Chapitre

3

# Circuits logiques combinatoires

Le chapitre précédent était une introduction à l'algèbre de Boole. Les ingénieurs s'arrêtent rarement aux considérations purement mathématiques de ces travaux. Ce qui les intéresse au premier chef est de tirer parti de la théorie pour élaborer des méthodes de conception de circuits logiques, et que ces méthodes soient pratiques et performantes. Nous sommes donc plus concernés par les travaux de Shannon que de ceux de Boole à proprement parler. Aussi, il importe de présenter les techniques et outils développés dans cette perspective pour manipuler les circuits logiques. Nous nous attarderons dans le présent chapitre à définir le vocabulaire des circuits combinatoires et les concepts jugés importants pour les développements à suivre. Nous aurons ainsi établi le corpus théorique à même de nous pourvoir des techniques d'optimisation des circuits logiques que nous aborderons au chapitre suivant.

## 3.1 Vocabulaire des circuits logiques combinatoires

Les circuits logiques combinatoires (CLC) sont la première forme de circuits que nous aurons à étudier dans ce cours. Les circuits logiques combinatoires sont des circuits caractérisés par leur propriété déterministe, associant à toute combinaison d'entrées une seule et même combinaison de sorties. Pour cette raison, les circuits combinatoires sont souvent vus comme des *boîtes noires* régies par ce fonctionnement déterministe entrées/sorties et dont *l'implémentation* sera pour nous sujette à discussion.

### 3.1.1 Circuits MISO et MIMO

Les circuits logiques que nous avons vus jusqu'ici étaient l'équivalent d'une fonction logique, et de ce fait ne possédaient qu'une seule sortie. Il est plus courant cependant de rencontrer des circuits possédants plusieurs sorties. Nous utiliserons l'appellation MISO et MIMO pour désigner respectivement les circuits combinatoires du type *entrées multiples, sortie unique* et *entrées multiples, sorties multiples* ; MISO et MIMO étant respectivement les acronymes anglais de *Multiple Inputs, Single Output* et *Multiple Inputs, Multiple Outputs*.

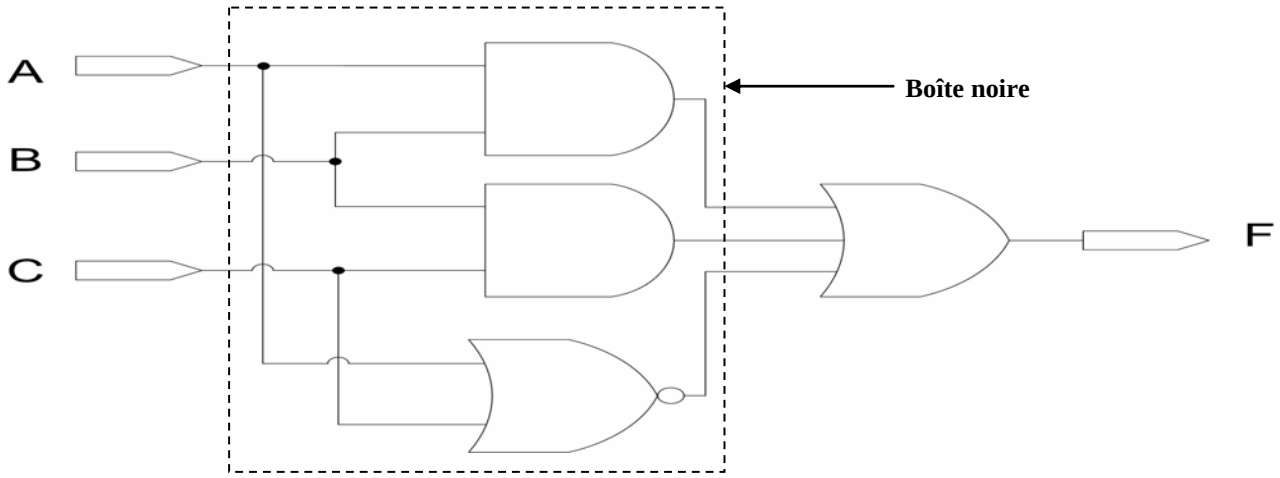
Dans la plupart du temps, nous manipulerons des circuits de type MISO, respectant ainsi l'association faite entre des entrées et une sortie telle que décrite par les tables de vérité vues jusqu'ici. Les MIMO prendront de l'importance pour nous lorsque nous aurons à considérer des circuits plus évolués, remplissant des fonctions utiles aux systèmes logiques.

Les circuits MIMO sont autant de MISO que de sorties qu'ils comportent. Les circuits MIMO offrent un avantage important car ils permettent une économie de portes logiques lorsque nombre des circuits MISO le composant partagent des entrées. Cette question se posera au chapitre suivant lorsque nous nous intéresserons à l'optimisation des circuits combinatoires.

### 3.1.2 La boîte noire et son implémentation.

Il est d'usage de faire la distinction entre le comportement d'un circuit logique et sa réalisation. Le comportement d'un circuit combinatoire est régi par la table de vérité ou (de manière équivalente) par la fonction logique qui le décrit. Ce comportement peut se résumer au lien unissant les combinaisons d'entrées aux combinaisons de sorties, et cela sans qu'on ait à se soucier de ce qui compose le circuit de l'intérieur. On parle alors de boîte noire pour séparer conceptuellement l'extérieur (entrées/sorties) du circuit de son intérieur (les portes logiques le composant).

Prenons l'exemple de circuit MISO suivant :




Ce circuit combine les entrées A, B et C pour former la sortie F. Un carré a été ajouté autour du circuit de façon à illustrer le concept de boîte noire. La fonction logique réalisée par le circuit à l'intérieur de la boîte noire est donné par la fonction F :

$$F = AB+BC+\overline{A+C}$$

Pour toute combinaison des entrées A, B et C, il existe une seule valeur de sortie possible F. Ces valeurs de sortie sont décrites par la table de vérité de la fonction F.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

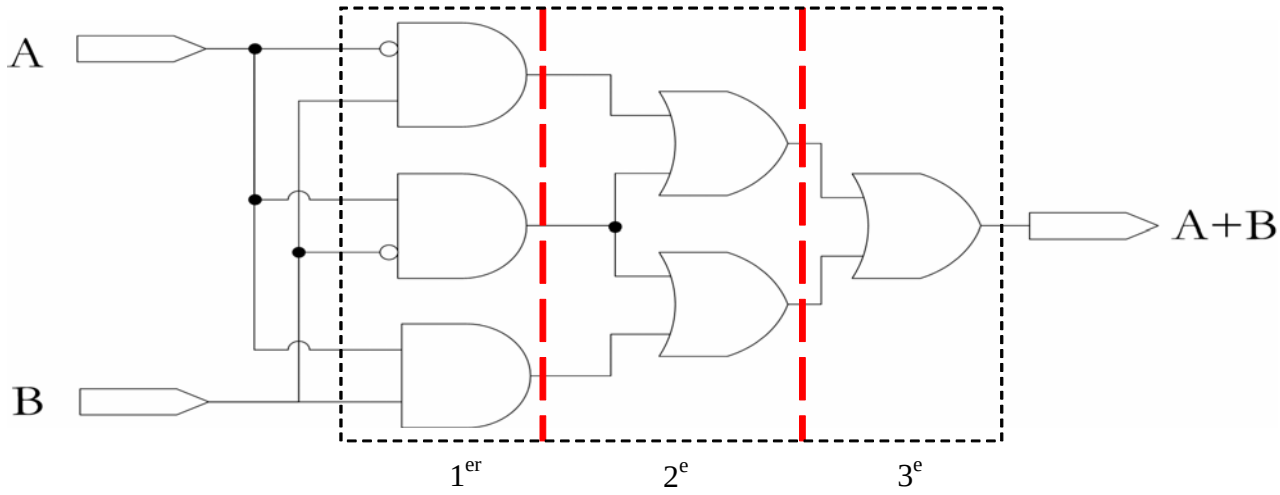
On notera que cette table de vérité est aussi celle des fonctions $\overline{A}\ \overline{C}+B$ et $\overline{A+C}+B$.

Aussi, l'*intérieur* de la boîte noire peut être réalisé de différentes manières et il n'existe pas a priori de réalisation unique. La réalisation de la table de vérité à l'intérieur de la boîte noire sera appelée l'*implémentation* de la fonction logique. Une fonction logique peut être *implémentée* de différentes manières et seul compte son comportement vu de l'extérieur. Aussi, le concepteur à libre jeu de faire preuve de virtuosité pour *simplifier* le circuit et en réduire le *coût*.

### 3.1.3 Circuits à multiples niveaux

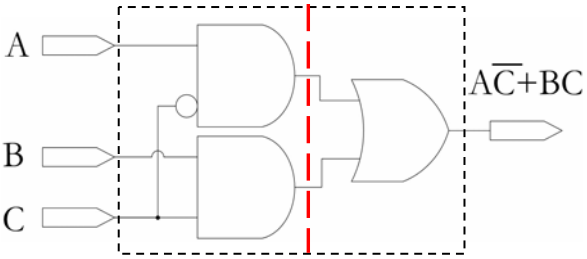
Les circuits logiques combinatoires sont construits en mettant en cascade plusieurs portes logiques. Il est important, d'un point de vue d'analyse de circuit, de segmenter cette hiérarchie de portes de sorte à bien comprendre la progression des signaux logiques de l'entrée jusqu'aux sorties.

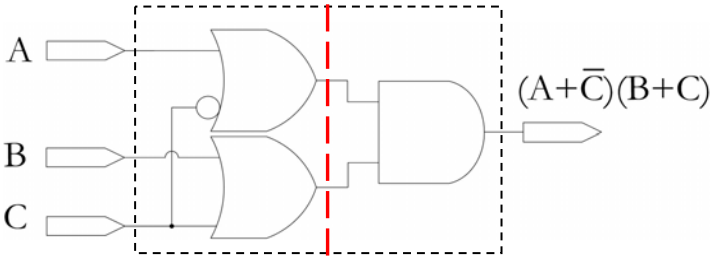
La délimitation des niveaux s'effectue en commençant le compte depuis la sortie, et en évoluant progressivement vers les entrées. Le compte est incrémenté à chaque porte rencontrée. L'exemple de circuit MISO qui suit présente un circuit à trois niveaux, implémentant la fonction logique A+B :




Les circuits logiques combinatoires à deux niveaux sont ceux que l'on rencontre le plus fréquemment. La raison est qu'ils découlent d'un formalisme — dit de Somme de produits (SOP[1]) et de produit de sommes (POS[2]) — que nous allons exposer plus bas. Par convention, les inverseurs aux entrées ne rentrent pas en ligne de compte dans le dénombrement des niveaux.

Voici quelques exemples d'une somme de produits et d'un produit de sommes :




## 3.2 Outils théoriques pour les CLC

Toutes les notions présentées jusqu'ici tiennent davantage du vocabulaire que de la science. Ce lexique doit néanmoins être assimilé et devenir familier au futur ingénieur qui aura à en faire usage à plus d'une reprise.

[1] Acronyme anglais de *Sum Of Products*.
[2] Acronyme anglais de *Product Of Sums*.

La section qui suit comporte un contenu théorique d'importance. Ce contenu nous servira autant dans ce chapitre que dans les suivants et doit être compris en profondeur. La notation dont nous ferons usage utilise la représentation binaire des nombres qui fut introduite au chapitre 1. Le lecteur est donc invité à s'y référer au besoin.

### 3.2.1 Produit de somme et somme de produits

Les tables de vérité ont été introduites dans le chapitre précédent de manière relativement succincte. On tâche généralement, en présentant l'ensemble des combinaisons des entrées, à les arranger par ordre croissant selon la représentation binaire des entiers telle que présentée à la section 1.4.3 et illustrée au tableau suivant :

Table 3.1 Table de vérité générique pour une fonction logique à trois variables $f(x_1, x_2, x_3)$

| Ligne | $x_1$ | $x_2$ | $x_3$ | $f$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $f(0,0,0)$ |
| 1 | 0 | 0 | 1 | $f(0,0,1)$ |
| 2 | 0 | 1 | 0 | $f(0,1,0)$ |
| 3 | 0 | 1 | 1 | $f(0,1,1)$ |
| 4 | 1 | 0 | 0 | $f(1,0,0)$ |
| 5 | 1 | 0 | 1 | $f(1,0,1)$ |
| 6 | 1 | 1 | 0 | $f(1,1,0)$ |
| 7 | 1 | 1 | 1 | $f(1,1,1)$ |

Cette représentation nous permet de numéroter les lignes de 0 à 7 pour une fonction de 3 variables, et plus généralement de 0 à $2^n$-1 pour une fonction à $n$ variables. La numérotation prendra toute son importance dans l'écriture des expressions canoniques qui suivent.

Reprenons la décomposition de Shannon présentée à la section 2.1.5. Soit $f$ une fonction logique de $n$ variables $x_1, x_2, \ldots x_n$. On note alors :

$$f(x_1, x_2, \ldots, x_n) = \overline{x_1}\, f(0, x_2, \ldots, x_n) + x_1 f(1, x_2, \ldots, x_n)$$

En appliquant la décomposition de manière récursive, c'est-à-dire en considérant successivement l'ensemble des variables $x_i$ de $i=1$ *à* $i=n$, on trouve progressivement :

$$f(x_1, x_2, \ldots, x_n) = \overline{x_1}\,(\,\overline{x_2}\, f(0, 0, x_3, \ldots, x_n) + x_2 f(0, 1, x_3, \ldots, x_n)\,)$$

$$+ x_1\,(\,\overline{x_2}\, f(1, 0, x_3, \ldots, x_n) + x_2 f(1, 1, x_3, \ldots, x_n)\,)$$

$$f(x_1, x_2, \ldots, x_n) = \overline{x_1}\,\overline{x_2}\,(\,\overline{x_3}\, f(0, 0, 0, x_4, \ldots, x_n) + x_3 f(0, 0, 1, x_4, \ldots, x_n)\,)$$

$$+ \overline{x_1}\, x_2\,(\,\overline{x_3}\, f(0, 1, 0, x_4, \ldots, x_n) + x_3 f(0, 1, 1, x_4, \ldots, x_n)\,)$$

$$+ x_1\,\overline{x_2}\,(\,\overline{x_3}\, f(1, 0, 0, x_4, \ldots, x_n) + x_3 f(1, 0, 1, x_4, \ldots, x_n)\,)$$

$$+ x_1\, x_2\,(\,\overline{x_3}\, f(1, 1, 0, x_4, \ldots, x_n) + x_3 f(1, 1, 1, x_4, \ldots, x_n)\,)$$

En continuant le processus jusqu'à $x_n$, on aboutit à une écriture unique de la fonction logique $f(x_1, x_2, \ldots, x_n)$. Cette expression est dite canonique disjonctive (en référence à la disjonction des termes opérée par

l'opérateur logique (OU,+)). On l'appelle également forme canonique de somme de produits.

. Cette écriture est dite canonique car elle est unique pour chaque fonction. Une expression canonique n'est cependant pas *optimale* ; nous verrons au chapitre suivant qu'elle sert de base aux méthodes de simplification.

On identifie, selon le principe de dualité propre à l'algèbre de Boole, deux formes canoniques : la somme de produits (présentée plus haut) et le produit de sommes (appelé forme canonique conjonctive).

. Le produit de somme canonique peut être obtenu en appliquant la forme duale de la décomposition de Shannon de manière duale à celle présentée plus haut.

Afin d'illustrer l'écriture canonique, considérons l'exemple d'une fonction à trois variables $f(x_1, x_2, x_3)$. Suivant la décomposition récursive de Shannon, nous avons la forme canonique disjonctive suivante :

$$
\begin{aligned}
f(x_1, x_2, \ldots, x_n) = {} & \overline{x_1}\,\overline{x_2}\,\overline{x_3} f(0,0,0) \\
& + \overline{x_1}\,\overline{x_2}\,x_3 f(0,0,1) \\
& + \overline{x_1}\,x_2\,\overline{x_3} f(0,1,0) \\
& + \overline{x_1}\,x_2 x_3 f(0,1,1) \\
& + x_1\overline{x_2}\,\overline{x_3} f(1,0,0) \\
& + x_1\overline{x_2}\,x_3 f(1,0,1) \\
& + x_1 x_2\,\overline{x_3} f(1,1,0) \\
& + x_1 x_2 x_3 f(1,1,1)
\end{aligned}
\qquad
\begin{aligned}
f(x_1, x_2, \ldots, x_n) = {} & m_0 f(0,0,0) \\
& + m_1 f(0,0,1) \\
& + m_2 f(0,1,0) \\
& + m_3 f(0,1,1) \\
& + m_4 f(1,0,0) \\
& + m_5 f(1,0,1) \\
& + m_6 f(1,1,0) \\
& + m_7 f(1,1,1)
\end{aligned}
$$

Les termes $m_i$ sont définis par identification terme à terme des deux expressions et sont appelés *minterms* (vocable anglais pour *termes mineurs*). Il est intéressant de noter que l'indice des $m_i$ est choisi de sorte à renvoyer au numéro de ligne indiqué à la table 3.1. De la même façon, il est possible d'exprimer $f(x_1, x_2, x_3)$ sous la forme canonique conjonctive, c'est-à-dire sous la forme d'un produit de sommes :

$$
\begin{aligned}
f(x_1, x_2, \ldots, x_n) = {} & (x_1 + x_2 + x_3 + f(0,0,0)) \\
& (x_1 + x_2 + \overline{x_3} + f(0,0,1)) \\
& (x_1 + \overline{x_2} + x_3 + f(0,1,0)) \\
& (x_1 + \overline{x_2} + \overline{x_3} + f(0,1,1)) \\
& (\overline{x_1} + x_2 + x_3 + f(1,0,0)) \\
& (\overline{x_1} + x_2 + \overline{x_3} + f(1,0,1)) \\
& (\overline{x_1} + \overline{x_2} + x_3 + f(1,1,0)) \\
& (\overline{x_1} + \overline{x_2} + \overline{x_3} + f(1,1,1))
\end{aligned}
\qquad
\begin{aligned}
f(x_1, x_2, \ldots, x_n) = {} & (M_0 + f(0,0,0)) \\
& (M_1 + f(0,0,1)) \\
& (M_2 + f(0,1,0)) \\
& (M_3 + f(0,1,1)) \\
& (M_4 + f(1,0,0)) \\
& (M_5 + f(1,0,1)) \\
& (M_6 + f(1,1,0)) \\
& (M_7 + f(1,1,1))
\end{aligned}
$$

On définit de la sorte les termes $M_i$, appelés *maxterms* suivant l'usage du vocable anglais pour *termes majeurs*. Il est important de noter que, là encore, l'indice des $M_i$ renvoie au numéro de ligne indiqué à la table 3.1.

De la première écriture (SOP), nous relevons que toute fonction peut être écrite sous la forme d'une somme des *minterms* $m_i$ pour lesquels $f$ vaut 1. De la même manière, nous relevons que l'écriture POS permet d'écrire toute fonction logique comme le produit des *maxterms* $M_i$ pour lesquels $f$ vaut 0. Nous exploiterons cette assertion à la section 3.2.3.

### 3.2.2 Écriture condensée des SOP et POS

La table 3.2 résume les *minterms* et *maxterms* pour une fonction logique à trois variables.

Table 3.2. Table de vérité générique pour une fonction logique à trois variables avec mise en évidence des *minterms* et *maxterms*

| Ligne | $x_1$ | $x_2$ | $x_3$ | $f(x_1,x_2,x_3)$ | *Minterm* ($m_i$) | *Maxterm* ($M_i$) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $f(0,0,0)$ | $\overline{x_1}\ \overline{x_2}\ \overline{x_3}$ | $x_1 + x_2 + x_3$ |
| 1 | 0 | 0 | 1 | $f(0,0,1)$ | $\overline{x_1}\ \overline{x_2}\ x_3$ | $x_1 + x_2 + \overline{x_3}$ |
| 2 | 0 | 1 | 0 | $f(0,1,0)$ | $\overline{x_1}\ x_2\ \overline{x_3}$ | $x_1 + \overline{x_2} + x_3$ |
| 3 | 0 | 1 | 1 | $f(0,1,1)$ | $\overline{x_1}\ x_2\ x_3$ | $x_1 + \overline{x_2} + \overline{x_3}$ |
| 4 | 1 | 0 | 0 | $f(1,0,0)$ | $x_1\ \overline{x_2}\ \overline{x_3}$ | $\overline{x_1} + x_2 + x_3$ |
| 5 | 1 | 0 | 1 | $f(1,0,1)$ | $x_1\ \overline{x_2}\ x_3$ | $\overline{x_1} + x_2 + \overline{x_3}$ |
| 6 | 1 | 1 | 0 | $f(1,1,0)$ | $x_1\ x_2\ \overline{x_3}$ | $\overline{x_1} + \overline{x_2} + x_3$ |
| 7 | 1 | 1 | 1 | $f(1,1,1)$ | $x_1\ x_2\ x_3$ | $\overline{x_1} + \overline{x_2} + \overline{x_3}$ |

Puisque nous avons numéroté les lignes de la table de vérité, il devient aisé d'écrire une fonction logique sous une forme canonique réduite n'utilisant que les *minterms* ou les *maxterms*, selon que l'on considère le produit de sommes (POS) ou la somme de produits (SOP).

Pour ce faire, on choisit de la table de vérité les termes (respectivement) majeurs ou mineurs où la fonction vaut (respectivement) 1 ou 0, tel que discuté à la section 3.2.2.

Nous allons illustrer cette méthode par un exemple.

Soit $f_1(x_1, x_2, x_3) = \overline{x_1}\ x_2 + x_1 x_3$.

La table de vérité de cette fonction est :

| Ligne | $x_1$ | $x_2$ | $x_3$ | $f(x_1,x_2,x_3)$ | *Minterm* ($m_i$) | *Maxterm* ($M_i$) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | $\overline{x_1}\ \overline{x_2}\ \overline{x_3}$ | |
| 1 | 0 | 0 | 1 | 0 | $\overline{x_1}\ \overline{x_2}\ x_3$ | $\overline{\quad}$ |
| 2 | 0 | 1 | 0 | 1 | $\overline{\quad}\ \overline{\quad}$ | $x_1 + \overline{x_2} + x_3$ |
| 3 | 0 | 1 | 1 | 1 | $\overline{\quad}$ | $x_1 + \overline{x_2} + \overline{x_3}$ |
| 4 | 1 | 0 | 0 | 0 | $x_1\ \overline{x_2}\ \overline{x_3}$ | $\overline{\quad}$ |
| 5 | 1 | 0 | 1 | 1 | $\overline{\quad}$ | $\overline{x_1} + x_2 + \overline{x_3}$ |
| 6 | 1 | 1 | 0 | 0 | $x_1\ x_2\ \overline{x_3}$ | $\overline{\quad}\ \overline{\quad}$ |
| 7 | 1 | 1 | 1 | 1 | | $\overline{x_1} + \overline{x_2} + \overline{x_3}$ |

On en déduit l'expression de *f* sous forme canonique disjonctive et ses écritures condensées :

$$f(x_1, x_2, x_3) = \overline{x_1}\, x_2 \overline{x_3} + \overline{x_1}\, x_2 x_3 + x_1 \overline{x_2}\, x_3 + x_1 x_2 x_3$$
$$= m_2 + m_3 + m_5 + m_7 = \sum m(2, 3, 5, 7)$$

Ainsi que l'expression de *f* sous forme canonique conjonctive et ses écritures condensées :

$$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x_3})(\overline{x_1} + x_2 + x_3)(\overline{x_1} + \overline{x_2} + x_3)$$
$$= M_0 M_1 M_4 M_6 = \Pi M(0, 1, 4, 6)$$

Notons qu'il est possible de passer d'une expression canonique disjonctive à une expression canonique conjonctive (et vice et versa) en considérant les termes manquants dans la duale. Ce point peut être constaté dans le cas des expressions de la fonction *f*.

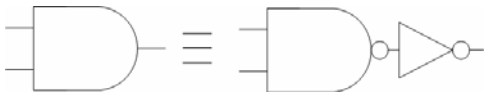
### 3.2.3 Coût d'un circuit

Afin de décider quelle implémentation d'une fonction logique est la plus appropriée, on propose une métrique d'évaluation du coût d'un circuit. Cette métrique n'est pas à proprement parler exacte et universelle, mais elle va nous permettre de mettre une valeur numérale sur un circuit et de décider de son caractère économique.

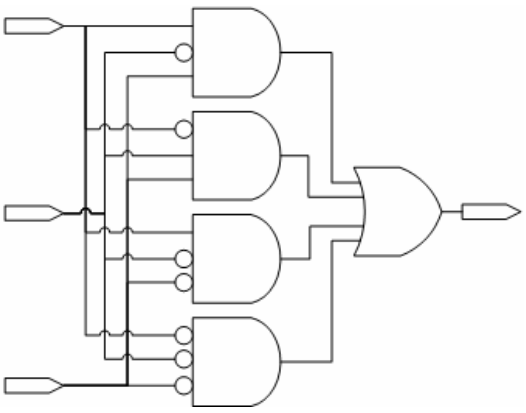
La métrique s'énonce comme suit :

Une porte (ET/OU) inversée avec N entrées a un coût de N+1
Une porte (ET/OU) non inversée avec N entrées a un coût de N+2

On fait une distinction entre les portes inversées et non inversées (on veut dire dont la sortie est inversée) pour refléter la réalité de la technologie CMOS dans laquelle une porte non inversée (ET, OU) est réalisée en faisant suivre d'un inverseur la version inversée de cette porte (NET, NOU) :



Pour des raisons de simplification des calculs on ne considère pas le coût des inverseurs à l'entrée d'une porte. On peut prendre pour exemple le circuit suivant :

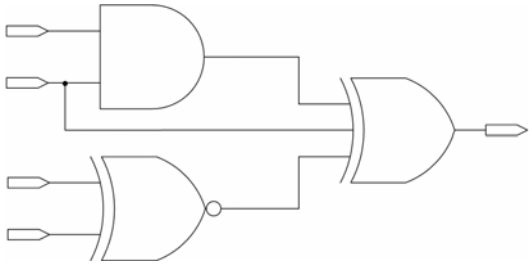


Nous aurons :

4 ET à 3 entrées : coût = 4(3+2)=20
1 OU à 4 entrées : coût = 4+2=6
Coût total = 20+6 = 26

Comme on peut le constater, nous n'avons pas pris en considération les inverseurs (cercles) aux entrées des portes ET du circuit, suivant la convention que nous avons établie.

Les circuits issus des expressions canoniques comportent uniquement des portes ET et des portes OU, en plus des inverseurs. Il arrive néanmoins que nous voulions évaluer le coût d'un circuit comportant des portes XOR ou XNOR. Par convention, on admettra que le coût associé à ces portes est de 2N, N étant le nombre d'entrée. Par exemple, pour le circuit suivant :



Nous aurons :

1 ET à 2 entrées : coût = 2+2=4
1 XOR à 3 entrées : coût = 2·3 = 6
1 XNOR à 2 entrées : coût = 2·2 = 4
Coût total = 4+6+4 = 14

### 3.2.4 Implémentation en NON-ET et NON-OU.

Pour chaque fonction logique, il existe une multitude d'implémentations possibles. Or nous souhaitons généralement réduire le coût du circuit, suivant la convention de la section 3.2.3. Comme nous avons pu le voir, le coût d'un NON-ET (respectivement NON-OU) est inférieur à celui d'un ET (respectivement OU). Aussi peut-il être avantageux d'impliquer ces deux portes dans la conception.

En partant des constats suivants :

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

$$\overline{A \cdot A} = \overline{A} \quad ; \quad \overline{\overline{A \cdot B}} = AB \quad ; \quad \overline{\overline{A} \cdot \overline{B}} = A+B$$

On voit que toute fonction logique, exprimée sous forme de termes combinés par les opérateurs ET, OU et NON, peut être ramenée à une expression utilisant uniquement l'opérateur NON-ET (ou uniquement l'opérateur NON-OU).

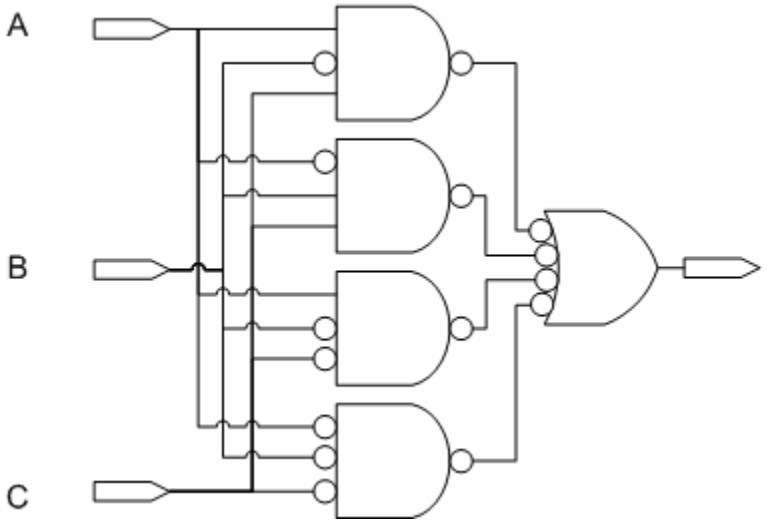
Reprenons à titre d'illustration le circuit de la section suivante, que nous exprimons en fonction des entrées A,B et C :

$$A\,\overline{B}\,C + \overline{A}\,BC + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$$

Le coût du circuit fut évalué à 26 à la section 3.2.3. Essayons de l'exprimer sous la forme de NON-ET. Il suffit pour ce faire d'appliquer les théorèmes de l'algèbre de Boole. Par exemple en appliquant une double inversion sur chaque *minterm* (sans rien changer à la valeur de l'expression), on trouve :

$$\overline{\overline{A\,\overline{B}\,C}} + \overline{\overline{\overline{A}\,BC}} + \overline{\overline{A\,\overline{B}\,\overline{C}}} + \overline{\overline{\overline{A}\,\overline{B}\,\overline{C}}}$$

L'implémentation de cette nouvelle expression donne un circuit dont le coût est 22=4(3+1)+(4+2) :



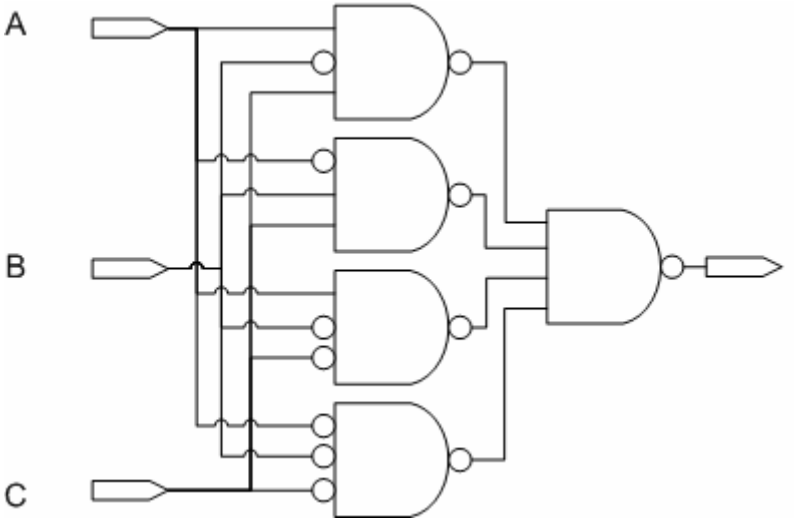

En appliquant une double inversion sur cette nouvelle expression, on obtient :

$$\overline{\overline{A\,\overline{B}\,C} + \overline{A}\,BC + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}}$$

Et en utilisant le théorème de De Morgan :

$$\overline{\left(\overline{A\,\overline{B}\,C}\right)\left(\overline{\overline{A}\,BC}\right)\left(\overline{A\,\overline{B}\,\overline{C}}\right)\left(\overline{\overline{A}\,\overline{B}\,\overline{C}}\right)}$$

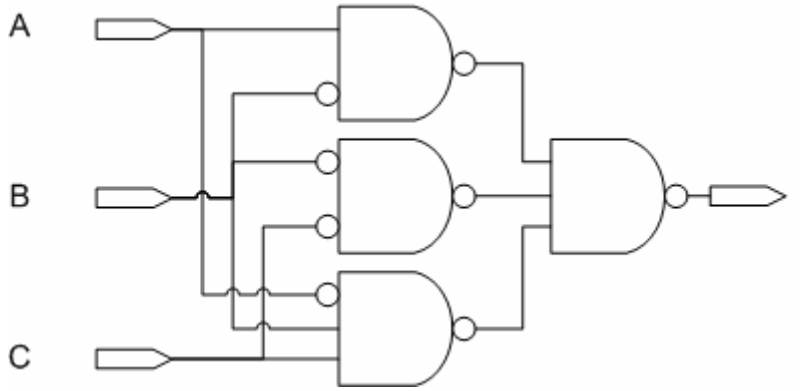
Ce qui donne un circuit dont le coût est : 4(3+1)+(4+1) = 21




Comme on peut le constater, le coût est passé de 26 (l'implémentation initiale) à 21 (implémentation en NON-ET). Les implémentations en NON-ET et en NON-OU sont souvent bien meilleures que leur équivalent SOP et POS.

Il est laissé en exercice la démonstration que le coût de l'implémentation en NON-OU de cette expression vaut 21 également (ce ne sera pas toujours le cas).

Notons pour finir que l'implémentation optimale de l'expression logique a un coût de 14. Les manipulations algébriques aboutissant à ce résultat sont laissées en exercice.




2 NON-ET à 2 entrées : coût = 2(2+1)=6
2 NON-ET à 3 entrées : coût = 2(3+1)=8
Coût total = 6+8 = 14

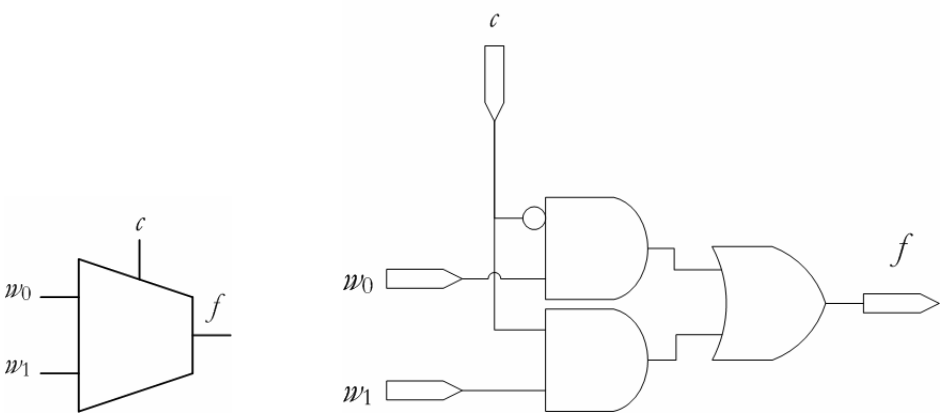
## 3.3 Circuits usuels non arithmétique

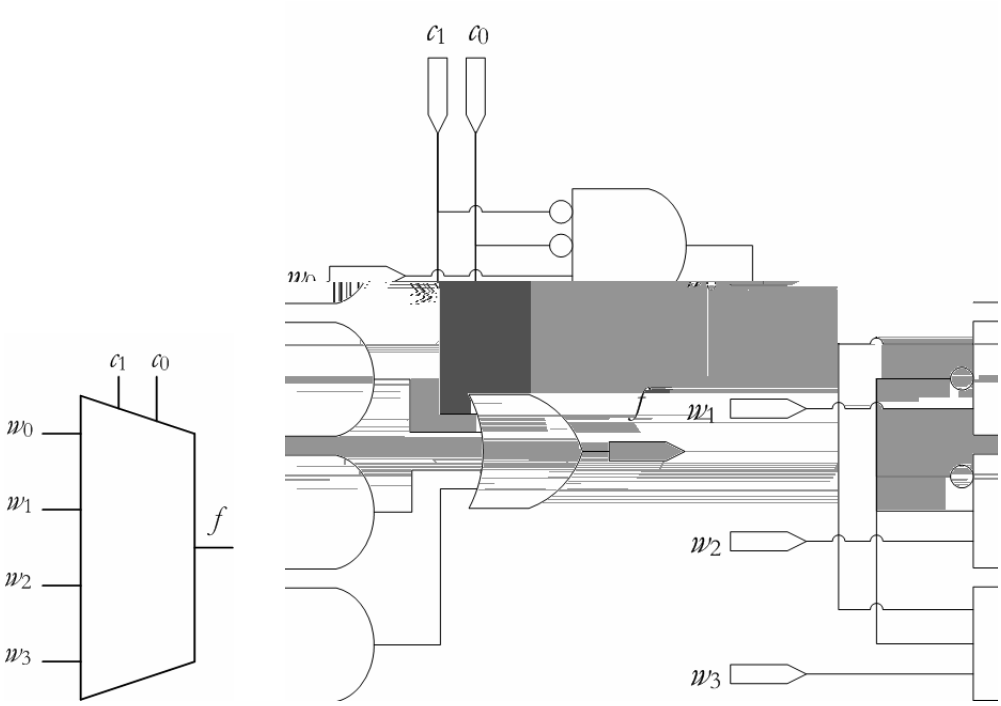
Dans ce qui précède, nous avons synthétisé des circuits de toutes pièces en partant d'une table de vérité ou d'une fonction logique et en les réalisant à l'aide de portes logiques. Dans la pratique, de nombreuses fonctions de haut niveau sont réalisées par des circuits prédéfinis, communément appelés circuits usuels, dont le concepteur fait usage pour traduire immédiatement le comportement de son circuit. Les circuits usuels que nous présentons ici sont ceux qui ne sont pas nécessairement impliqués dans des opérations arithmétiques.

### 3.3.1 Le multiplexeur

Le multiplexeur (souvent désigné par mux) est sans conteste le circuit usuel le plus utilisé. Le rôle du multiplexeur est d'acheminer au choix une entrée parmi plusieurs. Le multiplexeur possède donc plusieurs entrées et des signaux de contrôle permettant d'acheminer un de ces signaux vers sa sortie.

Voici quelques exemples de multiplexeurs et leur implémentation :






Le fait que le multiplexeur implémente une décomposition de Shannon est une de ses propriétés remarquables. On se souviendra ici que la décomposition de Shannon donne :

$$f(x_0, x_1, x_2, \dots, x_n) = \overline{x_0}\, f(0, x_1, x_2, \dots, x_n) + x_0 f(1, x_1, x_2, \dots, x_n)$$

Si on regarde l'implémentation du mux de 2 à 1, la fonction qu'il implémente est :

$f(c, w_0, w_1) = \overline{c}\, w_0 + c_0 w_1 = \overline{c} f(0, w_0, w_1) + c_0 f(1, w_0, w_1)$, où on trouve par identification des termes:

- $f(c, w_0, w_1) = w_0$ si $c = 0$
- $f(c, w_0, w_1) = w_1$ si $c = 1$

On voit clairement clair que le signal de contrôle $c$ permet de choisir (mettre en sortie) l'entrée $w_0$ si c=0, et l'entrée $w_1$ si c = 1. Le même raisonnement peut être appliqué au mux de 4 à 1 en appliquant récursivement la décomposition de Shannon.

$$f(x_0, x_1, x_2, \dots, x_n) = \overline{x_0}\,\overline{x_1}\, f(0, 0, x_2, \dots, x_n) + \overline{x_0}\, x_1 f(0, 1, x_2, \dots, x_n)$$
$$+ x_0 \overline{x_1}\, f(1, 0, x_2, \dots, x_n) + x_0 x_1 f(1, 1, x_2, \dots, x_n)$$

Or si on regarde l'implémentation donnée plus haut du mux de 4 à 1, on a :

$$f(c_0, c_1, w_0, w_1, w_2, w_3) = \overline{c_0}\,\overline{c_1}\, w_0 + \overline{c_0}\, c_1 w_2 + c_0 \overline{c_1}\, w_1 + c_0 c_1 w_3$$
$$= \overline{c_0}\,\overline{c_1}\, f(0, 0, w_0, w_1, w_2, w_3) + \overline{c_0}\, c_1 f(0, 1, w_0, w_1, w_2, w_3)$$
$$+ c_0 \overline{c_1}\, f(1, 0, w_0, w_1, w_2, w_3) + c_0 c_1 f(1, 1, w_0, w_1, w_2, w_3)$$

Ce qui donne dans l'ordre numéral d'indexation des variables $c_i$:

- $f(c_0, c_1, w_0, w_1, w_2, w_3) = w_0$ si $c_1$=0 et $c_0$=0
- $f(c_0, c_1, w_0, w_1, w_2, w_3) = w_1$ si $c_1$=0 et $c_0$=1
- $f(c_0, c_1, w_0, w_1, w_2, w_3) = w_2$ si $c_1$=1 et $c_0$=0
- $f(c_0, c_1, w_0, w_1, w_2, w_3) = w_3$ si $c_1$=1 et $c_0$=1

En clair, les signaux de contrôle $c_0$ et $c_1$ permettent de choisir (mettre en sortie) les signaux d'entrée $w_i$, de sorte que la sortie soit:

- $w_0$ si $c_1$=0 et $c_0$=0
- $w_1$ si $c_1$=0 et $c_0$=1
- $w_2$ si $c_1$=1 et $c_0$=0
- $w_3$ si $c_1$=1 et $c_0$=1

Notons aussi que l'index de l'entrée correspond à l'indice des *minterms* :

- $w$ si $c_1c_0 \equiv m_0$
- $w$ si $c_1\ c_0 \equiv m_1$
- $w$ si $c_1\ c_0 \equiv m_2$
- $w$ si $c_1\ c_0 \equiv m_3$

Le multiplexeur peut être généralisé à un composant possédant un nombre d'entrées puissance de 2. On a donc le mux de 8 à 1 (avec 3 signaux de contrôle), des mux de 16 à 1(avec 4 signaux de contrôle), etc.

Un mux peut être utilisé pour générer des fonctions à plusieurs variables. On utilise alors la décomposition de Shannon.

### Exemple :

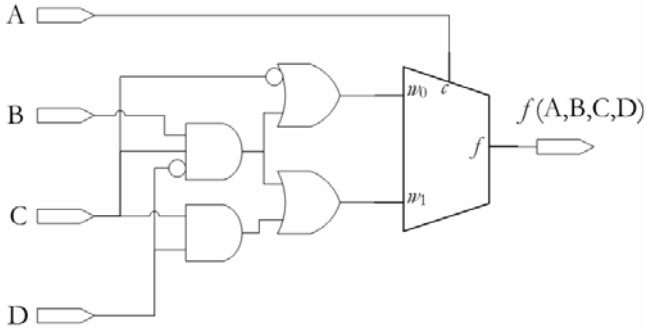
Soit une fonction $f$ de quatre variables logiques, A, B, C et D :

$$f(A,B,C,D) = \overline{A}\ \overline{C} + ACD + BC\overline{D}$$

On veut utiliser un mux 2 à 1 pour implémenter cette fonction. Pour ce faire, nous allons utiliser le signal A comme signal de contrôle. Sachant que

- $f(0,B,C,D) = \overline{C} + BC\overline{D}$
- $f(1,B,C,D) = CD + BC\overline{D}$

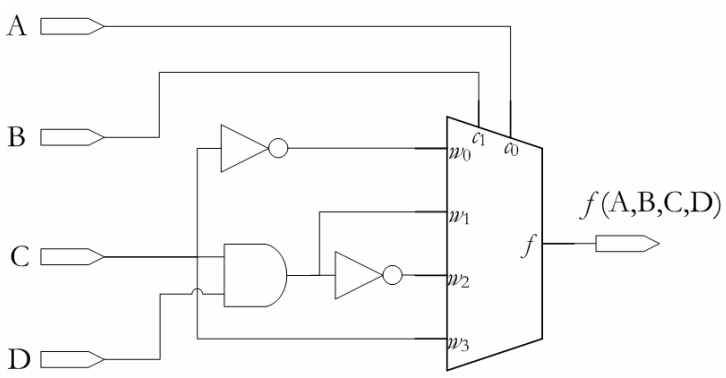
Le circuit à réaliser va de soi :



Nous aurions également pu utiliser un mux de 4 à 1. Pour ce faire, nous prenons A et B comme signaux de contrôle. Ayant :

- $f(0,0,C,D) = \overline{C}$
- $f(1,0,C,D) = CD$
- $f(0,1,C,D) = \overline{C} + C\overline{D} = \overline{C} + \overline{D} = \overline{CD}$
- $f(1,1,C,D) = CD + C\overline{D} = C$

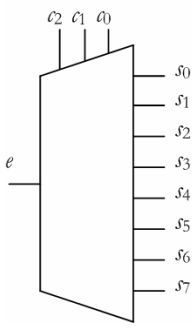
On pourra réaliser $f$ de la manière suivante (relever la position des entrées A et B) :




Il est ici laissé en exercice l'implémentation de cette même fonction à l'aide d'un multiplexeur à 8 entrées.

### 3.3.2 Le démultiplexeur

Le démultiplexeur (démux) fonctionne de façon inverse à celle du multiplexeur. Le démultiplexeur reçoit n signaux de contrôle et une entrée à acheminer vers l'une des $2^n$ sorties possibles. Les autres sorties donnent alors la constante 0. Voici un exemple de démux 1 à 8 avec sa table de vérité.



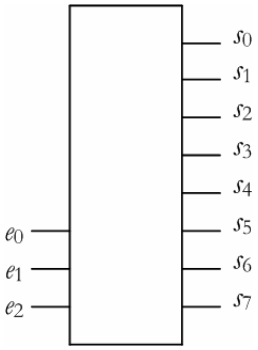

| $c_2$ | $c_1$ | $c_0$ | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $e$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | $e$ | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | $e$ | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | $e$ | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | $e$ | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | $e$ | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $e$ | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $e$ |

Il est important de noter ici que l'entrée $e$ a été inscrite dans la table de vérité, indiquant le report de l'entrée à la sortie. On relèvera que l'index i de la sortie $s_i$ par laquelle est acheminé $e$ correspond à la valeur binaire formé par les signaux de commande : $i=c_2c_1c_0$.

Dans l'application pratique, le démux est peu utilisé dans cette forme. On lui préfère un composant dérivé : le décodeur.

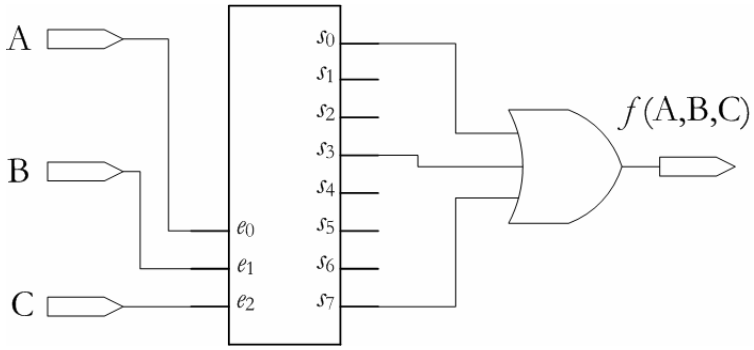
### 3.3.3 Le décodeur

Le décodeur fonctionne de façon très similaire à celle du démultiplexeur, l'entrée $e$ ayant été fixée à 1. Le décodeur reçoit n signaux de contrôle et donne $2^n$ sorties.




| $e_2$ | $e_1$ | $e_0$ | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Le décodeur est utilisé pour effectuer du décodage de ligne dans les composants de mémoire que nous verrons plus loin. Le circuit peut également être exploité pour réaliser des fonctions logiques en tenant compte du fait que les sorties correspondent aux *minterms* d'une expression logique. Par exemple, le circuit suivant :




Implémente la fonction :

$$f(C, B, A) = \Sigma s(0, 3, 7) = \overline{A}\ \overline{B}\ \overline{C} + AB\overline{C} + ABC$$

Relevons que l'ordre des signaux d'entrée est important.

### 3.3.4 L'encodeur de priorité

L'encodeur de priorité est un circuit à cheval entre la série décodeur/encodeur présentée jusqu'ici et celle des circuits arithmétiques qui seront vus plus loin. L'encodeur de priorité est un circuit qui détecte la position du premier bit 1 d'un mot en commençant par le bit le plus significatif (le plus à gauche).

Considérons les mots binaires suivants :

- 0010 ;
- 0110 ;
- 0001 ;
- 0100 ;
- 0110.

On considérant la notation

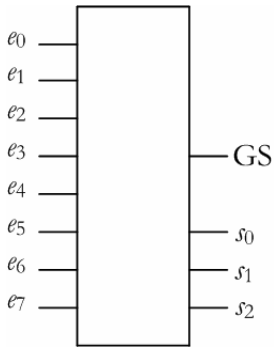
$a_3 a_2 a_1 a_0$

On cherche à connaître l'index i du premier bit $a_i=1$ en partant de la gauche. Si on reprend les nombres précédents, nous aurons :

- 0010 ⇨ 1 où $a_1=1$;
- 0110 ⇨ 2 où $a_2=1$;
- 0001 ⇨ 0 où $a_0=1$;
- 0100 ⇨ 2 où $a_2=1$;
- 0110 ⇨ 2 où $a_2=1$;
- 1000 ⇨ 3 où $a_3=1$.

De manière générale, l'encodeur de priorité prend en entrée un mot de $2^n$ bits et donne en sortie un mot de n bits correspondant à l'index du bit non nul le plus significatif. Une question se pose quand tous les bits à l'entrée valent 0 ? On ajoute alors un signal indicateur (appelé GS) qui vaut 1 uniquement dans ce cas.

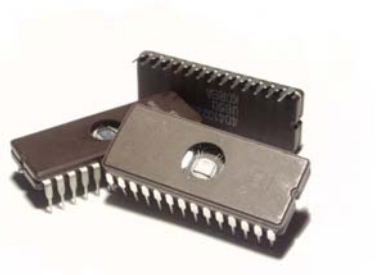
Pour le cas d'un encodeur de priorité à 8 entrées, nous aurons la table de vérité qui suit. Les tirets de la table de vérité signifient que les entrées peuvent indifféremment prendre la valeur 0 ou 1. Par exemple, les combinaisons d'entrées 10000000 et 1010111 tombent toutes deux dans le cas de la première ligne.




| $e_7$ | $e_6$ | $e_5$ | $e_4$ | $e_3$ | $e_2$ | $e_1$ | $e_0$ | $s_2$ | $s_1$ | $s_0$ | GS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | - | - | - | - | - | - | 1 | 1 | 1 | 0 |
| 0 | 1 | - | - | - | - | - | - | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | - | - | - | - | - | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | - | - | - | - | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | - | - | - | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | - | - | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | - | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## 3.4 Circuits logiques programmables

Dès l'invention des circuits intégrés en 1958 par le prix nobel de physique et employé de *Texas Instrument* Jack Kilby, la question de rendre les composants électroniques programmables est devenue un enjeu crucial dans le monde des circuits numériques.



*Circuits programmables avec ouverture sur le boîtier*

Les premiers circuits logiques étaient des mémoires mortes (ROM) qui encodaient physiquement l'ensemble des tables de vérité utilisées. En 1973, *National semiconductor* va introduire un nouveau circuit intégré programmable, le PLA, fortement inspiré de l'architecture des ROM. Ce composant va bouleverser et emballer toute l'industrie de l'électronique. Mais sa programmation demeure difficile et échaude nombre de concepteurs. C'est en 1978 que la compagnie Monolithic Memories Inc. (MMI), aujourd'hui rachetée par AMD, introduit le PAL, un dérivé du PLA beaucoup plus simple à utiliser et à programmer. Cette aisance d'utilisation du PAL va largement populariser les composants programmables et en accélérer le développement.

Les circuits programmables les plus utilisés aujourd'hui sont les circuits FPGA (acronyme anglais pour *Field Programmable Gate Array*). Ces composants sont beaucoup plus versatiles et puissants que leurs premiers ancêtres. Les FPGA permettant de réaliser des circuits complexes de très grande taille (plusieurs millions de portes logiques) et cette grande densité nécessite l'utilisation de langages de description (programmation) matérielles, appelés HDL pour *Hardware Description Language*, les plus populaires étant le VHDL et le Verilog.

Dans le restant de ce chapitre, nous allons tâcher de survoler l'architecture des premiers circuits programmables et d'éclaircir certains concepts qui s'y rattachent.

### 3.4.1 Charte symbolique des circuits programmables

Les circuits logiques programmables que nous allons considérer ici sont :

- PROM : Acronyme anglais de *Programmable Read-Only Memory* : Mémoire morte programmable
- PLA : Acronyme anglais de *Programmable Logic Array* : qui est un réseau de portes logiques programmables
- PAL™ : Acronyme anglais de *Programmable Array Logic* : est une marque déposé pour un circuit dérivé du PLA.

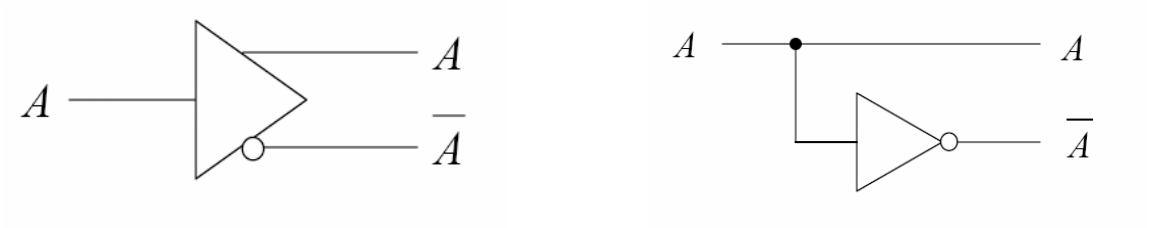
Mais avant de ce faire, il nous faut définir une charte symbolique pour représenter l'utilisation des fusibles dans les circuits numériques.

Il existe diverses façons de réaliser un circuit programmable. Une des techniques les plus anciennes, et certainement celle qui a été la plus popularisée, consiste à utiliser un fusible, c'est-à-dire un fil qu'il est possible de « brûler » si on le soumet à un courant fort.

On utilisera pour nos besoins d'illustration les symboles suivants. À gauche, une porte logique comportant des fusibles à son entrée. À droite, le symbole que nous utiliserons pour la même porte.

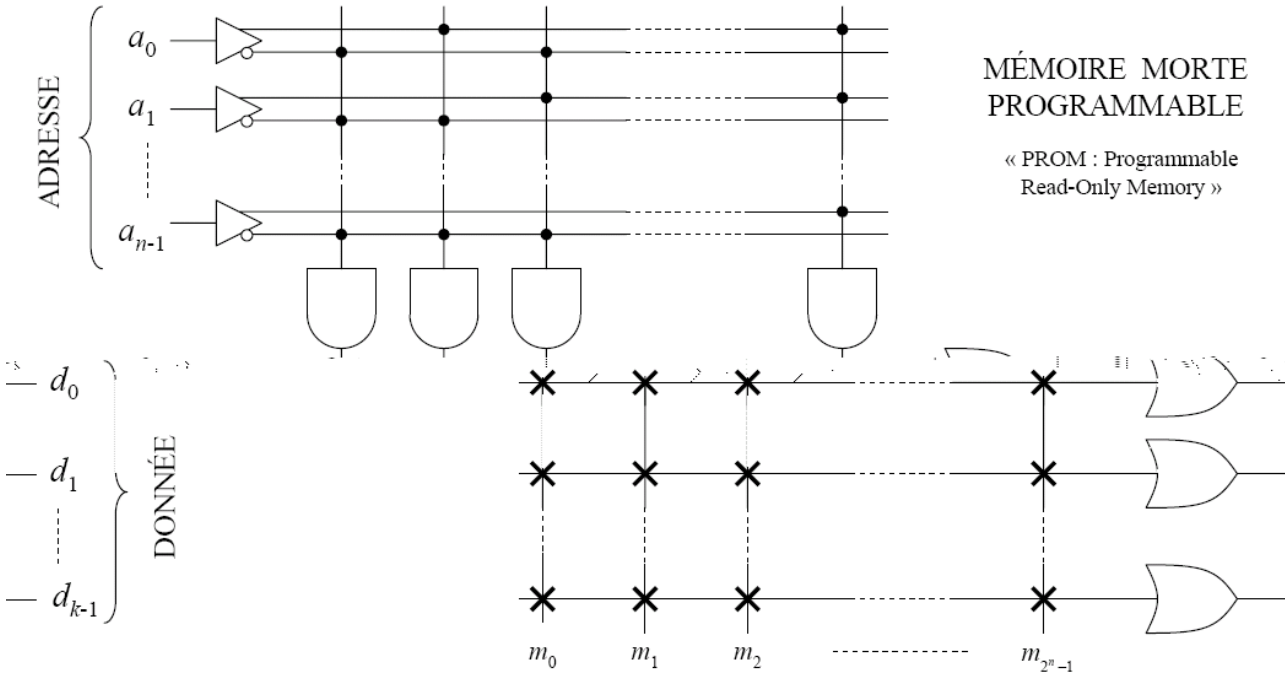

Cette écriture réduite permet donc de représenter le fusible par un X est un grand nombre d'entrées d'une porte logique par un seul fil. Si le X est absent à l'intersection, on déduira que le fusible a été brûlé :

Il arrive souvent que les signaux d'entrée aient besoin d'être utilisés sous une forme inversée et non inversée. Pour ce faire, on utilise le symbole suivant :




## 3.4.2 PROM

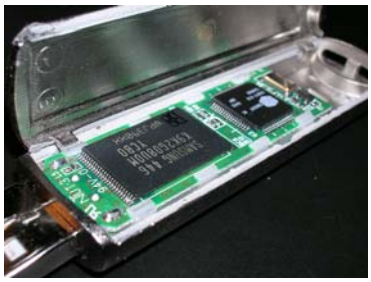
La PROM est une mémoire morte programmable. On entend par mémoire morte qu'il s'agit là de mémoire non-volatile, ne causant aucune perte de données si le circuit n'est plus alimenté. La figure suivante présente l'architecture conceptuelle d'une PROM :




La PROM a pour entrées des signaux $a_i$ qui forment ce qu'on appelle une adresse. Pour chaque vecteur d'adresse , la PROM présente en sortie un vecteur de sortie formé par les signaux $d_i$. Comme on peut le voir sur la figure, les signaux d'adresse sont traités par un réseau planaire de connexions permettant d'acheminer vers des portes ET chacun des *minterms* possibles pour le vecteur . Ainsi, une seule des portes ET présente active sa sortie si son adresse apparaît en entrée, ce qui n'est pas sans rappeler le fonctionnement d'un décodeur. Et de fait, ce réseau planaire de connexions est généralement implémenté par un décodeur.

Le reste est simple à comprendre. Une fois qu'une colonne $m_j$ est sélectionnée par l'activation d'une porte ET, chacun des signaux du vecteur $d_i$ va conduire ou non en fonction de la présence ou de l'absence du fusible. Si le fusible est présent, la sortie $d_i$ donnera 1, elle donnera 0 autrement.

Il existe différentes variantes de mémoires mortes. Certaines utilisent des technologies différentes de celle nécessitant l'emploi de fusibles. Par exemple, une technologie appelée EPROM (acronyme anglais de *Erasable Programmable Read Only Memory*) est une mémoire morte effaçable et reprogrammable. Pour ce faire, elle utilise une technologie basée sur des transistors qui recréent une connexion (régénère en quelque sorte leur fusible) quand on les expose à la lumière ultraviolette. On reconnaît les EPROM par la présence d'une petite ouverture sur le boîtier du circuit intégré, comme le montrait l'illustration en début de la section 3.4.



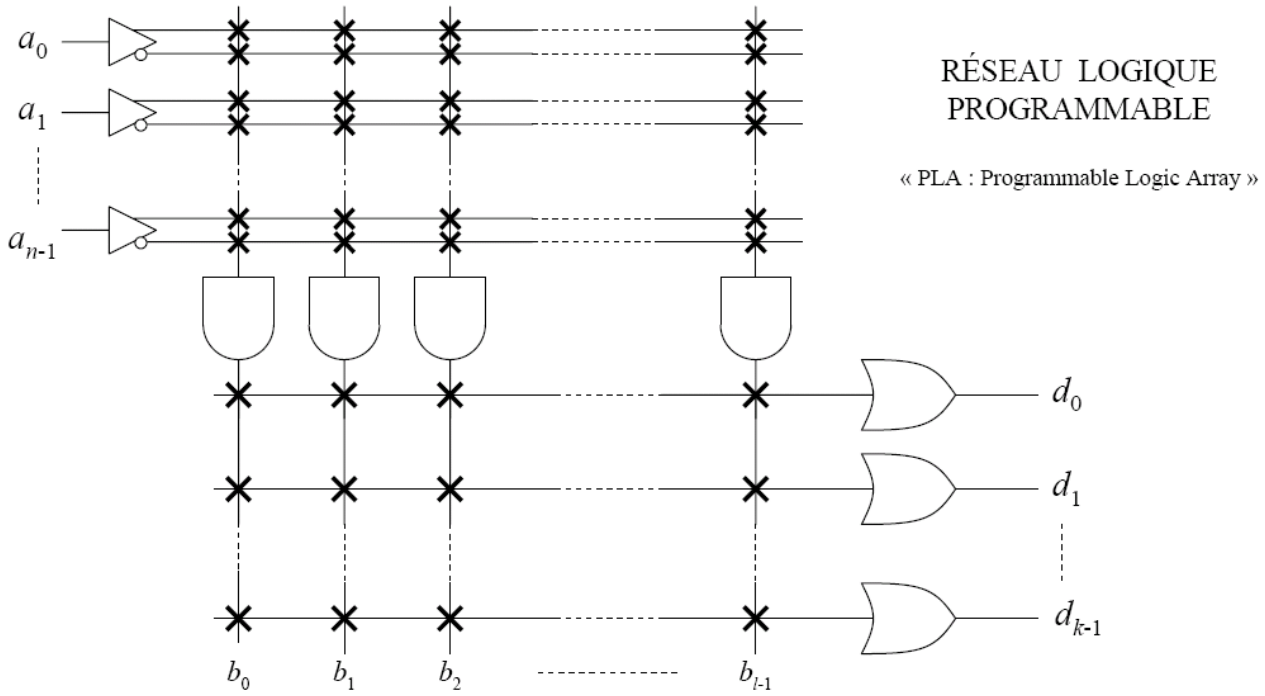
*Intérieur d'un boitier de mémoire Flash*

Les EEPROM (acronyme anglais de *Electrically Erasable Programmable Read Only Memory*) sont des mémoires mortes effaçable électriquement. On les retrouvait dans les ordinateurs où elles permettaient d'enregistrer les paramètres du BIOS et de reprogrammer le BIOS au besoin sans exposition à l'ultraviolet.

Une autre forme de mémoire morte fortement répandue de nos jours est la mémoire Flash. Cette technologie est également effaçable électriquement mais elle contient de la logique supplémentaire pour gérer des « pages » (un ensemble de données aux adresses contigües).

### 3.4.3 PLA

Le PLA est un réseau de portes logiques programmables permettant de réaliser des MIMO de façon rapide.




Un PLA est très similaire par son architecture à une PROM. Les signaux d'entrées du vecteur  traversent un réseau planaire de connexions appelé « plan de ET programmable ». Ces signaux permettent de sélectionner un nombre donné de colonnes $b_j$ à la fois, et ces colonnes sont alors acheminées par un seconde réseau planaire de connexions appelé « plan de OU programmable ». Il devient ainsi possible de réaliser plusieurs fonctions logiques sous une forme disjonctive. Néanmoins, le fait de devoir concevoir un circuit en pensant à programmer deux matrices de fusibles rend le PLA peu accessible.

Les PLA ont eu beaucoup de difficulté à atteindre leur public cible, les concepteurs de circuits numériques, qui trouvaient difficile la compréhension de leur fonctionnement du fait qu'il fallait concevoir un MISO en pensant à programmer deux matrices qui devaient agir en conjonction.

### 3.4.4 PAL

En 1978, la compagnie Monolithic Memories, Inc. (MMI) créa le PAL (acronyme anglais de *Programmable Array Logic*) qui s'inspire du PLA dans son architecture, mais n'utilise plus qu'un seul plan reconfigurable : le plan de ET. Le plan de OU est alors fixé par le manufacturier, ce qui limitait à l'évidence les fonctions réalisables mais convenait beaucoup mieux aux besoins pratiques de l'industrie de l'électronique.

Le PAL rencontra le succès espéré dans l'industrie, à tel point que la compagnie gratifia ses deux concepteurs du PAL d'une voiture de luxe. Il est à noter cependant que la compagnie IBM possédait un circuit similaire au PAL depuis 1975 qu'elle ne manufacturait que pour ses besoins internes.

La figure suivante présente le schéma d'un circuit PAL.