Université Mohammed V- Rabat-
Faculté des Sciences
Département de Physique

Electronique numérique

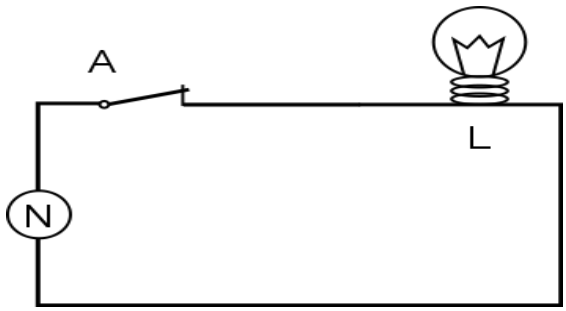
Introduction générale

Les fonctions logiques (dites Booléennes de Georges Boole, 1815-1864, savant anglais qui mis en équation la logique Aristotélicienne.) sont utilisées dans les expressions Booléennes des langages de programmation C ou Pascal, …, dans les systèmes automatiques à deux états (machines à outils, instruments électroniques…), et dans l'architecture des ordinateurs (mémoires, Entrées/Sorties, UAL). La matérialisation de ces fonctions logiques se fait à l'aide de circuits appelés **circuits numériques ou logigramme.**

Il existe une correspondance terme à terme entre les circuits électroniques numériques et les fonctions logiques. Pour n'importe quelle fonction Booléenne (qui correspond à un cahier de charges bien définie) on peut concevoir un circuit électronique et vice versa. Chaque fonction Booléenne demande seulement des opérateurs de base en logique : ET, OU, et NON, ……on peut donc construire n'importe quel circuit utilisant exclusivement ces opérateurs de base. Les fonctions Booléennes ET, OU, NON … etc correspondent aux circuits électroniques appelés respectivement : Portes ET, OU, NON…etc.

Codage binaire des informations

Le circuit ci-dessous peut être utilisé pour matérialiser un bit binaire.

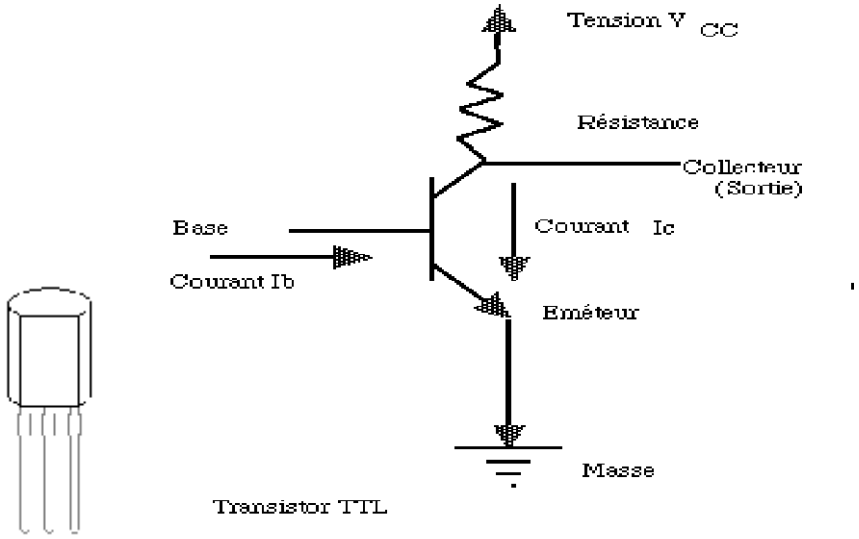


A est ouvert $\Rightarrow$ 1 logique ; L est éteinte $\Rightarrow$ 0 Logique

A est fermé $\Rightarrow$ 0 logique ; L est allumée $\Rightarrow$ 1 Logique

La couche physique d'un ordinateur repose sur le fonctionnement d'un composant électronique appelé transistor fonctionnant de la même manière que l'interrupteur « A ». Deux grandes technologies sont utilisées pour la fabrication des transistors : technologie bipolaire (appelée également TTL) et la technologie MOS (**M**étal **O**xyde **S**ilicium)

Exemple avec la technologie bipolaire



Quand La tension entre la Base et l'Emetteur (VBE) est à 5 V, la tension de sortie (Collecteur – Emetteur) VCE est à 0,3 V (~0V) et le courant I_C est non nul. Si la tension VBE est à 0V, la tension de sortie VCE est à 5 V et le courant I_C est nul.

Dans la logique positive :

0V ⇒ 0 Logique

5 V ⇒ 1 Logique

Dans la logique négative 0V (1 Logique) et 5V (0 Logique)

Un transistor peut donc manipuler deux valeurs ou états logiques : "0" logique et "1" logique.

Etat logique de l'entrée	Etat logique de la sortie
0	1
1	0

Ces deux valeurs peuvent être nommées de différentes façons :

Niveau logique "1"

Tout, Vrai, Fermé, Marche, Haut, Allumé, Oui … etc.

Niveau logique "0"

Rien, Faux, Ouvert, Arrêt, Bas, Éteint, Non … etc.

Ce qui permet l'utilisation de la base 2 (Base binaire) dont les caractères sont appelés binary digit abrégé par **bit**. Un bit peut donc stocker "0" logique ou "1" logique. Il peut coder deux

informations. Pour élargir le spectre de codage, il faudra augmenter le nombre de bits. Ceci a donné naissance à la notion de Byte ou d'octets (8 Bits) et à la notion de mots.

8 bits = 1 octet
16 bits = 2 octets 1 mot
32 bits = 4 octets double mot

Avec n bits on peut coder 2^n informations. Ainsi, le codage des instructions, des données numériques ou caractères est effectué par un ensemble de bits.

Le format général de codage est sous la forme :

2^{n-1}	2^{n-2}			2^2	2^1	2^0			
bn-1	bn-2	....				....	b2	b1	bo

BPS (**B**it le **P**lus **S**ignificatif)

BMS (**B**it le **M**oins **S**ignificatif)

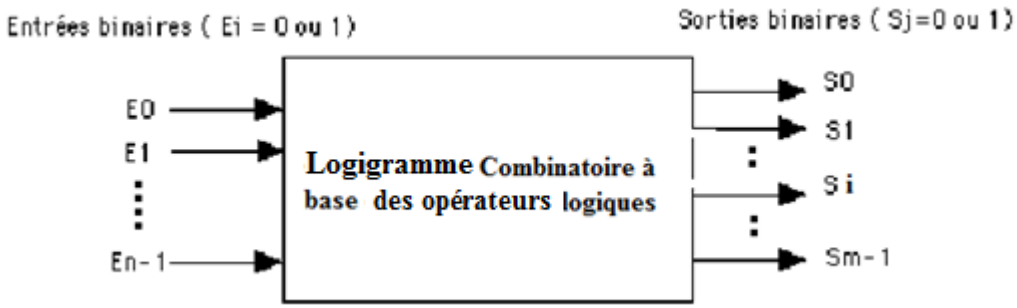
Le 1er bit est appelé **B**it le **M**oins **S**ignificatif : **BPS** ou **L**east **S**ignificatif **B**yte: **LSB**
Le dernier bit est appelé **B**it le **P**lus **S**ignificatif : **BPS** ou **M**ost **S**ignificatif **B**yte: **MSB**
Le 1er octet est appelé **octet faible** ou octet le **moins significatif**
Le dernier octet est appelé **octet fort** ou octet le **plus significatif**

Chapitre 1
Logique combinatoire

1.1 Définition d'une fonction logique

Une variable logique (ou variable binaire) ne peut prendre que deux valeurs distinctes : 0 ou 1. Elle peut représenter n'importe quel dispositif ou événement binaire appelé également tout ou rien : Contact (Ouvert/fermé), lampe (allumée éteinte), moteur (marche/arrêt), ... etc. ***Une variable binaire est codée par un seul bit.***

Une fonction logique est une fonction d'une ou plusieurs variables logiques. Ces variables sont appelées variables d'entrées. Ce groupe de variables sont reliées entre-elles par des opérateurs logiques ET, OU, NON, …etc. Un système logique est dit **combinatoire** lorsque l'état de la sortie est uniquement défini par la combinaison des états logiques des variables d'entrées **indépendamment du temps**. Sur le plan matériel, une fonction logique combinatoire est réalisée à l'aide d'un circuit logique ou logigramme combinatoire.



La Sortie **Si** est une fonction logique de n variables binaires définie par:

$E_0, E_1, ...E_{n-1}$	⟶	Si ($E_0, E_1, ...E_{n-1}$)

Représentation d'une fonction logique combinatoire:

Une fonction logique combinatoire de n variables binaires est définie par un cahier de charges permettant de la définir et ensuite de la représenter. Cette représentation est une compilation sous la forme d'une table de tous les états logiques de la sortie en fonction des états logiques des entrées. Plusieurs représentations sont utilisées parmi lesquelles on cite les représentations tabulaires : Table de vérité classique, table de Veitch, table de Karnaugh, table de Venn , table Johnston et de Caroll. Il est à noter que d'autres représentations sont utilisées : les représentations implicites et les représentations graphiques.

Une **table de vérité** est constituée de 2^n lignes. Chaque ligne de cette table représente une configuration binaire des entrées (E0, E1, ..., En-1) et les états de sorties correspondantes. Cette fonction est calculée pour chaque configuration ou état des entrées Ei.

Table de vérité

E0	E1	....	...	...	...	...	En-1	S0	S1	...	...	...	....	Sm-1
0	0	0	...	...	..	0	0	Si(0,0,0,.....,0) i de 0 à m-1						
0	0	0	...	...	..	0	1	Si(0,0,0,.....1)						
.							.	.						
.							.	.						
.							.	.						
1	1	1	..	..	..	1	1	Si(1,1,1,.....,1)						

A chaque ligne ou état binaire des entrées (E0, E1,,E2,....En-1) correspond une configuration ou état de sortie unique.

Le principal inconvénient de la table de vérité est qu'elle devient rapidement très encombrante lorsque le nombre de variables d'entrée augmente. Pour résoudre ce problème on peut utiliser d'autres types de représentations tabulaires telles que : Tables de Veitch, de Karnaugh **(très populaire)**, de Venn , Johnston et de Caroll

Notation :

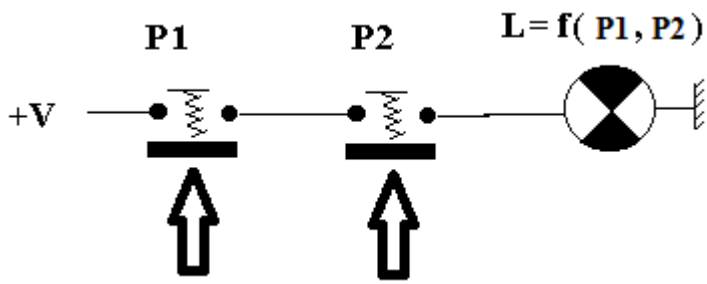
Par convention, une variable logique entrée ou une sortie active à un niveau haut (état logique 1) sera notée par exemple par A, E, MEM, ..etc : **Variables non complémentées**

Une entrée ou une sortie logique active à un niveau bas (état logique 0) sera notée par exemple par $\overline{C}$, $\overline{D}$, , $\overline{READY}$ …, etc. : **Variables complémentées (représentée en complément à 1)**

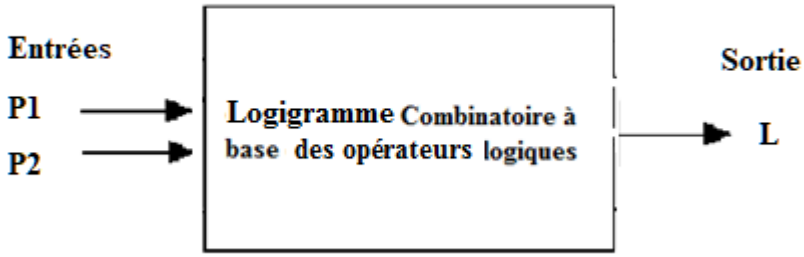
Exemple : Boutons poussoirs n = 2 ; une table de vérité constituée de 2^2 lignes

Cahier des charges :

deux entrées (2 variables logiques) et 1 seule sortie (1 variable logique) :



La variable logique de sortie **L** (état de la lampe : Allumée "1" ou éteinte "0") est une fonction logique des variables logiques d'entrées P1 et P2 liées à deux boutons poussoirs.



- ✓ **Entrées: deux boutons poussoirs P1 et P2**

Codage des états logiques des 2 entrées:

P1 ou P2 = 1 ⇒ Bouton enfoncé

P1 ou P2 = 0 ⇒ Bouton relaché

On a deux variables → n = 2 ; une table de vérité constituée de 2^2 lignes: 4 états logiques des 2 entrées

- ✓ **Sortie: la Lampe L**

Codage de l'état de la lampe : Allumée "1" ou éteinte "0

L est une fonction logique (une seule sortie) des 2 variables logiques P1 et P2 représentant les deux boutons poussoirs.

Analyse :

P1 ou P2 est ouvert ⇒ L = "0"

P1 et P2 sont ouverts ⇒ L = "0"

P1 et P2 sont fermés ⇒ L = "1"

Table de vérité

P1	P2	L= f(P1,P2)
0	0	0
1	0	0
0	1	0
1	1	1*

1.2 Forme canonique d'une fonction logique

Une fonction logique sera exprimée par les variables logiques d'entrées et par les opérateurs logiques:

ET logique : Produit logique noté par « **.** »
OU logique : Somme logique noté par « + »
NON logique : Inversion ou Complément à un noté par « ¯ » ……etc

On obtient alors sa forme canonique ou algébrique.

Une fonction logique est sous sa forme canonique si toutes les variables d'entrées apparaissent dans chacun de ses termes. Lorsque l'équation logique de la fonction est écrite à partir de sa table de vérité, elle est dans sa forme canonique.

Définitions

On appel **minterme** de n variables un produit logique (**.**) de ces variables (complémentées ou non). Avec n variables on construit 2^n mintermes (2^n possibilités binaires).

Exemple : Pour 2 variables (A, B), n = 2
on aura 4 mintermes : $\overline{A}.\overline{B}$, $\overline{A}.B$, $A.\overline{B}$ et $A.B$

On appel **maxterme** de n variables une somme logique (+) de ces variables (complémentées ou non). Avec n variables on construit 2^n maxtermes (2^n possibilités binaires).

Exemple : Pour 2 variables (A, B), n = 2
on aura 4 maxtermes : $\overline{A} + \overline{B}$, $\overline{A} + B$, $A + \overline{B}$ et $A + B$

L'extraction de la forme canonique ou algébrique de la fonction logique à partir d'une table de vérité :

On établit l'équation logique d'une fonction en sommant les **mintermes** pour lesquels la fonction logique est égale à "1". On obtient alors l'expression logique sous sa première forme canonique (forme normale disjonctive).
La deuxième forme canonique (forme normale conjonctive) d'une fonction logique est composée exclusivement d'un produit de maxtermes pour lesquels la fonction logique est égale à "0".

Les deux formes canoniques sont équivalentes

Une fonction logique peut être donc représentée soit par une forme canonique (représentation algébrique) ou par sa table de vérité (représentation tabulaire).

Exemples

- Boutons poussoirs

Table de vérité

P1	P2	L= f(P1,P2)
0	0	0
1	0	0
0	1	0
1	1	1*

1ère Forme canonique Somme des mintermes (forme disjonctive)

$$L = f(P1, P2) = P1.P2$$

2ème Forme canonique Produit des maxtermes (forme conjonctive)

$$L = f(P1, P2) = (P1+P2)(P1+\overline{P2})(\overline{P1}+P2)$$

- Fonction majoritaire : Soit une fonction logique f de 3 variables A,B,C (n = 3) définie par:

f(A,B,C) = 1 **Si** la majorité des variables d'entrées sont égales à 1
0 **Si** la majorité des variables d'entrées sont égales à 0

Table de vérité

A	B	C	f
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*

On utilisera la 1ère Forme canonique ⟶ Somme des mintermes

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

On remarque que pour matérialiser cette fonction on aura besoin des opérateurs ET (.) des opérateurs OU (+) et des opérateurs NON.

1.3 Opérateurs logique élémentaires

a) ***Opérateurs logiques*** (portes logiques)

Les fonctions logiques sont conçues à partir d'un groupe d'opérateurs élémentaires appelés « portes logiques». Chaque opérateur (porte logique) est représenté par un symbole et sa fonction logique (définie par la table de vérité).

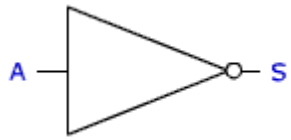
- ***Porte NON (NOT)***

Lorsque l'entrée A , figures ci-dessous, est au niveau logique haut ("1"), la sortie S est au niveau logique bas ("0") et inversement : A = "0" ⇒ S = "1". Ce fonctionnement est appelé ***Inversion logique*** ("0" ⇒ "1" et "1" ⇒ "0"), ⇒ une porte inverseuse élémentaire ou porte ***NON (NOT)***

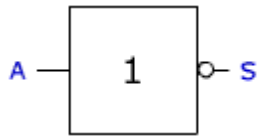
Table de vérité

Entrée	Sortie
A	S
0	1*
1	0

Symbole Américain



Symbole Européen



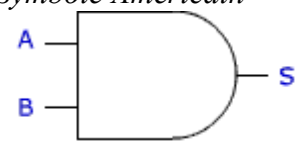
S= $\overline{A}$ (Inversion logique)

- ***Porte ET (AND)***

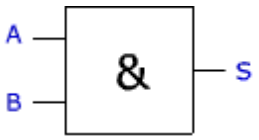
Table de vérité

Entrées		Sortie
A	B	S
0	0	0
1	0	0
0	1	0
1	1	1*

Symbole Américain



Symbole Européen



La sortie est à l'état haut "1" seulement quand A= 1 **et** B =1 : S = A . B (produit logique)

- ***Porte OU (OR)***

Table de vérité

Entrées	Sorties

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

Symbole Américain *Symbole Européen*

La sortie S est à l'état haut "1" si A =1 **ou** B =1 : S = A + B (somme logique).

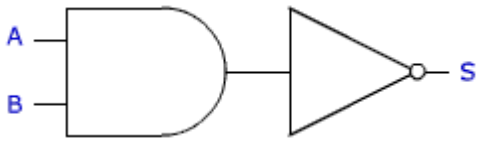
Pour réaliser les inversions des portes ET et OU on ajoute à leurs sorties une porte NON :

- ***Porte NON-ET (NAND)***

Symbole Américain *Symbole Européen*

$S = \overline{A.B}$

est équivalent à :

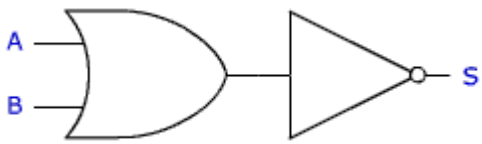


- ***Porte NON-OU (NOR)***

Symbole Américain *Symbole Européen*

$S = \overline{A + B}$

est équivalent à :



En plus des portes de base NON, ET, OU, on trouve les portes suivantes:

- ***Fonction OU exclusive (Exclusive OR : XOR)***

Table de vérité

A	B	$A \oplus B$
0	0	0
0	1	1*
1	0	1*
1	1	0

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

Symbole Américain *Symbole Européen*

Si A = B alors S = 0, Si A ≠ B alors S = 1 : S = $A \oplus B$ (Comparaison logique)

- ***Fonction NON OU exclusive (XNOR).***

Table de vérité

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

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

Symbole Américain *Symbole Européen*

Si A = B alors S = 1, Si A ≠ B alors S = 0 : S = $\overline{A \oplus B}$ (Comparaison logique)

Application des portes XOR et XNOR

- Comparaison
- Chiffrement (Cryptographie)
- Opérations arithmétiques
- etc

Opérateurs logiques en C

- **Opérateurs pour les tests logiques**

Opérateur	Signification	Exemple
&& || !	ET logique OU logique NON logique (négation)	x && y x || y ! x

- **Opérateurs Bit à bit**

Opérateur	Signification	Exemple
& ^ | ~ << >>	ET OU exclusif OU inclusif Négation Décalage de n bits à droite Décalage de n bits à gauche	x & y x ^ y x | y ~ x x <<n x >>n

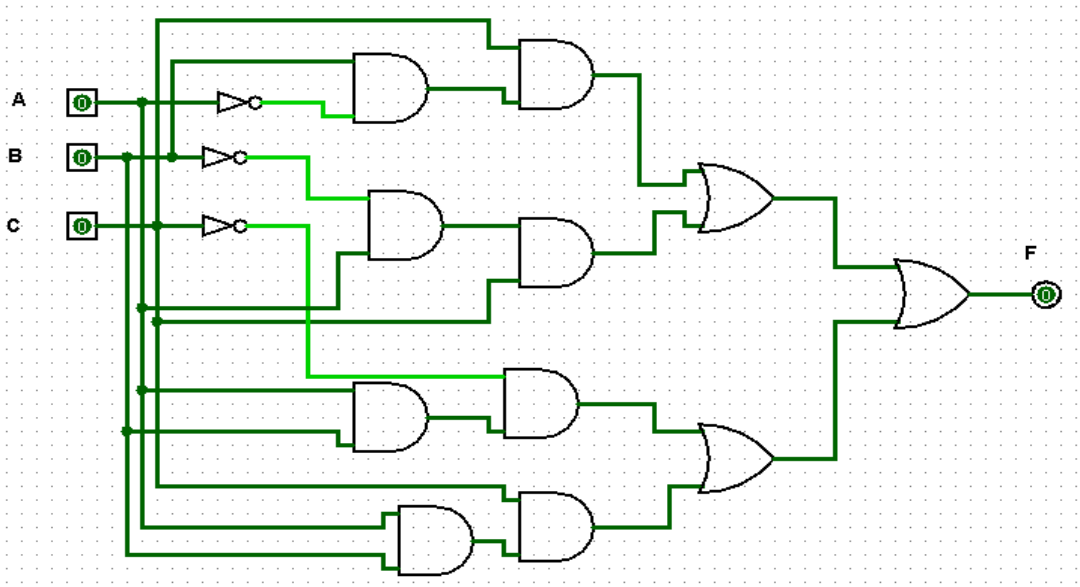
b) Logigramme

Un logigramme permet de matérialiser une fonction logique en associant des portes logiques.

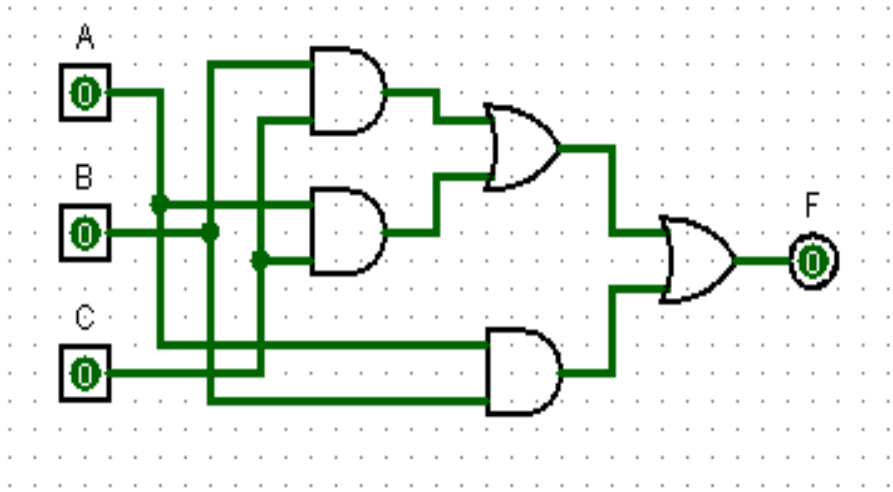
Exemple : Fonction majoritaire

A	B	C	F
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

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

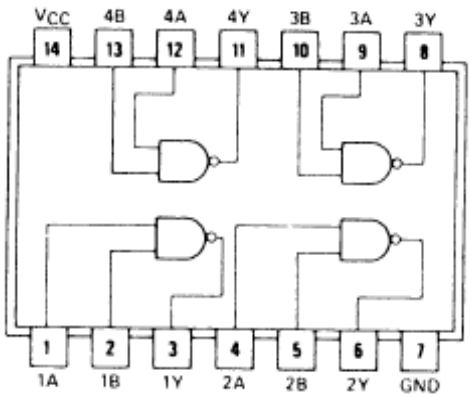


Après simplification :

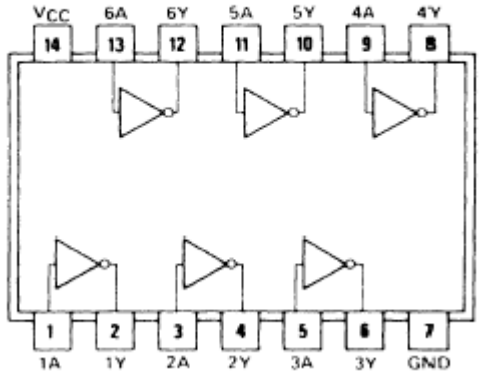


Exemples de quelques circuits intégrés série 74 :

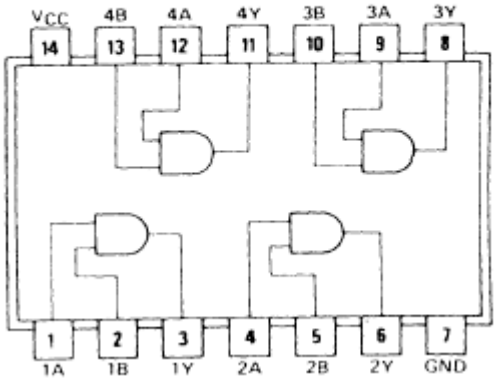
NAND (7400)



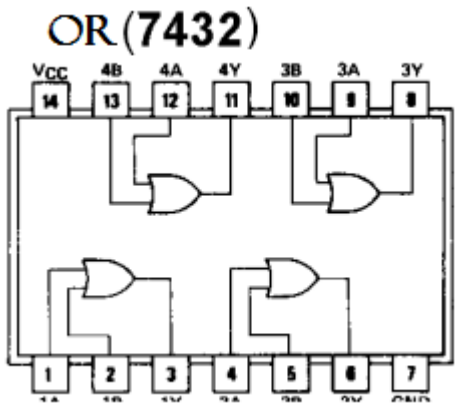
NOT (7404)



AND(7408)



OR(7432)



1.4 Algèbre de Boole

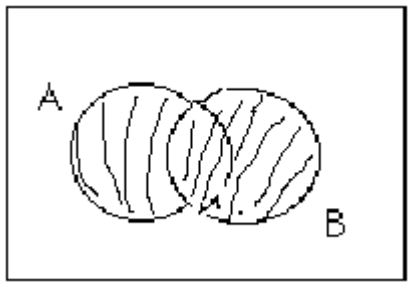
Le Mathématicien Anglais G. Boole développa en 1847 la théorie des ensembles. L'algèbre booléenne définit un cadre mathématique d'étude de propositions logiques portant sur des ensembles. En utilisant la Base 2, l'algèbre de Boole s'applique non plus à des ensembles mais à des variables logiques qui ne peuvent prendre que l'état logique "1" ou "0". L'algèbre de Boole est appelé également une algèbre binaire ou algèbre de commutation. De ce fait, les variables et fonctions logiques sont appelées booléennes.

L'algèbre de Boole permet de concevoir des circuits logiques qui matérialisent les fonctions logiques.

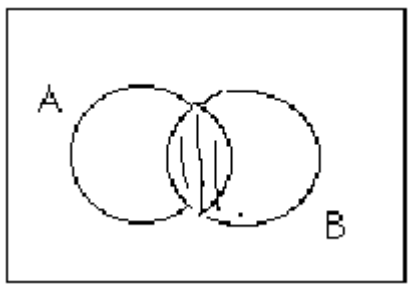
L'algèbre de Boole dans la base 2 est constituée :

1. d'un ensemble E,
2. de deux éléments particuliers de E : 0 et 1,
3. de deux opérations binaires sur E : + et · respectivement au OU logique (UNION) et ET logique (INTERSECTION) :

A U B est l'ensemble des éléments appartenant à A ou à B qu'on note A+B (somme logique), les éléments de cet ensemble sont appelés **maxtermes**.

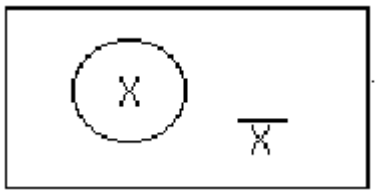


A ∩ B est l'ensemble des éléments appartenant à la fois à A et à B qu'on note A . B (produit logique), les éléments de cet ensemble sont appelés **mintermes**.



4. d'une opération unaire sur E : $\overline{\ }$ (correspondant à la négation logique : complément à 1).

Soit X un ensemble fini, un élément donné appartient soit à X soit à l'extérieur de X mais jamais aux deux simultanément. On dit que $\overline{X}$ (Non X) est le complémentaire de X.



Théorèmes de l'algèbre de Boole :

- **Théorèmes monovariable**

Identité :

À chaque opérateur correspond un élément neutre qui, lorsqu'il est opéré avec une variable quelconque A, donne un résultat identique à cette variable.

$$A + 0 = A, \qquad A \cdot 1 = A$$

Élément nul :

À chaque opérateur correspond un élément nul qui, lorsqu'il est opéré avec une variable quelconque A, donne un résultat identique à cet élément nul.

$$A + 1 = 1, \qquad A \cdot 0 = 0$$

Idempotence:

Le résultat d'une opération entre une variable A et elle-même est égal à cette variable.

$$A \cdot A = A$$
$$A + A = A$$

Complémentation :

$$A \cdot \overline{A} = 0 \qquad A + \overline{A} = 1$$

Involution :

Le complément du complément d'une variable A est égal à cette variable

$$\overline{\overline{A}} = A$$

- **Théorèmes multivariable**

Commutativité :

$$A.B = B.A , \qquad A + B = B + A$$

Associativité :

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

Distributivité:

$$A.(B + C) = A.B + A.C$$
$$(A + B).(A + C) = A.A + AC + BA + BC = A + B.C$$

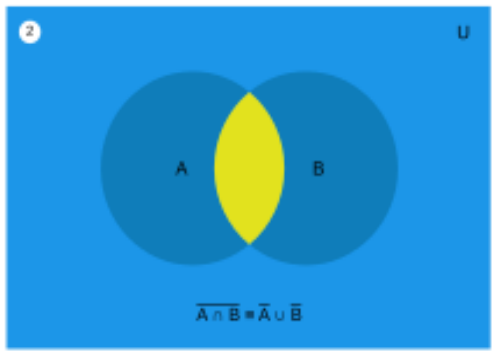
Absorption:

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

Ce théorème est particulièrement intéressant pour la conception de circuits numériques puisqu'il permet d'éliminer les termes inutiles et par conséquent de réduire la complexité d'un circuit logique.

Théorèmes de DE MORGAN

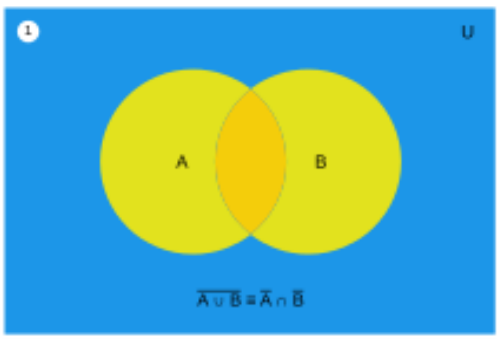
a)



Cette figure montre que:

$$\overline{A.B} = \overline{A} + \overline{B} \qquad (1)$$

b)



Cette figure montre que:

$$\overline{A + B} = \overline{A} \,.\, \overline{B} \qquad (2)$$

Ces règles peuvent être appliquées à des produits logiques ou des sommes logiques comportant un nombre quelconque de variables.

$$\overline{X1.X2.X3\ldots\ldots Xn} = \overline{X1} + \overline{X2} + \overline{X3} + \ldots\ldots + \overline{Xn}$$

$$\overline{X1 + X2 + X3 + \ldots\ldots + Xn} = \overline{X1} \,.\, \overline{X2} \,.\, \overline{X3} \,.. \ldots\ldots \overline{Xn}$$

1.5 Simplification des fonctions logiques

Pour simplifier l'expression canonique de la fonction logique et réduire par conséquence le nombre de variables d'entrée ou le nombre de termes (minterme ou maxterme) et par la suite le nombre de portes d'un circuit logique (diminuer le coût du circuit), on utilise soit des méthodes algébriques, graphiques ou algorithmiques. Les méthodes algébriques sont basées

sur l'algèbre de Boole et DEMORGAN, les méthodes graphiques sont basées sur les diagrammes suivants :

Karnaugh (traité dans ce cours), Venn, Johnston et Caroll et sur les méthodes algorithmiques suivantes: Quine/McCluskey, Petrick ou d' Espresso,

a) Simplification algébrique

Les théorèmes de Boole et de DE MORGAN sont utilisés pour simplifier la forme canonique d'une fonction logique

Exemple

$f1 = \overline{A}\,B + A\,\overline{B} + AB = B(\overline{A} + A) + A\,\overline{B} = B + A\,\overline{B} = \mathbf{A + B}$

$f2 = \overline{A}\,BC + A\,\overline{B}\,C + AB\,\overline{C} + ABC$

$= C(\overline{A}\,B + A\,\overline{B}) + AB(C + \overline{C})$

$= C(\overline{A}\,B + A\,\overline{B}) + AB = \mathbf{C(A \oplus B) + AB}$

$f3 = A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + \overline{A}\,BC + A\,\overline{B}\,C + \overline{A}\,\overline{B}\,\overline{C}$

(1) (2) (3) (3) (2) (1)

$= \overline{B}\,\overline{C} + \overline{B}\,C + \overline{A}\,B = \overline{B} + \overline{A}\,\mathbf{B} = \overline{A} + \overline{B}$

(1) (2)

Si le nombre de variables est supérieur à 4 la simplification algébrique devient difficile par exemple

$F4 = A\,\overline{B}\,\overline{C}\,DE + \overline{A}\,\overline{B}\,CDE + \overline{A}\,B\,\overline{C}\,D\,\overline{E} + \overline{A}\,BCDE + A\,\overline{B}\,CD\,\overline{E} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}$

$+ ABCDE + ABCD\,\overline{E} + AB\,\overline{C}\,DE + ABC\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}\,\overline{C}\,DE + A\,\overline{B}\,C\,\overline{D}\,\overline{E}$

b) Méthodes de simplification graphiques (Diagramme de Karnaugh)

La table de vérité d'une fonction logique de n variables est constituée de 2^n lignes. C'est une représentation linéaire. Une autre manière de représenter cette fonction est appelée: **table ou diagramme de Karnaugh**. Dans cette table, la fonction logique est représentée sous forme bidimensionnelle dans laquelle chaque ligne de la table de vérité correspond à une case dans la table de Karnaugh. Ainsi, dans cette table il y aura 2^n cases.

Principe de la méthode.

Chaque case de la table de Karnaugh représente l'état de la fonction logique "1" ou "0". Lorsqu'on parcourt la table de Karnaugh, de gauche à droite ou de haut en bas une et une seule variable doit changer d'état lors du passage **d'une colonne à une colonne adjacente ou d'une ligne à une ligne adjacente**. C'est pour cela que nous aurons, dans le cas de 2 variables par exemple, la succession 00 01 11 10 pour indiquer les 2^2 possibilités binaires des deux variables et elle indique également numéros des colonnes et des lignes. Ce qui correspond au codage par le code binaire de Grey (Codage réfléchi) au lieu du code binaire naturel.

Exemples de tables ***de Karnaugh***

n = 1, une seule variable

A	**S**
0	**1***
1	**0**

$\overline{A}$ A

1	**0**

$S = \overline{A}$

n = 2, deux variables

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

A

B / A	**0**	**1**
0	**0**	**1**
1	**1**	**0**

B

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

n = 3, trois variables

A	**B**	**C**	**S**
0	**0**	**0**	**1***
0	**0**	**1**	**1***
0	**1**	**0**	**0**
0	**1**	**1**	**1***
1	**0**	**0**	**0**
1	**0**	**1**	**1***
1	**1**	**0**	**0**
1	**1**	**1**	**0**

$S = \overline{A}\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + \overline{A}\ BC + A\ \overline{B}\ C$

A

B ----------

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

C

n = 4, quatre variables

A	B	C	D	S
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**1***
0	**0**	**1**	**0**	**1***
0	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**1***
0	**1**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**
1	**0**	**1**	**0**	**0**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**0**	**1***
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**0**	**0**
1	**1**	**1**	**1**	**0**

$$S= \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + AB\,\overline{C}\,\overline{D}$$

A

B ----------

D \ C \ CD/ AB	00	01	11	10
00	**0**	**1**	**1**	**0**
01	**1**	**0**	**0**	**0**
11	**0**	**0**	**0**	**0**
10	**1**	**0**	**0**	**0**

Remarque

La ligne supérieure est adjacente à la ligne inférieure de même pour les colonnes gauche et droite (une seule variable change d'état).

Simplification des fonctions logiques.

Le principe de la méthode de simplification se base sur les théorèmes de Boole: $A+ \overline{A} = 1$, $C + A\,\overline{C} = A + C$, etc. La méthode de Karnaugh permet d'automatiser la méthode algébrique.

Exemple n = 4,

A	B	C	D	S
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**
0	**0**	**1**	**0**	**0**
0	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**
0	**1**	**0**	**1**	**1***
0	**1**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**
1	**0**	**1**	**0**	**0**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**0**	**0**
1	**1**	**0**	**1**	**1***
1	**1**	**1**	**0**	**0**
1	**1**	**1**	**1**	**0**

Table de Karnaugh

A

B ----------

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

D

C

F= $\overline{A}\,B\,\overline{C}\,D + AB\,\overline{C}\,D$ (forme canonique).

Deux cases adjacentes peuvent être considérées comme un seul minterme, obtenu en ne gardant que les variables ayant le même état dans les deux cases.
Dans la table, la variable qui change d'état est la variable A. De ce fait, elle sera éliminée et on ne garde que les variables B, C, et D ayant l'état BCD dans les deux cases (on conserve les variables restantes inchangées). De ce fait, F = $B\,\overline{C}\,D$

On trouvera la même chose en appliquant les règles de Boole.

A partir de la table de vérité on extrait la forme canonique F= $A\,B\,\overline{C}\,D + B\,\overline{C}\,D\,\overline{A}$ puis on applique le théorème de la complémentation et obtient : F = $B\,\overline{C}\,D\,(A + \overline{A}) = B\,\overline{C}\,D$

En pratique on utilise le principe **d'intersection** des parties de la table affectées à ces variables. Dans l'exemple présenté, les deux cases sont l'intersection des variables, B, $\overline{C}$ et D.

Lois de simplification.

Il faudra constituer un groupement de cases adjacentes aussi grand que possible, le nombre de cases doit être une puissance de 2 dans chaque groupement :

- Un groupement de 2 cases réduit d'une unité le nombre de variables dans le minterne correspondant.

dans l'exemple ci dessus, on a un groupement de 2 cases → $\overline{A}\,B\,\overline{C}\,D$ + $AB\,\overline{C}\,D$ → $B\,\overline{C}\,D$, réduction d'une variable (F sera fonction de 3 variables au lieu de 4)

- Un groupement de 4 cases réduit de deux unités le nombre de variables dans le minterne correspondant.

Exemple

$$F = A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + AB\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D$$

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

(A : 11 10 ; B : 00 01 ; D : 00 01 ; C : 11 10)

$F = A\,\overline{C}$ (intersection de A et de $\overline{C}$)

F est fonction de 2 variables au lieu de 4

- Un groupement de 8 cases réduit de trois unités le nombre de variables dans le minterne correspondant.

Exemple

$F=A\,B\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + AB\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D + ABCD + A\,\overline{B}\,CD + ABCD - + A\,\overline{B}\,C\,\overline{D}$

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

$F = A$

F est fonction d'une seule variable au lieu de 4

Utiliser le maximum de recouvrement entre les groupements, pour avoir une simplification très poussée (on peut utiliser une même case plusieurs fois).

Exemple

$F= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\overline{C}\,\overline{D} + A\,B\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,D + A\overline{B}\,\overline{C}\,D$

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

$F = \overline{C}\,\overline{D} + A\overline{C}\,D$ (1)

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

$F = \overline{C}\,D + A\overline{C}$ (2)

La forme (2) utilise le principe de recouvrement, elle donc plus simplifiée.

Définir un voisinage externe sur la table comme si cette dernière pouvait être enroulée dans les 2 directions.

Exemple

$F= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,B\overline{C}\,\overline{D} + AB\overline{C}\,D + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,D + ABCD + A\overline{B}\,C\overline{D}$

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

$F = A\overline{D} + \overline{B}\,\overline{C}$

Si le nombre de variables dépasse 4, la méthode de simplification de Karnaugh devient moins aisée.. Dans ce cas, on utilisera les méthodes algorithmiques ; Quine/ McCluskey, Petrick ou Espresso (non traitées dans ce cours).

Remarque

On peut avoir dans certains cas des combinaisons d'entrée non définies appelées états indifférents (notés par « **x** »). On les remplacera dans le diagramme de Karnaugh par 1 ou 0 de façon à avoir la simplification la plus optimale.

1.6 Réalisation des circuits logiques combinatoires (Logigrammes)

A l'aide de ces portes logiques de base on peut concevoir un circuit logique quelconque réalisant une fonction logique combinatoire. La méthode classique pour réaliser ce circuit se résume comme suit:

- Etablir la table de vérité à partir d'un cahier de charges
- Calculer la forme canonique de la fonction logique
- Simplifier la forme canonique
- Réaliser le circuit par les portes logiques de base

Schématisation du circuit logique combinatoire (logigramme) :

- Réaliser l'inversion par des portes NOT des variables d'entrées si leurs compléments figurent sur la forme simplifiée de la fonction.
- Construire les portes ET pour réaliser les différents mintermes de la fonction.
- Etablir le câblage des portes ET avec les entrées appropriées.
- Réunir l'ensemble des sorties des portes ET vers une porte OU dont la sortie est le résultat de la fonction logique.

Exemple

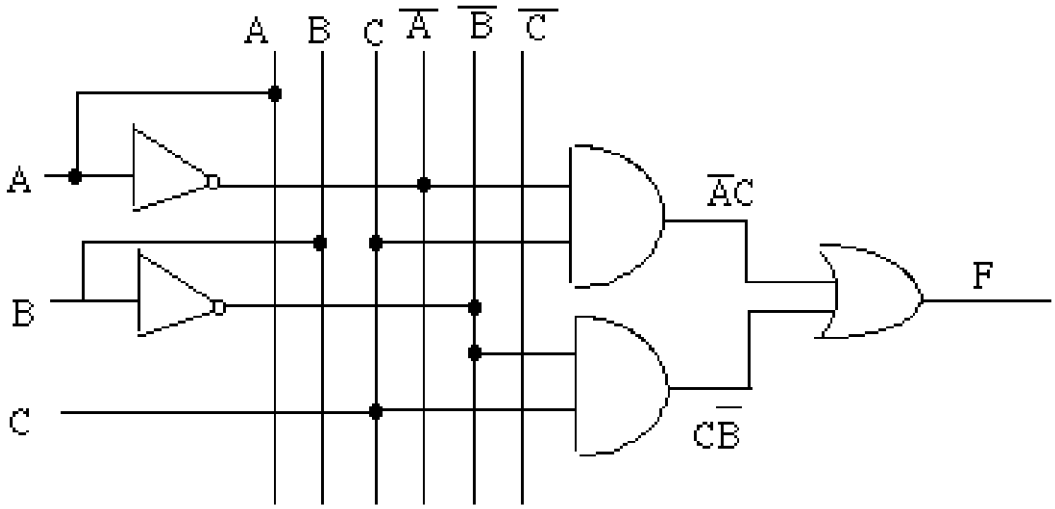
A	B	C	F
0	0	0	0
0	0	1	1*
0	1	0	0
0	1	1	1*
1	0	0	0
1	0	1	1*
1	1	0	0
1	1	1	0

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

(B, A, C)

Forme simplifiée: $F = \overline{A}C + C\overline{B}$

Logigramme



1.7 Equivalence entre les circuits logiques combinatoires

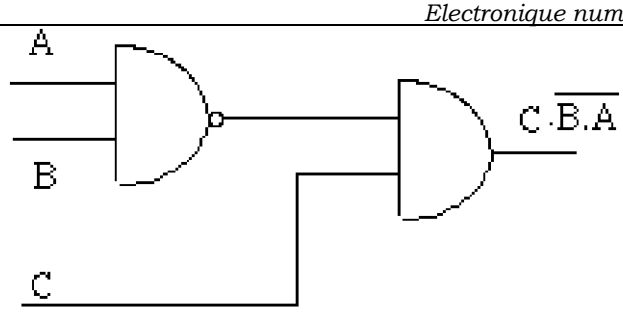
Après la simplification d'une fonction logique on peut utiliser les théorèmes de Boole et de DE MORGAN pour réaliser les fonctions logiques par plusieurs circuits équivalents.

Exemple

Le circuit de l'exemple précédent peut être obtenu d'une autre façon.

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

Distributivité Loi de Morgan.

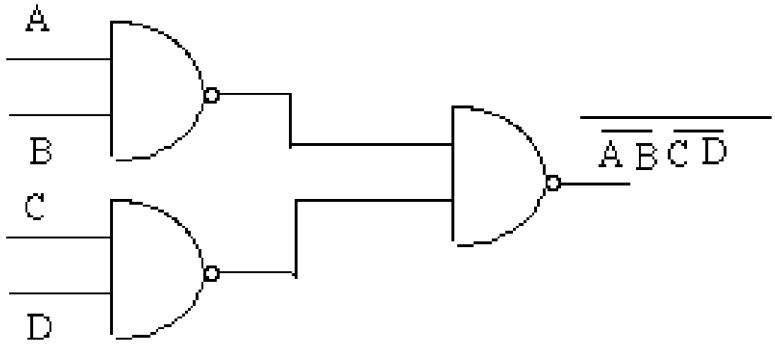


Par ailleurs, on peut réaliser un circuit logique à l'aide d'un seul type de portes. Par exemple, réaliser un circuit logique à base de portes NAND ou à base de porte AND ou une combinaison NOT/OR, AND/OR …etc

Exemple 1 F = AB + CD

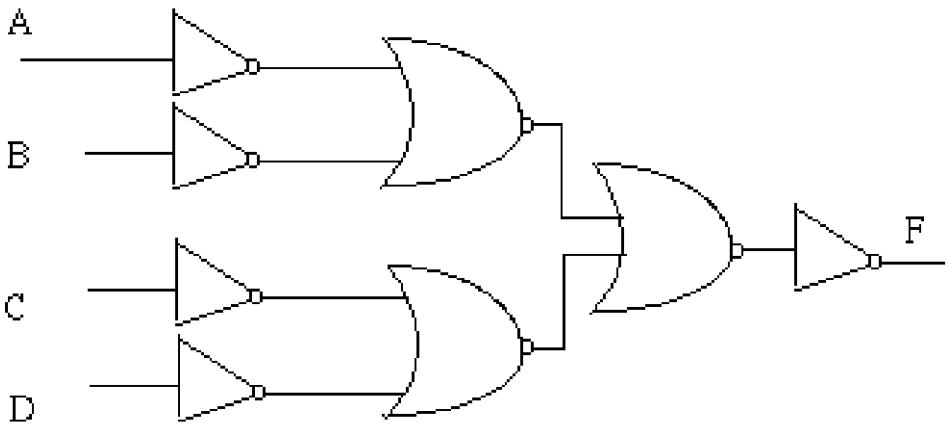
- Réaliser cette fonction par des portes NON ET (NAND)

F= AB + CD = $\overline{\overline{AB}.\overline{CD}}$ Loi de DE MORGAN.



- Réaliser la fonction F par des portes NOR et NOT

F= AB+CD = $\overline{\overline{A}+\overline{B}+\overline{C}+\overline{D}}$ Loi de DE MORGAN.



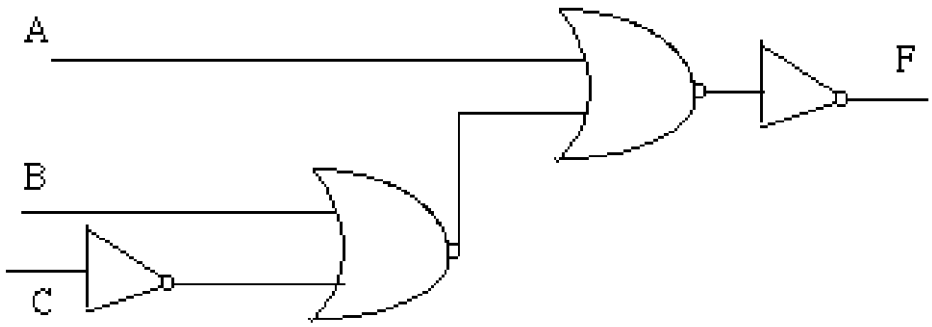
Exemple 2

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

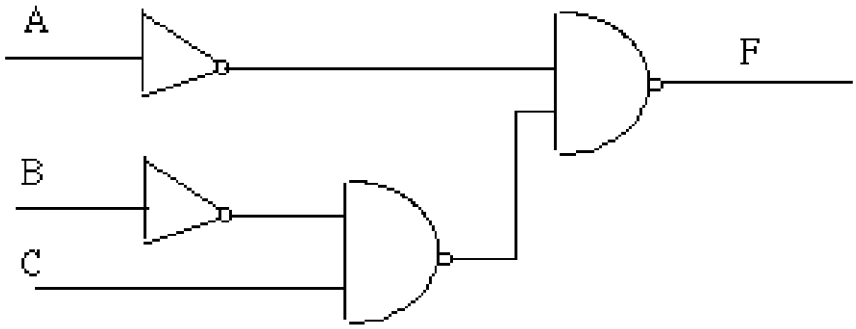
1) Réaliser le circuit par des portes NON et des portes Non OU (**NOT et NOR**)
2) Réaliser le circuit par des portes NON et des portes Non ET (**NOT et NAND)**
3) Réaliser le circuit par des portes NON et ET et OU (**NOT et AND et OR)**

Réponses

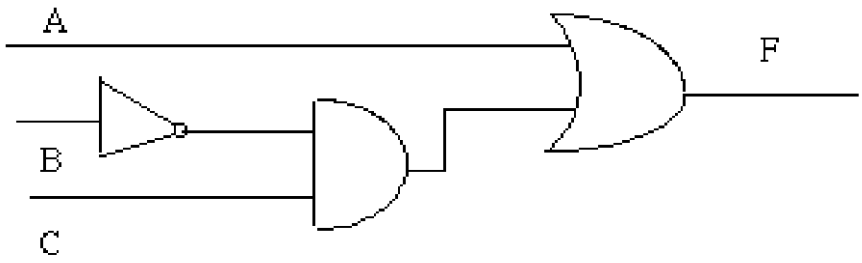
1) $F= A + \overline{B}\,C = \overline{\overline{A} + \overline{B} + \overline{C}}$ Loi de DE MORGAN.



2) $F = A + \overline{B}\,C = \overline{\overline{A}\overline{B}.C}$ Loi de DE MORGAN.



3) $F= A + \overline{B}\,C$



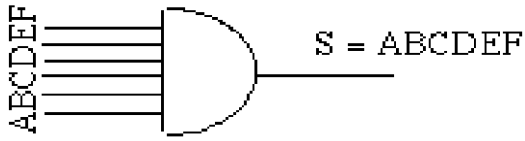
Les circuits 1, 2 et 3 sont équivalents.

1.8 Portes à entrées multiples

Jusqu'à maintenant nous avons utilisé des portes logiques à deux entrées. En pratique, on trouve des portes logiques à entrées multiples (> à 2) .

Exemple

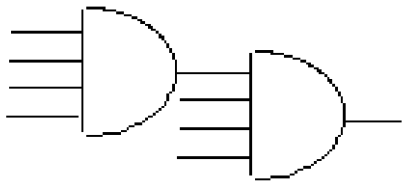
Portes ET à 6 entrées.



On pourra augmenter le nombre d'entrée en combinant des portes ayant un même nombre d'entrées.

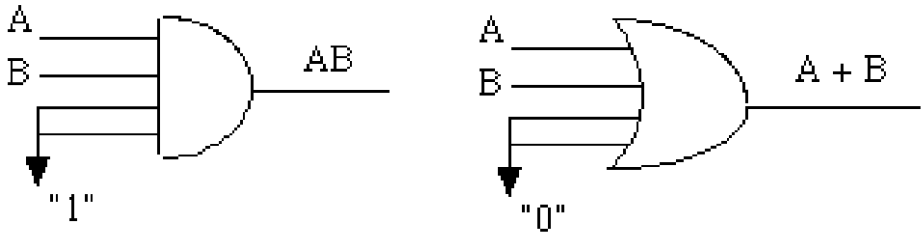
Exemple

En combinant deux portes ET à 4 entrées on peut obtenir à une porte ET à 7 entrées.



Ceci est également vrai pour les autres portes logiques de base.

Les entrées non utilisées des portes à entrées multiples peuvent être éliminées en les positionnant à "0" ou à "1" suivant la nature de la porte.



1.9 Circuits de calcul arithmétiques

a- *Circuit demi-additionneur d'un bit*

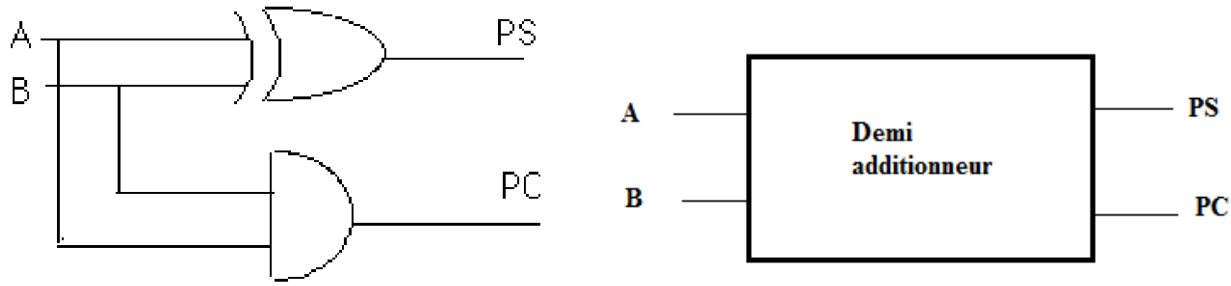
Ce circuit fait l'addition, sans tenir compte d'une éventuelle retenue générée par l'addition de deux bits précédents. Il calcul une somme partielle PS et génère une retenue partielle PC.

Table de vérité

A	B	PS	PC
0	0	0	0
0	1	1*	0
1	0	1*	0
1	1	0	1*

$PS = \overline{A}\,B + A\,\overline{B} = A \oplus B$ ====➔ (**EXOR**).
PC = A.B ====➔ (**AND**)

Logigramme



b- Circuit Additionneur complet d'un bit

Ce circuit fait l'addition de deux bits en tenant compte la retenue générée éventuellement par l'addition de deux bits précédents. Il contiendra 3 entrées à savoir deux entrées pour les bits A et B et une entrée à laquelle on applique la retenue interne (Carry in notée Cin) de l'addition précédente. Il génère une somme finale S et une retenue externe (Carry in notée :Cout).

Table de vérité

A	B	Cin	S	Cout
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*

Formes canoniques

Somme

$$\mathbf{S} = \overline{A}\,\overline{B}\,Cin + \overline{A}\,B\,\overline{Cin} + A\,\overline{B}\,\overline{Cin} + ABCin$$
$$= \overline{Cin}\,(\overline{A}\,B + A\,\overline{B}) + Cin\,(\overline{A}\,\overline{B} + AB)$$
$$= \overline{Cin}\,(A \oplus B) + Cin\,\overline{A \oplus B}$$
$$= \overline{Cin}.PS + Cin.\,\overline{PS}$$
$$= \mathbf{PS \oplus Cin}$$

avec $PS = A \oplus B$ ====➔ la somme partielle de A et B

Retenue externe

$$\mathbf{Cout} = \overline{A}\,BCin + A\,\overline{B}\,Cin + AB\,\overline{Cin} + ABCin$$
$$= Cin\,(\overline{A}\,B + \overline{B}\,A) + AB$$
$$= AB + Cin.PS$$
$$= \mathbf{PC1 + PC2}$$

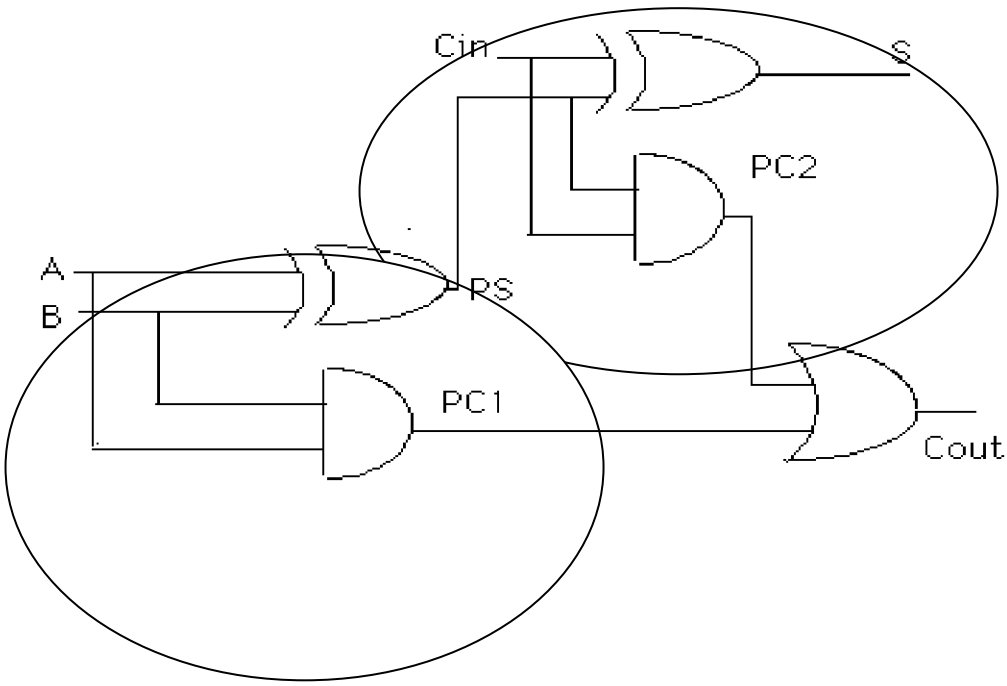
avec PC1= A.B ====➔ retenue générée lors de la somme partielle de A et B
PC2 = Cin.PS ===➔ retenue générée lors de la somme de PS et Cin

Ainsi, l'additionneur complet d'un bit est constitué principalement par deux demi-additionneur. ===➔

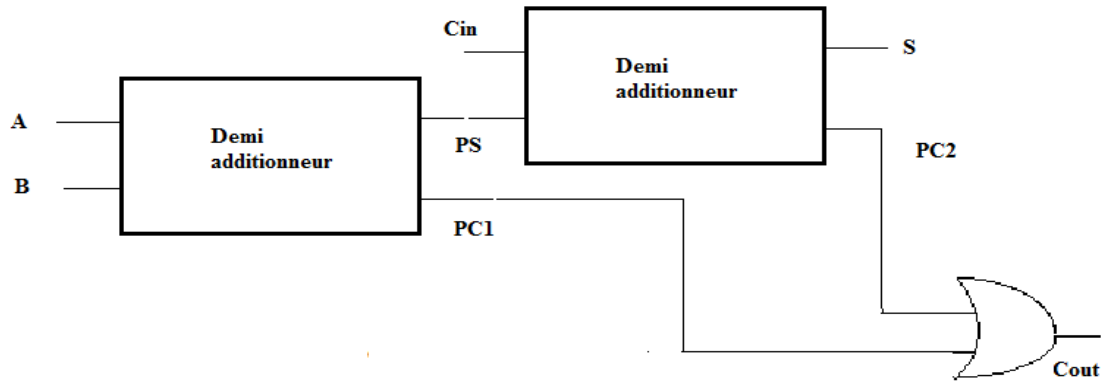
1er demi additionneur: entrées A et B sorties PS et PC1

2ème demi additionneur: entrées PS et Cin sorties S et Cout

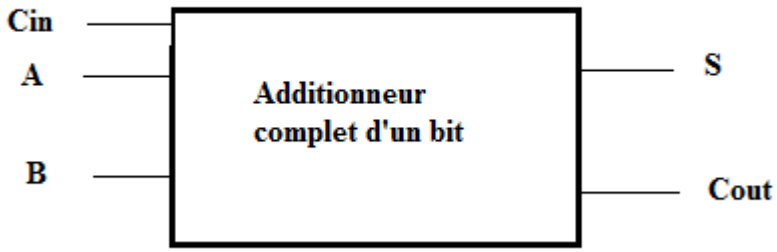
Logigramme



Il est équivalent à :



Il est également équivalent à :



c- *Soustraction d'un bit par addition du complément à 1*

On rappel que la soustraction de deux bits peut se faire sous la forme d'une addition:

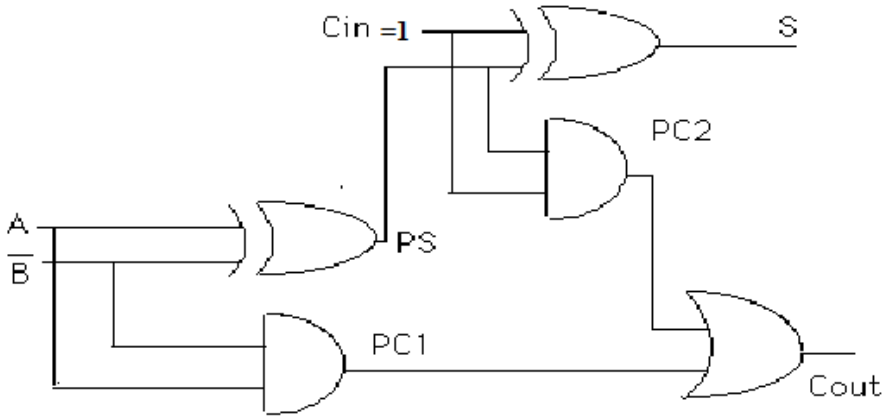
$$A - B = A + (-B) = A + \overline{B} + 1$$

Puisqu'on ajoute le "1" aux bits les bits moins significatifs on peut le considérer comme une retenue provenant d'une addition précédente. On utilise donc un additionneur complet en initialisant Cin à « 1 » lors de l'addition des 2 bits les bits moins significatifs de A et B.

Pour réaliser le Complément à 1 d'un nombre négatif il suffira de générer son inverse.

Logigramme

S= A - B

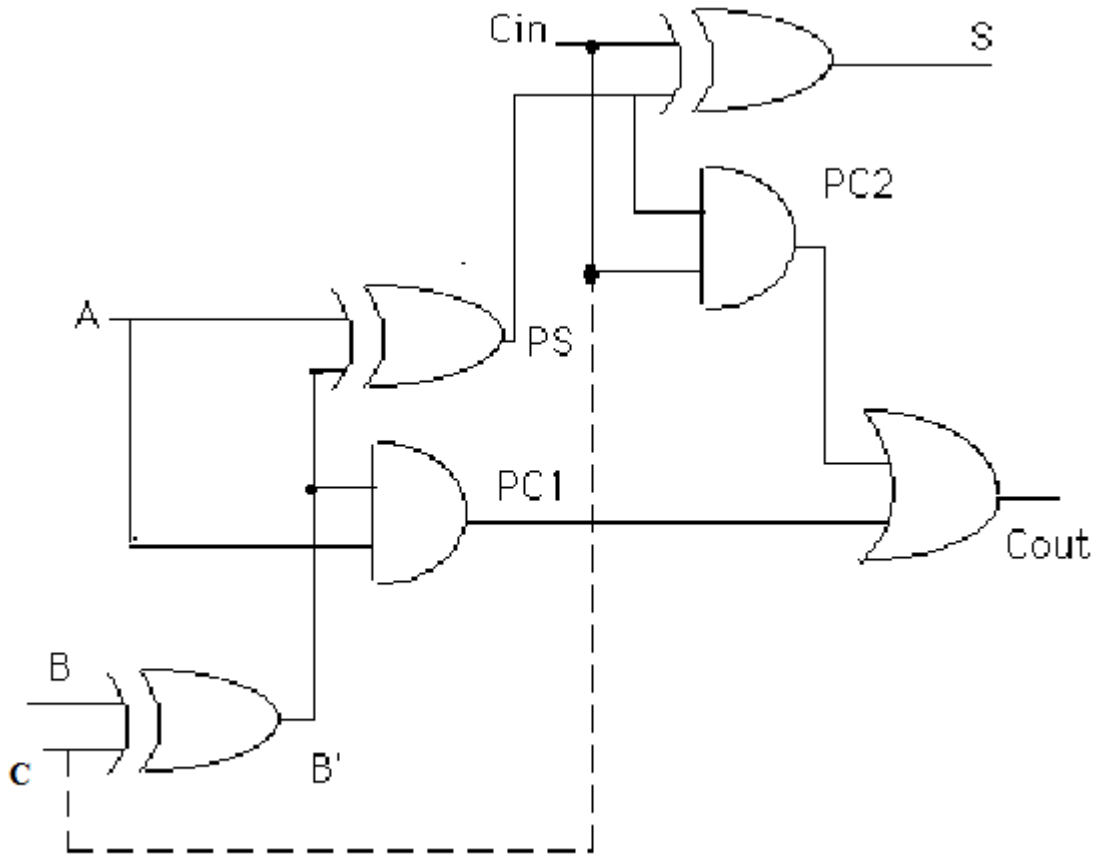


Si on veut réaliser un circuit qui effectue l'addition et la soustraction il faudra détecter la nature de l'opération par une ligne de commande C, par exemple :.

C	nature de l'opération	B'	Cin
1	A - B	$\overline{B}$	1
0	A + B	B	0

=====➔ Cin = C et B' = $\overline{C}B + C\overline{B} = B \oplus C$

Logigramme



En ajoutant une porte "XOR" on peut transformer un additionneur complet en un circuit Additionneur / Soustracteur complet. (dans le cas d'une soustraction il faudra exprimer les nombres à soustraire en complément à 2).

1.10 Opérations arithmétiques sur n bits

Les logigrammes étudiés jusqu'à présent s'appliquent à des opérations effectuées sur un seul bit. En pratique ces opérations s'effectuent sur plusieurs bits et permettent de réaliser des circuits de calcul appelés Unité Arithmétiques et Logique : **UAL (voir TD).**

1.11 Circuits logiques spécialisés

a) Comparateur

Un comparateur binaire est un logigramme logique qui effectue la comparaison entre 2 variables binaires notés A et B.

Il possède 3 sorties notées Egale : E (A = B), Supérieur : S(A > B) et Inférieur : I (A < B) qui indiquent le résultat de la comparaison:

− Si le nombre A est égal au nombre B (A = B), la sortie E passe à l'état 1 tandis que les sorties S et I passent à l'état 0.

− Si le nombre A est strictement supérieur au nombre B, seule la sortie S passe à l'état 1.

− Si le nombre A est strictement inférieur au nombre B, seule la sortie I passe à l'état 1.

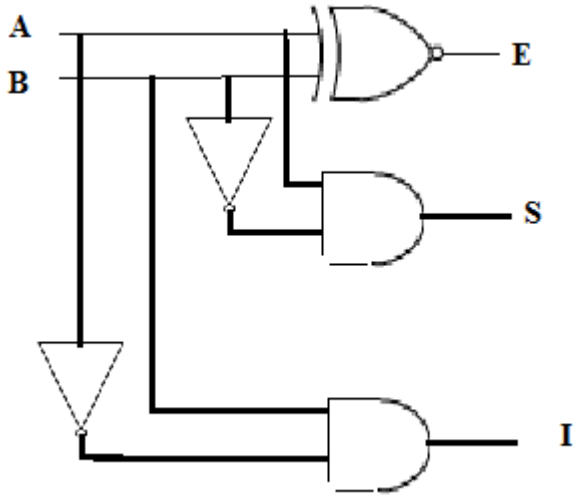
Le principe consiste à comparer d'abord les bits les plus significatifs (M S B) . S'ils sont différents, il est inutile de continuer la comparaison. Par contre s'ils sont égaux, il faut comparer les bits de poids immédiatement inférieur et ainsi de suite.

Exemple : n = 1 bit

Table de vérité

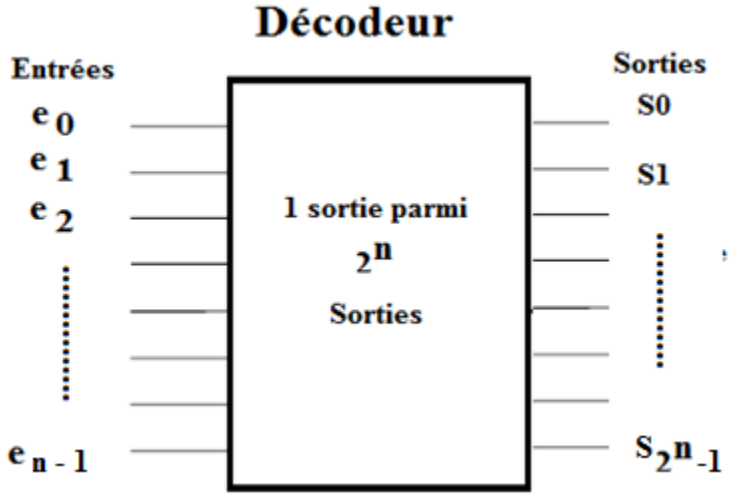
B	A	E	S	I
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

Logigramme



b) Décodeur

Le décodeur est un logigramme qui possède n entrées et 2^n sorties. Parmi toutes ces sorties une seule sera active (positionnée à 1). On parle d'un décodeur 1 parmi 2^n.

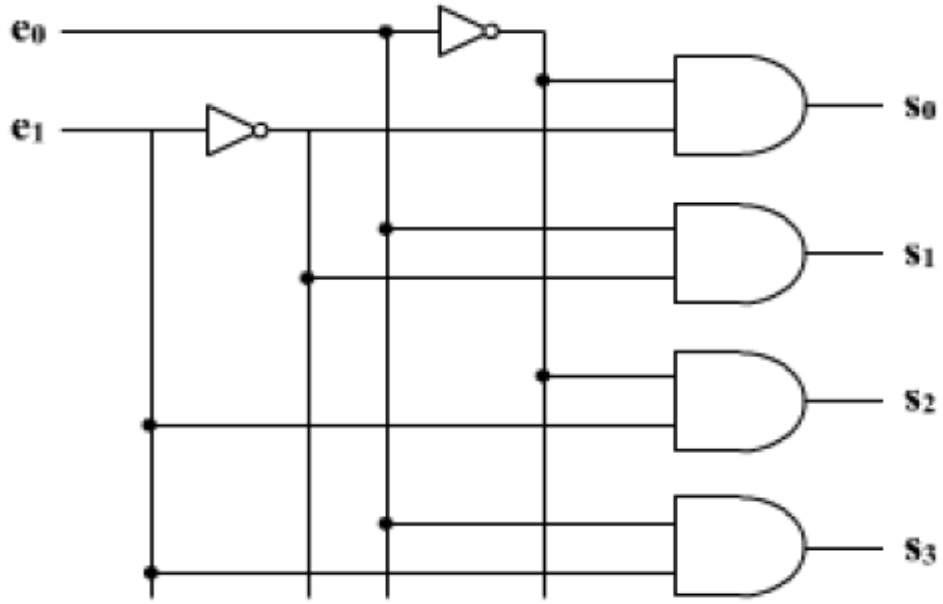


Exemple : Décodeur 1 parmi 4 : 2 entrées et 2^2 sorties

Table de vérité :

Entrées		Sorties			
e_1	e_0	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

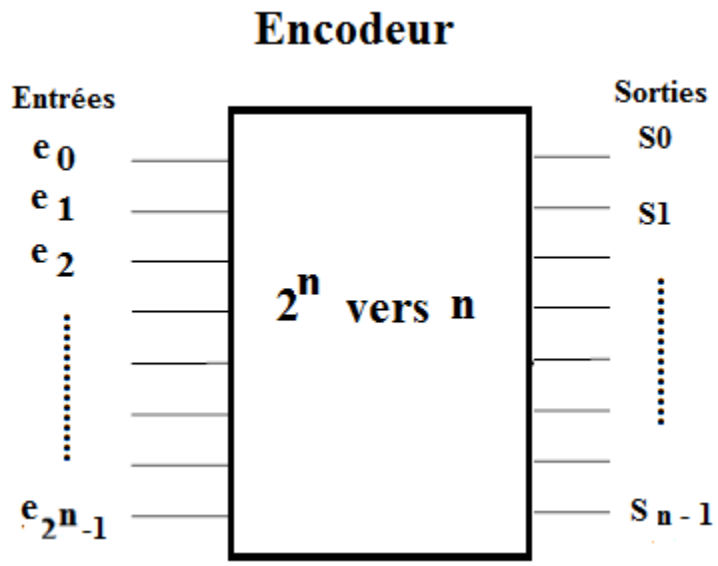
Logigramme



Remarque : L'indice de la sortie correspond à la conversion en décimal des états binaires des entrées, elle décode donc la valeur binaire présente aux entrées.

c) Encodeur

C'est une opération inverse du décodeur, c'est un logigramme à 2^n entrées dont une seulement est active (positionnée à 1). Il délivre sur n sorties (en code binaire naturel ou autre) le numéro binaire de cette entrée. On parle d'un encodeur de 2^n vers n.

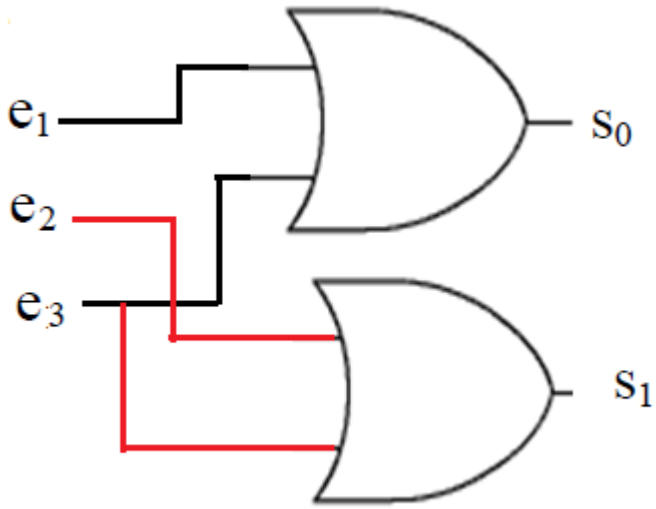


Exemple : Encodeur 4 vers 2 : 2^2 entrées vers 2 sorties

Table de vérité :

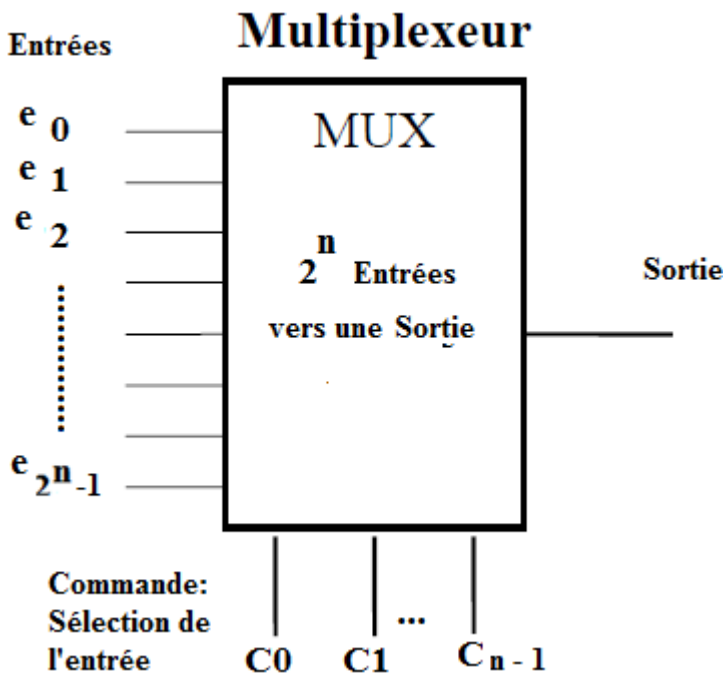
e_3	e_2	e_1	e_0	S_1	S_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logigramme



d) Multiplexeur

On aiguille une des (2^n) entrées, vers **une sortie**. La sortie «recopie» l'entrée sélectionnée. La sélection d'une entrée parmi $2^{\mathbf{n}}$ entrées est effectuée par une commande composée de n bits (Adresse). On parle alors d'un multiplexeur de $2^{\mathbf{n}}$ vers 1.

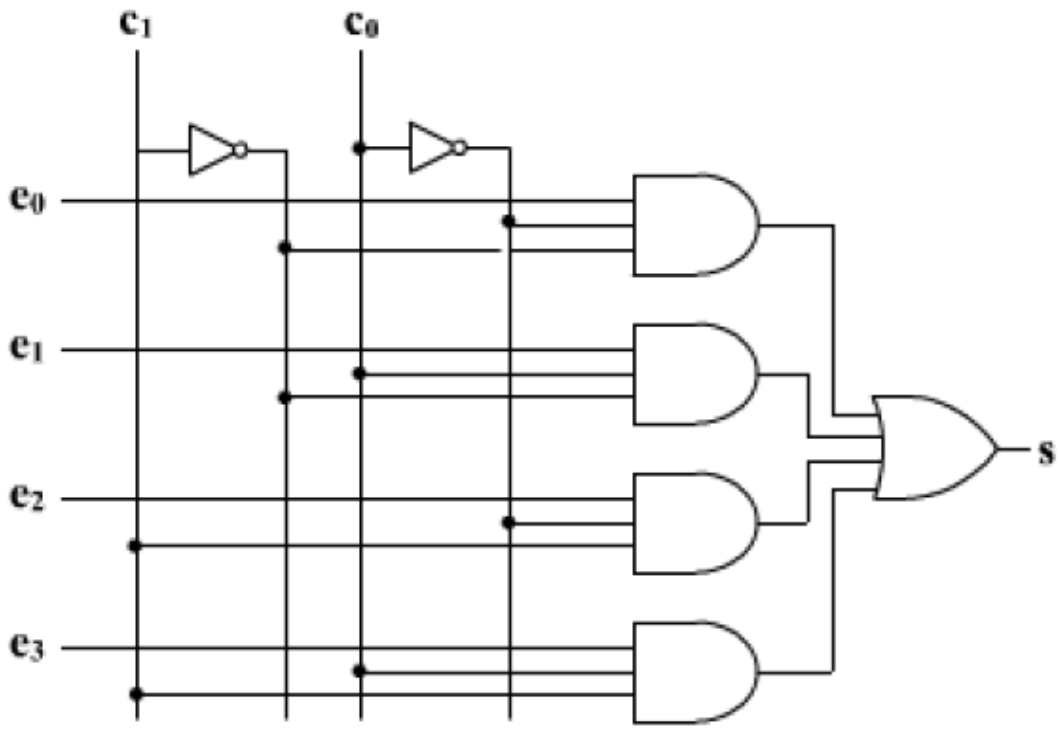


La commande C est composée de n entrées : C_0, C_1,…,C_{n-1} , elle est codé en binaire naturel.
Avantage d'un multiplexeur est de pouvoir utiliser un équipement unique par plusieurs équipements.

Exemple : Multiplexeur 4 à 1 : 2^2 entrées et 1 sortie, avec une commande de sélection de 2 entrées.
Table de vérité :

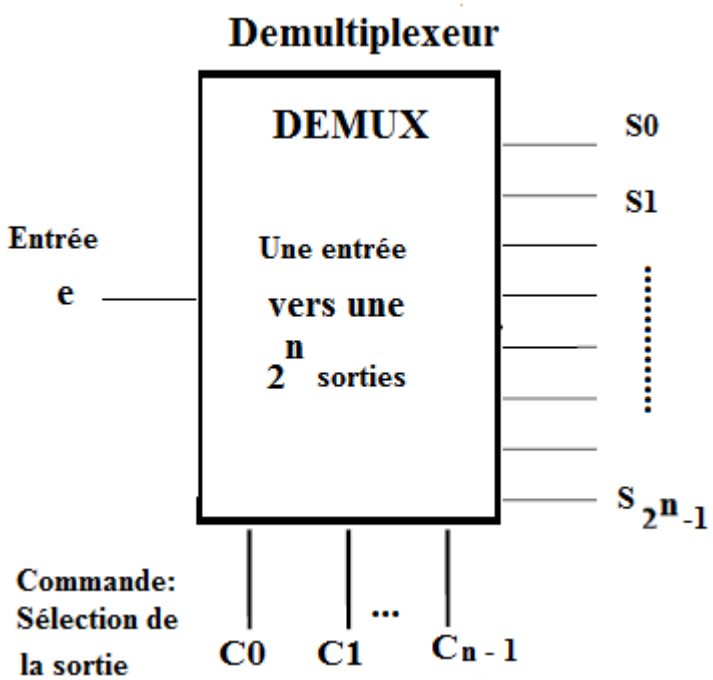
c_1	c_0	S
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3

Logigramme



e) Démultiplexeur

Ce logigramme possède 1 entrée et 2^n sorties. L'aiguillage de l'unique entrée vers une sortie (parmi 2^n sorties) est effectuée une commande composée de n bits (C0, C1,…,Cn-1). On parle d'un démultiplexeur de 1 vers 2^n

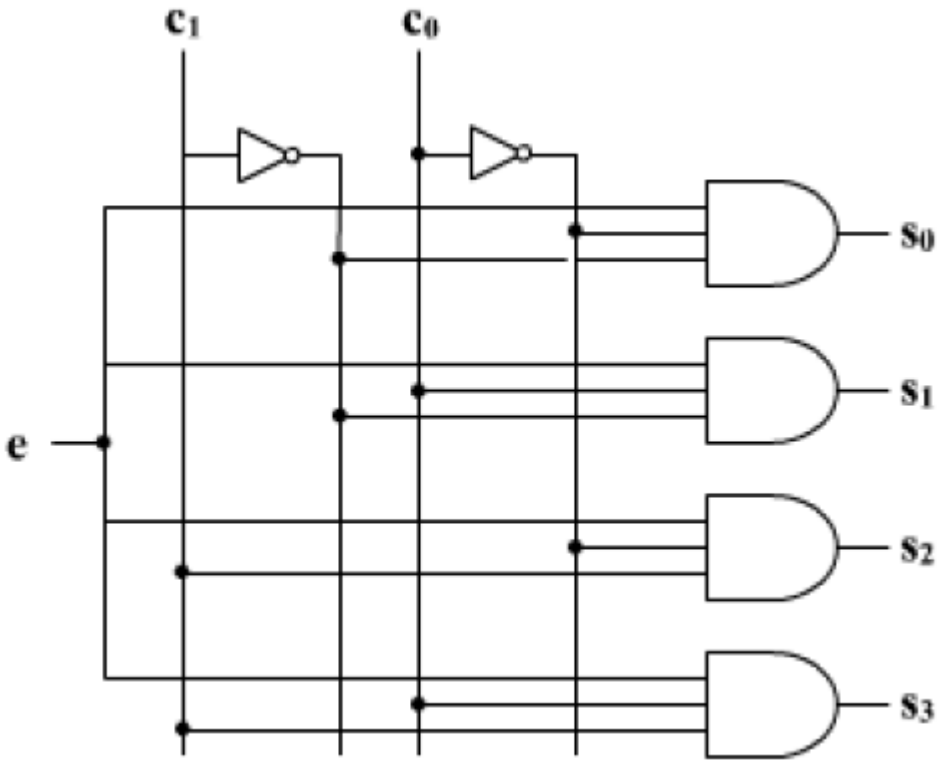


Exemple : Démultiplexeur 1 à 4 : 1 entrée, 2^2 sorties et 2 entrées de commande

Table de vérité

c_1	c_0	s_0	s_1	s_2	s_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



Applications des circuits spécialisés

- Les multiplexeurs / démultiplexeurs peuvent être utilisés dans le domaine de transmissions de données numériques lorsque les liaisons entre l'émetteur et le récepteur sont réalisées par un seul fil (communication série bit par bit).
- Les multiplexeurs sont utilisés pour effectuer les conversions parallèle-série, les bits des données disponibles sur les 2^n entrées se présentent en parallèle et sont mises en série (Sérialisation de données) au niveau de la sortie. Ils apparaissent donc bit/bit. L'opération désérialisation: conversion série-parallèle est réalisée par un démultiplexeur. Les bits se présentent en série au niveau de l'entrée et apparaissent en parallèle sur les 2^n sorties. Il est à noter que la matérialisation des fonctions logiques et le décodage peuvent être effectués respectivement par un multiplexeur et par un démultiplexeur.
- Le décodeur est un logigramme logique très employé dans les boîtiers mémoires. Il permet de localiser 1 case mémoire parmi 2^m cases. Il effectue donc un décodage des adresses des cases mémoires. Il permet également de calculer la capacité totale de stockage des boîtiers mémoires. Il est à noter que la matérialisation des fonctions logiques peut être également effectuée par décodeur.