

1. Introduction aux systèmes combinatoires

Comme tout système physique, les systèmes numériques peuvent être caractérisés par leur **temps de réponse** et leur **caractéristique d'entrée-sortie**. Les systèmes numériques ont besoin de **données binaires** en entrée et fournissent en sortie le même type d'informations.

Les fonctions liant l'entrée et la sortie d'un système binaire sont appelées des **fonctions logiques**. Elles sont régies par un ensemble de règles mathématiques appelées l'**algèbre de Boole**.

Dans le cas le plus simple, la notion de temps n'intervient pas et il existe une **relation directe entre l'entrée et la sortie**. On parle alors de systèmes **combinatoires**.

Les informations traitées par ces systèmes sont alors appelées des **variables logiques** ou **booléennes**. Elles ne peuvent avoir que **2 états** : "*vrai*" (ou '1' logique) ou "*faux*" (ou '0' logique). Ces niveaux logiques sont en pratique associés à des tensions.

2. Représentation des systèmes combinatoires

Il existe plusieurs façons de représenter un système combinatoire :

- par son **équation logique** ;
- par sa **table de vérité** ;
- par son **logigramme**.

L'**équation logique** permet de décrire **mathématiquement** la relation qu'il existe entre l'entrée du système et sa sortie. Chacune des sorties est une fonction des entrées. Cette représentation est basée sur l'**algèbre de Boole** (voir section suivante). Tout système combinatoire peut être réalisé à l'aide d'un petit nombre de fonctions logiques de base appelées **opérateurs logiques** ou **portes**.

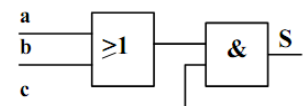
La **table de vérité** donne la liste des valeurs de sortie pour toutes les combinaisons possibles de l'entrée, classées selon l'ordre du code binaire naturel. Ainsi, la table de vérité d'une fonction de n variables a autant de ligne que d'états d'entrée, soit 2^n . Pour chacun de ces états, la sortie peut prendre la valeur "0" ou "1".

Variables d'entrée		Variable de sortie
a	b	S
0	0	1
0	1	1
1	0	1
1	1	0

Evolution des variables d'entrée (à gauche de la table) et Evolution de la variable de sortie (à droite de la table).

Le **logigramme** est un formalisme issu du **monde électronique**. Les expressions logiques sont traduites en un **câblage** reliant des symboles qui représentent les opérateurs.

Deux normes sont actuellement utilisées dans le monde, la plus ancienne est la norme ISO, toujours très utilisée dans le monde (en particulier par les logiciels de saisie de schéma). A cette norme, s'ajoute depuis 1984 la norme créée par la Commission Électrotechnique Internationale (IEC - norme IEEE 91-1984).



3. Algèbre de Boole

George BOOLE a défini, vers 1847, une algèbre qui s'applique à des fonctions logiques de variables booléennes.

Un ensemble E possède une structure d'algèbre de Boole si on a défini dans cet ensemble les éléments suivants :

- **Une relation d'équivalence** notée " = " ;
- **Deux lois de composition internes** notées " + " et " . " (addition et multiplication booléenne) ;
- **Une opération unaire** : loi qui associe à tout élément a de E son complément \bar{a} , cette loi est appelée **complémentation**.

Les lois énoncées ci-dessus s'écrivent :

+	0	1
0	0	1
1	1	1

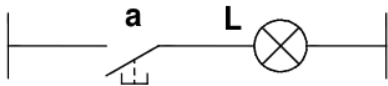
.	0	1
0	0	0
1	0	1

a	0	1
\bar{a}	1	0

3.1. Opérateurs fondamentaux

3.1.1 L'opérateur OUI

Schéma électrique :



Equation :

$$L = a$$

Table de vérité :

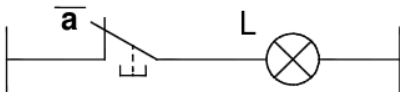
a	L
0	0
1	1

Symbole :



3.1.2 L'opérateur NON

Schéma électrique :



Equation :

$$L = \bar{a}$$

Table de vérité :

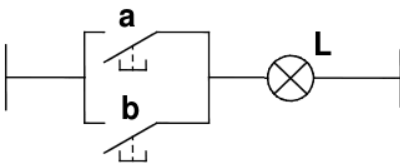
a	L
0	1
1	0

Symbole :



3.1.3 L'opérateur OU

Schéma électrique :



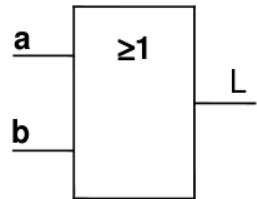
Equation :

$$L = a + b$$

Table de vérité :

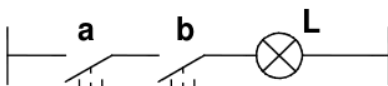
a	b	L
0	0	0
0	1	1
1	0	1
1	1	1

Symbole :



3.1.4 L'opérateur ET

Schéma électrique :



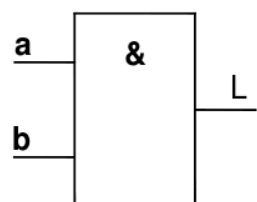
Equation :

$$L = a . b$$

Table de vérité :

a	b	L
0	0	0
0	1	0
1	0	0
1	1	1

Symbole :

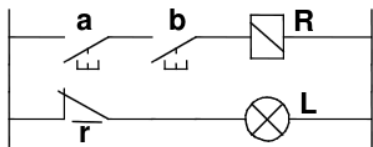


3.2. Opérateurs dérivés

A partir des opérateurs fondamentaux, on peut obtenir des opérateurs plus complexes. On parle alors d'**opérateurs dérivés**. Les 4 opérateurs suivants sont parmi les plus utilisés.

3.2.1 L'opérateur NON-ET (ou NAND)

Schéma électrique :



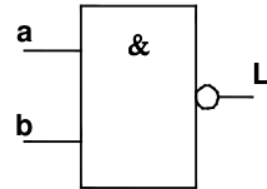
Equation :

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

Table de vérité :

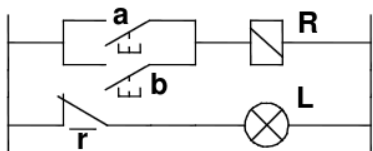
a	b	L
0	0	1
0	1	1
1	0	1
1	1	0

Symbole :



3.2.2 L'opérateur NON-OU (ou NOR)

Schéma électrique :



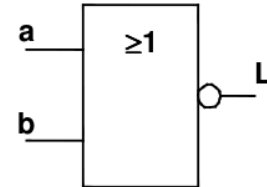
Equation :

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

Table de vérité :

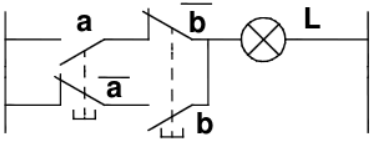
a	b	L
0	0	1
0	1	0
1	0	0
1	1	0

Symbole :



3.2.3 L'opérateur OU-EXCLUSIF (ou XOR)

Schéma électrique :



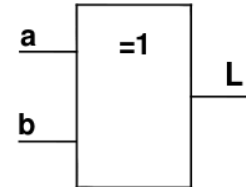
Equation :

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

Table de vérité :

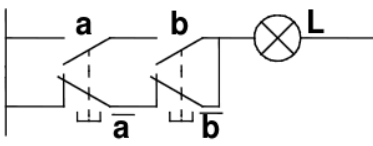
a	b	L
0	0	0
0	1	1
1	0	1
1	1	0

Symbole :



3.2.4 L'opérateur NON-OU-EXCLUSIF (ou XNOR)

Schéma électrique :



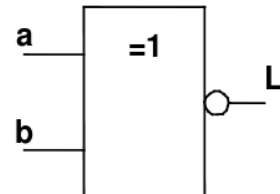
Equation :

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

Table de vérité :

a	b	L
0	0	1
0	1	0
1	0	0
1	1	1

Symbole :



3.3. Propriétés des opérations

3.3.1 Dualité des expressions

Chaque expression logique possède une **expression duale**.

L'expression duale est obtenue en échangeant les opérateurs et les valeurs logiques dans l'expression d'origine. Ainsi :

- l'opérateur + devient · ;
- l'opérateur · devient + ;
- la valeur logique "0" devient "1" ;
- la valeur logique "1" devient "0" .

Par exemple, on a : $X + 0 = X$ qui devient : $X \cdot 1 = X$

3.3.2 Propriétés des opérations

Les colonnes gauche et droite représentent les formes duales.

Commutativité	
$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
Associativité	
$(X + Y) + Z = X + (Y + Z) = X + Y + Z$	
$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$	
Distributivité	
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$ $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$	
Eléments neutres	
$X + 0 = X$	$X \cdot 1 = X$
$X + 1 = 1$	$X \cdot 0 = 0$
Complémentarité	
$X + \bar{X} = 1$	$X \cdot \bar{X} = 0$
Théorème d'idempotence	
$X + X = X$	$X \cdot X = X$

3.3.3 Théorèmes et axiomes de l'algèbre de Boole

Les colonnes gauche et droite représentent les formes duales.

Théorème d'involution	
$\overline{(\bar{X})} = X$	
Théorème de DeMorgan	
$\overline{(X + Y + Z + \dots)} = \bar{X} \cdot \bar{Y} \cdot \bar{Z} \cdot \dots$ $\overline{(X \cdot Y \cdot Z \cdot \dots)} = \bar{X} + \bar{Y} + \bar{Z} + \dots$	
$f(X_1, X_2, X_3, \dots, X_n, 0, 1, +, \cdot) = f(\bar{X}_1, \bar{X}_2, \bar{X}_3, \dots, \bar{X}_n, 1, 0, \cdot, +)$	
Théorème de multiplication et de factorisation	
$(X + Y) \cdot (\bar{X} + Z) = X \cdot Z + \bar{X} \cdot Y$	
$X \cdot Y + X \cdot \bar{Z} = (X + Z) \cdot (\bar{X} + Y)$	
Théorème du consensus	
$X \cdot Y + Y \cdot Z + \bar{X} \cdot Z = X \cdot Y + \bar{X} \cdot Z$	
$(X + Y) \cdot (Y + Z) \cdot (\bar{X} + Z) = (X + Y) \cdot (\bar{X} + Z)$	

4. Synthèse de circuits combinatoires

La **synthèse** de fonctions combinatoires consiste, à partir d'une table de vérité ou d'une expression booléenne, à spécifier les **opérateurs matériels** permettant l'implémentation de la table ou de l'expression correspondante dans un système réel. En effet, chaque **opérateur élémentaire** possède sa solution technologique sous forme de **circuit intégré** (circuits CMOS : 4001 / porte XOR, 4081 / porte ET, ...).

Dans la pratique, les systèmes combinatoires peuvent rapidement devenir complexes à mettre en oeuvre. L'équation complète régissant les sorties en fonction des entrées n'est pas nécessairement la **forme minimale** et donc pas la forme la plus économique à réaliser. Avant d'aborder l'implémentation du système sur une carte, il est donc intéressant de **simplifier l'équation** afin d'obtenir le nombre minimal d'opérateurs logiques.

Pour cela, il existe au moins trois méthodes différentes :

- une **méthode analytique**, se basant sur les théorèmes de l'algèbre de Boole ;
- une **méthode graphique**, se basant sur l'utilisation de tableau de Karnaugh ;
- une **description comportementale**², à l'aide de langage de description matérielle de haut niveau (type VHDL ou Verilog).

4.1. Méthode analytique de simplification d'équations

Le but de la simplification d'équations booléennes est de **réduire** au maximum le **nombre d'opérateurs logiques** qui interviennent dans la relation entrées-sorties du système.

Pour cela, on peut utiliser tous les **outils algébriques** qui sont mis à notre disposition dans l'algèbre de Boole (voir section précédente) ainsi que les quelques règles de simplification suivantes.

Théorèmes de simplification	
$X \cdot Y + X \cdot \bar{Y} = X$	$(X + Y) \cdot (X + \bar{Y}) = X$
$X + X \cdot Y = X$	$X \cdot (X + Y) = X$
$(X + \bar{Y}) \cdot Y = X \cdot Y$	$(X \cdot \bar{Y}) + Y = X + Y$

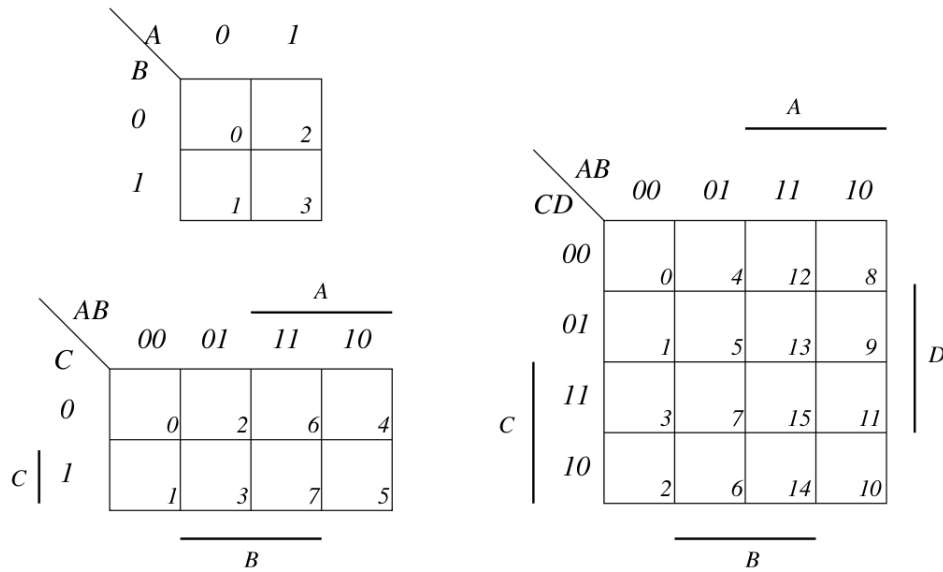
4.2. Méthode graphique de simplification d'équations (Karnaugh)

Les simplifications précédentes peuvent être réalisées graphiquement à l'aide d'un **tableau de Karnaugh**. Cette méthode se fonde sur une manière de représenter la table de vérité qui fait apparaître des **symétries sur les variables**, et en particulier sur le **théorème d'unification** : $A \cdot (B + \bar{B}) = A$

Pour rendre plus visibles les cases adjacentes (et donc les symétries), la table de vérité linéaire à numérotation binaire naturelle est remplacée par une **table rectangulaire** dont la numérotation des lignes et des colonnes suit la logique du **code Gray** (une seule variation d'une position à la position suivante).

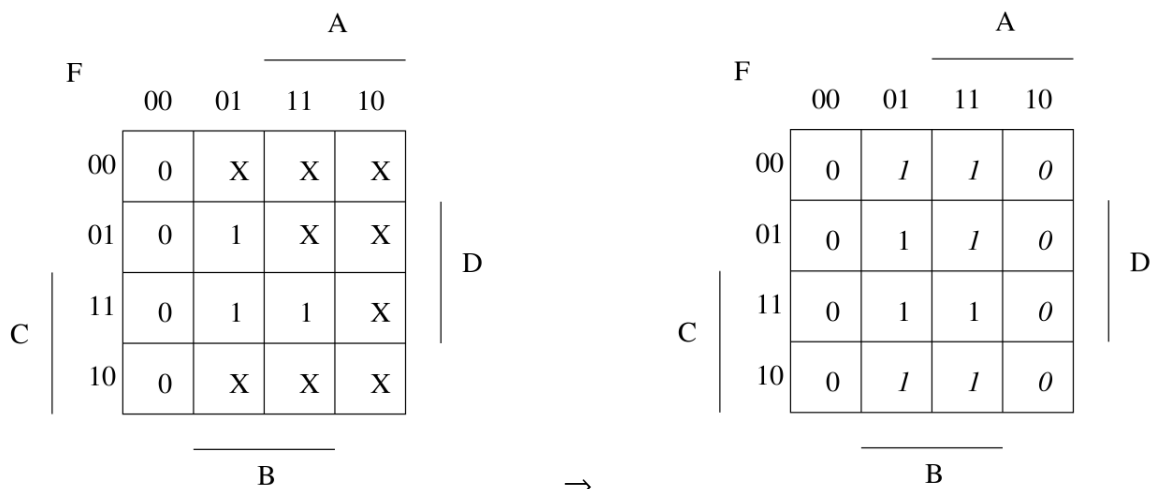
La méthode du diagramme de Karnaugh est efficace pour les expressions booléennes ayant au plus 4 entrées. Au delà, la représentation graphique devient complexe, il est difficile de mettre en évidence les symétries, et la méthode devient inutilisable. Les variables sont affectées (par couples le cas échéant) aux colonnes et lignes du tableau. La fonction est reportée dans les cases du tableau. On peut aussi reporter dans les cases les numéros d'indexation.

2. Cette dernière méthode sera étudiée plus en détail dans une autre partie du cours, ainsi qu'en travaux pratiques.



- Les règles pour la simplification des fonctions booléennes avec le tableau de Karnaugh sont les suivantes :
- tous les termes pour lesquels la fonction est à 1 devront être pris au moins une fois dans un **regroupement**, ou seuls si aucun regroupement n'est possible ;
 - les regroupements doivent être de **taille maximale**, de manière à éliminer le plus grand nombre possible de variables dans les termes de l'expression ;
 - les regroupements doivent contenir un nombre égal à une **puissance de deux** d'éléments ;
 - les recouvrements entre regroupements sont possibles ;
 - une case d'un bord est aussi **adjacente** à celle correspondante du bord opposé ;
 - un regroupement de 2 cases permet l'élimination d'une variable, un regroupement de 4 cases l'élimination de deux variables, etc...

Les problèmes réels conduisent souvent à des spécifications incomplètes des fonctions logiques. Des valeurs indéterminées (notées 'X') se présentent alors dans le tableau. On peut adopter la valeur qui conduit à la meilleure simplification.

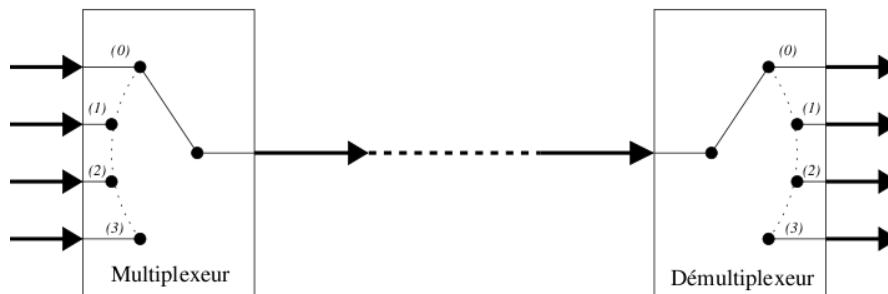


5. Fonctions combinatoires standard

Il existe des fonctions combinatoires plus complexes que les opérateurs logiques vus jusqu'à maintenant qui sont régulièrement utilisés dans les systèmes numériques.

5.1. Multiplexeurs / Démultiplexeurs

Les fonctions logiques **multiplexeur** et **démultiplexeur** sont en relation étroite avec les problèmes liés aux **transmissions d'informations multiples** par un canal de transport unique. Ce problème s'est posé dès les premières transmissions téléphoniques. Comment relier plusieurs sources à plusieurs destinataires via un seul (et coûteux) câble ?



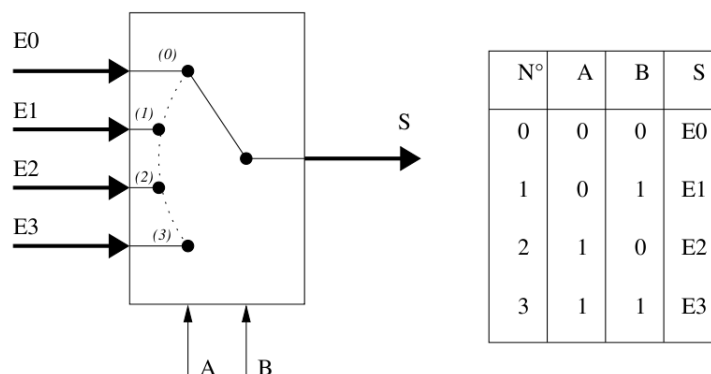
Bien entendu, il n'était pas question de transmettre les signaux des sources *simultanément* mais *successivement*. Les centraux téléphoniques sont une image de cette fonction.

Les dénominations habituelles sont :

- un multiplexeur "1 parmi N" (N entrées et 1 sortie) ;
- un démultiplexeur à N voies (1 entrée et N sorties).

5.1.1 Réalisation d'un multiplexeur pour 4 signaux binaires

La commande du sélecteur de voie (1 parmi N) est réalisée en utilisant le codage binaire du numéro de la voie choisie (de 0 à N-1). Ce code est appliqué au moyen de 2 variables logiques de sélection A et B. Le schéma et la table de vérité correspondants sont donnés ci-dessous :

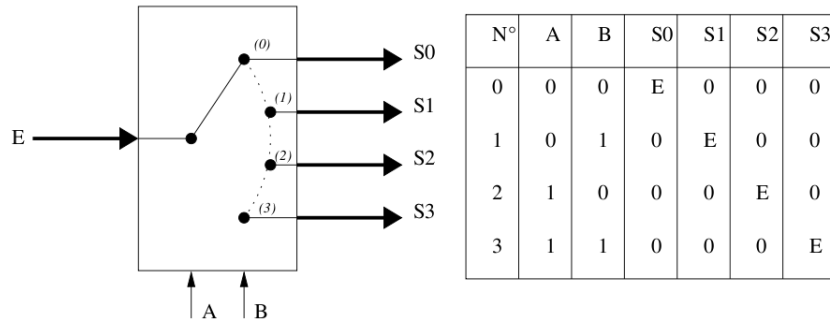


L'expression logique de la fonction S se déduit aisément :

$$S = \bar{A}.\bar{B}.E_0 + \bar{A}.B.E_1 + A.\bar{B}.E_2 + A.B.E_3$$

5.1.2 Réalisation d'un démultiplexeur pour 4 signaux binaires

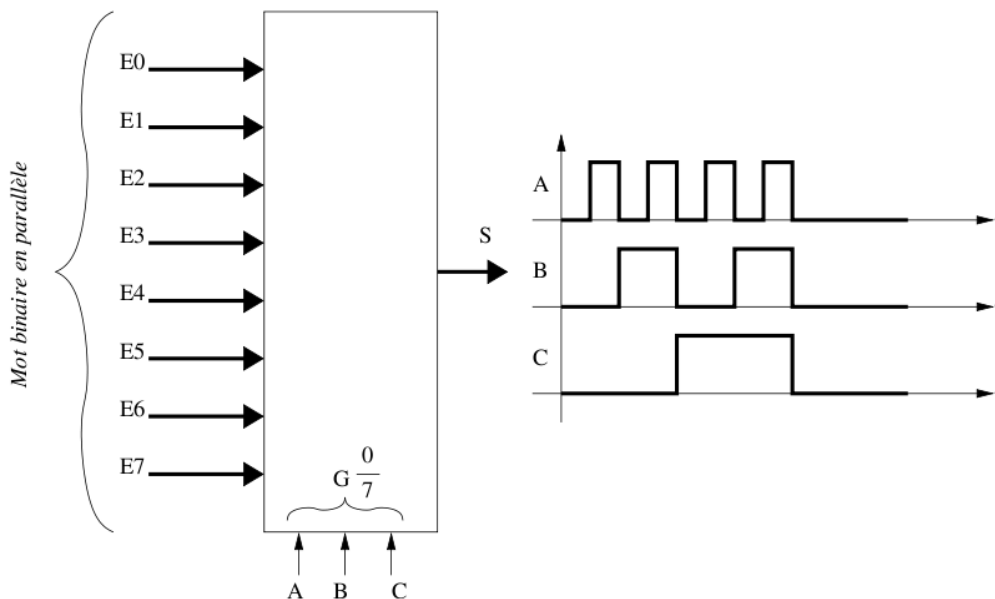
Un raisonnement similaire conduit au schéma et aux tables de vérité du démultiplexeur à 4 voies représenté ci-dessous.



Les fonctions logiques S_0, S_1, S_2, S_3 se déduisent aisément : $S_0 = \bar{A}.\bar{B}.E$, $S_1 = \bar{A}.B.E$, $S_2 = A.\bar{B}.E$, $S_3 = A.B.E$.

5.1.3 Convertisseurs parallèle-série

Associé à un compteur (autre système logique qui sera étudié par la suite dans ce module) qui génère les signaux A, B et C de la façon décrite ci-dessous sur les entrées de sélection, le multiplexeur permet de **convertir une donnée parallèle vers une liaison série**.



5.2. Codeurs et décodeurs

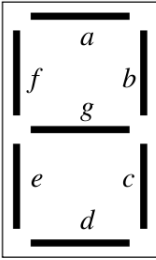
Ce sont des dispositifs qui assurent la traduction d'une représentation de l'information vers une autre. La dénomination varie selon les usages.

Codeurs

- Codeur binaire : possédant N entrées numérotées de 0 à $N - 1$, il délivre sur ses sorties le code binaire du numéro de l'entrée activée. S'il est possible que plusieurs entrées soient simultanément activées, ce sera le code de l'entrée de plus fort numéro qui sera présent (codeur de priorité).

Décodeurs

- Décodeur décimal : cette fonction transforme le code binaire naturel d'un nombre compris entre 0 et 9 en une indication décimale sous la forme de l'activation d'une des 10 lignes de sortie : le code binaire 0000 active la sortie S_0 , le code 0001 la sortie S_1 , etc.. jusqu'au code 1001 qui active la sortie S_9 .
- Décodeur "7 segments" : il fonctionne à la manière du précédent, mais au lieu de fournir un code de sortie "un parmi dix", il produit une combinaison de signaux aptes à allumer les segments d'un afficheur à 7 segments pour former le chiffre correspondant.

				d3	d2	d1	d0	a	b	c	d	e	f	g
	0	0	0	0	1	1	1	1	1	1	1	0		
	0	0	0	1	0	1	1	0	0	0	0	0		
	0	0	1	0	1	1	0	1	1	0	1			
	0	0	1	1	1	1	1	1	0	0	0			
	0	1	0	0	0	1	1	0	0	1	1			
	0	1	0	1	1	0	1	1	0	1	1			
	0	1	1	0	0	0	1	1	1	1	1			
	0	1	1	1	1	1	1	0	0	0	0			
	1	0	0	0	1	1	1	1	1	1	1			
	1	0	0	1	1	1	1	0	0	1	1			

Transcodeurs

- Transcodeur Gray vers binaire : à partir d'un code de Gray à N bits, il fournit le code binaire correspondant.

5.3. Fonctions arithmétiques

La plupart des **fonctions arithmétiques** sont indispensables dans les circuits numériques : addition de variables, multiplication de signaux... Les **microcontrôleurs**³, par exemple, possèdent une **unité de calculs arithmétiques et logiques** précablée.

Tous ces calculs arithmétiques peuvent se faire à l'aide de circuits de la logique combinatoire.

5.3.1 Comparateur de 2 nombres entiers positifs

Soient 2 nombres entiers A et B codés sur n bits, le problème posé est de disposer de signaux logiques $S_=$, $S_<$ et $S_>$ indiquant respectivement si $A = B$, $A < B$ et $A > B$.

Sur des nombres codés sur 2 bits par exemple, on obtient la table de vérité suivante :

A		B		$S_=$	$S_<$	$S_>$	A		B		$S_=$	$S_<$	$S_>$
0	0	0	0	1	0	0	1	0	0	0	0	0	1
0	0	0	1	0	1	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	1	0	1	0	1	0	0
0	0	1	1	0	1	0	1	0	1	1	0	1	0
0	1	0	0	0	0	1	1	1	0	0	0	0	1
0	1	0	1	1	0	0	1	1	0	1	0	0	1
0	1	1	0	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	1	0	1	1	1	1	1	0	0

3. Ces composants seront étudiés plus tard dans l'année.

5.3.2 Additionneur à un bit (demi-additionneur)

La somme de deux nombres codés en binaire naturel sur un bit nécessite une représentation sur 2 bits (un bit de somme et un bit de retenue). Soient a et b les 2 nombres, s le bit somme et co le bit de retenue. Leurs tables de vérité respectives sont indiquées ci-dessous :

a	b	s	co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ce circuit est souvent appelé **demi-additionneur** : il ne peut être utilisée dans une addition à plus d'un bit, car il ne dispose pas d'une entrée permettant de prendre en compte une retenue provenant d'une addition de rang inférieur.

5.3.3 Additionneur à plusieurs bits, additionneur complet

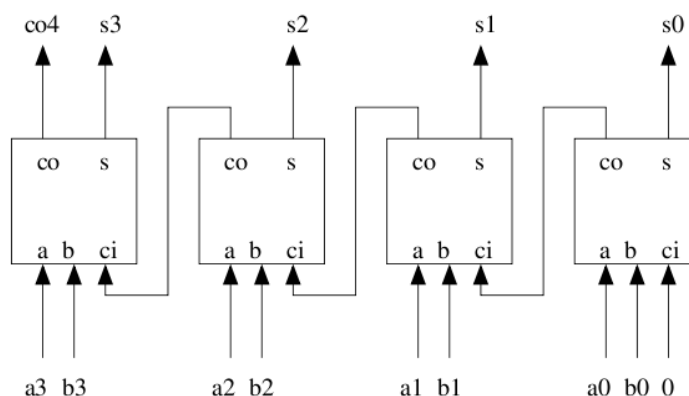
L'addition à plusieurs bits se réalise à la main avec le même algorithme que l'addition décimale :

$$\begin{array}{r}
 \text{retenues} \quad 1 \quad 1 \quad 1 \quad 1 \\
 \quad \quad \quad 1 \quad 1 \quad 0 \quad 1 \\
 + \quad \quad \quad 1 \quad 0 \quad 1 \quad 1 \\
 \hline
 \quad \quad \quad 1 \quad 1 \quad 0 \quad 0 \quad 0
 \end{array}$$

On s'aperçoit que l'addition doit traiter, en général, 3 opérandes : a , b et la retenue ci . Il doit toujours produire 2 résultats s et co . On obtient alors la table de vérité de l'additionneur complet :

a	b	ci	s	co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Pour réaliser un **additionneur à plusieurs bits**, il suffit alors de mettre en cascade plusieurs **additionneurs complets**.



5.3.4 Multiplication de 2 nombres entiers positifs

L'algorithme de multiplication n'est pas lié à la base de numération, nous pouvons donc procéder de la même manière qu'en base 10 : additions répétées et décalages à gauche.

La multiplication de 2 nombres à plusieurs bits s'effectue en calculant les produits partiels décalés selon le rang du bit du multiplicateur, puis en ajoutant ces produits partiels.

$$\begin{array}{r} 1\ 0\ 1\ 1 \\ \times 1\ 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 1 \\ 0\ 0\ 0\ 0 \\ 1\ 0\ 1\ 1 \\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \end{array}$$

Annexe 1 : Formes canoniques des expressions logiques

Afin de comparer des expressions booléennes exprimées sous forme algébrique, il est utile de disposer d'une forme standard qui représente la fonction. Une telle forme standard est appelée *forme canonique*. Il existe deux formes canoniques couramment employées : *somme de produits* et *produit de sommes*.

Pour constituer de telles formes, on part de la table de vérité d'une fonction logique. Nous considérerons la fonction F ci-dessous, ainsi que sa forme complémentée \overline{F} .

A	B	C	F	\overline{F}
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

8.1. Somme de produits

Cette forme est aussi appelée *développement en «minterms»* ou *forme disjonctive normale*. Pour obtenir cette forme, on considère chaque «1» de la table de vérité de F. On peut alors lire la table de vérité ainsi :

$F=1$ si (A=0 et B=1 et C=0) ou (A=1 et B=0 et C=0) ou (A=1 et B=0 et C=1) ou (A=1 et B=1 et C=0) ou (A=1 et B=1 et C=1).

Cette équivalence peut se formuler autrement :

$$F = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

et de même :

$$\overline{F} = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}$$

Ce sont les expressions en *somme de produits* de F et de \overline{F} .

Mais, en utilisant les propriétés de la numération binaire pour identifier les lignes, on peut s'abstenir de développer complètement les produits, mais on peut indiquer simplement le nombre binaire constitué par la combinaison de A, B et C pris dans cet ordre précédé de la lettre m faisant référence à *minterm*. Par exemple, pour F, les minterms seront m_3, m_4, m_5, m_6, m_7 . Nous écrirons alors :

$$F(A, B, C) = m_3 + m_4 + m_5 + m_6 + m_7 = \sum m(3, 4, 5, 6, 7)$$

Nous aurons aussi :

$$\overline{F}(A, B, C) = \sum m(0, 1, 2) = m_0 + m_1 + m_2$$

Il est à remarquer que le développement en minterms ne garantit pas que la forme obtenue est minimale. On peut d'ailleurs le vérifier car F peut se mettre sous la forme : $F = B \cdot C + A$.

8.2. Produit de sommes

La forme en produit de sommes est parfois appelée *forme conjonctive normale* ou bien *développement en maxterms*.

En appliquant le théorème de DeMorgan à l'expression en somme de produits de \bar{F} rappelée ci-dessous :

$$\bar{F} = \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C}$$

On obtient :

$$F = (A + B + C) \cdot (A + B + \bar{C}) \cdot (A + \bar{B} + C)$$

En utilisant cette fois la numération binaire complétée pour identifier les lignes correspondantes de la table de vérité, et le symbole M faisant référence à *Maxterm* :

$$F(A, B, C) = \prod M(0, 1, 2) = M_0 \cdot M_1 \cdot M_2$$

8.3. Passage d'une forme canonique à une autre

Pour réaliser rapidement ce passage, il suffit de remarquer que les minterms et les Maxterms sont mutuellement exclusifs dans les développements : si un minterm m_i est présent dans le développement en minterms, le Maxterm de même indice M_i sera absent du développement en Maxterms. Donc, pour passer d'un développement à l'autre, on doit rechercher tous les termes absents du développement d'origine, les changer de nom, puis transformer les sommes en produits et vice versa.

Par exemple :

$$F(A, B, C) = \prod M(0, 2, 3) = M_0 \cdot M_2 \cdot M_3 \Leftrightarrow F(A, B, C) = \sum m(1, 4, 5, 6, 7) = m_1 + m_4 + m_5 + m_6 + m_7$$

8.4. Fonctions à spécification incomplète

Il arrive fréquemment dans les applications réelles que pour certaines combinaisons de variables (ou assertions), la fonction logique étudiée ne soit pas spécifiée (ni vraie, ni fausse). Dans ce cas, le concepteur a généralement le choix pour fixer le niveau logique à une valeur arbitraire, très souvent en vue de simplifier sa conception. Mais en tout état de cause, on doit compléter les expressions en minterms ou en Maxterms par des *indifférents* (don't care) notés d_j dans le développement en minterms et D_j dans le développement en Maxterms. A noter que contrairement aux min- et Max- terms, les indifférents sont communs aux deux développements.

Par exemple :

$$F(A, B, C) = \prod M(0, 2, 3)D(4, 7) = M_0 \cdot M_2 \cdot M_3 \cdot D_4 \cdot D_7$$

$$\Leftrightarrow F(A, B, C) = \sum m(1, 5, 6) + d(4, 7) = m_1 + m_5 + m_6 + d_7$$

Annexe 2 : Symboles normalisés des Portes Logiques