Chapitre 1 : Circuits Combinatoires

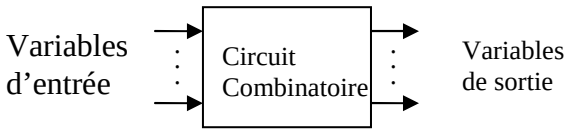
1. Introduction

Notons qu'il existe deux types de circuits logiques :

Circuits logiques séquentiels : c'est des circuits dont les valeurs de sortie dépendent d'entrée appliquées ultérieurement.

Circuits logiques combinatoires : c'est des circuits dont les valeurs de sortie ne dépendent que de ses valeurs d'entrée.

Un circuit logique combinatoire est constitué d'éléments logiques élémentaires appelés portes logiques, elles reçoivent des signaux appliqués en entrée et produisent des signaux en sortie.



Symbole logique d'un circuit combinatoire

L'étude des circuits combinatoires, consiste en deux étapes :

Synthèse : réaliser le circuit combinatoire à partir de l'énoncé décrivant les fonctions ou le rôle du circuit, en question.

Analyse : déterminer le rôle du circuit combinatoire à partir de son logigramme.

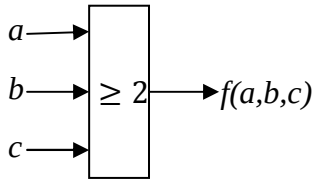
2. Synthèse d'un circuit combinatoire

Pour générer le logigramme d'un circuit logique à partir de son fonctionnement, on suit, en général, les étapes suivantes :

1. Lire et comprendre les énoncés du problème afin de déterminer le nombre de variables d'entrée et de variables de sortie du circuit à réaliser.
2. Donner des noms symboliques aux variables d'entrée et de sortie.
3. Etablir la table de vérité.
4. Simplifier les fonctions logiques correspondantes aux sorties.
5. Réaliser le logigramme à partir des fonctions logiques simplifiées.

Exemple : (Circuit 2/3)

Etablissons le logigramme d'un circuit logique comportant 3 entrées et 1 sortie, celle – ci étant à l'état 1 si 2 au moins des trois entrées sont à l'état 1. Un tel circuit, on le représente par le schéma bloc suivant :



Symbole logique du circuit 2/3

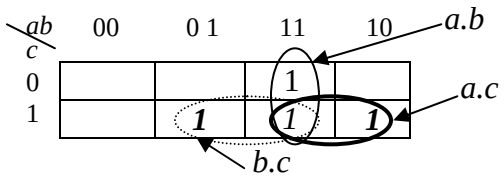
- Table de vérité

a	*B*	*c*	*f(a, b, c)*
0	*0*	*0*	*0*
0	*0*	*1*	*0*
0	*1*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*1*

- Expression logique

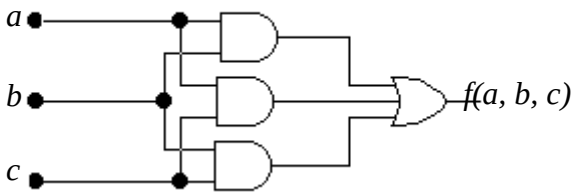
$$f(a,b,c) = \bar{a}.b.c + a.\bar{b}.c + a.b.\bar{c} + a.b.c.$$

- Simplification



$$f(a,b,c) = a.b + a.c + b.c$$

- Logigramme



Circuit 2/3

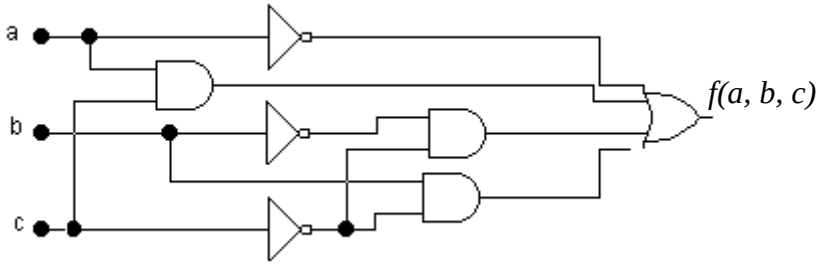
3. Analyse d'un circuit combinatoire

Pour analyser un circuit combinatoire, on suit les étapes suivantes :

1. Déterminer les expressions logiques des variables de sortie.
2. Dresser la table de vérité du circuit et la traduire par un énoncé décrivant le rôle du circuit.

Exemple

Quel est le rôle de la fonction *f* représentée par le logigramme suivant :



- Expression logique

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

- Table de vérité

a	*b*	*c*	*f(a, b, c)*
0	*0*	*0*	*1*
0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*1*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*1*

Le rôle du circuit est de produire la constante 1.

4. Exemples de circuits combinatoires

On fera la synthèse des circuits combinatoires les plus utilisés dans la technologie des ordinateurs et ailleurs.

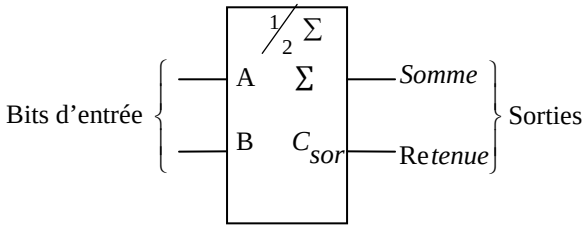
4.1. Additionneurs de base

4.1.1. Demi -additionneur

Rappelons les règles d'addition binaire :

Entrées *A B*	*Sorties* *Somme ($\sum$)*	*Retenue (C_{sor})*
0 + 0	*0*	*0*
0 + 1	*1*	*0*
1 + 0	*1*	*0*
1 + 1	*0*	*1*

Ces opérations s'effectuent par un circuit logique appelé un demi-additionneur, qu'on note DA. Un DA est symbolisé par le symbole logique suivant :

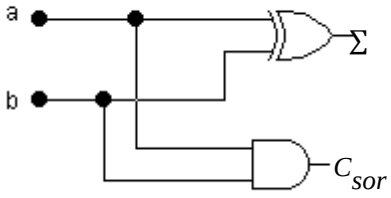


Symbole logique d'un DA

- Expressions logiques

$$\sum = A \oplus B$$
$$C_{sor} = A.B$$

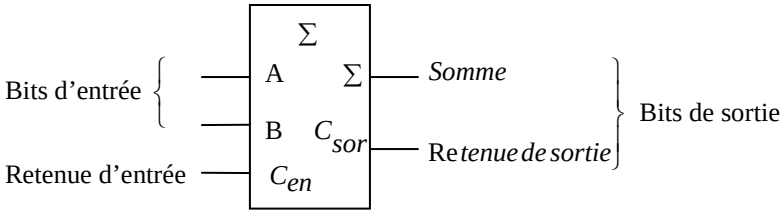
- Logigramme



Demi-Additionneur

4.1.2. Additionneur complet

Il faut tenir compte de la retenue éventuelle des bits de poids inférieurs, d'où la nécessité d'un circuit additionneur comportant trois entrées et deux sorties ; c'est ce qu'on appelle : Additionneur Complet, qu'on note AC et dont le symbole logique est :



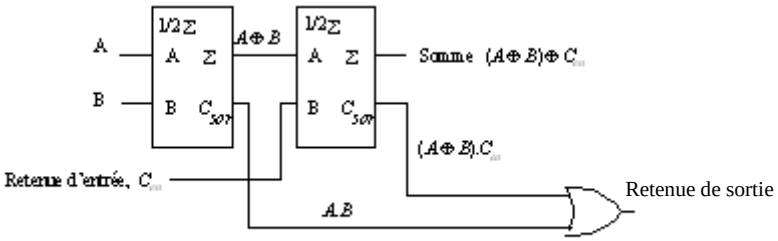
Symbole logique d'un AC

- Table de vérité

A	B	C_{en}	Σ	C_{sor}
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*

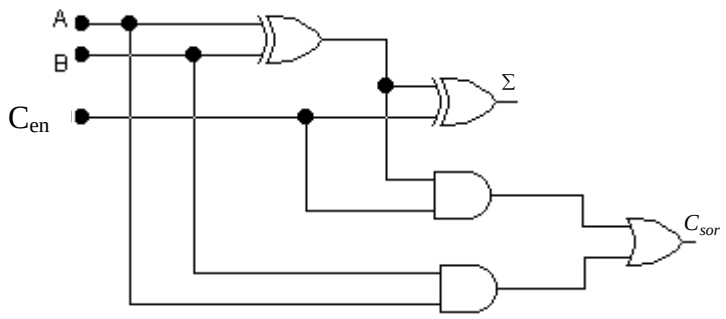
- Expressions logiques

$$\sum = (A \oplus B) \oplus C_{en}$$
$$C_{sor} = A.B + (A \oplus B).C_{en}$$



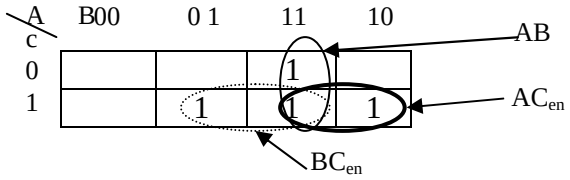
Représentation d'un AC par deux DA

- Logigramme



Additionneur complet

En pratique pour minimiser le nombre de composants, ou de portes logiques dans un circuit intégré, un tel additionneur n'est pas réalisé directement. Nous pouvons simplifier l'expression de C_{sor} en utilisant la table de Karnaugh.



Nous en déduisons :

$$C_{sor} = AB + AC_{en} + BC_{en}$$

De même, on peut démontrer que

$$\overline{C_{sor}} = \overline{A}\,\overline{B} + \overline{A}\,\overline{C_{en}} + \overline{B}\,\overline{C_{en}}$$

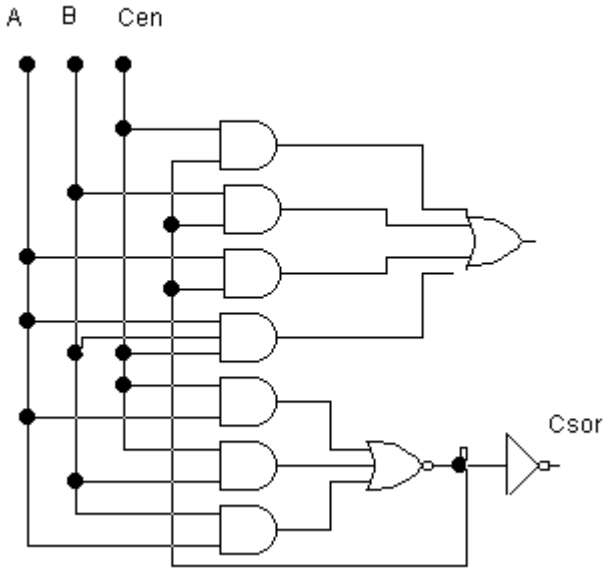
A partir de cette relation, nous pouvons écrire :

$$\begin{cases} A\overline{C}_{sor} = A\overline{B}\,\overline{C_{en}} \\ B\overline{C}_{sor} = \overline{A}B\overline{C_{en}} \\ C_{en}\overline{C}_{sor} = \overline{A}\,\overline{B}C_{en} \end{cases}$$

Ce qui nous permet de réécrire l'expression de $\sum$:

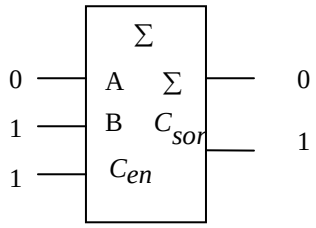
$$\sum = (A + B + C_{en})\overline{C_{sor}} + ABC_{en}$$

Soit le logigramme d'un additionneur complet



Additionneur complet

Exemple



4.1.3. Additionneurs binaires parallèles

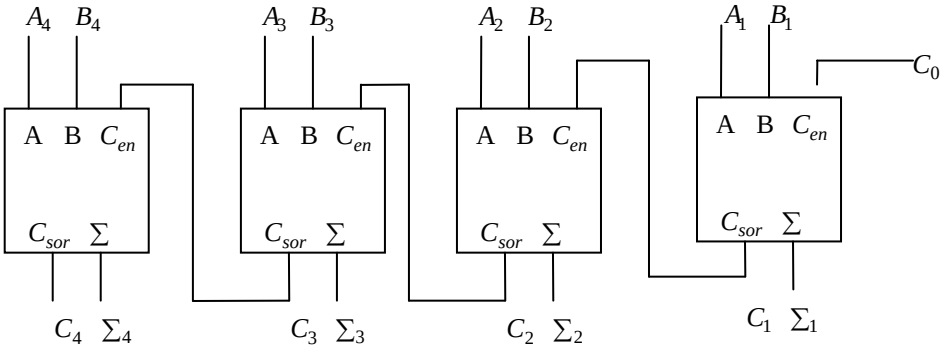
Sachant qu'un additionneur complet ne peut traiter que deux nombres de 1 bit et une retenue d'entrée ; pour additionner des nombres de plus d'un bit, il faut utiliser des additionneurs complets supplémentaires.

Un additionneur parallèle à *n* bits est le branchement en cascade de *n* additionneurs complets ; où la sortie de retenue de chaque additionneur est connectée à l'entrée de retenue de l'additionneur du bit de rang plus élevé suivant.

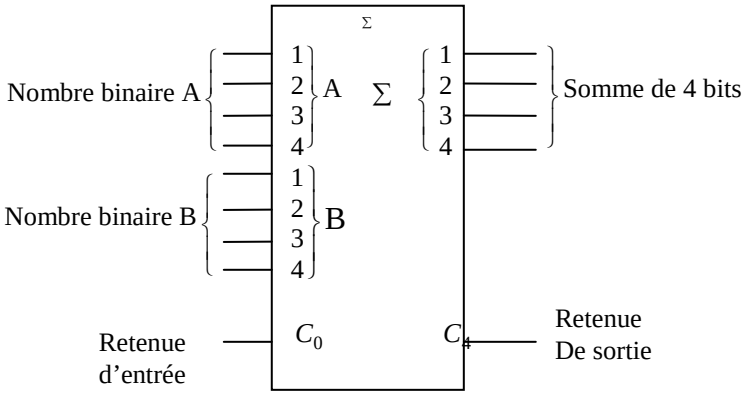
Notez qu'on peut utiliser un DA pour la position de poids le plus faible, ou relier l'entrée de retenue d'un AC à la masse (0), puisqu'il n'y a pas d'entrée de retenue pour la position de bit de poids le plus faible.

Exemple : additionneur parallèle à *4* bits

Soit à additionner les nombres binaires : $A = A_4A_3A_2A_1$ *et* $B = B_4B_3B_2B_1$.

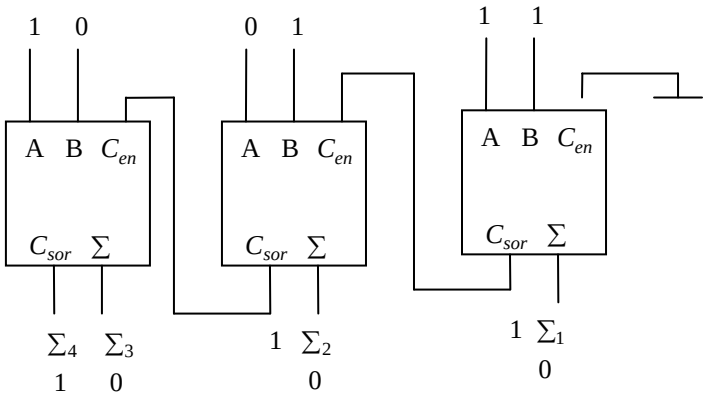


Additionneur parallèle à 4 bits



Symbole logique d'un additionneur parallèle à 4 bits

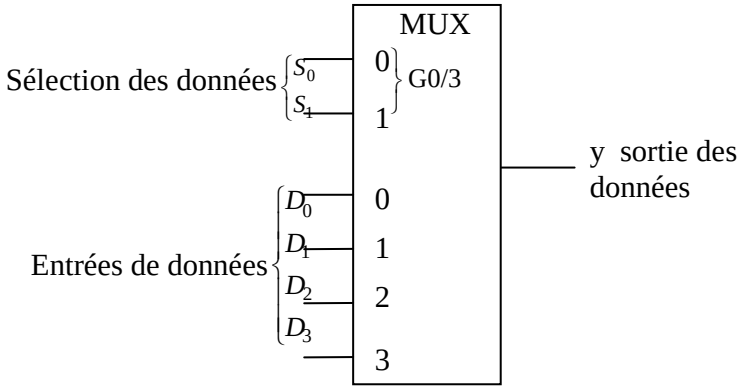
Exemple
Soit à additionner : 101+011.



4.2. Multiplexeurs

Un multiplexeur (MUX) est un circuit logique permettant d'acheminer les informations numériques de plusieurs sources sur une seule ligne, donc un MUX de base possède 2^n lignes de données d'entrée et une seule ligne de sortie. Il comporte aussi *n* entrées de sélection des données, permettant de désigner l'adresse de l'entrée choisie comme sortie. Les Mux sont aussi connus sous le nom de sélecteurs de données.

Exemple : MUX à 4 entrées



Symbole logique d'un MUX 4 → 1

Ce MUX possède 2 lignes de sélection des données, puisqu'il est possible de sélectionner l'une ou l'autre des 4 lignes d'entrée de données avec seulement 2 bits. Soit, la table de vérité suivante :

Entrées de sélection		*Entrée sélectionnée*
S_1	S_0	
0	*0*	D_0
0	*1*	D_1
1	*0*	D_2
1	*1*	D_3

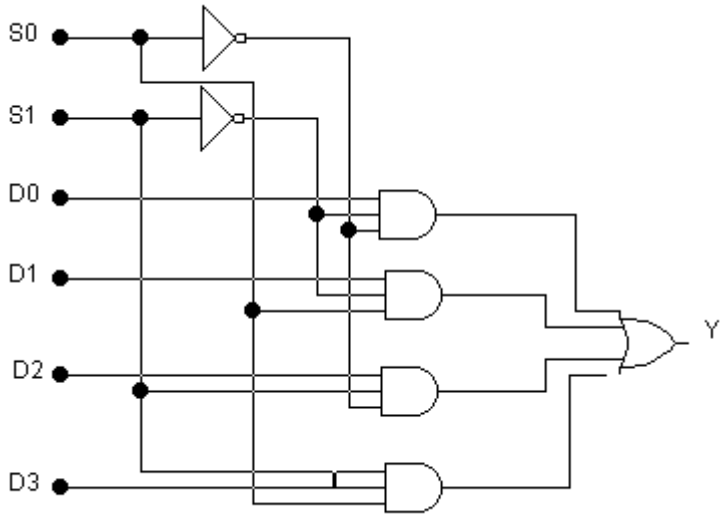
La sortie des données est égale à D_0 seulement si $S_1 = 0$ et $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

La sortie des données est égale à D_1 seulement si $S_1 = 0$ et $S_0 = 1$: $Y = D_1\overline{S}_1 S_0$.

La sortie des données est égale à D_2 seulement si $S_1 = 1$ et $S_0 = 0$: $Y = D_2 S_1 \overline{S}_0$.
La sortie des données est égale à D_3 seulement si $S_1 = 1$ et $S_0 = 1$: $Y = D_3 S_1 S_0$.
D'où la fonction de sortie :

$$Y = D_0.\overline{S}_1.\overline{S}_0 + D_1.\overline{S}_1.S_0 + D_2.S_1.\overline{S}_0 + D_3.S_1.S_0 .$$

Soit, le logigramme correspondant est :



MUX 4 → 1

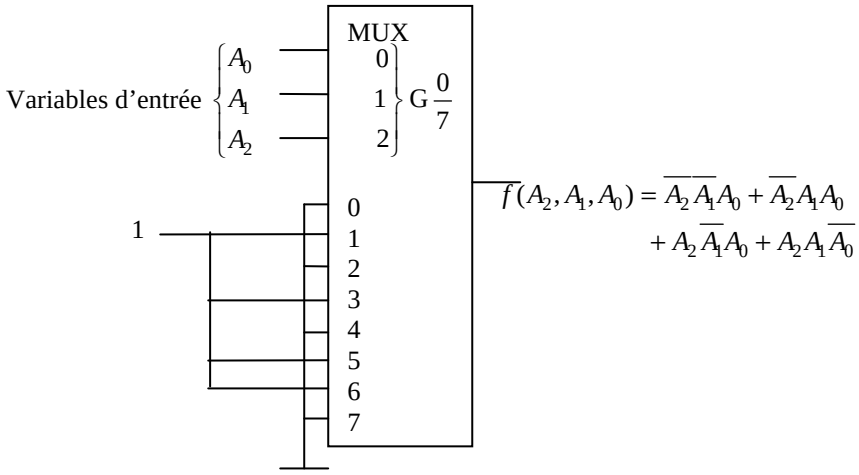
Générateur de fonctions logiques : Une application utile du multiplexeur est de générer des fonctions logiques combinatoires sous forme de sommes de produits. Il permet donc une transcription directe de la table de vérité.

Exemple 1

Considérons par exemple la fonction à trois variables $f(A_2, A_1, A_0)$ définie par la table suivante :

Entrées			*Sortie*
A_2	A_1	A_0	$f(A_2, A_1, A_0)$
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

La table de vérité montre que *f* vaut *1* pour les combinaisons de variables d'entrée : *001*, *011*, *101* et *110*, *f* vaut *0* pour toutes les autres combinaisons. Afin de mettre en œuvre cette fonction avec le sélecteur de données, l'entrée sélectionnée pour chacune des combinaisons précédentes doit être connectée à un niveau haut. Toutes les autres entrées doivent être connectées à un niveau bas.



Remarque

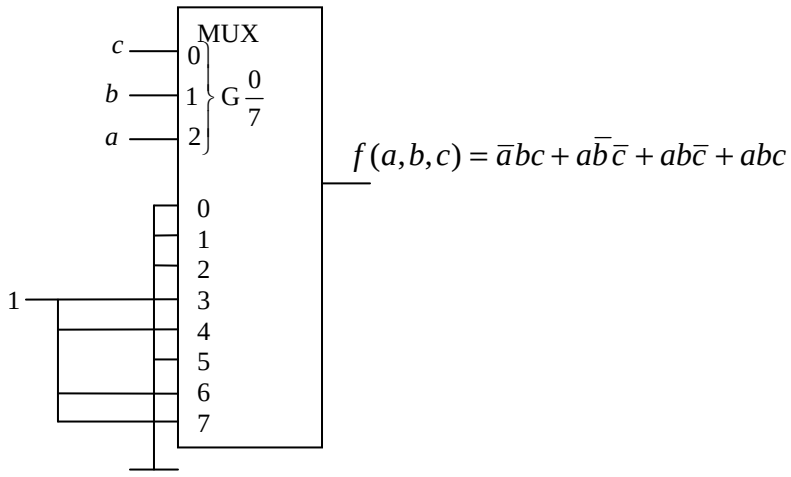
En général, toute fonction à *n* variables peut être réalisée avec un multiplexeur à 2^{n-1} entrées. Pour une telle fonction, *(n-1)* variables sont reliées aux lignes d'adresses, les lignes de données sont commandées par la variable restante ou son complément ou 0 ou 1.

Exemple 2

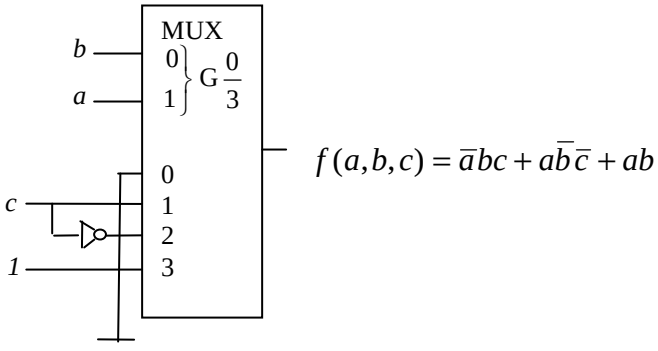
Soit la fonction logique *f* définie par sa table de Karnaugh suivante :

c \ *ab*	00	01	11	10
0	*0*	*0*	*1*	*1*
1	*0*	*1*	*1*	*0*

1. Réalisation de *f* par un MUX8 → 1.



2. Réalisation de *f* par un MUX4 → 1.

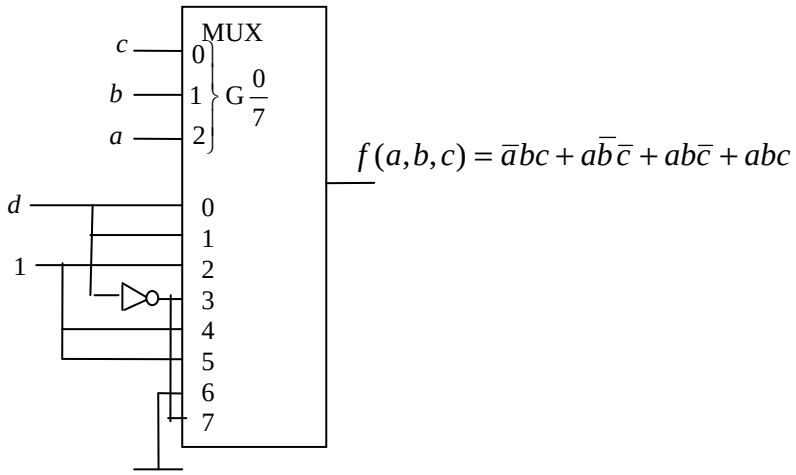


Exemple 3

Soit la fonction logique *f* définie par sa table de Karnaugh suivante :

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

Réalisons la fonction *f* par un MUX8 → 1.



4.3. Démultiplexeurs

Un démultiplexeur (DEMUX) effectue la fonction inverse d'un multiplexeur ; donc c'est un circuit logique ayant une seule entrée de donnée, *n* entrées de sélection et 2^n sorties. L'entrée de donnée est reliée à la sortie désirée par l'adresse indiquée par les entrées de sélection. On l'appelle aussi distributeur de données.

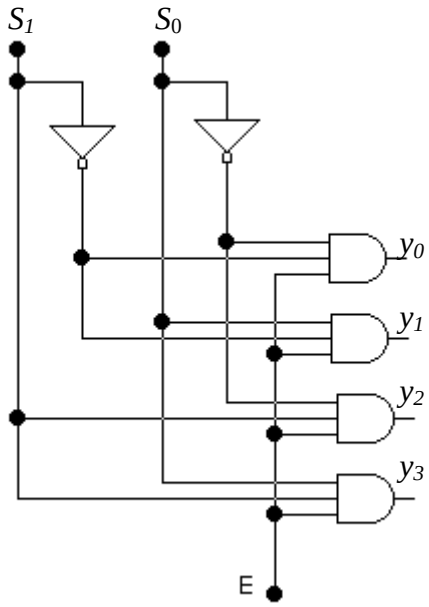
Exemple : DEMUX à 4 sorties

Faisant la synthèse d'un DEMUX à 4 sorties : y_0, y_1, y_2, y_3, une entrée *E* et deux entrées de sélection : S_1, S_2

- Table de vérité

S_1	S_0	y_0	y_1	y_2	y_3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

- Logigramme



DEMUX à 4 sorties

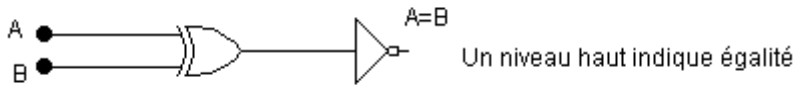
La ligne d'entrée est connectée à toutes les portes ET. Les deux lignes de sélection de données valident une seule porte à la fois. Les données de la ligne d'entrée traversent la porte sélectionnée jusqu'à sa ligne de sortie.

4.4. Comparateurs

4.4.1. Egalité

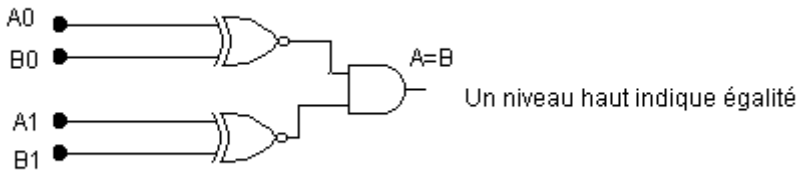
Le ou-exclusif peut servir de comparateur de base, puisque sa sortie vaut 1 si les deux bits à ses entrées sont différents :

A	B	$A \oplus B$	$\overline{A \oplus B}$
0	*0*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*1*

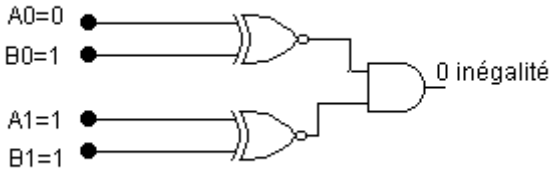


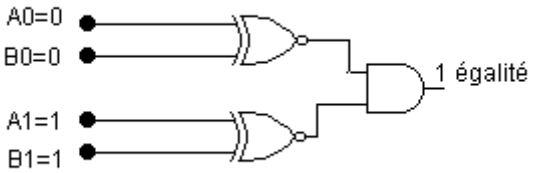
Exemple

Comparateur de deux nombres binaires de deux bits chacun $A=A_1\ A_0$ et $B=B_1\ B_0$. Pour comparer les nombres A et B il faut utiliser une porte ou exclusif additionnelle ; une pour comparer les bits du poids le plus faible et l'autre pour comparer les bits du poids le plus fort. Pour produire une sortie simple indiquant une condition d'égalité ou d'inégalité, on peut utiliser des inverseurs et une porte logique ET.



Exemple



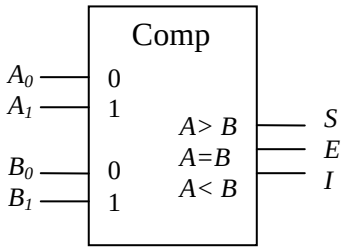


4.4.2. Inégalité

Un comparateur à *n* bits est un circuit capable de comparer deux nombres binaires à *n* bits chacun : $A=A_{n-1}\ A_{n-2}\ldots A_1\ A_0$ et $B=B_{n-1}\ B_{n-2}\ldots B_1\ B_0$. Il a trois sorties *S*, *E* et *I* tel que ; *S=1* ssi *A> B*, *E=1* ssi *A=B* et *I=1* ssi *A< B*.

Exemple

Comparateur à deux bits : Soient les deux nombres binaires $A=A_1\ A_0$ et $B=B_1\ B_0$.



Symbole logique d'un comparateur à 2 bits

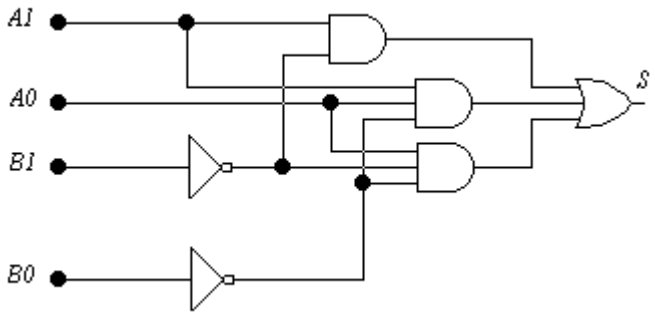
- Table de Karnaugh générale.

$B_1\ B_0$ \ $A_1\ A_0$	00	01	11	10
00	*E*	*S*	*S*	*S*
01	*I*	*E*	*S*	*S*
11	*I*	*I*	*E*	*I*
10	*I*	*I*	*S*	*E*

- Table de Karnaugh de *S*

$B_1 B_0$ \ $A_1 A_0$	00	01	11	10
00		1	1	1
01			1	1
11				
10			1	

$$S = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

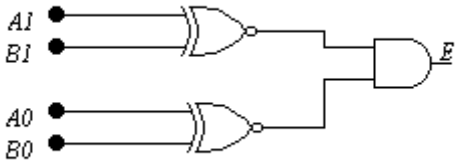


Fonction S

- Table de Karnaugh de *E*

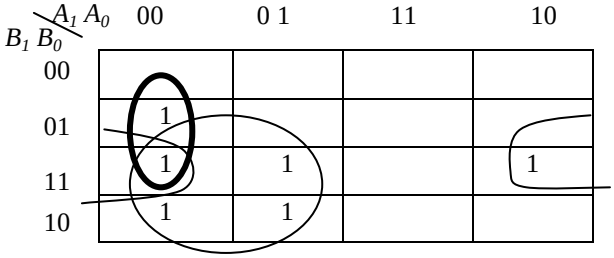
$B_1 B_0$ \ $A_1 A_0$	00	01	11	10
00	1			
01		1		
11			1	
10				1

$$E = \overline{A}_1\overline{A}_0\overline{B}_1\overline{B}_0 + \overline{A}_1A_0\overline{B}_1B_0 + A_1A_0B_1B_0 + A_1\overline{A}_0B_1\overline{B}_0$$
$$= \overline{(A_1 \oplus B_1)}.\overline{(A_0 \oplus B_0)}$$

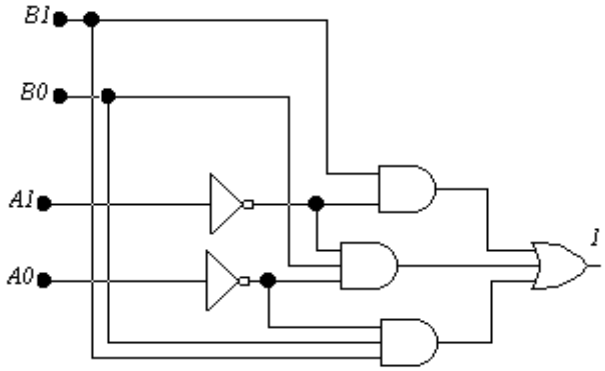


Fonction E

- Table de Karnaugh de *I*



$$I = \overline{A}_1 B_1 + \overline{A}_1 \overline{A}_0 B_0 + \overline{A}_0 B_1 B_0$$

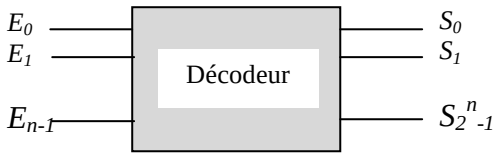


Fonction I

4.5. Décodeurs

Un décodeur permet d'identifier quelle combinaison est active ; sa fonction consiste à faire correspondre à un code présent en entrée sur *n* bits une seule sortie active (=1) parmi plusieurs sorties possibles. Toutes les autres sorties sont inactives.

Par exemple, un décodeur à n bits est un circuit logique ayant *n* entrées et 2^n sorties.



Décodeur à *n* entrées

Quant le code prend la valeur *i* la sortie S_i est activée.

Exemple : Décodeur 1 parmi 4

Ce décodeur dispose de deux entrées *a* et *b* et quatre sorties où chacune correspond à une des quatre combinaisons de ces variables d'entrées.

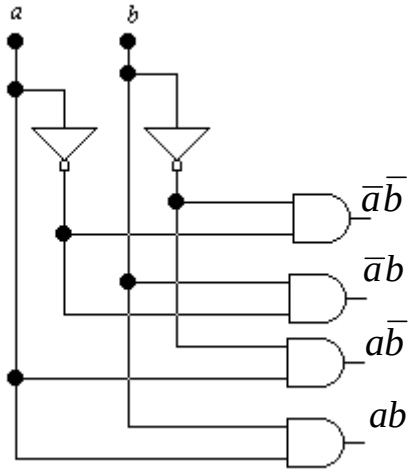
- Table de vérité

a	b	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- Expressions logiques

  $S_0 = \overline{a}\overline{b};\ S_1 = \overline{a}b; S_2 = a\overline{b}\ ; S_3 = ab.$

- Logigramme

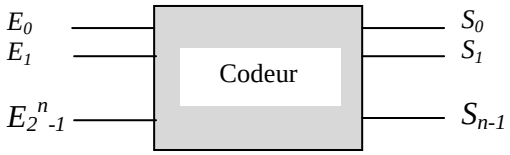


Décodeur 1 parmi 4

4.6. Codeurs

Le codeur réalise l'opération inverse d'un décodeur, il génère un code de sortie différent pour chaque signal d'entrée qui est sélectionné c'est à dire, à une entrée active (=1), parmi plusieurs entrées il fait correspondre un code en sortie. Par exemple : décimal ou octal convertit en BCD ou binaire.

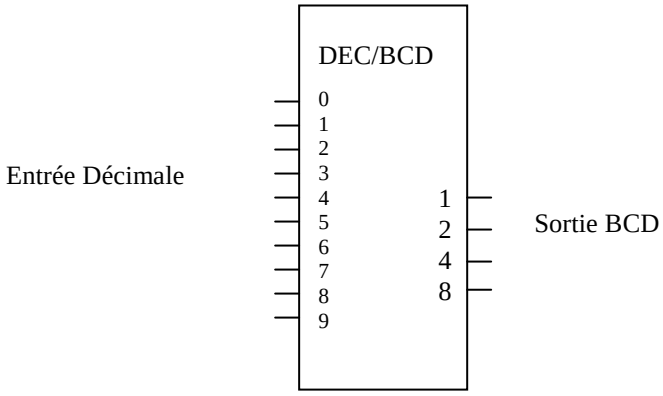
Par exemple, un codeur à *n* bits est un circuit logique ayant 2^n entrées et *n* sorties.



Codeur binaire à 2^n entrées et *n* sorties.

Exemple 1 : Codeur Décimal – BCD

Ce codeur a 10 entrée et 4 sorties qui correspondent au code BCD.



Symbole logique d'un codeur DEC/BCD

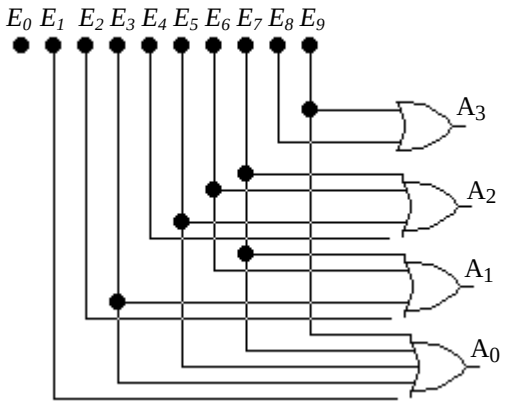
- Table de vérité

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	A_3	A_2	A_1	A_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

- Expressions logiques

$$A_3=E_8+E_9\ ;$$
$$A_2=E_4+E_5+E_6+E_7\ ;$$
$$A_1=E_2+E_3+E_6+E_7\ ;$$
$$A_0=E_1+E_3+E_5+E_7+E_9.$$

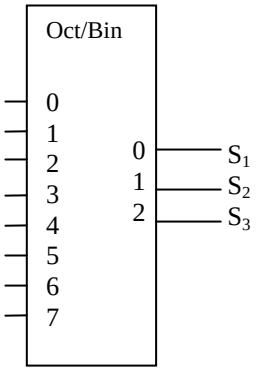
- Logigramme



Codeur DEC/BCD

Exemple 2

Réalisons un codeur exprimant les digits 0 à 7 du système octal en leurs équivalents binaires.



Symbole logique d'un codeur Oct/Bin

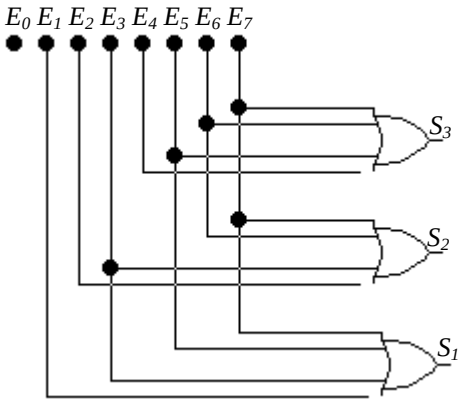
- Table de vérité

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	S_3	S_2	S_1
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- Expressions logiques

$$S_1=E_1+E_3+E_5+E_7\ ;$$
$$S_2=E_2+E_3+E_6+E_7\ ;$$
$$S_3=E_4+E_5+E_6+E_7.$$

- Logigramme



Codeur Oct/Bin