Université Ibn Tofail
Ecole Nationale des Sciences Appliquées, Kénitra.

Cours d'électronique numérique

Année Universitaire : 2020-2021

Filière : Cycle Préparatoire (S4)

Module : Electronique analogique & numérique

Responsable : Tarik Boujiha

Ecole Nationale des Sciences Appliquées, Campus Universitaire, B.P 242, Kénitra-Maroc

Tél : (+212) 5 37 32 92 46 Fax : (+212) 5 37 32 92 47

Sommaire

Chapitre I : Systèmes de numération - Arithmétique binaire codes

1. Systèmes de numération

Le système de numération binaire est le plus important de ceux utilisés dans les circuits numériques, bien qu'il ne faille pas pour autant négliger l'importance d'autres systèmes. Le système décimal revêt de l'importance en raison de son acceptation universelle pour représenter les grandeurs du monde courant. De ce fait, il faudra parfois que des valeurs décimales soient converties en valeurs binaires avant d'être introduites dans le circuit numérique. Par exemple, lorsque vous composez un nombre décimal sur votre calculatrice (ou sur le clavier de votre ordinateur), les circuits internes convertissent ce nombre décimal en une valeur binaire.

De même, il y aura des situations où des valeurs binaires données par un circuit numérique devront être converties en valeurs décimales pour qu'on puisse les lire. Par exemple, votre calculatrice (ou votre ordinateur) calcule la réponse à un problème au moyen du système binaire puis convertit ces réponses en des valeurs décimales avant de les afficher.

1.1. Système décimal

Le système décimal comprend 10 nombres ou symboles qui sont 0, 1, 2, 3,4, 5, 6, 7, 8, 9 ; en utilisant ces symboles comme chiffres dans un nombre, on parvient à exprimer n'importe quelle grandeur. Le système décimal, appelé aussi système à base de 10, s'est imposé tout naturellement à l'homme puisque ce dernier possède dix doigts.

Le système décimal est dit à poids positionnels, en ce sens que la valeur d'un chiffre dépend de sa position (rang) dans le nombre. Par exemple, le nombre 3567 est le résultat de la somme $3.10^3 + 5.10^2 + 6.10^1 + 7.10^0$ alors le nombre 3567 est écrit dans le système de numération décimal ou encore système à base 10.

D'une façon générale, dans un système de numération à base b, un nombre N de base b sera décomposable en fonction des puissances entières de b tel que :

$$N_b = \sum_{i=0}^{i=n} a_i b^i$$

Où a_i est un chiffre tel que $0 \le a_i \le b-1$, les i sont des entiers positifs et n est l'exposant de b du chiffre de poids fort. Les b^i sont appelés poids ou rang et indique la grandeur de la quantité représenté.

Exemple : $(3567)_{10} = 3\times10^3 + 5\times10^2 + 6\times10^1 + 7\times10^0$

On note que le chiffre 3 est celui qui a le poids le plus élevé (MSB) et le chiffre 7 a le poids le plus faible (LSB).

1.2. Système binaire

Le système de numération binaire n'est qu'une autre façon de représenter les quantités. A première vue, ce système semble plus complexe que le décimal ; il est pourtant plus simple puisqu'il ne possède que deux chiffres. Le binaire est donc un système à base de 2 dont les deux chiffres binaires, ou bits, sont le 1 et le 0.

La position du 1 ou du 0 dans un nombre binaire indique son poids positionnel et détermine sa valeur dans le nombre, tout comme nous l'avons vu pour le système décimal. Dans un nombre binaire, les poids positionnels correspondent à des puissances de deux.

Exemple : $(1101{,}011)_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3}$; avec 2^3 le poids le plus fort et 2^{-3} le poids le plus faible.

Ils existent deux autres systèmes de numération très répandus dans les circuits numériques. II s'agit des systèmes de numération octal (base de 8) et hexadécimal (base de 16) qui servent tous les deux au même but, soit celui de constituer un outil efficace pour représenter de gros nombres binaires. Comme nous le verrons, ces systèmes de numération ont l'avantage d'exprimer les nombres de façon que leur conversion en binaire, et vice versa, soit très facile.

Pour chacun des systèmes décimal, binaire, octal et hexadécimal, l'ensemble des symboles possibles est :

* {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} pour le décimal,
* {0, 1} pour le binaire,
* {0, 1, 2, 3, 4, 5, 6, 7} pour l'octal,
* {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} pour l'hexadécimal.

Dans un système numérique, il peut arriver que trois ou quatre de ces systèmes de numération cohabitent, d'où l'importance de pouvoir convertir un système dans un autre.

Changement de base

1.3. Nombres entiers positifs

- **Conversion d'un nombre de base décimale en un nombre de base b quelconque**

La méthode la plus simple de conversion d'un nombre entier décimal en un nombre de base b est celle de la division par la base b répétée. Par exemple, pour convertir le nombre décimal N dans la base b, on commence par diviser N par b. Chaque nouveau quotient est ensuite divisé par b jusqu'à ce que le quotient soit 0. Les restes générés par chacune des divisions forment le nombre N dans la base b. Le premier reste produit devient le bit de poids le plus faible (LSB)

du nombre N dans la base b, alors que le dernier reste produit devient le bit de poids le plus fort (MSB).

Exemple :

* Convertissez le nombre $(45)_{10}$ en son équivalent binaire.

45	2					
1	22	2				
	0	11	2			
		1	5	2		
			1	2	2	
				0	1	2
					1	0

$(45)_{10} = (101101)_2$

* Convertissez le nombre $(214)_{10}$ en son équivalent hexadécimal.

214	16	
6	13	16
	13	0

$(214)_{10} = (D6)_{16}$

- **Conversion d'un nombre de base b quelconque en décimal**

Tout nombre écrit dans une base b quelconque peut être transformé en son équivalent décimal simplement en additionnant les termes obtenus du produit de chaque chiffre par son poids positionnel.

Exemple :

* Convertissez le nombre $(11011)_2$ en son équivalent décimal.

$(11011)_2 = 1\times 2^4 + 1\times 2^3 + 0 \times 2^2 + 1\times 2^1 + 1\times 2^0$

$(11011)_2 = (27)_{10}$

* Convertissez le nombre $(377)_8$ en son équivalent décimal.

$(377)_8 = 3 \times 8^2 + 7 \times 8^1 + 7 \times 8^0$

$(377)_8 = (255)_{10}$

- **Conversion octal–binaire et binaire–octal**

La conversion octal – binaire s'effectue en transformant chaque chiffre du nombre octal en son équivalent binaire de trois chiffres. L'opération inverse consiste à faire avec le nombre binaire des groupes de trois bits en partant du chiffre de poids le plus faible, puis de convertir ces triplets en leur équivalent octal ; au besoin on ajoute des zéros à gauche du bit de poids le plus fort pour obtenir un nombre juste de triplets.

Exemple :

* Convertissez $(376)_8$ en binaire.

3	7	6	Chiffre décimaux
011	111	110	Equivalent binaire

$(376)_8 = (011111110)_2$

* Convertissez $(1101110011)_2$ en octal.

001	101	110	011	Triplet
1	5	6	3	Equivalent octal

$(1101110011)_2 = (1563)_8$

- **Conversion hexadécimal – binaire et binaire - hexadécimal**

La conversion hexadécimal – binaire s'effectue en transformant chaque chiffre du nombre hexadécimal en son équivalent binaire de quatre chiffres. L'opération inverse consiste à faire avec le nombre binaire des groupes de quatre bits en partant du chiffre de poids le plus faible, puis on substitue à chaque groupe son chiffre hexadécimal équivalent ; au besoin on ajoute des zéros à gauche pour obtenir un dernier groupe de 4 bits.

Exemple :

* Convertissez $(376)_{16}$ en binaire.

3	7	6	Chiffre hexadécimal
0011	0111	0110	Equivalent binaire

$(376)_{16} = (001101110110)_2$

* Convertissez $(11101110011)_2$ en hexadécimal.

0111	0111	0011	Nombre binaire
7	7	3	Equivalent hexadécimal

$(11101110011)_2 = (773)_{16}$

1.4. Nombres fractionnaires

Exemple : Convertissez le nombre décimal 24,3125 en binaire.

Pour convertir la fraction décimale 0,3125 en binaire, on commence par multiplier 0,3125 par 2. Nous multiplions ensuite chaque nouveau produit ainsi crée par 2, jusqu'à ce que le produit fractionnaire soit 0 ou que le nombre de décimales désiré soit atteint. Les chiffres reportés ou retenues, générés par les multiplications, forment le nombre binaire. La première retenue produite devient le MSB et la dernière retenue devient le LSB.

	Retenue	
$0,3125 \times 2 = 0,625$	0	MSB
$0,625 \times 2 = 1,25$	1	

$0,25 \times 2 = 0,50$ 0

$0,50 \times 2 = 1,00$ 1 LSB

L'équivalent binaire de $(24)_{10}$ est $(11000)_2$

L'équivalent binaire de $(24,3125)_{10}$ est $(11000, 0101)_2$

2. Arithmétique binaire

2.1. Addition binaire

L'addition binaire s'effectue avec les mêmes règles qui s'appliquent à l'addition des nombres décimaux. Cependant, il n'y a que quatre cas qui peuvent survenir lorsqu'on additionne deux chiffres binaires et cela quel que soit le rang (cf. table 1). On commence par additionner les bits correspondant au plus petits poids, les 1 de retenue sont considérés comme des nouveaux bits et additionnés avec ceux de la colonne de poids juste supérieur.

Table 1 :

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 avec 1 de retenue

Exemple : Effectuer la somme de $(11,011)_2$ et $(10,110)_2$

11,011

\+ <u>10,110</u>

110,001

2.2. Ecriture des nombres signés

Comme les calculateurs numériques traitent aussi bien les nombres négatifs que les nombres positifs, une certaine convention est adoptée pour représenter le signe du nombre (+ ou -). Généralement, un autre bit appelé bit de signe est ajouté au nombre. La convention la plus courante consiste à attribuer au nombre positif le bit de signe 0 et au nombre négatif le bit de signe 1. Le nombre est donc stocké en mémoire en deux parties dont l'une est réservée à la valeur absolue du nombre en binaire et l'autre, placée à gauche, au bit de signe.

<u>Exemple</u> : Nombre décimal +25 exprimé en un nombre binaire signé de 8 bits avec la notation signe-grandeur.

0	0	0	1	1	0	0	1

Bit de signe grandeur

Le nombre décimal -25 s'exprime comme suit :

1	0	0	1	1	0	0	1

Bit de signe grandeur

Bien que cette notation signe–grandeur soit directe, les calculateurs numériques n'y ont généralement pas recours, en raison de la complexité des circuits qui matérialisent cette notation d'où l'utilisation dans ces machines de la notation en complément à 2 pour représenter les nombres signés.

2.3. Notation en complément à 1

Le complément à 1 d'un nombre binaire s'obtient en changeant chaque 0 par 1 et chaque 1 par 0. Autrement dit, en complémentant chaque bit du nombre.

Exemple :

1 0 1 1 0 0 1 0 Nombre binaire

0 1 0 0 1 1 0 1 Complément à 1

2.4. Notation en complément à 2

Le complément à 2 d'un nombre binaire s'obtient en prenant le complément à 1 de ce nombre et en ajoutant 1 au bit de son rang de poids le plus faible.

Exemple :

1 0 1 1 0 0 1 0 Nombre binaire

0 1 0 0 1 1 0 1 Complément à 1

\+ 1 Addition de 1

0 1 0 0 1 1 1 0

Grâce à la notation en complément à 2, les opérations de soustraction deviennent des opérations d'addition, c'est le cas dans les calculateurs numériques puisqu' avec les mêmes circuits s'effectuent des soustractions et des additions.

2.5. Addition en utilisant le complément à 2

On va considérer un circuit additionneur à deux entrées sur lesquelles sont disponibles deux mots de quatre bits.

1er cas : deux nombres positifs.

L'addition de deux nombres positifs est immédiate.

Exemple :

6	0. 0110
+ 3	+ 0. 0011
9	0.1001

2ème cas : nombre positif et nombre négatif plus petit.

On complémente à 2 le nombre négatif et on effectue la somme des deux nombres en ajoutant la retenue de la somme des bits de poids le plus fort au bits de signe. La retenue de la somme de ces derniers est ignorée.

Exemple :

9	0.1001	
- 3	+ 1.1101	Complément à 2 de 0011
6	**1** 0.0110	
	Retenue à rejetée	

3ème cas : nombre positif et nombre négatif plus grand.

On effectue l'opération de la même manière que dans le deuxième cas, seulement que le résultat sera négatif. Alors pour avoir le résultat final, on procède à la complémentation à 2 du résultat de l'addition codée.

Exemple :

3	0.0011	
- 7	+ 1.1001	Complément à 2 de 0111
- 4	1.1100	

Le résultat final de l'opération d'addition est le complément à 2 de 1.1100 qui est 1.0100.

4ème cas : les deux nombres sont négatifs

C'est le même cas que précédemment, le résultat de la somme codée sera négatif d'où sa complémentation à 2 pour avoir le résultat final.

Exemple :

- 3	1.1101	Complément à 2 de 0011
- 7	+ 1.1001	Complément à 2 de 0111
- 10	1.0110	

Le résultat final de l'opération d'addition codée est le complément à 2 de 1.0110 qui est 1.1010.

Remarque :

Dans chacun des exemples d'addition et de soustraction que l'on vient d'étudier, les nombres que l'on a additionnés étaient constitués à la fois d'un bit de signe et de 4 bits de grandeur. Les réponses aussi comportaient un bit de signe et 4 bits de grandeur. Toute retenue faite sur le bit de sixième rang était rejetée. De même, il faut que le résultat de l'opération reste inférieur strictement à $(16)_{10}$ pour ne pas avoir un dépassement de capacité.

Exemple :

```
   9      0.1001
 + 8      0.1000
          1 0001
        bit de signe
```

Le bit de signe de la réponse est celui d'un nombre négatif, ce qui est manifestement une erreur. La réponse devrait être +17. Etant donné que la grandeur est 17, il faut plus de 4 bits pour l'exprimer, et il y a donc un dépassement sur le rang du bit de signe.

3. Les principaux codes

Les circuits numériques fonctionnent avec des nombres binaires exprimés sous une forme ou sous une autre durant leurs opérations internes, malgré que le monde extérieur soit un monde décimal. Cela implique qu'il faut effectuer fréquemment des conversions entre les systèmes binaire et décimal. On peut avoir plusieurs combinaisons de bits 0 et 1 pour une même valeur décimale dont chaque combinaison correspond à un code.

3.1. Code binaire pur

Quand on fait correspondre à un nombre décimal son équivalent binaire, on dit qu'on a fait un codage binaire pur.

Nbre décim	Code binaire 8	4	2	1	Nbre décim	Code binaire 8	4	2	1
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

3.2. Code binaire décimal (DCB)

3.2.1. Généralités

Pour réaliser ces codes, il faut 10 combinaisons différentes de bits 0 et 1 afin de pouvoir réaliser les 10 symboles du système décimal.

1 seul bit permet 2 combinaisons : 0 ou 1

2 bits permettent 4 combinaisons : 00, 01, 10, 11

4 bits permettent 16 combinaisons : 0000, 0001,…..,1111

La réalisation de 10 combinaison implique donc l'emploi d'au moins 4 bits et il va donc falloir faire un choix de 10 combinaison parmi les 16 possibles.

Le choix parmi tous les codes possibles est effectué en fonction d'un certain nombre de critères.

3.2.2. Principaux codes binaires décimaux

♦ Code décimal binaire : DCB ou 8.4.2.1

C'est un code pondéré de poids 8, 4, 2, 1, il se présente sous la forme suivante :

Nbre	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Dans ce code chaque chiffre du nombre décimal doit être codé séparément par un groupe de 4 bits.

Exemple : 2571 s'écrit dans ce code : 0010 0101 0111 0001

Exemple d'opération arithmétique dans ce code :

Exemple 1 : 47 + 35 = 82

```
  0100   0111
+ 0011   0101
  0111   1100
+    1   0110
  1000   0010
```

On rencontre ici un mot codé qui ne correspond pas à une valeur connue ; il s'agit des six représentations codées de 4 bits interdites ou non valides. Cette représentation est apparue parce qu'on a additionné deux chiffres dont la somme dépasse 9. Pour résoudre ce problème, on ajoute $(6)_{10} = (0110)_2$ à ce mot codé inconnu afin de prendre en considération le fait qu'on saute six

représentations codées non valides. Si un report est produit, il sera ajouté à la somme DCB des chiffres du rang suivant.

Exemple 2 : 95 + 83 = 178

```
      1001  0101
    + 1000  0011
  1 0001  1000
    + 0110
0001  0111  1000
```

Dans le cas où l'addition de deux chiffres donne un report (celui-ci est additionné avec le chiffre de rang immédiatement à gauche), on ajoute une correction de $(0110)_2$ au résultat de la somme.

3.3. Code Gray

Le code gray est un code non pondéré et ne convient pas aux calculs arithmétiques, en ce sens qu'il n'y a pas de poids spécifiques qui correspondent aux positions des bits. Ce code est caractérisé par le fait qu'en passant d'une combinaison à la suivante un seul bit change de valeur, ce qui minimise les erreurs lors du codage. Comme par exemple les codeurs à positionnement rotatif, où la prédisposition aux erreurs augmente selon la quantité de bits changés entre deux nombres consécutifs d'une séquence.

Le tableau suivant énumère le code gray de 4 bits pour les nombres décimaux de 0 à 15.

Nbre	Code Gray				Nbre	Code Gray			
0	0	0	0	0	8	1	1	0	0
1	0	0	0	1	9	1	1	0	1
2	0	0	1	1	10	1	1	1	1
3	0	0	1	0	11	1	1	1	0
4	0	1	1	0	12	1	0	1	0
5	0	1	1	1	13	1	0	1	1
6	0	1	0	1	14	1	0	0	1
7	0	1	0	0	15	1	0	0	0

- **Conversion binaire-code gray**

Pour effectuer la conversion d'un nombre binaire en code gray, on procède selon les règles suivantes :

1- le bit de poids le plus fort du code gray, situé à l'extrême gauche, est le même que le MSB correspondant au nombre binaire.

2- En vous déplaçant de gauche à droite, additionnez chaque paire de bits adjacente du code binaire pour obtenir le bit suivant du code gray. Rejetez les retenues.

Exemple :

1	0	1	1	0	Binaire
1	1	1	0	1	Code Gray

- **Conversion code gray-binaire**

Pour effectuer la conversion d'un nombre codé en gray en code binaire, on procède selon les règles suivantes :

1- Le bit de poids le plus fort du code binaire, situé à l'extrême gauche, est identique au bit correspondant au code gray.

2- Additionnez chaque nouveau bit de code binaire crée au bit de code gray adjacent suivant (situé immédiatement à droite) en rejetant les retenues.

Exemple :

1	1	0	1	1	Code Gray
1	0	0	1	0	Binaire

3.4. Code ASCII: American standard code for Information Interchange

C'est un code alphanumérique universel utilisé dans la plupart des ordinateurs. Ce code comprend 128 caractères et symboles représentés par un code binaire de 7 bits. En réalité, il s'agit d'un code à 8 bits dont le MSB est toujours égal à 0. Ce code de 8 bits correspond aux nombres hexadécimaux de 00 à 7F. Le code ASCII sert à coder l'information alphanumérique transmise entre un ordinateur et ses périphériques d'entrée/sorties comme les écrans de visualisation ou les imprimantes. Un ordinateur recourt aussi à ce code pour stocker les informations envoyées par l'opérateur depuis son clavier.

CHAPITRE II : ELEMENTS D'ALGEBRE DE BOOLE

Un processeur est composé de transistors permettant de réaliser des fonctions sur des signaux numériques. Ces transistors, assemblés entre eux forment des composants permettant de réaliser des fonctions très simples. A partir de ces composants il est possible de créer des circuits

réalisant des opérations très complexes. L'**algèbre de Boole** (du nom du mathématicien anglais *Georges Boole 1815 - 1864*) est un moyen d'arriver à créer de tels circuits.

L'algèbre de Boole est une algèbre se proposant de traduire des signaux en expressions mathématiques. Pour cela, on définit chaque signal élémentaire par des variables logiques et leur traitement par des fonctions logiques. Des méthodes (table de vérité) permettent de définir les opérations que l'on désire réaliser, et à transcrire le résultat en une expression algébrique. Grâce à des règles appelées lois de composition, ces expressions peuvent être simplifiées. Cela va permettre de représenter grâce à des symboles un circuit logique, c'est-à-dire un circuit qui schématise l'agencement des composants de base (au niveau logique) sans se préoccuper de la réalisation au moyen de transistors (niveau physique).

1. Règles générales de l'algèbre de Boole

1.1. Postulats de l'algèbre de Boole

Une algèbre de Boole est un ensemble quelconque d'éléments E, à valeurs dans l'ensemble { 0, 1}, sur lequel on a défini :

- Une relation d'équivalence (égalité =)
- Deux lois de composition interne :
  - l'addition ou somme logique (+, ou)
  - la multiplication ou produit logique (. , et)
- Une loi de complémentation, telle que :

pour tout élément a de E, il existe un complément noté $\bar{a}$, on a $\bar{0} = 1$, $\bar{1} = 0$.

1.2. Axiomes de l'algèbre de Boole

- Commutativité

$\forall \; a, b \in E \qquad a + b = b + a \qquad (1)$

$a.b = b.a \qquad (1')$

- Associativité

$\forall \; a, b, c \in E \qquad (a + b) + c = a + (b + c) \qquad (2)$

$(a.b).c = a.(b.c) \qquad (2')$

- Double distributivité

$\forall \; a, b, c \in E \qquad a.(b + c) = a.b + a.c \qquad (3)$

$a + (b.c) = (a + b).(a + c) \qquad (3')$

- Pour chacune des deux opérations, il existe un élément neutre tel que :

$\forall \; a \in E \qquad a + 0 = a \qquad (4)$

$a.1 = a \qquad (4')$

- Chaque élément admet un inverse ou complémentaire tel que :

$\forall \; a \in E \qquad a + \bar{a} = 1 \qquad (5)$

$a.\bar{a} = 0 \qquad (5')$

1.3. Conséquences directes des axiomes

- Idempotence

$\forall \; a \in E \qquad a + a = a \qquad (6)$

$$a.a = a \qquad (6')$$

- $a+1=1\ ,\quad a.0=0 \qquad (7)$
- L'élément neutre 1 et l'élément neutre 0 sont uniques.
- Loi d'absorption :
  - Dans une somme booléenne, un terme absorbe ses multiples.
  - Dans un produit booléen, un facteur absorbe tous les facteurs composés de sommes qui le contiennent.

$$\forall\ a,b \in E \quad a+b.a = a \qquad (8)$$
$$a.b+a = a \qquad (8')$$

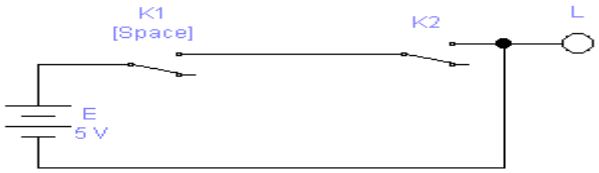
2. Les fonctions logiques

2.1. Définition

On appelle « **fonction logique** » une entité acceptant plusieurs valeurs logiques en entrée et dont la sortie (il peut y en avoir plusieurs) peut avoir deux états possibles : 0 ou 1.

En réalité ces fonctions sont assurées par des composants électroniques admettant des signaux électriques en entrée, et restituant un signal en sortie. Les signaux électroniques peuvent prendre une valeur de l'ordre de 5 Volts (c'est l'ordre de grandeur général) que l'on représente par un 1, ou 0 V que l'on représente par un 0.

Exemple :



$L = f(K_1, K_2)$
$L = 1$ si $K_1 = 1$ et $K_2 = 1$

Pour réaliser toutes les fonctions logiques, on a besoin de trois fonctions logiques de base : négation, intersection et la réunion.

Ces fonctions sont représentées par des schémas appelés logigrammes. Ils sont représentés soient par :

- les symboles européens actuels (norme CEI : commission d'électronique internationale)
- anciens symboles américains (norme MIL)
- symboles DIN (Deutch Industrie Normes)

2.2. Table de vérité

L'ensemble des valeurs prises par une fonction logique pour toutes les combinaisons possibles de ses variables est rangé dans un tableau, appelé table de vérité, comportant autant de colonne que de nombre de variables, plus une colonne pour ranger les valeurs de la fonction, et autant de ligne qu'il est possible de faire des combinaisons différentes avec les variables.

Exemple : somme logique

a	b	S = a + b	$\bar{S}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Cette table peut être traduite par une expression algébrique :

$$S = \bar{a}\,b + a\,\bar{b} + a\,b$$

Ceci traduit bien que S = 1 pour l'une ou l'autre des 3 combinaisons et que S = 0 pour la combinaison restante.

2.3. Théorème de MORGAN

Il peut être intéressant, connaissant l'expression d'une fonction booléenne, de trouver celle de la fonction complémentaire : ce que permet le théorème de DE MORGAN.

Théorème :

Nous obtenons l'expression de la fonction complémentaire d'une fonction F, en complémentant les variables dans l'expression de F et en intervertissant les signes . et +.

Ainsi : $F = a + b \leftrightarrow \bar{F} = \bar{a} \cdot \bar{b}$

$F = a \cdot b \leftrightarrow \bar{F} = \bar{a} + \bar{b}$

Dans le cas général de n variables :

$$\overline{\sum_{i=1}^{n} a_i} = \prod_{i=1}^{n} \bar{a_i}$$

$$\overline{\prod_{i=1}^{n} a_i} = \sum_{i=1}^{n} \bar{a_i}$$

Remarque : concernant la recherche de l'expression d'une fonction à partir du tableau de vérité de celle-ci. Reprenons l'exemple de la fonction S = a + b.

Lorsque la fonction comporte un plus grand nombre de 1 que de 0, il peut être plus simple de passer par l'intermédiaire de la fonction complémentaire.

Ainsi, pour éviter d'avoir à effectuer la simplification de S, nous povons écrire la fonction complémentaire $\bar{S}$ puis l'inverser en appliquant le théorème de DE MORGAN.

Soit : $\bar{S} = \bar{a}\,\bar{b}$

Puis : $S = \bar{\bar{S}} = a + b$

2.4. Formes canoniques

Toute fonction F de n variables prend l'état 1 pour certaines combinaisons des états de ces variables. Cette fonction peut être représentée dans E par un sous-ensemble F, union des sous-ensembles élémentaires correspondants chacun à une combinaison donnant la valeur 1 à la fonction.

Ainsi, par correspondance, l'expression algébrique de la fonction F de n variables pourra toujours se présenter sous la forme de la somme d'un certain nombre de termes constitués de produit de n variables, chacun de ces termes étant l'expression correspondant à un sous-ensemble élémentaire.

Evidemment, la somme sera composée au maximum de C termes (avec $C = 2^n$).

Cette expression sera dite première forme canonique de la fonction F.

a- *Exemple :*

soit la fonction F définie par sa table de vérité.

c	b	a	F	$\bar{F}$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

La forme canonique apparaît immédiatement à la lecture de la table (en notant les 1 de la fonction) :

$$F = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}bc$$

Remarque : une fonction F peut ne pas apparaître sous la forme canonique. Nous pouvons nous y ramener en homogénéisant l'expression de F.

Ainsi pour :

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

b- *Deuxième forme canonique*

L'expression canonique de la fonction F peut également apparaître sous la forme d'un produit de somme. Cette deuxième forme canonique peut s'obtenir en écrivant l'expression de $\bar{F}$ à partir de la table de vérité, puis en inversant cette expression pour obtenir F.

Ainsi en reprenant l'exemple précédent :

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

Finalement : $F = \bar{\bar{F}} = (a+b+c)(\bar{a}+\bar{b}+c)(a+b+\bar{c})(\bar{a}+b+\bar{c})(\bar{a}+\bar{b}+\bar{c})$

2.5. Les fonctions logiques à une seule variable binaire

Elles matérialisent les fonctions booléennes à une seule variable.

Ecrivons la table de vérité des fonctions d'une variable a :

a	f_0	f_1	f_2	f_3
0	0	1	0	1
1	0	0	1	1

Les expressions algébriques correspondantes aux 4 fonctions sont :

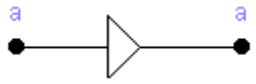
$$f_0 = 0 \ , \ f_1 = \bar{a} \ , \ f_2 = a \ , \ f_3 = 1$$

Si on exclut les deux fonctions constantes f (a) = 0 et f (a) = 1, il en reste deux qui présentent un intérêt particulier :
f (a) = a : c'est le buffer ou amplificateur
$f(a) = \bar{a}$: c'est l'inverseur.

- Le buffer
  - fonction booléenne : $a \rightarrow f(a) = a$
  - table de vérité :

a	f (a)
0	0
1	1

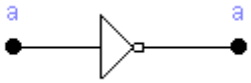
  - logigrammes



- l'inverseur
  - fonction booléenne : $a \rightarrow f(a) = \bar{a}$
  - table de vérité :

a	f (a)
0	1
1	0

  - logigrammes



D'une façon générale, pour les fonctions de n variables les colonnes des f_i comporte C lignes. Il y aura donc N fonctions différentes avec :

$$N = 2^C$$

Mais C n'est rien d'autres que le nombre de combinaisons des états des variables, soit :

$$C = 2^n$$

D'où finalement : $N = 2^C = 2^{2^n}$

2.6. Les fonctions logiques à deux variables binaires

Dans le cas où n =2, nous aurons $N = 4^{2^2} = 16$ fonctions. Le tableau suivant regroupe d'une manière condensée les 16 tables de vérité qu'il est possible d'écrire avec deux variables.

a	b	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Ces 16 fonctions ont comme expression algébrique :

$f_0 = 0$, $f_1 = \bar{a}\,\bar{b}$, $f_2 = \bar{a}\,b$, $f_3 = \bar{a}\,\bar{b} + \bar{a}\,b = \bar{a}$, $f_4 = a\,\bar{b}$, $f_5 = \bar{a}\,\bar{b} + a\,\bar{b} = \bar{b}$, $f_6 = \bar{a}\,b + a\,\bar{b}$, $\bar{f}_7 = a\,b \rightarrow f_7 = \bar{a} + \bar{b}$, $f_8 = a\,b$, $f_9 = \bar{a}\,\bar{b} + a\,b$, $f_{10} = \bar{a}\,b + a\,b = b$

$\bar{f}_{11} = a\,b \rightarrow f_{11} = \bar{a} + b$, $f_{12} = a\,\bar{b} + a\,b = a$, $\bar{f}_{13} = \bar{a}\,b \rightarrow f_{13} = a + \bar{b}$, $\bar{f}_{14} = \bar{a}\,\bar{b} \rightarrow f_{14} = a + b$, $f_{15} = 1$

Outre les opérateurs fondamentaux ET, OU, NON, nous trouvons quelques opérateurs importants tels que le NOR, le NAND, le OU exclusif et la fonction coïncidence.

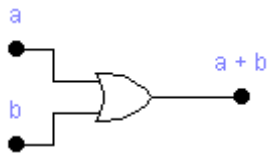
- **Somme logique**

Elle s'appelle aussi OU, OR (ou inclusif), réunion.

- fonction booléenne : $(a,b) \rightarrow f(a,b) = a + b$
- table de vérité :

a	b	f
0	0	0
0	1	1
1	0	1
1	1	1

- logigrammes :
-



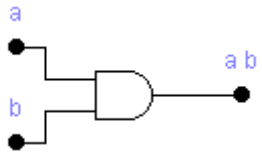
- **Produit logique**

Elle s'appelle aussi fonction ET, AND, intersection.

- fonction booléenne : $(a,b) \rightarrow f(a,b) = a \cdot b$
- table de vérité :

a	b	f
0	0	0
0	1	0
1	0	0
1	1	1

- logigrammes :



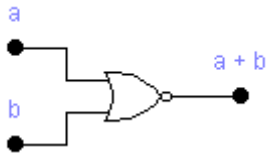
- **L'opérateur NOR**

- fonction booléenne : $(a,b) \rightarrow f(a,b) = \overline{a + b} = \bar{a} \cdot \bar{b}$
- table de vérité :

a	b	f
0	0	1

0	1	0
1	0	0
1	1	0

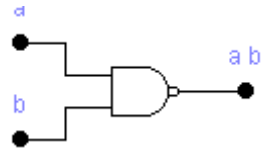
- logigrammes :



- **L'opérateur NAND**
  - fonction booléenne : $(a,b) \rightarrow f(a,b) = \overline{a \cdot b} = \bar{a} + \bar{b}$
  - table de vérité :

a	b	f
0	0	1
0	1	1
1	0	1
1	1	0

  - logigrammes :



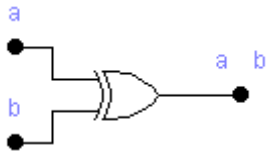
- **L'opérateur OU exclusif (XOR)**

C'est un opérateur qui donne un 1 logique à sa sortie si exclusivement une seule entrée sur les deux est à l'état 1.

  - fonction booléenne : $(a,b) \rightarrow f(a,b) = a \oplus b = \bar{a}\, b + a\, \bar{b}$
  - table de vérité :

a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

  - logigrammes :



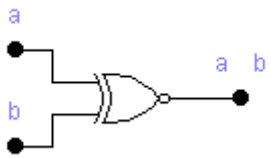
- **L'opérateur coïncidence**

C'est l'opérateur XOR complémenté. Il donne un 0 logique à sa sortie si exclusivement une seule des entrées est à l'état 1.

  - fonction booléenne : $(a,b) \rightarrow f(a,b) = \overline{a \oplus b} = \bar{a}\, \bar{b} + a\, b$
  - table de vérité :

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1

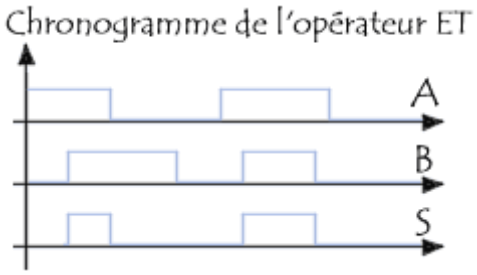
- logigrammes :



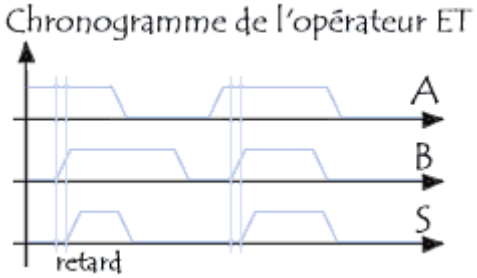
Chronogramme :

Un chronogramme est un diagramme montrant l'évolution des entrées et des sorties en fonction du temps.

Voici par exemple ce à quoi pourrait ressembler un chronogramme de l'opérateur ET :



En réalité les signaux électriques ne passent pas instantanément de 0 à 1, les pentes (ici verticales) sont obliques, et le traitement des entrées cause un retard sur les sorties :



CHAPITRE III: Simplification des fonctions logiques

Afin d'assurer la réalisation physique d'une fonction logique d'une façon plus simple, économique, il est nécessaire de chercher l'expression la plus simple de cette fonction.

Simplifier une expression booléenne revient à réduire :

- Le nombre des opérateurs, ou le nombre des entrées sur les opérateurs réalisant la fonction logique.
- La place disponible pour les opérations.
- Le nombre des interconnexions (réduire les aléas).
- Le temps de propagation de l'information à travers les circuits.

Cette simplification peut être faite soit par :

- des méthodes purement algébriques.
- Des méthodes graphiques

1. Méthodes algébriques

1.1. Mise en facteur

Ex 1 : $a + ab = a(1 + b) = a$

Ex 2 : $a\bar{b} + ab = a(\bar{b} + b) = a$

1.2. Adjonction à une somme d'un terme existant

Ex : $y = \bar{a}b + a\bar{b} + \bar{a}\bar{b}$

$y = \bar{a}b + a\bar{b} + a\bar{b} + \bar{a}\bar{b}$

$y = \bar{a}(b + \bar{b}) + \bar{b}(a + \bar{a})$

$y = \bar{a} + \bar{b}$

1.3. Adjonction à une somme d'un terme nul

Ex : $y = \bar{a}\bar{b} + a\bar{c} + \bar{b}\bar{c}$

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

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

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

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

1.4. Utilisation de l'inversion et des théorèmes de De Morgan

Ex : $y = c(a + \bar{a}b)$

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

$\bar{y} = \bar{c} + \bar{a}\bar{b}$

$y = c(a+b)$

1.5. Utilisation des distributivités

Ex : $y = a + \bar{a}b$

$y = (a + \bar{a})(a + b)$

$y = a + b$

1. Méthodes graphiques

1.1. Tables de Karnaugh

La table de Karnaugh est un outil graphique qui permet de simplifier de manière méthodique des expressions booléennes. Elle ressemble à une table de vérité en ce sens qu'elle présente toutes les valeurs possibles des variables d'entrée et la sortie résultante pour chaque valeur. Au lieu d'un arrangement de colonnes et de lignes comme dans une table de vérité, la table de karnaugh est un tableau de carrés. Chacun d'entre eux représente une valeur binaire des variables d'entrée. L'arrangement des carrés est conçu pour permettre la simplification d'une expression donnée en groupant ceux-ci selon des règles précises.

Le nombre de carrés d'une table de Karnaugh est égal au nombre total de combinaisons possibles des variables d'entrée. Pour n variables, le nombre de carrés est égal à 2^n.

Dans le cas de 3 variables, la table de Karnaugh est un tableau de 8 cases dont chacune d'elles correspond à un minterme. Dans le cas de 4 variables, c'est un tableau de 16 cases.

a \ bc	00	01	11	10
0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

ab \ cd	00	01	11	10
00	$\bar{a}\bar{b}\bar{c}\bar{d}$	$\bar{a}\bar{b}\bar{c}d$	$\bar{a}\bar{b}cd$	$\bar{a}\bar{b}c\bar{d}$
01	$\bar{a}b\bar{c}\bar{d}$	$\bar{a}b\bar{c}d$	$\bar{a}bcd$	$\bar{a}bc\bar{d}$
11	$ab\bar{c}\bar{d}$	$ab\bar{c}d$	$abcd$	$abc\bar{d}$
10	$a\bar{b}\bar{c}\bar{d}$	$a\bar{b}\bar{c}d$	$a\bar{b}cd$	$a\bar{b}c\bar{d}$

Avec l'ordre du remplissage choisi, lorsqu'on passe d'une case du tableau à sa voisine, il n'y a qu'une seule variable qui change d'état.

La notion essentielle pour l'utilisation de ces tables est celle d'états adjacents, ils correspondent :

- Soit à des cases voisines. Par exemple dans la table représentée ci-dessus, la case $\bar{a}\bar{b}c$ est adjacente aux cases $\bar{a}\bar{b}\bar{c}$, $\bar{a}bc$ et $a\bar{b}c$.
- Soit à des cases qui seraient voisines si on rapprochait les bords parallèles qui limitent le rectangle. C'est-à-dire que chaque case de la ligne du haut est adjacente à la case correspondante de la ligne du bas et chaque case de la colonne la plus à gauche est adjacente à la case correspondante de la colonne la plus à droite. Par exemple, la case $\bar{a}\bar{b}\bar{c}\bar{d}$ est adjacente à la case $a\bar{b}\bar{c}\bar{d}$ et à la case $\bar{a}\bar{b}c\bar{d}$.
- Soit encore à des cases symétriques par rapport aux frontières qui délimitent des carrés de 4 × 4 cases dans le cas d'un nombre de variables supérieur à 4.

Le but essentiel des tables de Karnaugh est la minimisation des expressions logiques. Cette minimisation consiste à supprimer les termes superflus et à réduire le plus possible le nombre

des termes utiles. La réduction se fait en essayant de grouper le plus grand nombre possible de cases adjacentes, les cases correspondantes aux termes de l'expression à réduire étant repérées, par exemple, par des "1".

Au cours du processus de réduction, une case du tableau peut être utilisée dans plusieurs groupements afin de rendre ceux-ci les plus grands possibles.

1.2. Exemples

1er exemple : Lorsqu'une fonction est définie par sa table de vérité, il suffit de placer des "1" dans les cases correspondantes aux monômes où la fonction est vraie. Habituellement, les 0 ne sont pas inscrit sur la table de karnaugh.

A	B	C	F(A,B,C)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

a \ bc	00	01	11	10
0	1		1	1
1		1		

L'expression simplifiée de la fonction F est :

$$F(A,B,C) = \bar{a}\bar{c} + \bar{a}b + a\bar{b}c$$

2ème exemple : Lorsque la fonction est définie par une somme de monôme, il faudra porter des "1" dans les cases correspondantes aux termes de cette somme.

$$F(A,B,C,D) = \bar{A}BC\bar{D} + \bar{B}CD + AC + \bar{B}$$

- Le monôme à 4 variables sera représenté par un "1" dans une seule case. On remplie donc la case du monôme $\bar{A}BC\bar{D}$ par "1".
- Le monôme à 3 variables sera représenté par 2 "1" dans deux case. En effet, pour le monôme $\bar{B}CD$ qui correspond à la combinaison 011, elle manque la variable A. On joigne cette variable, avec toutes ces combinaisons possibles, au monôme. Ce qui permet de remplir deux cases (la case $A\bar{B}CD$ et la case $\bar{A}\bar{B}CD$) par des "1".
- Le monôme à 2 variables sera représenté par 4 "1" dans quatre cases. Au monôme AC on rejoigne toutes les combinaisons possibles des deux variables manquantes B et D, ce qui permet de remplir 4 cases qui correspondent aux monômes $A\bar{B}C\bar{D}$, $A\bar{B}CD$, $ABC\bar{D}$ et $ABCD$.
- Le monôme à 1 variable sera représenté par 8 "1" dans huit cases. On rejoigne donc les 8 combinaisons des variables qui manquent A, C et D au monôme $\bar{B}$ et on remplie par des "1" les 8 cases correspondantes.

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

Pour simplifier la fonction booléenne, on regroupe les cases "1" pour former les boucles les plus grandes possibles jusqu'à ce que tous les "1" soient entourés (un "1" peut être entouré plusieurs fois).

On obtient, après simplification, la fonction suivante :

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

3ème exemple : il arrive parfois que des combinaisons de variables d'entrée ne soient pas permises dans une application. Dans le premier chapitre, nous avons vu que six combinaisons ne sont pas valides dans le code DCB : 1010, 1011, 1100, 1101, 1110, 1111. Comme ces états ne sont pas permis et qu'ils ne se produiront jamais dans une application fonctionnant avec le code DCB, ils peuvent être traités comme des conditions "**indifférentes**". Ces états prennent indifféremment la valeur 0 ou 1.

Les termes indifférents peuvent être utilisés pour simplifier d'avantage une expression booléenne. En effet, on place un X pour chaque terme indifférent dans la table de Karnaugh. Lorsque les 1 sont groupés, les X peuvent servir de 1 pour permettre un groupement de plus grande taille. Plus la taille du groupe est grande, plus le terme qui en découle est simple.

Soit la fonction F(A, B, C,D) définie par la table de vérité suivante :

A	B	C	D	F(A, B, C, D)
0	0	0	0	1
0	0	0	1	X
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	X
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	X

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

Si les conditions indifférentes sont remplacées par des 1, l'expression simplifiée de F s'écrira : $F(A,B,C,D) = \overline{C}\,D + \overline{B}\,\overline{D} + A\,D$

Sinon : $F(A,B,C,D) = B\,\overline{C}\,D + \overline{B}\,\overline{D} + A\,D$

CHAPITRE IV: Synthèse des circuits combinatoires

Méthodes de recherche des équations d'un circuit combinatoire

La méthode consiste à réaliser successivement les opérations suivantes :

- Déterminer les différentes variables et les fonctions à calculer
- Déterminer la table de vérité de chaque fonction
- Ecrire les équations logiques des fonctions des sorties
- Simplifier ces expressions
- Etablir le schéma correspondant

Au cours de cette dernière étape, il conviendra de se rappeler que les portes ne sont pas les seuls éléments possibles pour réaliser le schéma. Nous verrons en effet qu'il existe sous forme intégrée de nombreuses fonctions plus ou moins complexes qui conduisent à des solutions très élégantes et très simples.

1. Additionneur de base

1.1. Demi-additionneur

Rappelons les règles de base de l'addition binaire :

0 + 0 = 0 retenue 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. Ce dernier prend deux nombres binaires à ses entrées et produit deux nombres binaires à ses sorties : un bit de somme et un bit de retenue.

Symbole logique :

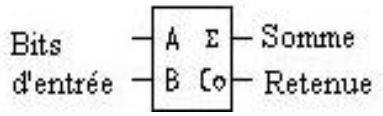


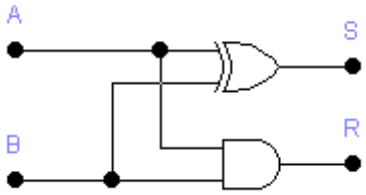
Table de vérité :

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Equation logique :

$S = A \oplus B$ $R = A\ B$

Diagramme logique :



1.2. Additionneur complet

L'additionneur complet prend deux bits d'entrée et une retenue et produit une sortie de somme et une retenue de sortie.
La différence fondamentale entre un demi-additionneur et un additionneur complet est que ce dernier traite une retenue d'entrée.

Symbole logique :

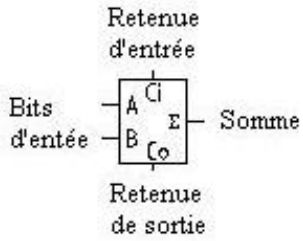


Table de vérité :

A	B	Ci	S	Co

0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Equation logique :

$$S = A \oplus B \oplus Ci \qquad Co = A\,B + (A \oplus B)\,Ci$$

Diagramme logique :

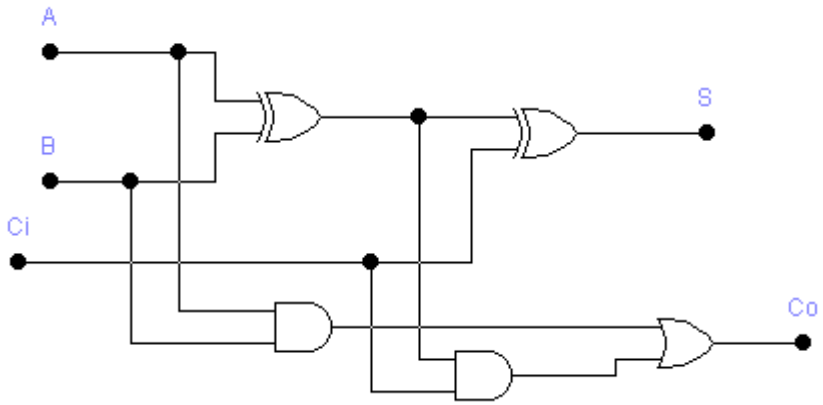
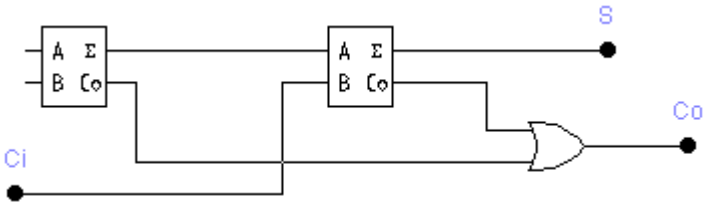


Schéma d'additionneur complet à partir de deux demi-additionneurs :

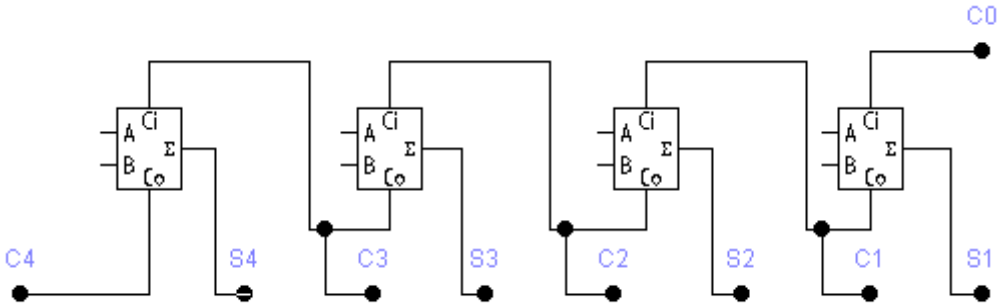


2. Additionneurs binaires parallèles

Le branchement de deux additionneurs complets (ou plus) forme un additionneur binaire parallèle. Pour additionner deux nombres binaires, il faut un additionneur complet pour chaque bit des nombres. Il faut donc deux additionneurs pour des nombres de 2 bits, quatre additionneurs pour des nombres de 4 bits, etc. La sortie de retenue de chaque additionneur est connectée à l'entrée de retenue de l'additionneur de bit de rang plus élevé suivant. Notez qu'on peut utiliser un demi-additionneur pour la position de poids le plus faible, ou relier l'entrée de retenue d'un additionneur complet à la masse, puisqu'il n'y a pas d'entrée de retenue pour la position du bit de poids le plus faible.

Additionneur parallèle de 4 bits contenant quatre additionneurs complets :

Diagramme d'ensemble :



Symbole logique :

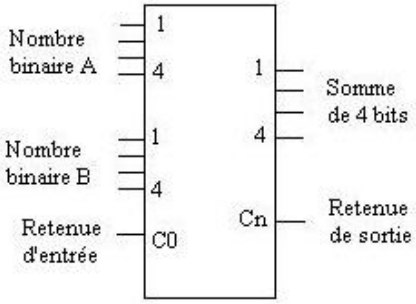


Table de vérité :

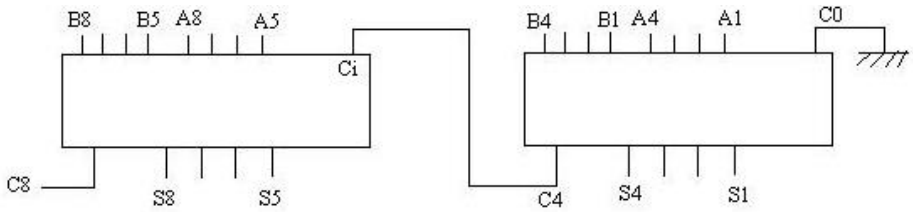
C_{n-1}	A_n	B_n	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Additionneur MSI (circuits d'intégration à moyenne échelle) :74LS83A, 74LS283.

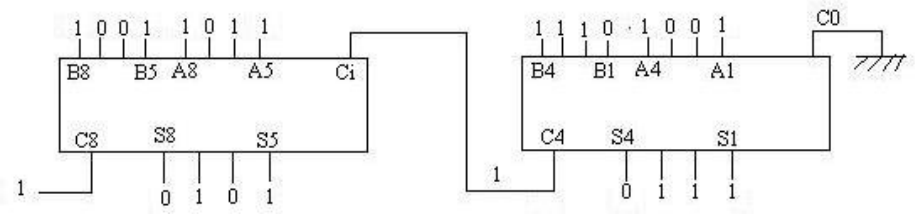
Montage en cascade d'additionneurs :

On peut utiliser un montage spécial de deux additionneurs de 4 bits pour faire la somme de deux nombres de 8 bits. On connecte l'entrée de retenue de l'additionneur des rangs de poids faibles (C_0) à la masse et on connecte la sortie de retenue de l'additionneur des rangs de poids faibles à l'entrée de retenue de l'additionneur de rangs de poids forts. Ce procédé est connu sous le nom de montage en cascade. L'additionneur de rangs de poids faibles est celui qui traite les 4 bits de poids les plus faibles des nombres et l'additionneur des rangs de poids forts est celui qui traite les 4 bits de poids les plus forts des nombres de 8 bits.

Schéma de mise en cascade d'additionneur de 4 bits pour former un additionneur 8 bits :



Exemple :



Additionneurs des rangs de poids forts

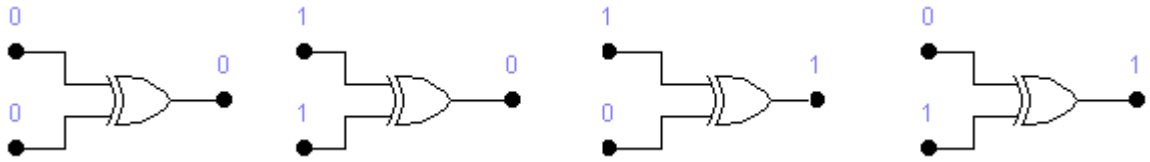
Additionneurs des rangs de poids faibles

3. Comparateur

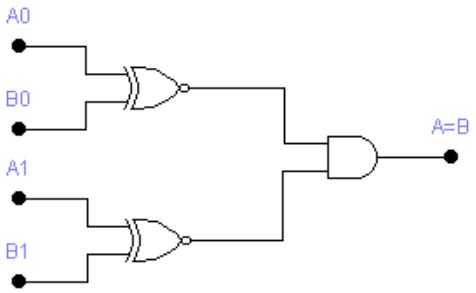
La fonction principale d'un comparateur est de comparer les grandeurs de deux quantités binaires afin de déterminer la relation existante entre ces quantités. Egalité et inégalité.

Egalité :

La porte ou exclusif peut servir de comparateur de base puisque sa sortie vaut 1 si les deux bits à ses entrées sont différents et qu'elle vaut 0 si les bits d'entrées sont identiques.



Pour comparer deux nombres de 2 bits, on peut utiliser le circuit suivant :



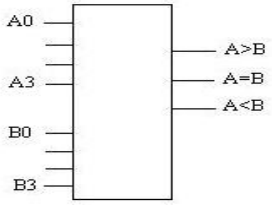
$A=A_1A_0$, $B=B_1B_0$

La sortie de la porte Et indique l'égalité (1) ou l'inégalité (0) de deux nombres.

Inégalité :

En plus de la sortie d'égalité, la plupart des comparateurs à circuit intégré sont munis de sorties additionnelles indiquant quel nombre binaire est le plus grand. Une sortie indique la condition lorsque le nombre A est plus grand que le nombre B (A>B) et une autre sortie indique si le nombre A est plus petit que le nombre B (A<B), comme l'indique le symbole logique du comparateur de 4 bits suivant :

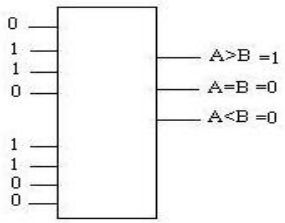
Symbole logique :



Les conditions possibles sont les suivantes :
Si A3=1 et B3=0 : A>B
Si A3=0 et B3=1 : A<B

Si A3=B3, on examine le rang de poids plus faible suivant pour identifier une condition d'inégalité.

Exemple :

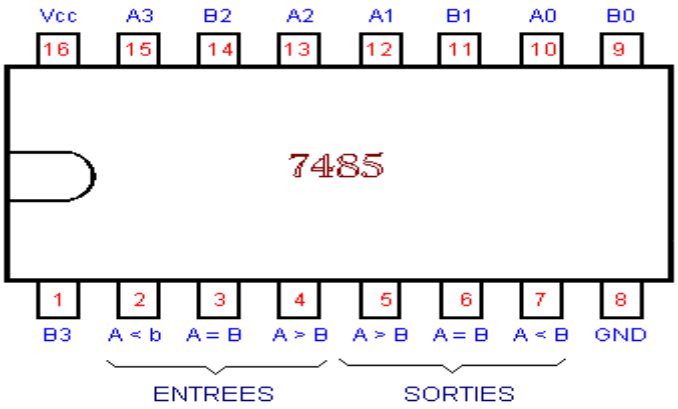


ANALYSE D'UN COMPARATEUR INTÉGRÉ : LE 7485

Le circuit intégré 7485 est un comparateur 4 bits, c'est-à-dire qu'il effectue la comparaison de deux nombres de 4 bits.

De plus, il dispose de 3 entrées notées A = B, A > B et A < B qui autorisent la mise en cascade de plusieurs circuits comparateurs du même type.

Ainsi, on peut comparer des nombres de 8, 12, 16 bits....



Brochage du circuit intégré 7485

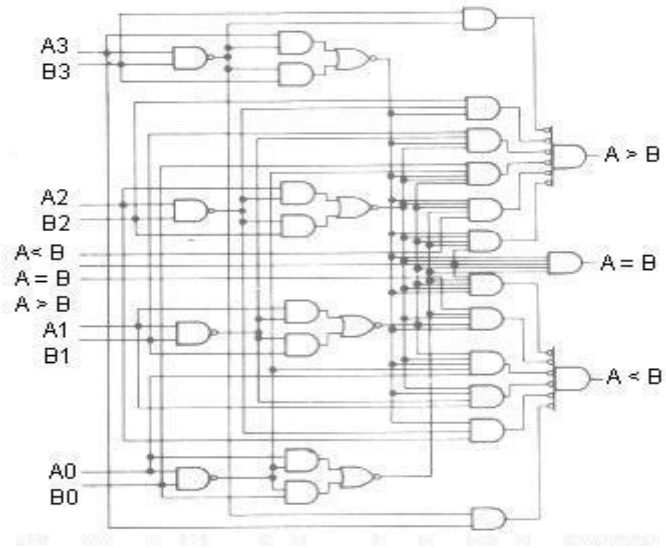


Schéma logique du circuit intégré 7485

Avec ce circuit, on compare le nombre A composé des bits A3, A2, A1 et A0 (A3 = MSB et A0 = LSB) avec le nombre B composé des bits B3, B2, B1 et B0 (B3 = MSB et B0 = LSB).

La table de vérité ci-dessous met en évidence l'action des entrées A > B, A < B et A = B.

Entrées des nombres				Entrées cascadables			Sorties		
A3, B3	**A2, B2**	**A1, B1**	**A0, B0**	**A > B**	**A < B**	**A = B**	**A > B**	**A < B**	**A = B**
A3 > B3	X	X	X	X	X	X	1	0	0
A3 < B3	X	X	X	X	X	X	0	1	0
A3 = B3	A2 > B2	X	X	X	X	X	1	0	0
A3 = B3	A2 < B2	X	X	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	1	0	0
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	1	0	0
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	0	0	1	0	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	1	0	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	0	1	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	X	X	1	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	1	0	0	0	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	0	0	1	1	0

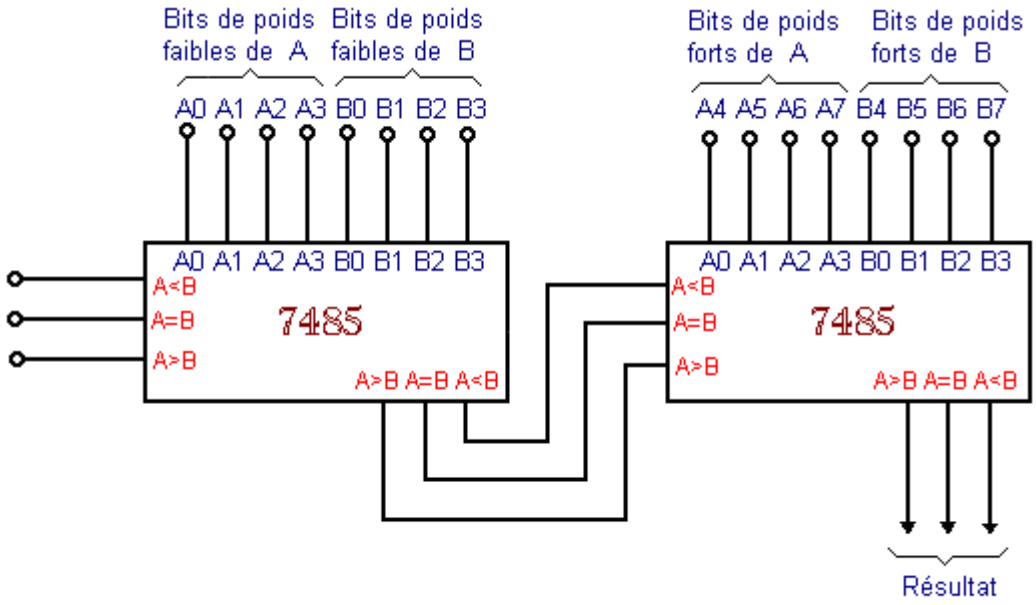
▶ Si l'on souhaite que la sortie A = B passe à l'état 1 chaque fois que les deux nombres binaires sont égaux, il suffit de porter l'entrée A = B à l'état 1, l'état des entrées A < B et A > B n'ayant alors pas d'importance.

▶ Si l'on souhaite que la sortie A > B passe à l'état 1 également dans le cas où les deux nombres binaires sont égaux, il suffit de porter l'entrée A > B à l'état 1 et de porter les entrées A < B et A = B à l'état 0.

Dans cette configuration de l'état des entrées A > B, A < B et A = B, la sortie A > B est à l'état 1 lorsque le nombre binaire A est supérieur au nombre binaire B ou quand ces deux nombres sont égaux. Elle indique donc si A ³ B.

▶ De même, en portant l'entrée A < B à l'état 1 et les entrées A > B et A = B à l'état 0, la sortie A < B indique le nombre binaire A est inférieur ou égal au nombre binaire B.

En mettant en série deux comparateurs 7485, on peut comparer deux nombres de 8 bits. Il suffit de relier la sortie A = B du premier comparateur à l'entrée correspondante du second et de faire de même avec les sorties A > B et A < B. Les liaisons à effectuer sont indiquées sur la figure suivante :



Mise en cascade de deux circuits intégrés 7485.

Ainsi, on compare le nombre A formé des 8 bits A7 à A0 (A7 = MSB et A0 = LSB) et le nombre B formé des 8 bits B7 à B0 (B7 = MSB et B0 = LSB).

Le premier circuit compare les poids faibles de A avec le poids faibles de B. Le résultat de cette comparaison est transmis aux entrées A < B, A = B et A > B du deuxième circuit.

Celui-ci compare les poids forts de A avec les poids forts de B et, en fonction du résultat de la comparaison des bits de poids faibles de A et B, indique sur ses sorties A > B, A = B et A < B le résultat de la comparaison des nombres A et B.

4. Décodeurs

Un décodeur est un circuit à n entrées d'adresses et 2^n sorties dont une seule est active à la fois, son rang étant déterminé par la valeur binaire matérialisé par l'état des n entrées. Ce circuit s'utilise pour choisir un seul élément à la fois parmi 2^n.

Table de vérité d'un décodeur à deux entrées et quatre sorties (active à 1) :

Entrées d'adresses		Sorties			
B (2^1)	A (2^0)	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Décodeur 1 parmi 16 MSI : 74HC154 :

L'élément 74HC154 est un exemple type de décodeur MSI. Son symbole logique est illustré à la figure ci-dessous. Ce composant est muni d'une fonction de validation (VAL), qui comprend une porte NON-OU utilisée en mode ET négatif. Il faut appliquer un niveau Bas à chaque entrée de validation $\overline{CS}_1$ et $\overline{CS}_2$ du circuit intégré pour que la porte de validation (VAL) produise un niveau Haut à sa sortie.

Symbole logique :

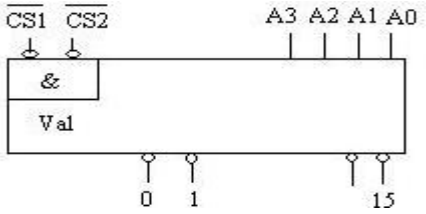
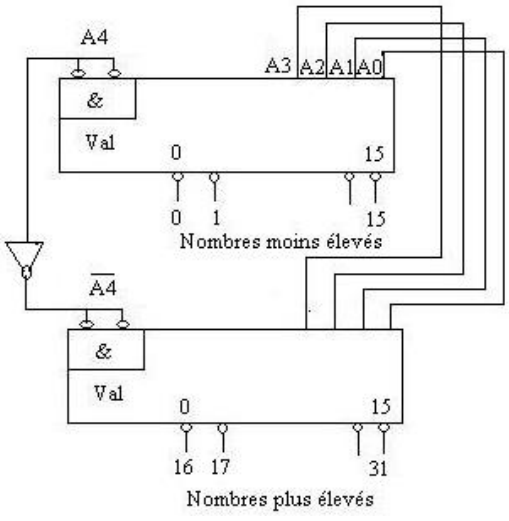


Table de vérité :

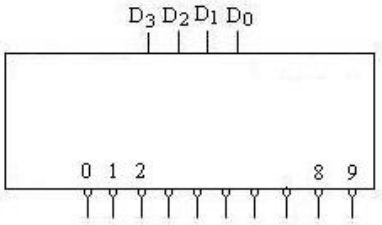
Entrées						Sorties															
$\overline{CS_1}$	$\overline{CS_2}$	A_3	A_2	A_1	A_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Exemple : décodage d'un nombre de 5 bits représenté par le format : $A_4A_3A_2A_1A_0$



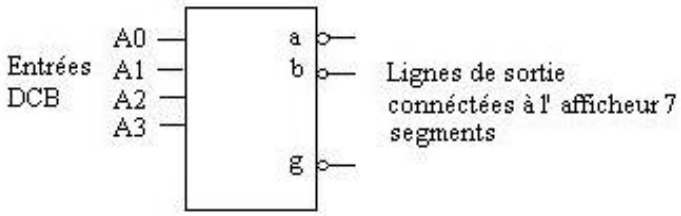
Décodeur DCB-décimal :

C'est un décodeur 4 lignes d'entrée et 10 lignes de sortie ou décodeur 1 parmi 10
Exemple : 74HC42 est un décodeur DCB décimale.

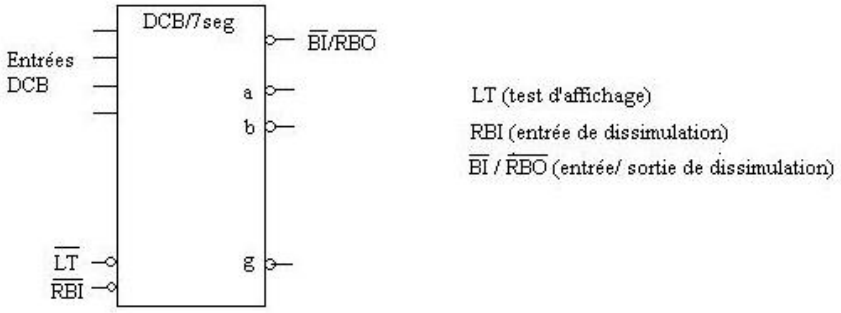


Décodeur DCB- 7 segments :

Le décodeur DCB-7 segment reçoit un code DCB à ses entrées et produit des sorties pour piloter des afficheurs à 7 segments, afin d'obtenir un affichage décimal.



Exemple : 74LS47 est un exemple de composant MSI qui permet de décoder une entrée DCB et de piloter un afficheur à 7 segments.



5. Codeurs

Un codeur est un circuit logique combinatoire effectuant la fonction inverse du décodeur. C'est un circuit à 2n entrées dont une seule est active et qui délivre sur n sorties le numéro codé de cette entrée.

Codeur décimal-DCB :

Ce type de codeur possède 10 entrées (un pour chaque chiffre décimal) et 4 sorties qui correspondent au code DCB. Il s'agit d'un codeur 10 lignes sorties 4 lignes de base.
Le numéro de l'entrée active est codé en DCB sur 4 bits, sa table de vérité est la suivante :

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	S_0	S_1	S_2	S_3
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

L'état de sortie pour la valeur 0 codée par 0000 s'obtient dans le cas où aucune entrée n'est activée et dans le cas où l'entrée E0 qui est activée, alors pour faire la différence entre les deux, on ajoute un signal supplémentaire de contrôle qui est active à 1 si l'une des 10 entrées passe au niveau 1.

Codeur de priorité :

C'est un codeur qui produit la même fonction de codage de base discuté précédemment. Un codeur muni d'une fonction de priorité produit une sortie binaire ou DCB qui correspond à l'entrée du chiffre décimal le plus élevé d'état valide et ignore toutes les autres entrées valides des valeurs inférieures.

Exemple : Codeur de priorité 74F148 (8 vers 3).

C'est un codeur possédant 8 entrées d'état valide Bas et 3 sorties binaires d'état valide Bas. Il possède une entrée de validation qui est active au niveau Bas. Le circuit comporte aussi une sortie de validation E0 utilisée comme extension dans certains montages et une sortie GS qui indique qu'au moins une entrée est valide.

Symbole logique :

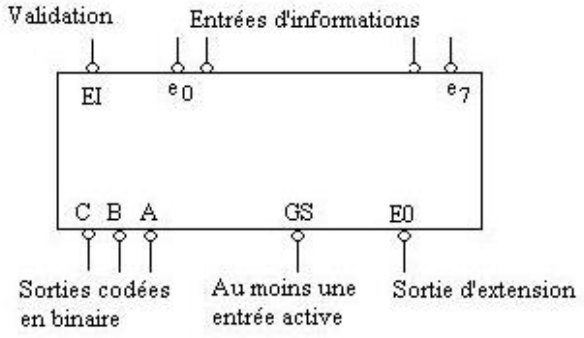


Table de vérité :

Entées									Sorties				
EI	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	C	B	A	GS	E0
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	1	1	0	1
0	x	x	x	x	x	0	1	1	0	0	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	0	1	1	1	1	1	1	1	1	0	1
0	x	0	1	1	1	1	1	1	1	0	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

Codeur entrée 16 lignes sortie 4 lignes à partir de deux codeurs 74F148 :

6. Multiplexeurs

Un multiplexeur est un composant permettant d'acheminer les informations numériques de plusieurs sources sur une seule ligne, afin de les transmettre vers une destination commune. C'est un circuit à 2^n entrées d'information, n entrées d'adresses et une sortie. L'entrée d'information sélectionnée à la sortie est celle dont le rang correspond à l'équivalent binaire de l'adresse sélectionnée.

Exemple : multiplexeur 1 parmi 4

Symbole logique :

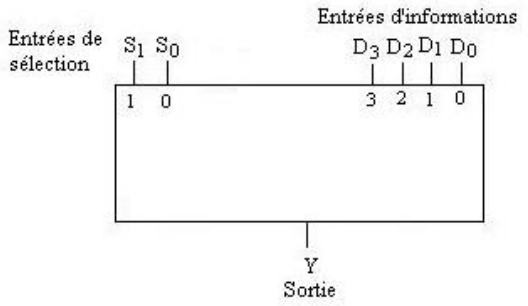


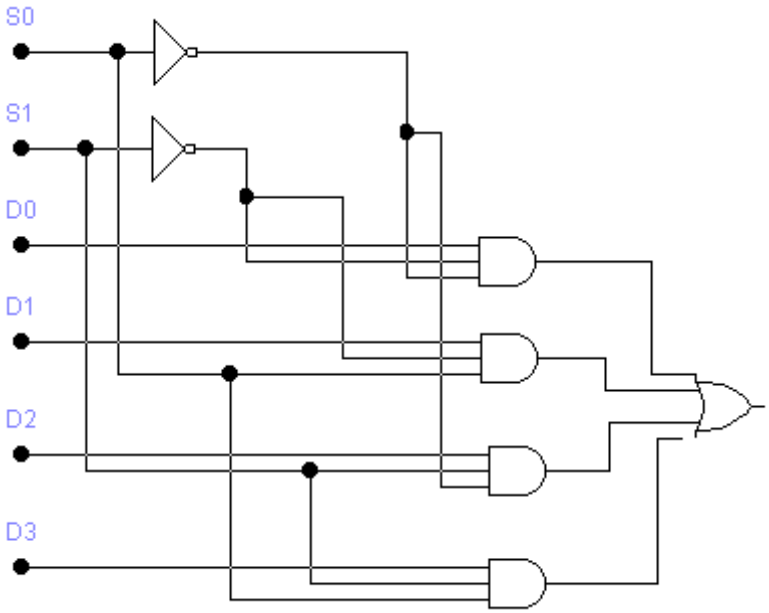
Table de vérité :

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

Expression logique :

$$Y = D_0\bar{S}_0\bar{S}_1 + D_1\bar{S}_1S_0 + D_2S_1\bar{S}_0 + D_3S_1S_0$$

Logigramme :


S0
S1
D0
D1
D2
D3