République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université ABDERRAHMANE MIRA – Béjaia –
Faculté de la Technologie
Département : Automatique-Télécom-Electronique (A.T.E)

SUPPORT DE COURS DU MODULE :

LOGIQUE COMBINATOIRE ET SEQUENTIELLE

U.E.F 22.12

Réalisé par :

DR. SLIMANE HADJI

Année universitaire 2022/2023

Table des matières

Introduction générale

Ce cours est destiné aux étudiants de 2ème année des spécialités : Automatique, Télécommunications et Electronique (A.T.E); ayant acquis la 1ère année de sciences technologiques (ST).

C'est la porte de l'électronique numérique et des systèmes embarqués. Le but principal est de connaître les circuits combinatoires usuels, savoir représenter quelques applications des circuits combinatoires en utilisant les outils standards qui sont les tables de vérité et les tables de Karnaugh, introduire les circuits séquentiels à travers les circuits bascules et les compteurs.

Le module « Logique Combinatoire et Séquentielle » est placé dans une unité fondamentale avec coefficient : 2 et crédits : 4. Son volume horaire est :

Volume horaire semestriel : 45 heures (réparties en 15 semaines)

Volume hebdomadaire : Cours : 1h30, Travaux Dirigés : 1h30

L'évaluation est partagée entre control continu (travaux dirigés TD) : 40 %, et un examen final : 60 %.

Le cours est composé de six (6) chapitres : Un chapitre de rappel sur les systèmes de numération et codage de l'information, deux chapitres sur la logique combinatoire, deux autres sur la logique séquentielle et un chapitre sur la technologie des circuits intégrés.

Chapitre I. Systèmes de numération et Codage de l'information

Introduction

La représentation de l'information a largement évolué, en passant par différentes formes : dessins, paroles, écriture, et finir par des fichiers numériques.

Bien avant la naissance des calculateurs, l'homme s'est inspiré de ses doigts pour représenter les nombres où les chiffres sont des multiples de 10 (décimal !) ; Mais pour arriver à traiter l'information par des calculateurs, l'information doit être représentée par la forme la plus simple possible, c'est juste avec des « 0 » et des « 1 » ! Ou simplement le binaire.

Dans ce chapitre, on verra comment représenter les nombres dans les différentes bases et le passage d'une base à une autre, on verra aussi comment faire des opérations arithmétiques simples (c'est la base des logiciels les plus complexes). On se limitera ensuite au binaire pour aborder la représentation des nombres signés et quelques formes de codage de l'information binaire.

I.1. Systèmes de numération

Passant par différentes représentations, le nombre est aujourd'hui représenté en forme décimale (base 10) où le chiffre est de 0 à 9, le chiffre (digit) le plus significatif se place à gauche.

➢ ***Exemple :*** $\mathbf{2046 = 6 + 40 + 2000 = 6.10^0 + 4.10^1 + 0.10^2 + 2.10^3}$

Le calculateur traite l'information (stockage, transmission, ..) codée en binaire (base 2) où les chiffres (bits) sont des « 0 » ou des « 1 », ce qui est traduit physiquement par un niveau haut de tension (1) ou niveau bas (0).

Pour simplifier la représentation des chaines binaires, dans les systèmes numériques, on utilise la représentation en base 8 (octal) dans laquelle un digit regroupe 3 bits, ou en base 16 (hexadécimal) où un digit regroupe 4 bits, pour ce dernier la valeur des chiffres A, B, C, D, E, F correspond, respectivement, à 10, 11, 12, 13, 14 et 15 en décimal.

D'une manière générale, un nombre N dans une base a peut s'écrire avec la représentation suivante :

$$N = \left(\overbrace{c_n c_{n-1} \dots c_1 c_0}^{(n+1)\ chiffres} \right)_a \qquad \text{Avec : } c_{i=0\dots n} < a$$

Le chiffre le plus significatif (Poids le plus fort) de rang 'n'

Le chiffre le moins significatif (Poids le plus faible) de rang '0'

I.2. Conversion d'une base à une autre

I.2.1. D'une base quelconque (a) vers le décimal (base 10)

Le nombre N, donné précédemment en base a, s'écrit en décimal comme suit :

$$N = (c_n c_{n-1} \dots c_1 c_0)_a = (c_0.a^0 + c_1.a^1 + \cdots + c_{n-1}.a^{n-1} + c_n.a^n)_{10}$$

➢ ***Exemples :***

1. Du binaire (base 2) au décimal

$$\overset{128\ 64\ 32\ 16\ 8\ 4\ 2\ 1}{(1\ 0\ 0\ 1\ 1\ 0\ 1\ 1)_{bin}} = 1.2^0 + 1.2^1 + 0.2^2 + 1.2^3 + 1.2^4 + 0.2^5 + 0.2^6 + 1.2^7$$

$$= 1 + 2 + 8 + 16 + 128 = (155)_{dec}$$

2. De l'octal (base 8) au décimal

$$1047_{oct} = 7.8^0 + 4.8^1 + 0.8^2 + 1.8^3 = 551_{dec}$$

3. De l'hexadécimal (base 16) au décimal

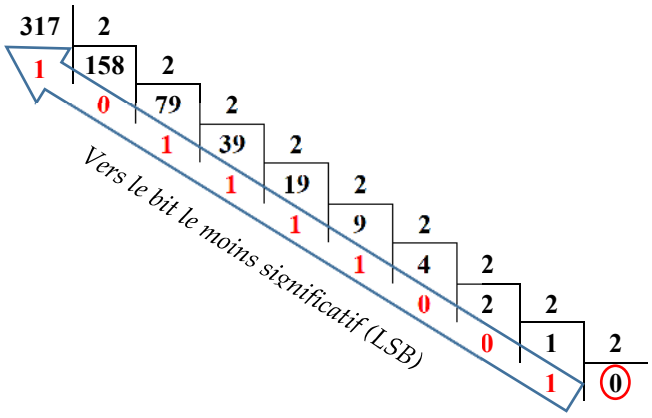
$$1DF_{hex} = 15.16^0 + 10.16^1 + 1.16^2 = 431_{10}$$

I.2.2. Du décimal vers une base quelconque (a)

On procède par une division successive du nombre en décimal sur la base a jusqu'à l'obtention du zéro, le résultat sera tiré du reste de la division.

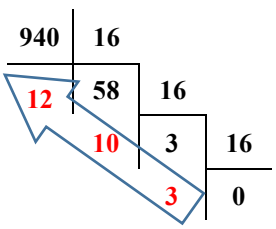
➢ ***Exemples :***

1. Du décimal au binaire ($317_{10} \rightarrow ?_b$)



$317_{10} = 100111101_b$

2. Du décimal à l'hexadécimal ($940_{10} \rightarrow ?_h$)



Donc : $940_{10} = 3AC_h$

Avec :

$$(10,11,12,13,14,15)_{dec} = (A,B,C,D,E,F)_h$$

I.2.3. D'une base a vers a^n ou l'inverse

Chaque groupe de nn chiffres de la base aa correspond à un seul chiffre dans la base aa^{nn}, le regroupement se fait du de droite à gauche pour la partie entière et de gauche à droite pour la partie fractionnelle (Rajouter des zéros si nécessaire).

➢ ***Exemples :***

1. *Du binaire à l'octal (base 8=2^3) :* $1010011101{,}11001_2 \rightarrow ?_8$

$$\underset{1}{001}\,\underset{2}{010}\,\underset{3}{011}\,\underset{5}{101}{,}\underset{6}{110}\,\underset{2}{010} = 1235{,}62_{oct}$$

2. *Du binaire à l'hexadécimal (base 16=2^4) :* $10110011101_2 \rightarrow ?_{16}$

$$\underset{5}{0101}\,\underset{9}{1001}\,\underset{13}{1101}_b = 59\mathrm{D}_h$$

3. *De l'hexadécimal (base 16=2^4) au binaire : $40\mathrm{DF}_h \rightarrow ?_b$*

$$40\mathrm{DF}_h = \underset{5}{0100}\,\underset{9}{0000}\,\underset{13}{1101}\,\underset{16}{1111}_b$$

I.2.4. D'une base quelconque à une autre

Il n'existe pas de méthode pour passer d'une base a_1 à une autre base a_2 directement. Dans ce cas il faut convertir le nombre de la base a_1 à la base 10, en suite convertir le résultat de la base 10 à la base a_2.

➢ ***Remarque :***

Pour le passage de l'octal à l'hexadécimal (ou l'inverse) on peut passer par le binaire.

➢ ***Exemple :***

$$6D_{16} = \overset{1}{0}\,\overset{5}{0}\,\overset{5}{1}\,1\,0\,1\,1\,0\,1_2 = 155_8$$
$$\qquad\qquad\quad 6 \qquad\quad D$$

I.2.5. Conversion des nombres fractionnaires

I.2.5.1. D'une base a vers la base 10

D'une manière générale on peut écrire :

$$\left(\overbrace{x_n x_{n-1}\cdots x_1 x_0}^{(n+1)digits}, \overbrace{x_{-1}x_{-2}\cdots x_{-m}}^{m\ digits}\right)_a = \left(\sum_{i=-m}^{n} x_i . a^i\right)_{10}$$

➢ ***Exemples :***

1. De l'octal au décimal :

$$27{,}04_8 = 7.8^0 + 2.8^1 + 0.8^{-1} + 4.8^{-2} = 7 + 16 + 0 + 0{,}0625 = 23{,}0625_{10}$$

2. Du binaire au décimal :

$$101{,}011_2 = 1.2^0 + 0.2^1 + 1.2^2 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3} = 1 + 4 + 0{,}25 + 0.125 = 5{,}375_{10}$$

I.2.5.2. Du décimal vers une base a:

- La conversion de la partie entière se fait par la division successive sur la base (voir précédemment)
- Pour la partie fractionnelle on effectue une multiplication par la base a (décalage successif à gauche) et à chaque fois on soustrait la partie entière et on continue le décalage.

➢ ***Exemple :*** *Du décimal au binaire* $(5{,}3125_{10} \rightarrow ?)$

1. $5_{10} = 101_b$
2. $0{,}3125_{10} \rightarrow ?_b$

0,3125 × 2 = 0 , 625
0, 625 × 2 = 1 , 25
0, 25 × 2 = 0 , 5
0, 5 × 2 = 1 , **0**

$0{,}3125_{10} = 0{,}0101_b$

Donc : $5{,}3125_{10} \rightarrow 101{,}0101_b$

I.3. Les opérations arithmétiques sur les nombres non signés

Les opérations arithmétiques se font de la même manière qu'en décimal.

➢ ***Exemples :***

1. Opérations arithmétiques en binaire :

$$\begin{array}{r} 1 \\ + \quad 1 \\ \hline 1\,0 \end{array}$$

$10_2 = 2_{10}$

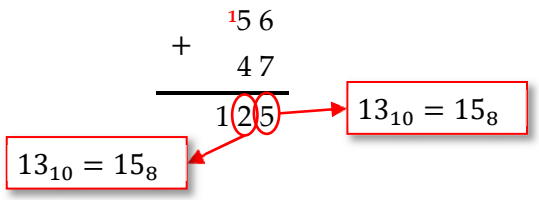
$$\begin{array}{r} 1\,{}_1\,0\,1 \\ - \quad {}_1\,1\,1 \\ \hline 0\,1\,0 \end{array}$$

$10_2 = 2_{10}$

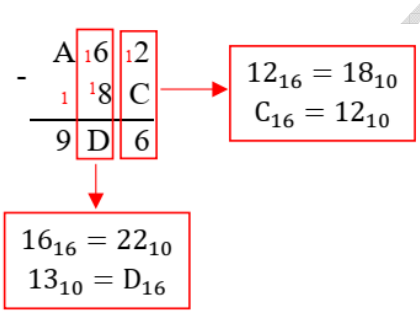
$$\begin{array}{r} 101 \\ \times \quad 11 \\ \hline 1\,0\,1 \\ 1\,0\,1\,. \\ \hline 1\,1\,1\,1 \end{array}$$

1101	10
10	110,1
101	
10	
1	
0	
10	
10	
0	

2. Opérations arithmétiques en octal :



3. Opérations arithmétiques en hexadécimal :



I.4. Représentation des nombres entiers signés

❖ La valeur maximale en décimal d'un nombre binaire non signé (positif) sur n bits est :

$$\overbrace{11\ldots111}^{n\ bits} = \sum_{i=0}^{n-1} 2^i = (2^n - 1)_{10}$$

Donc pour un nombre non-signé N sur n bits ⇒ $N \in [0, (2^n - 1)]$

➢ ***Exemple :***

Sur 8 bits, la valeur maximale : $\overbrace{1111\,1111}^{8\ bits} = 2^8 - 1 = 255_{10}$

Donc on peut représenter un nombre décimal (non-signé) ∈ [0,255]

❖ Pour les nombres signés, le dernier bit (à gauche) nous informe sur le signe (+ ou -) donc sur n bits on peut représenter un nombre maximal $(2^{n-1} - 1)$.

➢ ***Remarques importantes :***

- ✓ Avant de représenter un nombre signé il faut connaitre le nombre de bits (il est représenté sur un nombre de bits fixe, parfois il faut compléter par des zéros).
- ✓ Le chiffre de poids fort d'un nombre positif est le zéro (0).
- ✓ Le chiffre de poids fort d'un nombre négatif est le un (1).

On donnera, dans ce qui suit, trois formes de représentation des nombres signés : Représentation signe-magnitude SM, représentation en complément à un CA1 et la représentation en CA2.

I.4.1. Représentation signe-magnitude SM

Dans la représentation SM (signe + valeur absolue), le bit de poids fort (le plus à gauche) représente le signe (0 pour un nombre positif et 1 s'il est négatif) et le reste la valeur absolue du nombre.

➢ ***Exemple :***

Représenter +6 et -6 sur 8 bits :

$$6 = 110 \Rightarrow sur\ 8\ bits \begin{cases} +6 = 0000\ 0110 \\ -6 = 1000\ 0110 \end{cases}$$

➢ ***Remarque :***

$$Par\ exemple, sur\ 4\ bits \longrightarrow \begin{cases} +0 = 0000 \\ -0 = 1000 \end{cases}$$

Donc, deux représentations possibles pour le nombre zéro!

I.4.2. Représentation complément à 1 « CA1 »

Il est appelé aussi complément logique ou complément restreint CR. Le signe (-) est obtenu en appliquant le complément logique (ce qui reste pour devenir complet) :

$$-N = CA1(N) = \bar{N} = max - N$$

max **:** Le digit max de la base (en décimal c'est 9, en binaire c'est 1, …)

⇒ En binaire : $\bar{1} = 0$ et $\bar{0} = 1$

➢ ***Exemples :***

1. Déterminer, sur 8 bits, la représentation de +45 et − 45 en CA1.

➢ $+45_{10}$ est un nombre signé positif sa représentation en binaire signé est la même qu'on binaire non-signé mais il faut rajouter des « 0 » pour atteindre 8 bits.

$45 = 101101 \Rightarrow sur\ 8\ bits : +45 = 0010\ 1101$

➢ Déterminer la représentation du nombre (-45_{10}) revient à faire le CA1 du nombre positif.

$$-45 = \text{CA1}(+45) = \text{CA1}(0010\ 1101) = \overline{0010\ 1101} = 1101\ 0010$$

On peut remarquer que le bit du poids fort est un 0 pour +45 et 1 pour −45.

2. Représenter +0 et -0 en CA1 sur 4 bits.

$$\left.\begin{matrix} +0 = 0000 \\ -0 = \overline{0000} = 1111 \end{matrix}\right\} \longrightarrow$$ +0 et -0 n'ont pas la même représentation !

➢ ***Remarques :***

1. En binaire, le CA1 est obtenu en inversant les zéros et les uns.
2. En représentation SM ou CA1, le nombre zéro (0) possède deux représentation (−0 et +0) par conséquence l'intervalle d'un nombre signé N représenté sur n bits (par symétrie) est donné par :

$$N \in [-(2^{n-1} - 1), (2^{n-1} - 1)] \quad \text{Ex. Sur 8 bits (n=8) : } N \in [-127, +127]$$

3. Puisque $-(-N) = N$ donc : $\mathrm{CA1}(\mathrm{CA1(N)}) = \overline{\overline{\mathrm{N}}} = \mathrm{N}$
4. Le bit du poids fort informe sur le signe du nombre.

I.4.3. Représentation complément à 2 « CA2 »

Appelé aussi complément vrai CV ou complément arithmétique. Le signe (-) est obtenu en appliquant le CA2 qui est le CA1 et en rajoutant 1 :

$$-N = CA2(N) = CA1(N) + 1 = \bar{N} + 1$$

➢ ***Exemples :***

3. Déterminer la représentation de +45 et −45 en CA2 sur 8 bits.

➢ $+45_{10}$ est un nombre signé positif, il est représenté de la même forme que SM et CA1 :

$$+45 = 0010\ 1101$$

➢ $-45 = \mathrm{CA2}(0010\ 1101) = \mathrm{CA2}(0010\ 1101) + 1 = 1101\ 0010 + 1 = 1101\ 0011$

Nombre négatif

✓ Vérifier que : $\mathrm{CA2}(11010011) = 0010\ 1101$, cela revient à vérifier que : $-(-45) = +45$

4. Représenter +0 et -0 en CA2 sur 8 bits.

9ème bit éliminé (voir plus loin)

$$-0 = CA2(0000\ 0000) = \overline{0000\ 0000} + 1 = 1111\ 1111 + 1 = 1\ \overbrace{0000\ 0000}^{8\ bits} = +0$$

⇒ En CA2, une représentation unique pour le nombre zéro.

➢ ***Remarque :***

Avec la représentation en CA2 , le nombre zéro possède une seule représentation ($-0 = +0 \Rightarrow$ occupent le même espace mémoire) par conséquence pour un nombre N représenté sur n bits :

$N \in [-2^{n-1}, (2^{n-1} - 1)]$ Ex. Sur 8 bits (n=8) : $N \in [-128, +127]$

I.4.4. Les opérations arithmétiques sur les nombres signés

➢ Les nombres signés se représentent sur un nombre donné de bits, donc il faut connaitre le nombre de bits de représentation (rajouter des '0' à gauche pour atteindre le nombre de bits prédéfini) ;

➢ une soustraction est transformée en addition, en effet : $A - B = A + (-B)$ et pour obtenir $(-B)$ on applique la représentation demandée, par exemple : En CA2 : $-B = CA2(B)$;

➢ le résultat aura la même représentation (La somme de 2 nombres en CA2 donne un résultat en CA2) ;

➢ se rappeler que le bit de poids fort est 0 pour les nombres positifs et 1 pour les nombres négatifs.

I.4.4.1. Les opérations arithmétiques en CA1

Pour que le résultat soit juste il faut rajouter l'éventuelle retenue supplémentaire au résultat de l'addition. Par exemple sur 8 bits, la retenue supplémentaire est le 9ème bit.

➢ ***Exemples :*** Faire en CA1 sur 4 bits les opérations suivantes : **2−3** et **−2 − 3**

❖ $2-3=(+2)+(-3)\Rightarrow$ Représenter (+2) et (-3) en CA1 sur 4 bits puis faire l'addition :

$+2=0010_{CA1}$

$+3=0011\Rightarrow -3=CA1(3)=\overline{0011}=1100_{CA1}$

2	+	$0\,0\,1\,0_{CA1}$
-3		$1\,1\,0\,0_{CA1}$
$-N$		$1\,1\,1\,0_{CA1}$

Le résultat est négatif, pour le transformer en décimal il faut le rendre positif donc lui appliquer le CA1 :

$$N=CA1(-N)=\overline{1110}=0001=1_{10}\Rightarrow 1110_{CA1}=-1_{10}$$

❖ $-2-3=(-2)+(-3)$

$+2=0010\Rightarrow -2=CA1(2)=\overline{0010}=1101_{CA1}$

- 2	+	1 1 0 1
- 3		1 1 0 0
		1 (1 0 0 1) (4 bits)
		+1
$-N$		(1 0 1 0) (4 bits)

Retenue supplémentaire

Bit du signe

➢ Dans ce cas, on remarque l'existence d'une retenue supplémentaire (5ème bit) qui doit être rajoutée au résultat !

Vérifier le résultat qui est négatif : $N=CA1(-N)=\overline{1010}=0101=5_{10}\Rightarrow 1010_{CA1}=-5_{10}$

I.4.4.2. Les opérations arithmétiques en CA2

L'addition se fait par retranchement (ignorer) de la retenue supplémentaire sur une longueur de bits prédéfinie.

➢ ***Exemples :*** Faire en CA2 sur 4 bits les opérations précédentes.

❖ $2-3=(+2)+(-3)\Rightarrow$ Représenter (+2) et (-3) en CA2 sur 4 bits puis faire l'addition :

$+2=0010_{CA2}$

$+3=0011\Rightarrow -3=CA2(3)=\overline{0011}+1=1101_{CA2}$

2	+	0 0 1 0
-3		1 1 0 1
$-N$		1 1 1 1

Le résultat est négatif, pour le transformer en décimal il faut le rendre positif donc lui appliquer le CA2 :

$$N=CA2(-N)=\overline{1111}+1=0001=1_{10}\Rightarrow 1111_{CA2}=-1_{10}$$

❖ $-2-3=(-2)+(-3)$

$+2=0010 \Rightarrow -2=CA2(2)=\overline{0010}+1=1101+1=1110_{CA2}$

-2	+	1 1 1 0
-3		1 1 0 1
$-N$		$\overbrace{1\,\mathbf{1}\,0\,1\,1}^{4\ bits}$

5ème bit, Retenue supplémentaire

Bit du signe

➢ Dans ce cas, la retenue supplémentaire (5ème bit) est tout simplement éliminée !

Le résultat est donc négatif : $-N=1011$

$$\Rightarrow N=CA2(-N)=\overline{1011}+1=0100+1=0101=5_{10} \Rightarrow 1011_{CA2}=-5_{10}$$

I.4.4.3. Notion du débordement :

Le débordement (dépassement de capacité) se produit lorsque le résultat ne peut être représenté sur le nombre de bits prédéfini, dans ce cas on obtient un nombre positif à la somme de deux nombres négatifs ou un nombre négatif à la somme de deux nombres positifs, et le résultat par conséquence est FAUX.

➢ Donc, Suivant le nombre de bits prédéfini et la représentation utilisée, on doit déterminer les limites de l'intervalle qui doit contenir le résultat, pour un nombre signé N sur n bits :

$$SM\ ou\ CA1 \Longrightarrow N \in [-(2^{n-1}-1)\,,+(2^{n-1}-1)]$$

$$CA2 \Longrightarrow N \in [-2^{n-1}\,,+(2^{n-1}-1)]$$

$$Par\ exemple, sur\ 8\ bits \Rightarrow \begin{cases} \text{En CA1 ou SM : Le résultat} \in [-127,+127] \\ \text{En CA2 : Le résultat} \in [-128,+127] \end{cases}$$

➢ ***Exemples :***

❖ *Faire sur 4 bits en CA1 :* $(5+6)\ et\ (-5-3) \Longrightarrow le\ résultat \in [-7,+7]$

5	+	0 1 0 1	*Positif*
6		0 1 1 0	*Positif*
11		1 0 1 1	*Négatif*

Débordement
$11 \notin [-7,+7]$

-5	+	1 0 1 0	*Négatif*
-3		1 1 0 0	*Négatif*
		(1) 0 1 1 0	
		+1	
-8		0 1 1 1	*Positif*

Débordement
$-8 \notin [-7,+7]$

❖ *Faire sur 4 bits en CA2 :* $(-5-3) \Longrightarrow le\ résultat \in [-8,+7]$

- 5	+	1 0 1 1	*Négatif*
- 3		1 1 0 1	*Négatif*
- 8	~~1~~	1 0 0 0	*Négatif*

Pas de débordement
$-8 \in [-8,+7]$

I.5. Codage

C'est la représentation des éléments (l'information) suivant des règles bien définies, on distingue de types : Codage pondéré et codage non-pondéré.

I.5.1. Codage pondéré

Dans ce codage la position du bit dans la chaine lui donne un poids (une valeur).

I.5.1.1. Codes binaire, octal, hexadecimal

Chaque chiffre (x_i) dans une base (a) reçoit un poids proportionnel à sa position (i) :

$$(x_i)_a = x_i . a^i{}_{10}$$

I.5.1.2. Code BCD (Binary Coded Decimal)

C'est une suite binaire où chaque groupe de 4 bits représente un chiffre en base 10. Les chiffres en décimal ∈ [0,9], donc chaque groupe de bits ∈ [0000,1001]

➢ ***Exemple :***

$$1902_{10} = (0001\ 1001\ 0000\ 0010)_{BCD}$$
$$1 \quad 9 \quad 0 \quad 2$$

Les opérations arithmétiques en BCD

✓ Si l'addition de deux chiffres donne un chiffre > 9 alors il faut lui rajouter 6 (0110) en commençant par le chiffre le plus faible jusqu'à l'obtention d'un nombre en BCD (*en effet :* $-10_{10} = CA2(1010) = 0101$, *donc rajouter 0110 revient à soustraire 10 tout en rajoutant 1 au chiffre plus fort*).

✓ Dans le cas de la soustraction on soustrait 6 au chiffre résultant.

➢ ***Exemples :***

77	-	0 1 1 ^{1}1 ^{1}0 1 1 1
- 18		0 0 $_{1}$0 $_{1}$1 1 0 0 0
		0 1 0 1 1 1 1 1
	-	0 1 1 0
59		0 1 0 1 1 0 0 1

15 > 9

Soustraire 6

19	+	0 0 0 1 1 0 0 1
+ 85		1 0 0 0 0 1 0 1
		1 0 0 1 1 1 1 0
	+	0 1 1 0
		1 0 1 0 0 1 0 0
	+	0 1 1 0
104		1 0 0 0 0 0 1 0 0

14 > 9

10 > 9

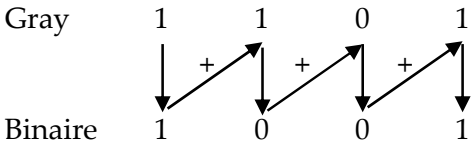
Rajouter 6

I.5.2. Codage non pondéré

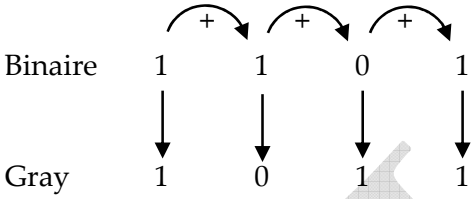
I.5.2.1. Code Gray (binaire réfléchi)

Le passage d'un nombre au suivant, un seul bit qui change. On dit que deux nombres qui se succèdent en gray sont adjacents (représentation miroir !).

Gray→Binaire



Binire→Gray



Déc.	Binaire				Gray			
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

I.5.2.2. Code détecteur d'erreur (bit de parité)

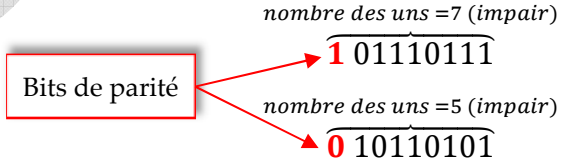
Dans une transmission de données on rajoute un bit de parité d'une façon à avoir la somme des bits en 1 (bit de parité inclus) est paire pour parité 'pair' et impaire pour parité 'impair'.

Remarque :

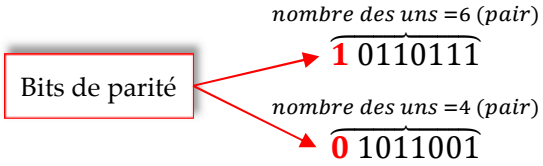
Le code de parité détecte le changement d'un bit sans autant corriger l'erreur.

- ***Exemples :***

1. Parité impaire :



2. Parité paire :



I.5.2.3. Code détecteur d'erreur (code Hamming)

Avec le code Hamming, on arrive à détecter l'erreur et la position du bit erroné ! (Donc pouvoir corriger l'erreur). Pour transmettre un message de n bits ($m_n \dots m_3 m_2 m_1$) avec le code Hamming on doit insérer p bits de contrôle de parité ($c_p \dots c_3 c_2 c_1$), donc pour une information sur n bits on transmit une suite de (n+p) bits, avec : $2^p - p \geq n + 1$

Par exemple, pour une information sur 4 bits (n=4) on doit insérer 3 bits de contrôle au minimum :

$$p = 3 \Longrightarrow (2^3 - 3 = 5) \geq (4 + 1 = 5)$$

Le tableau suivant résume le minimal des bits de contrôle p suivant le nombre des bits d'information n :

n	1	2	3	4	5	6	7	8	9	10	11	12	13	…
p	2	3	3	3	4	4	4	4	4	4	4	5	5	…

1. ***Placer les bits de contrôle de parité***

Les positions des bits dans la chaine codée « mot Hamming » (message + bits de contrôle) sont indicées par des nombres entiers commençant par 1 (1,2,3,4,…). Les indices des bits de contrôle insérés sont les puissances de 2 (2^0,2^1,2^2,…), donc : 1,2,4,8,…

➢ ***Exemple :***

Supposant que l'on souhaite envoyer un message de n=4 bits ($m_4 m_3 m_2 m_1$), on a vu qu'il faut p=3 bits de contrôle ($c_3 c_2 c_1$) -voir table-.

La chaine à envoyer est donc :

Positions des bits 7 6 5 4 3 2 1

$m_4\ m_3\ m_2\ c_3\ m_1\ c_2\ c_1$

2. ***Calculer les bits de contrôle***

La valeur de la suite $c_3 c_2 c_1$ donne la position de l'erreur, elle peut être de 1 à 7 (de 1 à n+p).

Si : $c_3 c_2 c_1 = 0 \Longrightarrow pas\ d'erreur$

Par ex. si l'erreur dans la position 5 (dans m_2) alors on trouvera : $c_3 c_2 c_1 = 101 = 5_{10}$

- ***Comment calculer $c_3 c_2 c_1$?***

➢ ***Exemple :***

Supposons que notre information est : $m_4 m_3 m_2 m_1 = 1010$, le mot Hamming à envoyer est :

$$m_4\ m_3\ m_2\ c_3\ m_1\ c_2\ c_1 = \overset{7\ 6\ 5\ 4\ 3\ 2\ 1}{1\ 0\ 1\ c_3\ 0\ c_2\ c_1}$$

Pour trouver les bits de contrôle on récupère les positions des **uns « 1 »**, dans notre cas : 7 et 5.

En parité paire, pour déterminer les bits de contrôle on additionne bit à bit (sans retenue) les positions des bits à '1' (pour avoir un nombre pair de « 1 »):

7	1 1 1
5	1 0 1
	0 1 0
	$c_3c_2c_1$

Donc $c_3c_2c_1 = 010$ et la chaine à envoyer est : 1 0 1 0 0 1 0

3. *Détection de l'erreur :*

Supposant que le récepteur reçoit le mot suivant : $\overset{7}{1}\ 0\ \overset{5}{1}\ 0\ \overset{3}{1}\ \overset{2}{1}\ 0$ (l'erreur au niveau du 3ème bit), on refait le même travail pour détecter l'erreur.

- Additionner sans retenue les positions des 1 dans la chaine reçue (dans notre cas : 2, 3, 5 et 7) et le résultat est simplement la position de l'erreur !

2	0 1 0
3	0 1 1
7	1 1 1
5	1 0 1
	0 1 1

Donc la position de l'erreur est $011 = 3$

⇒ Il suffit d'inverser le bit de la position 3 pour corriger le message reçu.

Si le message ne contient pas d'erreur on additionne les positions 2, 5 et 7

Ce qui donne : 000

2	0 1 0
7	1 1 1
5	1 0 1
	0 0 0

I.5.2.4. Le code ASCII

Dans la mémoire de l'ordinateur les caractères sont stockés suivant un code dit : ASCII (American Standard Code for Information Interchange). Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127) puis il a été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (caractères spécifiques à une langue autre que l'anglais), chaque touche ou une combinaison de touches du clavier correspond à un code ASCII.

TD n° 1 : *Systèmes de numération et codage*

Exercice 1 :

Effectuer les opérations suivantes puis vérifier le résultat en décimal (transformer) :

- $(1101,111)_2 + (11,1)_2 = (?)_2$
- $101001_b - 10011_b = (?)_b$
- $9B1_{16} + 257_8 = (?)_{16}$
- $(1542)_8 - (267)_8 = (?)_8$

Exercice 2 :

Effectuer les opérations suivantes en CA1 puis en CA2 sur 8 bits. Vérifier les résultats et indiquer les éventuels débordements. Comment peut-on détecter si le résultat est faux ?

115 − 26 115 + 35 26 − 35 − 38 – 96

Exercice 3 :

On reçoit par une ligne de transmission les suites binaires suivantes, sachant qu'elles sont transmises avec un bit de parité « paire» :

1001101001

1001011110

1010110111

Vérifier la possibilité d'erreur sur les données reçues.

Exercice 4 :

Soit un mot de Hamming de longueur 15 : 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1

1. Quels sont les bits de contrôle de parité ?
2. Quel est le message reçu ?
3. Est-ce que le message reçu correspond au message transmis ?
4. Si non, quel a été le message transmis ?

Exercices supplémentaires

Exercice 1:

On veut envoyer le mot 1011 avec un code Hamming, quels bits, doit-on lui adjoindre et quelle est la séquence à transmettre ?

Exercice 2:

Y a-t-il une erreur dans le mot Hamming suivant ?

1101101

Exercice 3:

Effectuer les opérations suivantes en BCD:

- ✓ $19_{BCD} + 85_{BCD} = ?$
- ✓ $77_{BCD} - 18_{BCD} = ?$

Exercice 4:

Faire les opérations suivantes sur 4 bits en complément à 2 (vérifier le résultat !) :

$6 - 2 = ?$ $2 + 6 = ?$

Exercice 5:

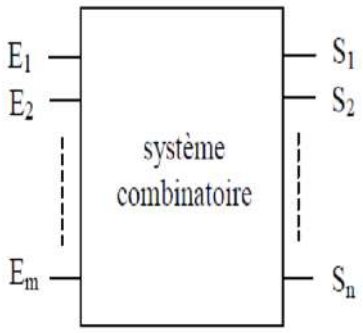
On veut transmettre la suite (1101) avec le code Hamming.

- ➢ Donner le nombre et l'emplacement des bits de contrôle à rajouter ;
- ➢ déterminer les bits de contrôle et la séquence transmise.

Chapitre II. Algèbre de Boole et Simplification des fonctions logiques

Introduction

Un système combinatoire comporte des entrées ($E_1, E_2, \dots, E_m$) et des sorties ($S_1, S_2, \dots, S_n$) logiques (0 ou 1), dans lequel chaque sortie Si (avec : $i = 1, \dots, n$) est exprimée en fonction des entrées par une fonction logique f_i, on peut écrire :



$$S_i = f_i(E_1, E_2, \dots, E_m) \qquad \text{Avec : } i = 1, \dots, n$$

Dans ce chapitre on abordera le système combinatoire donc les fonctions combinatoires f_i (représentation et simplification). Les lois applicables aux variables booléennes (logiques) sont définies par une algèbre dite de Boole.

II.1. Algèbre de Boole

Georges BOOLE, philosophe et mathématicien irlandais du 19ème siècle est l'auteur d'une théorie sur l'art de construire un raisonnement logique au moyen de propositions qui ont une seule réponse OUI (VRAI) ou NON (FAUX). L'ensemble des opérations formelles appliquées à ces propositions forme une structure mathématique appelée algèbre de Boole. A son époque, il s'agissait de développement purement théorique car on ignorait l'importance qu'allait prendre cette algèbre avec l'informatique.

Les concepts de la logique booléenne ont été ensuite appliqués aux circuits électroniques par Claude Shannon (1916-2001). Cette algèbre est applicable à l'étude des systèmes possédant deux états s'excluant mutuellement. Dans la logique positive (la plus couramment utilisée), on associe OUI à 1 et NON à 0. Dans la logique négative, c'est l'inverse.

L'algèbre booléenne binaire est à la base de la mise en œuvre de tous les systèmes numériques : ordinateurs, systèmes numériques portables, systèmes de communication numériques, etc. Elle permet entre autres de simplifier les fonctions logiques, et donc les circuits électroniques associés.

II.1.1. Fonctions logiques de base

Une fonction logique est une fonction d'une ou plusieurs variables logiques, combinées entre elles par 3 fonctions élémentaires simples : NON (NOT), OU (OR) et ET (AND). Il existe également des fonctions élémentaires composées de fonctions élémentaires simples : NON-ET (NAND), NON-OU (NOR), OU-EXCLUSIF (XOR), NON-OU-EXCLUSIF (XAND).

Elles peuvent être représentées schématiquement par des opérateurs logiques, encore appelés portes (Gates) logiques.

II.1.1.1. Fonctions élémentaires simples : NOT, OR, AND

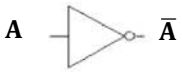
- ***Fonction NOT (ou fonction complément)***

Soit A une variable logique, la fonction complément de la variable A est notée :

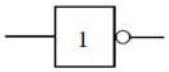
$$\boldsymbol{NOT(A) = \overline{A}}$$ (On peut prononcer "A barre").

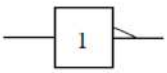
Cette fonction affecte à la variable de sortie l'état complémentaire de la variable d'entrée.

Symboles **:** Le symbole, appelé porte logique (GATE), de la fonction NOT est le suivant :



Le symbole américain





Les symboles européens

Table de vérité :

On peut exprimer cette propriété sous forme d'un tableau entrée/sortie, appelé table de vérité :

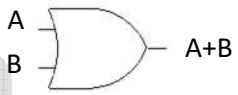
A	$\overline{A}$
0	1
1	0

- ***Fonction OR (ou somme logique)***

La fonction logique OU est également appelée "somme logique" est notée :

$$\boldsymbol{A\ (OR)\ B\ = A + B}$$

Symboles



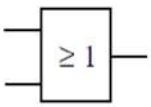
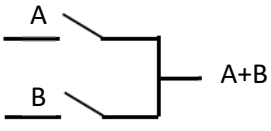


Table de vérité :

Avec la porte OR à plus de 2 entrées, le seul cas où la sortie serait à 0 serait le cas où toutes les entrées sont à 0.



A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Propriétés

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

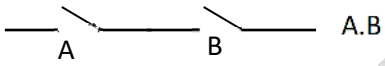
➢ ***Fonction AND (ou produit logique)***

La fonction logique ET est notée : $\boldsymbol{A\ (AND) B\ = A.B}$

Symboles

Table de vérité

Avec la porte AND à plus de 2 entrées, le seul cas où la sortie serait à 1 serait le cas où toutes les entrées sont à 1.



A	B	A. B
0	0	0
0	1	0
1	0	0
1	1	1

Propriétés

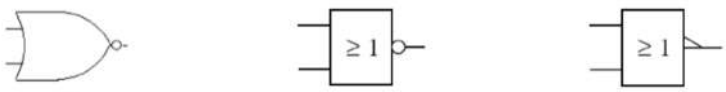
$\boldsymbol{A.A = A}$ $\boldsymbol{A.\overline{A} = 0}$ $\boldsymbol{A.0 = 0}$ $\boldsymbol{A.1 = A}$

II.1.1.2. Fonctions élémentaires composées :

Les fonctions élémentaires composées (ou combinées) sont obtenues en combinant entre elles les fonctions élémentaires simples, l'ensemble des fonctions élémentaires définissent un ensemble complet d'opérateurs.

➢ ***Fonction NON-OU (NOR)***

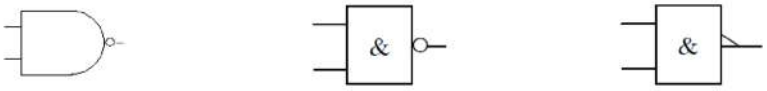
Notée : $\boldsymbol{A\ (NOR)\ B\ = \overline{A + B}}$



A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

➢ ***Fonction NON-ET (NAND)***

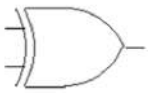
Notée : $\boldsymbol{A\ (NAND)\ B\ = \overline{A.B}}$

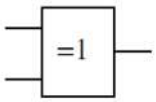


A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

➢ ***Fonction OU EXCLUSIF (XOR)***

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





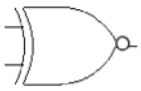
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

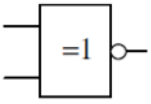
✓ Avec 2 variables, on peut considérer le XOR comme un détecteur d'inégalité, puisque sa sortie est à 1 quand ses 2 entrées sont différentes.

✓ Pour un nombre supérieur de variables, la fonction XOR vaut 1 quand les variables d'entrée à 1 sont en nombre impair. Il s'agit donc également d'un détecteur de parité « impaire ».

➢ ***Fonction NON-OU EXCLUSIF (XNOR)***

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





A	B	A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

✓ Avec 2 variables, on peut considérer le XNOR comme un détecteur d'égalité, puisque sa sortie est à 1 quand ses 2 entrées sont égales.

✓ Pour un nombre supérieur de variables, la fonction XNOR vaut 1 quand les variables d'entrée à 1 sont en nombre pair. Il s'agit donc également d'un détecteur de parité « paire».

II.1.2. Propriétés et théorèmes

➢ ***Principe de dualité :***

Toute expression logique vraie demeure vraie si l'on remplace les OR (+) par des AND (.), les 0 par des 1 et les 1 par des 0

Soient A, B et C des variables logiques, on peut vérifier ce qui suit :

Commutativité	$A + B = B + A$	$A.B = B.A$
Associativité	$A + (B + C) = (A + B) + C$	$A.(B.C) = (A.B).C$
Distributivité	$A + (B.C) = (A + B).(A + C)$	$A.(B + C) = (A.B) + (A.C)$
Elément neutre	$A + 0 = A$	$A.1 = A$

Elément absorbant	$\boldsymbol{A+1=1}$	$\boldsymbol{A.0=0}$	
Redondance	$\boldsymbol{A+A+\cdots+A=A}$	$\boldsymbol{A.A.\ldots.A=A}$	
Propriétés du complément	$\boldsymbol{\overline{\overline{A}}=A}$	$\boldsymbol{A+\overline{A}=1}$	$\boldsymbol{A.\overline{A}=0}$
Théorème de De Morgan	$\boldsymbol{\overline{A+B}=\overline{A}.\overline{B}}$	$\boldsymbol{\overline{A.B}=\overline{A}+\overline{B}}$	
Absorption	$\boldsymbol{A+(A.B)=A}$	$\boldsymbol{A.(A+B)=A}$	
Absorption du complément	$\boldsymbol{A+(\overline{A}.B)=A+B}$ $\boldsymbol{\overline{A}+(A.B)=\overline{A}+B}$	$\boldsymbol{A.(\overline{A}+B)=A.B}$ $\boldsymbol{\overline{A}.(A+B)=\overline{A}.B}$	
Théorème des consensus	$\boldsymbol{AB+\overline{A}C+BC=AB+\overline{A}C}$	$\boldsymbol{(A+B).(\overline{A}+C).(B+C)=(A+B).(\overline{A}+C)}$	

- ***Remarque :***

Du théorème de distributivité on parle aussi de factorisation, en effet :

$$\boldsymbol{(A+B).(A+C)=A+(B.C)}$$

- ***Exercice 1 :*** Démontrer le théorème de consensus

Pour ce faire on peut utiliser la table de vérité (voir plus loin) comme on peut le faire par l'algèbre de Boole :

$AB+\bar{A}C+BC=AB+\bar{A}C+BC.1$ ..... *Elément neutre*

$=AB+\bar{A}C+BC.(A+\bar{A})$ ..... *Propriétés du complément*

$=AB+\bar{A}C+BCA+BC\bar{A}$ ..... *Distributivité du AND sur le OR*

$=AB+\bar{A}C+ABC+\bar{A}CB$ ..... *Commutativité du AND*

$=AB+ABC+\bar{A}C+\bar{A}CB$ ..... *Commutativité du OR*

$=AB(1+C)+\bar{A}C(1+B)$ ..... *Factorisation*

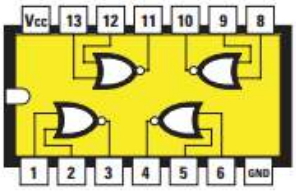
$=AB+\bar{A}C$ ..... *Elément absorbant et élément neutre*

- ***Exercice 2 :*** Démontrer la 2ème relation du théorème de consensus.

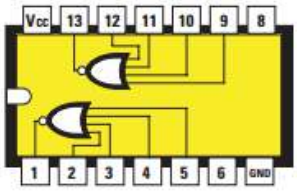
II.2. Circuits logiques

On a vu que chaque fonction logique élémentaire pouvait être représentée par une porte logique. C'est la représentation qui permet de passer à la réalisation pratique des fonctions logiques, c'est à dire avec des circuits intégrés. Tout circuit logique (qui représente une fonction logique quelconque) peut être réalisé à partir des opérateurs logiques élémentaires simples (NOT, OR, AND) et des opérateurs logiques élémentaires composés (NOR, NAND, XOR, XNOR), ces portes existent à deux ou à plusieurs entrées.

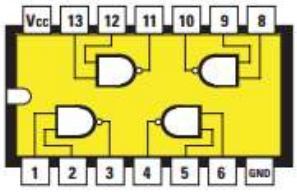
➢ ***Exemples :***



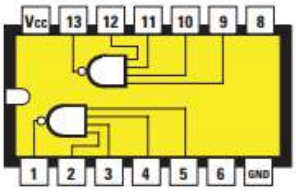
4001



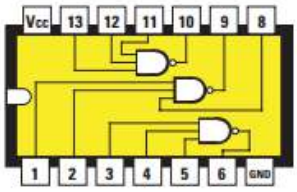
4002



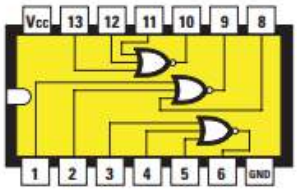
4011



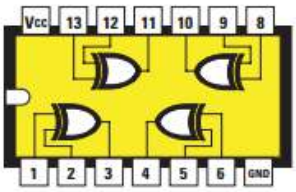
4012



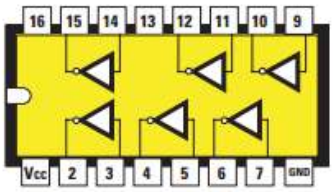
4023



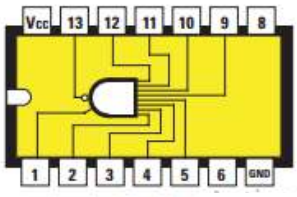
4025



4030



4049



4068

II.3. Portes universelles

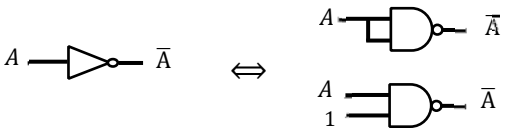
Les portes universelles permettent d'obtenir les 3 fonctions logiques de base (NOT, OR, AND), donc de représenter n'importe quelle fonction logique avec un seul type de porte, ceci est utile si l'on veut réduire le nombre de circuits intégrés à utiliser.

II.3.1. Porte universelle NAND

Les portes (fonctions) élémentaires simples NOT, OR et AND peuvent être représentées par des portes NAND.

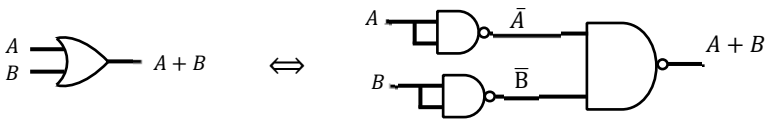
<u>NOT en fonction de NAND</u>

$$\bar{A} = \overline{A.A} = \overline{1.A}$$



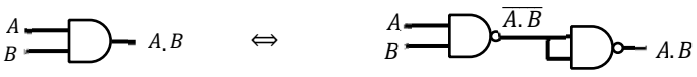
<u>OR En fonction de NAND</u>

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



<u>AND en fonction de NAND</u>

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

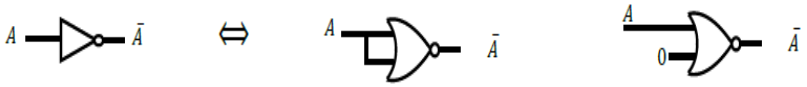


II.3.2. Porte universelle NOR

On peut aussi représenter toutes les portes élémentaires par des portes NOR.

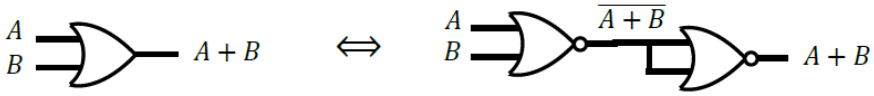
NOT en fonction de NOR

$$\bar{A} = \overline{A + A} = \overline{0 + A}$$



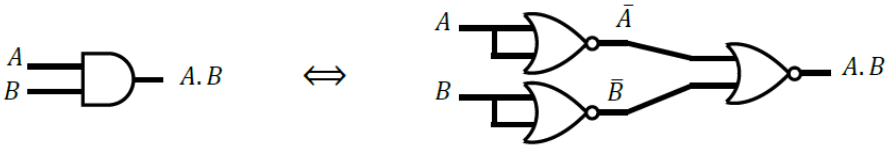
OR En fonction de NOR

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



AND en fonction de NOR

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



➢ ***Remarque :***

L'inconvénient pratique que comporte cette solution est que physiquement, une porte logique possède un temps de propagation, c'est à dire que lorsque des entrées lui sont appliquées, il existe un petit délai avant que la sortie correspondante soit disponible. Donc, si l'on utilise plusieurs portes en cascade, il faut ajouter les temps de propagation de toutes les portes.

➢ ***Exemples :***

1. Transformer la fonction suivante pour pouvoir la réaliser qu'avec des portes NAND :

$$F_1 = A\bar{B}C + \bar{A}BC + AB\bar{C} = \overline{\overline{A\bar{B}C + \bar{A}BC + AB\bar{C}}} = \overline{\overline{A\bar{B}C}.\overline{\bar{A}BC}.\overline{AB\bar{C}}}$$

2. Transformer la fonction suivante pour pouvoir la réaliser qu'avec des portes NOR à deux entrées :

$$F_2 = (A + BC)(D + E) = \overline{\overline{(A + BC)(D + E)}} = \overline{\overline{(A + BC)} + \overline{(D + E)}} = \overline{\overline{(A + \overline{\overline{BC}})} + \overline{(D + E)}}$$

$$= \overline{\overline{(A + \bar{B} + \bar{C})} + \overline{(D + E)}}$$

II.4. Représentation des fonctions logiques

Une fonction logique peut être représentée par trois manières : par table de vérité, expression algébrique ou schéma à portes logiques (logigramme).

II.4.1. Représentation par table de vérité

Une fonction logique peut être décrite par une table qui donne les valeurs de la fonction (0 ou 1) pour toutes les combinaisons possible des entrées, pour n entrées 2^n combinaisons possibles donc 2^n valeurs possibles de la fonction.

- ***Remarque***

Selon l'application, il peut y avoir des combinaisons d'entrée non-utilisées (fonction non définie).

- ***Exemple :***

La fonction de détection de parité « paire ».

	A	B	C	$F(A,B,C)$
0	0	0	0	**0**
1	0	0	1	**1**
2	0	1	0	**1**
3	0	1	1	**0**
4	1	0	0	**1**
5	1	0	1	**0**
6	1	1	0	**0**
7	1	1	1	**1**

II.4.2. Représentation algébrique

Il s'agit d'une représentation sous forme d'expression en fonction des entrées, on distingue deux formes : Forme disjonctive (SDP) et forme conjonctive (PDS).

II.4.2.1. La forme disjonctive (somme de produits - SDP)

C'est la somme des produits des variables, par exemple : $\mathrm{F(A,B,C)} = \overline{\mathrm{A}}\mathrm{BC} + \mathrm{A}\overline{\mathrm{C}} + \overline{\mathrm{A}}\mathrm{BC}$

<u>Forme disjonctive standard</u>

Dans la forme **standard**, chacun des termes de la somme doit comporter <u>toutes les variables</u> d'entrée. On dit également que la fonction est sous la **1ère forme canonique**. Sous cette forme, chacun des termes de la somme est appelé **minterme** (un minterme comporte toutes les variables d'entrée). Si une fonction n'est pas sous forme standard, on peut faire apparaître les variables manquantes.

- ***Exemple :***

Dans la fonction : $\mathrm{F(A,B,C)} = \overline{\mathrm{A}}\mathrm{BC} + \mathrm{A}\overline{\mathrm{C}} + \overline{\mathrm{A}}\mathrm{BC}$ il manque la variable B dans le 2ème terme. Pour le faire apparaître, on peut le multiplier par $(\bar{B} + B)$ car ce terme vaut 1, et le 1 est l'élément neutre pour le produit logique. On a donc : $F(A,B,C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

L'intérêt de la forme standard est de faciliter l'écriture de la table de vérité et des tableaux de Karnaugh (voir plus loin), elle permet de remplir directement ces derniers.

Détermination de la forme disjonctive à partir de la table de vérité.

Pour chaque ligne où la sortie vaut 1, on effectue les produits des variables d'entrée, complémentées si elles valent 0, non complémentées si elles valent 1. Puis on effectue la somme de ces différents produits.

Par exemple, pour la fonction OU, dont la table de vérité est ci-contre, la forme disjonctive de la fonction est :

$$F(A,B) = \bar{A}B + A\bar{B} + AB$$

	A	B	A + B
	0	0	0
$\bar{A}B$	0	1	**1**
$A\bar{B}$	1	0	**1**
AB	1	1	**1**

Par utilisation des règles de l'algèbre de Boole, on retrouve bien l'expression de la fonction OU :

$$F(A,B) = \bar{A}B + A\bar{B} + AB = \bar{A}B + \boldsymbol{AB} + A\bar{B} + \boldsymbol{AB} = B(\bar{A} + A) + A(\bar{B} + B)$$

$$\Longrightarrow F(A,B) = A + B$$

II.4.2.2. La forme conjonctive (produit de somme - PDS)

C'est le produit des sommes des variables, par exemple :

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

Forme conjonctive standard

Dans la forme conjonctive standard, chaque terme du produit contient toutes les variables. On dit également que la fonction est sous la **2ème forme canonique**. Chacun des termes de la forme conjonctive standard est appelé **maxterme**.

- ***Exemple :***

La fonction suivante n'est pas sous la forme standard :

$\mathrm{F(A,B,C)} = (\overline{\mathrm{A}} + \mathrm{B} + \mathrm{C}).(\mathrm{A} + \overline{\mathrm{C}})$ *(Il manque la variable B dans le 2ème terme).*

Pour la faire apparaître, on peut additionner avec $(\bar{B}B)$ car ce terme vaut 0, et le 0 est l'élément neutre pour la somme logique, et utiliser la propriété de distributivité de la somme par rapport au produit. On a donc :

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

Détermination de la forme conjonctive à partir de la table de vérité

Pour déterminer l'expression de la fonction sous forme conjonctive standard à partir de la table de vérité, on considère les lignes où la fonction vaut 0 (le résultat correspond à $\bar{F}$) ; chaque 0 de la sortie correspond à un terme du PDS.

On écrit d'abord la forme conjonctive (de $\bar{F}$) comme on l'a fait précédemment, puis pour déterminer F on doit alors transformer la SDP en PDS en utilisant les règles de l'algèbre de Boole.

Par exemple, pour la table de vérité du OU-EXCLUSIF (XOR) :

	A	B	F = A⊕B
$\bar{A}\bar{B}$	0	0	**0**
	0	1	1
	1	0	1
AB	1	1	**0**

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

➢ ***Remarque :***

On peut remarquer que cette forme peut être déduite directement de la table de vérité où les maxtermes sont la somme des compléments des variables (pour F=0).

➢ ***Exemple***

Pour déterminer la forme conjonctive de la fonction de table ci-contre, on prend les lignes où F=0, chaque maxterme est obtenu par l'addition logique des compléments des variables, on peut donc écrire :

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

	A	B	C	F
$A + \mathrm{B} + \mathrm{C}$	0	0	0	**0**
	0	0	1	1
$A + \overline{\mathrm{B}} + \mathrm{C}$	0	1	0	**0**
$A + \overline{\mathrm{B}} + \overline{\mathrm{C}}$	0	1	1	**0**
	1	0	0	1
	1	0	1	1
	1	1	0	1
	1	1	1	1

- Vérifier ce résultat avec la méthode précédente.

<u>Déterminer la forme conjonctive à partir de la forme disjonctive</u>

<u>1ère méthode :</u>

On peut passer de la forme disjonctive à la forme conjonctive en effectuant des factorisations, et en utilisant la propriété de distributivité de la somme par rapport au produit.

➢ ***Exemple***

Avec la fonction XOR sous forme PDS standard :

$$F = \bar{A}B + A\bar{B} = \bar{A}B + (A.\bar{B}) = (\bar{A}B + A).(\bar{A}B + \bar{B}) = (\bar{A} + A).(B + A).(\bar{A} + \bar{B}).(B + \bar{B})$$
$$= (A + B).(\bar{A} + \bar{B})$$

<u>2ème méthode :</u>

On peut également utiliser une méthode plus systématique, consistant à déterminer $\bar{F}$ (de la table de vérité par ex.) et la mettre sous forme disjonctive puis lui appliquer le principe de dualité, en :

- ✓ Remplaçant $\bar{F}$ par F ;
- ✓ Remplaçant AND (×) par OR (+) et vice-versa ;
- ✓ Complémentant les variables.

➢ ***Exemple***

Reprenons la fonction XOR, $\bar{F}$ sous forme disjonctive standard peut s'écrire : $\overline{F} = \bar{A}.\bar{B} + A.B$

Avec le principe de dualité : $F = (A + B).(\bar{A} + \bar{B})$

II.4.2.3. Expression numérique

Pour simplifier la représentation de la fonction, on peut l'exprimer sous forme numérique, cette forme indique la valeur décimale correspondant aux combinaisons binaires des variables.

➢ ***Exemples :***

1. La fonction XOR, peut être notée :

$$F(A,B) = \sum(1,2)$$

	A	B	F(A, B)
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

2. La fonction de détection de parité « paire » :

$$F(A,B,C) = \sum(1,2,4,7)$$

	A	B	C	F
0	0	0	0	0
1	0	0	1	**1**
2	0	1	0	**1**
3	0	1	1	0
4	1	0	0	**1**
5	1	0	1	0
6	1	1	0	0
7	1	1	1	**1**

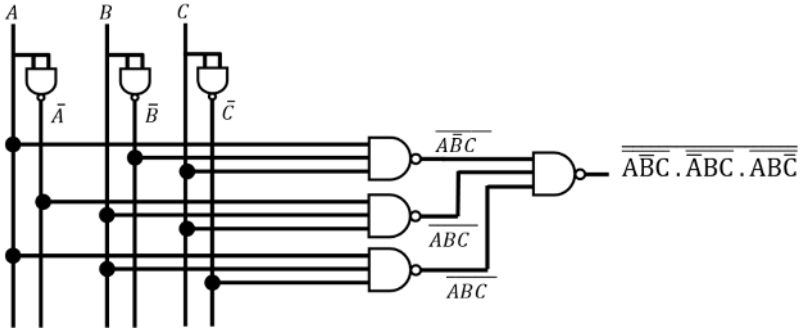
II.4.3. Le logigramme

C'est la traduction de la fonction logique en un schéma électronique. Le principe consiste à remplacer chaque opérateur logique par la porte logique qui lui correspond.

➢ ***Exemples :***

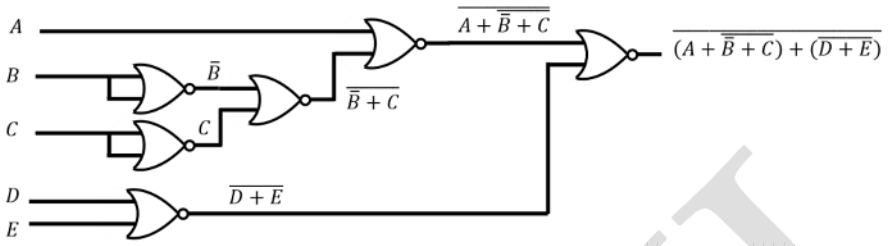
Reprenons les deux fonctions vues précédemment :

$F_1 = \overline{\overline{A\bar{B}C}.\overline{\bar{A}BC}.\overline{AB\bar{C}}}$

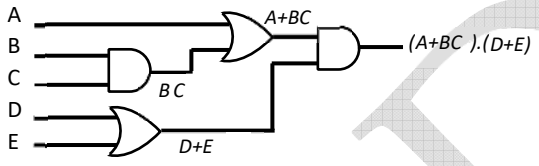


$$F_2 = (A + BC)(D + E) = \overline{\overline{\left(A + \overline{\bar{B} + \bar{C}}\right)} + \overline{(D + E)}}$$

- Avec des portes NOR à deux entrées uniquement :



- Avec des portes élémentaires simples :



II.5. Simplification des fonctions logiques

En pratique, les fonctions logiques sont réalisées avec des circuits électroniques. On cherche à représenter la fonction avec un minimum de termes car une fonction simplifiée utilisera moins de circuits. Elle sera exécutée plus rapidement (une porte logique possède un temps de propagation) et à un moindre coût.

Deux méthodes de simplification peuvent être utilisées : Méthode algébrique et tableaux de Karnaugh

II.5.1. Méthode algébrique

On utilise les différents théorèmes et propriétés de l'algèbre de Boole.

➢ ***Exemple***

Simplifier la fonction : $F(A,B,C) = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$

$$F(A,B,C) = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC = \bar{A}\bar{B}C + A\bar{B}\bar{C} + (A\bar{B}C + A\bar{B}C) + AB\bar{C} + ABC$$

$$= \bar{B}C.(\bar{A} + A) + A\bar{C}.(\bar{B} + B) + AC.(\bar{B} + B) = \bar{B}C + A\bar{C} + AC = \bar{B}C + A(\bar{C} + C) = \bar{B}C + A$$

II.5.2. Tableaux de Karnaugh

Les tableaux de Karnaugh constituent une autre représentation de la table de vérité de la fonction. Ils permettent de simplifier les fonctions logiques de manière graphique.

II.5.2.1. Principe représentation

✓ Les variables d'entrée sont placées dans la 1ère case en haut et à gauche du tableau. Les différentes combinaisons sont réparties en lignes et colonnes.

Par exemple, pour 3 variables, on peut en utiliser 2 pour constituer 4 colonnes (correspondant aux 4 combinaisons de ces 2 variables) et 1 pour constituer 2 lignes (correspondant aux 2 valeurs de cette variable).

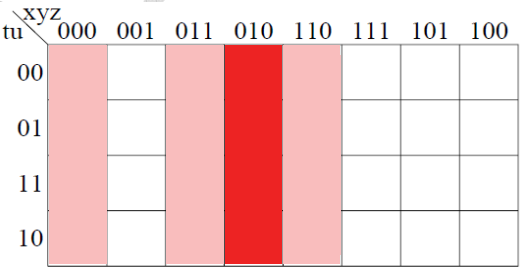
C \ AB	00	01	11	10
0				
1				

zt \ xy	00	01	11	10
00				
01				
11				
10				

✓ Entre 2 lignes (ou 2 colonnes) adjacentes, **1 seul bit change d'état** dans les combinaisons des variables. Cette règle doit être vraie également entre la dernière ligne (ou colonne) et la première. On peut pour cela utiliser le code binaire réfléchi (ou "code de Gray"), qui possède ces caractéristiques.

Par exemple : Le code binaire réfléchi pour 3 bits est : **000-001-011-010-100-101-111-110**

Le schéma ci-contre montre les colonnes adjacentes avec la colonne : $xyz = 010$



✓ Si l'on veut simplifier F on place, dans la table Karnaugh, les uns (1) correspondant à ses Mintermes (forme disjonctive) et si l'on cherche la simplification de $\bar{F}$ on doit placer les 0.

➢ ***Exemple***

Placer la fonction suivante :

$$F(A,B,C) = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

$\bar{A}\bar{B}C$

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

Comme dans la table de vérité, chaque terme de la somme se traduit par un 1 dans le tableau.

II.5.2.2. Règles de regroupement et simplification

Une fois le tableau de Karnaugh rempli, on cherche à effectuer des regroupements. Les règles de regroupement à respecter sont :

✓ Si l'on choisit de regrouper les 1, on obtient F ; si l'on choisit les 0, on obtient $\bar{F}$;

✓ Les regroupements doivent porter sur des uns (1), respectivement des zéros (0), **adjacents** ;

- ✓ Une case peut être regroupée plusieurs fois (théorème de redondance) ;
- ✓ Les regroupements peuvent porter sur $\mathbf{2^n}$ colonnes ou lignes, avec $\boldsymbol{n}$ entier naturel : 2, 4, 8,… etc.

Les groupements possibles pour l'exemple précédent sont :

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

- ✓ Un regroupement de 2 cases sur une ligne ou une colonne élimine une variable ; un regroupement de 2^n cases élimine n variables, **la variable à éliminer est celle qui change dans le regroupement**.

Dans l'exemple précédent, la fonction peut être réduite en une somme de 2 termes :

- Dans le groupe carré :

B et C qui changent de valeurs, elles vont disparaitre !

Il reste donc A, En effet :

$$AB\bar{C} + ABC + A\bar{B}\bar{C} + A\bar{B}C = AB + A\bar{B} = A$$

- Dans Le 2ème groupe c'est A qui change de valeur et $BC = 01$

La fonction peut donc se simplifier en : $F(A,B,C) = A + \bar{B}C$

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

A

$\bar{B}C$

➢ ***Exemple 1 :***

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

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

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

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

➢ ***Exemple 2 :***

On cherche à simplifier la fonction suivante : $F(A,,C,D,E) = \sum(0,4,8,12,13,15,16,17,19,23,29,31)$

ABC / DE	000	001	011	010	110	111	101	100
00	**1** (4)	**1** (12)	**1** (8)	**1** (24)	(28)	(20)	(16)	**1** (0)
01	(5)	(13)	**1** (9)	(25)	(29)	**1** (21)	(17)	**1** (1)
11	(7)	(15)	**1** (11)	(27)	(31)	**1** (23)	**1** (19)	**1** (3)
10	(6)	(14)	(10)	(26)	(30)	(22)	(18)	(2)

$A\bar{B}\bar{C}\bar{D}$

$\bar{A}\bar{D}\bar{E}$ BCE $A\bar{B}DE$

- Il y a deux groupes de quatre 1 : $(0,4,8,12) \longrightarrow \bar{A}\bar{D}\bar{E}$ et $(13,15,31,29) \longrightarrow BCE$
- Il y a 2 groupes de deux 1 : $(16,17) \longrightarrow A\bar{B}\bar{C}\bar{D}$ et $(9,23) \rightarrow A\bar{B}DE$

La fonction simplifiée est donc : $F = \bar{A}\bar{D}\bar{E} + BCE + A\bar{B}\bar{C}\bar{D} + A\bar{B}DE$

II.5.2.3. Combinaisons d'entrées non-utilisées

Il peut arriver qu'il y ait des combinaisons non-utilisées dans les variables d'entrée. Ces combinaisons correspondent à une sortie indéterminée (non-définies).

➢ ***Exemple 2 :***

Dans la table de vérité ci-contre, la fonction n'est pas définie pour les combinaisons (5,6,7).

	A	B	C	F(A,B,C)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5				
6				
7				

On met un X (une croix) dans l'emplacement correspondant dans tableau de Karnaugh, ces cases ne sont pas obligatoirement à regrouper mais peuvent être utilisées pour obtenir un groupement plus grand.

Si l'on a choisi de regrouper les 1, **<u>on peut</u>** considérer ces « X » comme des 1 de manière à permettre des regroupements plus grands (On peut les considérer comme des 0 si l'on regroupe les 0).

D'où la fonction simplifiée : $F = A + \bar{B}C$

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

TD n° 2 : *Algèbre de Boole et Simplification des fonctions logiques*

Exercice 1 :

En utilisant la table de vérité puis l'algèbre de BOOLE, démontrer la 2ème relation du théorème de consensus :

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

Exercice 2 :

Soit (A,B,C) définie par sa table de vérité :

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

1. Donner la forme canonique conjonctive et disjonctive de F.
2. Simplifier par la table de Karnaugh.
3. Donner le logigramme par des portes NOR

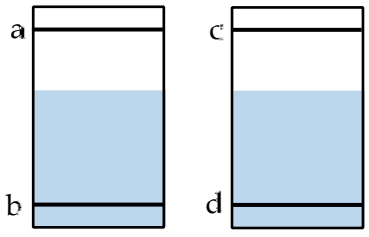
Exercice 3 :

Soient deux réservoirs R_1 et R_2 dont le niveau pour chacun est contrôlé par un détecteur de niveau haut (a pour R_1, c pour R_2) et un détecteur de niveau bas (b pour R_1, d pour R_2), un détecteur renvoie un (1) à la présence du liquide et 0 dans le cas inverse.

On dispose de trois voyants V_1, V_2, V_3, qui fonctionnent dans les conditions suivantes :

- $V_1 = 1$ si les deux réservoirs sont pleins.
- $V_2 = 1$ si les deux réservoirs sont vides.
- $V_3 = 1$ dans tous les autres cas (réservoir à moitié plein ou un plein un vide...).
- Un certain nombre de combinaisons sont technologiquement impossibles, les sorties V_1, V_2, V_3, prendront dans ces cas-là une valeur indifférente X (*ex. cd=10 !*)

1. Etablir la table de vérité de ce système.
2. Déterminer les équations logiques simplifiées des voyants.
3. Réaliser le logigramme de V_1, V_2, V_3 avec des portes NAND.



Exercices supplémentaires

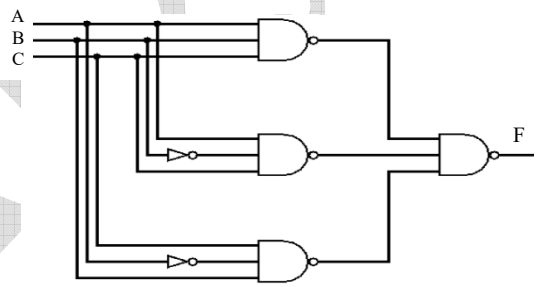
Exercice 1 :

1. Soit la fonction : $F(A,B,C) = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$
   - Déterminer $F(1,0,1), F(1,0,1), et\ F(1,1,0)$
2. En précisant à chaque fois les propriétés utilisées, montrer que $F(A,B,C) = A + \bar{B}.C$
3. Calculer le complément de la fonction : $F = (a+b)(\bar{a}+\bar{b})$

Exercice 2 :

1. Déterminer la forme disjonctive (SDP) standard de la fonction suivante : $F = \bar{A}\bar{B} + AB\bar{C}D$
2. Soit la fonction suivante : $F = (\bar{A} + BC)(A + \bar{B}C)(B + D)$
   - Simplifier cette fonction avec l'algèbre de Boole ;
   - Ecrire F sous la 1ère forme canonique.

Exercice 3 :



3. Trouver l'expression logique de la fonction F.
4. Simplifier cette expression par la méthode algébrique.
5. Proposer un logigramme plus simple de cette fonction

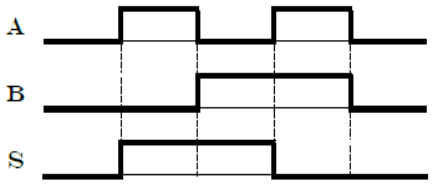
Exercice 4 :

On considère la fonction logique suivante : $F = XY + XZ + Y\bar{Z}$

1. Déterminer sa forme minimale disjonctive et conjonctive.
2. Réaliser la fonction simplifiée à l'aide d'opérateurs NAND.
3. En déduire le circuit à base d'opérateurs NOR.
4. Retrouver ce résultat à partir de l'expression de la fonction.

Exercice 5 :

On donne le chronogramme d'une fonction combinatoire $S(A,B)$.



- Donner la table de vérité et déduire son expression ;
- Représenter S par une seule porte logique

Exercice 6 :

Simplifier les fonctions suivantes algébriquement puis avec le tableau de Karnaugh.

$F_1 = (A + \bar{B})(A + B + \bar{C})(\bar{A} + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C} + \bar{D})(\bar{A} + \bar{B} + \bar{C} + D)$

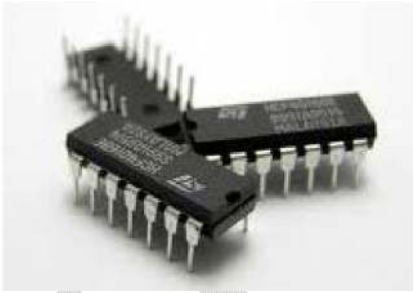
$F_2 = (\bar{A}.B + A.B + A.\bar{B}).(C.\bar{D} + \bar{C}.\bar{D}) + \bar{C}.D.(\bar{A}.B + A.B)$

Chapitre III. Circuits combinatoires

Introduction

Un certain nombre de fonctions logiques sont très utilisées dans la conception des systèmes combinatoires, ces fonctions sont disponibles dans des circuits intégrés, on peut trouver principalement :

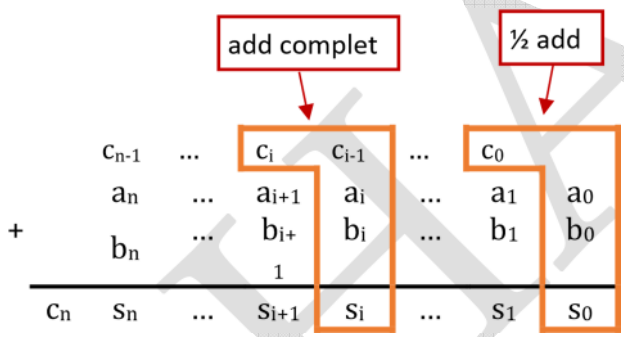
- Les additionneurs
- Comparateur
- Multiplexeur/ Démultiplexeur
- Encodeur / Décodeur
- Transcodeur



On peut utiliser ces circuits pour réaliser différents systèmes combinatoires.

III.1. Les Additionneur/Soustracteur

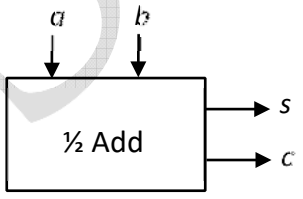
L'addition arithmétique des nombres binaires se fait par un circuit combinatoire, l'addition de deux bits est réalisée par un demi-additionneur (sans retenue précédente) ou additionneur complet (avec retenue précédente).



III.1.1. Demi-additionneur

Un demi-additionneur (Half-Adder) réalise l'addition de deux bits sans retenue précédente.

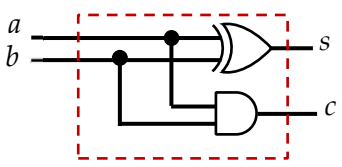
Soit l'addition de deux bits ***a*** et ***b***, on obtient la somme ***s*** et une éventuelle retenue ***c*** (carry-out).



a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

De la table de vérité on détermine l'expression de s et c :

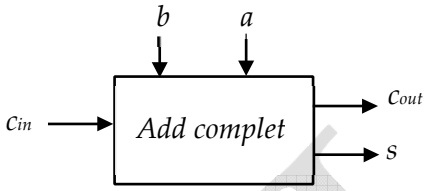
$$\begin{cases} s = \bar{a}b + a\bar{b} = a \oplus b \\ c = ab \end{cases}$$



III.1.2. Additionneur complet

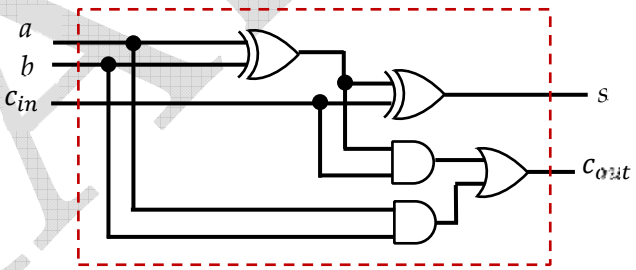
Dans un additionneur complet (Full-Adder) on prend en compte une retenue précédente (carry-in).

a	b	c_{in}	s	c_{out}
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



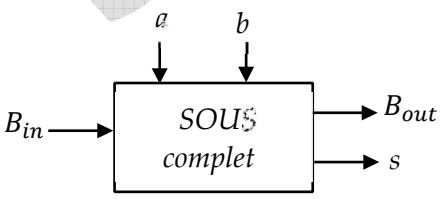
$$s = \bar{a}\bar{b}c_{in} + \bar{a}b\overline{c_{in}} + a\bar{b}\overline{c_{in}} + abc_{in} = (\bar{a}b + a\bar{b}).\overline{c_{in}} + (\bar{a}\bar{b} + ab).c_{in} = (a \oplus b).\overline{c_{in}} + \overline{(a \oplus b)}.c_{in}$$
$$= a \oplus b \oplus c_{in}$$

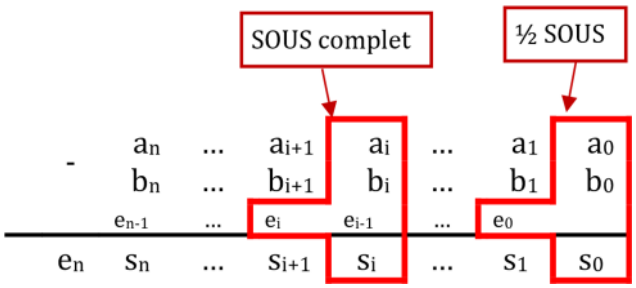
$$c_{out} = \bar{a}bc_{in} + a\bar{b}c_{in} + ab\overline{c_{in}} + abc_{in} = (\bar{a}b + a\bar{b})c_{in} + ab(\overline{c_{in}} + c_{in})$$
$$= (a \oplus b).c_{in} + ab$$



III.1.3. Demi-Soustracteur et Soustracteur complet

De la même manière, un demi-soustracteur (Half-Subtractor HS) réalise la soustraction de deux bits sans tenir compte d'un précédent emprunt (Borrowing), par exemple la soustraction des LSB (Least Significant Bits), et un soustracteur complet (Full-Subtractor) tient compte d'un éventuel emprunt B_{in} (e_{i-1} sur le schéma suivant), c'est le cas de la soustraction des deux bits au milieu d'une chaine binaire (voir le schéma).



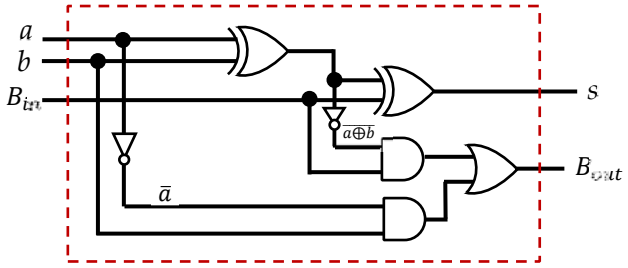


Le résultat de la soustraction **s** et l'emprunt du poids plus fort B_{out} sont donnés par :

$s = a \oplus b \oplus B_{in}$

$B_{out} = \bar{a}\bar{b}B_{in} + \bar{a}b\bar{B}_{in} + \bar{a}bB_{in} + abB_{in} = (\bar{a}\bar{b} + ab)B_{in} + \bar{a}b(\bar{B}_{in} + B_{in})$

$= (\overline{a \oplus b}).B_{in} + \bar{a}b$



a	b	B_{in}	s	B_{out}
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

➢ ***Remarque :***

On peut réaliser un soustracteur de deux bits en utilisant le CA2, en effet :

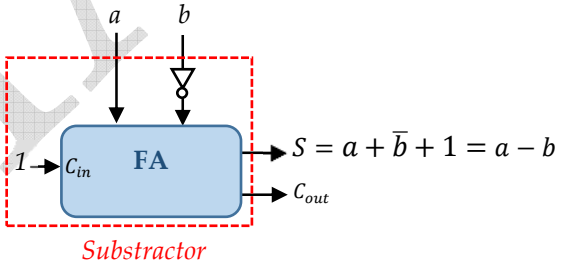
$$a - b = a + CA2(b) = a + \bar{b} + 1$$

Attention **!**

Le signe + dans la relation précédente est un opérateur arithmétique (addition) et non un opérateur logique (OR)

➢ ***Exemple :***

En utilisant la relation précédente, on peut réaliser un soustracteur de deux bits (a-b) à l'aide d'un ADD-complet (Full-adder FA), il s'suffit de rajouter 1 (à l'entrée C_{in}) et de complémenter b :



Substractor

III.1.4. Additionneur de deux nombres

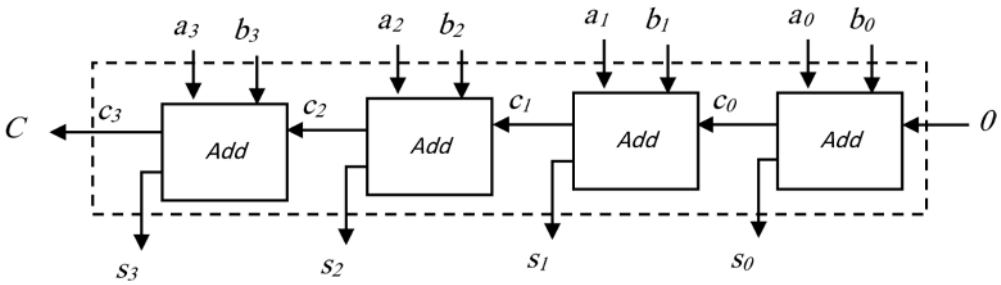
$S = A + B$

Avec : $A = a_3a_2a_1a_0$

$B = b_3b_2b_1b_0$

$S = a_3a_2a_1a_0$

	C_3	C_2	C_1	C_0	0
A		a_3	a_2	a_1	a_0
B	+	b_3	b_2	b_1	b_0
S	C	S_0	S_0	S_0	S_0



III.2. Comparateur

Un comparateur permet de comparer 2 mots binaires, c'est à dire d'indiquer si ces 2 mots sont égaux, mais également, si ce n'est pas le cas, lequel est le plus grand. Il est basé sur l'utilisation d'un comparateur de deux bits.

III.2.1. Comparateur d'égalité

On a vu précédemment que la fonction XNOR (NOT-XOR) à deux entrées est un détecteur d'égalité ; donc pour deux bits a et b, la fonction d'égalité E est donnée par : $E = a \odot b$

Pour deux mots binaires : $A = a_n \dots a_1 a_0$ et $B = b_n \dots b_1 b_0$, on peut écrire :

$$E = (a_n \odot b_n) \dots (a_1 \odot b_1)(a_0 \odot b_0)$$

III.2.2. Comparateur complet

Un comparateur complet de deux bits a et b possède trois sorties :

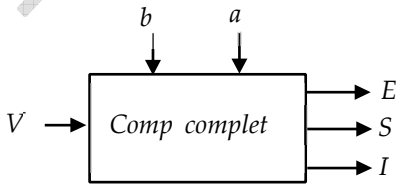
$$\begin{cases} a = b \Rightarrow E = 1 \\ a > b \Rightarrow S = 1 \\ a < b \Rightarrow I = 1 \end{cases} \Rightarrow \begin{cases} E = \bar{a}\bar{b} + ab = a \odot b \\ S = a\bar{b} \\ I = \bar{a}b \end{cases}$$

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

Si l'on rajoute une entrée de validité V du circuit :

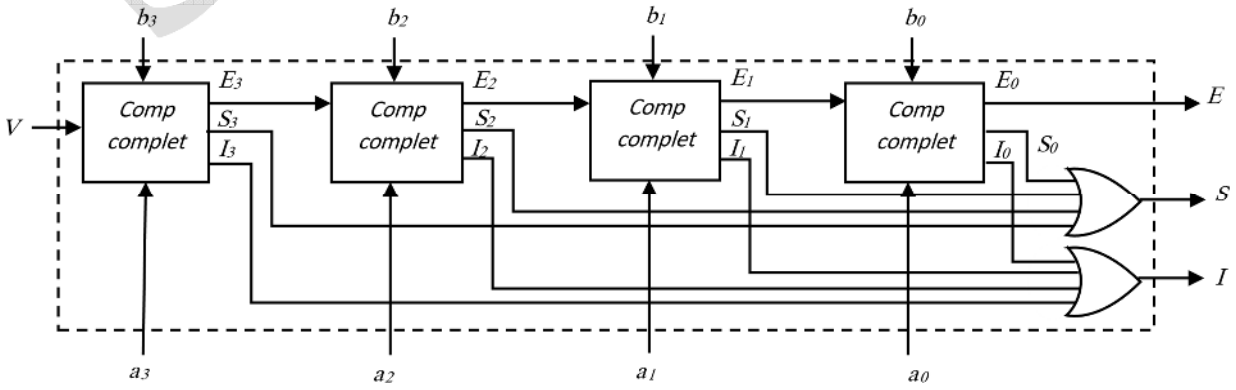
$$\begin{cases} V = 0 \Rightarrow Toutes\ les\ osrties = 0\ (circuit\ désactivé) \\ V = 1 \Longrightarrow Le\ ciruit\ fonctionne\ normalement \end{cases}$$

$$\Rightarrow \begin{cases} E = (a \odot b).V \\ S = (a\bar{b}).V \\ I = (\bar{a}b).V \end{cases}$$



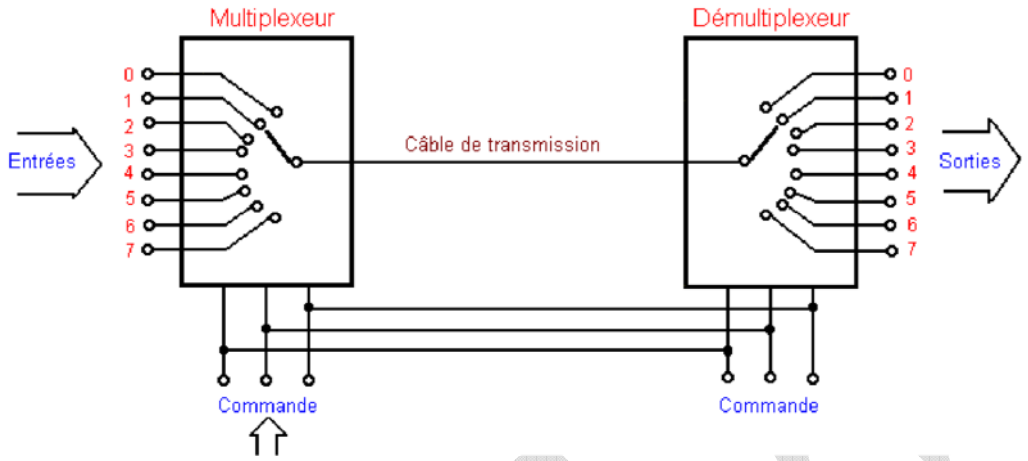
III.2.3. Comparateur de deux mots

Pour deux mots binaires : $A = a_n \dots a_1 a_0$ et $B = b_n \dots b_1 b_0$, on peut utiliser des comparateurs de deux bits. On commence à comparer les bits du poids fort, si $E = 1$ on compare les bits suivants sinon ($E = 0$) on arrête la comparaison (on connait le résultat $S = 1\ ou\ I = 1$), dans ce cas les autres comparateurs seront désactivés on peut donc utiliser E comme entrée de validité du prochain comparateur.



III.3. Multiplexeur/ Démultiplexeur

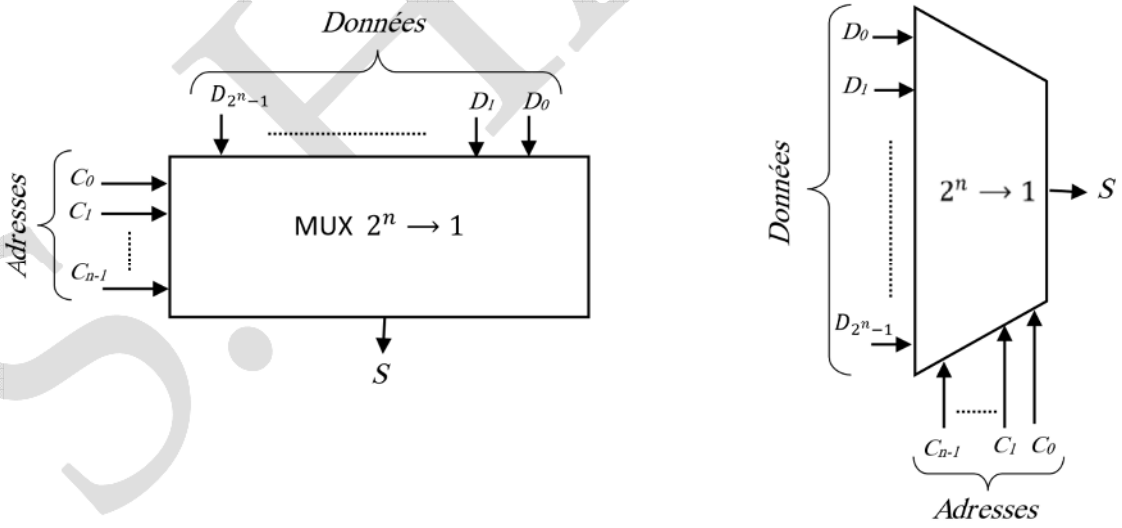
Si l'on veut, par exemple, transmettre plusieurs signaux sur un seul canal, ce dernier doit être partagé par tous les signaux et ceci peut être réalisé par les Multiplexeurs/Démultiplexeurs, le schéma suivant explique le principe :



Les Multiplexeurs/ Démultiplexeurs sont donc utilisés pour la conversion série/parallèle, mais également pour synthétiser n'importe quelle fonction logique.

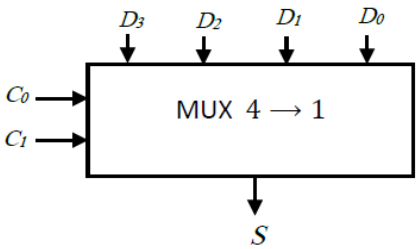
III.3.1. Multiplexeur (MUX)

Un multiplexeur est un circuit combinatoire qui permet de sélectionner une seule entrée d'information D_i (donnée) parmi 2^n entrées, ceci suivant les combinaisons de n entrées de commande (adresses).



Multiplexeur : 4 vers 1

Pour chaque combinaison des entrées de sélection une seule donnée est transmise à la sortie.



C_1	C_0	S
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$S = \overline{C_1}\,\overline{C_0}.D_0 + \overline{C_1}C_0.D_1 + C_1\overline{C_0}.D_2 + C_1C_0.D_3$$

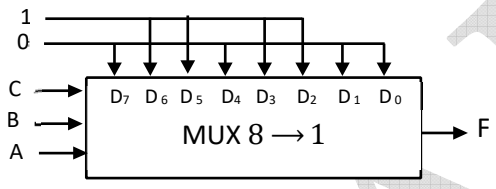
- ***Exemples :*** *Réaliser une fonction logique par un MUX*

1. Réaliser la fonction $F = \sum(2,3,5,6)$ avec un MUX 8 vers 1.

MUX $(2^3 \rightarrow 1) \Longrightarrow$ 3 entrées de sélection

Chaque combinaison correspond à une en entrée de donnée

⇒ La valeur décimale de la combinaison donne l'indice de l'entrée



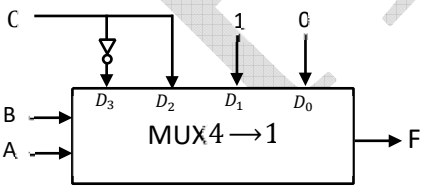
	A	B	C	F	
0	0	0	0	0	D_0
1	0	0	1	0	D_1
2	0	1	0	1	D_2
3	0	1	1	1	D_3
4	1	0	0	0	D_4
5	1	0	1	1	D_5
6	1	1	0	1	D_6
7	1	1	1	0	D_7

2. Réaliser la fonction $F = \sum(2,3,5,6)$ avec un MUX 4 vers 1.

⇒ 2 entrées de sélection (par ex. A et B) et 4 entrées de données

On peut facilement déterminer les valeurs des entrées *Di* :

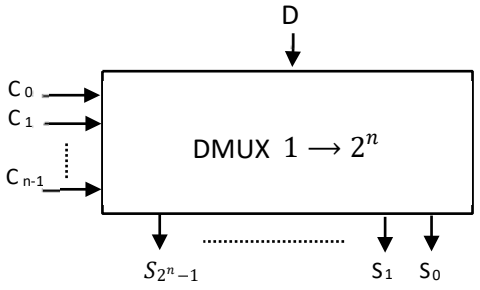
$$D_0 = 0\,, D_1 = 1\,, D_2 = C\,, D_3 = \overline{C}$$

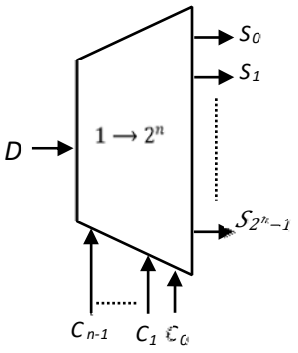


	A	B	C	F	
0	0	0	0	0	D_0
	0	0	1	0	
1	0	1	0	1	D_1
	0	1	1	1	
2	1	0	0	0	D_2
	1	0	1	1	
3	1	1	0	1	D_3
	1	1	1	0	

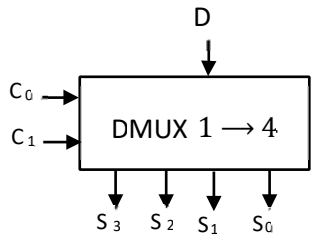
III.3.2. Démultiplexeur (DMUX)

Un démultiplexeur réalise l'opération inverse de celle du multiplexeur, il permet de faire passer une information D dans l'une des sorties Si selon les combinaisons des entrées de commandes.





Démultiplexeur : 1 vers 4



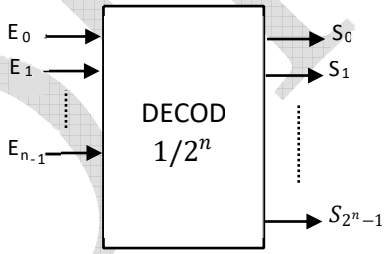
C_1	C_0	S_3	S_2	S_1	S_0
0	0	0	0	0	**D**
0	1	0	0	**D**	0
1	0	0	**D**	0	0
1	1	**D**	0	0	0

Avec : $S_0 = \bar{C_1}\bar{C_0}.D$ $S_1 = \bar{C_1}C_0.D$ $S_2 = C_1\bar{C_0}.D$ $S_3 = C_1C_0.D$

III.4. Décodeur / Encodeur

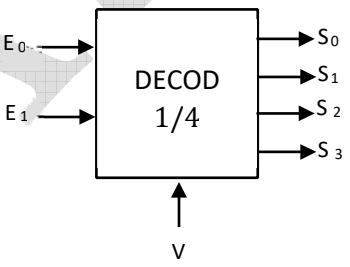
III.4.1. Décodeur

C'est un circuit combinatoire qui est constitué de n entrées de données et 2^n sorties (Décodeur 1 parmi 2^n). Pour chaque **combinaison** en entrée **une seule** sortie est active à la fois.



Décodeur 1/4

V	E_1	E_0	S_3	S_2	S_1	S_0
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0



Avec : V une entrée de validité

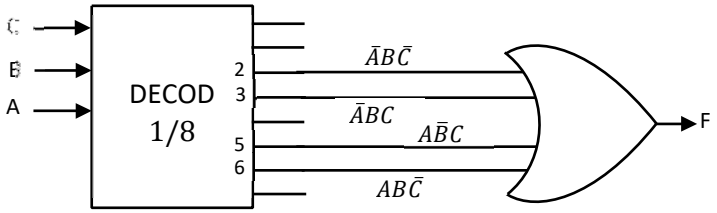
Pour V=1 : $S_0 = \bar{E}_1\bar{E}_0$ $S_1 = \bar{E}_1E_0$ $S_2 = E_1\bar{E}_0$ $S_3 = E_1E_0$

- ***Remarque***

On peut remarquer qu'on retrouve en sortie toutes les combinaisons possibles des entrées (tous les mintermes), en récupérant les mintermes désirés on peut exprimer toute fonction logique.

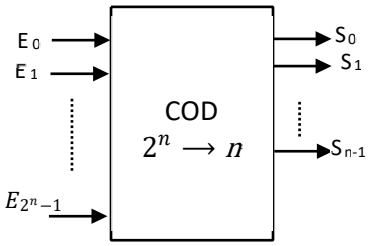
- ***Exemple :***

Réaliser la fonction $F = \sum(2,3,5,6)$ avec un décodeur 1/8.

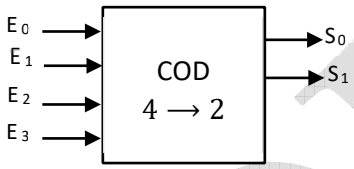


III.4.2. L'encodeur binaire

Il joue le rôle inverse d'un décodeur. Pour chaque entrée active on aura son numéro (en binaire) à la sortie.



L'encodeur : 4 vers 2



1er cas

On suppose qu'une seule entrée est active à la fois (les autres sont à 0), le cas où plus d'une entrée à 1 est impossible.

E1 E0 \ E3 E2	00	01	11	10
00	X	**0**	X	**1**
01	**0**	X	X	X
11	X	X	X	X
10	1	X	X	X

$S_0 = E_1 + E_3$

E1 E0 \ E3 E2	00	01	11	10
00	X	**1**	X	**1**
01	**0**	X	X	X
11	X	X	X	X
10	**0**	X	X	X

$S_1 = E_2 + E_3$

E3	E2	E1	E0	S1	S0
0	0	0	**1**	0	0
0	0	**1**	0	0	1
0	**1**	0	0	1	0
1	0	0	0	1	1

Dans ce cas on peut aussi procéder d'une manière plus simple puisque une seule entrée active à la fois. Par ex :

S0 =1 lorsque *E1* est active (E1=1) ou *E3=1*, donc :

$$S_0 = E_1 + E_3$$

$$S_1 = E_2 + E_3$$

L'entrée active (=1)	S1	S0
E0	0	0
E1	0	1
E2	1	0
E3	1	1

2ème cas

Dans la pratique on peut avoir plus d'une entrée à 1, on définit alors l'encodeur prioritaire où l'entrée du poids plus fort est prioritaire (par ex. si $E_3 = 1$ et $E_2 = 1 \Rightarrow S_1S_0 = 11$).

$E_1 E_0$ \ $E_3 E_2$	00	01	11	10
00	**X**	**0**	**1**	**1**
01	**0**	**0**	**1**	**1**
11	**1**	**0**	**1**	**1**
10	**1**	**0**	**1**	**1**

$$S_0 = E_1\overline{E_2} + E_3$$

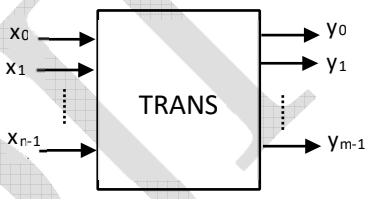
$E_1 E_0$ \ $E_3 E_2$	00	01	11	10
00	**X**	**1**	**1**	**1**
01	**0**	**1**	**1**	**1**
11	**0**	**1**	**1**	**1**
10	**0**	**1**	**1**	**1**

$$S_1 = E_2 + E_3$$

E_3	E_2	E_1	E_0	S_1	S_0
0	0	0	**1**	0	0
0	0	**1**	X	0	1
0	**1**	X	X	1	0
1	X	X	X	1	1

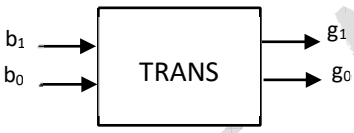
III.5. Le transcodeur

C'est un circuit combinatoire qui permet de transformer un code X (sur n bits) en entrée en un code Y (sur m bits) en sortie.



➢ ***Exemple 1 :***

Réaliser un transcodeur Binaire-Gray



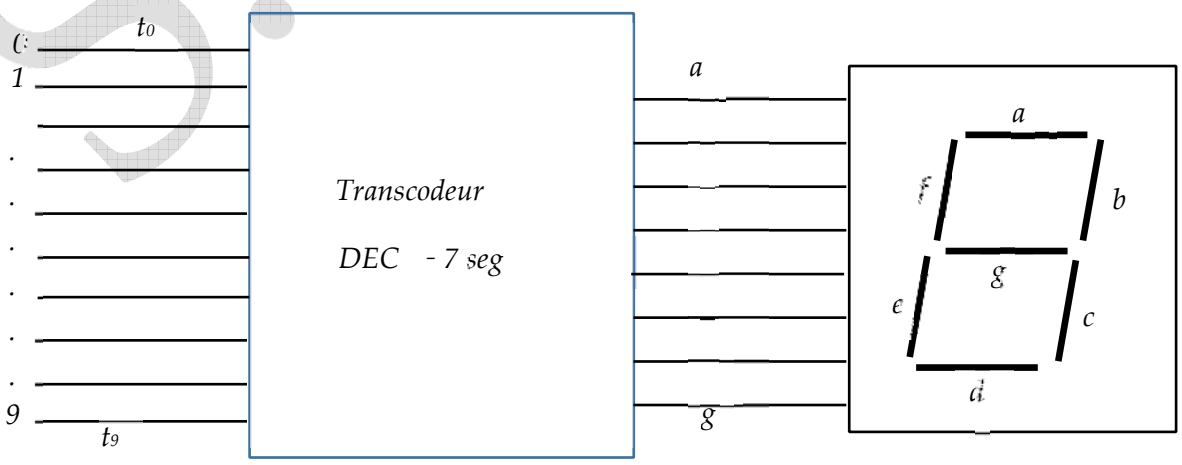
b_1	b_0	g_1	g_0
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

On aura :

$$g_0 = \bar{b}_1 b_0 + b_1 \bar{b}_0 = b_1 \otimes b_0 \qquad g_1 = b_1 \bar{b}_0 + b_1 b_0 = b_1$$

➢ ***Exemple 2 :***

Un clavier composé de 10 touches (interrupteurs) de 0 à 9 représentant les chiffres décimaux, une touche est activée (=1) en cliquant dessus. Réaliser un transcodeur Décimal-7segments qui reçoit un chiffre décimal sur l'une des dix entrées logiques (une seule entrée active à la fois) et le visualise sur un afficheur 7 segments.

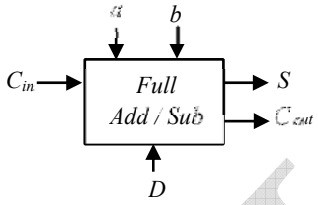


TD n° 3 : *Circuits combinatoires*

Exercice 1 :

En utilisant des portes logiques, réaliser un ADD/SOUS complet de deux bits a et b avec une entrée de sélection D, avec :

$$\begin{cases} D = 0 \Rightarrow S = a + b \\ D = 1 \Rightarrow S = a - b \end{cases}$$



Exercice 2 :

- Donner la table de vérité d'un comparateur complet de deux bits a et b ;
- Réaliser ce comparateur avec un décodeur 1/4 et une porte OR.

Exercice 3 :

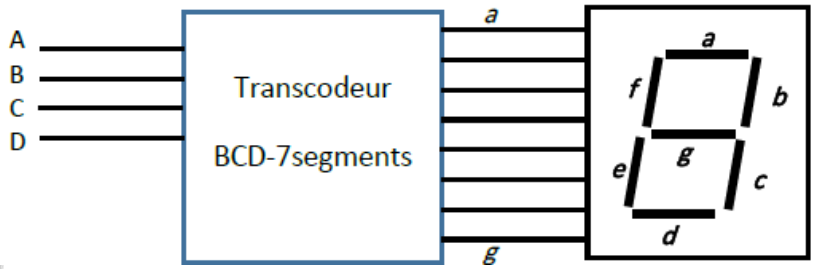
Soit la fonction donnée par la table de vérité ci-contre.

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

- Réaliser cette fonction avec un MUX 8 vers 1 puis un MUX 4 vers 1

Exercice 4 :

Un transcodeur BCD (**B**inary **C**oded **D**ecimal) vers 7 segments permet de visualiser sur un afficheur à 7 LEDs (**L**ight-**E**mitting **D**iodes) la valeur décimale d'un chiffre codé en binaire.



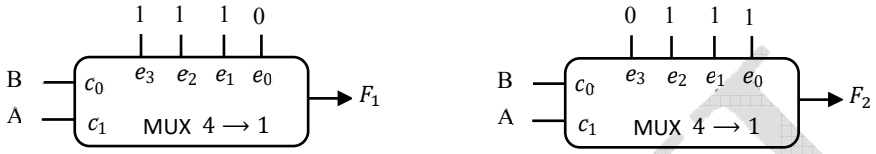
1. On suppose que les combinaisons des variables d'entrée ne correspondant pas à un chiffre décimal peuvent se produire et que l'on souhaite que tous les segments restent éteints dans ces cas-là. Déterminer la fonction logique permettant d'obtenir le segment "a".
2. On suppose maintenant que ces combinaisons ne risquent pas de se produire. En effectuant les simplifications adéquates, déterminer la fonction logique correspondant à chaque segment de l'afficheur.

Exercices supplémentaires

Exercice 1 :

Soit le schéma de ci-après :

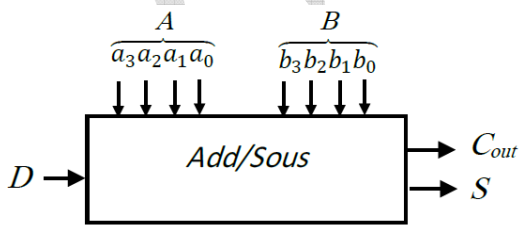
- Déterminer l'expression simplifiée de $F_1(A,B)$ et $F_2(A,B)$
- Réaliser le logigramme de chaque fonction avec une seule porte logique.



Exercice 2 :

En utilisant la soustraction en CA2 et à partir des additionneurs complets de deux bits, réaliser un ADD/SOUS de deux nombres $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$ avec une entrée de sélection D, avec :

$$\begin{cases} D = 0 \Rightarrow S = A + B \\ D = 1 \Rightarrow S = A - B \end{cases}$$

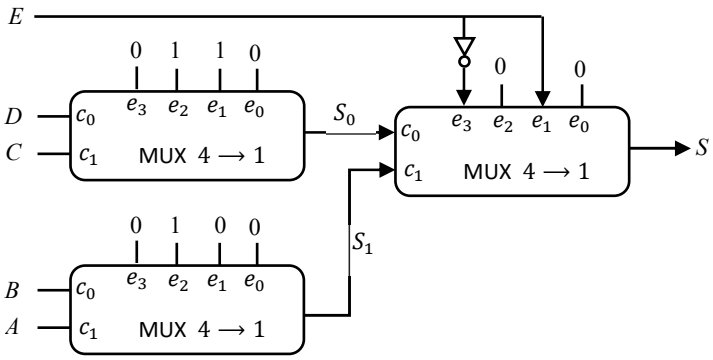


Exercice 3 :

1. Donner les expressions des sorties d'un décodeur à 3 entrées (1/8).
2. On souhaite réaliser la fonction XOR à 3 entrées ($F = A \oplus B \oplus C$) à l'aide de ce décodeur.
   - Ecrire la forme disjonctive de cette fonction ;
   - Donnez le logigramme de F en utilisant le décodeur précédent.

Exercice 4 :

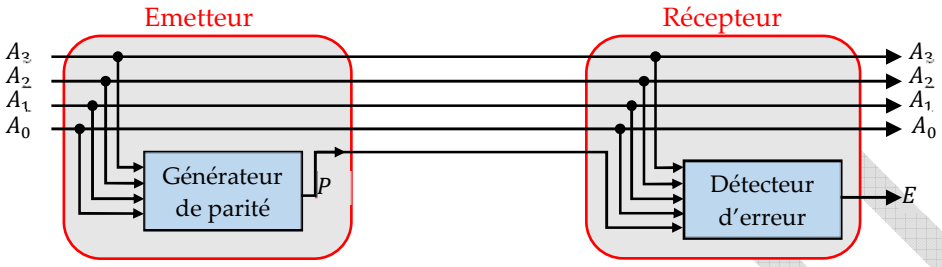
Soit le schéma suivant, Donnez l'équation simplifiée de S en fonction de A, B, C, D, et E.



Exercice 5 :

On veut réaliser un transmetteur/récepteur avec un bit de parité suivant le schéma suivant.

1. Donner la table de vérité du générateur de parité « paire » du 5ème bit (P).
   - ➢ Réaliser son logigramme.
2. Donner la table de vérité du détecteur d'erreur (E) puis réaliser son logigramme.

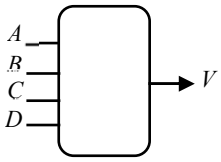


Exercice 6 :

Une société à 4 actionnaires ayant le nombre suivant d'actions *A(60), B(100), C(160), D(180)*. On désire construire une machine permettant le vote automatique lors des réunions. Chaque actionnaire, dont le poids de vote est proportionnel au nombre d'actions, appuie sur un bouton qui porte son nom (A, B, C ou D).

- Si un actionnaire vote OUI, sa variable vaut 1, s'il vote NON, elle vaut 0.
- Une résolution sera votée (V=1) si la somme des actions correspondant aux votes OUI représente au moins la moitié des actions plus 1.

  Exemple : $AC = 11\ (220)\ et\ BD = 00\ (280) \Rightarrow V = 0$



1. Donner la table de vérité du système *(Remarquez qu'il y a autant de 0 de V que de 1)* ;
2. Ecrire l'expression logique de V puis simplifier ;
3. Représenter cette fonction avec un multiplexeur 8 vers 1 (3 entrées d'adresse).
4. Dessiner le logigramme de la sortie avec des portes NAND.

Exercice 7 :

Le comité directeur d'une entreprise est constitué de quatre membres : le directeur D et ses trois adjoints A, B, C. Lors des réunions, la décision est prise (V=1) à la majorité.

- Chaque membre appuie sur un bouton qui porte son nom (A, B, C ou D).
- Si un membre vote OUI, sa variable vaut 1, s'il vote NON, elle vaut 0.
- En cas d'égalité du nombre de voix, celle du directeur compte double.

1. Donner la table de vérité du système *(Remarquez qu'il y a autant de 0 de V que de 1)* ;
2. Ecrire l'expression logique de V puis simplifier ;
3. Dessiner le logigramme de la sortie avec des portes NAND ;
4. Représenter cette fonction avec un multiplexeur 8 vers 1 (3 entrées d'adresse).

Exercice 8 :

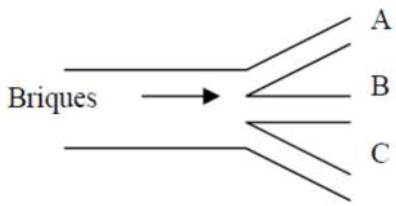
On dispose de 4 critères pour déterminer si une brique est bonne ou non (0 incorrect, 1 correct) :

Le poids P, la longueur L, la largeur G, la hauteur H.

En fonction de ces critères, les briques sont rangées suivant 3 catégories :

A. Poids et au moins deux dimensions correctes,
B. Seul le poids est incorrect, ou le poids est correct et une dimension est correcte au maximum,
C. Le poids est incorrect et 2 dimensions sont correctes au maximum.

1. Donner la table de vérité des sorties A, B et C suivant les critères P, L, G et H ;
2. Ecrire les équations simplifiées (par tableau de Karnaugh) ;
3. Dessiner le logigramme à l'aide de 2 circuits intégrés contenant 3 NAND à 3 entrées et de 1 circuit intégré contenant 4 NOR à 2 entées.



Chapitre IV. Les bascules

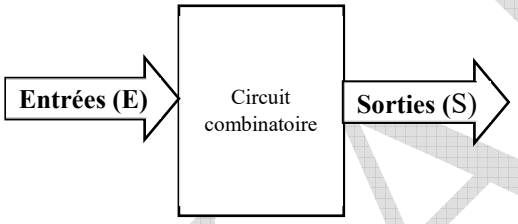
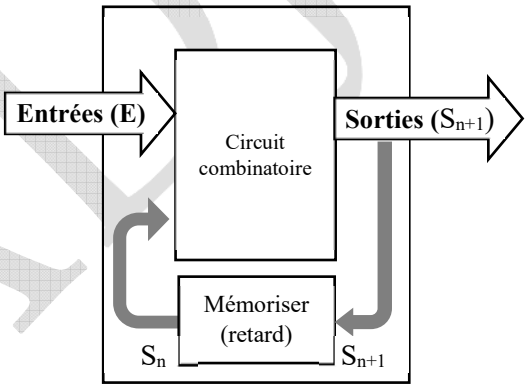
Introduction :

On a vu que dans un système combinatoire les sorties dépondent uniquement des entrées, pour les mêmes entrées on a toujours les mêmes sorties.

Dans un système séquentiel, les sorties dépendent non seulement des entrées mais aussi des sorties précédentes (le système mémorise les sorties précédentes), donc pour les mêmes entrées on n'a pas toujours les mêmes sorties.

Dans la table de vérité d'un système séquentiel on trouve, en plus des entrées, les sorties à l'état précédent.

Le tableau suivant résume la différence entre les deux systèmes :

	Système Combinatoire	**Système séquentiel**
Architecture		
Equation	$S = f(E)$	$S_{n+1} = f(E, \boldsymbol{S_n})$
Table de vérité	E | S	E | S_n | S_{n+1}

On distingue deux types de systèmes logiques séquentiels :

- Les circuits séquentiels **asynchrones**, dans lesquels les sorties évoluent dès qu'il y a un changement sur l'une des entrées ;
- Les circuits séquentiels **synchrones**, dans lesquels les sorties ne peuvent évoluer que si un signal d'horloge est actif.

Les bascules sont les circuits logiques de base de la logique séquentielle, elles possèdent deux sorties complémentaires Q et $\bar{Q}$. On note Q_{n+1} la sortie à l'état actuel et Q_n la sortie mémorisée (à l'état précédent).

Il existe des bascules asynchrones (sans horloge) et des bascules synchrones (avec horloge).

IV.1. Bascules asynchrones

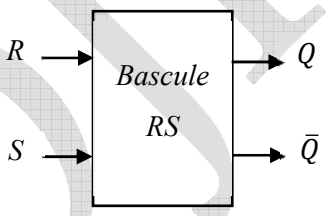
Une bascule asynchrone n'évoluent pas au rythme d'un signal d'horloge mais suivant les états de ses entrées et son état précédent.

IV.1.1. Bascule RS

Une bascule RS comporte deux entrées principales : Une entrée R (Reset) de mise à zéro (la sortie $Q = 0$) et une entrée S (Set) de mise à un (($Q = 1$).

Son fonctionnement se résume ainsi :

S	R	Q_{n+1}	
0	0	Q_n	Etat mémoire
0	1	0	Mise à 0
1	0	1	Mise à 1
1	1	X	Cas interdit (indéfini)



La table de vérité est donc : $Q_{n+1} = f(S, R, Q_n)$

S	R	Q_n	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	X
1	1	1	X

Q_n \ SR	00	01	11	10
0			**X**	**1**
1	**1**		**X**	**1**

Q_n \ SR	00	01	11	10
0	**0**	**0**	**X**	
1		**0**	**X**	

$$Q_{n+1} = S + \bar{R}Q_n \qquad \overline{Q_{n+1}} = R + \bar{S}\bar{Q}_n$$

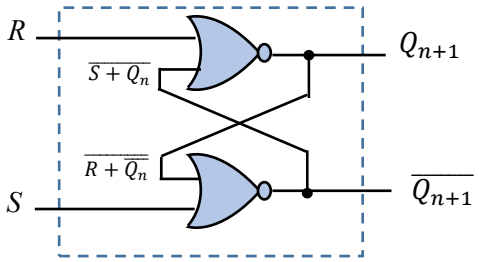
Remarque : $SR = 11 \Rightarrow Q_{n+1} = 1 \; et \; \overline{Q_{n+1}} = 1$!

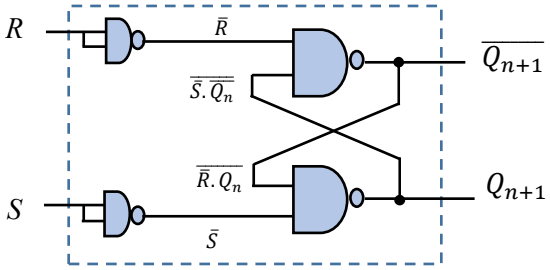
Logigramme avec des portes NOR :

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

Avec des portes NAND :

$$Q_{n+1} = \overline{\bar{S}.\overline{\bar{R}.Q_n}} \quad , \quad \overline{Q_{n+1}} = \overline{\bar{R}.\overline{\bar{S}.\overline{Q_n}}}$$





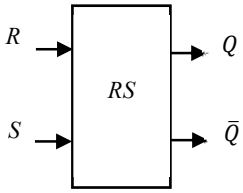
Le chronogramme :

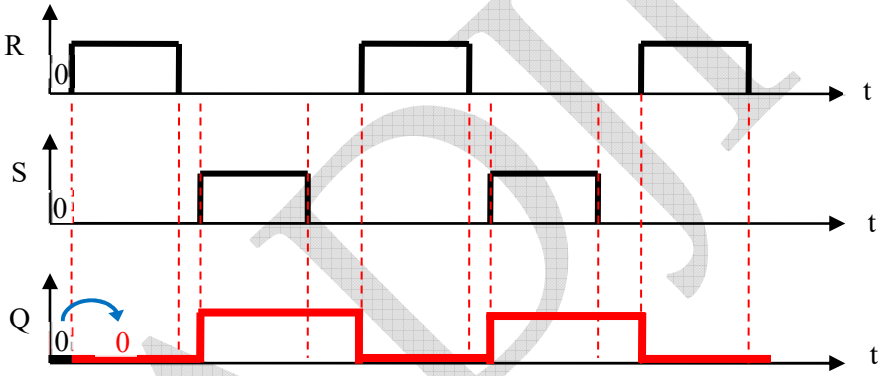
C'est l'évolution des variables logiques en fonction du temps.

Dans un système séquentiel, la sortie Q dépond des entrées mais aussi de **sa valeur précédente**.

➢ ***Exemple :***

Au départ Q=0

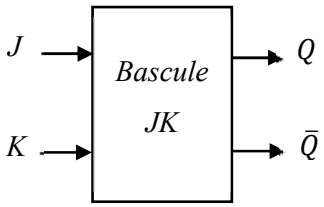




IV.1.2. Bascule JK

La bascule JK vient prendre en charge le cas indéterminé de la bascule RS, son fonctionnement est défini par la table suivante :

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



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

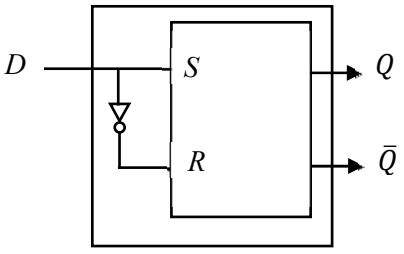
$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

IV.1.3. Bascule D

Dans cette bascule la sortie Q reproduit l'entrée D

D	Q_{n+1}
0	0
1	1

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

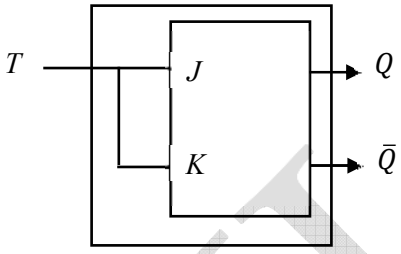


Sous cette forme, la bascule D n'a pas grand intérêt puisqu'il s'agit d'un bloc fonctionnel qui ne fait que recopier son entrée, en permanence. On verra plus loin que son utilité apparaît avec le signal "d'horloge" (version synchrone).

On peut également réaliser une bascule D à partir d'une bascule JK, il suffit de poser $J = D$ et $K = \overline{D}$.

IV.1.4. Bascule T

T	Q_{n+1}	
0	Q_n	Mémoire
1	$\overline{Q}_n$	Basculement



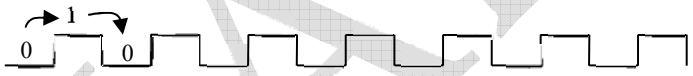
Bascule T

$$Q_{n+1} = \overline{T}Q_n + T\overline{Q_n} = T \oplus Q_n$$

IV.2. Bascules synchrones

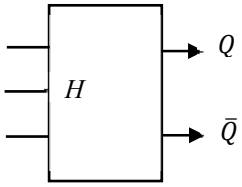
Dans la bascule synchrone, la sortie évolue si un signal d'horloge (noté H ou T) est actif. Ce dernier est une séquence de 0 et 1, il peut être actif sur un niveau (0 ou 1) ou sur un front (montant ou descendant).

Signal d'horloge :



Synchronisation sur niveau « Haut »

La bascule est active au niveau haut de l'horloge (pour H=1), et bloquée ou H=0.

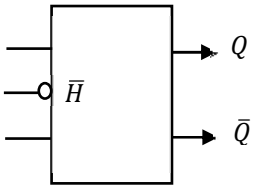


$H = 0 \Rightarrow Q_{n+1} = Q_n$ (La bascule ne change pas d'état)

$H = 1 \Rightarrow$ Fonctionnement normal.

Bascule active au niveau « Bas »

La bascule active à l'état bas de l'horloge, donc :



$H = 1 \Rightarrow Q_{n+1} = Q_n$ (La bascule ne change pas d'état)

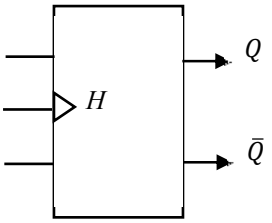
$H = 0 \Rightarrow$ Fonctionnement normal.

Bascule active sur front

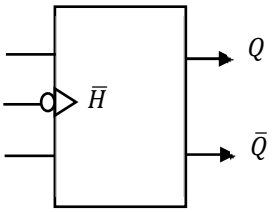
Les bascules synchrones sur front changent d'état uniquement sur un front du signal d'horloge ; en dehors de ces fronts, elle fonctionne en mémoire.

Ce mode de fonctionnement protège d'éventuels parasites sur les entrées car les entrées ne sont prises en compte que pendant la durée d'un front, qui est très courte.

Ce front peut être : montant  ou descendant 



Bascule active sur front montant



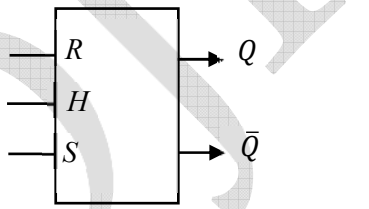
Bascule active sur front descendant

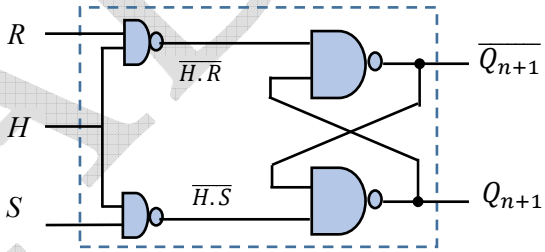
IV.2.1. Bascule RS synchrone (RST ou RSH)

On rajoute une entrée d'horloge H (validation) à la bascule.

Bascule active au niveau « Haut »

H	S	R	Q_{n+1}
0	**X**	**X**	$\boldsymbol{Q_n}$
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	X





$$Q_{n+1} = \overline{HR}Q_n + HS = \overline{\overline{HR}.Q_n.\overline{HS}}$$

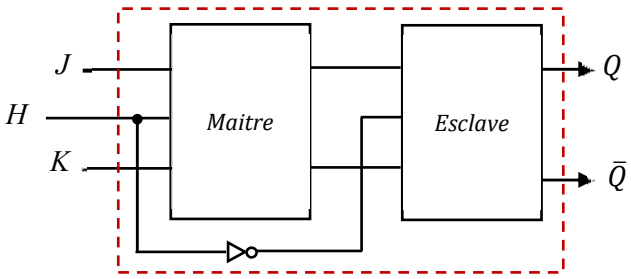
IV.2.2. Bascule JK synchrone (JKT ou JKH)

De même on peut avoir une bascule JK qui fonctionne avec un signal d'horloge.

La limitation de la JKT est quand JK=11 : La sortie oscille entre 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).

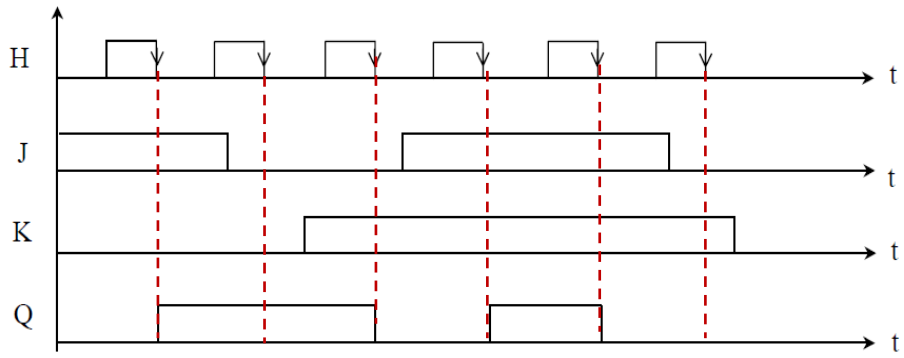
Une autre solution consiste à utiliser une bascule JK Maître-Esclave

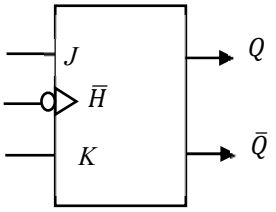
(***Etudier son fonctionnement !***)



Bascule JK Maître-Esclave

Le chronogramme suivant est un exemple de fonctionnement d'une bascule JK sur front descendant, avec au départ Q=0 :

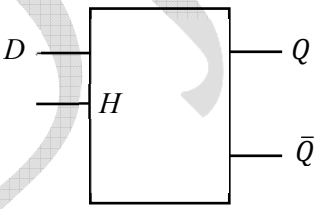




IV.2.3. Bascule D à verrou (D latch)

On a vu que la bascule D recopie l'entrée en sortie. Avec une entrée d'horloge, l'entrée D est transmise à la sortie sauf si l'horloge est active ; quand l'horloge est inactive, la bascule garde l'état précédent.

H	Q_{n+1}	
0	Q_n	Garde l'état précédent
1	D	L'entrée D est transmise à la sortie

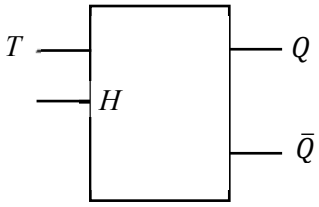


$$Q_{n+1} = \bar{H}Q_n + HD$$

IV.2.4. Bascule T synchrone

On rajoutant un signal d'horloge la sortie bascule pour T=1 et H active, son équation est :

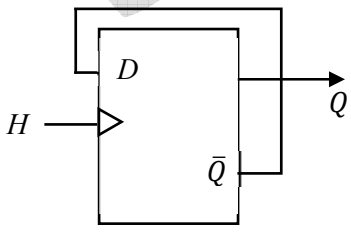
H	T	Q_{n+1}	
0	X	Q_n	Garde l'état précédent
1	0	Q_n	Garde l'état précédent
1	1	$\overline{Q_n}$	Basculement

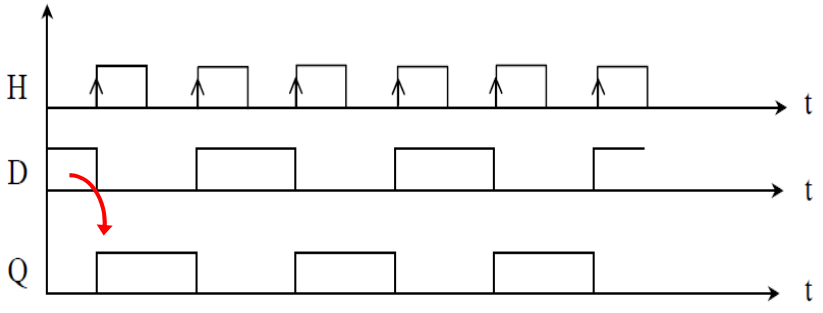


$$Q_{n+1} = (H.T) \oplus Q_n$$

IV.3. Applications

IV.3.1. Diviseur de fréquence

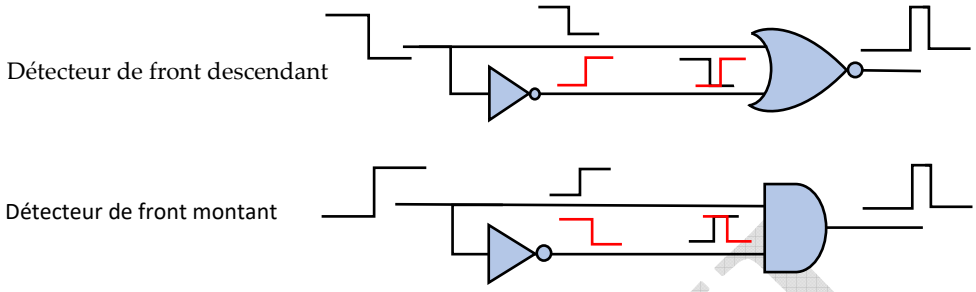




On remarque que la fréquence du signal en sortie est divisée par deux par rapport à celle de l'horloge.

IV.3.2. Détection des fronts

Le circuit suivant permet de générer une impulsion de courte durée, par exploitation du temps de propagation des portes logiques :



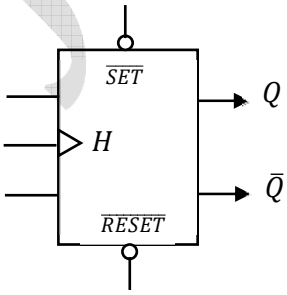
IV.4. Entrées de forçage

La plupart des bascules sont également munies d'entrées asynchrones appelées entrées de forçage, elles sont prioritaires et ne dépendant pas de l'horloge.

Il existe deux types de forçage :

Mise à 1 : SET ou PRESET ou RAU (forçage à 1) ;

Mise à 0 : RESET ou CLEAR ou RAZ (forçage à 0).



Sur le schéma des bascules, elles sont en général situées en haut et en bas du bloc fonctionnel, comme le sur la figure ci-dessus.

Elles sont généralement actives à l'état bas. Par exemple, un 0 sur l'entrée CLR provoque une remise à zéro de la bascule quelles que soient les valeurs des entrées synchrones.

➢ ***Remarque :***

Ces deux entrées asynchrones ne peuvent être actives (=0) en même temps.

IV.5. Table de transition

La table de transition donne les états dans lesquels doivent se trouver les entrées pour obtenir chacune des 4 transitions possibles de la sortie Q ($0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$, $1 \rightarrow 1$).

Quand l'état de l'entrée considérée est indifférent (0 ou 1), on le remplace par une croix.

Table de transition de la bascule JK

L'objectif est de déterminer les valeurs de J et K pour chaque changement d'état de la sortie Q.

Par exemple, la transition $0 \rightarrow 1$ (Q passe de 0 à 1) se produit par :

- Soit forcer Q à 1 (se produit pour JK=10),
- Soit en basculant l'état 0 précédent (JK=11 qui inverse l'état de la bascule)

On combine ces deux cas ($J\overline{K} + JK = J$), donc la transition 0 →1 est obtenue par mettre l'entrée J à 1 (l'état de K étant indifférent).

De la même manière, on détermine la table de transition complète :

Q_n	Q_{n+1}	J	K	
0	0	0	X	Remise à 0 ou état mémoire
0	1	1	X	Remise à 1 ou basculement
1	0	X	1	Remise à 0 ou basculement
1	1	X	0	Remise à 1 ou état mémoire

- ***Exercice :***

Déterminer la table de transition des autres bascules.

Conclusion

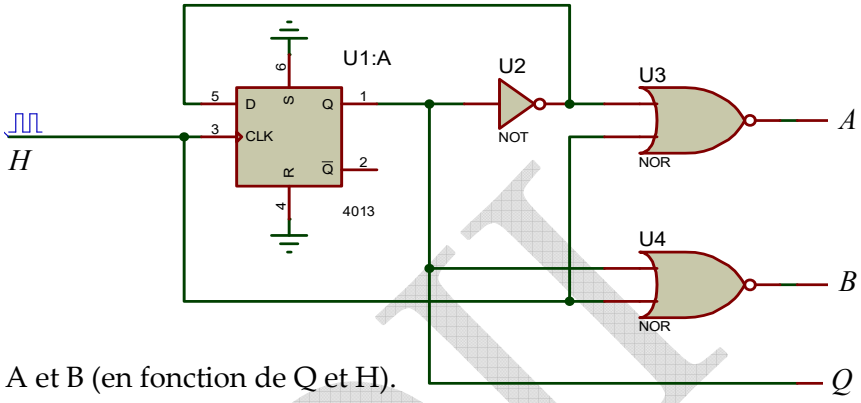
Dans ce chapitre j'ai présenté l'élément de base de la logique séquentielle qui est « la bascule », ses caractéristiques et son principe de fonctionnement (tables de vérité, logigrammes, quelques chronogrammes, exemples d'application...). On a vu que la bascule synchrone est cadencée par une entrée d'horloge et que l'on peut forcer la sortie d'une bascule par des entrées prioritaires de mise à 1 ou à 0.

J'ai aussi expliqué la table de transition d'une bascule qui est une étape très importante dans la synthèse des compteurs asynchrones (prochain chapitre).

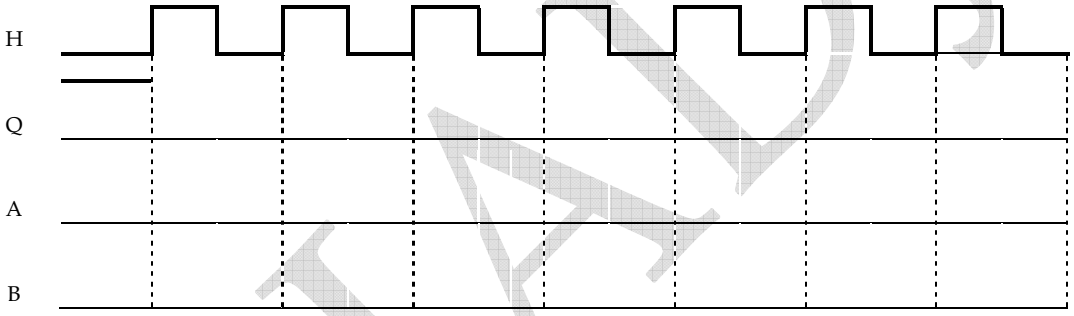
TD n° 4 : *Les bascules*

Exercice 1 :

On considère le montage de la Figure suivante réalisé avec une bascule D synchrone active sur les fronts montants du signal d'horloge.

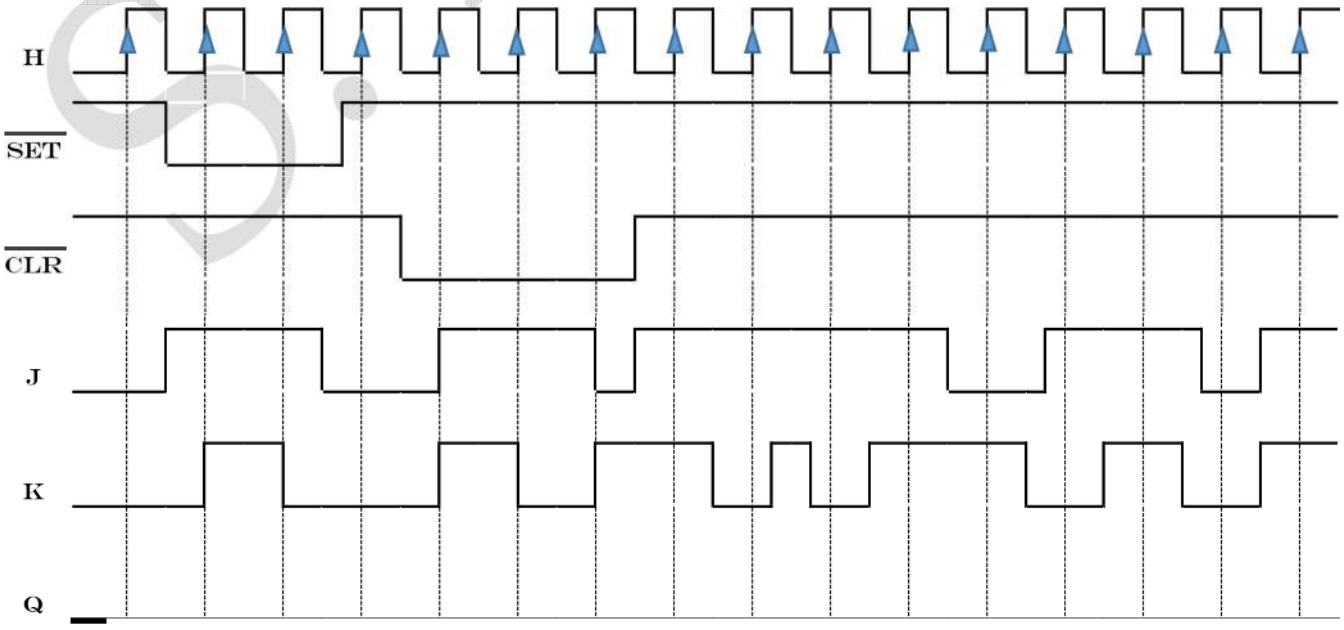


1. Donner les expressions des sorties A et B (en fonction de Q et H).
2. Compléter le chronogramme suivant :



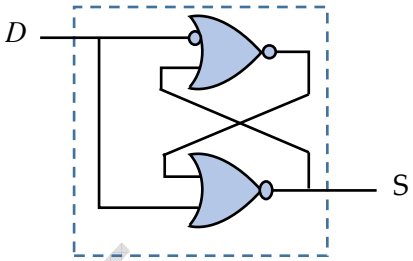
Exercice 2 :

Compléter le chronogramme d'une bascule JK fonctionnant sur front montant de l'horloge, avec des entrées de forçage actives à l'état BAS (par un 0).



Exercices supplémentaires

Exercice 1 :



1. Analyser le comportement du circuit de la figure suivante, lorsque l'entrée D est à 0 puis lorsqu'elle est à 1.
2. Que se passe-t-il si on remplace les portes NOR par des portes NAND ?

Exercice 2 :

On veut réaliser une bascule JK à partir d'une bascule T.

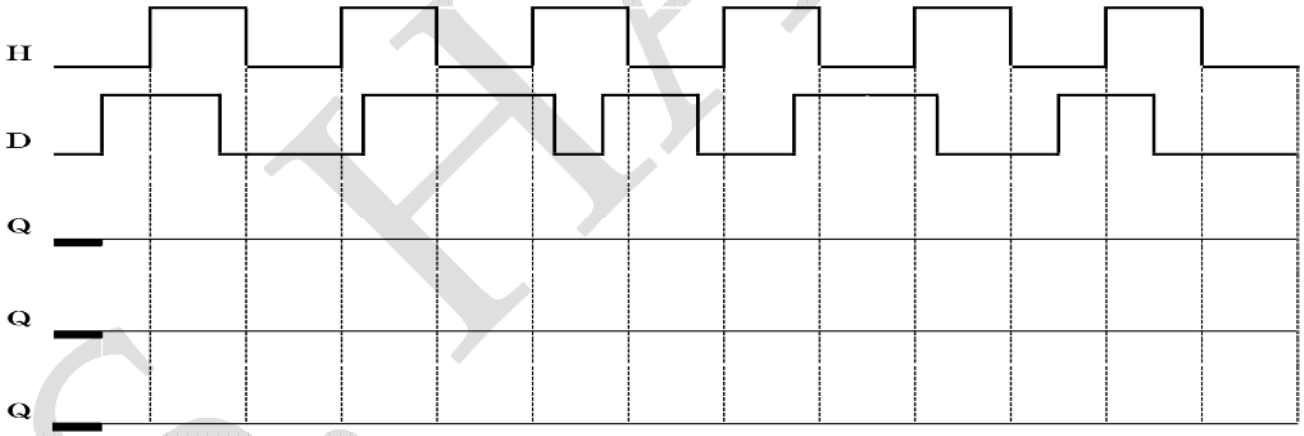
1. Remplir la table de transition ci-contre ;
2. Donner l'expression simplifiée de la fonction : $T = f(J, K, Q)$;
3. Représenter la bascule JK en utilisant la bascule T.

J	K	Q	Q^+	T
		0	0	
		1	0	
		0	1	
		1	1	

Exercice 3 :

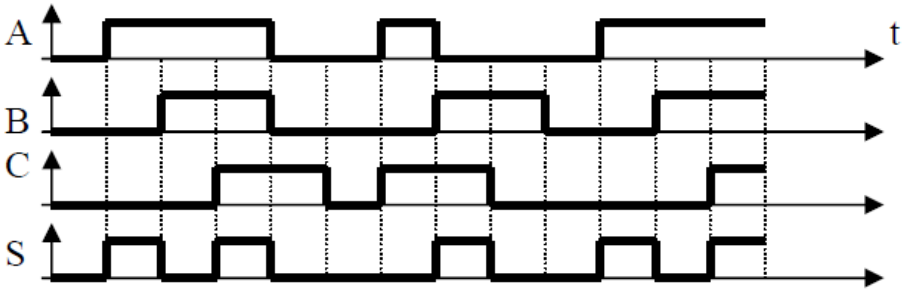
Compléter le chronogramme, de la bascule D, avec horloge (H) active au :

1. Niveau haut (=1)
2. Front montant.
3. Front descendant.



Exercice 4 :

1. Donner la table de vérité du chronogramme suivant :



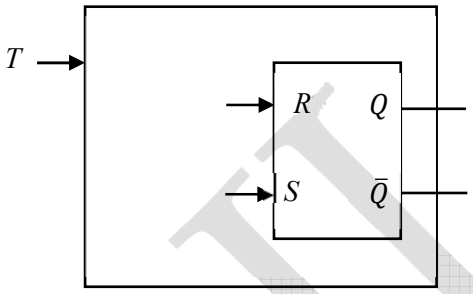
2. Le système est-il combinatoire ou séquentiel ?

Exercice 5 :

On veut réaliser une bascule T à partir d'une bascule RS.

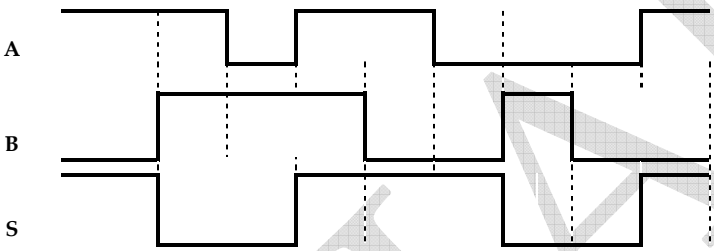
- Remplir la table de transition ci-contre et compléter la présentation.

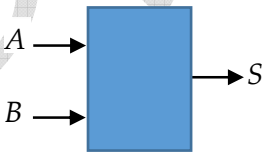
T	Q	Q^+	S	R
	0	0		
	1	0		
	0	1		
	1	1		



Exercice 6 :

Soit le système donné par le chronogramme suivant :





1. A partir du chronogramme on peut dire que ce système est séquentiel, pourquoi ?
2. Donner la table de vérité du système.
3. Déterminer l'équation simplifiée de la sortie S.

Chapitre V. Compteurs

Introduction :

Les compteurs et les registres sont des éléments essentiels de la logique séquentielle. Les compteurs sont utilisés notamment pour le comptage d'événements, pour diviser la fréquence, dans les automates programmables, etc.

Un compteur est constitué de bascules, les sorties de ces bascules donnent l'état du compteur, pour n bascules on aura 2^n états au maximum.

Les compteurs sont classés en deux catégories, les compteurs asynchrones (compteurs série) et les compteurs synchrones (compteurs parallèles).

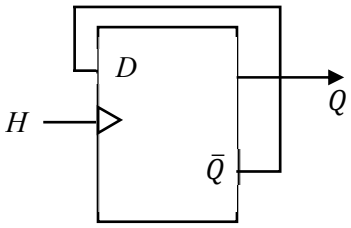
V.1. Compteurs Asynchrones

La première bascule est synchronisée par une horloge externe mais le déclenchement des autres bascules (l'entrée d'horloge) est déterminé par une combinaison logique des sorties des bascules précédentes.

La propagation de l'ordre de changement d'état se fait en cascade (la sortie de la bascule précédente active la prochaine), les sorties des bascules ne changent pas d'état exactement en même temps car elles ne sont pas reliées au même signal d'horloge.

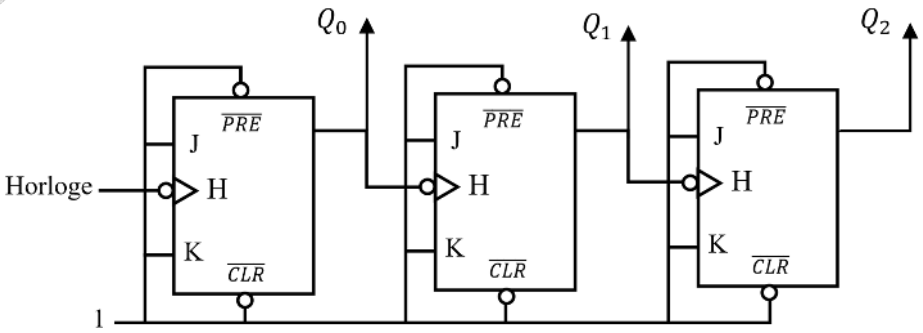
➢ ***Remarque :***

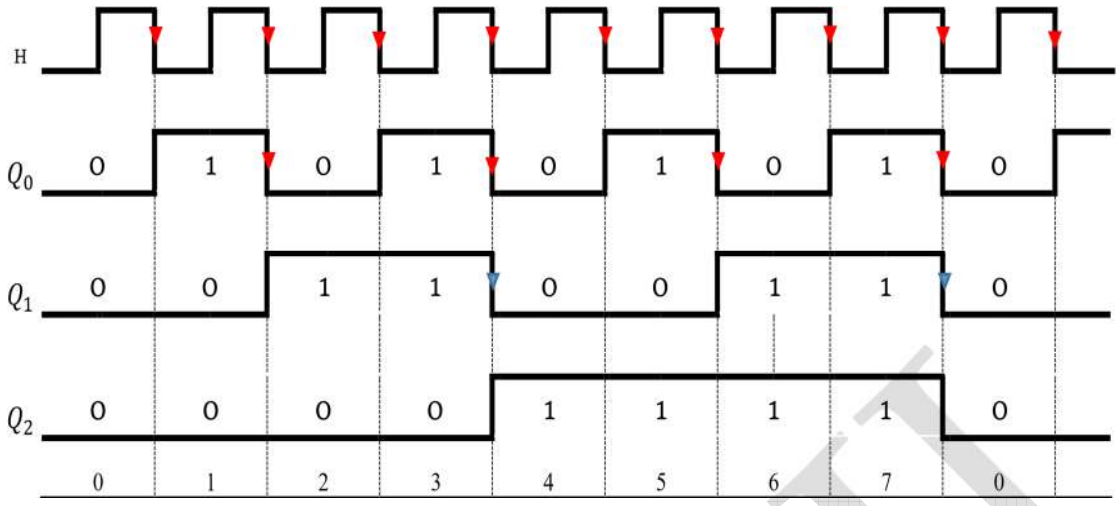
La bascule dans les compteurs asynchrones doit être en mode **BASCULEMENT** (pour la bascule JK : K=11), le schéma suivant donne la bascule D en mode basculement.



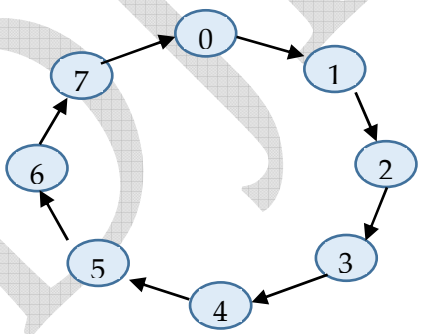
V.1.1. Compteur binaire (cycle complet)

Prenons comme exemple un compteur avec 3 bascules JK (fonctionnant au front descendant), le basculement est réalisé avec $JK = 11$. Voir le schéma :



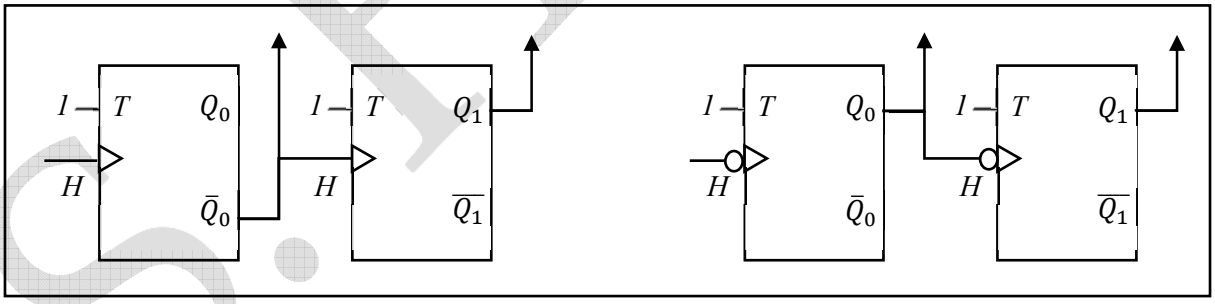


Le cycle de ce compteur est donné par la figure suivante :

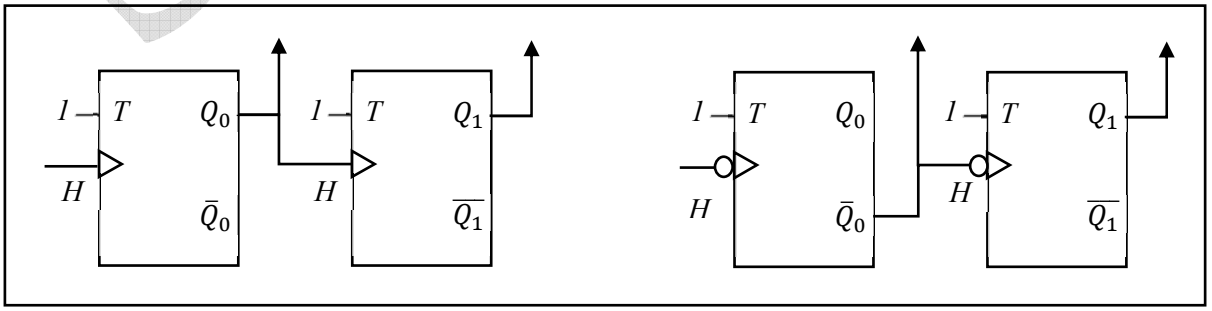


➢ ***Remarque :***

- Avec n bascules le compteur réalise, en boucle, un cycle complet de 2^n états.
- Au final, on obtient un diviseur de fréquence sur 2^n du signal d'entrée (H).
- Suivant le branchement de l'horloge, on peut réaliser un compteur ou un décompteur :



Branchement d'un compteur



Branchement d'un décompteur

V.1.2. Compteur asynchrone modulo N (cycle incomplet)

On appelle compteur modulo N, un compteur qui compte de 0 à N-1. Le compteur le plus utilisé est le compteur modulo 10 (de 0 à 9).

La réalisation d'un tel compteur passe par la réalisation d'un compteur binaire avec forçage de toutes les sorties à 0 à la détection du maximum (On utilise les entrées de forçage).

Un compteur modulo 10 comptera de 0 à 9 (de 0000 à 1001), il faut donc utiliser 4 bascules, une fois l'état 10 détecté toutes les sorties sont immédiatement remises à 0 donc on revient à l'état 0000.

Déterminer la fonction de forçage :

- La fonction de forçage f détecte l'état du compteur au moment du forçage, pour un compteur modulo 10 elle détecte le 10 ($f = 1$ si $Q_3Q_2Q_1Q_0 = 1010$).
- Le compteur sera remis à 0, une fois le 10 est détecté, donc les états supérieurs (11-12-13-14-15-16) sont impossibles.

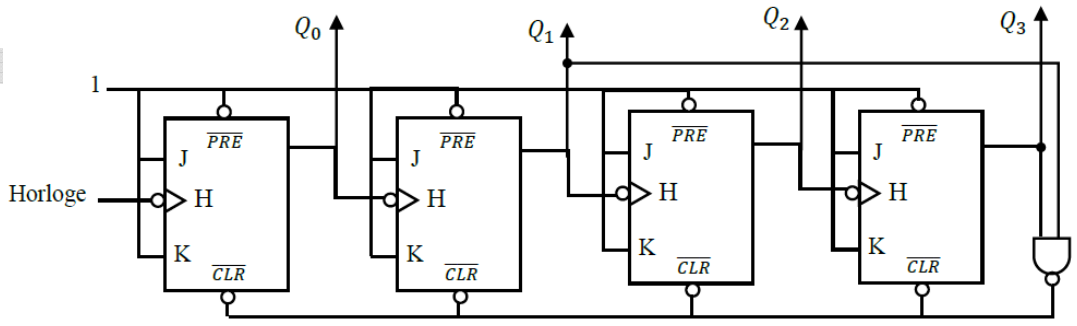
Avec la table de Karnaugh on obtient :

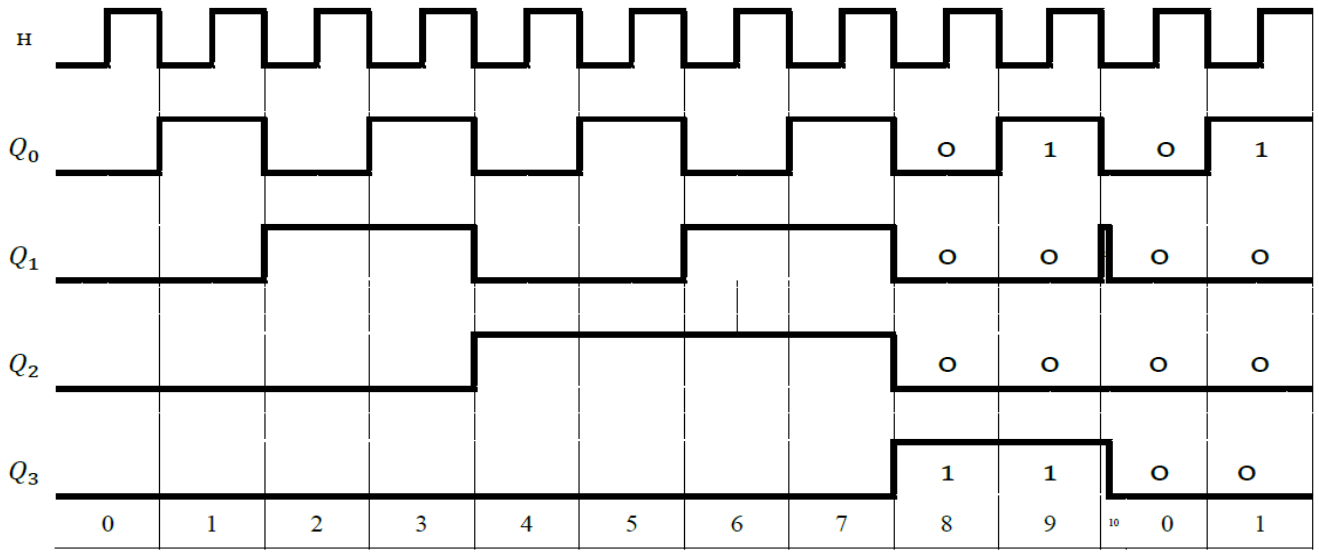
$Q_3\,Q_2$ / $Q_1\,Q_0$	00	01	11	10
00			**X**	
01			**X**	
11			**X**	**X**
10			**X**	**1**

$$f = Q_3Q_1$$

Puisque, dans notre cas, les bascules sont mises à 0 par l'état bas ($f = 0$), donc la fonction du forçage est :

$$\overline{f} = \overline{Q_3Q_1}$$





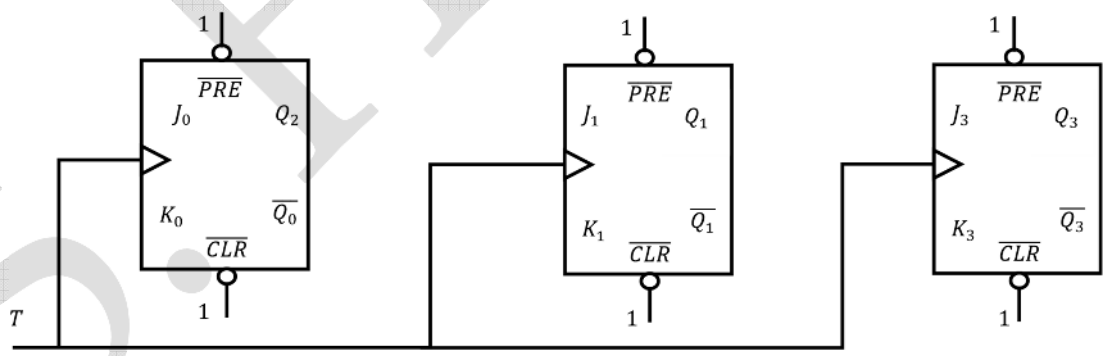
L'état 10 ($Q_3Q_2Q_1Q_0 = 1010$) est détecté pendant quelques nanosecondes (le temps de propagation du signal dans la porte NAND) et aussitôt le compteur sera réinitialisé à 0.

➢ ***Remarque :***

Le forçage peut être effectué vers n'importe quel état (forçage des bascules à 0 ou à 1).

V.2. Compteurs synchrones

Dans ce cas, toutes les bascules reçoivent le même signal d'horloge.



Dans le cas des bascules JK (par ex.), pour réaliser un compteur synchrone il faut déterminer les équations des entrées J_i et K_i en fonction des sorties Q_i, donc il faut passer par la table de transition.

➢ Ceci permettra de réaliser des séquences quelconques.

V.2.1. Compteur synchrone modulo N

C'est un compteur qui reproduit le même cycle dès qu'un état N est détecté.

➢ ***Exemple :***

Réaliser un compteur modulo 5 par la bascule D.

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

⇒ On aura besoin de 3 bascules

1. ***Réaliser la table de transition de la bascule :***

Rappel : Dans la bascule D la prochaine sortie Q^+ prend la valeur de l'entrée D.

$$\Rightarrow D = Q^+$$

2. ***Réaliser la table de transition du compteur :***

Chaque entrée D_i correspond à la transition de Q_i correspondante (par ex. la transition de Q_1 donne la valeur de D_1).

On peut représenter la table de transition du compteur par deux formes :

	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	0	0	0
5	1	0	1	X	X	X
6	1	1	0	X	X	X
7	1	1	1	X	X	X

	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	D_2	D_1	D_0
0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	0	1	0	0
4	1	0	0	0	0	0	0	0	0
5	1	0	1				X	X	X
6	1	1	0				X	X	X
7	1	1	1				X	X	X

3. ***Déterminer les D_i par la table de Karnaugh :***

Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	X	
1			X	X

$$D_0 = \overline{Q_2}.\overline{Q_0} = \overline{Q_2 + Q_0}$$

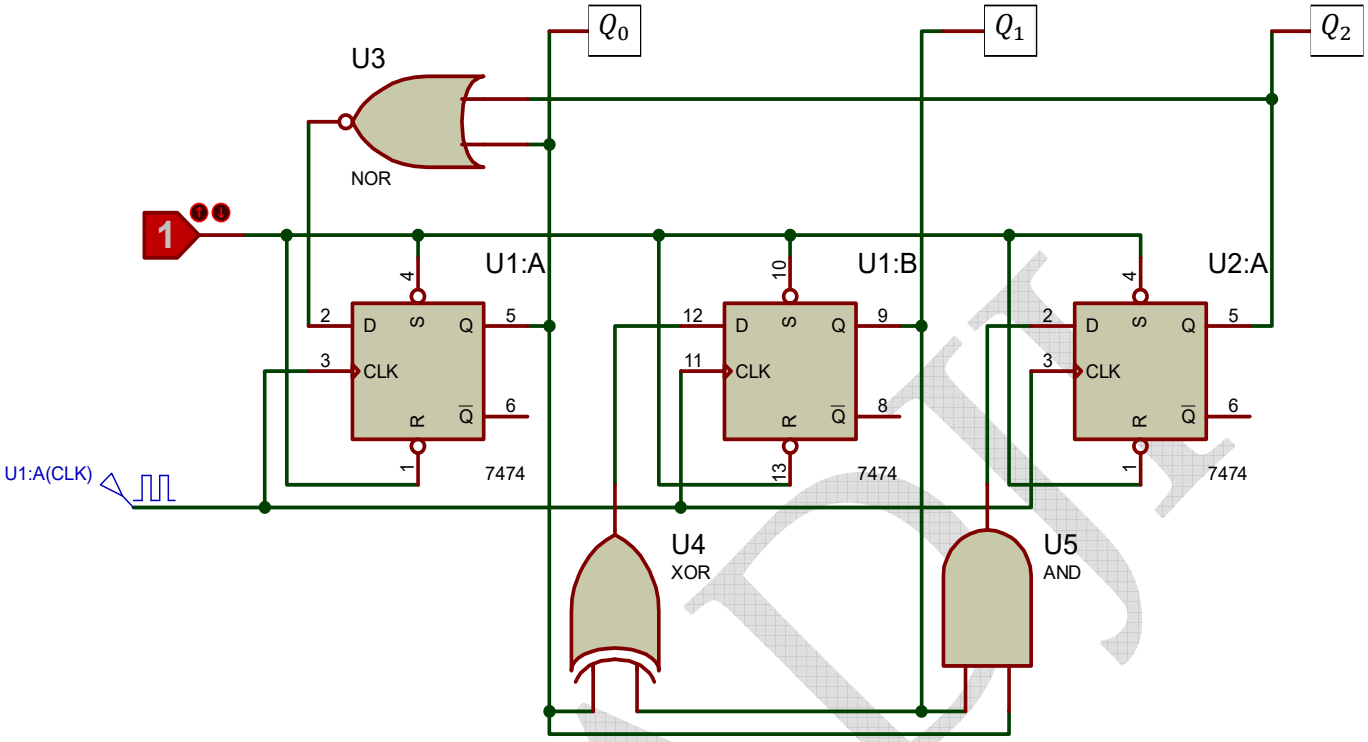
Q_0 \ Q_2Q_1	00	01	11	10
0		1	X	
1	1		X	X

$$D_1 = Q_1\overline{Q_0} + \overline{Q_1}Q_0 = Q_1 \oplus Q_0$$

Q_0 \ Q_2Q_1	00	01	11	10
0			X	
1		1	X	X

$$D_2 = Q_1Q_0$$

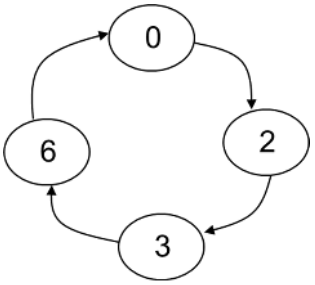
4. Le logigramme



V.2.2. Compteur à cycle quelconque :

➢ ***Exemple :***

Réaliser avec des bascules JK un compteur réalisant le cycle suivant :



1. La table de transition de la bascule JK:

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

2. La table de transition du compteur :

	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	0	X	1	X	0	X
1	0	0	1				X	X	X	X	X	X
2	0	1	0	0	1	1	0	X	X	0	1	X
3	0	1	1	1	1	0	1	X	X	0	X	1
4	1	0	0				X	X	X	X	X	X
5	1	0	1				X	X	X	X	X	X
6	1	1	0	0	0	0	X	1	X	1	0	X
7	1	1	1				X	X	X	X	X	X

3. Les entrées Ji et Ki :

Q_0 \ Q_2Q_1	00	01	11	10
0			**X**	**X**
1	**X**	**1**	**X**	**X**

$J_2 = Q_0$

Q_0 \ Q_2Q_1	00	01	11	10
0	**X**	**X**	**1**	**X**
1	**X**	**X**	**X**	**X**

$K_2 = 1$

Q_0 \ Q_2Q_1	00	01	11	10
0	**1**	**X**	**X**	**X**
1	**X**	**X**	**X**	**X**

$J_1 = 1$

Q_0 \ Q_2Q_1	00	01	11	10
0	**X**		**1**	**X**
1	**X**		**X**	**X**

$K_1 = Q_2$

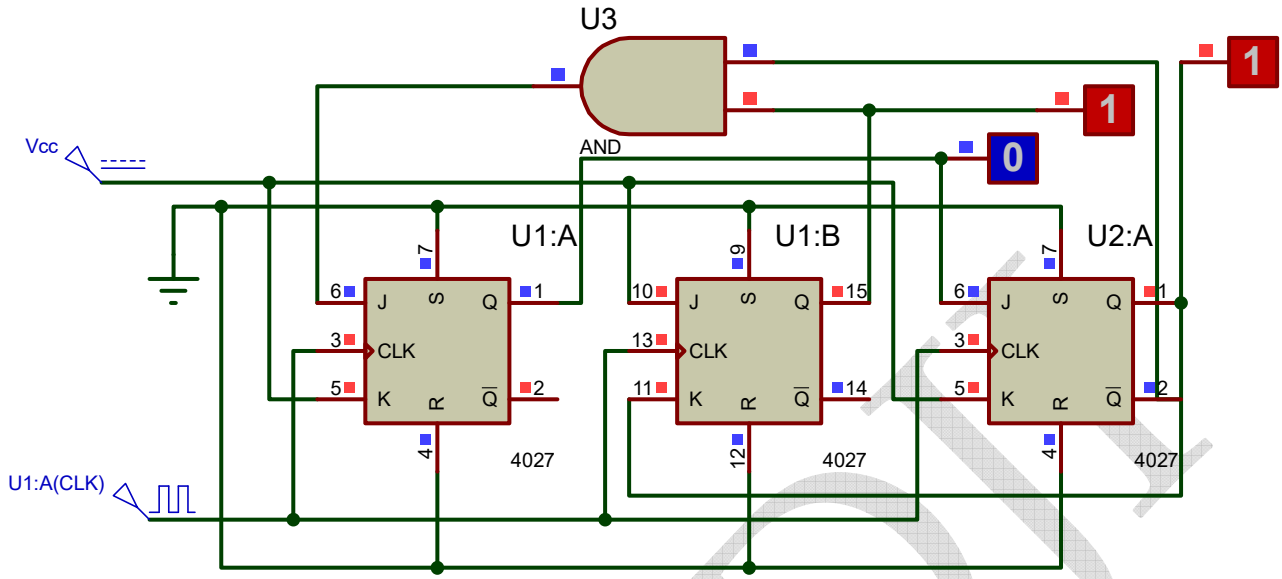
Q_0 \ Q_2Q_1	00	01	11	10
0		**1**		**X**
1	**X**	**X**	**X**	**X**

$J_0 = \overline{Q_2}Q_1$

Q_0 \ Q_2Q_1	00	01	11	10
0	**X**	**X**	**X**	**X**
1	**X**	**1**	**X**	**X**

$K_0 = 1$

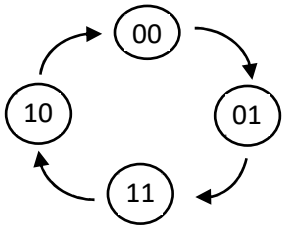
4. *Le logigramme*



TD n° 5 : *Les compteurs*

Exercice 1 :

On veut réaliser un compteur synchrone, à l'aide de bascules JK, qui réalise le cycle gray de la figure suivante :



- Donner la table de transition de la bascule JK puis la table de transition du compteur ;
- Réaliser ce compteur.

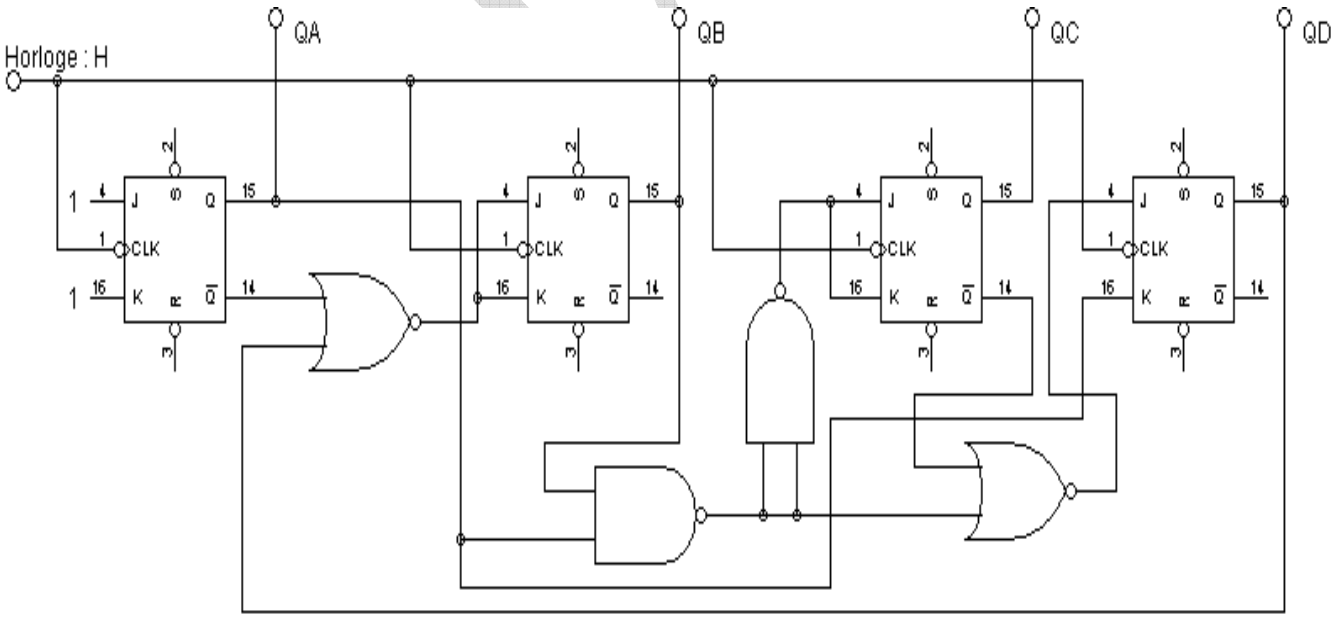
Exercice 2 :

On vous propose de réaliser un compteur asynchrone modulo 6 avec des bascules D.

- Donnez le branchement des entrées D_i de ces bascules ;
- Déterminez la fonction de remise à zéro et réalisez le compteur.

Exercice 3 :

Soit le compteur du circuit suivant :



1. Déterminer les équations des entrées J et K des bascules A, B, C et D.
2. On suppose que le compteur part de l'état : $Q_A Q_B Q_C Q_D = 0000$.
3. Donner la table de transition de ce compteur
4. Déterminez le modulo de ce compteur.

Exercices supplémentaires

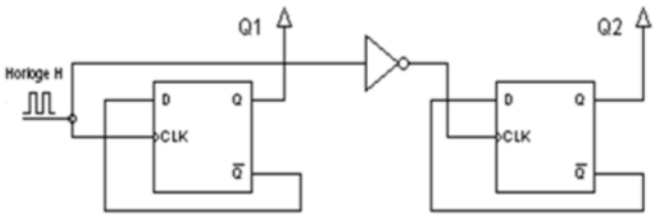
Exercice 1 :

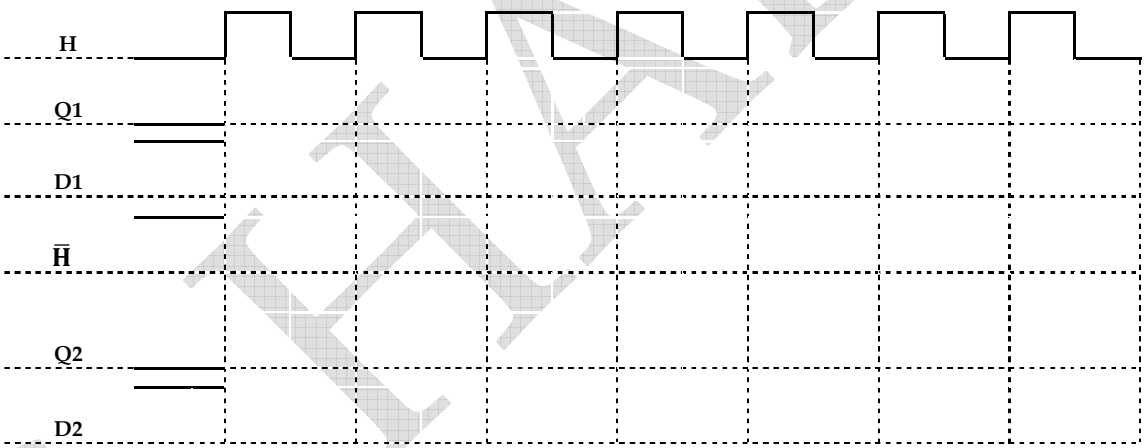
On désire réaliser un compteur/décompteur synchrone modulo 8 à bascules JK. Ce compteur comportera une entrée de commande X telle que : X=0, comptage ; X=1, décomptage.

1. Donner la table d'excitation (de transition) de la bascule JK.
2. Donner la table de transition du compteur/décompteur.
3. Réaliser le schéma.

Exercice 2 :

Compléter le chronogramme du montage ci-contre :

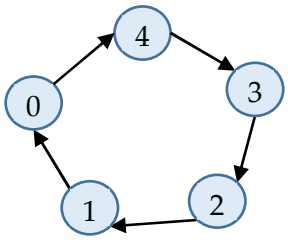




Exercice 3 :

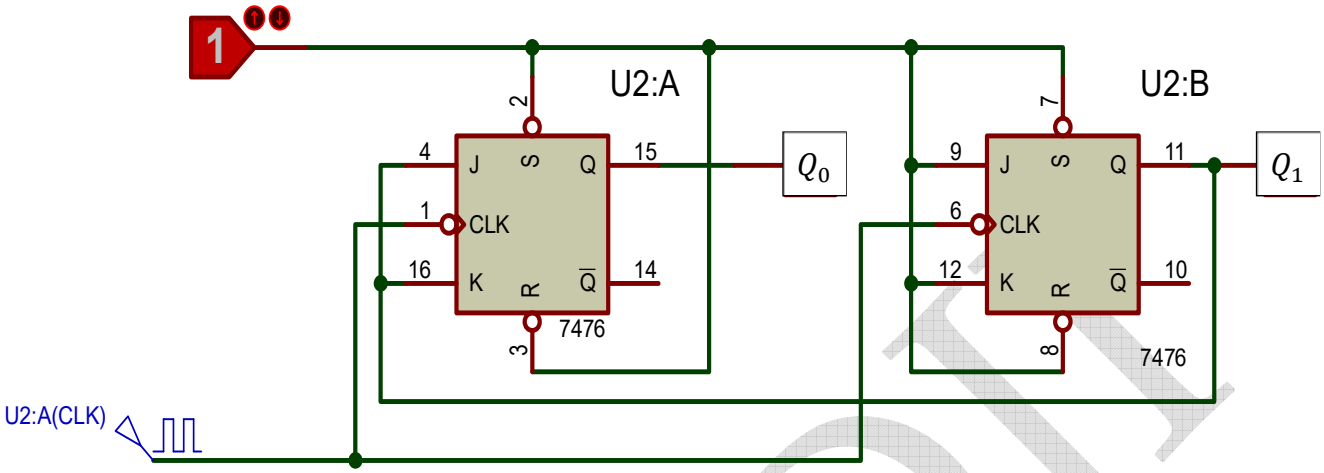
On veut réaliser un compteur synchrone avec des bascules T du cycle de la figure,

1. A partir de la table de transition de la bascule T, donner la table de transition du compteur
2. Dessiner le logigramme compteur.



Exercice 7 :

On donne et le compteur de la Figure suivante.



1. Ce compteur est-il synchrone ou asynchrone ? pourquoi ?
2. Donner la table de transition de la bascule JK et les expressions des entrées J_iK_i des bascules.
3. Dresser la table de transition du compteur.
4. Quel est le cycle de ce compteur ?

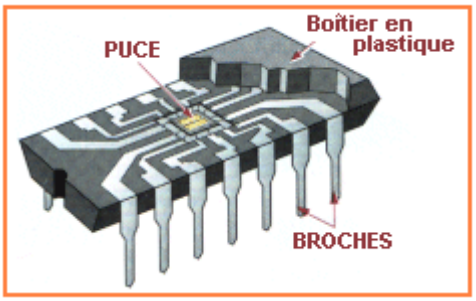
Chapitre VI. Technologie des circuits logiques intégrés

Introduction

Les circuits logiques (portes, circuits combinatoires, circuits séquentiels, etc.) se présentent sous forme d'un circuit électronique (puce) et ils sont basés sur des composants à semi-conducteurs.

Ce chapitre abordera les caractéristiques électriques de ces circuits et les différentes technologies utilisées pour leur fabrication (TTL et CMOS).

Une comparaison de ces deux technologies sera présentée avec les avantages et les inconvénients de chacune d'elle.



VI.1. Description des familles TTL et CMOS

Les circuits de la famille TTL « ***T**ransistor-**T**ransistor **L**ogic* » sont basés sur des transistors bipolaires, et les circuits de la famille CMOS « ***C**omplementary **M**etal-**O**xide **S**emiconductor* » sur des transistors à effet de champ.

VI.1.1. Famille TTL

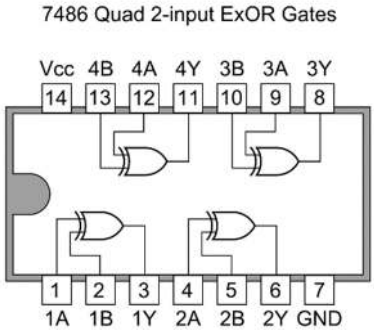
Les circuits intégrés TTL sont référencés par : ***74 xx nnn***

xx : La technologie (1 ou plusieurs caractères : L, LS, ALS, ...) ;

nnn : Le numéro de la fonction logique réalisée (2 à 4 chiffres : 00 à plus de 1000).

Exemple :

Le circuit ***7486*** comporte des portes XOR à deux entrées



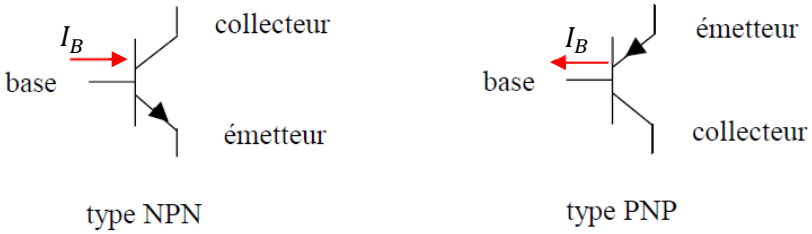
VI.1.1.1. Sous-familles TTL

Il existe d'autres sous-familles TTL plus rapides, basées sur des jonctions NP (notamment des diodes Schottky), on peut trouver les familles :

- **74L** (Low Power) : Consommation réduite ;
- **74H** (High Speed) : Consommation est plus importante mais temps de propagation plus faibles ;
- **74S** (Schottky) : 2 fois plus rapide que la série standard ;
- **74ALS** (Advanced Low Power Shottky) : La plus rapide et dont la consommation est la plus faible.

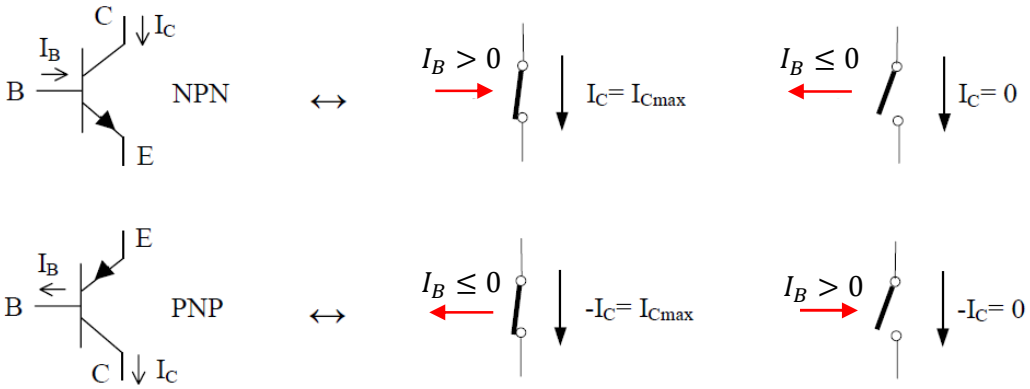
VI.1.1.2. Technologie utilisée

Les transistors bipolaires sont la base des circuits TTL standards, ils peuvent être de type NPN ou PNP. Le transistor NPN est commandé par un courant entrant par la base du transistor ; le transistor PNP est commandé par un courant sortant.



type NPN type PNP

La figure suivante illustre le mode saturation/blocage du transistor bipolaire.



VI.1.2. Famille CMOS

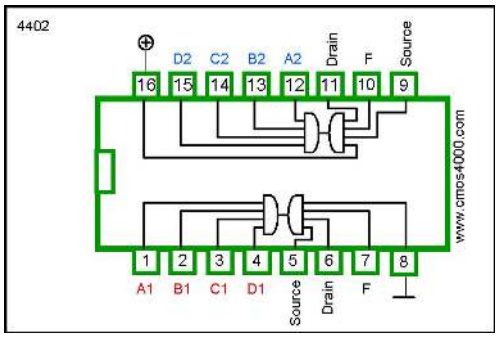
La série standard est la série **4000**. Ces composants sont référencés : ***4nnn xB***

nnn : Le numéro de la fonction logique réalisée (3 ou 4 chiffres à partir de 000) ;

xB : Le type d'étage de sortie.

Exemple :

Le circuit ***4402*** comporte des portes AND à quatre entrées.



VI.1.2.1. Sous-familles CMOS

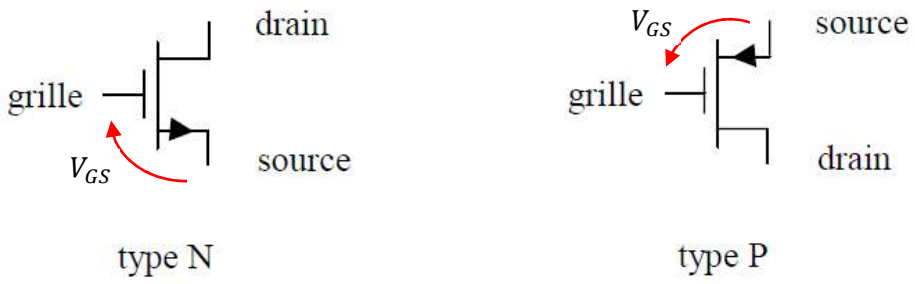
Il existe également des sous-familles CMOS dont les références commencent par 74 pour signaler le fait que le brochage des circuits intégrés est compatible avec ceux de la série TTL :

- **74HC / 74HCT / 74HCU** (Hi-Speed CMOS) : Faible consommation et haute vitesse et la capacité de commande de la famille TTL-LS ;
  - 74HCxxxx : Alimentation 2 à 6V ; consommation statique négligeable ; haute immunité au bruit ;
  - 74HCTxxxx : idem 74HC mais avec compatibilité avec la famille TTL (mêmes seuils de commutation) ;

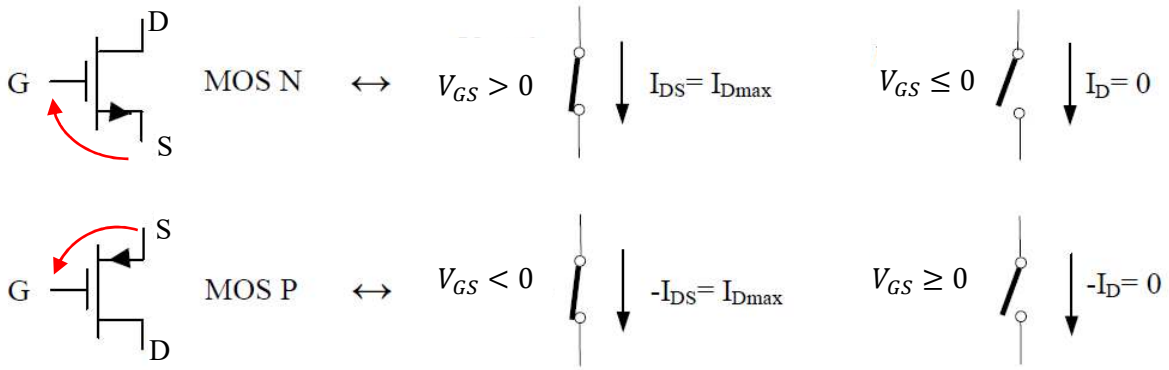
  Référence : ***74 HC nnn*** **(nnn** est la fonction logique correspondante de la famille TTL)
- **74LV** (Low Voltage LV-HCMOS) : Faible tension d'alimentation : de 1,2 à 3,6V (typiquement 3,3V). Elle est typiquement utilisée dans les systèmes portables.

VI.1.2.2. Technologie utilisée

Les transistors à effet de champ (MOS) sont la base des circuits CMOS, ils peuvent être de type N ou P. Le transistor de type N est commandé par une tension grille-source positive ou nulle ; le transistor de type P est commandé par une tension grille-source négative.

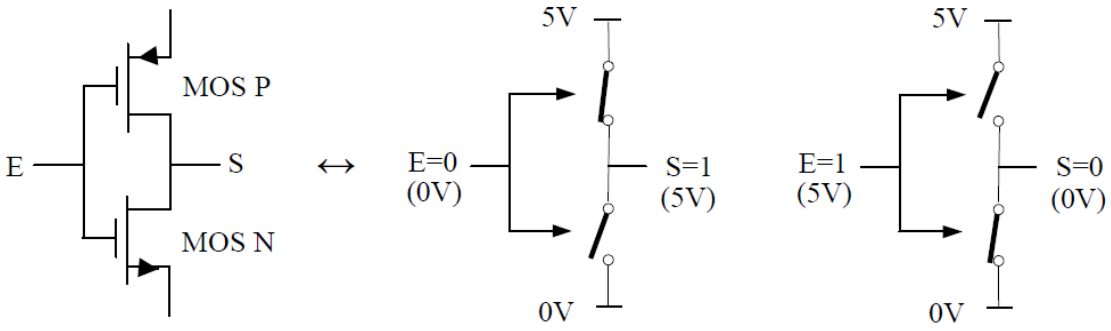


Le fonctionnement est illustré par la figure suivante.



VI.1.2.3. Réalisation d'une fonction logique élémentaire (NOT)

La fonction NOT est réalisée par le montage suivant :



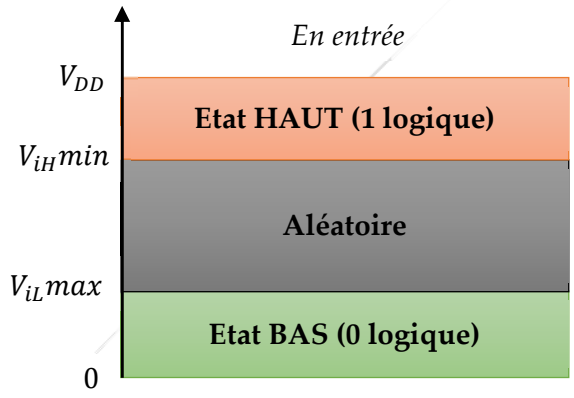
On peut remarquer que lorsque : $E = 0 \Rightarrow S = 1$ et si : $E = 1 \Rightarrow S = 0$

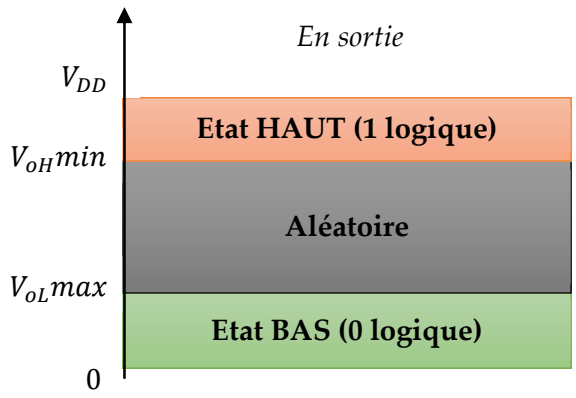
VI.2. Caractéristiques des familles TTL et CMOS

VI.2.1. Alimentation

TTL	CMOS
$V_{DD} = 5V \pm 5\%$	➢ Série 4000 : $[3 - 18]\ V$; ➢ 74HC : $[2 - 6]\ V$; ➢ 74LV : $[1.2 - 3.6]\ V$.

VI.2.2. Niveaux de tension de courant





	TTL	CMOS
Entrée	$V_{iH}min = 2\ V$ $V_{iL}max = 0.8\ V$	$V_{iH}min = 70\% . V_{DD}$ $V_{iL}max = 30\% . V_{DD}$
Sortie	$V_{oH}min = 2.4\ V$ $V_{oL}max = 0.4\ V$	$V_{oH}min = V_{DD} - 0.1\ V$ $V_{oL}max = 0.1\ V$

VI.2.3. Niveaux de tension de courant

Le courant consommé de la porte traversée et de la technologie utilisée, et aussi si le circuit est en fonctionnement (dynamique) ou juste alimenté (statique). Pour des raisons de comparaison, on donne dans le tableau suivant la puissance approximative consommée par une porte élémentaire simple avec une alimentation de 5V.

On peut remarquer la faible consommation des circuits CMOS

	TTL	CMOS
Statique	Environs 10 mW	Environs 2.5 nW
Dynamique	≈ la consommation statique	≈ 0,1 mW à 100 kHz, ≈ 1 mW à 1 MHz.

VI.2.4. Sortance

La sortance est le nombre maximal de portes pouvant être connectées en sortie d'une porte donnée, en régime dynamique.

Si la sortance est dépassée, le fonctionnement du circuit logique peut devenir incorrect.

TTL	CMOS
Des dizaines	Diminue avec la fréquence (environs 50 à 1Mhz)

VI.2.5. Temps de propagation t_p

C'est le temps moyen nécessaire pour traverser une porte logique, il diminue avec la tension d'alimentation et dépond aussi de la charge. Un exemple est donné pour une alimentation de 5V et une charge capacitive de 15pF.

TTL	CMOS
$\approx 9\ ns$	$\approx 35\ ns$

Les opérateurs CMOS standard sont donc moins rapides que ceux de la famille TTL, mais ceux de la série CMOS rapide 74HC sont du même ordre.

VI.2.6. Immunité aux bruits

L'immunité au bruit est la variation du signal d'entrée n'entraînant pas de modification de la sortie, donc :

- Au niveau bas : $V_{iL}max - V_{oL}max$
- Au niveau haut : $V_{oH}min - V_{iH}min$

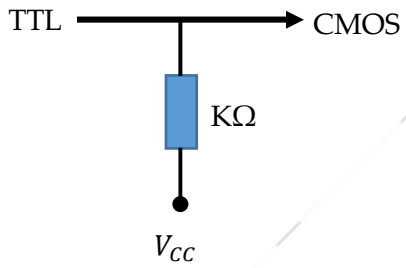
	TTL	CMOS
Niveau HAUT	$V_{iL}max - V_{oL}max = 0.8 - 0.4 = 0.4V$	$V_{iL}max - V_{oL}max = 30\% \, . V_{DD} - 0.1 \approx 30\% \, . V_{DD}$
Niveau BAS	$V_{oH}min - V_{iH}min = 2.4 - 2 = 0.4V$	$V_{oH}min - V_{iH}min = (V_{DD} - 0.1\,) - 70\% \, . V_{DD} \approx 30\% \, . V_{DD}$

Pour VDD=5V, la marge d'immunité au bruit est de 1,5V pour la famille CMOS, alors qu'elle n'est que de 0,4V pour la famille TTL.

VI.2.7. Entrées non-utilisées

TTL	CMOS
Correspond à un niveau 1	Prend une valeur aléatoire

VI.3. Association de portes des familles TTL et CMOS

TTL vers CMOS	CMOS vers TTL
	➢ De 74HC (CMOS) vers TTL

Conclusion

La famille CMOS standard est plus lente (voir temps de propagation) que la famille TTL, ce qui représente son principal inconvénient. Mais avec les séries rapides 74HC, les temps de propagation des circuits des 2 familles sont du même ordre.

Du point de vu de la consommation, la différence entre les circuits TTL et CMOS se fait sentir surtout en régime statique : on a vu que celle de la famille CMOS était quasi-nulle.

Un autre avantage de la famille CMOS sur la famille TTL est sa possibilité de grande intégration (LSI : Large Scale Integration). Cet avantage est décisif pour la miniaturisation des ordinateurs et autres systèmes numériques actuels.

Conclusion générale

Les étudiants, à travers ce cours, peuvent accueillir les notions de base de l'électronique numérique et des systèmes embarqués, savoir représenter quelques applications des circuits combinatoires en utilisant les outils standards qui sont les tables de vérité et les tables de Karnaugh, introduire les circuits séquentiels à travers les circuits bascules et les compteurs ; ce qui leur donnera le baguage académique nécessaire pour s'approfondir dans l'ère du numérique.

Ce module est fondamental pour les trois (03) filières, Automatique, Télécommunications et Electronique, ce qui illustre l'importance et le caractère fondamental des notions apportées par le contenu de cette matière, enrichies par les travaux dirigés et les travaux pratiques.

Bibliographie

1. J.C. Lafont, Cours et problèmes d'électronique numérique, 124 exercices avec solutions, Ellipses.
2. R. Delsol, Electronique numérique, Tomes 1 et 2, Edition Berti
3. P. Cabanis, Electronique digitale, Edition Dunod.
4. M. Gindre, Logique combinatoire, Edition Ediscience.
5. H. Curry, Combinatory Logic II. North-Holland, 1972
6. R. Katz, Contemporary Logic Design, 2nd ed. Prentice Hall, 2005.
7. M. Gindre, Electronique numérique : logique combinatoire et technologie, McGraw Hill, 1987
8. Brie, Logique combinatoire et séquentielle, Ellipses, 2002.
9. J-P. Ginisti, La logique combinatoire, Paris, PUF (coll. « Que sais-je? » n°3205), 1997.
10. J-L. Krivine, Lambda-calcul, types et modèles, Masson, 1990, chap. Logique combinatoire, traduction anglaise accessible sur le site de l'auteur.
11. Alexandre, Circuits numériques, Polycopié de cours électronique, Conservatoire national des arts et métiers, France, 2004.
12. Mc. Belaid et collectif, Logique combinatoire et séquentielle, Presses de Mitidja, Baraki, Alger, Algérie, 2010.
13. W. Kleitz, Digital electronics with VHDL, Pearson Education Hall, New Jersey, USA, 2006.
14. H. Lilen, Cours pratique de logique pour microprocesseurs, Ed Radio, Paris, France, 1986.
15. J. J. Mercier, Bit par bit : numération, binaire, logique combinatoire, Ellipses Editions Marketing, Paris, France, 2005.
16. J. J. Mercier, Computers 2, séquence après séquence, logique séquentielle, Ellipses Editions Marketing, Paris, France, 2006.
17. M. K. Mokhtari, I. Caid De l'algèbre de boole aux circuits numériques, cours et applications, Office des Publications Universitaires, Alger, Algérie, 2010.
18. M. Sbaï, Electronique numérique, logique combinatoire et composants numérique, Ellipses Editions Marketing, Paris, France, 2013
19. A. Tachet, Automatisme, Tome 1, logique combinatoire, Office des Publications Universitaires, Alger, Algérie, 1988.
20. Digital Systems Principles and Applications, 12th Edition by Neal S. Widmer, Gregory L. Moss, Ronald J. Tocci
21. Digital Electronics, Volume 1 Combinational Logic Circuits by Tertulien Ndjountche
22. Digital Electronics, Volume 2 Sequential and Arithmetic Logic Circuits by Tertulien Ndjountche