الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الآلية و الكهروتقنية
Département d'Automatique et Electrotechnique

Polycopie de cours UEF 2.2.1

Intitulé

Logique combinatoire et séquentielle

Préparé par : Dr KHELIFI OTMANE Khelifa

Année Universitaire 2024-2025

Ce cours est destiné aux étudiants en deuxième année licence Electrotechnique.

SOMMAIRE

Introduction

La logique combinatoire et séquentielle est l'élément de base de la réalisation des circuits numériques dans différents domaines industriels. Le plus courant est le domaine de l'informatique qui s'appuie fortement sur les circuits logiques (processeur, mémoires, registres) et les circuits séquentiels (bascules). Également, la logique combinatoire et séquentielle est omni présente dans de la robotique, dans l'électronique numérique, et dans le domaine de la commande des processus industriels. Par conséquent, l'étude de ce module est très nécessaire pour le cursus des étudiants de la spécialité électrotechnique.

Ce cours permet à l'étudiant de saisir les notions de base sur la conception des applications combinatoires et séquentielles. A l'issue de ce cours, il acquerra les différents outils utilisés pour concevoir et réaliser des circuits et applications logiques combinatoires et séquentielles qui lui serviront par la suite dans son domaine de spécialité (commande de processus, contrôle

industriel.), à savoir la table de vérité, le tableau de Karnaugh et les différentes mémoires comme les bascules, les compteurs, les registres.

La présente polycopie est réalisée conformément au canevas destiné aux étudiants de la deuxième année licence électrotechnique, il constitue un manuel de cours et quelques exercices et exemples pour chaque chapitre. Le polycopie est réparti en quatre chapitres:

Le premier chapitre comporte les systèmes de numération. Il aborde les bases de numération ainsi que les conversions entre bases et les opérations arithmétiques sur la base binaire et ainsi la représentation et codage des informations.

Le deuxième chapitre est consacré à l'étude d'algèbre de Boole et simplification des fonctions logiques.

Le troisième et chapitre illustre les bases fondamentales sur les circuits combinatoires et séquentielle.

Le quatrième chapitre est réservé pour exposer les éléments de bases des circuits logiques séquentiels (bascules, compteurs registres).

Chapitre 1 Systèmes de numération et Codage de l'information

Introduction

Dans ce chapitre, on va présenter les différents codes ainsi que les opérations arithmétiques de base nécessaires. Nous introduisons tout d'abord les systèmes de numération puis le codage des nombres ainsi que les opérations de base dans les calculateurs.

La définition d'un système pondéré repose sur les trois notions de **:**

Base du système**,**

Digit du système

Poids du digit selon son rang.

La base d'un système est un nombre entier quelconque soit *B*.

Les **digits** d'un système sont des caractères tous différents représentant chacun un élément de la base : $\alpha, \beta, \gamma, \delta$

L'écriture d'un nombre consiste à associer plusieurs digits dans un ordre déterminé.

Par exemple : $N= \alpha\ \beta\ \gamma\ \delta$

Chaque digit intervient avec un poids différent selon son rang.

Ce poids est de :

B_0 pour le 1^{er} digit ;

B_1 pour le $2^{ème}$ digit ;

B_2 pour le $3^{ème}$digit; ……

B_{n-1} pour le digit de rang n.

Le nombre *N* exprimé dans le système de base *B* vaut :

$\boldsymbol{N}= \delta * B^0 + \gamma * B^1 + \beta * B^2 + \alpha * B^3$

Dans un système de numération de base *b* ; tout nombre est représenté par une suite de chiffres allant de 0 à b-1. Les chiffres sont appelés des digits, la position de chacun représente une puissance entière (positive ou négative) de la base b ; la place occupée par le digit dans un nombre représente son rang.

Par exemple :

$N_b = a_n b^n + a_{n-1} b^{n-1} + \ldots\ldots + a_0 b^0 + a_{-1} b^{-1} + \ldots\ldots + a_{-m} b^{-m}$

Tel que : b^i ($i=n,\ldots\ldots,0, \ldots\ldots\ i=m$) : sont les puissances successives de la base *b* .

b=base et a_i : digit

On défini :

-Base : c'est le nombre de symboles distincts pour lesquels on peut réaliser n'importe quelle quantité, ces symboles sont représentés par des chiffres ou des lettres.

-Nombre : représentation d'une information dans un système de numération par l'association de chiffres. Exemple : 2035 : association de chiffres 2.0.3.5

-Digit : mot anglais désignant un chiffre ou une lettre quelconque soit la base.

-Bit : mot anglais désignant un chiffre binaire : représente 1 ou 0 en numération binaire.

1 Bases de numération

Les ordinateurs ne font pas le calcul en décimal (*base 10)*. Les êtres humains ont toujours travaillé avec le système décimal. Ce dernier est malheureusement difficile à adapter aux systèmes numériques, car il est difficile de concevoir du matériel électronique fonctionnant sur dix plages de tensions différentes. Le système de numération traitant des nombres binaires est appelé système binaire ou système à base 2. Ce système comporte deux digits **0** et **1**. Les chiffres binaires sont aussi appelés bits (**bi**nary digi**t**). Dans les circuits électroniques numériques, un bit 0 est représenté physiquement par une tension basse (LOW) et un bit 1, par une tension haute (HIGH).

Le système de numération décrit la façon avec laquelle les informations sont codées pour qu'elles soient facilement manipulées en machine. Autrement dit, cela revient à établir une correspondance pour passer sans ambigüité d'une représentation externe d'une information a une autre représentation interne (sous forme binaire) de la même information, suivant un ensemble de règles précises pour pouvoir passer d'une base à une autre.

1.1 Base décimale

Le système décimal est le système de numération le plus utilisé. Dans ce système, on dispose de dix symboles (chiffres) différents de 0 à 9 pour écrire tous les nombres.

Exemple 1.1

Soit le nombre X = 2025 nous obtenons le tableau suivant :

Digits	2	0	2	5
Rangs	3	2	1	0
Puissance	10^3	10^2	10^1	10^0
Pondération	2×10^3	0×10^2	2×10^1	5×10^0

La somme des pondérations donne :

Σ Pondérations = $2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 = (2025)_{10}$

Exemple 1.2

Soit un nombre décimal $X = (5723)_{10}$. Ce nombre est la somme de 3 unités, 2 dizaines, 7 centaines et 5 milliers.

$$X = (5 \times 1000) + (7 \times 100) + (2 \times 10) + (3 \times 1)$$

$$X = (5 \times \mathbf{10}^3) + (7 \times \mathbf{10}^2) + (2 \times \mathbf{10}^1) + (3 \times \mathbf{10}^0)$$

Ou **10** représente la base et les puissances de 0 à 3 représentent le rang de chaque chiffre. Quel que soit la base, le chiffre de droite est celui des unités, et le chiffre de gauche est celui qui a le poids le plus élevé.

D'une manière générale, toute base *N* est composée de *N* symboles de *0* à *N-1*.

1.2 Base binaire

C'est la base utilisée en informatique pour la représentation des informations au niveau machine. Ce système possède deux symboles : **0** et **1**. Ces deux états sont les seuls que la machine peut assimiler. Le chiffre binaire qui peut prendre ces deux états est nommé **bit,** on peut coder deux états avec un bit. Avec deux bits nous pouvons coder quatre états, et avec trois bits on peut coder huit états.

Une suite de huit bits est nommée un **octet** (byte). Avec un octet, on peut écrire 2^8 = 256 nombres binaires (de 0 à 255).

Exemple 1.3

Soit le nombre binaires $(1101)_2$

Bits	1	1	0	1
Rangs	3	2	1	0
Puissance	2^3	2^2	2^1	2^0
Pondération	1×2^3	1×2^2	0×2^1	1×2^0

La somme des pondérations donne :

Σ Pondérations = $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$

Exemple 1.4

Le nombre X= 1011 exprimé en binaire signifie :

X== $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (11)_{10}$

En Général:

Par notation on écrit un nombre binaire *A*.

$(A)_2 = a_{n-1}.a_{n-2}.a_{n-3}\ldots\ldots a_1.a_0$.

a_i prend *0 ou 1*

Soit le nombre *A*= (101.11) exprimé en binaire signifie :

Bits	1	1	1	1	1
Rangs	2	1	0	-1	-2
Puissance	2^2	2^1	2^0	2^{-1}	2^{-2}
Pondération	1×2^2	1×2^1	1×2^0	1×2^{-1}	1×2^{-2}

L'addition des pondérations donne l'équivalent décimal du nombre binaire considéré.

Le nombre *A*= $1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

A=4+2+1+0.5+0.25=7.75

D'où *A*= $(111.11)_2 = (7.75)_{10}$.

1.3 Base octale

Le système octale (base = 8) utilise huit chiffres {0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7} utilisé en informatique il y'a un certain temps. Il permet de coder 3 bits par un seul symbole.

Exemple 1.5:

Le nombre 324 exprimé en octal signifie :

$$324 = 3x8^2+2x8^1+4x8^0.$$
$$= 3x64+2x8+4x1$$
$$=192+16+4=212$$
$$\text{D'où } (324)_8 =(212)_{10}$$

1.4 Base Hexadécimal

Le système hexadécimal (base = 16) utilise 16 chiffres : {0 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; A ; B ; C ; D ; E ; F}, tel que A = 10 ; B = 11 ; C = 12 ; D = 13 ; E = 14 ; F = 15. La plupart des systèmes numériques traitent des données binaires en groupes qui sont des multiples de quatre bits, rendant le nombre hexadécimal très pratique car chaque chiffre hexadécimal représente un nombre binaire de 4 bits.

Le système de numération hexadécimal est souvent utilisé dans un système numérique comme une sorte réduction pour représenter des chaînes de bits. La représentation hexadécimale est utilisée pour les adresses et le contenu des emplacements de mémoire dans la mémoire principale d'un ordinateur. Le système de numération hexadécimal fournit un moyen de représentation condensée de grands nombres binaires stockées et traitées dans l'ordinateur. De même, le contenu de la mémoire lorsqu'il est représenté sous forme hexadécimale est très facile à manipuler. Un tel exemple est de représenter les adresses des différents emplacements de mémoire.

Par exemple, on suppose qu'une machine a 64 Ko de mémoire. Une telle mémoire a 64K octet (= 216 = 65 536) emplacements de mémoire et besoin de 65 536 adresses différentes. Alors ces adresses peuvent être désignées comme étant de 0 à 65 535 dans le système de numération décimale et de 00000000 00000000 à 11111111 11111111 dans le système de numération binaire. Le système décimal n'est pas utilisé dans les ordinateurs et la notation binaire apparaît ici trop lourde et peu pratique à manipuler. Dans le système de numération hexadécimal, les 65 536 adresses différentes peuvent être exprimées avec quatre symboles de 0000 à FFFF.

Exemple 1.6: Le nombre 3DA9 exprimé en hexadécimal signifie comme suit dans le tableau suivant:

Digits (chiffres)	3	D	A	9
Rangs	3	2	1	0
Puissance	16^3	16^2	16^1	16^0
Pondération	3×16^3	13×16^2	10×16^1	9×16^0

La somme des pondérations donne :

Σ Pondérations = $3 \times 16^3 + 13 \times 16^2 + 10 \times 16^1 + 9 \times 16^0$

= 12288+3328+160+9

= $(15785)_{10}$

1.5 Représentation signée des nombres binaires

Par convention, le MSB (bit de poids fort) est interprété comme bit de signe. Le nombre est négatif lorsque le bit de signe est égal à 1. Cependant pour conserver les propriétés de calcul, il est nécessaire d'utiliser la notation en complément à 2. Dans la notation en complément à 2, les nombres positifs sont représentés sans changement par rapport à l'écriture binaire. En revanche, les nombres négatifs sont obtenus de la manière suivante :

– on inverse les bits de l'écriture binaire de la valeur absolue ;

– on ajoute 1 au résultat (les dépassements sont ignorés).

La même opération effectuée sur un nombre négatif permet de retrouver le nombre positif de départ.

2 Changement de base, conversions.

La conversion consiste le passage d'une base vers une autre en respectant des règles pour garder et maintenir la cohérence des informations codées.

Il existe trois types de conversion :

- Conversion du système décimal en un autre système: cette opération s'appelle **le codage.**
- Conversion d'un système autre que le décimal en un système décimal: cette opération s'appelle **le décodage.**
- Conversion entre deux systèmes non décimaux: cette opération s'appelle **le transcodage.**

2.1 Conversion binaire – décimal

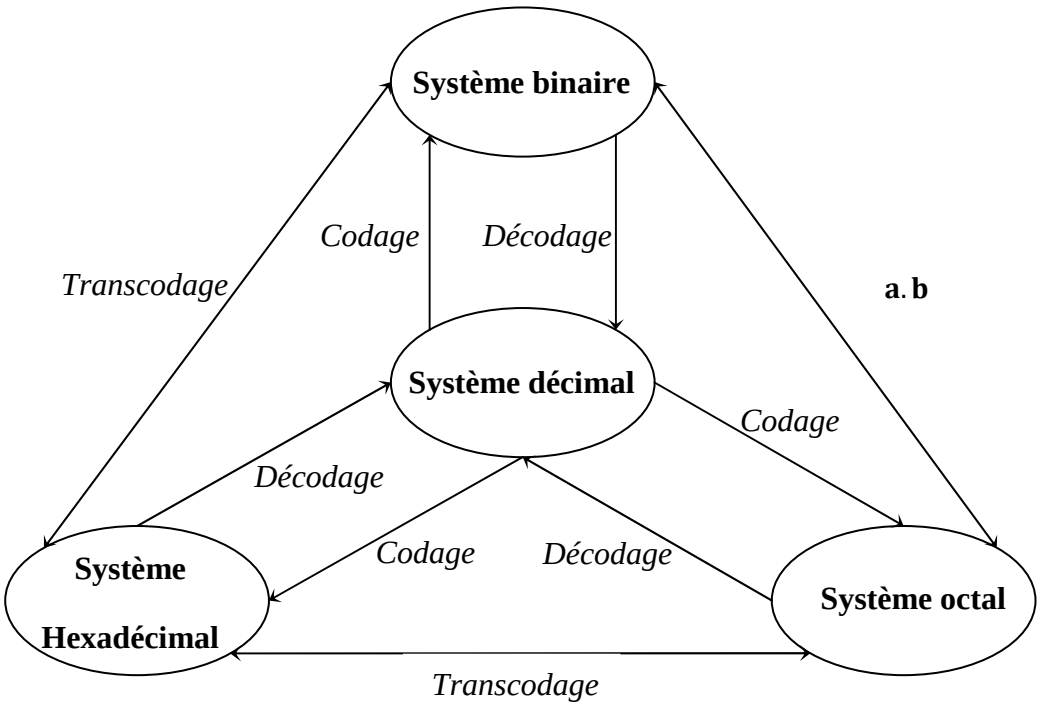
La conversion de la base binaire vers la base 10 se fait par la multiplication de chaque bit par le chiffre **2** élevé à une puissance, croissante, comptée à partir de zéro en partant de la droite, puis on effectue la somme.

Exemple 1.7

Convertir le nombre $(100101)_2$ en décimal :

$(100101)_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5$

$= 1 + 0 + 4 + 0 + 0 + 32 = (37)_{10}$



2.2 Conversion octal – décimal

La conversion de la base octale vers la base 10 se fait par la multiplication de chaque chiffre par **8** élevé à une puissance, croissante, comptée à partir de zéro en partant de la droite, puis on effectue la somme.

Exemple 1.3

Convertir le nombre $(1743)_8$ en décimal :

$(1743)_8 = 3 \times 8^0 + 4 \times 8^1 + 7 \times 8^2 + 1 \times 8^3$

$= 3 + 32 + 448 + 512$

$= (995)_{10}$

2.3 Conversion hexadécimal – décimal

La conversion de la base hexadécimale vers la base 10 se fait par la multiplication de chaque chiffre par **16** élevé à une puissance, croissante, comptée à partir de zéro en partant de la droite, puis on effectue la somme.

Exemple 1.4

Convertir le nombre $(1AF)_{16}$ en décimal :

$(1AF)_{16} = 15 \times 16^0 + 10 \times 16^1 + 1 \times 16^2$

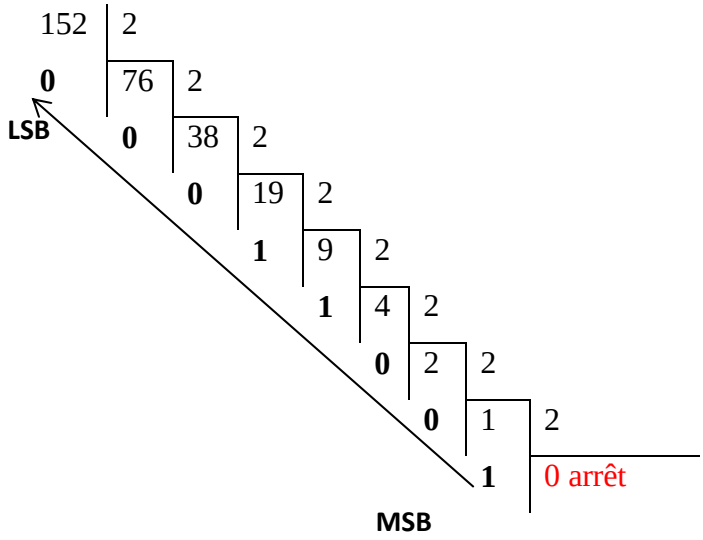
$= 15 + 160 + 256$

$= (431)_{10}$

2.4 Conversion décimal – binaire

Elle consiste à diviser le nombre décimal successivement par **2** jusqu'à obtenir un quotient nul. Ensuite on écrit les restes dans l'ordre inverse de celui dans lequel ils ont été obtenus. Pour les nombres réels, la partie fractionnaire est multiplié par **2** (résultat nul ou selon la précision demandée).

Exemple 1.8 Convertir le nombre $(152)_{10}$ en binaire :



MSB : le bit de poids fort (***M**ost **S**ignificant **B**it*)

LSB : le bit de poids faible (***L**ow **S**ignificant **B**it*)

Le résultat est donc : $(152)_{10} = (10011000)_2$

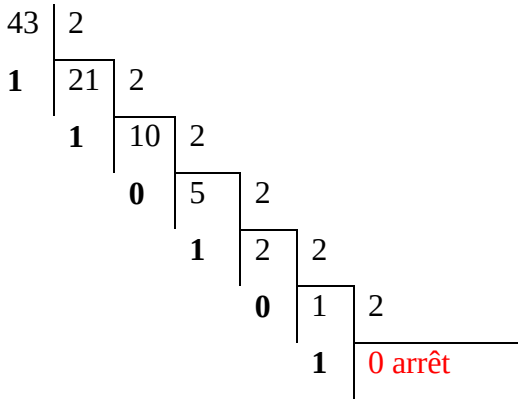
Dans le cas de nombres fractionnaires, on multiplie la partie fractionnaire par 2 :

Si après la multiplication, un 1 apparaît à gauche de la virgule, on ajoute 1 à la fraction binaire en formation.

Si après multiplication, c'est un 0 qui apparaît ; on ajoute un 0.

Exemple 1.9 Convertir le nombre $(43{,}125)_{10}$ en binaire :

- La partie entière PE=43



Donc :$(43)_{10} = (101011)_2$

- La partie fractionnaire PF=0,125

0,125 x 2 = **0**,25

0,25 x 2 = **0**,5

0,5 x 2 = **1**,0 arrêt

$(0,125)_{10} = (0,001)_2$

Donc $(43,125)_{10} = (101011,001)_2$

Dans le cas de nombres fractionnaires, on multiplie la partie fractionnaire par 2 :

Si après la multiplication, un 1 apparaît à gauche de la virgule, on ajoute 1 à la fraction binaire en formation.

Si après multiplication, c'est un 0 qui apparaît ; on ajoute un 0.

2.5 Conversion décimal – octal

Elle consiste à diviser le nombre décimal successivement par **8** jusqu'à obtenir un quotient nul. Ensuite on écrit les restes dans l'ordre inverse de celui dans lequel ils ont été obtenus.

Exemple 1.10

Convertir le nombre $(126)_{10}$ en octal :

126	8
6	15 8
	7 1 8
	1 0 arrêt

Le résultat est donc : $(126)_{10} = (176)_8$

2.6 Conversion décimal – hexadécimal

Elle consiste à diviser le nombre décimal successivement par **16** jusqu'à obtenir un quotient nul. Ensuite on écrit les restes dans l'ordre inverse de celui dans lequel ils ont été obtenus.

Exemple 1.11

Convertir le nombre $(190)_{10}$ en hexadécimal :

190	16
14=E	11 16
	11=B 0 arrêt

Le résultat est donc : $(190)_{10} = (BE)_{16}$

2.7 **Conversion binaire – octal et vice versa**

Les chiffres de système octal sont de zéro à sept, ces chiffres nécessitent trois bits pour être codée en binaire. Donc pour convertir un nombre octal en binaire, il suffit de remplacer chaque chiffre par son équivalant binaire sur trois bits.

Exemple 1.12

$$(64)_8 = (110100)_2$$

De façon inverse, pour convertir un nombre binaire en octal, on regroupe les bits en groupe de trois (03) de droite à gauche pour la partie entier (on ajoute des zéros à gauche si c'est nécessaire) et de gauche à droite pour la partie fractionnaire (on ajoute des zéros à droite si c'est nécessaire), en suite on remplace chaque groupe par son équivalant octal.

Exemple 1.10

$$(10101011)_2 = (010101011)_2$$
$$= (253)_8$$
$$(10011,1)_2 = (010011,100)_2$$
$$= (23,4)_8$$

2.8 **Conversion binaire – hexadécimal et vice versa**

Les chiffres de système hexadécimal sont de **0** à **9** et de **A** à **F**, ces chiffres nécessitent quatre bits pour être codée en binaire. Donc pour convertir un nombre hexadécimal en binaire, il suffit de remplacer chaque chiffre par son équivalant binaire sur quatre bits.

Exemple 1.13

$$(1EC)_{16} = (0001\ 1110\ 1100)_2$$
$$= (111101100)_2$$

De façon inverse, pour convertir un nombre binaire en hexadécimal, on regroupe les bits en groupe de quatre (04) de droite à gauche pour la partie entier (on ajoute des zéros à gauche si c'est nécessaire) et de gauche à droite pour la partie fractionnaire (on ajoute des zéros à droite si c'est nécessaire), en suite on remplace chaque groupe par son équivalant hexadécimal.

Exemple 1.14

$$(110011011)_2 = (0001\ 1001\ 1011)_2$$
$$= (19B)_{16}$$
$$(111010,111)_2 = (0011\ 1010,1110)_2$$
$$= (\mathbf{3A,E})_{16}$$

2.9 Récapitulatif

Le tableau suivant présente la correspondance entre les nombres des différentes bases :

Décimal (base 10)	Binaire (base 2)	Octal (base 8)	Hexadécimal (base 16)
0	0000	0	0
1	0001	1	1
2	0010	3	2
3	0011	4	3
4	0100	5	4
5	0101	6	5
6	0110	7	6
7	0111	10	7
8	1000	11	8
9	1001	12	9
10	1010	13	A
11	1011	14	B
12	1100	15	C
13	1101	16	D
14	1110	17	E
15	1111	20	F

3 Opérations arithmétiques dans le système binaire

Les opérations d'addition, de soustraction, de division et de multiplication dans le système binaire se font de la même manière que dans le système décimal.

3.1 Addition

0+0=0

0+1=1

1+0= 1

1+1=10

1+1+1=11

Exemple 1.15

Effectuer l'opération arithmétique suivante : $(11011+11110)_2$

		1	**1**	**1**		
		1	**1**	**1**	**1**	**0**
+		**1**	**1**	**0**	**1**	**1**
=	**1**	**1**	**1**	**0**	**0**	**1**

Donc $(11011+11110)_2 = (111001)_2$

3.2 Soustraction

0-0=0

0-1=1 (*on retient* 1)

1-0= 1

1-1=0

1-1-1=1(*on retient* 1)

Exemple 1.16

Effectuer l'opération arithmétique suivante : $(11011+11110)_2$

	1	**1**	^{1}**0**	^{1}**0**	^{1}**0**
_	**0**	$_{1}$**0**	$_{1}$**1**	$_{1}$**1**	**1**
=	**1**	**0**	**0**	**0**	**1**

Donc $(111000-111)_2 = (10001)_2$

3.2.1 Soustraction en complément à 2

L'intérêt de la soustraction en complément à 2 est la possibilité de transformer la soustraction en addition. Par définition le complément à 2 d'un nombre binaire c'est le complément bit à bit de ce nombre +1, le 1 étant rajouté au bit le moins significatif.

$$A - B = A + CP_2(B)$$

$$\text{Avec } CP_2(B) = \overline{B} + 1$$

Exemple 1.17

Effectuer la soustraction suivante en utilisant le complément à 2 sur **8 bits** : 9-11

$(9)_{10}=(1001)_2 =(00001001)_2$

$(11)_{10}=(1011)_2=(00001011)_2$

$(-11)_{10} = CP_2(00001011) = CP_1(00001011) +1 =11110100+1= (11110101)_2$

$(9-11)_{10} = (00001001)_2 + (11110101)_2= (11111110)_2$

Le code $(11111110)_2$ correspond au nombre signé $(-2)_{10}$

Vérification :

$(2)_{10}= (00000010)_2$,

$(2-2)_{10} = (00000010-11111110)_2=(\mathbf{1}\ \ 00000000)_2$

Le bit **9** est ignoré donc le résultat = $(00000000)_2$

3.3 Multiplication

0 x 0=0

0 x 1=0

1 x 0= 0

1 x 1=1

Exemple 1.18

Effectuer l'opération arithmétique suivante : $(110 \times 111)_2$

				1	**1**	**0**
x				**1**	**1**	**1**
				1	**1**	**0**
+			**1**	**1**	**0**	**.**
+		**1**	**1**	**0**	**.**	**.**
=	**1**	**0**	**1**	**0**	**1**	**0**

Donc $(110 \times 111)_2= (101010)_2$

3.4 Division

0 / 1=0

1 / 1=1

Exemple 1.19

Effectuer l'opération arithmétique suivante : $(1001 / 11)_2$

	1	**0**	**0**	**1**	**11**	
-		**1**	**1**	⋮	**11**	
	0	**0**	**1**	**1**		
-			**1**	**1**		
=	**0**	**0**	**0**	**0**		

Donc $(1001 / 11)_2= (11)_2$

N.B: Les arithmétiques octal et hexadécimal se font de la même façon que dans les arithmétiques d´décimal et binaire.

4 Codage des nombres

Le codage de l'information est nécessaire pour le traitement informatique de celui-ci. Parmi les codes les plus rencontrés, autre que le code binaire naturel on cite le code BCD, le code GRAY, et le code ASCII

4.1 Code BCD (Binary Coded Decimal)

Sa propriété est d'associer 4 bits représentent chaque chiffre en binaire naturel. L'application la plus courante est celle de l'affichage numérique ou chaque chiffre est associé à un groupe de 4 bits portant le code BCD.

Décimal	0	1	2	3	4	5	6	7	8	9
Code BCD	0000	0001	0010	0011	0100	0101	0110	0011	1000	1001

Exemple 1.18

Trouver le code BCD du nombre 93

$(93)_{10} = (1001\ 0011)_{BCD}$

Ce codage est utilisé dans les horloges numériques, les thermomètres numériques, les compteurs numériques et d'autres appareils à sept segments utilisent généralement le code BCD pour simplifier l'affichage des nombres décimaux. Le BCD n'est pas aussi efficace que le binaire pur pour les calculs, mais il est particulièrement utile si un traitement limité est nécessaire, comme dans un thermomètre numérique.

4.2 Code Gray (binaire réfléchi)

Ce code ne permet que la représentation des chiffres décimaux à chaque augmentation d'une unité du chiffre décimal. Un seul bit du nombre binaire équivalent change de valeur par rapport au nombre binaire précédent. Et deux Gray symétriques par rapport à un axe de symétrie se diffèrent par l'état d'un seul bit.

Le code gray offre multiples utilisations :

- Le code Gray est utilisé dans les codeurs de position angulaire pour éliminer le problème d'erreur inhérent au code binaire. Le code Gray garantit qu'un seul bit changera entre les secteurs adjacents.
- Le code gray est utilisé dans certains convertisseurs analogique/numérique et dans des circuits d'entrée/sortie.
- Le code Gray est utilisé pour étiqueter les axes des tables de Karnaugh, une technique graphique utilisée pour la minimisation des expressions booléennes.

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

- Le code Gray est utilisé dans la transmission de signaux numériques car il minimise l'apparition d'erreurs.
- Le code Gray est préféré au code binaire direct dans les appareils de mesure d'angle. L'utilisation du code Gray élimine presque la possibilité d'un angle mal lu, ce qui est probable si l'angle est représenté en binaire naturel. La propriété cyclique du code Gray est un plus dans cette application.
- L'utilisation de codes Gray pour traiter la mémoire de programme dans les ordinateurs minimise la consommation d'énergie. Cela est dû au nombre inferieur de lignes d'adresse changeant d'état avec le progrès du compteur de programme.
- Les codes de Gray sont également très utiles dans les algorithmes génétiques puisque les mutations dans le code, en général permettent des changements incrémentales. Cependant, parfois un changement d'un bit peut entraîner un grand saut, conduisant ainsi à de nouvelles propriétés.

4.2.1 Méthode de conversion Binaire – Gray

- Le bit de gauche du mot binaire reste inchangé.
- Additionner le bit de poids le plus significatif au bit suivant et reporter le résultat sans tenir compte de la retenue.
- Continuer l'addition de chaque bit avec le bit suivant jusqu'au bit du pois le plus faible.

Exemple 1.19

Trouver le code Gray du nombre $(11010011)_2$

Code binaire	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
Code Gray	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**

$(11010011)_2 = (10111010)_{Gray}$

4.3 ASCII (American standard code for information interchange)

Le code ASII est un code alphanumérique. Les lettres, les chiffres, les signes sont représentés par des combinaisons binaires. Ce code est constitué de 8 bits (octet) dont un, celui de gauche, est en général un bit de parité. Ce code permet donc de représenter 256 caractères.

Exemple 1.20

Caractère 'A' = $(65)_{ASCII} = (01000001)_2 = (41)_{16}$

Caractère 'B' = $(66)_{ASCII} = (01000010)_2 = (42)_{16}$

Caractère 'Z' = $(90)_{ASCII} = (01011010)_2 = (5A)_{16}$

Caractère 'a' = $(97)_{ASCII} = (01100001)_2 = (61)_{16}$

Caractère '0' = $(48)_{ASCII} = (00110000)_2 = (30)_{16}$

Caractère '[' = $(91)_{ASCII} = (01011011)_2 = (5B)_{16}$

Tableau des codes ASCII décimaux

0	NUL	44	,	88	X	132	ä	176	▒	220	▄
1	SOH	45	-	89	Y	133	à	177	▒	221	¦
2	STX	46	.	90	Z	134	å	178	▓	222	Ì
3	ETX	47	/	91	[	135	ç	179	│	223	▀
4	EOT	48	0	92	\	136	ê	180	┤	224	Ó
5	ENQ	49	1	93	]	137	e	181	Á	225	ß
6	ACK	50	2	94	^	138	è	182	Â	226	Ô
7	BEL	51	3	95	_	139	ï	183	À	227	Ò
8	BS	52	4	96	`	140	î	184	©	228	õ
9	TAB	53	5	97	a	141	ì	185	╣	229	Õ
10	LF	54	6	98	b	142	Ä	186	║	230	µ
11	VT	55	7	99	c	143	Å	187	╗	231	þ
12	FF	56	8	100	d	144	É	188	╝	232	Þ
13	CR	57	9	101	e	145	æ	189	¢	233	Ú
14	SO	58	:	102	f	146	Æ	190	¥	234	Û
15	SI	59	;	103	g	147	ô	191	┐	235	Ù
16	DLE	60	<	104	h	148	ö	192	└	236	ý
17	DC1	61	=	105	i	149	ò	193	┴	237	Ý
18	DC2	62	>	106	j	150	û	194	┬	238	¯
19	DC3	63	?	107	k	151	ù	195	├	239	´
20	DC4	64	@	108	l	152	ÿ	196	─	240	-
21	NAK	65	A	109	m	153	Ö	197	┼	241	±
22	SYN	66	B	110	n	154	Ü	198	ã	242	‗
23	ETB	67	C	111	o	155	ø	199	Ã	243	¾
24	CAN	68	D	112	p	156	£	200	╚	244	¶
25	EM	69	E	113	q	157	Ø	201	╔	245	§
26	SUB	70	F	114	r	158	×	202	╩	246	÷
27	ESC	71	G	115	s	159	ƒ	203	╦	247	¸
28	FS	72	H	116	t	160	á	204	╠	248	°
29	GS	73	I	117	u	161	í	205	═	249	¨
30	RS	74	J	118	v	162	ó	206	╬	250	·
31	US	75	K	119	w	163	ú	207	¤	251	¹
32	Space	76	L	120	x	164	ñ	208	ð	252	³
33	!	77	M	121	y	165	Ñ	209	Ð	253	²
34	"	78	N	122	z	166	ª	210	Ê	254	■
35	#	79	O	123	{	167	º	211	Ë	255	
36	$	80	P	124	|	168	¿	212	È		
37	%	81	Q	125	}	169	®	213	ı		
38	&	82	R	126	~	170	¬	214	Í		
39	'	83	S	127	DEL	171	½	215	Î		
40	(	84	T	128	Ç	172	¼	216	Ï		
41	)	85	U	129	ü	173	¡	217	┘		
42	*	86	V	130	é	174	«	218	┌		
43	+	87	W	131	â	175	»	219	▒		

4.4 Code polynomiale CRC :

Contrôle de Redondance Cyclique(en anglais CRC) est un code couramment utilisé pour détecter des erreurs de transmission à un ou deux bits lorsque des données numériques sont transférées sur un lien de communication. Le lien de communication peut se trouver entre deux ordinateurs connectés à un réseau ou entre un périphérique de stockage numérique (tel qu'un CD, un DVD ou un disque dur) et un PC. S'il est correctement conçu, le CRC peut également détecter plusieurs erreurs pour un certain

nombre de bits en séquence (paquets d'erreurs). Dans CRC, un certain nombre de bits de contrôle, parfois appelé somme de contrôle, sont ajoutés aux bits de données (ajoutés à la fin) qui sont transmis. Les données transmises sont testées par le récepteur pour détecter les erreurs utilisant le CRC. Toutes les erreurs possibles ne peuvent être identifiées, mais la CRC est beaucoup plus efficace qu'une simple vérification de parité. Le CRC est souvent décrit mathématiquement (Polynomiaux) comme la division de deux polynômes pour générer un reste. Un polynôme est une expression mathématique qui est une somme de termes avec des exposants positifs. Lorsque les coefficients sont limités à 1s et 0s, on appelle un polynôme univarié.

4.5. Code de Hamming

Le code de Hamming est utilisé pour détecter et corriger une erreur d'un bit dans un code transmis. Il permet de transférer quatre bits de données et trois bits de contrôle, et de détecter et corriger une erreur sur les sept bits transmis. Les bits de contrôles sont placés dans le message final aux positions correspondant aux puissances de deux successives. Pour ce faire, quatre bits de redondance sont introduits dans un groupe de 7 bits de données. Ces bits de redondance sont intercalés aux positions de bits 2n (n = 0, 1, 2, 3) dans les bits de données d'origine. À la fin de la transmission, les bits de redondance doivent être retirés des bits de données. Une version récente du code Hamming place tous les bits de redondance à la fin des bits de données, rendant leur suppression plus facile que celle des bits intercalés.

EXERCICES

Exercice 1.1 :

Effectuer les conversions suivantes :

$(39)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16} = (\ldots\ldots)_{BCD} = (\ldots\ldots)_{GRAY}$

$(11101010)_2 = (\ldots\ldots)_{10} = (\ldots\ldots)_8$

$(A1)_{16} = (\ldots\ldots)_{10}$

$(12,25)_{10} = (\ldots\ldots , \ldots\ldots)_2$

Exercice 1.2

Effectuer les conversions suivantes :

$(81)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16} = (\ldots\ldots)_{BCD} = (\ldots\ldots)_{GRAY}$

$(11011011)_2 = (\ldots\ldots)_{10} = (\ldots\ldots)_8$

$(FD)_{16} = (\ldots\ldots)_{10}$

$(46,625)_{10} = (\ldots\ldots , \ldots\ldots)_2$

Exercice 1.3

Effectuer les conversions suivantes : $(103)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16} = (\ldots\ldots)_8$

$(10101010)_2 = (\ldots\ldots)_{10}$

$(123,75)_{10} = (\ldots\ldots , \ldots\ldots)_2$

Effectuer les opérations arithmétiques suivantes :

$(BE+D3)_{16}$,$(11101000100/1010))_2$,

Effectuer l'opération suivante en utilisant le complément à deux sur 8 bits

74 - 75

Exercice 1 .4

Effectuer les conversions suivantes : $(11)_{16} = (\ldots\ldots)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_8$

$(17)_8 = (\ldots\ldots)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16}$

Effectuer les opérations arithmétiques suivantes :

$(AA+56)_{16}$, $(110100 - 10011)_2$, $(1100100/1010)_2$, $(A* 3)_{16}$

Exercice 1.5

Effectuer les opérations arithmétiques suivantes :

$(11101101 + 1010)_2$

$(AB+E7)_{16}$

$(110111/101)_2$

Effectuer les deux opérations suivantes en utilisant le complément à deux sur 8 bits

13 – 7 14 - 19

Exercice 1.6

Effectuer les opérations arithmétiques suivantes :

$(110111001 + 10111)_{2}$

$(F3+F7)_{16}$

$(1001111101/111)_{2}$

Effectuer les deux opérations suivantes en utilisant le complément à deux sur 8 bits

57 – 68 80 – 101

Exercice 1.7

Effectuer les conversions suivantes :

Binaire	**Octal**	**Décimal**	**Hexadécimal**	**GRAY**
101101	…….	…….	…….	…….
…….	**77**	…….	…….	…….
…….	…….	**101**	…….	…….
…….	…….	…….	**3E**	…….

Effectuer les opérations arithmétiques suivantes:

$$\left(\begin{array}{r} EF \\ +\ F9 \\ \hline \end{array}\right)_{16} \qquad \left(\begin{array}{r} 754 \\ -\ 655 \\ \hline \end{array}\right)_{8} \qquad \left(\begin{array}{r} 1110 \\ *\ \ 101 \\ \hline \end{array}\right)_{2}$$

Donner la suite des nombres hexadécimaux entre : **598** et **5B0**

Exercice 1.8

Compléter le tableau suivant :

Binaire	Décimal	Hexadécimal	BCD	GRAY
1100010	…….	…….	…….	…….
…….	**92**	…….	…….	…….
…….	…….	**D3**	…….	…….
…….	…….	…….	**10101**	…….

Effectuer les quatre opérations arithmétiques suivantes:

$(AE+57)_{16}$, $(101100-111)_{2}$, $(100100*11)_{2}$, $(5C/4)_{16}$

Exercice 1.9

Effectuer les conversions suivantes :

Binaire	Octal	Décimal	Hexadécimal	BCD	GRAY
111101	…….	…….	…….	…….	…….
…….	**74**	…….	…….	…….	…….
…….	…….	**50**	…….	…….	…….
…….	…….	…….	**2B**	…….	…….
…….	…….	…….	…….	**101001**	…….

Effectuer les quatre opérations arithmétiques suivantes:

$(AF+58)_{16}$ $(101100-1111)_2$ $(10101*110)_2$ $(54/2)_{16}$

Exercice 1.10

1- Effectuer les conversions suivantes : $(47)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16} = (\ldots\ldots)_8$

$(10110110)_2 = (\ldots\ldots)_{10}$

$(49,5)_{10} = (\ldots\ldots , \ldots\ldots)_2$

2- Effectuer les opérations arithmétiques suivantes :

$(AE+B2)_{16}$, $(10101110/111010)_2$,

3- Effectuer l'opération suivante en utilisant le complément à deux sur 8 bits (**bit de signe inclus**):

32 - 30

Exercice 1.11

- Effectuer les conversions suivantes :

$(94)_{10} = (\ldots\ldots)_2 = (\ldots\ldots)_{16} = (\ldots\ldots)_8$

$(10111110)_2 = (\ldots\ldots)_{10}$

$(78,25)_{10} = (\ldots\ldots , \ldots\ldots)_2$

- Effectuer les opérations arithmétiques suivantes :

$(DF+A1)_{16}$,$(1000100/10001))_2$,

- Effectuer l'opération suivante en utilisant le complément à deux sur 8 bits (**bit de signe inclus**):

43 – 51

SOLUTIONS

Exercice 1.1

$(39)_{10} = (100111)_2 = (27)_{16} = (0011\ 1001)_{BCD} = (110100)_{GRAY}$

$(11101010)_2 = (234)_{10} = (352)_8$

$(A1)_{16} = (161)_{10}$

$(12,25)_{10} = (PE,PF)_2$ tel que PE est la partie Entière, et PF partie Fractionnaire

PE : $(12)_{10}=(1100)_2$

PF : 0 ,25*2 =**0**,5 0,5*2 =**1**,0 donc $(0,25)_{10}=(0,01)_2$

$(12,25)_{10}=(1100,01)_2$

Exercice 1.2

$(81)_{10} = (1010001)_2 = (51)_{16} = (1000\ 0001)_{BCD} = (111\ 1001)_{GRAY}$

$(11011011)_2 = (219)_{10} = (333)_8$

$(FD)_{16} = (253)_{10} = (3331)_4$

$(46,625)_{10} = (PE,PF)_2$ tel que PE est la partie Entière, et PF partie Fractionnaire

PE : $(46)_{10}=(101110)_2$

PF : 0 ,625*2 =**1**,25 0,25*2 =**0**,5 0,5*2 =**1**,0 donc

$(0,625)_{10}=(0,101)_2$

$(46,75)_{10}=(101110,101)_2$

Exercice 1 .3

$(103)_{10} = (1100111)_2 = (67)_{16} = (147)_8$

$(10101010)_2 = (170)_{10}$

$(123,75)_{10} = (PE,PF)_2$ tel que PE est la partie Entière, et PF partie Fractionnaire

PE : $(123)_{10}=(1111011)_2$

PF : 0 ,75*2 =**1**,5 0,5*2 =**1**,0 donc $(0,75)_{10}=(0,11)_2$

$(123,75)_{10}=(1111011,11)_2$

$(BE+D3)_{16} = (191)_{16}$

$(11101000100/1010)_2 = (10111010)_2$

74 – 75 : $(74)_{10} = (\mathbf{0}1001010)_2$ $(75)_{10} = (\mathbf{0}1001011)_2$

Complément à 1 de **0**1001011 est **1**0110100

Complément à 2 est 10110100 +1 =10110101 Donc $(-75)_{10}=(10110101)_2$

$(74-75)_{10}=(\mathbf{0}1001010+10110101)_2=(11111111)_2$

Vérification

Le bit **8** est égale à **1** représente le signe – **.**ce code représente le complément à 2 du module.

le complément à 1 du module est 11111111-1 =11111110 donc le module égale 00000001 qui représente le nombre 1 en décimal **Donc :** $(74-75)_{10}=(\mathbf{1}1111111)_2=(-1)_{10}$

Exercice 1 .4

$(11)_{16}=(17)_{10}=(10001)_2=(21)_8$

$(17)_8=(15)_{10}=(1111)_2=(F)_{16}$

$(AA+56)_{16}\,(10)_{16}$

$(110100-10011)_2=(100001)_2$

$(1100100/1010)_2=(1010)_2$

$(A*3)_{16}=(1E)_{16}$

Exercice 1.5

$(11101101+1010)_2=(11110111)_2$, $(AB+E7)_{16}=(192)_{16}$, $(110111/101)_2=(1011)_2$

13 – 7 : $(13)_{10}=(\mathbf{0}0001101)_2$ $(7)_{10}=(\mathbf{0}0000111)_2$

Complément à 1 de **0**0000111 est **1**1111000

Complément à 2 est 11111000 +1 =11111001 Donc $(-7)_{10}=(11111001)_2$

$(13-7)_{10}=(\mathbf{0}0001101+11111001)_2=(\mathbf{1}\,\mathbf{0}000\,0110)_2$

Le bit **9** est ignoré car le résultat est représenté sur 8 bits

Vérification Le bit **8** est égale à **0** représente le signe **+** et000 0110 représente 6 en décimal

Donc : $(13-7)_{10}=(\mathbf{0}000\,1010)_2=(+6)_{10}$

14 – 19 : $(14)_{10}=(\mathbf{0}0001110)_2$ $(19)_{10}=(\mathbf{0}0010011)_2$

Complément à 1 de **0**0010011 est **1**1101100

Complément à 2 est 11101100 +1 =11101101 Donc $(-19)_{10}=(11101101)_2$

$(14-19)_{10}=(\mathbf{0}0001110+11101101)_2=(\mathbf{1}1111011)_2$

Vérification

Le bit **8** est égale à **1** représente le signe – **.**ce code représente le complément à 2 du module.

le complément à 1 du module est **1**1111011-1 =**1**1111010 donc le module égale 00000101 qui représente 5 en décimal **Donc :** $(14 - 19)_{10}$= $(\mathbf{1}1111011)_{\mathbf{2}}$ = $(-5)_{10}$

Exercice 1.6

$(110111001 + 10111)_{\mathbf{2}}$= $(111010000)_{\mathbf{2}}$, $(F3+F7)_{\mathbf{16}}$= $(1EA)_{\mathbf{16}}$, $(1001111101/111)_{\mathbf{2}}$ = $(1011011)_{\mathbf{2}}$

57 – 68: $(57)_{\mathbf{10}}$ =$(\mathbf{0}0111001)_{\mathbf{2}}$ $(68)_{\mathbf{10}}$ =$(\mathbf{0}1000100)_{\mathbf{2}}$

Complément à 1 de **0**1000100 est **1**0111011

Complément à 2 est **1**0111011 +1 =**1**0111100 Donc $(-68)_{\mathbf{10}}$=$(10111100)_{\mathbf{2}}$

$(57 – 68)_{10}$= $(\mathbf{0}0111001 + 10111100)_{\mathbf{2}}$ = $(\mathbf{1}1110101)_{\mathbf{2}}$

Vérification

Le bit **8** est égale à **1** représente le signe **-.**ce code représente le complément à 2 du module.

le complément à 1 du module est **1**1110101-1 =**1**1110100 donc le module égale 00001011 qui représente 11 en décimal **Donc :** $(57 – 68)_{10}$= $(\mathbf{1}1110101)_{\mathbf{2}}$ = $(-11)_{10}$

80 - 101: $(80)_{\mathbf{10}}$ =$(\mathbf{0}1010000)_{\mathbf{2}}$ $(101)_{\mathbf{10}}$ =$(\mathbf{0}1100101)_{\mathbf{2}}$

Complément à 1 de **0**1100101 est **1**0011010

Complément à 2 est **1**0011010+1 =**1**0011011 Donc $(-101)_{\mathbf{10}}$=$(\mathbf{1}0011011)_{\mathbf{2}}$

$(80 – 101)_{10}$= $(01010000+10011011)_{\mathbf{2}}$ = $(\mathbf{1}1101011)_{\mathbf{2}}$

Vérification

Le bit **8** est égale à **1** représente le signe – **.**ce code représente le complément à 2 du module.

le complément à 1 du module est **1**1101011-1 =**1**1101010 donc le module égale 00010101 qui représente 21 en décimal

Donc : $(80 – 101)_{10}$= $(\mathbf{1}1101011)_{\mathbf{2}}$ = $(-21)_{10}$

Exercice 1.7

Effectuer les conversions suivantes :

Binaire	Octal	Décimal	Hexadécimal	GRAY
101101	55	45	2D	111011
111111	**77**	63	3F	100000
1100101	145	**101**	65	1010111
111110	76	62	**3E**	100001

Effectuer les quatre opérations arithmétiques suivantes:

$$\left(\begin{array}{r} \text{EF} \\ +\ \text{F9} \\ \hline \text{1E8} \end{array} \right)_{16} \qquad \left(\begin{array}{r} 754 \\ -\ 655 \\ \hline 077 \end{array} \right)_{8} \qquad \left(\begin{array}{r} 1110 \\ *\ \ \ 101 \\ \hline 1000110 \end{array} \right)_{2}$$

Donner la suite des nombres hexadécimaux entre : **598** et **5B0**

598 , 599, 59A, 59B, 59C, 59D, 59E, 59F, 5A0, 5A1, 5A2, 5A3, 5A4, 5A5, 5A6, 5A7, 5A8, 5A9, 5AA, 5AB, 5AC, 5AD, 5AE, 5AF, 5B0,

Exercice 1.8

Binaire	Décimal	Hexadécimal	BCD	GRAY
1100010	98	62	10011000	1010011
1011100	**92**	5C	10010010	1110010
11010011	211	**D3**	1000010001	10111010
1111	15	F	**10101**	1000

$(AE+57)_{16} = (105)_{16}$

$(101100-111)_2 = (100101)_2$

$(100100*11)_2 = (1101100)_2$

$(5C/4)_{16} = (17)_{16}$

Exercice 1.9

Binaire	Octal	Décimal	Hexadécimal	BCD	GRAY
111101	75	61	3D	1100001	100011
111100	**74**	60	3C	1100000	100010
110010	62	**50**	32	1010000	101011
101011	53	43	**2B**	1000011	111110
11101	35	29	1D	**101001**	10011

$(AF+58)_{16}= (107)_{16}$

$(101100-1111)_2= (11101)_2$

$(10101*110)_2= (1111110)_2$

$(54/2)_{16}= (2A)_{16}$

Exercice 1.10

$(47)_{10} = (101111)_2 = (2F)_{16}= (57)_8$

$(10110110)_2 = (182)_{10}$

$(49,5)_{10} =(PE,PF)_2$ tel que PE est la Partie Entier, et PF partie Fractionnaire

PE : $(49)_{10}=(110001)_2$

PF :

0 ,5*2 =**1**,0 donc $(0,5)_{10}=(0,1)_2$

$(49,5)_{10}=(110001,1)_2$

$(AE+B2)_{16}= (160)_{16}$

$(10101110/111010)_2= (11)_2$

32 - 30 : $(32)_{10} =(\mathbf{0}0100000)_2$ $(30)_{10} =(\mathbf{0}0011110)_2$

Complément à 1 de 00011110 est 11100001

Complément à 2 est 11100001 +1 =11100010 Donc $(-30)_{10}=(11100010)_2$

$(32 - 30)_{10}= (\mathbf{0}0100000+11100010)_2= (1\mathbf{0}0000010)_2$

Le bit **8**est égale à**0**représente le signe +**.**Le bit **9**est ignoré

Le code**0**0000010représente nombre 2.

Donc : $(32 – 30)_{10}= (\mathbf{0}0000010)_2 = (2)_{10}$

Exercice 1.11

$(94)_{10} = (1011110)_2 = (5E)_{16} = (136)_8$

$(10111110)_2 = (190)_{10}$

$(78,25)_{10} = (PE,PF)_2$ tel que PE est la Partie Entier, et PF partie Fractionnaire

PE : $(78)_{10}=(1001110)_2$

PF : 0 ,25*2 =**0**,5

0 ,5*2 =**1**,0 donc $(0,25)_{10}=(0,01)_2$

$(78,25)_{10}=(1001110,01)_2$

$(DF+A1)_{16}= (180)_{16}$

$(1000100/10001)_2= (100)_2$

43 - 51: $(43)_{10} =(\mathbf{0}0101011)_2$ $(51)_{10} =(\mathbf{0}0110011)_2$

Complément à 1 de **0**0110011 est **1**1001100

Complément à 2 est 11001100+1 =11001101Donc $(-51)_{10}=(11001101)_2$

$(43 – 51)_{10}= (\mathbf{0}0101011+11001101)_2= (11111000)_2$

Vérification

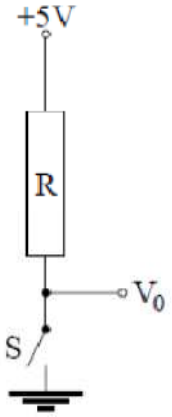
Le bit **8** est égale à**1**représente le signe – **.**ce code représente le complément à 2 du module.

le complément à 1 du module est 11111000-1 =11110111 donc le module égale 00001000 qui représente le nombre 8 en décimal **Donc :** $(43 – 51)_{10}= (\mathbf{1}1111000)_2 = (-8)_{10}$

Chapitre 2 Algèbre de Boole et Simplification des fonctions logiques

Introduction

Les informations traitées par les ordinateurs sont codées sous forme binaire. Un système binaire (signal, circuit, etc…) est un système qui ne peut exister que dans deux états autorisés. Le circuit de la figure suivante est un exemple plus que simpliste de circuit binaire: selon que l'interrupteur S est ouvert ou fermé la tension V_0 ne peut être égale qu'à +5 V ou 0 V.



La réalité technique est un peu plus complexe avec des interrupteurs commandés réalisés par des transistors. Diverses notations peuvent être utilisées pour représenter ces deux états :

Numérique : 1 et 0 (bit : binary digit)

Logique : vrai et faux (true et false), Oui et Non (yes et no)

physique : ouvert et fermé ON et OFF, haut et bas

La logique combinatoire utilise les lois de l'algèbre de Boole développée au XIXème siècle par Georges Boole; philosophe et mathématicien Anglais (1815 -1864). L'algèbre

de Boole permet d'établir des règles pour l'étude des circuits logiques ou des circuits d'automatisme.

Une variable logique ne peut prendre que 02 (deux) valeurs distinctes (les chiffres de la numération binaire 0 et 1).

Exemple : une lampe L est allumée.

Vrai : L=1

Faux : L=0

Deux types de logiques existent :

La logique positive :

État bas (0v) : logique « 0 »

État haut(5v) : logique « 1 »

La logique négative :

État bas (0v) : logique « 1 ».

État haut (5v) : logique « 0 ».

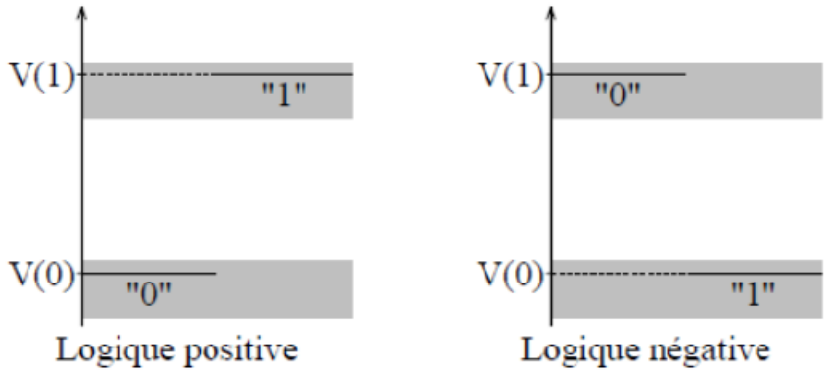
Les valeurs 0 et 1 ne sont des symboles représentant un état électrique et qu'elles n'ont aucune signification numérique.

L'algèbre de Boole concerne la logique des systèmes binaires. Une variable booléenne ne peut prendre que deux valeurs possibles 0 ou 1. En électronique les deux états d'une telle variable peuvent être associés à deux niveaux de tension : V(0) et V(1) pour les états 0 et 1 respectivement. On distingue les logiques positive et négative selon que V(1) > V(0) ou V(1) < V(0).

En pratique un niveau est défini par un domaine en tension ou en courant. Par exemple en technologie TTL, un niveau sera dit haut s'il est compris entre +2 V et +5 V et un niveau sera bas s'il est inférieur à +0.8 V. Dans la plage intermédiaire, l'état est indéterminé. Si les transitions sont inévitables, il est indispensable de traverser cette plage intermédiaire le plus rapidement possible. D'autre part, les signaux doivent être stabilisés avant d'être pris en compte par les circuits. Il y a donc des contraintes temporelles, spécifiées par les chronogrammes des feuilles de données (data sheets) fournis par les constructeurs.

1 Définition

L'algèbre de Boole, ou calcul booléen, est un ensemble de variables à deux états de vérités : 1 (vrai) et 0 (faux), manipuler par un nombre limité d'opérateurs : **et,ou**, **non**. Il contient un ensemble de théorèmes mathématiques qui précisent les fondements théoriques de la logique binaire ou booléenne.



2 Fonctions logiques

Une fonction logique est une expression logique (de valeur 0 ou 1) qui combine un ensemble de variables booléennes à l'aide des opérateurs logiques et, ou, non.

Une fonction logique est principalement présentée par sa table de vérité, sa table de Karnaugh, sa forme canonique, ou par son logigramme.

Une table de vérité est un tableau qui représente des entrées (en colonne) et des états binaires (0 et 1). Le résultat exprimé lui aussi sous forme binaire, se lit dans la dernière colonne.

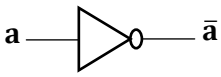
Une fonction de n variables présente 2^n résultats. Donc elle sera décrite par une table de vérité de 2^nlignes. Chaque ligne représente la valeur de la fonction par une combinaison binaire de n variables.

Exemple 2.1

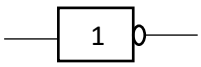
F = f(*a*,*b*,*c*) : est une fonction booléenne qui prend 0 si la majorité des variables vaut 0, et 1 si inversement. Le nombre de variable est n=3 donc le nombre de lignes est 2^3=8 lignes.

c	*b*	*a*	**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

2.1 Fonction complément NON (NOT) (inverseur)



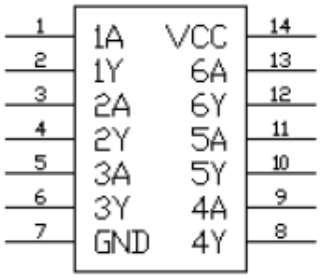
Symbole Américaine



Symbole Européenne

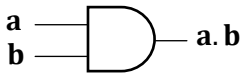
Entrée	***Sortie***
a	ā

Exemple de circuit intégré

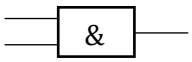


7404

2.2 Fonction produit logique ET (porte AND)



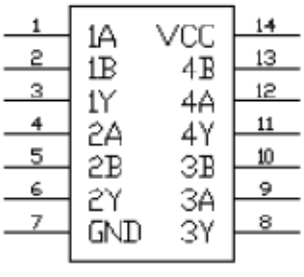
Symbole Américaine



Symbole Européenne

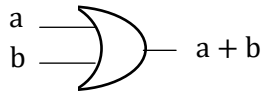
Entrée		***Sortie***
a	b	a. b
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Exemple de circuit intégré

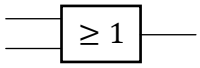


7408

2.3 Fonction somme logique OU (porte OR)



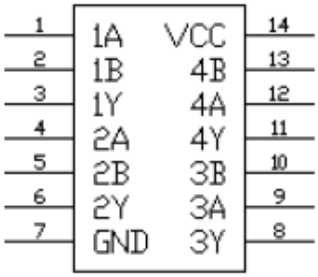
Symbole Américaine



Symbole Européenne

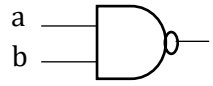
Entrée		***Sortie***
a	b	a + b
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Exemple de circuit intégré

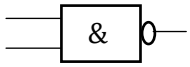


7432

2.4 Port NON-ET (NAND)



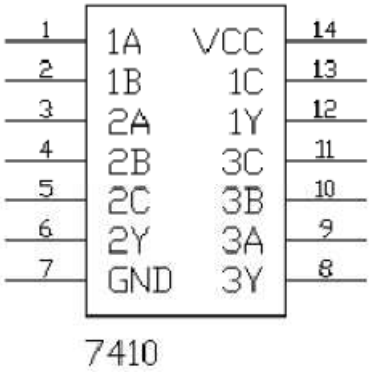
Symbole Américaine



Symbole Européenne

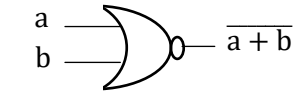
Entrée		***Sortie***
a	b	$\overline{a.b}$
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Exemple de circuit intégré

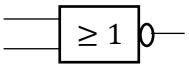


7410

2.5 Porte NON-OU (NOR)



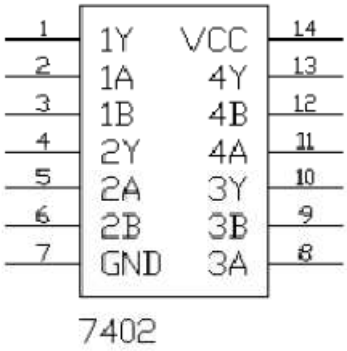
Symbole Américaine



Symbole Européenne

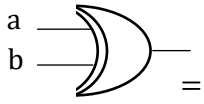
Entrée		***Sortie***
a	b	$\overline{a+b}$
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

Exemple de circuit intégré

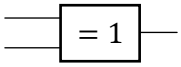


7402

2.6 Porte OU exclusif (XOR)



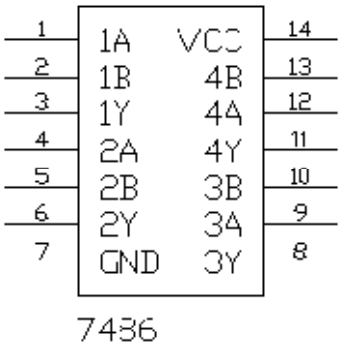
Symbole Américaine



Symbole Européenne

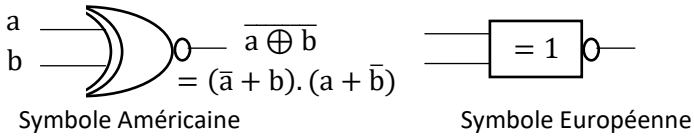
Entrée		***Sortie***
a	b	$a \oplus b$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Exemple de circuit intégré



7486

2.7 Porte NON-OU exclusif (XNOR)



Symbole Américaine Symbole Européenne

Entrée		***Sortie***
a	b	$\overline{a \oplus b}$
0	0	1
0	1	**0**
1	0	**0**
1	1	**1**

3 Propriétés des fonctions logiques de base

Soient a, b et c trois variables logiques.

3.1 Loi de commutativité

$$a.b = b.a$$

$$a + b = b + a$$

3.2 Loi d'associativité

$$(a.b).c = a.(b.c)$$

$$(a + b) + c = a + (b + c)$$

3.3 Loi de distributivité

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

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

3.4 Loi d'impotence

$$a + a = a$$

$$a.a = a$$

3.5 Loi de Complémentarité

$$\bar{a} + a = 1$$

$$\bar{a}.a = 0$$

3.6 Loi de la double négation

$$\bar{\bar{a}} = a$$

3.7 Loi d'absorption

$$a.0 = 0$$

$$a + 1 = 1$$

3.8 Loi de l'élément neutre

$a.1 = a$ 1 élément neutre du **ET**

$a + 0 = a$ 0 élément neutre du **OU**

3.9 Théorème de Morgan

$$\overline{a + b} = \bar{a}.\bar{b} \quad \overline{a + b + c} = \bar{a}.\bar{b}.\bar{c}$$

$$\overline{a.b} = \bar{a} + \bar{b} \quad \overline{a.b.c} = \bar{a} + \bar{b} + \bar{c}$$

4 Quelques relations utiles

En application des règles d'algèbre qui ont été énoncées plus haut, on peut démontrer un certain nombre de relation très utiles.

$$a + (a.b) = a$$

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

$$a.(a + b) = a$$

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

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

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

Ces relations permettent de simplifier l'écriture des fonctions logique.

5 Formes canoniques des fonctions logiques

Les formes canoniques sont basées sur les mintermes et le maxtermes des fonctions logiques. Un minterme est le produit logique des variables de la même ligne de la table de vérité. Un maxterme s'obtient en faisant la somme logique des variables sous forme inversée de la même ligne de la table de vérité.

Soit la table de vérité suivante :

a	**b**	**c**	**F**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

La fonction logique **F** peut être exprimée dans deux formes canoniques.

5.1 Première forme canonique

Elle est donnée par la réunion d'intersection, c'est-à-dire que c'est une Somme de Produits (**SP**)

La fonction F est égale à **1** lorsque :

$$\begin{cases} a = 0\,, b = 0, & c = 0 \\ a = 0\,, b = 0, & c = 1 \\ a = 0\,, b = 1, & c = 0 \\ a = 1\,, b = 0, & c = 0 \end{cases}$$

$$\text{Ou lorsque} \begin{cases} \bar{a}.\bar{b}.\bar{c} = 1 \\ \bar{a}.\bar{b}.c = 1 \\ \bar{a}.b.\bar{c} = 1 \\ a.\bar{b}.\bar{c} = 1 \end{cases}$$

On obtient finalement 04 termes équivalents binaires. La fonction **F** est définie par ses « **1** », on peut donc la représenter par l'équation suivante :

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

Pour ce type d'expression (somme logique de plusieurs produits logiques) : les portes utilisées sont des portes **NAND**. On a :

$$F = \bar{\bar{F}} = \overline{\overline{\bar{a}.\bar{b}.\bar{c} + \bar{a}.\bar{b}.c + \bar{a}.b.\bar{c} + a.\bar{b}.\bar{c}}}$$

La fonction **F** devient donc :

$$F = \overline{\overline{\bar{a}\bar{b}\bar{c}}.\overline{\bar{a}\bar{b}c}.\overline{\bar{a}b\bar{c}}.\overline{a\bar{b}\bar{c}}}$$

5.1 Deuxième forme canonique

Elle est donnée par l'intersection des réunions ; c'est-à-dire un produit de somme (**PS**), tel que à chaque **0** de la variable de sortie, on fait correspondre la somme des variables d'entrée de la même ligne sous **forme inversée**. Par suite, on fait le produit des différentes sommes (maxtermes).

La fonction **F** est égale à **0** pour 04 combinaisons lorsque :

$$\begin{cases} a = 0\,, b = 1, \;\; c = 1 \\ a = 1\,, b = 0, \;\; c = 1 \\ a = 1\,, b = 1, \;\; c = 0 \\ a = 1\,, b = 1, \;\; c = 1 \end{cases}$$

On obtient finalement 04 termes équivalents binaires. La fonction **F** est définie par ses « **0** », on peut donc la représenter par l'équation suivante :

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

Pour ce type d'expression (somme logique de plusieurs produits logiques) : les portes utilisées sont des portes **NOR**. On a :

$$F = \bar{\bar{F}} = \overline{(a + \bar{b} + \bar{c}).(\bar{a} + b + \bar{c}).(\bar{a} + \bar{b} + c).(\bar{a} + \bar{b} + \bar{c})}$$

La fonction **F** devient donc :

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

6 Simplification des fonctions logiques

La simplification est la recherche des termes ou variables inutiles pour satisfaire la fonction recherchée. Autrement dit, simplifier une fonction revient à réduire le nombre de ses termes ou le nombre de variables dans un même terme. L'intérêt de simplifier une fonction logique apparait dans la réalisation du circuit logique qui lui correspond puisque ça réduit le nombre de portes logiques utilisées pour la réalisation.

Exemple 2.2

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

Il existe deux méthodes pour simplifier une équation logique : algébrique et graphique.

6.1 Simplification algébrique

Elle consiste à utiliser les propriétés de l'algèbre de Boole.

Exemple 2.3 Soit la fonction

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

$$F = ab + \bar{a}c + bc.1$$

$$F = ab + \bar{a}c + bc.(\bar{a} + a)$$

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

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

$$F = ab(1 + c) + \bar{a}c(1 + b)$$

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

Exemple 2.4 Soit la fonction

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

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

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

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

$$F = \big((a + b).(a + b) + (a + b).c\big).\big((\bar{a} + c).(\bar{a} + c) + (\bar{a} + c).b\big)$$

$$F = \big((a + b) + (a + b).c\big).\big((\bar{a} + c) + (\bar{a} + c).b\big)$$

$$F = \big((a + b)(1 + c)\big).\big((\bar{a} + c)(1 + b)\big)$$

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

Exemple 2.5 Soit la fonction $F = a\bar{b}d + \bar{a}\bar{b}d + cd + b\bar{c}d$

$$F = a\bar{b}d + \bar{a}\bar{b}d + cd + b\bar{c}d$$

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

$$= \bar{b}d + d\big((c + b)(c + \bar{c})\big)$$

$$= \bar{b}d + d(c + b)$$

$$= d\big(\bar{b} + c + b\big)$$

$$= d(c + 1)$$

$$= d$$

Exemple 2.6 Soit la fonction $F = abc + a\bar{b}\bar{c} + abd + ab\bar{c}$

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

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

$$= ab + a\bar{b}\bar{c} + abd$$

$$= ab(1 + d) + a\bar{b}\bar{c}$$

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

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

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

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

$$= ab + a\bar{c}$$

6.2 **Simplification graphique (méthode de Karnaugh)**

On appelle tableau de Karnaugh une grille comportant un nombre de case égal au nombre de combinaison des variables de la fonction booléenne à étudier ; le tableau est organisé de telle façon qu'une seule variable change entre les cases voisines ; ceci est obtenu en énumérant les vecteurs dans un ordre particulier (code **Gray**).

Le nombre de cases pour *n* variables vaut 2^n, chaque case représente une combinaison des variables (minterme). Ainsi, la table de vérité est transportée dans le tableau en mettant dans chaque case la valeur de la fonction correspondante.

La fonction représentée par un tableau de KARNAUGH s'écrit comme la somme des produits associés aux différentes cases contenant la valeur **1**.

On groupe les « **1** » contenues dans les cases adjacentes pour former des boucles de 1, 2, 4, 8, etc. (2^n avec n = 0, 1, 2, 3, etc.). Tant que le nombre de cases regroupées est grand, l'expression sera plus simplifiée

Exemple 2.5

Soit à simplifier par la table de Karnaugh la fonction F(a,b,c)définie par la table de vérité suivante

a	**b**	**c**	**F**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Correspondant au tableau de Karnaugh suivant :

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

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

Exemple 2.6 Soit à simplifier par la table de Karnaugh la fonction suivante :

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

n	**a**	**b**	**c**	**d**	**F**
0	0	0	0	0	**0**
1	0	0	0	1	**0**
2	0	0	1	0	**0**
3	0	0	1	1	**0**
4	0	1	0	0	**1**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	**0**
8	1	0	0	0	**0**
9	1	0	0	1	**0**
10	1	0	1	0	**0**
11	1	0	1	1	**0**
12	1	1	0	0	**1**
13	1	1	0	1	**1**
14	1	1	1	0	**1**
15	1	1	1	1	**0**

La fonction **F** prend **1** dans les cases correspondant en décimal aux chiffres ***n*** = 4, 5, 6, 12, 13, et 14; qui s'écrit aussi sous la forme (somme des produits) suivante :

$$\mathbf{F} = \sum(4,5,6,12,13,14)$$

La fonction **F** prend **0** dans les cases correspondant en décimal aux chiffres ***n*** = 0, 1, 2, 3, 7, 8, 9, 10, 11, et 15 ; qui s'écrit aussi sous la forme (produit des sommes) suivante :

$$\mathbf{F} = \prod(0,1,2,3,7,8,9,10,11,15)$$

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

$$F = \bar{a}d + \bar{b}d$$

EXERCICES

Exercice 2.1

Simplifier algébriquement les relations logiques suivantes

$a\bar{b}d + \bar{a}\bar{b}d + cd + b\bar{c}d$

$abc + a\bar{b}\bar{c} + abd + ab\bar{c}$

$\bar{a}b\bar{c}d + \bar{a}\bar{b}\bar{c}d + \bar{a}bcd + abcd + \bar{a}bd$

$bd + ab + a\bar{d}$

$a\bar{b}\bar{d} + a\bar{b}d + abd$

$\bar{a}b + a\bar{b} + ab + \bar{a}\bar{b}$

Exercice 2.2

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, où $(\mathbf{n})_{10} = (\mathbf{abcd})_2$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F} = 1$ Lorsque $\mathbf{n} \in \{4,5,7,12,13,15\}$

$\mathbf{F} = 0$ ailleurs

1. Etablir la table de vérité de $\boldsymbol{F}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$

2. Extraire l'équation de F à partir de sa table de vérité.

3. Simplifiez la fonction F en utilisant la méthode de KARNAUGH

4. Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

Exercice 2.3

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, où $(\mathbf{n})_{10} = (\mathbf{abcd})_2$, le bit **a**représente le poids fort (MSB) tel que :

$\mathbf{F} = 1$ Lorsque $\mathbf{n} \in \{0,2,5,7,8,10,13,15\}$

$\mathbf{F} = 0$ ailleurs

1. Etablir la table de vérité de $\mathbf{F}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$

2. Extraire l'équation de F à partir de sa table de vérité.

3. Simplifiez la fonction F en utilisant la méthode de KARNAUGH

4. Tracer le logigramme correspondant en utilisant des portes **NOR** à **deux** entrées.

Exercice 2 .4

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})$, où $(\mathbf{n})_{10} = (\mathbf{abcd})_2$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F} = 1$ Lorsque $\mathbf{n} \in \{1,2,3,8,9,11,14\}$,

$\mathbf{F} = 0$ Lorsque $\mathbf{n} \in \{4,5,7,12\}$,

$\mathbf{F} = \mathbf{Ø}$ Ailleurs

1- Etablir la table de vérité de $\boldsymbol{F}(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d})$

2- Simplifiez la fonction **F** en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

Exercice 2.5

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})$, où $(\mathbf{n})_{10} = (\mathbf{abcd})_2$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F} = 1$ Lorsque $\mathbf{n} \in \{1,2,3,8,9\}$,

$\mathbf{F} = 0$ Lorsque $\mathbf{n} \in \{4,5,7,11,12,13,14,15\}$

$\mathbf{F} = \mathbf{Ø}$ ailleurs

1- Etablir la table de vérité de $\mathbf{F}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})$

2- Simplifiez la fonction **F** en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NOR** à **deux** entrées.

Exercice 2 .6 :

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d})$, où $(\mathrm{n})_{10} = (\mathrm{abcd})_2$, le bit **a** représente le poids fort (MSB)tel que :

$\mathbf{F} = \mathbf{1}$ Lorsque $\mathbf{n} \in \{\mathbf{4},\mathbf{5},\mathbf{12},\mathbf{14}\}$,

$\mathbf{F} = \mathbf{0}$ Lorsque $\mathbf{n} \in \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{8},\mathbf{13},\mathbf{15}\}$,

$\mathbf{F} = \mathbf{Ø}$ Ailleurs

1- Etablir la table de vérité de $\boldsymbol{F}(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d})$

2- Simplifiez la fonction F en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NOR** à **deux** entrées.

Exercice 2 .7:

Soit la fonction logique **F** définie par des séquences binaires **n**formées de **4 bits** $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, où $(\mathbf{n})_{\mathbf{10}} = (\mathbf{abcd})_{\mathbf{2}}$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F = 1}$ Lorsque$\mathbf{n} \in \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{8}, \mathbf{9}, \mathbf{11}, \mathbf{12}, \mathbf{14}\}$,

$\mathbf{F = 0}$ Lorsque$\mathbf{n} \in \{\mathbf{5}, \mathbf{7}\}$,

$\mathbf{F = Ø}$ ailleurs

1- Etablir la table de vérité de $\boldsymbol{F(a, b, c, d)}$

2- Simplifiez la fonction **F** en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

Exercice 2 .8:

Soit la fonction logique **F** définie par des séquences binaires **n** formées de **4 bits** $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, où

$(\mathbf{n})_{\mathbf{10}} = (\mathbf{abcd})_{\mathbf{2}}$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F = 1}$ Lorsque $\mathbf{n} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{6}, \mathbf{10}, \mathbf{11}, \mathbf{12}, \mathbf{14}\}$,

$\mathbf{F = 0}$ Ailleurs

1- Etablir la table de vérité de **F**

2- Simplifiez la fonction **F** en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

Exercice 2 .9:

Soit la fonction logique **F** définie par des séquences binaires **n** formées de **4 bits** $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$, où $(\mathbf{n})_{\mathbf{10}} = (\mathbf{abcd})_{\mathbf{2}}$, le bit **a** représente le poids fort (MSB) tel que :

$\mathbf{F = 1}$ Lorsque $\mathbf{n} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{6}, \mathbf{10}, \mathbf{11}, \mathbf{12}, \mathbf{14}\}$,

$\mathbf{F = 0}$ Ailleurs

1- Etablir la table de vérité de **F**

2- Simplifiez la fonction **F** en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

Exercice 2 .10:

Soit à réaliser un circuit logique qui effectue le **complément à deux** d'un nombre binaire de **4 bits**

Le complément à deux de $A=(a_3a_2a_1a_0)_2$ *est* $B=(b_3b_2b_1b_0)$.

1- Etablir sa table de vérité suivante

2- Simplifiez les fonctions b_0, b_1 , b_2 et b_3 en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant.

Exercice 2 .11:

Soit à étudier un comparateur binaire qui effectue la comparaison de deux nombres binaires ***A*** et ***B*** tel que $\boldsymbol{A}=(a_1a_0)_2$ et $\boldsymbol{B}=(b_1b_0)_2$ les sorties E (égalité), S (supérieur) et I ((inferieur) sont définies comme suit : si $(A=B)\ E=1$ si $(A>B)\ S=1$, si $(A<B)\ I=1$

1- Etablir sa table de vérité suivante

2- Simplifiez les fonctions *E, S* et *I* en utilisant la méthode de KARNAUGH

3- Tracer le logigramme correspondant.

SOLUTIONS

Exercice 2.1 $a\bar{b}d + \bar{a}\bar{b}d + cd + b\bar{c}d = \bar{b}d(a + \bar{a}) + d(c + b\bar{c})$

$$= \bar{b}d + d\big((c + b)(c + \bar{c})\big)$$

$$= \bar{b}d + d(c + b)$$

$$= d\big(\bar{b} + c + b\big)$$

$$= d(c + 1)$$

$$= d$$

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

$$= ab + a\bar{b}\bar{c} + abd$$

$$= ab(1 + d) + a\bar{b}\bar{c}$$

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

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

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

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

$$= ab + a\bar{c}$$

$$\bar{a}b\bar{c}d + \bar{a}\bar{b}\bar{c}d + \bar{a}bcd + abcd + \bar{a}bd = \bar{a}\bar{c}d\big(b + \bar{b}\big) + bcd(a + \bar{a}) + \bar{a}b$$

$$= \bar{a}\bar{c}d + bcd + \bar{a}bd(c + \bar{c})$$

$$= \bar{a}\bar{c}d + bcd + \bar{a}bcd + \bar{a}b\bar{c}d$$

$$= \bar{a}\bar{c}d(1 + b) + bcd(1 + \bar{a})$$

$$= \bar{a}\bar{c}d + bcd$$

$$bd + ab + a\bar{d} = bd + ab\big(d + \bar{d}\big) + a\bar{d}$$

$$= bd + abd + ab\bar{d} + a\bar{d}$$

$$= bd(1 + d) + a\bar{d}(1 + b)$$

$$= bd + a\bar{d}$$

$$a\bar{b}\bar{d} + a\bar{b}d + abd = a\bar{b}\big(\bar{d} + d\big) + abd$$

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

$$= a\big(\bar{b} + bd\big)$$

$$= a\big(\bar{b} + d\big)$$

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

$$\bar{a}b + a\bar{b} + ab + \bar{a}\bar{b} = 1$$

Exercice 2.2

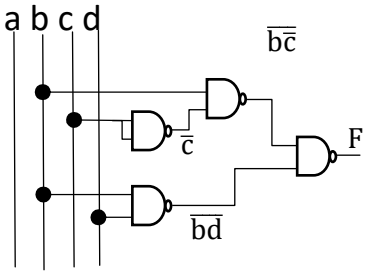
n	a	b	c	d	F
0	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**	**0**
2	**0**	**0**	**1**	**0**	**0**
3	**0**	**0**	**1**	**1**	**0**
4	**0**	**1**	**0**	**0**	**1**
5	**0**	**1**	**0**	**1**	**1**
6	**0**	**1**	**1**	**0**	**0**
7	**0**	**1**	**1**	**1**	**1**
8	**1**	**0**	**0**	**0**	**0**
9	**1**	**0**	**0**	**1**	**0**
10	**1**	**0**	**1**	**0**	**0**
11	**1**	**0**	**1**	**1**	**0**
12	**1**	**1**	**0**	**0**	**1**
13	**1**	**1**	**0**	**1**	**1**
14	**1**	**1**	**1**	**0**	**0**
15	**1**	**1**	**1**	**1**	**1**

$F = \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bcd + ab\bar{c}\bar{d} + ab\bar{c}d + abcd$

ab / **cd**	**00**	**01**	**11**	**10**
00	**0**	**1**	**1**	**0**
01	**0**	**1**	**1**	**0**
11	**0**	**1**	**1**	**0**
10	**0**	**0**	**0**	**0**

$F = b\bar{c} + bd$

$F = \overline{\overline{b\bar{c} + bd}} = \overline{\overline{b\bar{c}}.\overline{bd}}$



Exercice 2.3

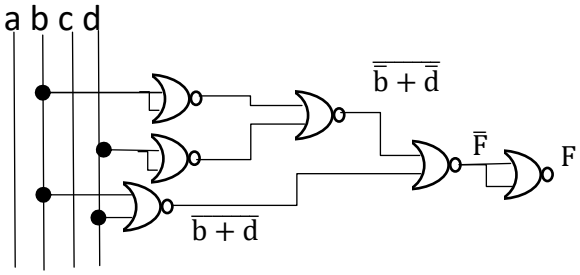
n	a	b	c	d	F
0	**0**	**0**	**0**	**0**	**1**
1	**0**	**0**	**0**	**1**	**0**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**0**
4	**0**	**1**	**0**	**0**	**0**
5	**0**	**1**	**0**	**1**	**1**
6	**0**	**1**	**1**	**0**	**0**
7	**0**	**1**	**1**	**1**	**1**
8	**1**	**0**	**0**	**0**	**1**
9	**1**	**0**	**0**	**1**	**0**
10	**1**	**0**	**1**	**0**	**1**
11	**1**	**0**	**1**	**1**	**0**
12	**1**	**1**	**0**	**0**	**0**
13	**1**	**1**	**0**	**1**	**1**
14	**1**	**1**	**1**	**0**	**0**
15	**1**	**1**	**1**	**1**	**1**

$$F = \bar{a}\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}d + ab\bar{c}d + \bar{a}bcd + abcd + \bar{a}\bar{b}cd + a\bar{b}cd$$

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

$$F = \bar{b}\bar{d} + bd$$

$$F = \overline{\overline{\bar{b}\bar{d}}} + \overline{\overline{bd}} = \overline{\overline{\bar{b}} + \overline{\bar{d}}} + \overline{\overline{b} + \overline{d}} = \overline{b + d} + \overline{\overline{b} + \overline{d}}$$



Exercice 2.4

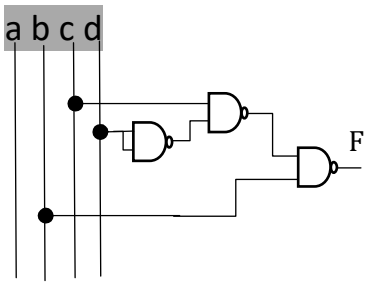
n	a	b	c	d	F
0	**0**	**0**	**0**	**0**	**Ø**
1	**0**	**0**	**0**	**1**	**1**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**1**
4	**0**	**1**	**0**	**0**	**0**
5	**0**	**1**	**0**	**1**	**0**
6	**0**	**1**	**1**	**0**	**Ø**
7	**0**	**1**	**1**	**1**	**0**
8	**1**	**0**	**0**	**0**	**1**
9	**1**	**0**	**0**	**1**	**1**

10	**1**	**0**	**1**	**0**	**Ø**
11	**1**	**0**	**1**	**1**	**1**
12	**1**	**1**	**0**	**0**	**0**
13	**1**	**1**	**0**	**1**	**Ø**
14	**1**	**1**	**1**	**0**	**1**
15	**1**	**1**	**1**	**1**	**Ø**

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

$$F = c\bar{d} + \bar{b}$$

$$F = \overline{\overline{c\bar{d} + \bar{b}}} = \overline{\overline{c\bar{d}}.\,b}$$



Exercice 2.5

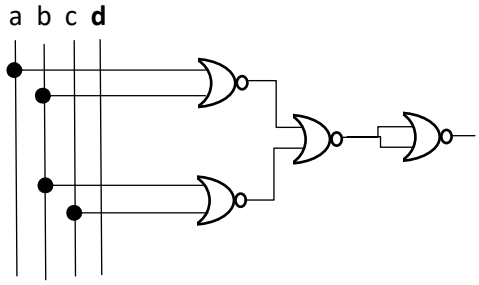
n	a	b	c	d	F
0	**0**	**0**	**0**	**0**	**Ø**
1	**0**	**0**	**0**	**1**	**1**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**1**

4	**0**	**1**	**0**	**0**	**0**
5	**0**	**1**	**0**	**1**	**0**
6	**0**	**1**	**1**	**0**	**Ø**
7	**0**	**1**	**1**	**1**	**0**
8	**1**	**0**	**0**	**0**	**1**
9	**1**	**0**	**0**	**1**	**1**
10	**1**	**0**	**1**	**0**	**Ø**
11	**1**	**0**	**1**	**1**	**0**
12	**1**	**1**	**0**	**0**	**0**
13	**1**	**1**	**0**	**1**	**0**
14	**1**	**1**	**1**	**0**	**0**
15	**1**	**1**	**1**	**1**	**0**

ab / *cd*	*00*	*01*	*11*	*10*
00	**Ø**	**0**	**0**	**1**
01	**1**	**0**	**0**	**1**
11	**1**	**0**	**0**	**0**
10	**1**	**Ø**	**0**	**Ø**

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

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



Exercice 2.6 :

1-table de vérité

n	**a**	**b**	**c**	**d**	**F**
0	**0**	**0**	**0**	**0**	**Ø**
1	**0**	**0**	**0**	**1**	**0**
2	**0**	**0**	**1**	**0**	**0**
3	**0**	**0**	**1**	**1**	**0**
4	**0**	**1**	**0**	**0**	**1**
5	**0**	**1**	**0**	**1**	**1**
6	**0**	**1**	**1**	**0**	**Ø**
7	**0**	**1**	**1**	**1**	**Ø**
8	**1**	**0**	**0**	**0**	**0**
9	**1**	**0**	**0**	**1**	**Ø**
10	**1**	**0**	**1**	**0**	**Ø**
11	**1**	**0**	**1**	**1**	**Ø**
12	**1**	**1**	**0**	**0**	**1**
13	**1**	**1**	**0**	**1**	**0**
14	**1**	**1**	**1**	**0**	**1**
15	**1**	**1**	**1**	**1**	**0**

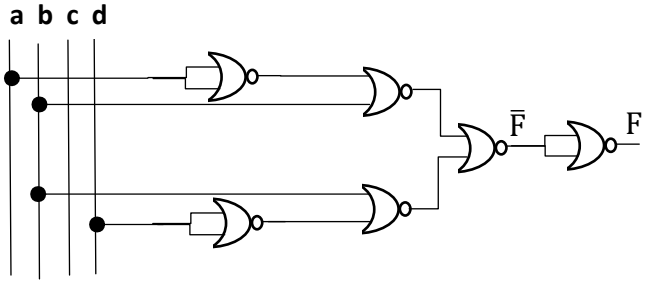
2-simplification par table de Karnaugh

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

$$\mathbf{F} = \bar{a}b + \mathbf{b}\bar{\mathbf{d}}$$

3- $F = \overline{\overline{\bar{a}b}} + \overline{\overline{b\bar{d}}} = \overline{\overline{\bar{a} + b}} + \overline{\overline{b + \bar{d}}}$

$$= \overline{\overline{\bar{a} + b} + \overline{b + \bar{d}}}$$



<u>Exercice 2.7 :</u> 1- table de vérité

n	a	b	c	d	F
0	**0**	**0**	**0**	**0**	**Ø**
1	**0**	**0**	**0**	**1**	**1**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**1**
4	**0**	**1**	**0**	**0**	**1**
5	**0**	**1**	**0**	**1**	**0**
6	**0**	**1**	**1**	**0**	**Ø**

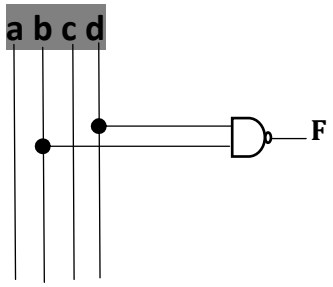
7	**0**	**1**	**1**	**1**	**0**
8	**1**	**0**	**0**	**0**	**1**
9	**1**	**0**	**0**	**1**	**1**
10	**1**	**0**	**1**	**0**	**Ø**
11	**1**	**0**	**1**	**1**	**1**
12	**1**	**1**	**0**	**0**	**1**
13	**1**	**1**	**0**	**1**	**Ø**
14	**1**	**1**	**1**	**0**	**1**
15	**1**	**1**	**1**	**1**	**Ø**

2- simplification par table de Karnaugh

ab / *cd*	*00*	*01*	*11*	*10*
00	**Ø**	**1**	**1**	**1**
01	**1**	**0**	**Ø**	**1**
11	**1**	**0**	**Ø**	**1**
10	**1**	**Ø**	**1**	**Ø**

$$\mathbf{F = \bar{d} + \bar{b}}$$

4- $\mathbf{F = \overline{\overline{\bar{d} + \bar{b}}} = \overline{d.b}}$



Exercice 2 : 1- table de vérité

n	a	b	c	d	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

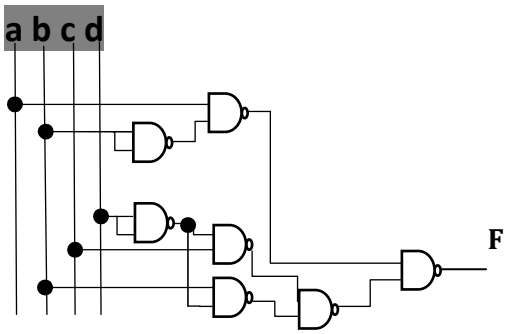
2- simplification par table de Karnaugh

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

$$\mathbf{F = a\bar{b} + b\bar{d} + c\bar{d}}$$

4- $\mathbf{F = \overline{\overline{a\bar{b}} + (b\bar{d} + c\bar{d})}}$

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



<u>Exercice 2 .9:</u> 1- table de vérité

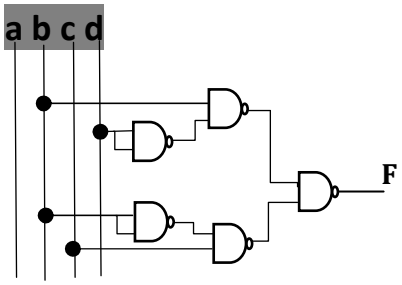
n	**a**	**b**	**c**	**d**	**F**
0	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**	**0**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**1**
4	**0**	**1**	**0**	**0**	**1**
5	**0**	**1**	**0**	**1**	**0**
6	**0**	**1**	**1**	**0**	**1**
7	**0**	**1**	**1**	**1**	**0**
8	**1**	**0**	**0**	**0**	**0**
9	**1**	**0**	**0**	**1**	**0**
10	**1**	**0**	**1**	**0**	**1**
11	**1**	**0**	**1**	**1**	**1**
12	**1**	**1**	**0**	**0**	**1**
13	**1**	**1**	**0**	**1**	**0**
14	**1**	**1**	**1**	**0**	**1**
15	**1**	**1**	**1**	**1**	**0**

2- simplification par table de Karnaugh

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

$$\mathbf{F = b\bar{d} + \bar{b}c}$$

4- $\mathbf{F = \overline{\overline{b\bar{d}} + \overline{\bar{b}c}} = \overline{\overline{b\bar{d}}}.\overline{\overline{\bar{b}c}}}$



Exercice 2 .11:

Compléter la table de vérité :

Entrées				**Sorties**		
b_1	b_0	a_1	a_0	*E* (*A=B*)	*S* (*A>B*)	*I* (*A<B*)
0	0	0	0	**1**	0	0
0	0	0	1	0	**1**	0
0	0	1	0	0	**1**	0
0	0	1	1	0	**1**	0
0	1	0	0	0	0	**1**
0	1	0	1	**1**	0	0
0	1	1	0	0	**1**	0
0	1	1	1	0	**1**	0

1	0	0	0	0	0	**1**
1	0	0	1	0	0	**1**
1	0	1	0	**1**	0	0
1	0	1	1	0	**1**	0
1	1	0	0	0	0	**1**
1	1	0	1	0	0	**1**
1	1	1	0	0	0	**1**
1	1	1	1	**1**	0	0

Simplifiez les fonctions *E*, *S* et *I* en utilisant la méthode de KARNAUGH

$a_1\ a_0$ / $b_1\ b_0$	*00*	*01*	*11*	*10*
00	**1**	**0**	**0**	**0**
01	**0**	**1**	**0**	**0**
11	**0**	**0**	**1**	**0**
10	**0**	**0**	**0**	**1**

$$E = \bar{a}_0\bar{a}_1\bar{b}_0\bar{b}_1 + a_0\bar{a}_1b_0\bar{b}_1 + a_0a_1b_0b_1 + \bar{a}_0a_1\bar{b}_0b_1$$

$a_1\ a_0$ / $b_1\ b_0$	*00*	*01*	*11*	*10*
00	**0**	**1**	**1**	**1**
01	**0**	**0**	**1**	**1**
11	**0**	**0**	**0**	**0**
10	**0**	**0**	**1**	**0**

$$S = a_0\bar{b}_0\bar{b}_1 + a_0a_1\bar{b}_0 + a_1\bar{b}_1$$

$a_1\ a_0$ / $b_1\ b_0$	*00*	*01*	*11*	*10*
00	**0**	**0**	**0**	**0**
01	**1**	**0**	**0**	**0**
11	**1**	**1**	**0**	**1**
10	**1**	**1**	**0**	**0**

$$I = \bar{a}_0\bar{a}_1b_0 + \bar{a}_0b_0b_1 + \bar{a}_1b_1$$

Chapitre 3 Les Circuits Combinatoires

Introduction :

Pour résoudre le problème de la logique combinatoire, il faut connaître l'algèbre de Boole. Cet algèbre permet de traduire des signaux en expressions mathématiques en remplaçant chaque signal élémentaire par des variables logiques et leur traitement par des fonctions logiques. Ces fonctions seront appelées fonctions combinatoires et l'étude de la logique combinatoire. Un circuit combinatoire est un circuit numérique dont les sorties dépendent uniquement des entrées.

Toute fonction logique peut être réalisée à l'aide des portes logiques. Pour cela, il faut :

1-Ecrire l'équation de la fonction à partir de sa table de vérité.

2-Simplifier l'équation.

3-Réaliser l'équation à l'aide des portes disponibles.

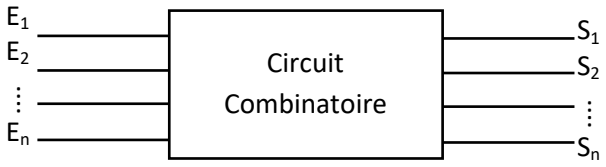


Schéma bloc

C'est possible pour utiliser des circuits combinatoires de base pour réaliser d'autres circuits plus complexes. Dans ce qui suit, on présente quelques exemples de circuits combinatoires.

3.1 Demi-Additionneur

Un additionneur est un circuit logique permette de réaliser une addition. Il porte sur un bit unique mène aux 4 cas notés dans la table de vérité suivante :

A	B	S (Somme)	R (Retenue)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$R = AB$$

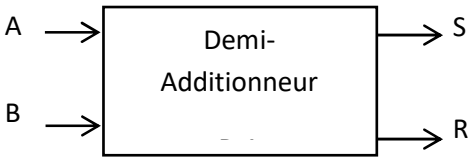
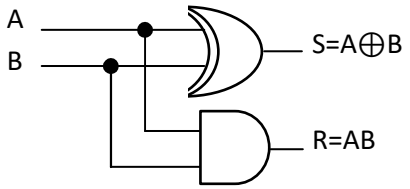


Schéma bloc



Logigramme de demi-additionneur

3.2 Additionneur Complet

La représentation de l'additionneur complet est donnée par le schéma suivant, où R_i indique la retenue et R_{i-1} la retenue précédente.

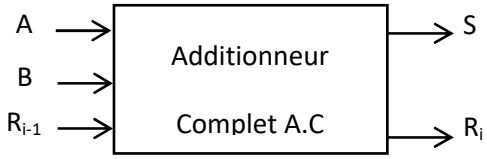


Schéma bloc

La table de vérité de l'additionneur complet est donnée comme suit

A	B	R_{i-1}	S	R_i
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

On fait extraire les équations logiques des sorties S et R_i

$$S = \overline{A}\,\overline{B}R_{i-1} + \overline{A}B\overline{R_{i-1}} + A\,\overline{B}\overline{R_{i-1}} + A\,BR_{i-1}$$

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

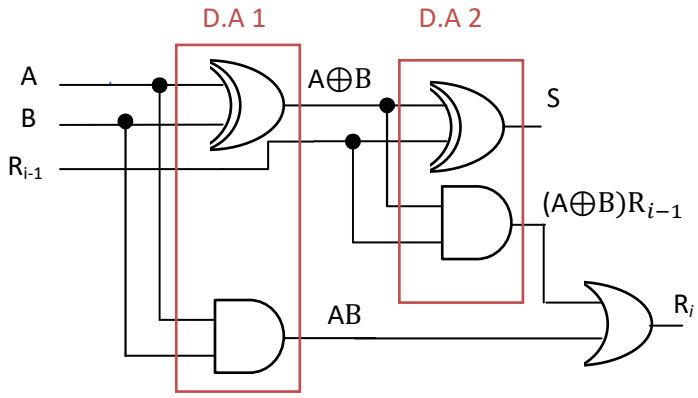
$$= R_{i-1}\left(\overline{A \oplus B}\right) + \overline{R_{i-1}}(A \oplus B)$$

$$= A \oplus B \oplus R_{i-1}$$

$$R_i = \overline{A}\,BR_{i-1} + A\,\overline{B}R_{i-1} + AB\overline{R_{i-1}} + ABR_{i-1}$$

$$= R_{i-1}(\overline{A}\,B + A\,\overline{B}) + AB(\overline{R_{i-1}} + R_{i-1})$$

$$= R_{i-1}(A \oplus B) + AB$$



Logigramme d'Additionneur Complet

3.3 Demi- Soustracteur

Un soustracteur est un circuit logique permette de réaliser une soustraction. Il porte sur un bit unique mène aux 4 cas notés dans la table de vérité suivante :

A	B	D (Différence)	R (Retenue)
0	0	0	0
0	1	**1**	**1**
1	0	**1**	0
1	1	0	0

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

$$R = \overline{A}B$$

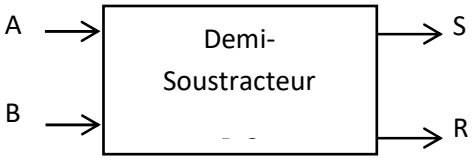
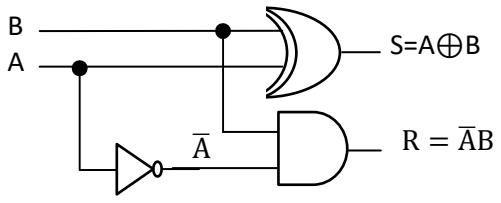


Schéma bloc



Logigramme de demi-soustracteur

3.4 Soustracteur complet

La représentation du soustracteur complet est donnée par le schéma suivant, où R_i indique la retenue et R_{i-1} la retenue précédente.

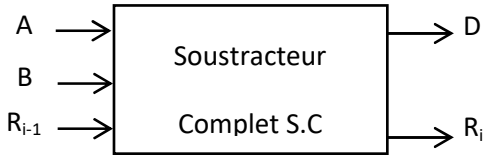


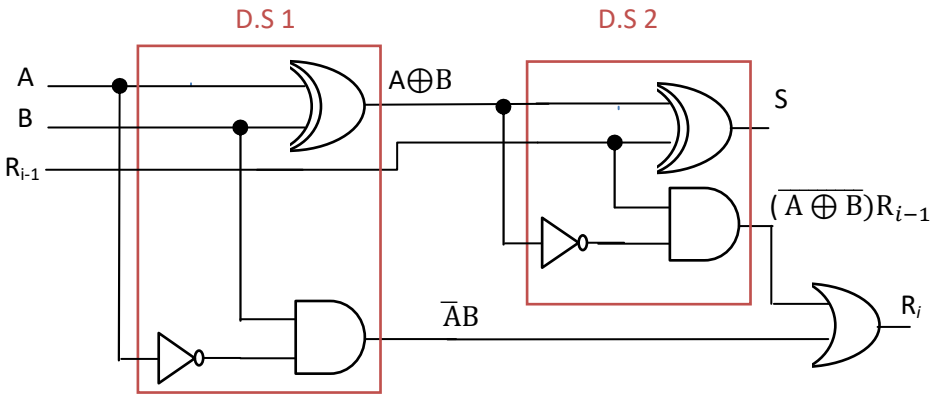
Schéma bloc

La table de vérité du soustracteur complet est donnée comme suit

A	B	R_{i-1}		D	R_i
0	0	0		0	0
0	0	1		1	1
0	1	0		1	1
0	1	1		0	1
1	0	0		1	0
1	0	1		0	0
1	1	0		0	0
1	1	1		1	1

On fait extraire les équations logiques des sorties S et R_i

$$D = \overline{A}\,\overline{B}R_{i-1} + \overline{A}B\overline{R_{i-1}} + \overline{A}\,BR_{i-1} + A\,BR_{i-1}$$
$$= R_{i-1}(\overline{A}\,\overline{B} + A\,B) + \overline{R_{i-1}}(\overline{A}B + A\,\overline{B})$$
$$= R_{i-1}\left(\overline{A \oplus B}\right) + \overline{R_{i-1}}(A \oplus B)$$
$$= A \oplus B \oplus R_{i-1}$$
$$R_i = \overline{A}\overline{B}R_{i-1} + \overline{A}B\overline{R_{i-1}} + \overline{A}B\overline{R_{i-1}} + ABR_{i-1}$$
$$= R_{i-1}(\overline{A}\,B + A\,\overline{B}) + \overline{A}B(\overline{R_{i-1}} + R_{i-1})$$
$$= R_{i-1}(\overline{A \oplus B}) + \overline{A}B$$



Logigramme d'un Soustracteur Complet

3.5 Comparateur

Le comparateur est un circuit arithmétique permettre de comparer deux nombres binaires A et B, avec A et B doivent avoir la même longueur (nombre de bits).

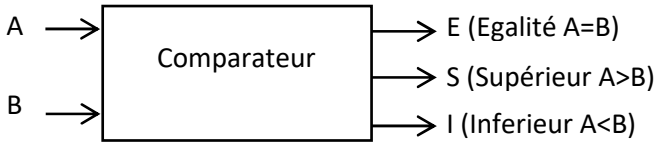


Schéma bloc

Par la suite on analyse par la table de vérité le fonctionnement d'un comparateur à 1 bit

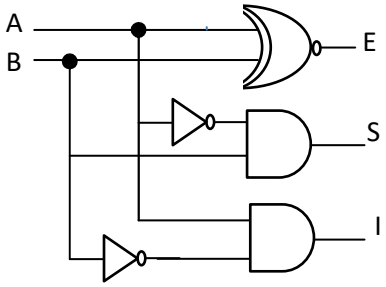
A	B		E	S	I
0	0		1	0	0
0	1		0	0	1
1	0		0	1	0
1	1		1	0	0

On fait extraire les équations logiques des sorties E, S et I.

$$E = \overline{A}\overline{B} + AB = \overline{A \oplus B}$$

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

$$I = \overline{A}B$$



Logigramme d'un comparateur à un bit

3.6 Encodeur

L'encodeur est un système combinatoire ayant pour fonction de retourner l'index d'activation d'une parmi 2^n entrées. L'index d'activation est donné sur *n* lignes d'adresse. Lorsque plusieurs entrées sont activées, l'encodeur accorde la priorité à l'entrée dont l'index est supérieur. La notation usuelle de l'encodeur est : encodeur 2^n à *n*. Par exemple, un encodeur 8 à 3 aura 8 entrées et 3 lignes d'adresse en sortie. (Il fournit en sortie le numéro de l'entrée active sur *n* bit).

3.6.1 Codeur 4 voies d'entrées et 2 bits de sortie

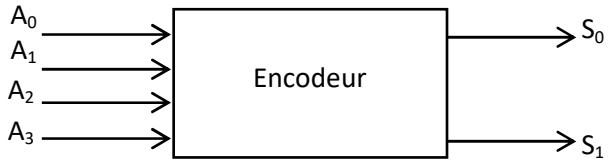


Schéma bloc

La table de vérité est la suivante :

Entrées					**Sorties**	
Codage 1 parmi 2^n					*Nombre binaire de n bits*	
A_3	A_2	A_1	A_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

Equation des sorties :

$$S_0 = \begin{cases} 1\ (pour\ (A_1 = 1\ et\ A_0 = A_2 = A_3 = 0)\ \ ou\ (A_3 = 1\ et\ A_1 = A_2 = A_3 = 0) \\ 0\ (pour\ (A_0 = 1\ et\ A_1 = A_2 = A_3 = 0)\ \ ou\ (A_2 = 1\ et\ A_0 = A_1 = A_3 = 0) \\ \emptyset\ (\text{indéfine ailleurs}) \end{cases}$$

$$S_1 = \begin{cases} 1\ (pour\ (A_2 = 1\ et\ A_0 = A_1 = A_3 = 0)\ \ ou\ (A_3 = 1\ et\ A_1 = A_2 = A_3 = 0) \\ 0\ (pour\ (A_0 = 1\ et\ A_1 = A_2 = A_3 = 0)\ \ ou\ (A_1 = 1\ et\ A_0 = A_2 = A_3 = 0) \\ \emptyset\ (\text{indéfine ailleurs}) \end{cases}$$

Ceci est illustré dans la table de vérité étendue suivante

Entrées					**Sorties**	
Codage 1 parmi 2^n					*Nombre binaire de n bits*	
A_3	A_2	A_1	A_0		S_1	S_0
0	0	0	0		Ø	Ø
0	0	0	1		0	0
0	0	1	0		0	1
0	0	1	1		Ø	Ø
0	1	0	0		1	0
0	1	0	1		Ø	Ø
0	1	1	0		Ø	Ø
0	1	1	1		Ø	Ø
1	0	0	0		1	1
1	0	0	1		Ø	Ø
1	0	1	0		Ø	Ø
1	0	1	1		Ø	Ø
1	1	0	0		Ø	Ø
1	1	0	1		Ø	Ø
1	1	1	0		Ø	Ø
1	1	1	1		Ø	Ø

On utilise la table de Karnaugh pour simplifier S_0

A_1A_0 \ A_3A_2	*00*	*01*	*11*	*10*
00	Ø	**0**	Ø	**1**
01	**0**	Ø	Ø	Ø
11	Ø	Ø	Ø	Ø
10	**1**	Ø	Ø	Ø

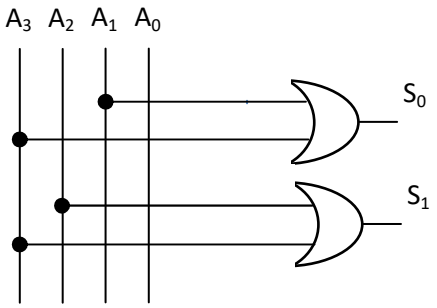
$$S_0 = A_1 + A_3$$

Simplification de S_1

A_1A_0 \ A_3A_2	*00*	*01*	*11*	*10*
00	Ø	**1**	Ø	**1**
01	**0**	Ø	Ø	Ø
11	Ø	Ø	Ø	Ø
10	**0**	Ø	Ø	Ø

$$S_1 = A_2 + A_3$$

Le logigramme correspondant est donné comme suit :



3.6.2 Codeur de priorité

Suivant le codeur 4 à 2 précédent, par exemple lorsque A_1 et A_2 sont activées simultanément, S_0 et S_1 sont indéfinis. Le codeur de priorité permet de donner une valeur définie aux sorties (**1** ou **0**) selon la priorité entre les entrées.

Par exemple on utilise un codeur de priorité 4 voies d'entrées et 2 bits de sortie qui choisit le plus grand nombre lorsque plusieurs entrées sont activées à la fois. La table de vérité devient comme suivante :

Entrées				**Sorties**	
Codage 1 parmi 2^n				*Nombre binaire de n bits*	
A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

On utilise la table de Karnaugh pour simplifier S_0

A_1A_0 \ A_3A_2	00	01	11	10
00	**0**	**0**	**1**	**1**
01	**0**	**0**	**1**	**1**
11	**1**	**0**	**1**	**1**
10	**1**	**0**	**1**	**1**

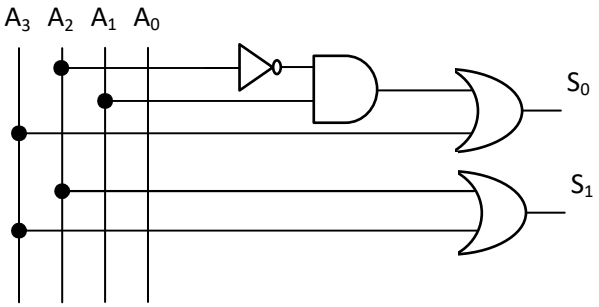
$$S_0 = A_1\overline{A_2} + A_3$$

Simplification de S_1

A_1A_0 \ A_3A_2	00	01	11	10
00	**0**	**1**	**1**	**1**
01	**0**	**1**	**1**	**1**
11	**0**	**1**	**1**	**1**
10	**0**	**1**	**1**	**1**

$$S_1 = A_2 + A_3$$

Le logigramme correspondant est donné comme suit :



3.7 Décodeur

Un décodeur est un circuit logique réalisant la fonction inverse du codeur. On prend l'exemple d'un décodeur 1 parmi 4. Pour pouvoir activer toutes les 8 voies on a besoin de 3 bits à l'entrée car 2^3=8.

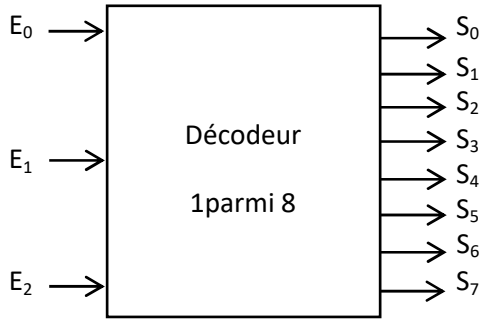


Schéma bloc

Entrées				**Sorties**							
Code binaire				Code 1 parmi 8							
E_2	E_1	E_0		S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0		0	0	0	0	0	0	0	**1**
0	0	1		0	0	0	0	0	0	**1**	0
0	1	0		0	0	0	0	0	**1**	0	0
0	1	1		0	0	0	0	**1**	0	0	0
1	0	0		0	0	0	**1**	0	0	0	0
1	0	1		0	0	**1**	0	0	0	0	0
1	1	0		0	**1**	0	0	0	0	0	0
1	1	1		**1**	0	0	0	0	0	0	0

Équation de sorties :

$$S_0 = \overline{E_0 E_1 E_2}$$

$$S_1 = E_0 \overline{E_1 E_2}$$

$$S_2 = \overline{E_0} E_1 \overline{E_2}$$

$$S_3 = E_0 E_1 \overline{E_2}$$

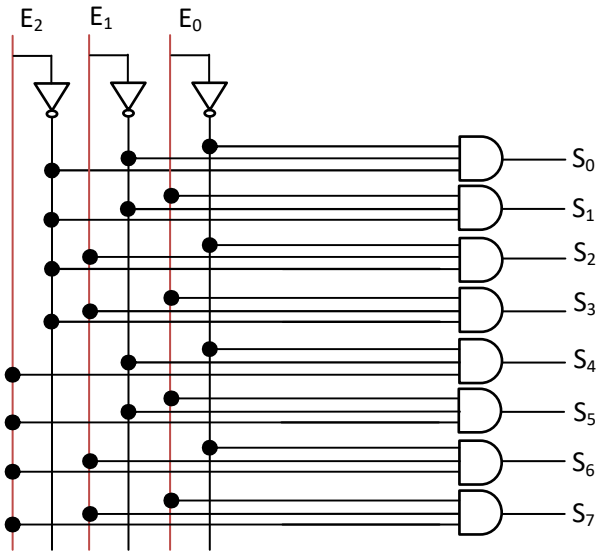
$$S_4 = \overline{E_0 E_1} E_2$$

$$S_5 = E_0 \overline{E_1} E_2$$

$$S_6 = \overline{E_0} E_1 E_2$$

$$S_7 = E_0 E_1 E_2$$

Le logigramme correspondant est donné comme suit :



3.8 Transcodeur

Un transcodeur est un circuit combinatoire permettant de passer d'un code à un autre. Comme exemple, soit à concevoir un transcodeur binaire vers Gray à 4 bits.Les variables *a, b, c* et *d* représentent le nombre en code binaire, et *A, B, C* et *D* représentent le même nombre en code Gray.

(*a, b, c, d*) sont les variables d'entrées,et (*A, B, C,D*) sont les variables d'entrées, en analysant par table de vérité:

	Entrées				**Sorties**			
N°	*a*	*b*	*c*	*d*	*A*	*B*	*C*	*D*
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

On utilise la table de Karnaugh pour simplifier *A,B, C,* et *D* respectivement :

Sortie A :

ab / *cd*	*00*	*01*	*11*	*10*
00	**0**	**0**	**1**	**1**
01	**0**	**0**	**1**	**1**
11	**0**	**0**	**1**	**1**
10	**0**	**0**	**1**	**1**

$$A = a$$

Sortie B :

ab / *cd*	*00*	*01*	*11*	*10*
00	**0**	**1**	**0**	**1**
01	**0**	**1**	**0**	**1**
11	**0**	**1**	**0**	**1**
10	**0**	**1**	**0**	**1**

$$B = a\bar{b} + \bar{a}\, b = a \oplus b$$

Sortie C :

ab / *cd*	*00*	*01*	*11*	*10*
00	**0**	**1**	**1**	**0**
01	**0**	**1**	**1**	**0**
11	**1**	**0**	**0**	**1**
10	**1**	**0**	**0**	**1**

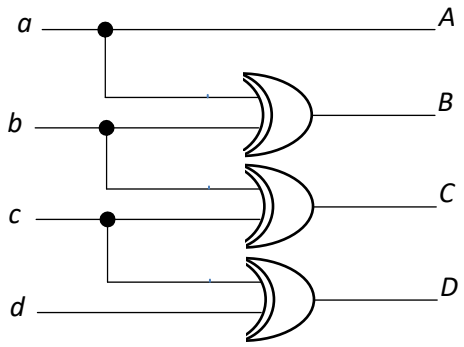
$$C = b\bar{c} + \bar{b}\, c = b \oplus c$$

Sortie D :

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

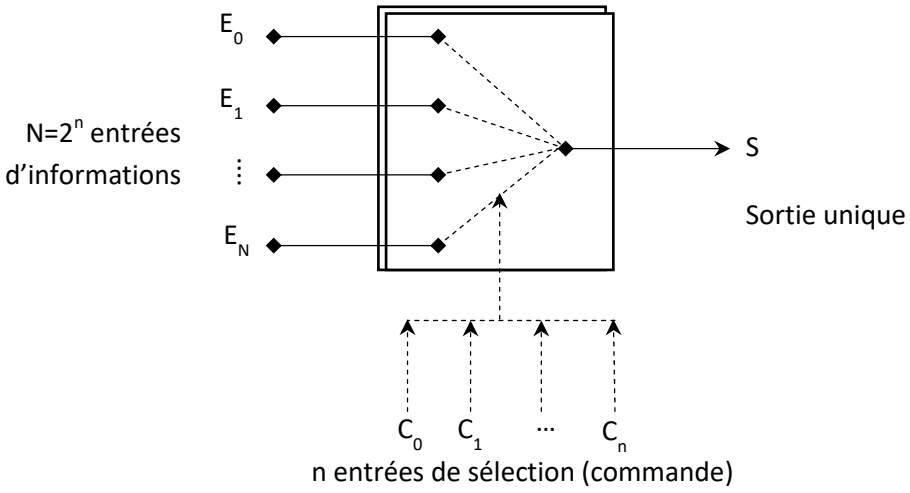
$$D = \bar{c}d + c\bar{d} = c \oplus d$$

Le logigramme correspondant est le suivant :



3.9 Multiplexeur

C'est un circuit combinatoire permettant de réaliser un aiguillage del'une des entrées en une sortie unique, son schéma bloc est donné comme suit :



Pour N = 2^n entrées (avec n entier positif) correspond à n éléments binaire de commande (sélection). Comme exemple, on prend un multiplexeur 4 vers 1, Il s'agit

d'un multiplexeur à 4 entrées (qu'on note E_0, E_1,E_2 et E_3), qui nécessite deux (2)entrée de commande (qu'on nomme C_0 , C_1) et une seule sortie (S).

Son fonctionnement en aiguillage se résume comme suit :

$S = E_0$ si $C_0 = 0$ et $C_1 = 0$

$S = E_1$ si $C_0 = 1$ et $C_1 = 0$

$S = E_2$ si $C_0 = 0$ et $C_1 = 1$

$S = E_3$ si $C_0 = 1$ et $C_1 = 1$

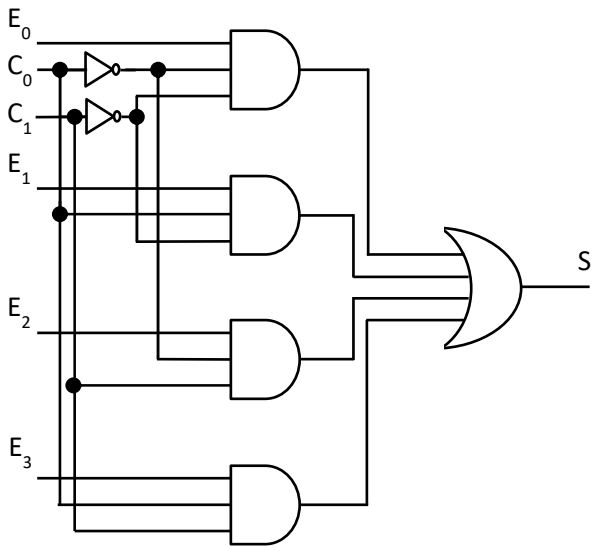
Cela est traduit par la table de vérité simplifiée suivante :

C_1	C_0	S
0	0	E_0
0	1	E_1
1	0	E_2
1	1	E_3

Son équation logique est :

$$S = \overline{C_0}\overline{C_1}E_0 + C_0\overline{C_1}E_1 + \overline{C_0}C_1E_2 + C_0C_1E_3$$

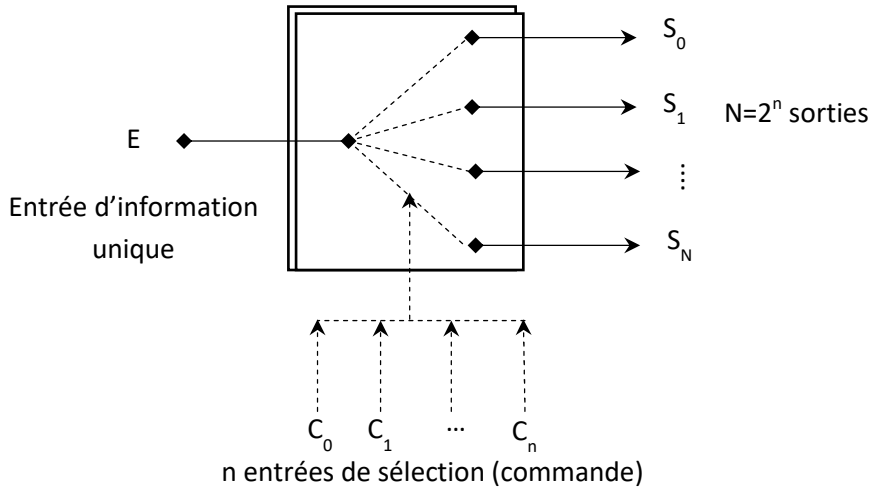
Le logigramme du multiplexeur 4 vers 1 :



3.10 Démultiplexeur

Le démultiplexeur réalise l'opération inverse de celle du multiplexeur. Il comporte une seule entrée d'information (ou de données) E, n entrées de commande C_i avec i = 0,

1, ..., n (appelées aussi entrées d'adresse ou de sélection) et N = 2^n sorties (S_0, S_1, ..., S_N).Son schéma bloc est donné comme suit :



Le démultiplexeur 1 vers 4 est un démultiplexeur à 4 (2^2) sorties (S_0, S_1, S_2 et S_3), qui nécessite 2 entrées de commande (C_0 et C_1) et une seule entrée(E).

Son fonctionnement est montré sur sa table de vérité simplifiée suivante :

C_1	C_0	S_3	S_2	S_1	S_0
0	0	0	0	0	E
0	1	0	0	E	0
1	0	0	E	0	0
1	1	E	0	0	0

Les équations logiques de sorties sont :

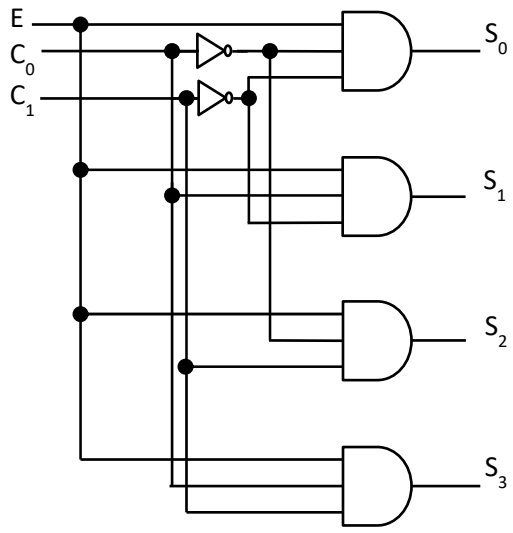
$$S_0 = \overline{C_0 C_1}\, E$$

$$S_1 = C_0 \overline{C_1}\, E$$

$$S_2 = \overline{C_0} C_1 E$$

$$S_3 = C_0 C_1 E$$

Le logigramme du multiplexeur 4 vers 1 :



3.11 Technologie des circuits intégrés

Les circuits intégrés logiques sont classés suivant leur technologie de fabrication (bipolaire TTL, bipolaire ECL, MOS,…). Pour un fonctionnement logique identique, chaque technologie offre des performances différentes sur le plan électrique (tensions, courants, puissances) et temporel (rapidité).

Une famille logique est caractérisée par ses paramètres électriques :

- la plage des tensions d'alimentation et la tolérance admise sur cette valeur,
- la plage des tensions associée à un niveau logique, en entrée ou en sortie,
- les courants pour chaque niveau logique, en entrée ou en sortie,
- le courant maximum que l'on peut extraire d'une porte logique et le courant absorbé en entrée,
- la puissance maximale consommée qui dépend aussi de la fréquence de fonctionnement.

Les performances dynamiques principales sont :

- les temps de montée (transition bas haut) et de descente (transition haut bas) des signaux en sortie d'une porte,
- les temps de propagation d'un signal entre l'entrée et la sortie d'une porte logique.

Les différentes notions abordées seront illustrées de valeurs numériques issues de la technologie **TTL**.

Nous présentons dans ce chapitre les caractéristiques les plus importantes des circuits intégrés ainsi que la matérialisation de quelques portes logiques.

Une famille de circuits logiques intégrés regroupe un ensemble d'éléments réalisant chacun une fonction logique élémentaire ou un système logique plus ou moins complexe, mais bâtis sur des principes de base communs. Ceci leur permet de pouvoir être associés au sein d'une même famille en respectant un comportement commun, c'est la compatibilité.

Au cours du développement industriel des circuits intégrés, plusieurs familles se sont succédé, chacune ayant sa structure propre. Initialement la famille DTL, abréviation de Diode-Transistor Logic, permettait de réaliser simplement une porte NAND. Des modifications complémentaires, notamment le remplacement des diodes d'entrée par un transistor multi-émetteur, ont abouti à la structure de base de la famille TTL, abréviation de Transistor-Transistor Logic.

La famille TTL, réalisée en technologie bipolaire, a acquis une position dominante pour les applications courantes jusqu'à l'apparition des circuits CMOS (Complementary Metal Oxide Semiconductor), issus d'une technologie plus récente à base de transistors MOS.

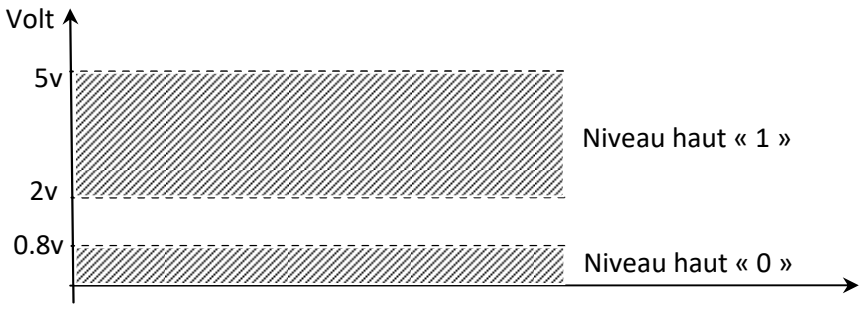
Pour des applications requérant une grande rapidité, la famille ECL (Emitter Coupled Logic), s'avère la plus performante.

En logique électronique, une variable logique est matérialisée par certaines valeurs d'une tension électrique et un circuit logique est constituée par des composants électroniques semi- conducteurs «diode, ou, tronsistor » associés pour former des circuits intégrés.

Par convention, on représente par les valeurs logiques << 0 >> ou << 1 >> certains intervalles de niveau de tension par rapport à une borne commune appelée « masse ».

En logique TTL (transistor-transistor logic):
La tension d'alimentation est de +5v.
- la tension comprise entre 0 et 0.8v représentent un niveau bas : << 0 logique »,
- la tension comprise entre 2v et 5.5v représentent un niveau haut : « 1 logique »,
- les tensions comprises entre 0.8v et2v n'apparaissent que de façon transitoire.

Matériellement, les CI se présentent sous forme d'un boîtier rectangulaire de faible épaisseur muni de broches.



Matérialisation du niveau haut et bas en logique positive

Les circuits intégrés (CI) sont groupés suivant leur complexité :

- SSI (*small scale integration*) ou moins de dix portes logiques sont intégrés.
- MSI (*medium scale integration*) ou le nombre de portes intégrés est entre 10 et 100, (*Large scale integration*) ou le nombre de portes intégrés est entre 100 et 1000.
- VLSI (*very large scale integration*) ou le nombre de portes logiques intégrés dépasse 1000. Ces CI sont fabriqués suivant différentes conceptions qu'on appelle famille logique.

Les CI d'une même famille logique sont dits compatibles et peuvent être facilement connectés entre eux ; tandis que lorsqu'on connecte deux CI de familles différentes, il faut prévoir des interfaces de passage dans les deux sens.

Les différentes familles de circuits logiques sont :

a- pour les familles bipolaires :

- RTL (résistance - transistor - logic)
- DTL (diode - transistor - logic)
- TTL (transistor - transistor - logic)
- ECL (Emitter - coupled- logic) - HTL.

b- Pour les familles MOS (Metal oxyde semiconductor)

-PMOS (P-canal MOS)
- NMOS (N- canal MOS)
- CMOS (complementary MOS).

La famille TTL introduite en 1964, est celle qui a connu les développements les plus importants. Actuellement il existe sept séries :

- La série **N** normale qui fût la première introduite sur le marché ;
- La série **H** « High speed » destinée aux applications nécessitant des vitesses de commutation élevées ;

- La série **L** « low power » pour les applications lentes.

Ces trois premières séries sont en voie de disparition. Les transistors fonctionnent en saturation et en blocage. La différence entre ces trois séries réside principalement dans la valeur des composants, ce qui explique les différences de consommation.

La seconde génération de circuits TTL utilise des transistors qui ne fonctionnent plus en saturation grâce à la diode Schottky placée entre base et collecteur. D'autre part les transistors ne sont plus dopés à l'or, ce qui réduit notamment leurs capacités parasites. Ces deux améliorations ont permis de diminuer de façon significative les temps de commutation des transistors. On distingue :

- **S** « Schottky » réservée aux applications rapides,
- **LS** « Low power Schottky » destinée à remplacer la série normale.

La troisième génération de circuit TTL est une amélioration technologique des circuits S et LS. Une réduction des dimensions des transistors a permis une réduction des capacités de jonction dans un rapport voisin de 50 à 60%. Les séries portent les nom de :

- **AS** « Advanced Schottky » ou F « Fairchild Advanced Schottky technology";
- **ALS** « Advanced Low power Schottky ».

On a standardisé les boîtiers (en plastique ou en céramique) à 8 broches et à 14, 16, 18, 20, 24, 28,40 exceptionnellement plus.

Exemple: Circuit SN 74 S20N

Tel que :

SN : circuit numérique standard.

74 : série industrielle (il existe la série 40 par exemple)

S : famille TTL schottky

20 : le numéro du circuit dans la famille.

N : boîtier dual in line en plastique.

En technologie TTL « transistor-transistor-logic » : logique à transistor bipolaires.

On définit les familles suivantes :

Std: TTL standard.

L: TTL faible consommation.

S : TTL schottky.

LS : low power schotty :

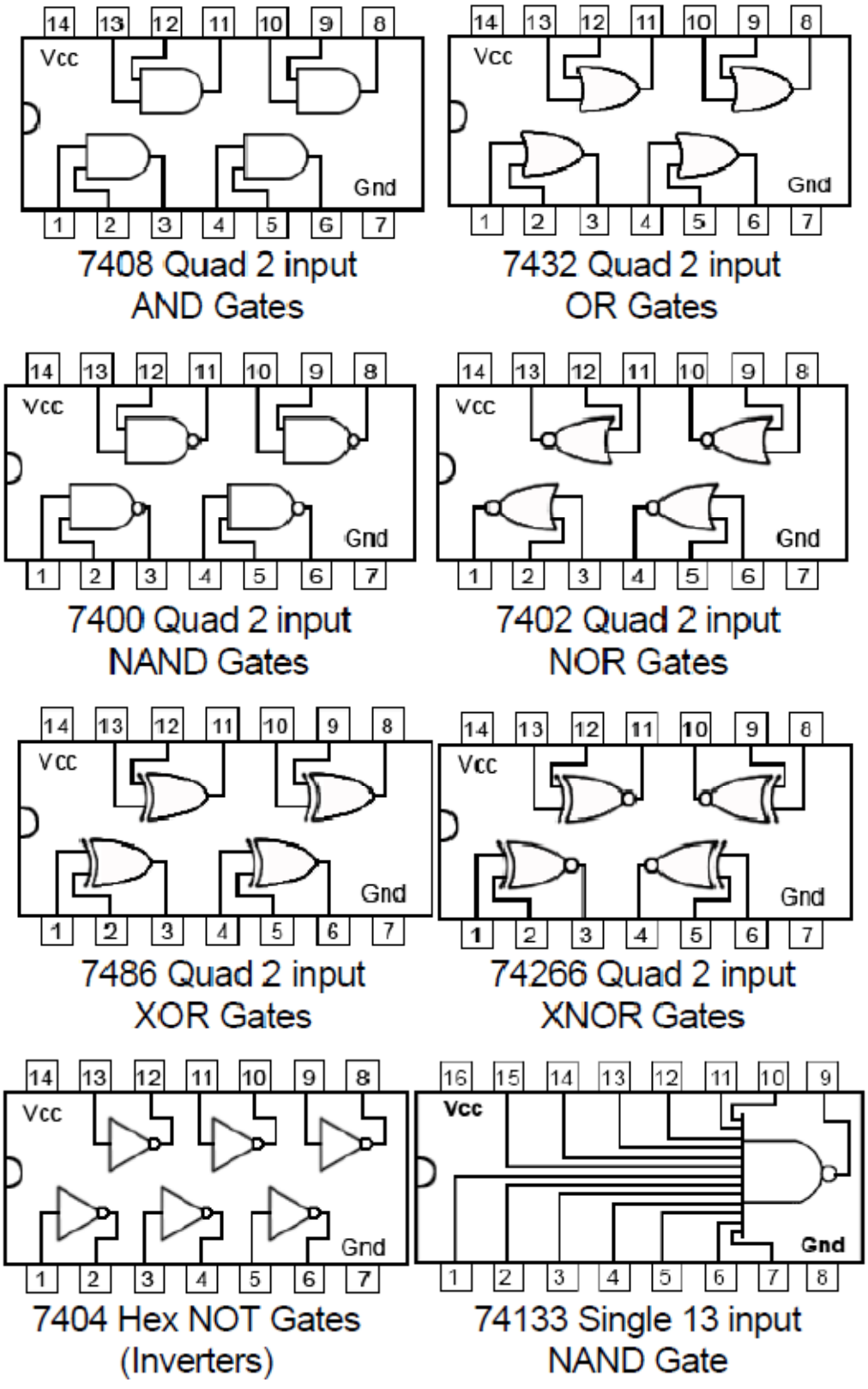
TTL schottky faible consommation.

AS : Advanced Schottky : TTL Schottky avancée.

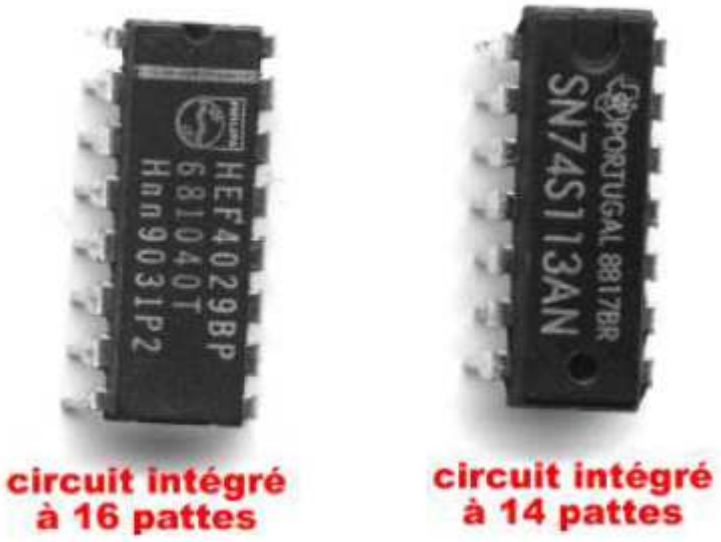
ALS: Advanced low power schottky : TTL scohttky avancée à faible puissance.

F: fast (TTL rapide).

La figure suivante montre quelques circuits intégrés de la famille TTL : 74xx



En électronique, les portes logiques sont fabriquées et renfermées dans des circuits intégrés. La photo suivante montre deux circuits intégrés.

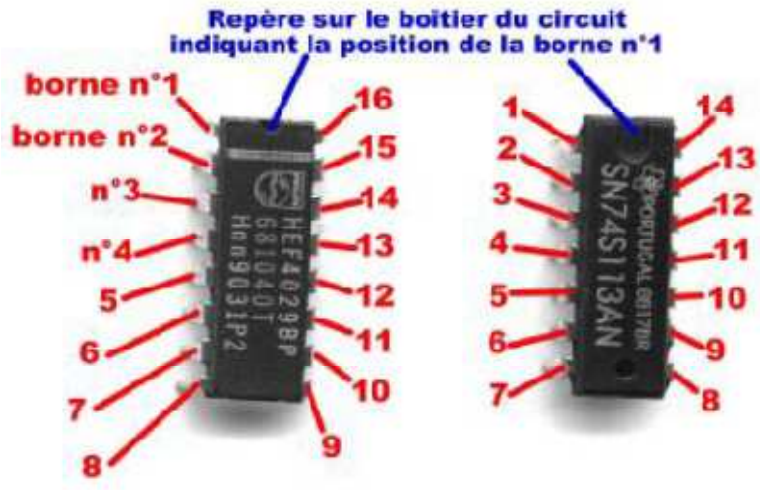


On peut remarquer sur le haut des circuits intégré un petit creux appelé « ergo ». L'ergo permet d'orienter correctement le circuit intégré afin de repérer les différentes bornes.

Un circuit intégré renferme plusieurs portes logiques, dont les entrées et les sorties sont accessibles sur les différentes bornes du circuit intégré. Pour identifier chaque borne sans ambiguïté, elles sont numérotées de manière normalisée en respectant le principe suivant :

- en regardant le circuit intégré avec l'ergo vers le haut, la borne n°1 est la borne située en haut à gauche
- les autres bornes sont numérotées en tournant dans le sens inverse des aiguilles d'une montre

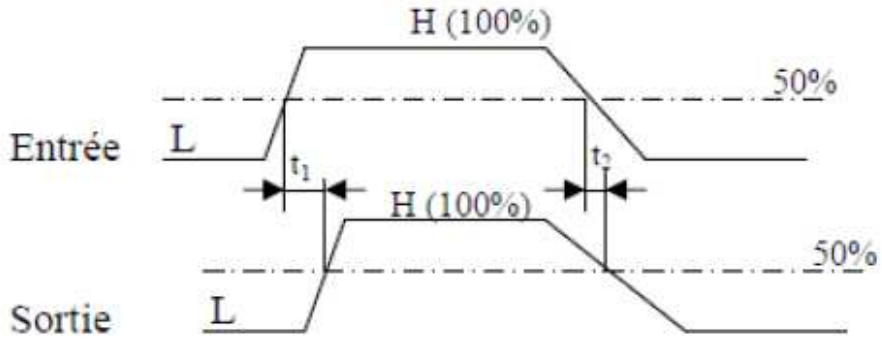
Ce principe reste vrai quelque soit le nombre de bornes (de « pattes ») du circuit intégré. La photo suivante montre le numéro de chacune des bornes pour un circuit intégré à 16 bornes (16 « pattes ») et pour un circuit intégré à 14 bornes (14« pattes »).



3. 11. 1 Temps de propagation d'une porte logique

C'est le temps moyen que met le signal pour franchir l'opérateur logique (de 2 à 100 ns). Le niveau de sortie d'une porte logique varie avec le niveau d'entrée, mais avec un certain retard du aux commutations internes.

Ces retards ou (T de p) définissent la fréquence maximale d'utilisation des portes ou circuits logiques ; ces retards varient de 3 nsec (TTL schottky) à 125 nsec.



Temps de propagation d'une porte

$T_{moy}=(t_1+t_2)/2$.

3. 10. 2 Circuits logiques à sorties 3 états (Tri states)

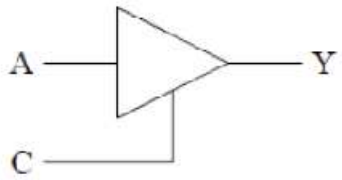
Les CI de la famille TTL (logique transistor - transistor) sont très utilisés. Une sortie typique d'une unité TTL peut être soit 1 ou 0 logiques. Pour cette raison, il n'est pas possible de connecter les sorties d'éléments TTL standard à un bus de micro (ex : les bus de données).

Des éléments TTL spéciaux ont été mis au point dont les sorties peuvent être interconnectés sur un bus partagé entre eux. Ces éléments TTL sont dits éléments à 3 états.

Ces éléments possèdent des états de sortie :

- 0 logique
- 1 logique.
- Un état particulier ; l'état à haute impédance.

La porte "3 états", ou "tri-state", n'est pas une porte logique au sens strict. Elle est principalement utilisée pour connecter une sortie sur une ligne commune à plusieurs circuits (un bus par exemple). Elle remplace généralement une porte ET. En effet, la mise en parallèle sur une même ligne de plusieurs portes ET introduit des capacités parasites. Ceci augmente les constantes de temps et a pour effet de détériorer les fronts de montée et de descente des signaux. Cela peut perturber le fonctionnement d'un système.



C	A	Y	Sortie
1	0	0	Faible impédance
1	1	1	Faible impédance
0	X	0	Haut impédance

Lorsque la commande C est à 0, l'impédance de sortie est très grande : pratiquement déconnectée. D'autre part, ces portes "3 états" fournissent une amplification de puissance.

La sortie d'un circuit logique 3 états peut prendre l'état haute impédance qui représente un niveau flottant. Ce 3éme état sert à isoler ou à déconnecter la sortie d'une porte logique du reste.

Chapitre 4 Circuits logiques séquentiels

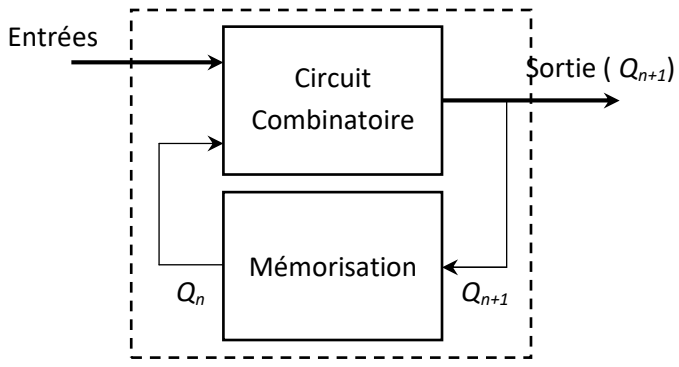
Introduction :

On a vu que dans un circuit combinatoire la sortie dépend uniquement des entrées du système, pour les mêmes entrées on a toujours la même sortie. Pour les circuits de logique séquentielle nous devons tenir compte de l'état du système. Ainsi les sorties dépendent des entrées mais également de l'état du système(le système mémorise les sorties précédentes), donc pour les mêmes entrées on n'a pas toujours les mêmes sorties.

Les circuits séquentiels présentent une caractéristique de mémoire. La différence entre la logique combinatoire et la logique séquentielle est que :

- La logique séquentielle a pour élément de base « ***la bascule*** » contrairement à la logique combinatoire qui avait pour élément de base« ***la porte logique***».
- Les états de sortie dépendent uniquement de la combinaison des variables d'entrées pour un circuit combinatoire. Tandis que pour un circuit séquentiel, l'état de la sortie dépend à la fois de la combinaison des variables d'entrée et de l'état antérieur de la sortie (temporelle).

Tout circuit séquentiel est constitué d'un circuit combinatoire couplé `a une mémoire comme indiqué par la figure suivante :



Circuit séquentiel

Donc : $Q_{n+1}=f(Entrées, Q_n)$

Les bascules sont des circuits à deux états stables (aussi appelé circuits bistables). Ces circuits sont utilisés pour mémoriser l'état des signaux d'entrée qui sont alors disponibles à la sortie des bascules. Elles possèdent deux sorties complémentaires Q et $\overline{Q}$. On note Q_{n+1}la sortie à l'état actuel et Q_n la sortie à l'état précédent (mémorisé).

On distingue deux types de bascules :

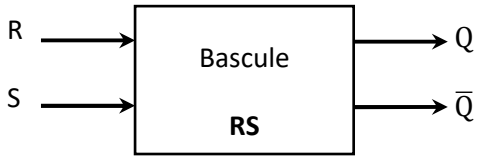
- Les bascules asynchrones, dans lesquels les sorties évoluent dès qu'il y a un changement sur l'une des entrées ;

- Les bascules synchrones, dans lesquels les sorties ne peuvent évoluer que si un signal d'horloge est actif.

4.1 Bascules asynchrones

4.1.1 Bascules RS

Un Une bascule RS comporte deux entrées : Une entrée **R** (**Reset**) de mise à zéro et une entrée **S** (**Set**) de mise à un. Son schéma bloc est donné comme suit :



Son fonctionnement se résume la table condensée suivante :

R	S	Q_{n+1}	$\overline{Q_{n+1}}$	Etat
0	0	Q_n	$\overline{Q_n}$	Mémorisation
0	1	1	0	Mise à 1
1	0	0	1	Mise à 0
1	1	Ø	Ø	Indéterminé

L'état R=S=1 est un état interdit puisqu'il nous donne le deux sorties complémentairesQet $\overline{Q}$ au même état ce qui n'est pas logique. Puisque la sortie Q_{n+1} est fonction de trois variable (R, S, et Q_n), la table de vérité étendue doit être dressée sur 2^3=8 combinaisons :

R	S	Q_n	Q_{n+1}	$\overline{Q_{n+1}}$	Etat
0	0	0	0	1	Mémorisation
0	0	1	1	0	Mémorisation
0	1	0	1	0	Mise à 1

0	1	1	1	0	Mise à 1
1	0	0	0	1	Mise à 0
1	0	1	0	1	Mise à 0
1	1	0	Ø	Ø	Indéterminé
1	1	1	Ø	Ø	Indéterminé

On utilise la table de Karnaugh pour déterminer l'équation logique correspondante :

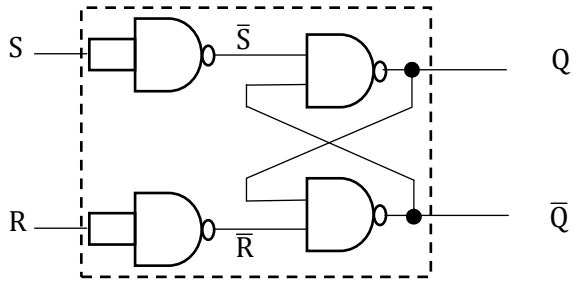
a) – En utilisant les mintermes (la sortie est représentée par ses **1** SDP) :

RS / Q_n	*00*	*01*	*11*	*10*
0	0	1	Ø	0
1	1	1	Ø	0

$$Q_{n+1} = S + \overline{R}Q_n$$

En s'appuyant sur cette équation pour réaliser la bascule RS par des portes logiques **NAND**

$$Q_{n+1} = \overline{\overline{S + \overline{R}Q_n}} = \overline{\overline{S} \cdot \overline{\overline{R}Q_n}}$$



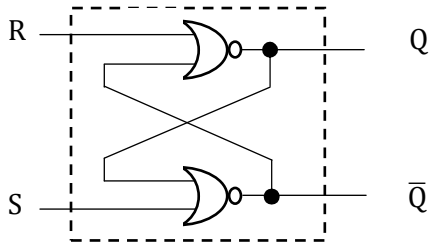
b) – En utilisant les maxtermes (la sortie est représentée par ses **0** PDS) :

RS / Q_n	*00*	*01*	*11*	*10*
0	0	1	Ø	0
1	1	1	Ø	0

$$Q_{n+1} = \overline{R} \cdot (S + Q_n)$$

En s'appuyant sur cette équation pour réaliser la bascule RS par des portes logiques **NOR**

$$Q_{n+1} = \overline{\overline{\overline{R} \cdot (S + Q_n)}} = \overline{R + \overline{(S + Q_n)}}$$



4.1.2 Bascules JK

La bascule JK est obtenue à partir d'une bascule RS dontles sorties sont rebouclées sur les entrées. Ceci permet d'éliminer l'état d'indétermination R=S=1. En effet, quand J=K=1(Au sens normalisé S sera remplacé par J, et R sera remplacé par K, et), on obtient la fonction de complémentation $Q_{n+1} = \overline{Q_n}$.

Son fonctionnement se résume la table condensée suivante :

J	K	Q_{n+1}	$\overline{Q_{n+1}}$	Etat
0	0	Q_n	$\overline{Q_n}$	Mémorisation
0	1	0	1	Mise à 1
1	0	1	0	Mise à 0
1	1	$\overline{Q_n}$	Q_n	Basculement

La sortie Q_{n+1} est fonction de trois variable (R, S, et Q_n), la table de vérité étendue doit être dressée sur 2^3=8 combinaisons :

J	K	Q_n	Q_{n+1}	$\overline{Q_{n+1}}$	Etat
0	0	0	0	1	Mémorisation
0	0	1	1	0	Mémorisation
0	1	0	0	0	Mise à 1
0	1	1	0	0	Mise à 1
1	0	0	1	1	Mise à 0
1	0	1	1	1	Mise à 0
1	1	0	1	0	Basculement
1	1	1	0	1	Basculement

On utilise la table de Karnaugh pour déterminer l'équation logique correspondante :

a) – En utilisant les mintermes (la sortie est représentée par ses **1** SDP) :

Q_n \ JK	*00*	*01*	*11*	*10*
0	0	0	1	1
1	1	0	0	1

$$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n$$

En s'appuyant sur cette équation pour réaliser la bascule RS par des portes logiques **NAND**

$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} + \overline{\overline{K}Q_n}} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{\overline{K}Q_n}}$$

On développe cette équation pour faire apparaitre les entrées J et K.

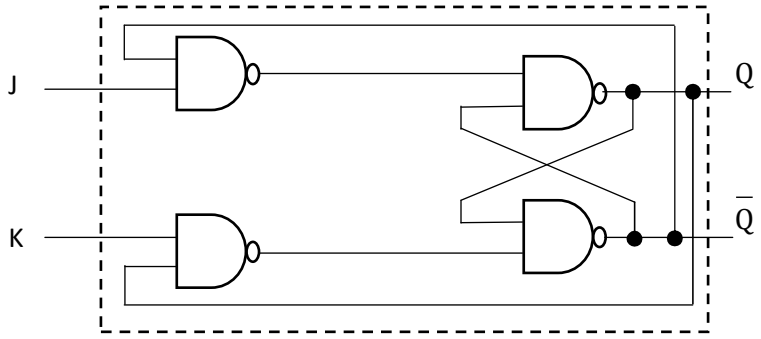
$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{\overline{K}Q_n + (\overline{Q_n}\,.\,Q_n)}}$$

$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{(\overline{K}Q_n + Q_n).(\overline{K}Q_n + \overline{Q_n})}}$$

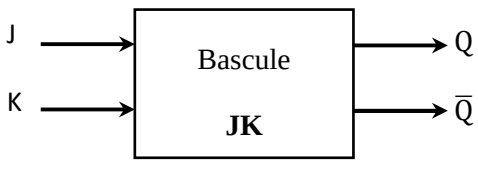
$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{Q_n(\overline{K}+1).((\overline{K}+\overline{Q_n})(Q_n + \overline{Q_n}))}}$$

$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{Q_n.(\overline{K} + \overline{Q_n})}}$$

$$Q_{n+1} = \overline{\overline{J\overline{Q_n}} \;.\; \overline{Q_n.(\overline{K}.Q_n)}}$$



Voici le schéma bloc de la bascule **JK**



4.1.3 Bascules D

Une bascule **D** (Delay) est obtenue à partir d'une bascule **JK** en envoyant simultanément une donnée sur l'entrée **J** et son inverse sur l'entrée **K**. Son principe de fonctionnement est donné par les tables suivantes :

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Par identification à l'équation de la bascule JK,

$$J = D\ , K = \overline{D}$$

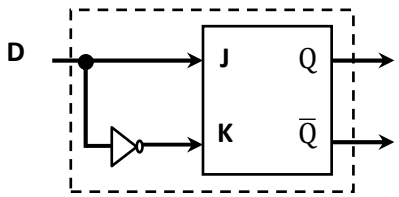
$$Q_{n+1} = D\overline{Q_n} + \overline{\overline{D}}Q_n$$

$$Q_{n+1} = D(\overline{Q_n} + Q_n)$$

$$Q_{n+1} = D$$

D	Q_{n+1}
0	0
1	1

On obtient le schéma logique de la bascule Dsuivant



On note que la bascule D n'a pas grand intérêt puisqu'il s'agit d'un bloc fonctionnel qui ne fait que recopier son entrée, en permanence. On verra plus loin que son utilité apparait avec le signal "d'horloge" (version synchrone).

4.1.4 Bascules T

La bascule T tire son nom du terme anglais *'Toggle'*. Si son entrée **T** est active, la sortie sera elle basculé, sinon elle conserve son état, comme indiqué par la table de fonctionnement suivante :

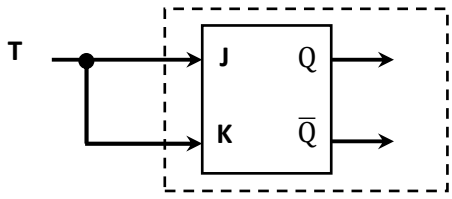
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

$$J = K = T$$

L'équation caractéristique est :$Q_{n+1} = T\overline{Q_n} + \overline{T}Q_n = T \oplus Q_n$

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

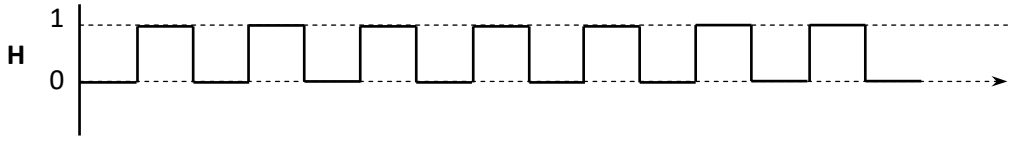
On obtient le schéma logique de la bascule T suivant



4.2 Bascules synchrones

Sur ces bascules, une entrée supplémentaire est destinée à recevoir les impulsions d'horloge. Le circuit n'est opérationnel que pendant la durée de l'impulsion d'horloge ; en dehors de ce temps la sortie reste figée (fixe quel que soit les états d'entrées).Ceci permet de préparer l'état futur et de l'enregistrer uniquement au moment d'une impulsion d'horloge. Cette entrée supplémentaire est notée CLK ou H.

Une horloge génère des impulsions périodiques, qui se présentent sous forme de signaux carrés ou rectangulaires.



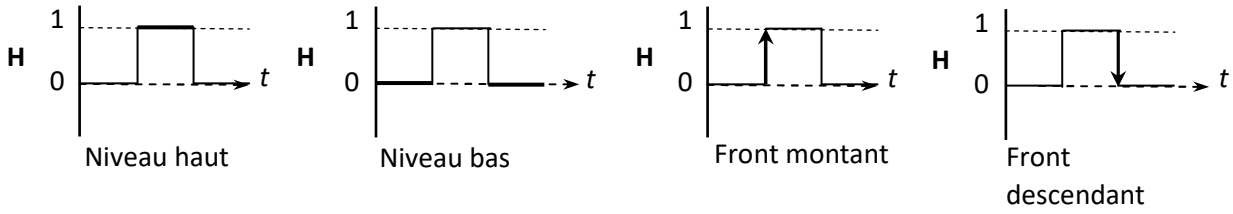
Les bascules sont synchronisées,

- Soit sur niveau haut du signal d'horloge,
- Soit sur niveau bas du signal d'horloge,
- Soit sur front montant du signal d'horloge,

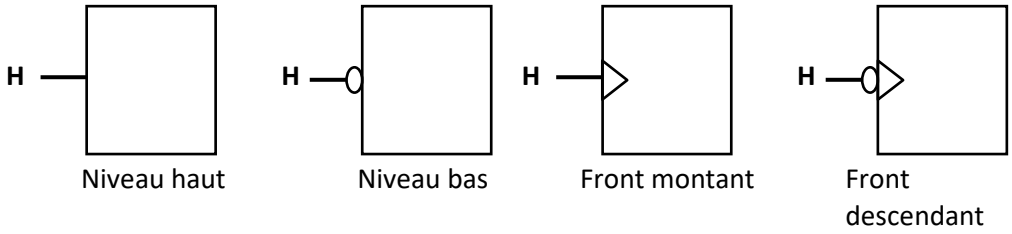
- Soit sur front descendant du signal d'horloge.

Le passage du niveau bas (état 0) au niveau haut (état 1) est appelé **: front montant** ;

Le passage du niveau haut au niveau bas est appelé : **front descendant**.

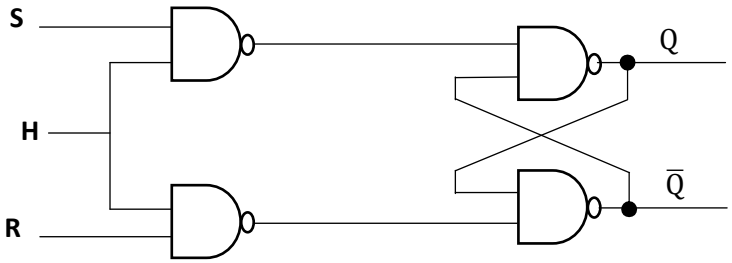


La représentation de l'entrée d'horloge est telle qu'illustrée par les schémas blocs suivants :

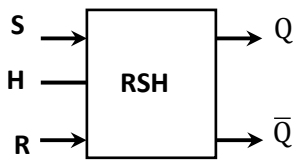


4.2.1 Bascule RS synchrone (RSH ou RS Clk)

La bascule RSH est une bascule pour laquelle les entrées S et R ne sont prises en compte qu'en coïncidence avec le signal de commande H. son logigramme est donné sous la forme suivante pour le cas d'une bascule synchronisée sur niveau haut.



Son schéma bloc est donné comme suit



Si **H**=0, $Q_{n+1} = Q_n$ quelque soit **R** et **S**(la bascule RS est désactivée)

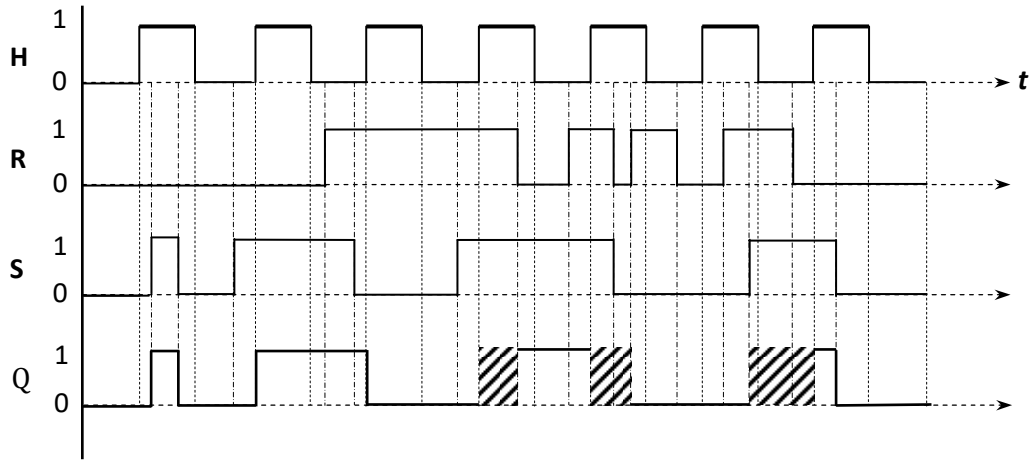
Si **H**=1, Retournement au fonctionnement normal de la bascule RS(la bascule RS est activée)

Son fonctionnement se résume ainsi :

R	S	H	Q_{n+1}	Etat
0	0	0	Q_n	Mémorisation
0	0	1	Q_n	Mémorisation
0	1	0	Q_n	Mémorisation
0	1	1	1	Mise à 1
1	0	0	Q_n	Mémorisation
1	0	1	0	Mise à 0
1	1	0	Q_n	Mémorisation
1	1	1	Ø	Indéterminé

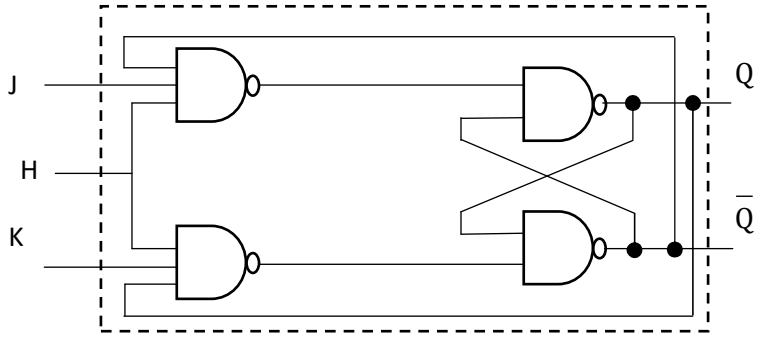
Exemple 4.1

Le chronogramme suivant est un exemple de fonctionnement d'une bascule RSH déclenchée sur niveau haut, avec au départ $Q = 0$:

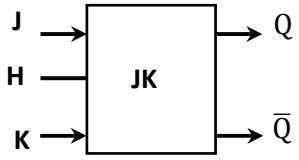


4.2.2Bascule JK synchrone (JKH ou JK Clk)

De même on peut avoir une bascule JK qui fonctionne avec un signal d'horloge. Son logigramme est donné sous la forme suivante pour le cas d'une bascule synchronisée sur niveau haut :



Son schéma bloc est donné comme suit



Si **H**=0, $Q_{n+1} = Q_n$ quelque soit **J** et **K** (la bascule JK est désactivée)

Si **H**=1, Retournement au fonctionnement normal de la bascule JK (la bascule JK est activée)

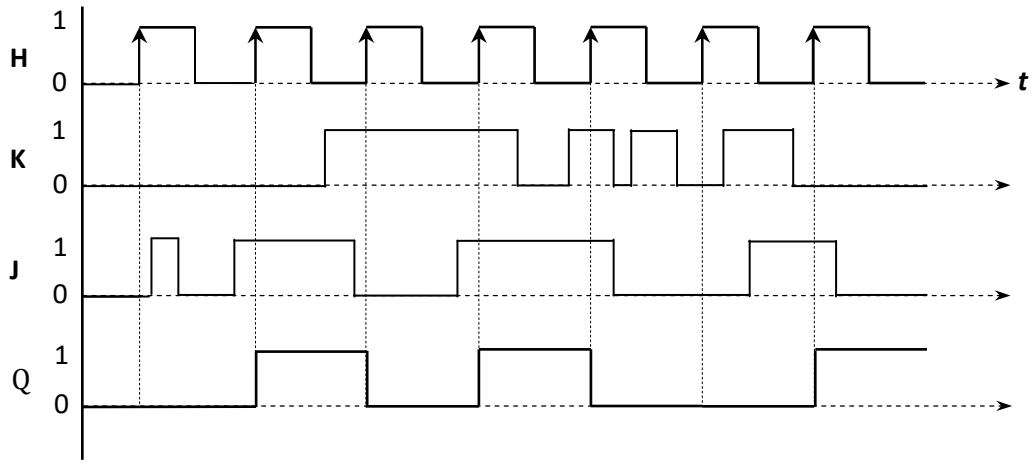
Son fonctionnement se résume ainsi :

K	J	H	Q_{n+1}	Etat
0	0	0	Q_n	Mémorisation
0	0	1	Q_n	Mémorisation
0	1	0	Q_n	Mémorisation
0	1	1	1	Mise à 1
1	0	0	Q_n	Mémorisation
1	0	1	0	Mise à 0
1	1	0	Q_n	Mémorisation
1	1	1	$\overline{Q_n}$	Basculement

La limitation de la JK synchrone est quand J = 1 et K = 1 : La sortie oscille en 0 et 1 pendant toute la durée de l´état haut du signal d'horloge, pour cela on utilise une bascule JK avec déclenchement sur front (montant ou descendant).Une autre solution consiste à utiliser une bascule JK Maître -Esclave.

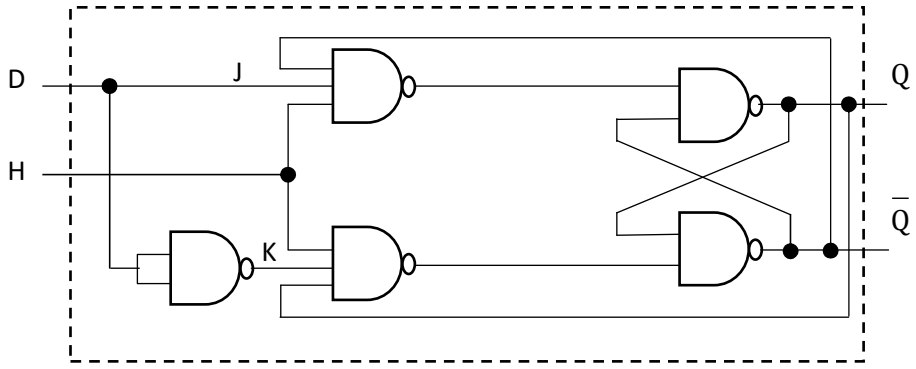
Dans l'exemple suivant on va considérer une bascule **JK** synchronisée sur front montant :

Exemple 4.2 Le chronogramme suivant est un exemple de fonctionnement d'une bascule JK synchrone déclenchée sur front montant du signal d'horloge **H**, avec au départ $Q = 0$:

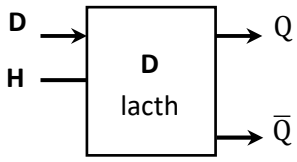


4.2.3 Bascule D à verrou (D latch)

Dans le cas où la bascule est synchronisée sur un niveau haut d'une horloge **H**, la bascule recopie l'entrée **D** en sortie Q quand l'horloge est active **H = 1**. Quand l'horloge est inactive H = 0, la bascule garde l'état précédent. Le schéma logique de la bascule 'D latch' à partir d'une bascule JK synchrone (utilisant les portes NAND uniquement) est le suivant :

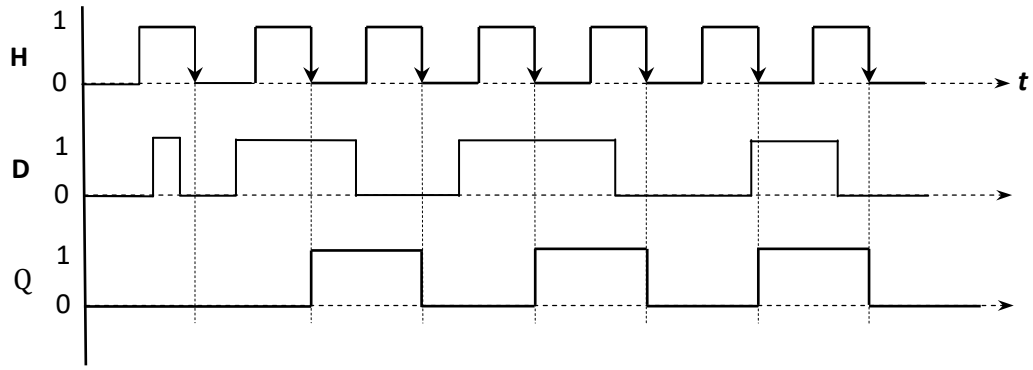


Son schéma bloc est donné comme suit



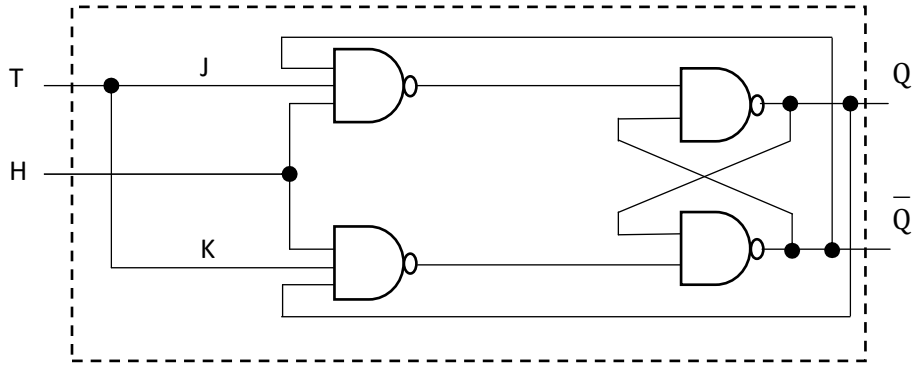
La bascule 'D latch' est très utilisée dans les compteurs synchrones. Dans l'exemple suivant on va considérer une bascule synchronisée sur front descendant :

Exemple 4.3 Le chronogramme suivant est un exemple de fonctionnement d'une bascule D déclenchée sur front descendant du signal d'horloge **H**, avec au départ Q = 0:

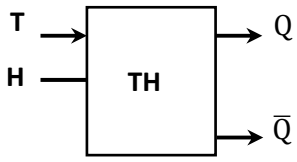


4.2.4 Bascule T synchrone (TH ou T clk)

En rajoutant un signal d'horloge à la bascule T, si la bascule est synchronisée sur un niveau haut d'une horloge **H**, la sortie bascule (changement d´état) dans le cas où l'horloge est active H = 1. Quand l'horloge est inactive H = 0, la bascule garde l'état précédent. Le schéma logique de la bascule 'T'à partir d'une bascule JK synchrone (utilisant les portes NAND uniquement) est le suivant :



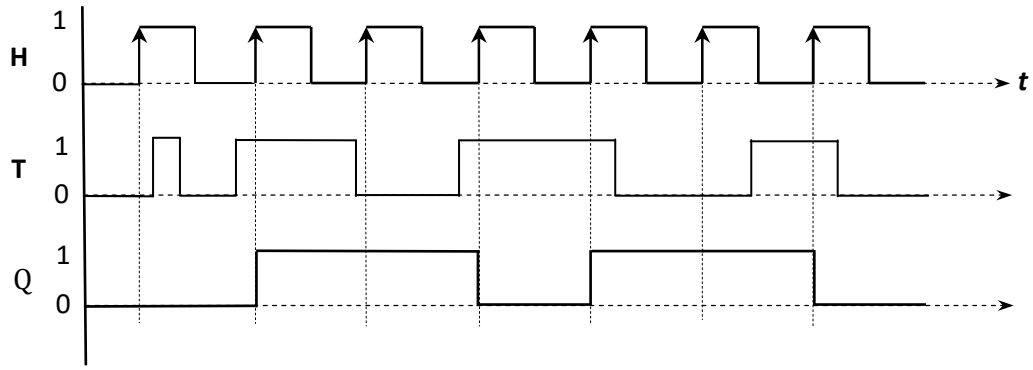
Son schéma bloc est donné comme suit



Dans l'exemple suivant on va considérer une bascule **T** synchronisée sur front montant :

Exemple 4.3

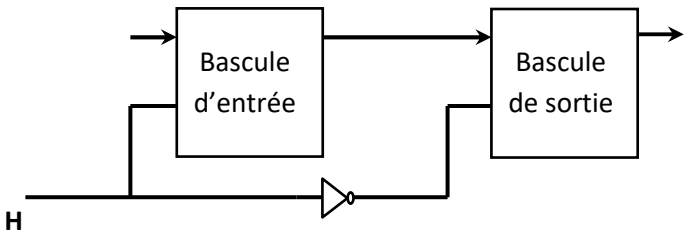
Le chronogramme suivant est un exemple de fonctionnement d'une bascule T déclenchée sur front montant du signal d'horloge **H**, avec au départ $Q = 0$:



4.2.5 Structure maître-esclave

Son principe est illustré dans la figure suivante. La bascule d'entrée est le maître, la bascule de sortie, qui recopie l'état du maître, est l'esclave. On s'aperçoit que l'entrée de la bascule esclave est reliée à la sortie de la sortie de la bascule maitre. La plupart des bascules synchrones sont basées sur ce type de structure.

On remarque que les entrées de commande (horloges) des deux bascules se situent toujours à des niveaux logiques opposés. Cela signifié que lorsque la bascule d'entrée est active, la sortie soit verrouillée, mais que les informations sur les entrées soient prises en compte, donc que les entrées soient ouvertes. A l'inverse, lorsque la bascule d'entrée est désactivée, les entrées devraient être verrouillées et les sorties ouvertes.



4.2.6 Bascule avec des entrées Preset et Clear

Les entrées Preset (Pr) et Clear (clr) sont des entrées asynchrones car ils ne dépendent pas au signal d'horloge, ils permettent d'assigner l'état initial de la bascule (1 ou 0 respectivement), par exemple juste après la mise sous tension pour éviter tout aléa. En fonctionnement normal ces deux entrées doivent être maintenues à 1. Nous avons la table de vérité suivante :

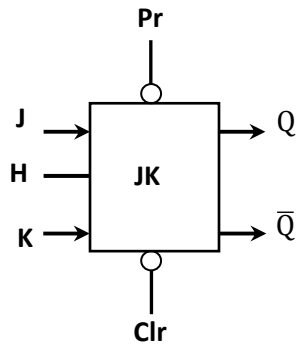
Pr	Clr	Q
1	1	Q
0	1	1
1	0	0

Le fonctionnement de Preset et Clear en logique négative est comme suit :

Preset=0 : entrée de mise à 1 est validée.

Clear=0 : entrée de mise à 0 est validée.

La figure suivante donne la représentation symbolique d'une bascule JK avec les entrées Preset et Clear.



La table de vérité condensée est comme suit :

Pr	Clr	Q
0	0	Cas interdit
0	1	1
1	0	0
1	1	Mode synchrone de la bascule

4.3 Compteurs

Un compteur est un circuit séquentiel constitué d'un circuit combinatoire et d'une succession de *n* bascules décrivant au rythme d'une horloge un cycle de comptage d'un maximum de 2^n combinaisons.

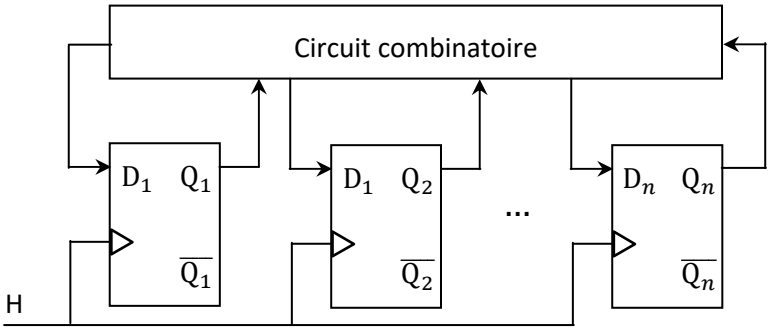
Un compteur constitué de *n* bascule est dit modulo N (tel que $2^n \geq$ N), s'il peut compter de 0 jusqu'à *n*-1. La $N^{ième}$ impulsion le remet, obligatoirement à 0.

Les *n* étages constituant un tel compteur permettent de présenter tous les états possibles. Si un certain nombre d'états ne seront jamais utilisées, en fonctionnement normal ($N = 2^n$),on parle d'un compteur incomplet.

Si $N = 2^n$ alors *n* est le nombre de bascules nécessaires.

4.4.1Compteurs synchrones

Un compteur est dit synchrone, si toutes les bascules sont déclenchées simultanément par le même signal d'horloge. La figure suivante illustre le schéma d'un compteur synchrone utilisant des bascules **D** à front montant.



<u>Exemple 4.4</u> Compteur binaire progressif

Le compteur binaire est dit progressif si son contenu passe d'une valeur binaire *m* à une valeur binaire *m + 1* (sens croissant) après l'application d'une impulsion d'horloge. On considère un compteur binaire formé de trois bascules JK à front montant, et en comptant de 0 jusqu'à 7 (modulo 8 complet). La table de vérité des transitions d'un tel compteur après chaque impulsion d'horloge est illustrée sur la table suivante.

Valeur en décimale	Contenu du compteur		
	Q_3	Q_2	Q_1
0	0	0	0
1	0	0	1
2	1	1	0
3	1	1	1
4	0	0	0
5	0	0	1
6	1	1	0
7	1	1	1

En s'appuyant sur la table d'excitation de la bascule JK suivante :

Q	Q^+	J	K	Observations
0	0	0	Ø	Mémorisation ou bien Reset
0	1	1	Ø	Basculement ou bien Set
1	0	Ø	1	Basculement ou bien Reset
1	1	Ø	0	Mémorisation ou bien Set

On note Q_1^+, Q_2^+, Q_3^+ les états futurs de Q_1, Q_2, Q_3 respectivement. On obtient la table de vérité d'implication suivante pour les trois bascules :

Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	0	Ø	0	Ø	1	Ø
0	0	1	0	1	0	0	Ø	1	Ø	Ø	1
0	1	0	0	1	1	0	Ø	Ø	0	1	Ø
0	1	1	1	0	0	1	Ø	Ø	1	Ø	1
1	0	0	1	0	1	Ø	0	0	Ø	1	Ø
1	0	1	1	1	0	Ø	0	1	Ø	Ø	1
1	1	0	1	1	1	Ø	0	Ø	0	1	Ø
1	1	1	0	0	0	Ø	1	Ø	1	Ø	1

On utilise la table de Karnaugh pour déterminer les équations logiques correspondantes de $J_1, K_1, J_2, K_2, J_3, K_3$ en fonction de Q_1, Q_2, Q_3

$J_1 = ?$

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	1	1	1	1
1	Ø	Ø	Ø	Ø

$$J_1 = 1$$

$K_1 = ?$

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	Ø	Ø	Ø	Ø
1	1	1	1	1

$$K_1 = 1$$

J_2 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	0	Ø	Ø	0
1	1	Ø	Ø	1

$$J_2 = Q_1$$

K_2 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	Ø	0	0	Ø
1	Ø	1	1	Ø

$$K_2 = Q_1$$

J_3 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	0	0	Ø	Ø
1	0	1	Ø	Ø

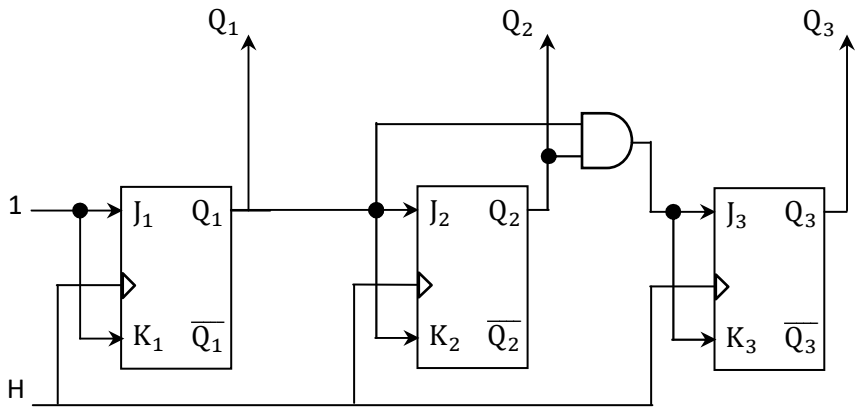
$$J_3 = Q_1Q_2$$

K_3 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	Ø	Ø	0	0
1	Ø	Ø	1	0

$$K_3 = Q_1Q_2$$

Son logigramme est donné comme suit :



Exemple 4.4

Compteur binaire progressif Compteur à cycle incomplet modulo 6 en utilisant des bascules T :

Ce compteur compte de 0 à 5Sachant que $2^2<6<2^3$) donc on y besoin de 3 bascules.

La table de vérité des transitions d'un tel compteur après chaque impulsion d'horloge est illustrée sur la table suivante.

Valeur en décimale	Contenu du compteur		
	Q_3	Q_2	Q_1
0	0	0	0
1	0	0	1
2	1	1	0
3	1	1	1
4	0	0	0
5	0	0	1

La table d'excitation de la bascule T est :

Q	Q^+	T	Observations
0	0	0	Mémorisation
0	1	1	Basculement
1	0	1	Basculement
1	1	0	Mémorisation

On complète la table d'implication du compteur synchrone progressif et régulier à cycle incomplet modulo 6.

Q_3	Q_2	Q_1	Q_3^+	Q_2^+	Q_1^+	T_3	T_2	T_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	Ø	Ø	Ø
1	1	1	0	0	0	Ø	Ø	Ø

On utilise la table de Karnaugh pour déterminer les équations logiques correspondantes de T_1, T_2, T_3 en fonction de Q_1, Q_2, Q_3

T_1 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	1	1	Ø	1
1	1	1	Ø	1

$$T_1 = 1$$

T_2 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	0	0	Ø	0
1	1	1	Ø	1

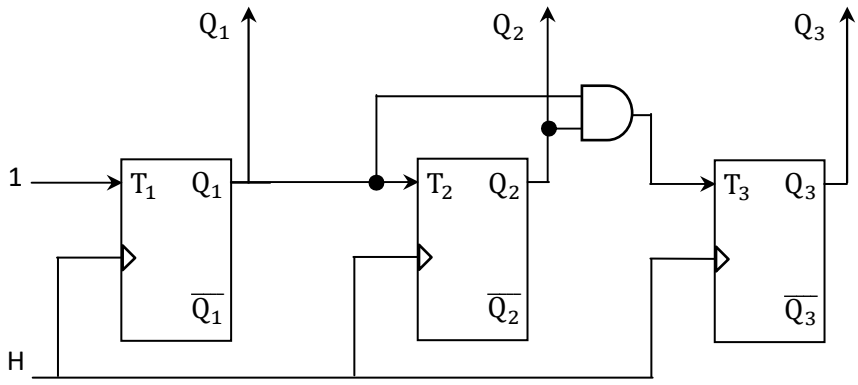
$$T_2 = Q_1$$

T_3 =?

Q_1 \ Q_3Q_2	*00*	*01*	*11*	*10*
0	0	0	Ø	0
1	0	1	Ø	0

$$T_3 = Q_1Q_2$$

Son logigramme est donné comme suit :



4.4.2 Compteurs asynchrones

Le compteur est dit asynchrone, si le signal d'horloge est appliqué seulement à la première bascule, et l'état de chaque bascule est fonction des états des bascules précédentes.

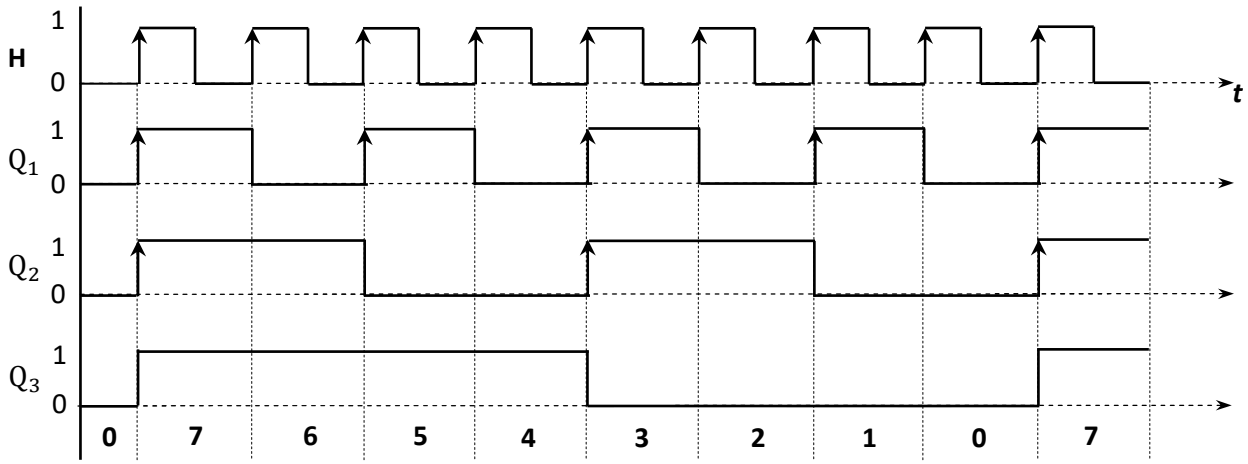
Exemple 4.5Compteur asynchrone régressif et régulier modulo 8 (décompteur de 7 à 0): $8 = 2^3$donc on y besoin de 3 bascules. Soit 3 bascules JK à front montant. Son fonctionnement est :

Q_1 change d'état (basculement) pour chaque front montant de l'horloge H. Cela est obtenu en imposant $J_1 = K_1 = 1$

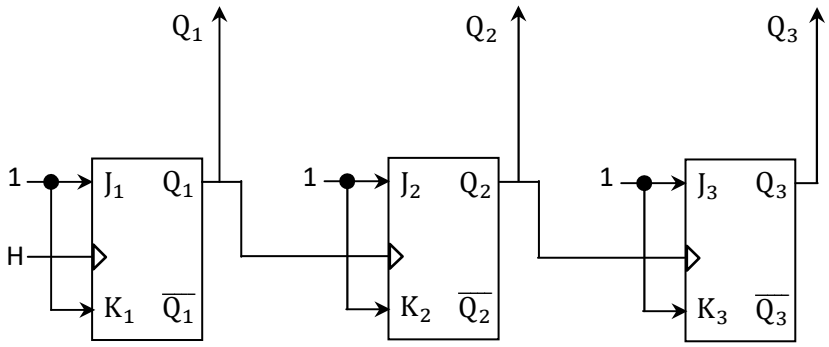
Q_2 change d'état (basculement) pour chaque front montant de Q_1. Cela est obtenu en imposant $J_2 = K_2 = 1$

Q_3 change d'état (basculement) pour chaque front montant de Q_2. Cela est obtenu en imposant $J_3 = K_3 = 1$

Le chronogramme de fonctionnement du compteur asynchrone régressif et régulier modulo8 (à cycle complet) est le suivant :



Le schéma du décompteur asynchrone régulier modulo 8 est le suivant :



<u>Exemple 4.6</u> Compteur asynchrone progressif et régulier modulo 5 (de 0 à 4) :

Ce compteur compte de 0 à 4 Sachant que $2^2<5<2^3$) donc on y besoin de 3 bascules JK à front descendant. Le compteur modulo 5 réalise un compte de 0 à 4 (de 000 à 100 en binaire), arrivé à 4, le comptage doit être interrompu pour recommencer de 0. On doit, donc, remettre toutes les bascules à 0 après l'apparition de 4. Pour cela, on utilise les entrées asynchrones de remise à zéro (Clr).

On détermine l'équation F de l'entrée de remise à zéro (entrée de forçage `a 0). La table de vérité d'implication séquentielle est :

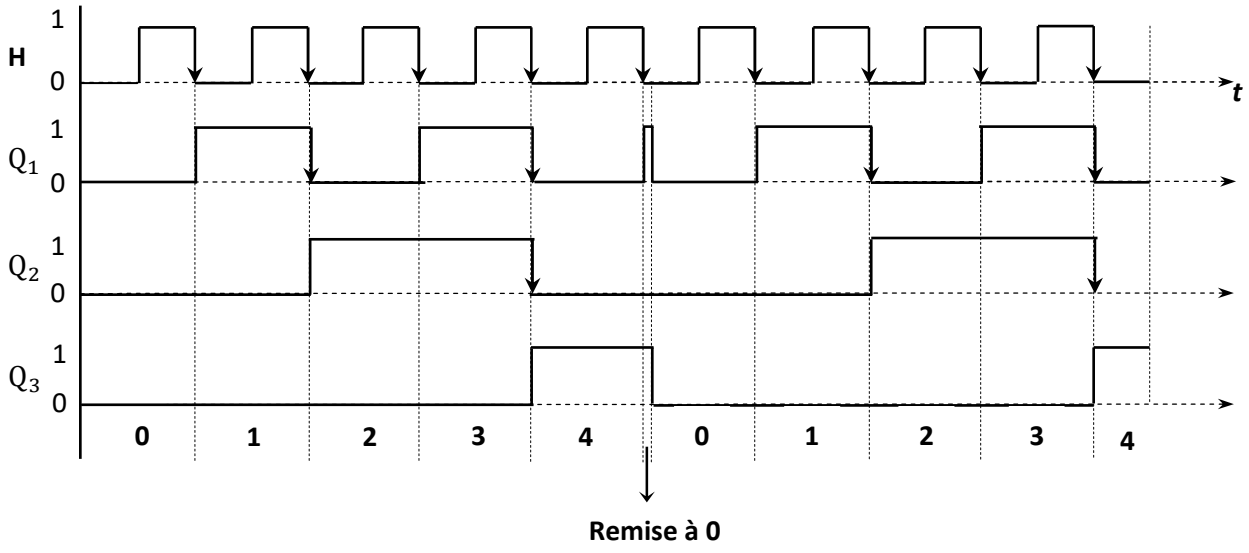
N°	Q_3	Q_2	Q_1	F	
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	0	
4	1	0	0	0	
5	1	0	1	1	Remise à zéro

$Q_1Q_2Q_3 = 101$est un état temporaire existant pendant une durée très courte (temps de réponse à la remise à 0), on néglige ce temps transitoire. On utilise la table de Karnaugh pour déterminer l'équation logique de F en fonction de Q_1, Q_2, Q_3

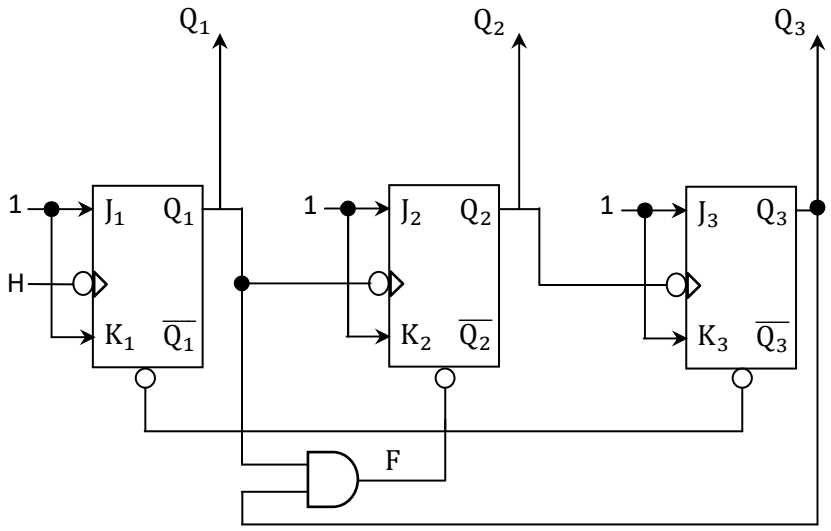
Q_3Q_2 / Q_1	*00*	*01*	*11*	*10*
0	0	0	Ø	0
1	0	0	Ø	1

$$F = Q_1Q_3$$

Le chronogramme de fonctionnement du compteur asynchrone à front descendant, progressif et régulier modulo 5 (à cycle incomplet) est le suivant :



Le schéma du compteur asynchrone régulier modulo 5, en utilisant les bascules JK, est le suivant :



4.5 Registres

Un registre permet la mémorisation de *n* bits. Il est donc constitué de *n* bascules, mémorisant chacune un bit. L'information est emmagasinée sur un signal de commande et disponible en lecture. Un registre est une association de mémoires unitaires (exemple : bascule D) avec laquelle il est possible d'effectuer les trois opérations suivantes :

a) – Enregistrer à un instant déterminé une information constituée par une suite quelconque de 0 et de1 que l'on appelle mot binaire. (exemple : 1101 est un mot de 04 bits).C'est la fonction écriture (**write**) ou chargement (**load**) du registre.

b)- conserver ce mot en mémoire aussi longtemps que c'est nécessaire c'est la fonction mémorisation (**storage)**.

c)- restituer le mot enregistré à la demande une ou plusieurs fois de suite : extraire l'information contenue dans le registre sans la détruire. C'est la fonction lecture (**Read**).

On peut considérer les registres comme des mémoires actives ayant une capacité d'un seul mot binaire de N bits. Ce mot binaire de N bits peut être exploité :

- En **série** ; si on opère sur chaque bit l'un après l'autre.
- En **parallèle** ; si on opère sur tous les bits simultanément.

Il y a quatre types principaux des registres :

1/- A écriture et lecture en parallèle.

2/- A écriture et lecture série.

3/- A écriture en parallèle et lecture en série.

4/- A écriture en série et lecture en parallèle.

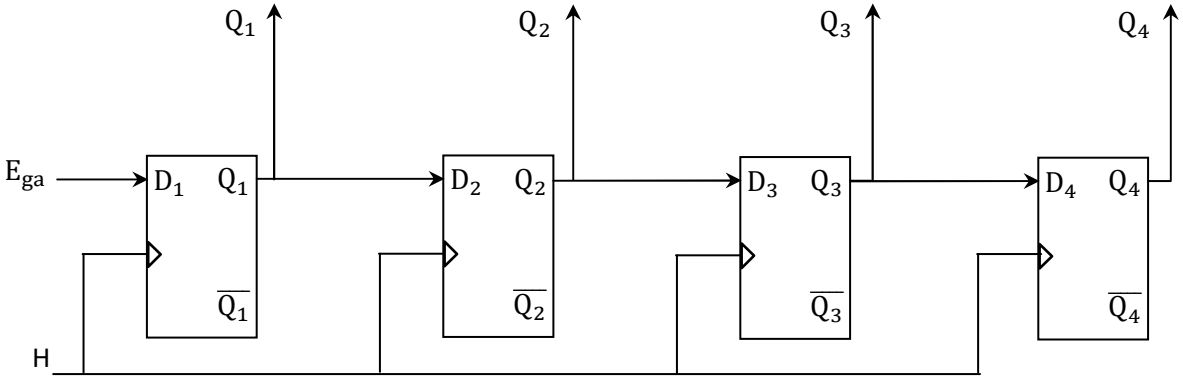
Le registre à écriture et lecture série est très utilisé comme dispositif d'introduction d'un retard dans la transmission d'informations série : le registre est alors utilisé comme **« un registre à décalage ».**Dans ce type, les bits sont présentés les uns après les autres sur l'entrée de la première bascule et sortent les uns après les autres, dans le même ordre sur la sortie Q de la dernière bascule. Il y'a alors un décalage entre l'apparition d'un bit donné sur la sortie du registre et sa présentation à l'entrée du registre, d'autant d'impulsions d'horloges que le registre comporte de bascules.

4.5.1Registre à décalage à droite

C'est un registre dont les informations introduites en série dans la première bascule située la plus à gauche se décale vers la droite au rythme d'impulsions d'horloge. Ce registre possédant une entrée à gaucheE_{ga} et *n* sorties ($Q_1, Q_2, \ldots Q_n$).Q_1est de poids fort MSB.

<u>Exemple 4.7</u>Registre à décalage à droite formé de 4 bascules de type D à front montant : soit à charger le contenu du registre par la valeur 1010.

Le schéma du registre à décalage à droite à 4 bascules D à front montant est représenté par la figure suivante :



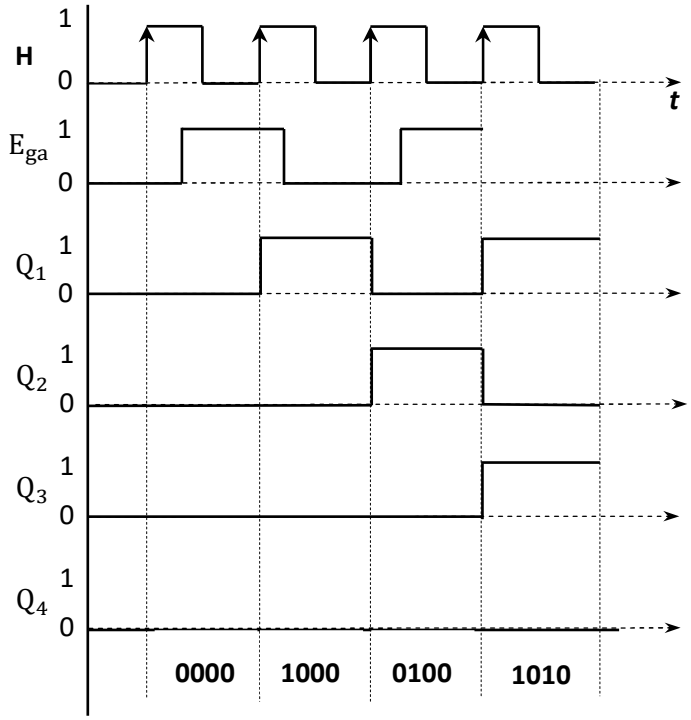
A l'état initial toutes les bascules sont considérées à zéro. Selon la caractéristique de la bascule D, pour chaque impulsion d'horloge on a

$$\begin{cases} D_4 = Q_3 \\ D_3 = Q_2 \\ D_2 = Q_1 \\ D_1 = E_{ga} \end{cases}$$

Le chargement de valeur 1010 dans le registre à décalage à droite est montré dans le chronogramme suivant, on constate qu'après quatre impulsions on a :

$$\begin{cases} Q_1 = 1 \\ Q_2 = 0 \\ Q_3 = 1 \\ Q_4 = 0 \end{cases}$$

Et E_{ga} doit être préparé avant chaque front montant(0_1 _0_1)



4.5.2 Registre à décalage à gauche

C'est un registre dont les informations introduites en série dans la première bascule située la plus à droite se décalent vers la gauche au rythme d'impulsions d'horloge. Ce registre possédant une entrée à gauche E_{dr} et *n* sorties $(Q_1, Q_2, \ldots Q_n)$.Q_1 est de poids fort MSB.

Exemple 4.8

Registre à décalage à gauche formé de 4 bascules de type D à front montant : soit à charger le contenu du registre par la valeur 1001.

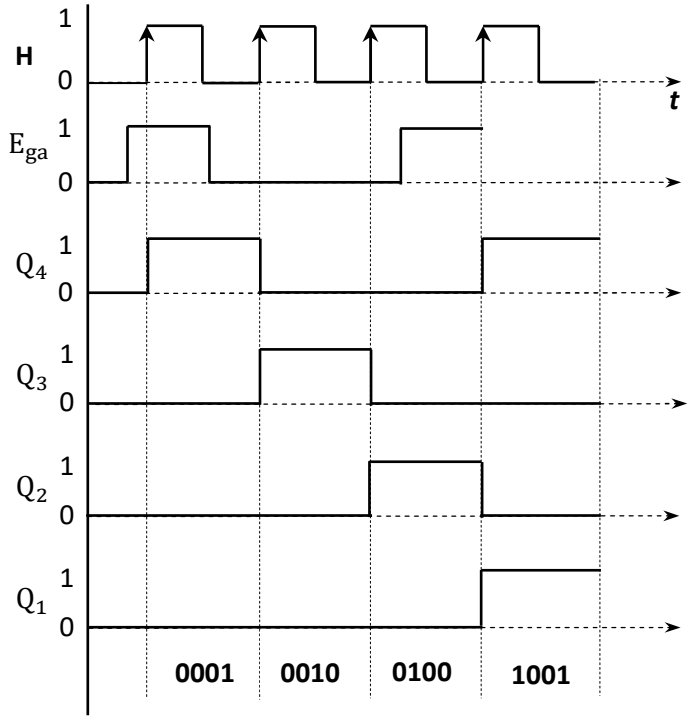
Dans ce registre, pour chaque impulsion d'horloge on a

$$\begin{cases} D_1 = Q_2 \\ D_2 = Q_3 \\ D_3 = Q_4 \\ D_4 = E_{dr} \end{cases}$$

Le chargement de valeur 1001 dans le registre à décalage à gauche est montré dans le chronogramme suivant, on constate qu'après quatre impulsions on a :

$$\begin{cases} Q_1 = 1 \\ Q_2 = 0 \\ Q_3 = 0 \\ Q_4 = 1 \end{cases}$$

Et E_{dr} doit être préparé avant chaque front montant (1_0_0_1)



EXERCICES

Exercice 3.1

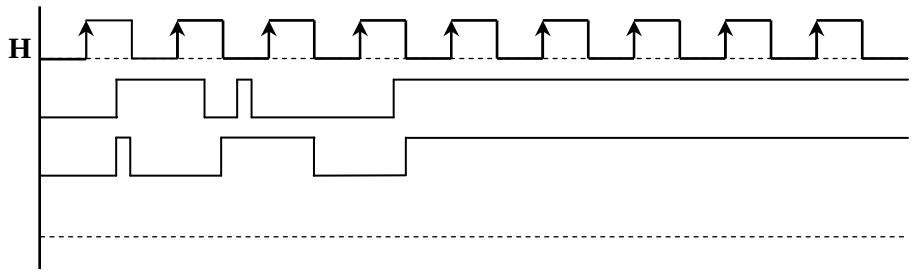
1- Rappeler la table de vérité d'une bascule **JK**.

2- Donner l'expression logique de $\boldsymbol{Q_{n+1}}$ en fonction de $\boldsymbol{J}$, $\boldsymbol{K}$ et $\boldsymbol{Q_n}$.

3- Prouver que : $\boldsymbol{Q_{n+1}} = \overline{\boldsymbol{J}.\overline{\boldsymbol{Q_n}}} \;.\; \overline{(\overline{\boldsymbol{K}.\boldsymbol{Q_n}}).\boldsymbol{Q_n}}$

4- Tracer le logigramme correspondant en utilisant des portes **NAND** à **deux** entrées.

5- Remplir le chronogramme suivant sachant que la bascule **JK** est synchronisée sur **front montant** d'une horloge H.

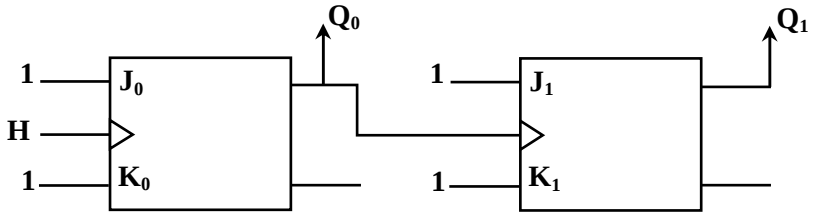


6- Comment peut-on synthétiser une bascule **T** (Toggle) à partir d'une bascule **JK**? Donner sa table de vérité réduite.

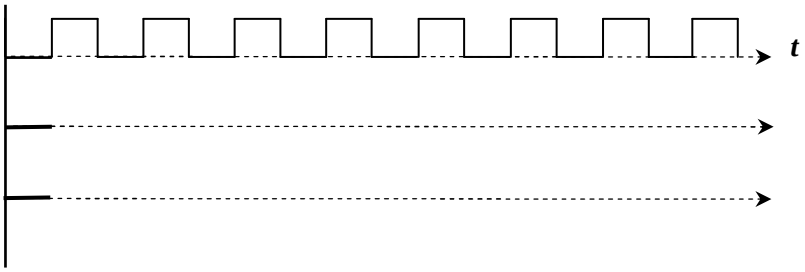
Exercice 3.2

1- Rappeler la table de vérité d'une bascule **JK**.

2- Soit le schéma structurel suivant :



- Sur quel front fonctionnent les deux bascules ($\mathbf{J_0K_0}$ et $\mathbf{J_1K_1}$) ?
- Sur quel front fonctionnent les deux bascules($\mathbf{J_0K_0}$ et $\mathbf{J_1K_1}$) ?
- Compléter les chronogrammes de $\boldsymbol{Q_0}$ et $\boldsymbol{Q_1}$ tel que ($\boldsymbol{Q_0} = \boldsymbol{Q_1} = 0$ à $t = 0$)

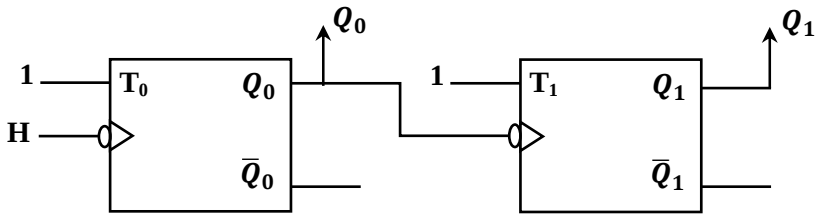


3- Sachant que l'horloge **H** est de fréquence **f=10KHz**, Déterminer les fréquences des signaux de sorties $\boldsymbol{Q_0}$et $\boldsymbol{Q_1}$.

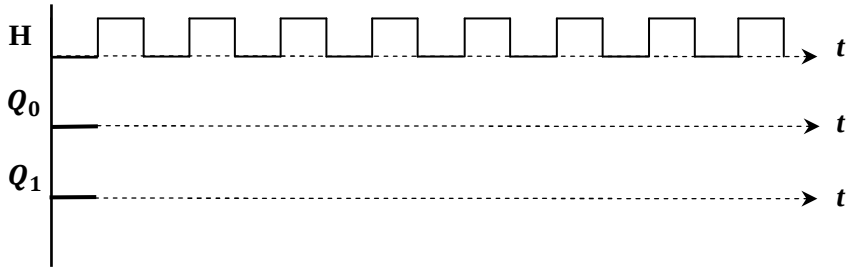
Exercice 3.3

1- Rappeler la table de vérité d'une bascule **T**.

2- Soit le schéma structurel suivant :



- Sur quel front fonctionnent les deux bascules($\mathbf{T_0}$ et$\mathbf{T_1}$) ?
- Compléter les chronogrammes de $\boldsymbol{Q_0}$et $\boldsymbol{Q_1}$ tel que ($\boldsymbol{Q_0} = \boldsymbol{Q_1} = 0$ à $t = 0$)

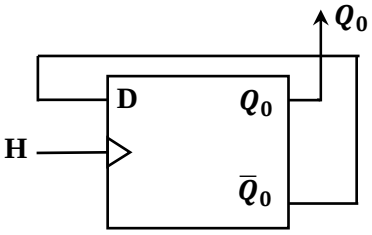


3- Sachant que l'horloge **H** est de fréquence **f=20KHz**, Déterminer les fréquences des signaux de sorties $\boldsymbol{Q_0}$et $\boldsymbol{Q_1}$.

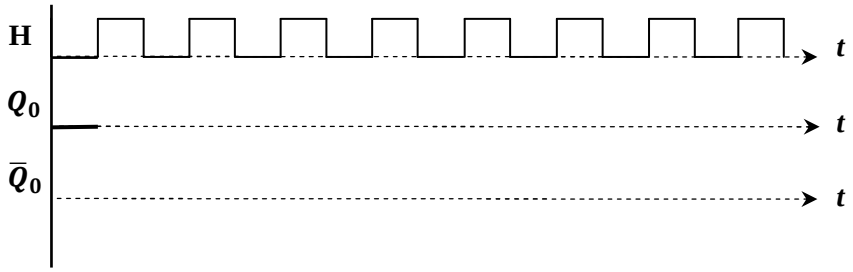
Exercice 3.4

Rappeler la table de vérité d'une bascule **D**.

1- Soit le schéma structurel suivant :



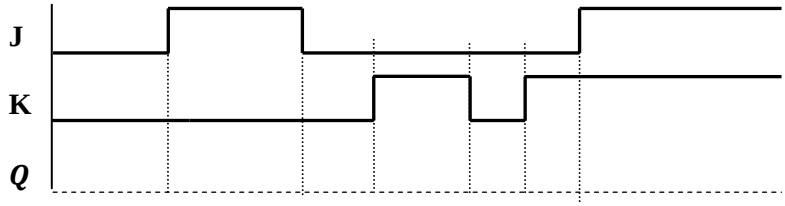
- Sur quel front fonctionne la bascule **D** ?
- Compléter les chronogrammes de $\boldsymbol{Q_0}$et $\overline{\boldsymbol{Q}}_{\boldsymbol{0}}$ tel que ($\boldsymbol{Q_0} = 0$ à $t = 0$)



2- Sachant que l'horloge **H** est de fréquence **f=5KHz**, Déterminer la fréquence du signal de sortie$\boldsymbol{Q_0}$.

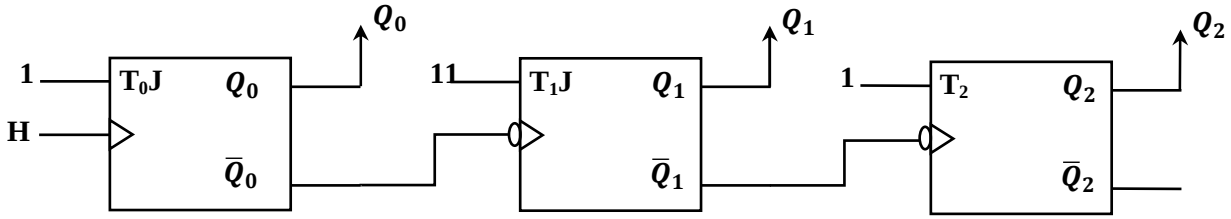
Exercice 3.5

Compléter le chronogramme suivant pour une bascule **JK** asynchrone tel que ($\boldsymbol{Q} = 0$ à $t = 0$)

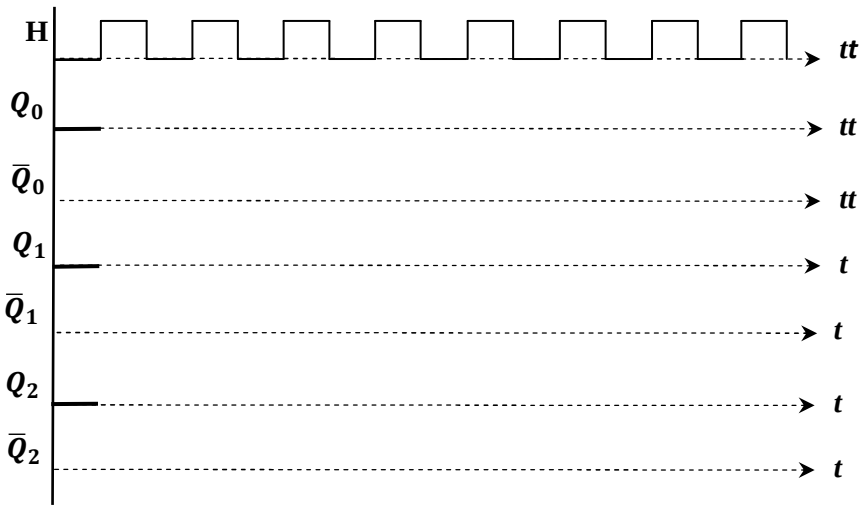


Exercice 3.6

1- Soit le schéma structurel suivant :



- Sur quel front fonctionnent les trois bascules ($\mathbf{T_0}$, $\mathbf{T_1}$ et $\mathbf{T_2}$) ?
- Compléter les chronogrammes suivants tel que ($\boldsymbol{Q_0 = Q_1 = Q_2 = 0}$ à $t = 0$)



2- Sachant que l'horloge **H** est de fréquence **f=40KHz**, Déterminer les fréquences des signaux de sorties $\boldsymbol{Q_0}$, $\boldsymbol{Q_1}$ et $\boldsymbol{Q_2}$.

SOLUTIONS

Exercice 3.1

1- Table de vérité d'une bascule JK

Table de vérité réduite

J	K	$\boldsymbol{Q_{n+1}}$
0	0	$\boldsymbol{Q_n}$
0	1	**0**
1	0	**1**
1	1	$\boldsymbol{\overline{Q_n}}$

Table de vérité étendue

J	K	Q_n	$\boldsymbol{Q_{n+1}}$
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

Expression logique de $\boldsymbol{Q_{n+1}}$ en fonction de **J**, **K** et $\boldsymbol{Q_n}$.

Table de KARNAUGH $\boldsymbol{Q_{n+1}}$

$\boldsymbol{Q_n}$ \ *JK*	*00*	*01*	*11*	*10*
0	**0**	**0**	**1**	**1**
1	**1**	**0**	**0**	**1**

$$Q_{n+1} = J\overline{Q_n} + \overline{K}\,Q_n$$

2-

$$Q_{n+1} = J\overline{Q_n} + \overline{K}\,Q_n = \overline{\overline{J\overline{Q_n}} + \overline{\overline{K}\,Q_n}}$$

$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{\overline{K}Q_n}}$$

$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{\overline{K}Q_n + (Q_n.\overline{Q_n})}}$$

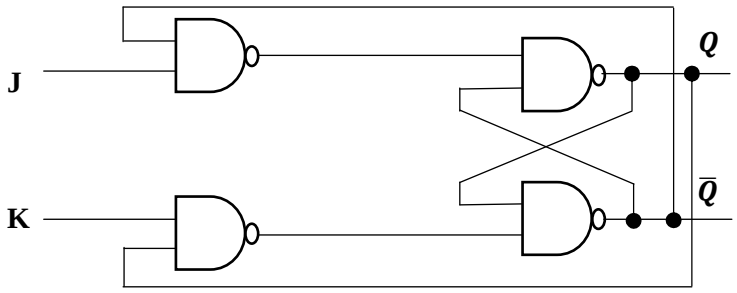
$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{(\overline{K}Q_n+Q_n).(\overline{K}Q_n + \overline{Q_n})}}$$

$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{(\overline{K}+1)Q_n\ .\ (Q_n + \overline{Q_n})(\overline{K} + \overline{Q_n})}}$$

$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{Q_n\,.\,(\overline{K} + \overline{Q_n})}}$$

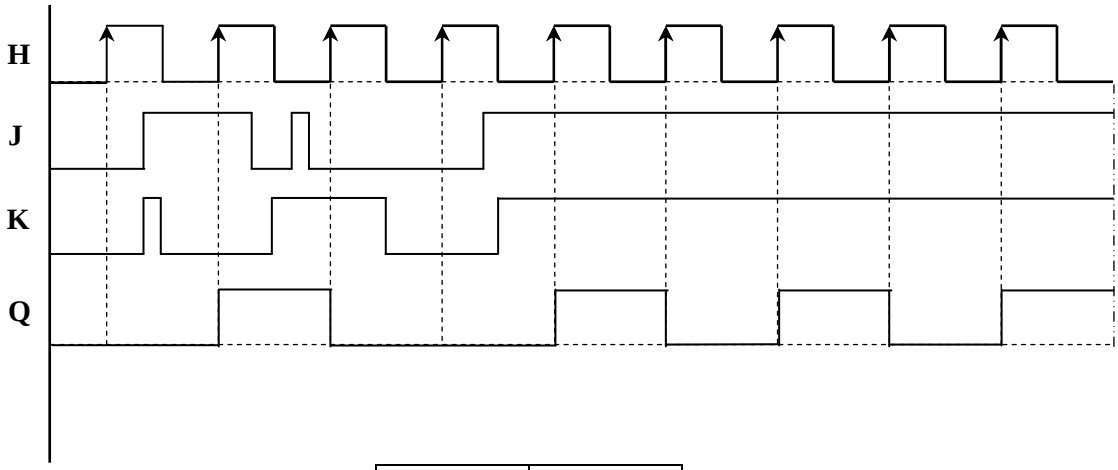
$$= \overline{\overline{J\overline{Q_n}}\ .\ \overline{Q_n\,.\,(\overline{K.Q_n})}}$$

3- Réalisation de la bascule **JK**à l'aide desportes **NAND**



4- Chronogramme

5- On peut synthétiser une bascule **T**en prenant **T=J=K**



T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

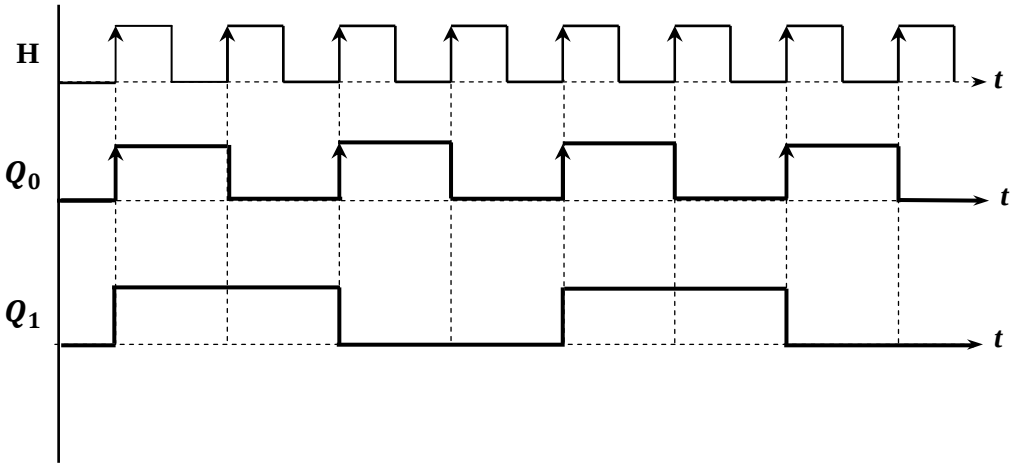
Exercice 3.2

2- Table de vérité d'une bascule JK **(voir exercice précédant**)

La première bascule (**J_0K_0**) est active sur **front montant** de l'horloge **H**.

La deuxième bascule (**J_1K_1**) est active sur **front montant**du signal Q_0

- Chronogramme



La période de l'horloge **H** est $T_H = 1/f$,

d'apré le chronogramme : la période de $\boldsymbol{Q_0}$ est $T_0 = 2.T_H$

la période de $\boldsymbol{Q_1}$ est $T_1 = 2.T_0 = 4.T_H$

donc : la fréquence de $\boldsymbol{Q_0}$ est $f_0 = 1/T_0 = 1/(2.T_H) = f/2 = 10KHz/2 = 5$ KHz

la fréquencede $\boldsymbol{Q_1}$est $f_1 = 1/T_1 = 1/(2.T_0) = f_0/2 = 5KHz/2 = 2.5$ KHz

Exercice 3.3

1- Table de vérité d'une bascule T

Table de vérité réduite

T	$\boldsymbol{Q_{n+1}}$
0	$\boldsymbol{Q_n}$
1	$\boldsymbol{\overline{Q_n}}$

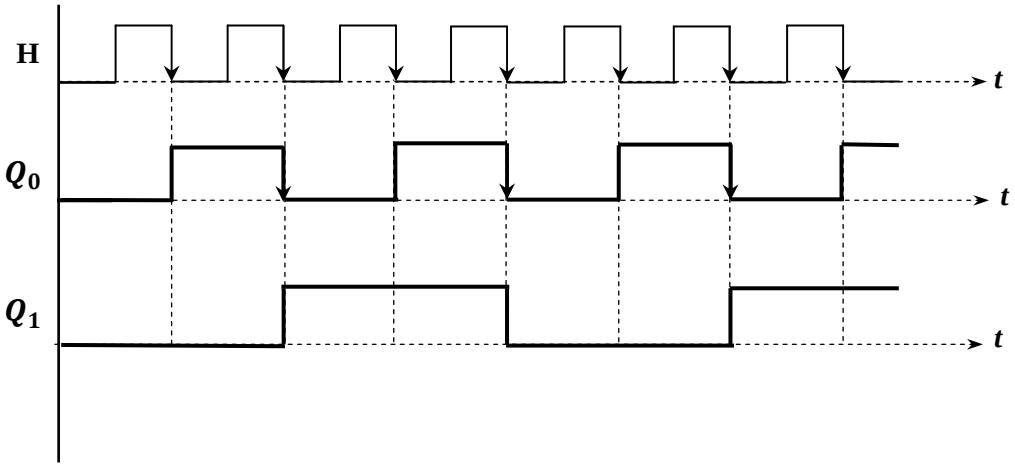
Table de vérité étendue

T	Q_n	$\boldsymbol{Q_{n+1}}$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

La première bascule ($\mathbf{T_0}$) est active sur **front descendant** de l'horloge **H**.

La deuxième bascule ($\mathbf{T_1}$) est active sur **front descendant** du signal $\boldsymbol{Q_0}$

- Chronogramme



La période de l'horloge **H**est t_H =1/f ,

d'apré le chronogramme : la période de $\boldsymbol{Q_0}$ est t_0 =2.t_H

la période de $\boldsymbol{Q_1}$est t_1 =2.t_0 = 4.t_H

donc :

la fréquence de $\boldsymbol{Q_0}$ est f_0= 1/t_0 = 1/(2.t_H) = f/2 = 20KHz/2 = 10 KHz

la fréquence de $\boldsymbol{Q_1}$est f_1 = 1/t_1 = 1/(2.t_0) = f_0/2 = 10KHz/2 = 5 KHz

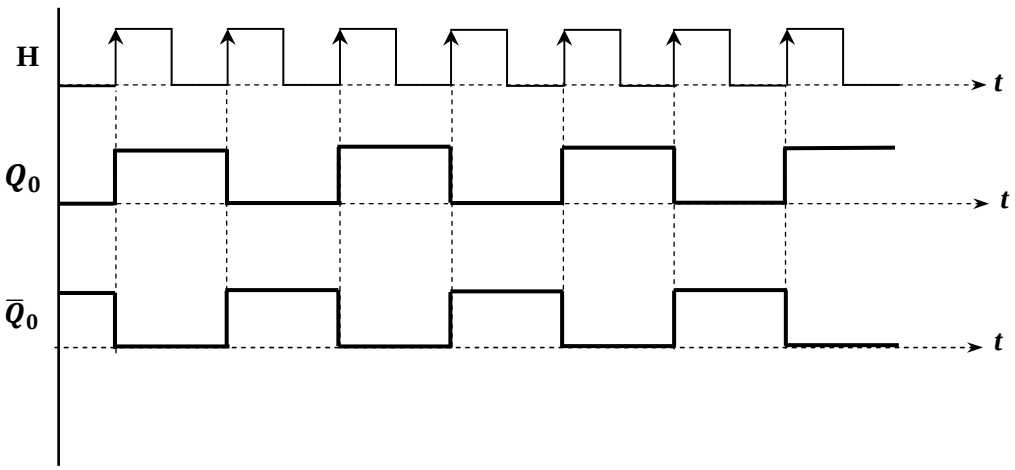
Exercice 3.4

1- Table de vérité d'une bascule D

D	Q_{n+1}
0	**0**
1	**1**

2- La bascule (**D**) est active sur **front montant** de l'horloge **H**.

- Chronogramme

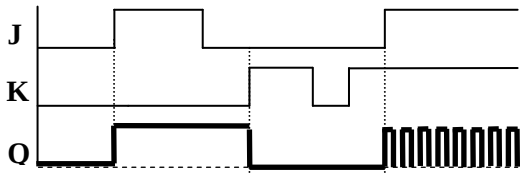


La période de l'horloge **H**est t_H =1/f ,

d'apré le chronogramme : la période de $\boldsymbol{Q_0}$ est $t_0 = 2.t_H$

donc : la fréquence de $\boldsymbol{Q_0}$ est $f_0 = 1/t_0 = 1/(2.t_H) = f/2 = 5KHz/2 = 2.5$ KHz

Exercice 3.5



Bibliographie

[1] Mc. Belaid et collectif, Logique combinatoire et séquentielle, Presses de Mitidja,Baraki, Alger, Algérie, 2010.

[2] S. Tisserant , ESIL ,« Architecture et Technologie des Ordinateurs »2003.

[3] M. Sbaï, Electronique numérique, logique combinatoire et composants numérique, Ellipses Editions Marketing, Paris, France, 2013.

[4] C. Brie, Logique combinatoire et séquentielle, Ellipses, 2002.

[5] R. Katz, Contemporary Logic Design, 2nd ed. Prentice Hall, 2005.