	E1 E2 s 0 1 1 0 1 1 1 0 1 1 1 0 Complém entari-té	$S = \overline{e}$ Se lit : l 'état de s est égal à l'état de $\overline{e}$. (e barre ou non e).



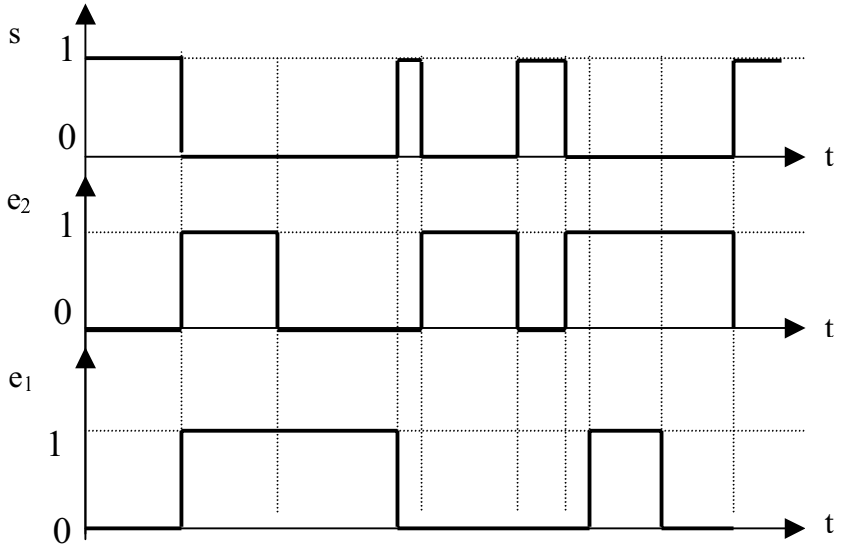
b- la fonction OU-NON :

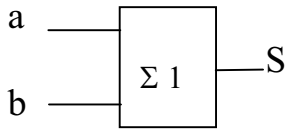
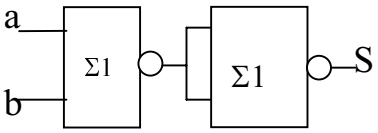
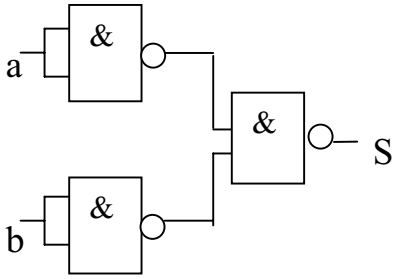
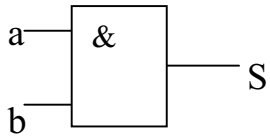
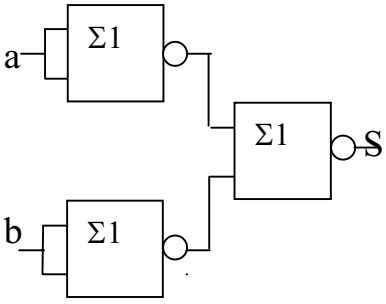
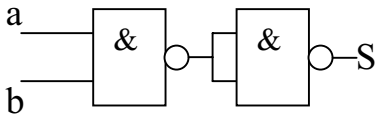
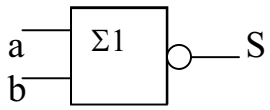
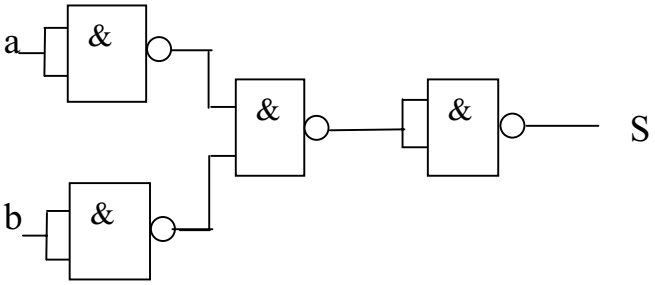
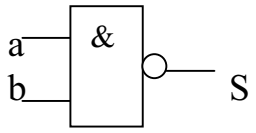
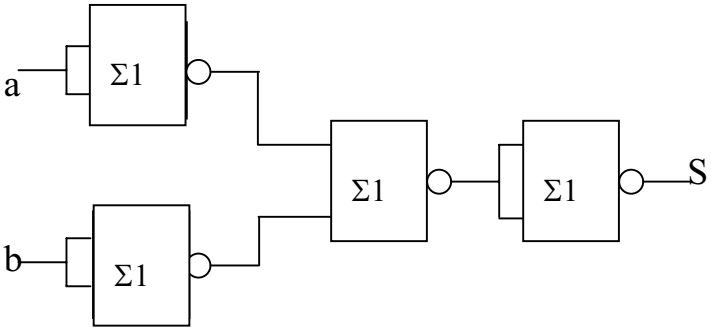
- la sortie est à l'état 1 si et seulement si toutes les entrées sont a l'état 0.

SCHEMA A CONTACTS	SYMBOLES LOGIQUES		TABLE DE VERITE	EQUATION LOGIQUE
	NF C 03-212	américain		

				$S = \overline{e}$
La lampe L est à l'état 1 si et	>=1 et le cercle indiquent la négation	Se désigne NOR	Complémentarité	Se lit : l 'état de s est égal à l'état de e. (e barre ou non e).

Le chronogramme ci-contre met en évidence que la sortie S est à l'état 1 si et seulement si les deux entrées e1 ou l'entrée e2 sont à l'état 0.



FONCTION OU	 $S = a + b$	Deux opérateurs OU-NON.  $S=\overline{\overline{a} \cdot \overline{b}} = a + b$	Trois opérateurs ET-NON. 
FONCTION ET	 $S = a \cdot b$	 $S=\overline{\overline{a} + \overline{b}} = \overline{a}\,\overline{b} = a \cdot b$	 $S=\overline{\overline{a} + \overline{b}} = a \cdot b$
FONCTION NON OU	 $S=\overline{a} + \overline{b}$	 $S=\overline{a + b} = \overline{a} \cdot \overline{b}$	
FONCTION NON ET	 $S=\overline{a} \cdot \overline{b}$	 $S=\overline{a \cdot b} = \overline{a} + \overline{b}$	

3-1 Logique des prédicats

On appelle proposition ou prédicat une "phrase" qui peut être soit vraie, soit fausse. La logique des prédicats est donc un premier exemple de Tout ou Rien (et est utilisée dans les problèmes de reconnaissance de la parole et d'analyse syntaxique).

exemples :

(P1) il pleut

(P2) 6 est supérieur à 4

on notera vrai=1, faux=0 . Donc P2=1, P1 vaut 0 ou 1 suivant les cas.

On peut avoir des propositions dépendant de variables:

X est supérieur à 4

X + Y = 0

On peut également définir des opérateurs : ET (noté . , AND ou ^), OU (+, OR, v) et complément (¯ ou /, j'utiliserai / dans ce document car la barre est trop dure à gérer). On peut alors définir pour chacune de ces opérations leur table de vérité qui définit, dans tous les cas, le résultat de l'opération :

P	Q	P.Q
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	P+Q
0	0	0
0	1	1
1	0	1
1	1	1

P	$\bar{P}$
0	1
1	0

On peut également utiliser un tableau à deux entrées pour obtenir un tableau de vérité :

P.Q	0	1
0	0	0
1	0	1

2 colonnes pour les états possibles de Q

2 lignes pour les états possibles de P

En essayant toutes les combinaisons, on peut définir 16 opérateurs binaires (fonctions de deux variables) :

P	Q	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Certains cas sont de peu d'intérêt : a (toujours faux), p (vrai), ou ne dépendent en fait que d'une variable : d (=P), f (=Q), m (=/P), k (=/Q). Les autres ont toutes un nom :

b: ET, h: OU inclusif, o: ON ou NAND, i: NI ou NOR, c: P | Q (P inhibe Q), e: Q | P (Q inhibe P), n: P => Q (implique), l: Q => P (implique), g: OU exclusif (XOR, ⊕), j: P <=> Q (équivalent).

3-2 Algèbre de BOOLE

George Boole (mathématicien anglais, 1815-1864) a démontré que si l'on peut trouver un espace dans lequel certains axiomes se vérifient, alors on se trouve dans un cas singulier, où un certain nombre de théorèmes peuvent s'appliquer.

3.2.1 axiomes

Pour qu'une algèbre puisse être dite de Boole, elle doit vérifier :

commutativité	a+b=b+a	a.b=b.a
associativité	(a+b)+c=a+(b+c)	(ab)c=a(bc)
distributivité	a(b+c)=ab+ac	a+(bc)=(a+b)(a+c)
éléments neutres	a+0=a	a.1=a

complémentation	$\bar{a}+a=1$	$\bar{a}.a=0$

3.2.2 Théorèmes

Une algèbre de Boole vérifie les théorèmes suivants :

idempotence	a+a=a	aa=a
absorption	a+ab=a	a(a+b)=a
Morgan	$\overline{a+b}=\bar{a}.\bar{b}$	$\overline{a.b}=\bar{a}+\bar{b}$
élément neutre	a+1=1	a.0=0

De plus les fonctions suivantes sont définies :

$$a \text{ XOR } b = a\bar{b}+\bar{a}b$$

$$a \text{ NOR } b = \bar{a}.\bar{b}$$

$$a \text{ NAND } b = \bar{a}+\bar{b}$$

3.2.3 Décomposition en NAND - NOR

On peut exprimer toutes les fonctions existantes uniquement à l'aide des fonctions ET, OU et NON (voir exemples précédents). Ceci est pratique dans la mesure où, en automatisme, chaque fonction est représentée par un composant (appelé PORTE). De plus ces trois fonctions correspondent aux fonctions logiques utilisées dans le langage courant. Mais on peut se limiter à 2 fonctions, ET et NON par exemple :

$$\overline{a+b}=\overline{a}.\overline{b}$$

On peut même se limiter à une seule fonction (NAND ou NOR) :

$$a.b=\overline{a \text{ NAND } b} \quad ; \quad a+b=\overline{a} \text{ NAND } \overline{b} \quad ; \quad a=\overline{a \text{ NAND } a}$$

3.2.4 Fonctions booléennes à n variables

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

On peut essayer de simplifier son équation à l'aide des relations de l'algèbre de Boole. Mais on peut aussi représenter graphiquement cette fonction par son tableau de vérité :

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

$f' = \overline{b}.c + a.c$

Ce tableau nous donne l'état de f (0 ou 1) dans tous les cas (ceci permet de ne pas oublier certains cas particuliers).Dans la pratique on remplit dans le tableau

tous les cas où f vaut 1, on complète les zéros dans les cases restantes. On peut remarquer que la fonction f' est représentée par le même tableau. f' et f sont donc égales, puisqu'ayant la même valeur dans tous les cas.

3.2.4 Passage ET/OU en NAND

$\overline{abc}$

on notera $\overline{abc}$ par NAND[a,b,c]

On décompose la fonction sous forme canonique : (comme vous pouvez le voir, je trouve que j'ai assez travaillé, je garde mes équations en mode texte bien que ce soit moins beau)

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

on peut alors écrire :

$$f = \overline{\overline{(abc)} + \overline{(abc)} + \overline{(abc)}}$$

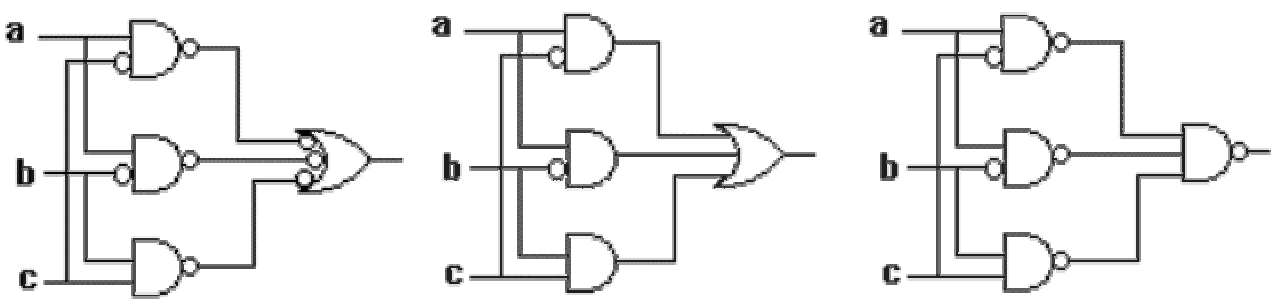
$$f = \overline{\overline{(a\bar{b}c)} \cdot \overline{(ab\bar{c})} \cdot \overline{(\bar{a}b\bar{c})}}$$

$$f = NAND[\ \overline{(a,\bar{b},c)}\ ,\ \overline{(a,b,\bar{c})}\ ,\ \overline{(\bar{a},b,\bar{c})}\ ]$$

donc :

$$f = NAND\ [\ NAND(a,\bar{b},c)\ ,\ NAND(a,b,\bar{c})\ ,\ NAND(\bar{a},b,\bar{c})\ ]$$

Dans la pratique, les schémas de fonctions définies par ET / OU se transforment facilement en réseaux composés uniquement de NAND. Il suffit d'inverser toutes les extrémités des liaisons internes :



3.2.5 Applications de l'algèbre de BOOLE

3.2.5.1 logique des prédicats

propositions, vrai, faux, et, ou, non...

exercice : calculez : NON(Pat est beau ou Pat est intelligent)

ensembles

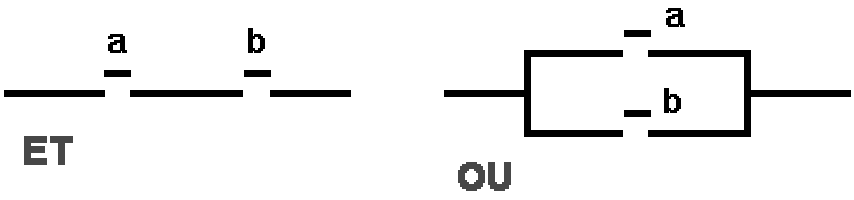
intersection ^, union U, complément

ex: absorption: $A \cap (A \cup B) = A$

3.2.5.2 Circuits électriques

0 représente un interrupteur (je devrais dire en toute rigueur contacteur) ouvert (le courant ne passe pas)
1 représente un interrupteur fermé (le courant passe)
Un contacteur normal laisse passer le courant quand on l'actionne, un contacteur inverse quand on le laisse au repos.

On effectue une fonction ET par la liaison de 2 interrupteurs (il faut appuyer sur a ET b pour que le courant passe) en série (il faut appuyer sur a ET b pour que le courant passe), la fonction OU par la liaison parallèle (il faut appuyer sur a OU b pour que le courant passe).



On peut vérifier les axiomes de l'algèbre de Boole en représentant les différents schémas

On pourrait de même vérifier ce que donnent les théorèmes (inutile de les démontrer puisque les axiomes sont vérifiés).

Applications : Soient un moteur M fonctionnant en 220 V, un relais électromagnétique R comportant le contact inverseur, 3 boutons contacts A B et C fonctionnant sous ambiance humide (tension de commande 12 V). Le moteur doit fonctionner si l'on appuie sur A et B simultanément. Toute action sur C (arrêt d'urgence bistable) doit arrêter le moteur. (solution R = a.b.c, M = R). Par mesure de sécurité (centrale nucléaire par exemple), on triple les capteurs. Ces trois capteurs (mesurant la même chose) commandent un relais. On veut que, en cas de défaillance d'un capteur, le relais continue à fonctionner mais qu'une des 3 lampes L1, L2, L3 indique le capteur défaillant. Les 3 capteurs sont appelés a,b,c, le relais R.

$$R = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

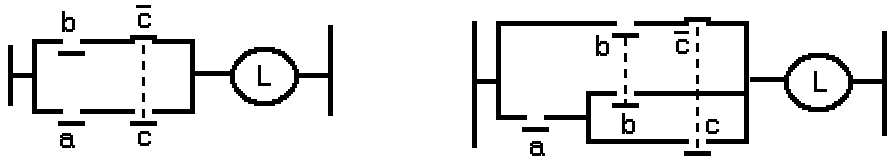
$$L1 = \bar{a}bc + a\bar{b}\bar{c} \;;\; L2 = \bar{a}bc + ab\bar{c} \;;\; L3 = \bar{a}bc + a\bar{c}b$$

3.2.6 Aléas technologiques

Réalisons la fonction L = b.(/c) + ac en faisant le schéma électrique. On peut remarquer que si a=1 et b=1, par passage par exemple de c=1 à c=0, il y a un instant où L est coupé (entre le contact c et le contact inverse). C'est l'aléa technologique. Pour y remédier, il faut faire des recouvrements dans le tableau de Karnaugh. On obtient :

$$L = \bar{b}c + ac + ab$$

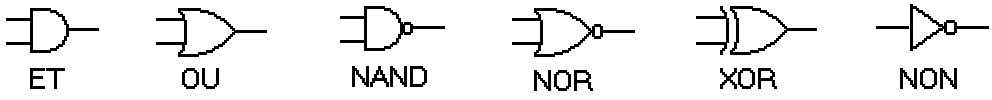
On peut vérifier sur le schéma électrique que l'aléa a disparu. En fait on a un aléa quand pour un passage d'une case valant 1 à une case adjacente valant également 1 (un capteur a changé), on traverse une discontinuité.



3.2.7 Electronique (portes)

On utilise, comme en pneumatique seconde convention, un niveau 1 (5V par ex), et un niveau 0 (0V). Les fonctions sont effectuées à l'aide de "portes", (circuits intégrés). On trouve couramment les inverseurs, NOR et NAND, mais aussi ET et OU. Les portes comportent entrée(s) et sortie, le courant ne pouvant pas "remonter" au travers d'une porte, ce qui évite le problème des anti-retour (effectués, au cas où, à l'aide d'une diode). De toute façon, le courant ne transite pas par les portes. Chacune est alimentée. Si sa sortie doit être à 1, la sortie est commutée sur l'alimentation. Donc même si le signal d'entrée n'est pas propre (mais reste dans l'intervalle de tolérance prévu, au dessus de 2,7V en TTL), le signal de sortie sera propre.

Représentation des portes : J'utiliserai principalement la norme MILSTD :



Il faut remarquer que du fait du faible coût de ces composants, il revient généralement moins cher de réaliser une partie commande électronique, et d'utiliser des interfaces appropriés pour commander les actionneurs. Ce n'est cependant pas vrai dans deux cas : si le problème est très simple (pour allumer la lumière de mon garage je mets directement les interrupteurs en parallèle, pas besoin d'une porte OU et d'un interfaçage 220V - TTL, encore moins d'un

automate programmable en Grafcet) ou en cas d'environnement incompatible (humide, parasité...).

4. Minimisation des fonctions logiques

Minimiser une fonction revient à diminuer le nombre de terme qui intervient dans sa définition et ainsi on réduit le nombre de circuits nécessaires à sa réalisation.

4.1. Méthode algébrique de simplification

On utilise les règles et les postulats de l'algèbre de boole.

Exemple :

$$F_1 = x + \bar{x}.y = \left(x + \bar{x}\right).\left(x + y\right) = 1.\left(x + y\right) = x + y$$

$$F_2 = x.\left(\bar{x} + y\right) = x.\bar{x} + x.y = 0 + x.y = x.y$$

$$F_3 = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y} = \bar{x}.z\left(y + \bar{y}\right) + x.\bar{y} = \bar{x}.z + x.\bar{y}$$

$$F_4 = x.y + \bar{x}.z + y.z = x.y + \bar{x}.z + y.z\left(x + \bar{x}\right)$$

$$= x.y + \bar{x}.z + y.z = x.y + \bar{x}.z + y.z.x + y.z.\bar{x}$$

$$= x.y\left(1 + z\right) + \bar{x}.z\left(1 + y\right) = x.y + \bar{x}.z$$

$$F_5 = \left(x + y\right).\left(\bar{x} + y\right)\left(z + y\right) = \left(x + y\right).\left(\bar{x} + z\right)$$

4.2. Méthode graphique de Karnaugh

C'est une méthode systématique et pratique lorsque le nombre de variables est inférieur à 5.

Une fois que les valeurs de la fonction logique sont introduites dans le tableau, on cherche à simplifier la fonction en regroupant les « 1 » ou les « 0 » qui se trouvent dans des carrés adjacents dans les boucles les plus larges. On peut faire des groupements de 2^n carrés adjacents avec n = 0, 1, 2, 3.....etc.

Une variable est éliminée de l'expression logique si elle se présente dans la boucle sous forma directe et inversée (en effet la variable + le complément de la variable = 1).

Les variables qui sont les mêmes pour tous les carrés de la boucle doivent figurer dans l'expression finale.

Remarque La dernière ligne est adjacente à la première et la dernière colonne est adjacente à la première.

Exemple :

AB \ C	0	1
00	1	0
01	0	0
11	0	0
10	1	0

$X(A,B,C) = \bar{B}.\bar{C}$

AB \ C	0	1
00	0	1
01	0	1
11	0	1
10	0	1

$X(A,B,C) = C$

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$X(A,B,C,D) = \bar{A}.\bar{B}.C + A.\bar{B}.\bar{D}$

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$X(A,B,C,D) = \bar{B}.\bar{D}$

AB \ CD	00	01	11	10
00	0	1	0	0
01	0	1	1	1

11	1	1	1	0
10	0	0	1	0

$$X(A,B,C,D) = B.C.D + A.C.D + A.B.C + A.C.D$$

CD / AB	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$$X(A,B,C,D) = B$$

CDE / AB	**000**	**001**	**011**	**010**	**110**	**111**	**101**	**100**
00	1	0	0	1	1	0	0	1
01	0	1	1	0	0	1	1	0
11	0	1	1	0	0	1	1	0
10	0	1	0	0	0	0	1	0

$$X(A,B,C,D,E) = B.E + A.\overline{D}.E + \overline{A}.\overline{B}.\overline{E}$$

Pour trouver l'expression en Nand (ON) d'une fonction logique on écrit son développement en somme de produits (dans le tableau de Karnaugh on fait des groupement de « 1 ») et l'on remplace les signes opératoires OU et ET par le signe opératoire Nand.

Pour trouver l'expression en NOR (NI) d'une fonction logique on fait son développement en produit de sommes (dans le tableau de Karnaugh on fait des groupements de « 0 ») et l'on remplace les signes opératoires OU et ET par le signe opératoire NOR.

Exemple : Soit $F(A,B,C,D) = \{0,\ 1,\ 2,\ 5,\ 8,\ 9,\ 10\}$

CD / AB	**00**	**01**	**11**	**10**

00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

$$F(A,B,C,D) = \bar{B}.\bar{C} + \bar{B}.\bar{D} + \bar{A}.\bar{C}.D$$
$$= \left(\bar{B}/\bar{C}\right)/\left(\bar{B}/\bar{D}\right)/\left(\bar{A}/\bar{C}/D\right)$$

De même :

$$\bar{F}(A,B,C,D) = C.D + A.B + B.\bar{D}$$
$$F(A,B,C,D) = \overline{C.D + A.B + B.\bar{D}} = \left(\bar{C}\downarrow\bar{D}\right)\downarrow\left(\bar{A}\downarrow\bar{B}\right)\downarrow\left(\bar{B}\downarrow D\right)$$

il y a des applications où certaines combinaisons des variables d'entrée ne se produisent jamais. Le code BCD par exemple possède 6 combinaisons qui ne sont pas utilisées. un circuit digital utilisant ce code fonctionne avec l'hypothèse que ces combinaisons ne se produisent jamais à condition bien sûr que le système fonctionne correctement. comme conséquence de ceci la fonction de sortie est indifférente pour ces combinaisons de variables.

Par convention le symbole de l'indifférent est : X

Ces indifférents seront utilisés pour simplifier davantage la fonction logique comme le montre l'exemple suivant :

A	**B**	**C**	**Z**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	X
1	0	0	X
1	0	1	1
1	1	0	1
1	1	1	1

AB \ **C**	**0**	**1**
00	0	0
01	0	X
11	1	1
10	X	1

$Z = A$

indifférents

4.3. Méthode tabulaire de Quine-MC Cluskey

Le nombre de variables doit être supérieur à 5 ou à 6, pratique avec l'utilisation d'un ordinateur dont il y a deux parties :

- recherche des termes candidats à être inclus dans la fonction simplifiée : ces termes sont appelés : impliquant premiers.
- choix à travers les impliquant premiers de ceux qui donnent une expressions logique avec le moindre nombre de variables.

Exemple :

Soit $F(W,X,Y,Z) = \{1,\ 4,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 15\}$

D'abord détermination des impliquant premiers

WXYZ (a)					**(b) WXYZ**		**(c) WXYZ**	
0	0	0	1	1	1,9	-001	8, 9, 10, 11	10--
0	1	0	0	4	4, 6	01-0		
1	0	0	0	8	8, 9	100-0		
0	1	1	0	6	8, 10	10-0		
1	0	0	1	9	6, 7	011-		
1	0	1	0	10	9, 11	10-1		
0	1	1	1	7	10, 11	101-		
1	0	1	1	11	7, 15	-111		
1	1	1	1	15	11, 15	1-11		

Les impliquant premiers sont les termes qui ne sont pas marqués par une trame de fond.

* -001 : $\bar{X}.\bar{Y}.Z$

* 01-0 : $\bar{W}.X.\bar{Z}$

* 011- : $\bar{W}.X.Y$

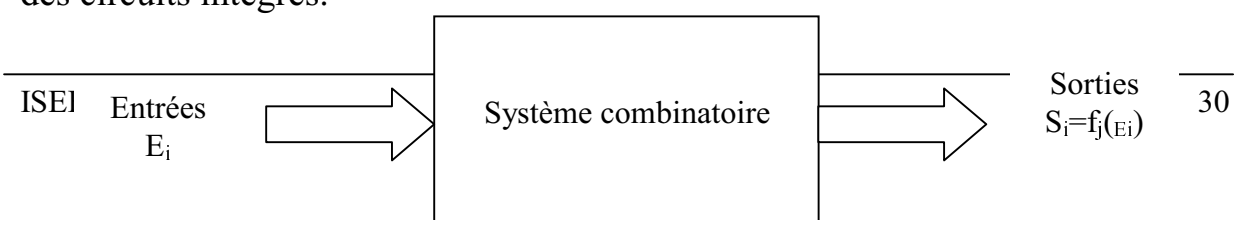
* -111 : $X.Y.Z$

* 1-11 : $W.Y.Z$

* 10-- : $W.\bar{X}$

5. Fonctions combinatoires usuelles et principaux types de circuits MSI

Les fonctions combinatoires fondamentales sont des problèmes combinatoires complexes communs à nombreux utilisateurs, réalisables dans des circuits intégrés.



5.1. Comparaison entre information

Il s'agit d'un circuit logique combinatoire qui compare deux grandeurs binaires et produit des sorties désignant l'égalité de ces grandeurs ou lequel est le plus grand ou le plus petit.

Les comparateurs de grandeurs sont employés assez intensivement dans les circuits de décodage des adresses des ordinateurs. Ce sont eux qui permettent de sélectionner le périphérique ou de localiser la zone mémoire contenant les données que l'on veut trouver. Ces éléments comparent le code d'adresse envoyé par le processeur central à un code d'adresse matériel ; si les deux coïncident, la sortie $S_{A=B}$ du comparateur active le dispositif ayant l'adresse correspondante.

Les comparateurs sont également très utiles dans les applications de régulation où un nombre binaire représentant une variable physique (comme la vitesse, la position, le courant, etc…) est comparé à une valeur de consigne. Le sorties du comparateur servent à l'envoi de signaux pour la conduite des mécanismes qui ramènent la variable physique vers son point de consigne. De même, les sorties du comparateur peuvent servir comme déclencheurs d'alarmes en cas d'anomalies dans certaines applications.

5.1.1. Principe

Soient a_i et b_i deux éléments binaires et F1, F2 et F3 les trois sorties permettant de calculer respectivement ($a_i=b_i$) ou ($a_i>b_i$) ou ($a_i<b_i$):

a_i	b_i	F1 ($a_i=b_i$)	F2 ($a_i>b_i$)	F3 ($a_i<b_i$)
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$F1=\overline{a_i \oplus b_i}=a_i \otimes b_i$: Fonction égalité

$F2=a_i\overline{b_i}$, $F_3=\overline{a_i}b_i$

5.1.2. Extension pour n éléments binaires

Soient deux nombres binaires A et B représentés par n éléments binaires :

$$A=a_{n-1}a_{n-2}\cdots a_1a_0 \quad \text{et} \quad B=b_{n-1}b_{n-2}\cdots b_1b_0$$

$F1_{(A=B)}$=1 SI $(a_0=b_0)$ ET $(a_1=b_1)$ ET ….ET $(a_{n-1}=b_{n-1})$

$$F1=\overline{(a_0\oplus b_0)}\bullet\overline{(a_1\oplus b_1)}\bullet\cdots\bullet\overline{(a_{n-1}\oplus b_{n-1})}$$

$F2_{(A>B)}$=1 SI $(a_{n-1}>b_{n-1})$ OU $(a_{n-1}=b_{n-1})$ ET $(a_{n-2}>b_{n-2})$ OU ….

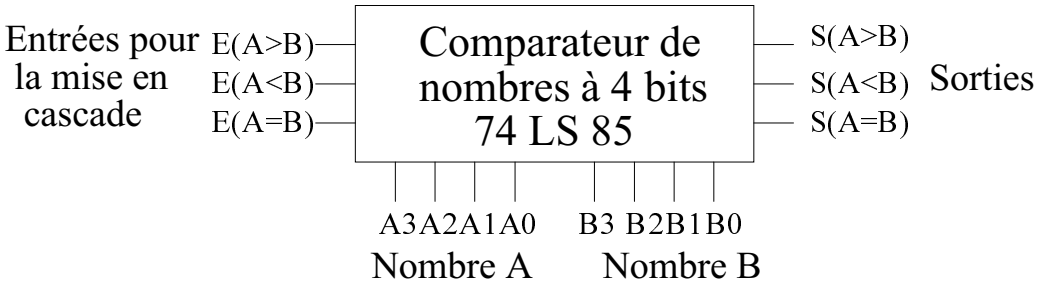
$$F2=a_{n-1}\overline{b_{n-1}}+\overline{(a_{n-1}\oplus b_{n-1})}\bullet a_{n-2}\overline{b_{n-2}}+\cdots$$

$F3_{(A<B)}$=1 SI $(a_{n-1}<b_{n-1})$ OU $(a_{n-1}=b_{n-1})$ ET $(a_{n-2}<b_{n-2})$ OU ….

$$F3=\overline{a_{n-1}}b_{n-1}+\overline{(a_{n-1}\oplus b_{n-1})}\bullet\overline{a_{n-2}}b_{n-2}+\cdots$$

5.1.3. Exemple de circuits MSI : 7485

C'est un comparateur de grandeurs à 4 éléments binaires. Les entrées de cascade permettent la comparaison de nombres de longueur quelconque par association des boîtiers entre eux. Le schéma synoptique ainsi que la table de fonctionnement de ce circuit sont données par les figures suivantes :



D'autres circuits MSI assurent la fonction de comparaison de grandeurs binaires telle que:

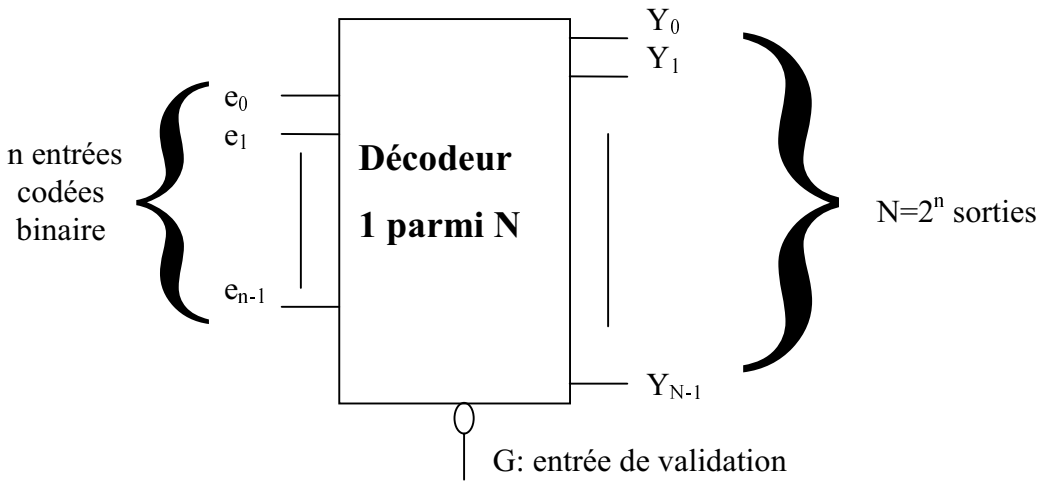
* 74 LS 682: Comparateur non cascadable de deux nombres à 8 eb.
* 74 LS 686: Comparateur cascadable de deux nombres à 8 eb.

5.2. décodage binaire

5.2.1. Principe

Une quantité codée en binaire avec n eb peut représenter $N=2^n$ valeurs. La fonction de décodage binaire consiste à transformer un codage binaire naturel (ou pur) en codage 1 parmi N (avec $N=2^n$).

Le décodeur binaire est un circuit à n entrées, $N=2^n$ sorties et éventuellement une ou plusieurs entrées G validant les fonctions de sortie.



Si G=1, toutes les sorties de décodeur sont inactives quelque soit l'état de l'information binaire sur $e_0\, e_1 \cdots e_{n-1}$.

Si G=0, pour une valeur binaire i sur les entrées $e_0\, e_1 \cdots e_{n-1}$, la sortie de rang i est active et toutes les autres sont inactives. L'équation d'une sortie de rang i s'écrit: $Y_i = \overline{G}(e_0 e_1 \cdots e_{n-1} = i)$; $i : 0 \ldots 2^{n-1}$.

5.2.2. Exemple de circuits MSI : 74 138

Le circuit 74 138 est un décodeur 3 vers 8 qui répond aux principes généraux précédents. Les figures suivantes en donnent respectivement le schéma synoptique et la table de vérité.

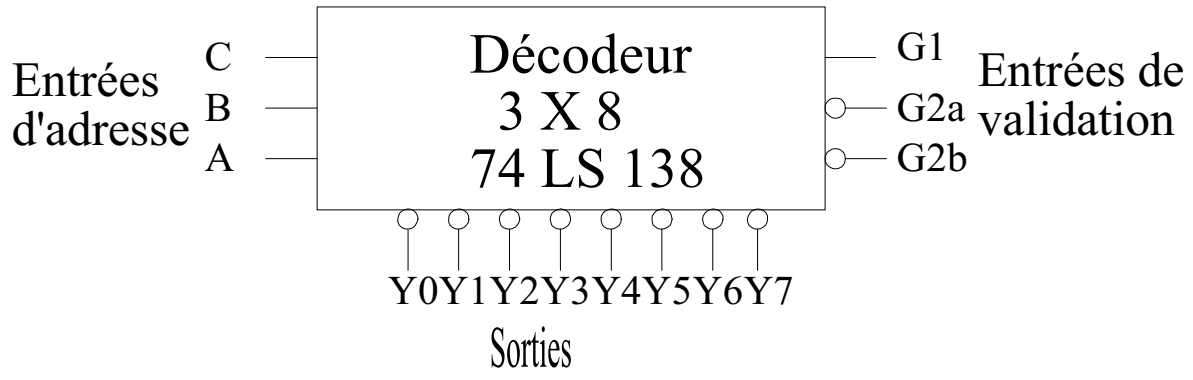


Table de vérité

Entrées		Sorties
G_1 G_{2a} G_{2b}	C B A	Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7
1 0 0	0 0 0	0 1 1 1 1 1 1 1
1 0 0	0 0 1	1 0 1 1 1 1 1 1
1 0 0	0 1 0	1 1 0 1 1 1 1 1
1 0 0	0 1 1	1 1 1 0 1 1 1 1
1 0 0	1 0 0	1 1 1 1 0 1 1 1
1 0 0	1 0 1	1 1 1 1 1 0 1 1
1 0 0	1 1 0	1 1 1 1 1 1 0 1
1 0 0	1 1 1	1 1 1 1 1 1 1 0
X 1 X	X X X	1 1 1 1 1 1 1 1
X X 1	X X X	1 1 1 1 1 1 1 1
0 X X	X X X	1 1 1 1 1 1 1 1

D'autres circuits MSI assurent la fonction de décodage binaire telle que:

* 74 LS 154: Décodeur 16 sorties avec deux entrées de validation (boîtier 24 broches)

* 74 LS 42: Décodeur 10 sorties (boîtier 16 broches).

* 74 LS 139: Double décodeur à 4 sorties avec entrées de validation (boîtier 16 broches, deux décodeurs indépendants).

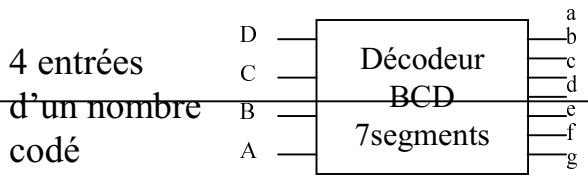
* 74 LS 155 (74 LS 156): Double décodeur à 4 sorties à adresse unique et double validation (boîtier 16 broches).

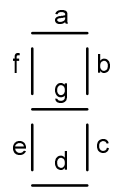
5.3. décodage BCD-7 segments

5.3.1. Principe

Dans de nombreux affichages numériques, les dix chiffres du code décimal (0 à 9), et parfois les caractères hexadécimaux (A à F), sont configurés au moyen de 7 segments. Chaque segment est constitué d'un matériau qui émet la lumière quand il est traversé par un courant. Les matériaux les plus utilisés sont les diodes électroluminescentes (LED) et les filaments incandescents.

La fonction de décodage consiste à traduire la série d'entrée de valeurs représentées par les quatre bits du code BCD(8421) en sept sorties correspondant aux valeurs des sept segments de l'afficheur. Les sorties actives permettent de faire passer un courant dans le segments qui forment le chiffre décimal correspondant au quatre bits du BCD.





Décimal	BCD (8421)				Sorties						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

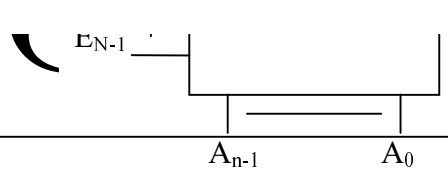
5.3.2. Exemple de circuits MSI :

Comme exemple de décodeur de BCD à sept segment, citons les circuits intégrés 7442, 7445, 7446, 7447, 7448, 7449 et 74247. Parmi les caractéristiques techniques de ces décodeurs, relevons l'affichage de signes particuliers en plus des nombres décimaux.

5.4. Multiplexage de données

5.4.1. Principe

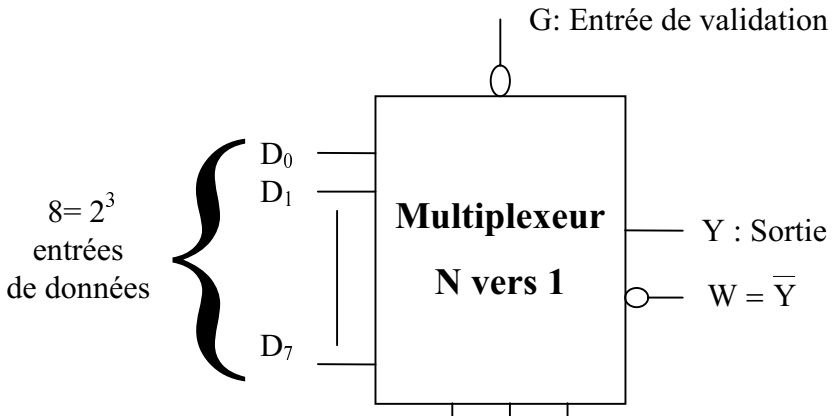
Le multiplexeur est un circuit qui permet de sélectionner une ligne d'entrée par une adresse et de faire apparaître à la sortie l'état de cette ligne, c'est à dire le niveau haut ou bas.



Entrées d'adressage

Un multiplexeur se comporte comme un commutateur dans lequel un code numérique appliqué aux entrées d'adressage commande les entrées de données qui sont raccordées à la sortie. Autrement dit, un multiplexeur choisit une source de données d'entrée parmi N et transmet celle-ci à la seule voie de sortie existante.

5.4.2. Exemple de circuits MSI :74151



Entrées			Sortie
Addresse		Validation	
C	B	A	G	Y
X	X	X	1	0
0	0	0	0	D0
0	0	1	0	D1
0	1	0	0	D2
0	1	1	0	D3
1	0	0	0	D4
1	0	1	0	D5
1	1	0	0	D6
1	1	1	0	D7

5.5. l'Unité Arithmétique et Logique

5.5.1. Principe

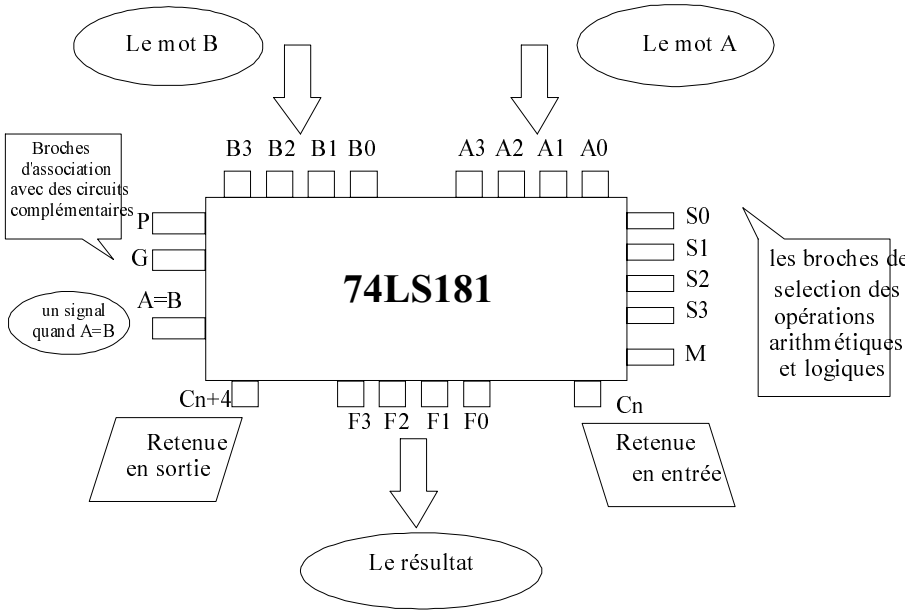
Nous nous rapprochons très sérieusement du microprocesseur avec ce composant. En effet, l'unité arithmétique et logique est un circuit qui permet de faire un certain nombre d'opérations entre deux groupes d'entrées et qui délivre un résultat sur un groupe de sortie.

5.5.2. Exemple de circuits MSI :74181

Supposons que notre UAL soit le circuit 74181: C'est une UAL à 4 bits, ce qui signifie qu'elle est capable d'effectuer des opérations sur des

valeurs binaires jusqu'à 4 bits. Chaque groupe est un mot. De même, les UAL peuvent être branchées en cascade, avec 8, 16, 32 bits ...

Vous trouvez ci-joint le schéma synoptique, ainsi que le résumé de la table de vérité de l'UAL.



La table de fonctionnement est la suivante:

Sélection					***Fonction***
S4	**S3**	**S2**	**S1**	**S0**	**F**
0	0	0	0	0	A
0	0	0	0	1	A+B
0	0	0	1	0	$A + \overline{B}$
0	0	0	1	1	0 moins 1
0	0	1	0	0	A plus $A\overline{B}$
0	0	1	0	1	(A+B) plus $A\overline{B}$
0	0	1	1	0	A moins B moins 1
0	0	1	1	1	$A\overline{B}$ moins 1
0	1	0	0	0	A plus AB
0	1	0	0	1	A plus B
0	1	0	1	0	$(A + \overline{B})$ plus AB
0	1	0	1	1	AB moins 1
0	1	1	0	0	A plus A
0	1	1	0	1	(A+B) plus A
0	1	1	1	0	$(A + \overline{B})$ plus A
0	1	1	1	1	A moins1
1	0	0	0	0	$\overline{A}$
1	0	0	0	1	$\overline{A + B}$

1	0	0	1	0	$\overline{A}B$
1	0	0	1	1	0
1	0	1	0	0	$\overline{AB}$
1	0	1	0	1	$\overline{B}$
1	0	1	1	0	$A \oplus B$
1	0	1	1	1	$A\overline{B}$
1	1	0	0	0	$\overline{A} + B$
1	1	0	0	1	$\overline{A \oplus B}$
1	1	0	1	0	B
1	1	0	1	1	AB
1	1	1	0	0	1
1	1	1	0	1	$A + \overline{B}$
1	1	1	1	0	A+B
1	1	1	1	1	A

« + » pour le OU logique, « plus » pour l'addition et « moins » pour la soustraction

5.5.3. Réalisation d'un additionneur et d'un soustracteur:

L'une des fonctions importantes de l'UAL est l'addition, pour savoir le noyau qui a fourni à la fin ce CI très important, on va essayer de déterminer un circuit simulant cette fonction arithmétique.

(a) Donner, en utilisant des portes logiques simples, le logigramme du demi additionneur sachant que ce dernier calcule la somme S de deux bits a et b et leur retenu C sans tenir compte du retenu de l'étage précèdent.

(b) Pour obtenir un additionneur complet de deux bits a_i et b_i qui calcule la somme S_i et le retenu C_i, il suffit de tenir compte du retenu de l'étage précèdent C_{i-1} comme entrée supplémentaire. Donner le logigramme correspondant.

(c) La somme de deux nombres quelconques étant basées sur les circuits précédents mis en cascade. Donner le logigramme réalisant la somme de deux nombres de deux bits chacun.

(d) Refaites les mêmes étapes pour réaliser la soustraction de deux nombres de deux bits chacun.

(e) En analysant les deux circuits (additionneur de deux nombres et soustracteur de deux nombres), on remarque que la différence est très faible (inverser ou non quelques variables), prévoir une entrée supplémentaire qui sélectionne le type de la fonction désirée (addition ou soustraction de deux

nombres de deux bits chacun) en utilisant le minimum de portes logiques. Donner un logigramme complet avec une table de vérité décrivant le fonctionnement.