Licence d'Ingénierie Electrique 1ere année

Electronique numérique

Logique combinatoire et séquentielle

Luc MUSEUR

Université Paris 13, Institut Galilée.

Chapitre 1

Algèbre de Boole

En électronique numérique on manipule des variables logiques conventionnellement repérées par les valeurs 0 ou 1. Ces grandeurs obéissent à des règles d'algèbre particulières qu'il est indispensable de maîtriser avant d'entreprendre l'analyse ou la synthèse de circuits numériques. Dans ce chapitre nous énoncerons les principes et les règles de calcul de l'algèbre logique, appelé aussi algèbre de Boole, puis nous les appliquerons à l'écriture et à la manipulation des fonctions logiques.

1.1. *Variables et fonctions logiques.*

1.1.1. Variables logiques

On appelle variable logique une variable qui ne peut prendre que deux valeurs conventionnellement repérées par 0 et 1. On parle aussi de variable binaire. Chacune de ces deux valeurs est associée à une grandeur physique, par exemple la tension collecteur d'un transistor, ce qui permet de faire le lien entre une étude théorique utilisant l'algèbre de Boole et un circuit électronique. Deux cas de figure se présentent :

	Logique positive	Logique négative
0	Valeur algébrique minimum	Valeur algébrique maximum
1	Valeur algébrique maximum	Valeur algébrique minimum

Dans ce cours nous nous placerons toujours dans le cas de la logique positive si bien que

- La variable 0 sera associée à un niveau bas (typiquement une tension nulle)

- La variable 1 sera associée à un niveau haut (une tension positive de 5 V par exemple dans le cas des circuits électroniques réalisés en technologie TTL[1])

1.1.2. Fonctions logiques

Une fonction logique F des n variables logiques $(x_1, x_2 ... x_n)$, notée par exemple $F = F(x_1, x_2 ... x_n)$, associe une valeur 0 ou 1 aux différentes combinaisons possibles des n variables logiques $(x_1, x_2 ... x_n)$. Chaque variable logique x_i pouvant prendre la valeur 0 ou 1, il y a au total 2^n combinaisons possibles des variables logiques $(x_1, x_2 ... x_n)$ et on définit complètement une fonction logique en donnant sa valeur pour chacune de ces combinaisons.

Les fonctions logiques peuvent être représentées sous forme de tables, appelées **tables de vérité**, donnant la valeur de la fonction pour chaque combinaison des variables logiques. Considérons par exemple une fonction F de deux variables x et y. Il y a donc 2^2 = 4 combinaisons possibles de ces deux variables. Une table de vérité donne la valeur de F pour chacune des 2^2 combinaisons possibles de ces 2 variables. On trouve généralement 2 types de représentations comme indiqué ci-dessous.

x y	F
0 0	0
1 0	1
1 1	0
0 1	1

y \ x	**0**	**1**
0	0	1
1	0	1

F

L'écriture de la table de vérité fait partie de l'analyse d'un système donné. A l'inverse une fois la table de vérité connue, il faut pouvoir déterminer le schéma électronique permettant de réaliser cette table : c'est la phase de synthèse.

[1] Transistor Transistor Logic

1.2. *Définition d'une algèbre logique.*

Une algèbre logique se définit par l'existence de trois lois, ou fonctions logiques de base.

1.2.1. Fonctions logiques de base.

Fonction inversion NON (NOT).

Cette fonction est également appelée complément

Notation : $F = \overline{x}$

Table de vérité

x	$F = \overline{x}$
0	1
1	0

Relation caractéristique $\overline{\overline{x}} = x \quad \forall x$

Fonction OU (OR).

C'est une fonction de deux variables également appelée somme logique

Notation : $F = x + y$

Table de vérité

x y	$F = x + y$
0 0	0
1 0	1
1 1	1
0 1	1

La fonction **OU** vaut 1 si au moins une des variables vaut 1.

Relations caractéristiques :

$$x+0=x$$
$$x+1=1$$
$$x+x=x$$
$$x+\bar{x}=x$$

Fonction ET (AND).

C'est une fonction de deux variables également appelée produit logique

Notation : $F=x\bullet y$

Table de vérité

x y	$F=x\bullet y$
0 0	0
1 0	0
1 1	1
0 1	0

La fonction **ET** ne vaut 1 que si toutes les variables valent 1.

Relations caractéristiques :

$$x\bullet 0=0$$
$$x\bullet 1=x$$
$$x\bullet x=x$$
$$x\bullet \bar{x}=0$$

1.2.2. Propriétés des fonctions logiques de base.

1. Les représentations des fonctions **ET** et **OU** par les symboles • et + sont faites par analogie avec la multiplication et l'addition en algèbre ordinaire en considérant les éléments neutres. En principe aucune confusion n'est à craindre ! Nous ne

manipulerons jamais à la fois les lois **ET** et **OU** et celles de l'algèbre ordinaire. Notons également qu'il n'existe pas de « lois inverses » analogue à la soustraction ou la division en algèbre ordinaire.

2. Les fonctions **ET** et **OU** sont commutatives.

$$x \bullet y = y \bullet x$$
$$x + y = y + x$$

3. Les fonctions **ET** et **OU** sont distributives l'une par rapport à l'autre.

$x \bullet (y + z) = (x \bullet y) + (x \bullet z)$ distributivité de **ET** par rapport à **OU**

$x + (y \bullet z) = (x + y) \bullet (x + z)$ distributivité de OU par rapport à ET

Il faut s'habituer à la distributivité de **OU** par rapport à **ET** qui n'a pas d'analogue en algèbre ordinaire!

4. Dans les expressions logiques (formules ne faisant intervenir que des variables logiques et les trois lois ci-dessus) il existe un ordre de priorité qui est le suivant en décroissant :

NON, ET , OU .

Ces règles de priorité dispensent d'un certain nombre de parenthèses. Par exemple

$$x + (y + z) = (x + y) + z = x + y + z$$
$$x \bullet (y \bullet z) = (x \bullet y) \bullet z = x \bullet y \bullet z$$
$$x + (y \bullet z) = x + y \bullet z$$
$$\neq (x + y) \bullet z$$

1.2.3. Théorème de De Morgan

C'est une des propriétés les plus importantes des fonctions logiques. Elle repose sur la remarque suivante :

Les relations caractéristiques des lois **ET** et **OU** sont invariantes dans leur ensemble lors de la transformation $+ \to \bullet,\ \bullet \to +,\ x \to \bar{x},\ \bar{x} \to x$. Partons par exemple des relations constitutives de la loi **ET**

$0 \bullet x = 0$, $1 \bullet x = x$ quelque soit x

Elles se transforment en

$$1+\bar{x}=1\ ,\ 0+\bar{x}=\bar{x}\ \text{ quelque soit } x$$

qui ne sont rien d'autre que les relations caractéristiques de la loi **OU**.

Le théorème de De Morgan est symbolisé par :

$$\overline{x \bullet y}=\bar{x}+\bar{y}$$

mais il est très général et porte sur toutes relations. Ainsi le complément d'une fonction logique sera obtenu en remplaçant les variables par leur complément, les signes + par des • et les signes • par des +. Ainsi

$$F = f(\ldots x_i\ \ldots\ ,\ +\ ,\ \bullet) \text{ entraine } \bar{F} = f(\ldots \bar{x_i}\ \ldots\ ,\ \bullet,\ +\ )$$

1.2.4. Quelques relations utiles.

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

$$x + x \bullet y = x$$
$$x + \bar{x} \bullet y = x + y$$
$$x \bullet (x + y) = x$$
$$x \bullet (\bar{x} + y) = x \bullet y$$
$$x \bullet A + \bar{x} \bullet B + A \bullet B = x \bullet A + \bar{x} \bullet B$$
$$(x + A) \bullet (\bar{x} + B) \bullet (A + B) = (x + A) \bullet (\bar{x} + B)$$

Ces relations permettent, avec un peu de pratique, de simplifier l'écriture des fonctions logiques. Les deux dernières relations sont connues sous le nom de **relations du consensus**.

1.2.5. Formes canoniques des expressions logiques.

Une expression logique F peut s'écrire sous un grand nombre de formes différentes. Deux d'entre elles, dites **formes canoniques,** sont particulièrement utiles.

1. *F* = somme de produits : $F = x \bullet y + y \bullet B \bullet A + x \bullet B$

2. F' = produit de sommes : $F' = (x + A) \bullet (B + A + x) \bullet (B + y)$

La recherche d'une forme canonique correspond en fait à la première étape de simplification d'une fonction logique. Cela peut se faire, soit en utilisant les règles de

l'algèbre de Boole, soit directement à partir de la table de vérité de la fonction. Considérons par exemple la fonction $F = (y + x) \bullet (\overline{y \bullet x})$, son expression sous forme canonique peut s'obtenir de deux façons.

- A partir des règles d'algèbre :

$$F = (y + x) \bullet (y \bullet x) = (y + x) \bullet (y + x) = \text{produit de sommes}$$
$$= y \bullet y + y \bullet x + x \bullet y + x \bullet x$$
$$= y \bullet x + x \bullet y = \text{somme de produits}$$

- A partir de la table de vérité de la fonction logique.

On commence par dresser la table de vérité en calculant la valeur de la fonction F pour les 4 combinaisons possibles des variables x et y.

x y	F
0 0	0
1 0	1
0 1	1
1 1	0

En utilisant les opérateurs logiques NON, ET et OU, on écrit ensuite les différentes combinaisons des variables d'entrées pour lesquelles F = 1. On obtient ainsi

$$F = \underbrace{y \bullet \overline{x}}_{ligne\,2} + \underbrace{x \bullet \overline{y}}_{ligne\,3} = \text{ somme de produits}$$

1.3. *Simplification des fonctions logiques.*

Nous venons de voir que toute fonction logique peut être associée à deux expressions logiques (correspondant aux deux formes canoniques). On entend en général par simplification la réduction de ces expressions à un minimum de termes contenant chacun un minimum de variables.

1.3.1. Généralités

La base des opérations de simplification réside dans les identités (où A et B sont des expressions quelconques)

$$A \bullet x + A \bullet \bar{x} = A \quad \text{et} \quad (A + x) \bullet (A + \bar{x}) = A$$

Il faut leur ajouter les expressions tirées de la règle du consensus

$$x \bullet A + x \bullet B + A \bullet B = x \bullet A + x \bullet B \quad \text{et} \quad (x + A) \bullet (x + B) \bullet (A + B) = (x + A) \bullet (x + B)$$

Avec un peu d'habitude un examen attentif des expressions logiques suffit à dégager les simplifications. Le point de départ le plus commode est la première forme canonique car les simplifications y paraissent plus familières compte tenu des analogies avec la distributivité de la multiplication par rapport à l'addition.

Pour les fonctions de quatre ou cinq variables au plus, il existe cependant « une méthode graphique » permettant d'obtenir simplement la forme la plus simplifiée de la fonction logique.

1.3.2. Simplification d'une fonction logique par la méthode des tables de Karnaugh

Commençons par définir ce que nous appelons une table de Karnaugh.

a. Tables de Karnaugh

Il s'agit d'un tableau à double entrée dans lequel chaque combinaison des variables d'entrée est associée à une case qui contient la valeur de la fonction. Ce sont donc des tables de vérité ! Cependant, la disposition des cases est telle que deux cases contiguës correspondent à des combinaisons adjacentes des variables d'entrée, c'est à dire des combinaisons ne différant que par la complémentation d'une seule variable.

On donne ci-dessous, pour la fonction $F = x \bullet y + z \bullet t + x \bullet t + z \bullet y$, deux tables de vérité dont seule celle de gauche est une table de Karnaugh. En effet sur la table de droite les variables x et y changent de valeur lorsque l'on passe de la 2eme à la 3eme colonne.

x / y / z t	**0** **0**	**0** **1**	**1** **1**	**1** **0**
0 **0**	0	1	1	0
0 **0**	0	1	1	0
1 **1**	1	1	1	1
1 **0**	0	1	1	0

x / y / z t	**0** **0**	**0** **1**	**1** **0**	**1** **1**
0 **0**	0	1	0	1
0 **0**	0	1	0	1
1 **1**	1	1	1	1
1 **0**	0	1	0	1

Bien que cela ne soit pas évident à priori, il faut se rendre compte que, sur une ligne donnée, les cases de la première et de la quatrième colonnes correspondent à des combinaisons adjacentes ne différant que par la valeur de variable x (0 dans la première colonne et 1 dans la quatrième). Il faut donc s'imaginer la table de Karnaugh comme enroulée sur elle-même de sorte que les colonnes 1 et 4 se touchent. Le même raisonnement tient aussi pour les lignes 1 et 4 puisque, sur une colonne donnée, les combinaisons de ces deux lignes ne diffèrent que par la valeur de la variable z (1 sur la ligne du bas et 0 sur celle du haut). La table de Karnaugh doit donc également se concevoir comme étant enroulé sur elle-même de bas en haut.

b. Simplification des fonctions logiques.

La méthode de simplification utilisant les tables de Karnaugh permet d'obtenir les fonctions logiques sous leur première forme canonique la plus simple possible. Elle repose sur la remarque suivante : **Deux combinaisons adjacentes de termes dans l'expression de la fonction *F*, par exemple $x \bullet y \bullet z$ et $x \bullet y \bullet \overline{z}$ correspondront toujours à deux 1 contigus dans la table de Karnaugh.** On peut alors effectuer la simplification

$$x \bullet y \bullet z + x \bullet y \bullet \overline{z} = x \bullet y$$

L'idée est donc de repérer les 1 contigus dans la table de Karnaugh et d'effectuer la simplification correspondante. On procède en trois étapes :

1. On regroupe les cases contiguës contenant des 1 en rectangles (ou carrés) de 1, 2, 4 ou 8 éléments les plus grands et les moins nombreux possible. La même case peut servir dans plusieurs regroupements puisque

$$A \bullet x \bullet y + A \bullet \overline{x} \bullet y + A \bullet x \bullet \overline{y} = (A \bullet x \bullet y + A \bullet \overline{x} \bullet y) + (A \bullet x \bullet y + A \bullet x \bullet \overline{y}) = A \bullet y + A \bullet x$$

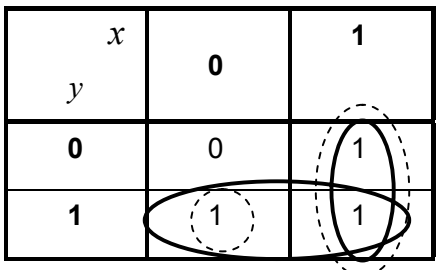
2. On traduit les regroupements obtenus en produits logiques, c'est à dire que l'on écrit la combinaison correspondante des variables d'entrée.

3. On fait la somme logique des différents produits obtenus à l'étape précédente.

Dans la pratique il y a intérêt à faire porter les premiers regroupements sur les 1 les plus isolés sous peine d'effectuer des regroupements non indispensables.

c. Exemples.

1. Simplifions la fonction F_1 définie par le tableau suivant:



F_1

Sans aucune simplification la fonction F_1 s'écrit $F_1 = \bar{x} \bullet y + x \bullet y + x \bullet \bar{y}$. Deux simplifications sont possibles qui correspondent aux deux regroupements en traits pleins représentés sur la table

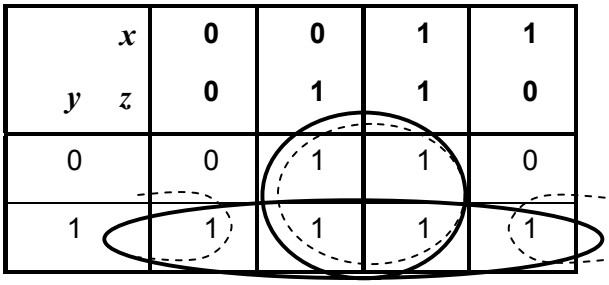
$$F_1 = \underbrace{\bar{x} \bullet y + x \bullet y}_{\substack{regroupement \\ horizontal = y}} + \underbrace{x \bullet y + x \bullet \bar{y}}_{\substack{regroupement \\ vetical = x}}$$

$$F_1 = y + x$$

Un mauvais choix des regroupements comme par exemple celui représenté en pointillés sur la table ci dessus, aboutit à une expression plus compliquée pour la fonction F_1, en l'occurrence : $F_1 = \bar{x} \bullet y + x$.

Avec un peu de pratique on écrit directement la fonction simplifiée sans faire figurer explicitement les simplifications réalisées.

2. Simplifions la fonction F_2 définie par :



F_2

Les regroupements en traits pleins conduisent à l'expression

$$F_2 = \underbrace{z}_{\substack{carré \\ central}} + \underbrace{y}_{\substack{ligne \\ du\ bas}}$$

Là encore, un mauvais choix des regroupements, comme celui en pointillés, aboutit à une expression de F_2 qui n'est pas la plus simplifiée

$$F_2 = z + y \bullet \bar{z}$$

3. Soit la fonction F_3 définie par :

	x	**0**	**0**	**1**	**1**
z	t y	**0**	**1**	**1**	**0**
0	**0**	1	0	0	1
0	**1**	0	0	1	0
1	**1**	0	1	1	0
1	**0**	1	1	1	1

F_3

La simplification n'est pas plus compliquée que dans les exemples précédents si l'on pense à regrouper ensemble les 1 figurants dans les quatre coins de la table. Ainsi :

$$F_3 = \underbrace{y \bullet z}_{\substack{carré \\ central}} + \underbrace{\bar{y} \bullet \bar{t}}_{4\ coins} + \underbrace{x \bullet y \bullet t}_{\substack{petit \\ regroupement \\ vertical}}$$

4. Simplifions la fonction F_4 définie par la table suivante

	x	**0**	**0**	**1**	**1**
z	t y	**0**	**1**	**1**	**0**
0	**0**	1	1	0	0
0	**1**	1	1	0	0
1	**1**	1	1	0	0
1	**0**	1	1	1	1

F_4

Deux regroupements sont nécessaires pour obtenir l'expression simplifiée :

$$F_4 = \underbrace{\bar{x}}_{\substack{rectangle \\ vertical}} + \underbrace{z \bullet \bar{t}}_{\substack{ligne \\ du\ bas}}$$

5. Il peut arriver que la valeur de la fonction logique ne soit pas définie pour certaines combinaisons des variables logiques. Cela signifie que pour ces combinaisons la valeur de la fonction est indifférente (souvent parce que ces combinaisons ne se produisent jamais ou qu'elles ne sont pas jugées pertinentes). C'est le cas par exemple de la fonction F_5 définie par la table de vérité ci-contre.

La fonction F_5 n'est définie que pour 10 des 16 combinaisons possibles des 4 variables logiques x, y, z et t. Pour les 6 autres combinaisons la valeur de F_5 est sans importance et peut donc être choisie librement. La table de Karnaugh correspondante est écrite ci-dessous.

x	y	z	t	F_5
0	0	0	0	1
0	0	1	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
0	1	1	0	0
1	1	0	0	0
1	1	0	1	0
1	0	0	1	0
1	0	1	1	1

Les six cases correspondant aux combinaisons pour lesquelles la fonction F_5 n'est pas définie sont remplies par des φ. Pour ces six combinaisons la valeur de la fonction peut être choisie librement (φ = 0 ou 1).

	x	**0**	**0**	**1**	**1**
z	**t**	**y**	**0**	**1**	**1**	**0**
0	**0**		1	φ	0	φ
0	**1**		φ	1	0	0
1	**1**		1	1	φ	1
1	**0**		0	0	φ	φ

F_5

Lors de la simplification, **on choisit pour chaque φ** la valeur 0 ou 1 afin d'obtenir la fonction la plus simple possible. Dans cet exemple, trois φ sont intégrés dans des regroupements et prennent donc la valeur 1, alors que les trois autres prennent la valeur 0. La fonction simplifiée s'écrit ainsi à partir de deux regroupements :

$$F_5 = \bar{x} \bullet \bar{t} + z \bullet t$$

Les fonctions incomplètement spécifiées laissent donc certains degrés de liberté pour effectuer les regroupements. On doit veiller à ne pas tomber dans les situations extrêmes qui consisteraient à imposer les valeurs 1 ou 0 pour tous les φ. Le choix doit s'effectuer au cas par cas en fonction de la possibilité de regroupements les plus gros et les moins nombreux possible.

1.3.3. Conclusion

La méthode des tables de Karnaugh est très efficace pour simplifier les fonctions logiques et ne pose aucun problème si elle est appliquée correctement en respectant les règles suivantes :

1. Il faut regrouper tous les 1 par groupes de 2, 4, 8 ou 16 (!) les plus gros et les

moins nombreux possibles. Chaque case avec 1 peut appartenir à plusieurs regroupements différents

2. Il est généralement astucieux de commencer par les 1 les plus isolés.
3. Il peut parfois être plus rapide de calculer $\overline{F}$ en regroupant les 0 puis de revenir à F en complémentant le résultat.
4. Lorsque la valeur de la fonction logique n'est pas précisée pour certaines combinaisons des variables logiques, on écrit un φ dans les cases correspondantes de la table de Karnaugh. Au moment de la simplification, on donne à chaque φ la valeur 0 ou 1 de façon à simplifier au maximum la fonction

1.4. *Exercices.*

Exercice 1

Démontrer les relations suivantes en utilisant les règles de calculs de l'algèbre de Boole

$$x + x \bullet y = x$$
$$x + \bar{x} \bullet y = x + y$$
$$x \bullet (x + y) = x$$
$$x \bullet (\bar{x} + y) = x \bullet y$$
$$x \bullet A + \bar{x} \bullet B + A \bullet B = x \bullet A + \bar{x} \bullet B$$
$$(x + A) \bullet (\bar{x} + B) \bullet (A + B) = (x + A) \bullet (\bar{x} + B)$$

Exercice 2 Ecrire les expressions logiques simplifiées des fonctions F et G définies par les tables de vérités suivantes :

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

x	y	z	G
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Exercice 3 Simplifier les fonctions logiques suivantes en utilisant la méthode des tables de Karnaugh:

$$f_1 = a \bullet \overline{b} + a \bullet b \bullet \overline{d} + a \bullet b \bullet \overline{c} \bullet d$$

$$f_2 = \overline{a} \bullet b \bullet \overline{c} + \overline{a} \bullet \overline{c} \bullet d + \overline{a} \bullet \overline{b} \bullet \overline{d} + a \bullet c + b \bullet c \bullet \overline{d}$$

$$f_3 = a \bullet b \bullet \overline{c} + \overline{c} \bullet d \bullet a + a \bullet \overline{b} \bullet c \bullet d$$

1.5. *Correction des exercices*

Exercice 1 :

L'exercice se résout en utilisant les propriétés des lois ET et OU (paragraphes 1.2.1 et 1.2.2)

a) $x + x \bullet y = x \bullet (1 + y) = x$

b) $x + \overline{x} \bullet y = (x + \overline{x}) \bullet (x + y) = x + y$ en utilisant la propriété de distributivité du OU par rapport au ET

c) $x \bullet (x + y) = x \bullet x + x \bullet y = x + x \bullet y = x \bullet (1 + y) = x$

d) $x \bullet (\overline{x} + y) = x \bullet \overline{x} + x \bullet y = x \bullet y$

d) Il s'agit de la première relation du consensus

$$\begin{aligned} x \bullet A + \overline{x} \bullet B + A \bullet B &= x \bullet A + \overline{x} \bullet B + A \bullet B \bullet (x + \overline{x}) \\ &= x \bullet A + \overline{x} \bullet B + A \bullet B \bullet x + A \bullet B \bullet \overline{x} \\ &= x \bullet A(1 + B) + \overline{x} \bullet B(1 + A) \\ &= x \bullet A + \overline{x} \bullet B \end{aligned}$$

e) On applique le théorème de De Morgan à la relation précédente en effectuant la transformation $x \to \overline{x},\ \overline{x} \to x,\ A \to \overline{A},\ B \to \overline{B},\ + \to \bullet\ et\ \bullet \to +$ ce qui donne :

$$(\overline{x} + \overline{A}) \bullet (x + \overline{B}) \bullet (\overline{A} + \overline{B}) = (\overline{x} + \overline{A}) \bullet (x + \overline{B})$$

On retrouve bien l'expression cherchée. Pour s'en convaincre il suffit de réécrire cette expression avec le changement de variables $x \to \overline{y},\ \overline{x} \to y,\ \overline{A} \to C, \overline{B} \to D$ on obtient alors

$$(y + C) \bullet (\overline{y} + D) \bullet (C + D) = (y + C) \bullet (\overline{y} + D)$$

Exercice 2 :

Pour obtenir l'expression algébrique d'une fonction logique à partir de sa table de vérité il suffit de faire la somme logique des différentes combinaisons des variables pour lesquelles la fonction vaut 1.

a) La fonction F =1 si $x=0$ ET $y=0$ ET $z=1$ OU $x=0$ ET $y=1$ ET $z=0$ OU $x=1$ ET $y=0$ ET $z=1$ OU $x=1$ ET $y=1$ ET $z=1$. Ce qui se traduit mathématiquement par :

$$F=\bar{x}\bullet\bar{y}\bullet z+\bar{x}\bullet y\bullet\bar{z}+x\bullet\bar{y}\bullet z+x\bullet y\bullet z$$

On peut simplifier cette expression pour obtenir

$$\begin{aligned}F&=\bar{x}\bullet\bar{y}\bullet z+\bar{x}\bullet y\bullet\bar{z}+x\bullet\bar{y}\bullet z+x\bullet y\bullet z\\&=\bar{y}\bullet z\bullet(\bar{x}+x)+\bar{x}\bullet y\bullet\bar{z}+x\bullet y\bullet z\\&=\bar{y}\bullet z+\bar{x}\bullet y\bullet\bar{z}+x\bullet y\bullet z\end{aligned}$$

b) La fonction G =1 si $x=0$ ET $y=0$ ET $z=0$ OU $x=0$ ET $y=0$ ET $z=1$ OU $x=1$ ET $y=0$ ET $z=0$ OU $x=1$ ET $y=0$ ET $z=1$. Ce qui se traduit mathématiquement par :

$$G=\bar{x}\bullet\bar{y}\bullet\bar{z}+\bar{x}\bullet\bar{y}\bullet z+x\bullet\bar{y}\bullet\bar{z}+x\bullet\bar{y}\bullet z$$

On peut simplifier cette expression pour obtenir

$$\begin{aligned}G&=\bar{x}\bullet\bar{y}\bullet(z+\bar{z})+x\bullet\bar{y}\bullet(z+\bar{z})\\&=\bar{x}\bullet\bar{y}+x\bullet\bar{y}\\&=\bar{y}\bullet(x+\bar{x})=\bar{y}\end{aligned}$$

Exercice 3

1. Il faut d'abord remplir une table de Karnaugh. On rappelle qu'une table de Karnaugh est une table de vérité dans laquelle on passe d'une case à une case adjacente en ne changeant la valeur que d'une seule des variables logiques ! Pour remplir cette table il est possible de calculer la valeur de la fonction f_1 pour chacune des seize combinaisons possibles des variables *a, b, c et d*. En pratique, il est cependant beaucoup plus rapide de considérer séparément chacun des termes de l'expression de f_1 et d'en déduire les combinaisons pour lesquelles la fonction f_1 vaut 1. Ainsi dans l'expression $f_1 = a \bullet \overline{b} + a \bullet b \bullet \overline{d} + a \bullet b \bullet \overline{c} \bullet d$

- le terme $a \bullet \overline{b}$ implique que $f_1 = 1$ lorsque $a = 1$ et $b = 0$. On peut donc remplir la quatrième colonne de la table de Karnaugh suivante avec des 1.
- le terme $a \bullet c \bullet \overline{d}$ implique que $f_1 = 1$ lorsque $a = 1$, $b = 1$ et $d = 0$. On peut donc mettre 1 sur la première et la quatrième ligne de la troisième colonne.
- le terme $a \bullet c \bullet \overline{c} \bullet d$ implique que $f_1 = 1$ lorsque $a = 1$, $b = 1$, $c = 0$ et $d = 1$. On peut donc mettre 1 sur la deuxième ligne de la troisième colonne.

On obtient ainsi très rapidement la table de Karnaugh

	a	**0**	**0**	**1**	**1**
c ***d***	***b***	**0**	**1**	**1**	**0**
0	**0**	0	0	1	1
0	**1**	0	0	1	1
1	**1**	0	0	0	1
1	**0**	0	0	1	1

f_1

A partir des trois regroupements proposés la fonction f_1 s'écrit $f_1 = a \bullet \overline{d} + a \bullet \overline{c} + a \bullet \overline{b}$

Autocorrection

2. On procède de la même façon pour la fonction f_2 dont une table de Karnaugh s'écrit :

		a	**0**	**0**	**1**	**1**
c	**d**	**b**	**0**	**1**	**1**	**0**
0	**0**		1	1	0	0
0	**1**		1	1	0	0
1	**1**		0	0	1	1
1	**0**		1	1	1	1

F_2

Avec les regroupements proposés la fonction f_2 se simplifie sous la forme :

$$f_2 = \overline{a} \bullet \overline{c} + a \bullet c + c \bullet \overline{d}$$

Dans cet exemple d'autres regroupements sont possibles qui aboutissent à une expression différente de f_2 .

		a	**0**	**0**	**1**	**1**
c	**d**	**b**	**0**	**1**	**1**	**0**
0	**0**		1	1	0	0
0	**1**		1	1	0	0
1	**1**		0	0	1	1
1	**0**		1	1	1	1

F_2

Autocorrection

On obtient alors $f_2 = \bar{a} \bullet \bar{c} + a \bullet c + \bar{a} \bullet \bar{d}$.

Les deux expressions de f_2 sont rigoureusement équivalentes et l'on passe de l'une à l'autre en appliquant les règles de l'algèbre de Boole. En effet

$$\begin{aligned} f_2 &= \bar{a} \bullet \bar{c} + a \bullet c + c \bullet \bar{d} \\ &= \bar{a} \bullet \bar{c} + a \bullet c + c \bullet \bar{d} + \bar{a} \bullet \bar{d} \quad \text{(relation du consensus)} \\ &= \bar{a} \bullet \bar{c} + a \bullet c + \bar{a} \bullet \bar{d} + c \bullet \bar{d} \\ &= \bar{a} \bullet \bar{c} + a \bullet c + \bar{a} \bullet \bar{d} \quad \text{(relation du consensus)} \end{aligned}$$

3. Pour la fonction f_3 une table de Karnaugh s'écrit :

A	**0**	**0**	**1**	**1**
c d b	**0**	**1**	**1**	**0**
0 0	0	0	1	0
0 1	0	0	1	1
1 1	0	0	0	1
1 0	0	0	0	0

F3

On obtient alors l'expression simplifiée $f_3 = a \bullet b \bullet \bar{c} + a \bullet \bar{b} \bullet d$

Chapitre 2

Représentation des nombres, codage

L'électronique numérique manipule des variables logiques 0 ou 1 obéissant aux règles de l'algèbre de Boole. Nous allons voir dans ce chapitre comment une information en utilisant des variables logiques.

2.1. *Représentation des nombres, codes pondérés.*

2.1.1. Les systèmes de numération.

a. Numération en base b

Les nombres entiers ou décimaux peuvent être représentés dans plusieurs bases différentes. De manière générale l'expression d'un nombre en base B est de la forme:

$$A = a_{n-1}a_{n-2}a_{n-3}\cdots a_2 a_1 a_0 \,,a_{-1}a_{-2}a_{-3}\cdots$$

Chaque coefficient a_i est un symbole (le plus souvent un chiffre) compris entre 0 et b-1. Dans un système de numération en base *b* on attribue au chiffre qui occupe la position $1+i$, le poids b^i. La position est repérée par rapport à la virgule et croit de la droite vers la gauche. La valeur en base B du nombre A précédent est ainsi:

$$(A)_b = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_1 b^1 + a_0 b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \ldots$$

Les symboles disponibles pour écrire un nombre dépendent de la base utilisée et sont compris dans l'intervalle $[0, b-1]$.

b. Numération décimale (code décimal).

C'est le système de numération usuel dans la vie quotidienne. Puisque $b=10$ il dispose de 10 symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. L'entier 1596 correspond à

$$(1596)_{10} = 1\times 10^3 + 5\times 10^2 + 9\times 10^1 + 6\times 10^0$$

Cette base, très pratique lorsque l'on a 10 doigts, n'est pas adaptée au fonctionnement des microprocesseurs pour lesquels on fait appel à d'autres bases.

c. Numération binaire (code binaire naturel).

La numération en base 2 ou numération binaire utilise deux symboles 0 et 1. Cette base est très pratique en électronique numérique pour distinguer deux états logiques. On écrit :

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_2 = a_{n-1}\times 2^{n-1} + a_{n-2}\times 2^{n-2}\cdots + a_1\times 2^1 + a_0\times 2^0$$

La partie droite de l'équation donne la valeur en décimal du nombre binaire écrit à gauche. Par exemple:

$$(1011,01)_2 = 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} = 11,25$$

Un élément binaire 0 ou 1 est appelé un bit. En code binaire naturel, le poids du bit de rang n est 2^n (attention on compte le rang 0). Le bit de poids le plus fort est appelé MSB (*Most Significant Bit*) et celui de poids le plus faible LSB (*Less Significant Bit*)

d. Numération hexadécimale (code hexadécimal)

L'utilisation de la base $b=16$ résulte du développement des micro-ordinateurs. Les symboles utilisés dans cette base sont les dix chiffres de 0 à 9 complétés par les lettres A (pour 10), B (pour 11), C (pour12), D (pour 13), E (pour 14) ef F (pour 15). On écrit

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_2 = a_{n-1}\times 16^{n-1} + a_{n-2}\times 16^{n-2}\cdots + a_1\times 16^1 + a_0\times 16^0$$

Comme précédemment la partie droite de l'équation donne la valeur en décimal du nombre hexadécimal écrit à gauche. Par exemple:

$$(AA)_{16} = A\times 16^1 + A\times 16^0 = 10\times 16 + 10\times 1 = (170)_{10}$$

$$(B7)_{16} = B\times 16^1 + 7\times 16^0 = 11\times 16 + 7\times 1 = (183)_{10}$$

2.1.2. Changement de base, conversions.

a. Conversions vers la base décimale.

Le principe de la conversion résulte directement des définitions précédentes et de la façon dont on écrit un nombre dans une base donnée :

$$(N)_b = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_1 b^1 + a_0 b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \ldots$$

où b est codé en décimal. La conversion est ainsi réalisée automatiquement puisque le résultat est écrit directement en base 10.

b. Conversion de la base 10 vers la base b.

L'opération repose sur la remarque suivante : *le chiffre recherché de poids le plus faible est le reste de la division du nombre par la base* b. En effet, soit N le nombre à convertir on peut écrire :

$$\begin{aligned}(N)_b &= a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_1 b^1 + a_0 b^0 \\ &= (a_{n-1}b^{n-2} + a_{n-2}b^{n-2} + \ldots a_2 b^1 + a_1) \times b + a_0 \quad \text{avec } a_0 < b\end{aligned}$$

Le chiffre suivant s'obtient de la même façon à partir du quotient et ainsi de suite jusqu'au chiffre de poids le plus fort.

Prenons l'exemple de la conversion de $(89)_{10}$ en base 2. L'application de l'algorithme précédent donne $(89)_{10} = (1011001)_2$. En effet :

89 = 44 x 2 + 1	**1**	bit de poids le plus faible
44 = 22 x 2 + 0	**0**	
22 = 11 x 2 + 0	**0**	
11 = 5 x 2 + 1	**1**	
5 = 2 x 2 + 1	**1**	
2 = 1 x 2 + 0	**0**	
1 = 0 x 2 + 1	**1**	bit de poids le plus fort

Il est possible de procéder différemment en retranchant directement de *N* les

différentes puissances de b pour obtenir les chiffres dans l'ordre des poids décroissants. Cette seconde méthode appliquer à l'exemple précédent se traduit par la séquence suivante:

$89 = 64 + 25 = 25 + 2^6 \times 1$	**1**	bit de poids le plus fort
$25 = 25 + 2^5 \times 0$	**0**	
$25 = 16 + 9 = 9 + 2^4 \times 1$	**1**	
$9 = 8 + 1 = 1 + 2^3 \times 1$	**1**	
$1 = 1 + 2^2 \times 0$	**0**	
$1 = 1 + 2^1 \times 0$	**0**	
$1 = 1 + 0 = 0 + 2^0 \times 1$	**1**	bit de poids le plus faible

Le résultat final est bien sûr identique $(89)_{10} = (1011001)_2$

b. Conversions directes binaire – hexadécimal.

Les conversions entre les bases binaire et hexadécimale peuvent se faire par l'intermédiaire de la base 10 en appliquant les méthodes précédentes. Il est cependant possible d'effectuer la conversion directement.

La méthode consiste à découper le nombre à convertir en "paquets" facilement convertible dans la base désirée. Par exemple :

$$(110110000111)_2 = (\underbrace{1101}_{D}\ \underbrace{1000}_{8}\ \underbrace{0111}_{7})_2 = (D87)_{16}$$

ou bien dans l'autre sens :

$$(BD9)_{16} = (\underbrace{B}_{1011}\ \underbrace{D}_{1101}\ \underbrace{9}_{1001})_{16} = (101111011001)_2$$

2.2. *Opération arithmétiques.*

En électronique numérique l'opération fondamentale est l'addition, la soustraction n'étant rien d'autre que l'addition d'un nombre négatif. Il est donc important d'avoir une représentation des nombres négatifs qui permette d'effectuer simplement les soustractions.

2.2.1. Représentation des nombres négatifs.

Dans un circuit électronique (par exemple un ordinateur), les nombres sont représentés par des bits 0 ou 1 stockés dans des mémoires ou des registres (physiquement cela correspond à des états différents de tension électrique ou d'aimantation). Un groupe de 8 bits forme un octet[2]. Les circuits électroniques manipulent des mots formés de plusieurs octets. Les ordinateurs actuels, par exemple, utilisent des mots de 8 octets c'est-à-dire 64 bits.

Les nombres représentables sont donc en nombre fini. On ne peut pas tout représenter ni tout calculer. Les mots de 64 bits manipulés par les ordinateurs autorisent 2^{64} représentations différentes soit quelques milliards. Dans ces conditions se pose la question de la représentation la plus appropriée pour les nombres négatifs. Pour les nombres exprimés en code binaire naturel il existe au moins trois types de représentations.

a. Représentation par un bit de signe et une valeur absolue.

Le premier bit indique le signe : 0 pour le signe + et 1 pour le signe -. Le reste des bits représente la valeur absolue (en base 2). Par exemple avec 3 bits on représente les nombres :

+ 3 est représenté par 0 1 1	- 3 est représenté par 1 1 1
+ 2 est représenté par 0 1 0	- 2 est représenté par 1 1 0
+ 1 est représenté par 0 0 1	- 1 est représenté par 1 0 1
+ 0 est représenté par 0 0 0	- 0 est représenté par 1 0 0

L'inconvénient de cette convention est double ! Il y a deux représentations possibles pour le nombre 0. Par ailleurs la soustraction vue comme une addition bit à bit ne fonctionne pas. En effet

```
  0 0 0 (+0)                    0 0 1 (+1)
+ 1 0 1 (-1)        mais      + 1 1 0 (-2)
-------------                 -------------
= 1 0 1 (-1) Correct          = 1 1 1 (-3) Faux
```

[2] Les anglo saxons utilisent le terme byte

b. Représentation par le complément à 1 ou complément restreint (CR).

Il s'agit du complément par rapport au plus grand des nombres représentables. Ce nombre est composé de bits tous égaux à 1. En pratique, le complément à 1 d'un nombre x écrit sur n bits est obtenu en complémentant chacun des bits. On a alors $CR(x) + x = \underbrace{111\ldots111}_{n\ bits} = 2^n - 1$

Les nombres représentables avec $n = 3$ bits sont donnés dans le tableau ci-dessous. La représentation des nombres positifs ne change pas, il s'agit toujours d'un bit de signe égal à 0 suivi de la valeur absolue, en revanche, pour les nombres négatifs le bit de signe est toujours 1 mais la valeur absolue n'apparaît plus.

+ 3 est représenté par 0 1 1	- 3 est représenté par 1 0 0
+ 2 est représenté par 0 1 0	- 2 est représenté par 1 0 1
+ 1 est représenté par 0 0 1	- 1 est représenté par 1 1 0
+ 0 est représenté par 0 0 0	- 0 est représenté par 1 1 1

Il y a toujours l'inconvénient de deux représentations pour le chiffre 0 mais nous verrons que cette représentation permet d'effectuer des soustractions

c. Représentation par le complément à 2 ou complément vrai (CV).

Le complément vrai d'un nombre s'obtient en ajoutant 1 au complément restreint $CV(x) = CR(x) + 1$. On a alors $CV(x) + x = 2^n$ et les nombres représentables avec $n = 3$ bits sont les suivants :

+ 3 est représenté par 0 1 1	- 3 est représenté par 1 0 1
+ 2 est représenté par 0 1 0	- 2 est représenté par 1 1 0
+ 1 est représenté par 0 0 1	- 1 est représenté par 1 1 1
+ 0 est représenté par 0 0 0	- 4 est représenté par 1 0 0

Par rapport à la représentation en complément à 1 il n'y a plus qu'une seule représentation pour le nombre 0 ce qui permet de libérer la combinaison 1 0 0 pour

représenter – 4. En effet $4 = 100$ donc $-4 = CV(4) = 011+1 = 100$ [3].

2.2.2. Réalisation pratique de la soustraction.

Quel est l'intérêt des représentations complémentées à 1 ou 2? Supposons que l'on manipule des nombres entiers écrits en binaires sur n bits. La soustraction se réduisant à l'addition d'un nombre négatif, toutes les opérations se font modulo 2^n puisque, sur n bits, le nombre 2^n n'est représenté que par des 0.

a. Utilisation du complément à 2.

Considérons deux nombres positifs M et N représentés sur n bits, la soustraction $M - N$ s'écrit sous la forme

$$N - M = N - M + 2^n = N + \underbrace{(2^n - M)}_{CV(M)}$$

La complémentation à 2 permet donc de ramener très simplement la soustraction à l'addition. Dans la situation où $N < M$ le résultat est directement obtenu dans la représentation complément à 2 :

$$N - M = -(M - N) = \underbrace{2^n - (M - N)}_{CV(M-N)} \quad si\ N < M$$

Les exemples suivants illustrent ce fonctionnement. Lorsque $N > M$ l'addition $N + CV(M)$ peut conduire à une retenue, c'est-à-dire à un bit $n+1$ égal à 1. Cette retenue n'est pas prise en compte puisque l'on ne dispose que de n bits pour stocker le résultat de l'opération.

	0	1	0	(+2)			0	1	1 (+3)
+	1	0	1	(-3)		+	1	1	1 (-1)
=	1	1	1	(-1)	Correct	=	1	0	1 0 (+2) Correct
							↑		
							retenue		

La complémentation à 2 permet donc de ramener très simplement la soustraction à

[3] Attention le codage sur 3 bits d'un nombre signé ne permet de représenter le nombre 4.

l'addition, par contre sa réalisation n'est pas immédiate.

b. utilisation du complément à 1.

La complémentation à 1 rend la soustraction un peu plus compliquée puisqu'il faut maintenant écrire :

$$N - M = N - M + 2^n - 1 + 1 = N + \underbrace{(2^n - 1 - M)}_{CR(M)} + 1 \quad si\ N > M$$

$$N - M = -(M - N) = 2^n - 1 - (M - N) = N + \underbrace{(2^n - 1 - M)}_{CR(M)} \quad si\ N < M$$

Dans le premiers cas il faut additionner le complément à 1 de M puis ajouter encore 1. Dans le second cas il suffit d'ajouter le complément à 1 de M pour obtenir le résultat complémenté à 1. Ces deux situations sont illustrées sur les exemples ci-dessous. de nouveau la retenue qui peut éventuellement apparaître n'est pas stockée sur les n bits disponibles et peut donc être oubliée.

	0	1	0	(+2)				0	1	1	(+3)
+	1	0	0	(-3)		mais	+	1	1	0	(-1)
							+			1	
=	1	1	0	(-1)	Correct		=	~~1~~	0	1	0 (+2) Correct
								↑			
								retenue			

La complémentation à 1 est plus commode à calculer que la représentation à 2 (il suffit de complémenter les bits), en revanche la réalisation de la soustraction nécessite d'ajouter 1 dans certaines circonstances. En pratique cela n'est pas une vraie complication puisqu'il suffit d'utiliser l'entrée de retenue des additionneurs.

c. Problèmes de débordement.

Quelque soit la représentation adoptée pour réaliser une opération arithmétique il faut toujours s'assurer que le résultat obtenu reste dans les limites de la représentation (entre -4 et +3 dans le cas de la représentation par complément à 2 sur 3 bits). Si ce n'est pas le cas les résultats obtenus sont aberrants.

	1	1	0	(-2)			0	1	1	(+3)
+	1	0	1	(-3)	+	0	1	0	(+2)	
=	0	1	1	(+3) Faux	=	1	0	1	(-3)	Faux

2.3. *Codage des nombres.*

Un code est une correspondance arbitraire entre un ensemble de symboles et un ensemble d'objets. Les symboles peuvent être des lettres, des chiffres, des signes de ponctuation … Certains codes permettent d'effectuer des opérations arithmétiques, d'autres permettent de détecter des erreurs lors d'une transmission de données voir de les corriger.

2.3.1. Les codes pondérés.

a. Codes binaire, décimal et hexadécimal.

Les codes du paragraphe 2.1.1 sont des codes pondérés. Dans une base donnée, chaque bit est affecté d'un poids proportionnel à sa position. Il existe d'autres types de codes pondérés notamment les représentations décimales codées binaires.

b. Code BCD (Binary Coded Decimal)

Il s'agit d'une représentation des nombres en base 10, où chaque chiffre est codé en binaire. Il faut 4 bits pour représenter les 10 chiffres de la base 10 et chaque bit d'un groupe de 4 est affecté de son poids naturel. On écrit ainsi par exemple $(421)_{10} = (0100\ 0010\ 0001)_{BCD}$ au lieu de $(110100101)_2$ en binaire naturel.

En code BCD un nombre de n chiffres occupe toujours $4n$ bits.

2.3.2. Les codes non pondérés.

a. Le code ASCII (American Standard Code for Information Interchange).

C'est le code le plus utilisé dans les transmissions entre une unité centrale et ses périphériques. Il sert à coder des lettres, des chiffres et un certain nombre d'ordres qui correspondent souvent aux touches du clavier (par exemple la touche ENTREE). Ces symboles sont codés en binaire sur 7 bits ce qui permet $2^7 = 128$ possibilités.

La transmission d'une information se fait en réalité sur 8 bits. le dernier bits est en principe un bit de parité servant à la détection des erreurs: il est mis à 0 si le nombre de bits du signal est pair , et à 1 dans le cas inverse. On peut ainsi détecter un erreur se produisant sur un bit. En réalité le code ASCII, qui a été mis au point pour la langue anglaise, ne contient pas de caractères accentués ni de caractères propres à une langue. Le 8eme bit est donc souvent utilisé pour transmettre ces caractères. On parle alors de code ASCII étendu.

b. Codes adjacents.

Lorsque deux chiffres ou nombres consécutifs ont toujours des représentations qui ne différent que par un seul bit on dit qu'il s'agit d'un code adjacent. Si l'adjacence est complète (avec retour au point de départ) on parle de code cyclique. Ces codes permettent de diminuer les risques de comportement erratique lors des changements de combinaison.

On a représenté ci-dessous la construction des codes cycliques appelés code GRAY, pour les numérations hexadécimale et décimale.

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

Il arrive que deux chiffres changent simultanément entre deux nombres consécutifs:

par exemple lors du passage de 19 à 20. Dans ce cas on perd le bénéfice du caractère cyclique du code GRAY . Pour éviter cet inconvénient on utilise la convention suivante.

- Le chiffre des unités est représenté par le code GRAY si la dizaine est paire (par exemple 08, 24 …)
- Le chiffre des unités est représenté par le code CRAY de son complément à 9 (en décimal) si la dizaine est impaire (18, 35 …)

2.3.3. Codes correcteurs d'erreurs.

Les transmissions numériques nécessitent des rapports signal/bruit beaucoup plus faibles que les transmission analogiques. Cela étant, même si un rapport signal/bruit de 1 est acceptable il est toujours possible qu'un bit soit modifié lors d'une transmission de données. Des codes on donc été développés pour détecter et éventuellement corriger ces erreurs.

Nous nous contenterons de donner l'exemple de deux codes détecteurs d'erreurs.

a. Ajout d'un bit de parité.

On ajout un bit supplémentaire à l'information que l'on souhaite transmettre. Ce bit est tel que le nombre total de 1 soit pair comme indiqué sur l'exemple ci-dessous

Information	**Bit de parité**
1 0 1 0	0
1 0 0 0	1
1 1 1 0	1

On détecte ainsi les erreurs qui portent sur un seul bit, mais ce sont les plus fréquentes sauf si le taux d'erreur est exorbitant!

b. Codes p parmi n

Dans ce type de code les chiffres de 0 à 9 sont représentés par des combinaisons qui comportent toujours p bits à 1 parmi les n bits codant le chiffre. Le tableau suivant donne l'exemple du code 2 parmi 5. Ce type de code permet également de détecter les erreurs portant sur un seul bit.

Code 2 parmi 5 A B C D E		binaire standard *a b c d*
1 1 0 0 0	**0**	0 0 0 0
0 0 0 1 1	**1**	0 0 0 1
0 0 1 0 1	**2**	0 0 1 0
0 0 1 1 0	**3**	0 0 1 1
0 1 0 0 1	**4**	0 1 0 0
0 1 0 1 0	**5**	0 1 0 1
0 1 1 0 0	**6**	0 1 1 0
1 0 0 0 1	**7**	0 1 1 1
1 0 0 1 0	**8**	1 0 0 0
1 0 1 0 0	**9**	1 0 0 1

2.4. *Exercices.*

Exercice 1.

Effectuer les conversions suivantes:

a. $(7852)_{10}$ en base hexadécimal puis en binaire.

b. $(1101001011)_2$ en hexadécimal puis en décimal

c. $(2EA)_{16}$ en binaire puis en décimal.

Exercice 2.

On représente des *entiers signés* sur 16 bits.

1. Quel est le plus grand entier positif que l'on puisse écrire? Quel est le plus petit entier négatif que l'on puisse écrire?

2. Ecrire, en valeur absolue, les entiers précédents en base hexadécimal et décimal.

3. Donner les compléments à 1 et 2 de l'entier le plus grand.

Exercice 3.

Effectuer les opérations suivantes en complément à 2 sur 8 bits. Vérifier les résultats et indiquer les éventuels débordements. Comment peut on détecter que le résultat est faux ?

a. 125 – 26
b. 105 + 35
c. 40 – 60
d. - 38 – 96

2.5. *Correction des exercices*

Exercice 1

1. on applique la méthode exposée dans le paragraphe 2.1.2

7852 = 16 x 490 + 12 =16 x 490 + C	**C**
490 = 16 x 30 + 10 = 16 x 30 + A	**A**
30 = 16 x 1 + 14 = 16 x 1 + E	**E**
1 = 16 x 0 + 1	**1**

soit $(7852)_{10} = (1EAC)_{16}$

la conversion vers la base 2 est immédiate : $(1EAC)_{16} = (1\ 1110\ 1010\ 1100)_2$

2. Vers la base hexadécimale $(\underbrace{11}_{3}\underbrace{0100}_{4}\underbrace{1011}_{11=B})_2 = (34B)_{16}$

Par ailleurs $(34B)_{16} = 3 \times 16^2 + 4 \times 16^1 + 11 \times 16^0 = (843)_{10}$

3. $(2EA)_{16} = 2 \times 16^2 + 14 \times 16^1 + 10 \times 16^0 = (746)_{10}$

et $(\underbrace{2}_{10}\ \underbrace{E}_{0100}\ \underbrace{A}_{1010})_{16} = (1011101010)_2$

Exercice 2.

1. Les entiers étant signés un bit est nécessairement consacré au signe (celui le plus à gauche). Le plus grand entier positif s'écrit 0111111111111111.

Le plus petit entier négatif est 1111111111111111

2. Les deux nombres sont identiques en valeur absolue et s'écrivent $(7FFF)_{16} = (32767)_{10} = 2^{15} - 1$.

Autocorrection

3. Le complément à 1 s'obtient en complémentant tous les bits composants le nombre soit 1 000 0000 0000 0000. Le complément à 2 s'obtient en ajoutant 1 au complément à 1. On a ainsi 1 000 0000 0000 0001

Exercice 3.

Pour commencer il faut représenter chacun des nombres sur 8 bits, en utilisant le code CBN pour les nombres positifs et le code complément à 2 pour les nombres négatifs. Le résultat, si il est négatif, est obtenu en complément à 2 et on ne garde que les 8 premiers bits.

a.

	Retenues	**1** 1 1 1 1 1 0 0
125		0 1 1 1 1 1 0 1
- 26		1 1 1 0 0 1 1 0
= 99		= **1** 0 1 1 0 0 0 1 1

↑
Débordement

b.

	Retenues	**0** 1 1 0 0 0 1 1
105		0 1 1 0 1 0 0 1
+ 35		0 0 1 0 0 0 1 1
= 140		= 0 1 0 0 0 1 1 0 0 **FAUX**

Autocorrection

c.

	Retenues	**0** 0 0 0 0 0 0 0
40		0 0 1 0 1 0 0 0
- 60		1 1 0 0 0 1 0 0
= -20		= 0 1 1 1 0 1 1 0 0

d.

	Retenues	**1** 0 0 1 0 0 0 0
- 38		1 1 0 1 1 0 1 0
- 96		1 0 0 1 0 0 0 1
=- 134		= **1** 0 1 1 0 1 0 1 1 **FAUX**

↑
Débordement

Compte tenu du nombre de bits le résultat de l'opération doit être compris entre – 127 et +127. En dehors de cette plage le résultat est nécessairement faux comme c'est le cas pour opérations b) et d). En pratique deux situations mènent à un résultat faux :

- Il y a une retenue du bit 7 vers le bit 8 et pas de retenue externe (ou débordement) (cas b)
- Il n'y a pas de retenue du bit 7 vers le bit 8 mais il y a une retenue externe (cas d)

Dans les deux cas le bit de signe est changé accidentellement.

Chapitre 3

Logique combinatoire.

Un circuit gouverné par les règles de la logique combinatoire possède une ou plusieurs entrées, et une ou plusieurs sorties, et obéit à la propriété suivante :

L'état de la (ou des) sortie(s) à un instant donné ne dépend que du circuit et de la valeur des entrées à cet instant.

Une même combinaison des entrées donnera ainsi toujours la même valeur des sorties. Cette propriété, qui peut paraître de bon sens, n'est pas systématiquement vérifiée. Nous verrons dans le chapitre 4 qu'il existe des circuits pour lesquels létat des sorties est influencé par l'histoire du circuit. Nous parlerons alors de logique séquentielle.

3.1. *Représentation schématique des fonctions logiques de base.*

3.1.1. Les fonctions NON, ET, OU.

Sur les schémas de circuits électroniques les fonctions logiques sont représentées par des symboles que l'on appelle généralement "portes logiques". Les fonctions **NON, ET** et **OU** sont associées aux symboles représentés sur la Figure 1.

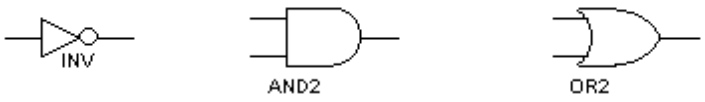


Figure 1 Symboles associés aux fonctions logiques NON, ET, OU

On rencontre aussi d'autres fonctions logiques réalisées à partir des 3 fonctions précédentes.

3.1.2. La fonction NON ET (NAND).

La fonction **NON ET** est obtenue en complémentant la fonction **ET** $F = \overline{x \bullet y}$. La table de vérité et le symbole associés à cette fonction sont :

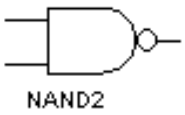


Figure 2 Symbole associé à la fonction NON ET (NAND)

x y	$F = \overline{x \bullet y}$
0 0	1
1 0	1
1 1	0
0 1	1

3.1.3. La fonction NON OU (NOR).

La fonction **NON OU** est obtenue en complémentant la fonction **OU** $F = \overline{x + y}$. La table de vérité et le symbole associés à cette fonction sont :

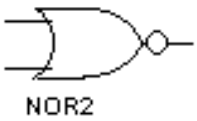


Figure 3 Symbole associé à la fonction NON OU (NOR)

x y	$F = \overline{x + y}$
0 0	1
1 0	0
1 1	0
0 1	0

3.1.4. La fonction OU EXCLUSIF (XOR).

La fonction **OU EXCLUSIF** ne vaut 1 que si les deux entrées sont différentes. Elle s'écrit $F = x \oplus y = x \bullet \overline{y} + \overline{x} \bullet y.$ La table de vérité et le symbole associés à cette fonction sont :

Figure 4 Symbole associé à la fonction OU EXCLUSIF (XOR)

x y	$F = x \oplus y$
0 0	0
1 0	1
1 1	0
0 1	1

3.2. *Réalisation matérielle d'une fonction logique.*

En combinant entre elles les différentes portes logiques on peut à priori réaliser n'importe quelle fonction logique. On appelle logigramme la réalisation d'une fonction complexe à l'aide des portes de base. A titre d'exemple réalisons la fonction OU EXCLUSIF (XOR) en n'utilisant que des portes NON, ET, OU.

- ***1ére méthode :***

On réalise la fonction $F = x \oplus y = x \bullet \overline{y} + \overline{x} \bullet y$ telle qu'elle est écrite. Obtient alors le schéma de la

Figure **5**

Cette solution n'est pas satisfaisante pour au moins deux raisons

1. On n'a pas chercher minimiser le nombre de portes utilisées.

2. On utilise 3 types de portes différents donc 3 boîtiers différents sur le montage. Sachant qu'un boîtier contient plusieurs portes (4 ou 6 généralement) on peut facilement gagner de la place en n'utilisant qu'un seul type de portes.

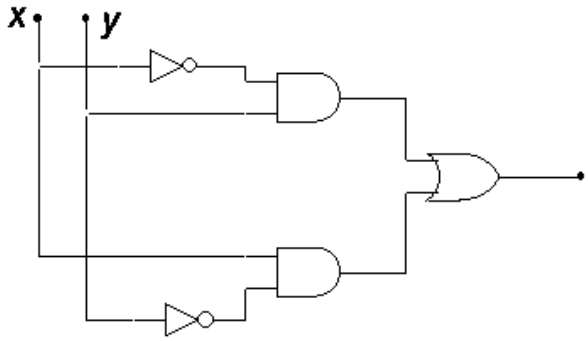


Figure 5 Réalisation de la fonction OU EXCLUSIF (XOR)

2eme méthode :

Pour simplifier la réalisation, on cherche a n'utiliser qu'un seul type de portes, par exemple des portes **NAND.**

La fonction s'écrit alors

$$F = x \oplus y = x \bullet \overline{y} + \overline{x} \bullet y = \overline{\overline{x \bullet \overline{y}} \bullet \overline{\overline{x} \bullet y}}$$

et obtient le schéma suivant (Figure 6), avec 5 portes **NAND**, en notant que $\overline{x \bullet x} = \overline{x}.$

Il faut noter ici que la fonction **NAND,** comme d'ailleurs la fonction **NOR**, est dite complète car elle permet de réaliser à elle seule toutes les fonctions logiques.

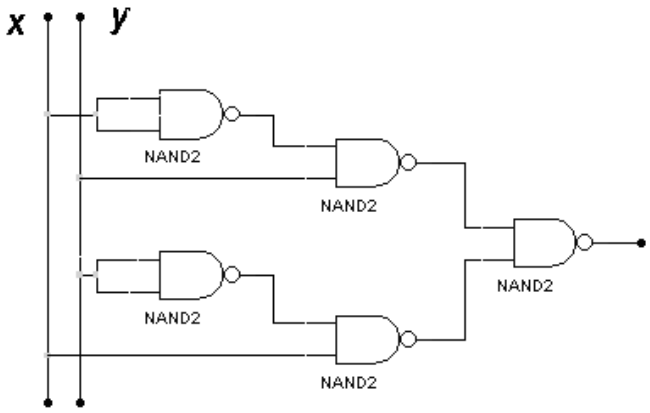


Figure 6 Réalisation de la fonction OU EXCLUSIF uniquement avec des portes NAND

3eme méthode :

En étant astucieux on remarque que

$$x \bullet \overline{y} = x \bullet (\overline{y} + \overline{x}) = x \bullet \overline{x \bullet y}$$
$$\overline{x} \bullet y = y \bullet (\overline{y} + \overline{x}) = y \bullet \overline{x \bullet y}$$

La fonction *F* peut alors se réécrire sous la forme

$$F = \overline{\overline{x \bullet \overline{x \bullet y}} \bullet \overline{y \bullet \overline{x \bullet y}}}$$

ce qui conduit au schéma de la Figure 7 qui ne comporte que 4 portes **NAND**!

Une remarque s'impose : Ce n'est pas nécessairement l'expression la plus simplifiée de la fonction logique *F* qui donne le logigramme (et donc le circuit électronique) le plus simple ou le plus optimisé. Ici nous avons gagné une porte logique en écrivant $F = x \bullet (\overline{y} + \overline{x}) + y \bullet (\overline{y} + \overline{x})$ au lieu de $F = x \bullet \overline{y} + \overline{x} \bullet y.$

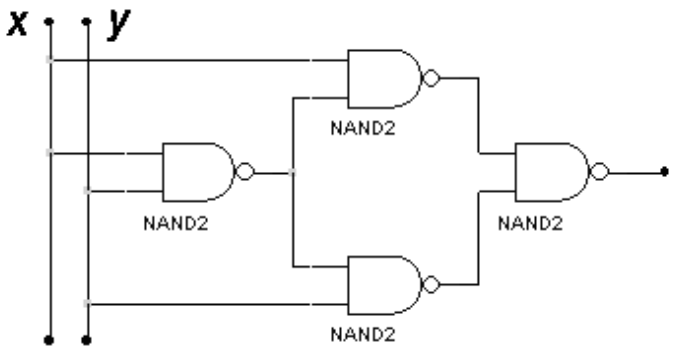


Figure 7 Réalisation de la fonction OU EXCLUSIF avec le minimum de portes logiques.

3.3. *Les aléas en logique combinatoire.*

Pour établir les schémas précédents nous n'avons pas considéré les temps caractéristiques de commutation des circuits. Nous avons implicitement supposé qu'un changement des entrées entraînait, éventuellement, un basculement instantané de la sortie. En réalité il n'en est pas ainsi, la sortie d'une porte logique commute toujours avec un certain retard par rapport aux entrées. Ce retard, dû au temps de propagation à travers

la porte, est à l'origine de phénomènes parasites appelés aléas.

3.3.1. Un exemple simple d'aléa.

On considère le circuit de la Figure 8. La fonction réalisée est très simple, puisqu'il s'agit de $s = e \bullet \bar{e}$ et la sortie vaut toujours 0 quelque soit la valeur de l'entrée *e*. Pour le vérifier on a tracé les chronogrammes de *e* et *s* en tenant compte, ou pas, du temps de retard Δ introduit par chacune des portes (Figure 9). La prise en compte du temps de retard fait apparaître une impulsion parasite sur la sortie s.

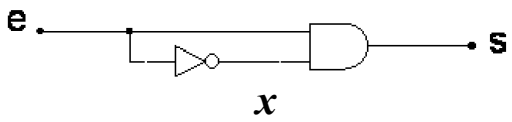


Figure 8

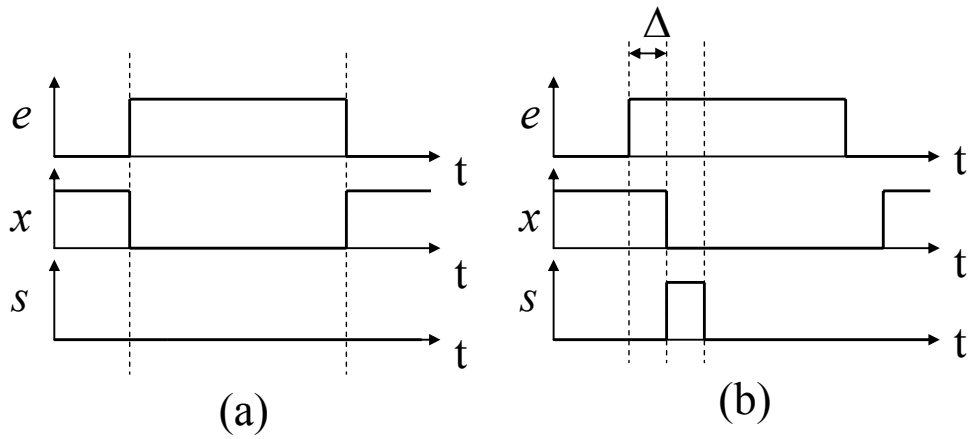


Figure 9 Chronogrammes de e, $x = \bar{e}$ et $s = e \bullet x$, (a) sans tenir compte du temps de retard, (b) en incluant le temps de propagation Δ

De façon générale un aléa risque de se manifester à la sortie d'un circuit élémentaire **ET** ou **OU** à l'occasion de transitions quasi-simultanées sur les entrées : (01) $\rightarrow$ (10) ou l'inverse.

3.3.2. Remèdes aux aléas.

Considérons la fonction logique *F* des variables logiques *a*, *b* et *c* définie par la table de Karnaugh suivante :

a \ *b* *c*	**0** **0**	**0** **1**	**1** **1**	**1** **0**
0	0	0	1	1
1	0	1	1	0

F

et dont l'expression la plus simplifiée $F = \bar{a} \bullet b + a \bullet c,$ aboutie à la réalisation de la Figure 10. On constate sur le chronogramme (Figure 11) que lors de la transition $(abc) = (111) \rightarrow (011)$ (où l'on ne change que la variable a) la sortie *F* passe un court instant par la valeur 0, alors que la table de Karnaugh indique qu'elle devrait rester à 1. Dans cet exemple ce sont les deux variables intermédiaires $x = \bar{a} \bullet b$ et $y = a \bullet c$ qui provoquent l'aléa en raison de leur léger retard l'une par rapport à l'autre lors du changement $(xy) = (01) \rightarrow (10)$.

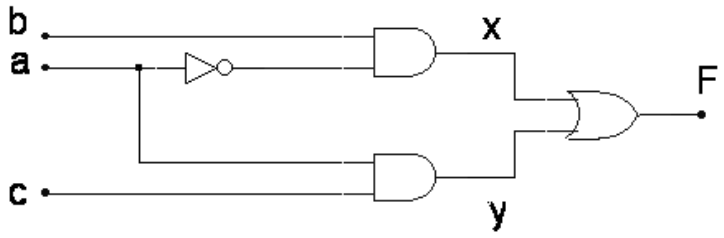


Figure 10 Schéma logique de la fonction $F = \bar{a} \bullet b + a \bullet c = x + y$

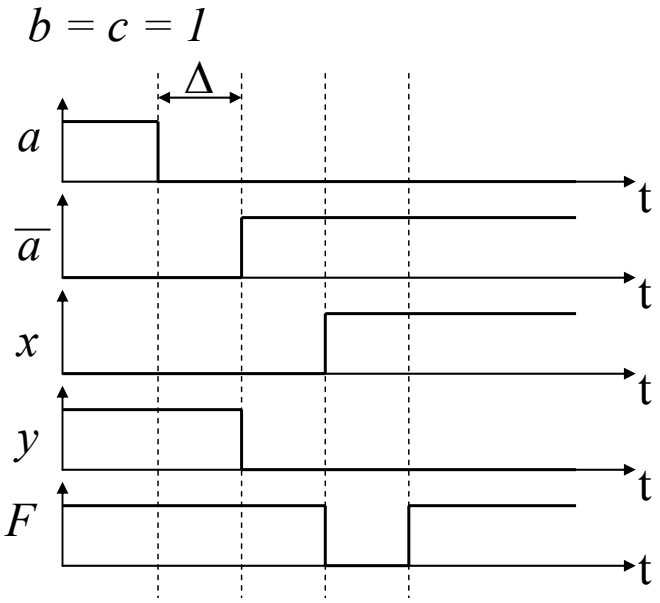


Figure 11 Chronogramme de la fonction $F = \bar{a} \bullet b + a \bullet c.$

Les deux variables secondaires x et y correspondent aux deux termes de l'expression logique ou, ce qui est équivalent, au deux regroupements réalisés dans la table de Karnaugh. La solution au problème de l'aléa découle de cette remarque: il faut que le changement de a "laisse toujours 1" à l'intérieur d'un des termes de l'expression logique. Ceci implique qu'il faut ajouter des termes supplémentaires à l'expression de *F*, qui ne sont autres que les termes de consensus. On écrira donc $F = \bar{a} \bullet b + a \bullet c + b \bullet c.$ Le terme supplémentaire $b \bullet c$ assure que la fonction *F* reste toujours à 1 dans la transition $(abc) = (111) \rightarrow (011)$

De façon plus visuelle on peut également remarquer qu'il y a un risque d'aléa lorsque, dans un tableau de Karnaugh, deux regroupements ont des cases adjacentes. Il peut alors être nécessaire de réaliser des regroupements redondants (ici $b \bullet c$ représenté en pointillés dans la table ci-dessous) afin d'assurer un recouvrement entre les regroupements initiaux $\bar{a} \bullet b$ et $a \bullet c$. A titre d'exercice on pourra vérifier que le schéma du **OU EXCLUSIF** réalisé avec 4 portes **NAND** sur Figure 7 n'induit pas d'aléa.

b / *a* *c*	**0** **0**	**0** **1**	**1** **1**	**1** **0**
0	0	0	1	1
1	0	1	1	0

F

Une nouvelle fois on constate que la simplification d'une fonction logique ne doit pas être une fin en soi. Non seulement la forme la plus simplifiée ne conduit pas obligatoirement à la réalisation matérielle la plus simple, mais elle peut également induire des phénomènes parasites comme les aléas.

3.3.3. Conséquences des aléas.

Tout d'abord il convient de noter que seul le régime transitoire des circuits combinatoires est affecté par les aléas. Le régime permanent reste correct, c'est pourquoi les aléas peuvent souvent passer inaperçus. Les conséquences des aléas se font néanmoins sentir dans au moins deux domaines:

- Ils déterminent la bande passante d'un montage (un compteur par exemple), c'est à dire la fréquence maximum pour un fonctionnement fiable.
- Ils peuvent engendrer des dysfonctionnements importants si le circuit contient des systèmes séquentiels (bascules, compteurs…). Comme nous le verrons dans le chapitre 4 que les systèmes séquentiels peuvent être contrôlés par des impulsions et sont donc sensibles aux aléas.

3.4. *Quelques circuits logiques "complexes".*

En pratique il n'est pas nécessaire de réaliser des fonctions logiques complexes en utilisant les portes logiques de base que nous venons de décrire. On trouve dans les catalogues des différents constructeurs un grand nombre de circuits logiques intégrés réalisant des fonctions logiques complexes. Nous verrons simplement ici le multiplexeur, l'encodeur et le décodeur.

3.4.1. Le multiplexeur (sélecteur de données).

- ***Principe du multiplexeur.***

Un multiplexeur possède plusieurs entrées et une seule sortie. Il agit comme un sélecteur de données en orientant vers sa sortie la donnée présente sur l'une de ses entrées. Pour fixer les idées, considérons le multiplexeur le plus simple représenté sur la Figure 12. Il s'agit un circuit logique à deux entrées, D_0 et D_1, permettant d'afficher sur sa sortie *F* la donnée présente sur une des deux entrées. Ceci n'est réalisable que si il existe une entrée supplémentaire S_0, dite entrée de sélection ou d'adresse, telle que

$$F = D_0 \text{ si } S_0 = 0$$
$$F = D_1 \text{ si } S_0 = 1$$

Un multiplexeur comporte donc deux types d'entrées : les entrées de données (ou d'informations), et les entrées de sélection (ou d'adresse) dont les combinaisons servent à numéroter les entrées d'informations

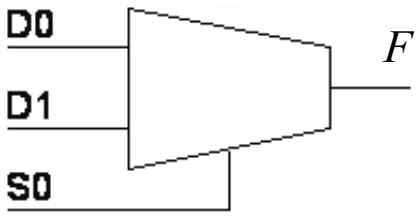


Figure 12 Multiplexeur à deux entrées de données D_0 et D_1, et une entrée de sélection S_0 (entrée d'adresse)

Généralisation.

Pour sélectionner parmi quatre entrées il faut impérativement disposer de deux entrées de sélection: chacune des quatre combinaisons possibles des entrées de sélection correspondra à l'aiguillage d'une entrée, et d'une seule, vers la sortie. De la même façon pour sélectionner parmi huit entrées données, trois entrées de sélections sont nécessaires ($2^3=8$). De façon générale, un multiplexeur possédant n entrées de sélection permet de sélectionner une entrée parmi 2^n.

La sélection de l'entrée est réalisée en affectant un poids aux entrées d'adresse $S_n \ldots S_0$. On peut, par exemple, associer S_n au bit de poids fort et S_0 a celui de poids faible. La combinaison binaire $S_n \ldots S_0$ ainsi obtenue est appelée **adresse** et le multiplexeur aiguillera vers la sortie *F* l'entrée D_i dont l'indice décimal i correspondra à l'adresse binaire $i = (S_n \ldots S_0)_2$. Ainsi, dans un multiplexeur à 8 entrées de données, l'entrée E_5 sera aiguillée vers la sortie si l'adresse binaire écrite sur les entrées de sélection est[4]:

S_2	S_1	S_0
1	0	1

Les multiplexeurs ont de nombreuses applications. Ils peuvent par exemple être utilisés comme :

- sélecteur de données.
- convertisseur parallèle-série. Le multiplexeur reçoit en parallèle des données qu'il peut transmettre l'une après l'autre sur sa sortie.
- générateur de fonctions logiques.

Utilisation en générateur de fonction logique.

La transmission de la donnée présente sur l'entrée D_5, dont l'adresse est 101, (voir plus haut) consiste pour le multiplexeur à réaliser un **ET** logique entre D_5 et le monôme $S_2 \bullet \overline{S_1} \bullet S_0$. En fait un multiplexeur à n entrées d'adresses (et donc 2^n entrées de données) peut réaliser toutes les fonctions logiques combinatoires de $n+1$ variables.

Considérons par exemple, la réalisation de la fonction $F = x \bullet y \bullet \overline{z} + \overline{x} \bullet \overline{y} \bullet z + y \bullet z$ avec un multiplexeur à 2 entrées d'adresse.

[4] En supposant que S_2 correspond au bit de poids fort de l'adresse (MSB) et S_0 au bit de poids faible.

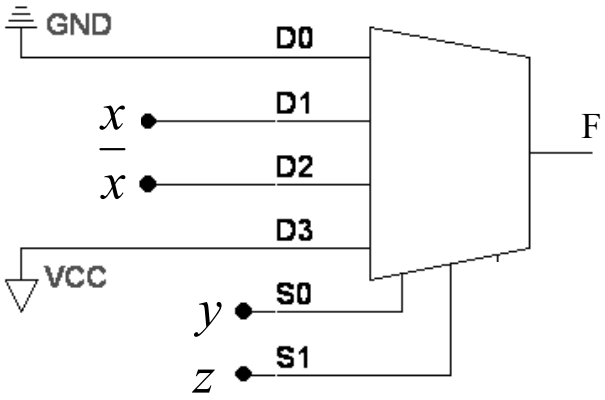


Figure 13 Réalisation de la fonction logique $F = x \bullet y \bullet \bar{z} + \bar{x} \bullet \bar{y} \bullet z + y \bullet z$ avec un multiplexeur à deux entrées d'adresse

On connecte 2 des variables aux entrées d'adresse ou de sélection. Par exemple y à S_0 et z à S_1. On doit ensuite connecter convenablement les 4 entrées de données de façon à reproduire les différents termes de la fonction F.

1. Pour le terme $x \bullet y \bullet \bar{z}$ on doit relier l'entrée D_1 dont l'adresse est 01 ($z = 0,\ y = 1$) à x.

2. Pour le terme $\bar{x} \bullet \bar{y} \bullet z$ on doit relier l'entrée D_2 dont l'adresse est 10 ($z = 1, y = 0$) à $\bar{x}$.

3. Pour le terme $z \bullet y$ on relie l'entrée dont l'adresse est 11 ($z = 1,\ y = 1$) au niveau logique 1 (par exemple 5volts en technologie TTL).

4. Toutes les autres entrées sont connectées au niveau logique 0 (la masse) puisque $F = 0$ pour les valeurs de y et z correspondantes.

Le schéma correspondant est représenté sur la Figure 13.

3.4.2. Encodeur prioritaire.

C'est un circuit à m entrées et n sorties. Les sorties délivrent le code de l'entrée active si il n'y en a qu'une ou de l'entrée prioritaire si il y en a plusieurs. Si le code est le code binaire standard alors on a $m=2^n$ et l'encodeur est dit binaire. Il existe bien entendu des encodeurs pour les codes BCD, GRAY, ASCII ... Dans le cas du code ASCII l'encodeur est utilisé pour coder les touches d'un clavier, c'est à dire pour générer le code ASCII

associé au caractère ou à la fonction de la touche enfoncée. Le fonctionnement d'un encodeur prioritaire a quatre entrée est décrit par la table de vérité suivante. Les entrées sont actives au niveau haut (1), lorsque plusieurs entrées E_i sont actives, seul le code correspondant à l'entrée d'indice le plus élevé est présent sur les sorties $a_1\, a_0$. Dans cet exemple l'entrée E_3 a la plus forte priorité.

E_0	E_1	E_2	E_3	a_1	a_0
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1

3.4.3. Le décodeur-démultiplexeur.

Un démultiplexeur est un aiguilleur à une entrée de donnée, n entrées d'adresse et m sorties. La valeur de l'entrée se retrouve sur la sortie dont le numéro est codé par l'adresse. Dans cette fonction le circuit joue le rôle inverse du multiplexeur.

Ces circuits sont aussi des décodeurs : si l'entrée est maintenue active, le numéro de la sortie reflète le code de l'adresse. Dans le cas d'un code binaire on a $m=2^n$. Mais il existe aussi des décodeurs pour les codes BCD, GRAY , …

Un démultiplexeur dont l'entrée E est maintenue active au niveau 1 peut également servir de générateur de fonctions logiques. En effet, puisque l'on retrouve sur les sorties toutes les combinaisons possibles des entrées d'adresse, une porte OU suffit pour fabriquer une fonction logique sous sa première forme canonique.

Soit la fonction logique de trois variables $f=\bar{a}\bullet b\bullet c+a\bullet\bar{b}\bullet c+a\bullet b\bullet\bar{c}+a\bullet b\bullet c$. Le codage des sorties avec l'adresse $(c\ b\ a)$ correspond aux sorties 6, 5 , 3 et 7. D'où le schéma suivant

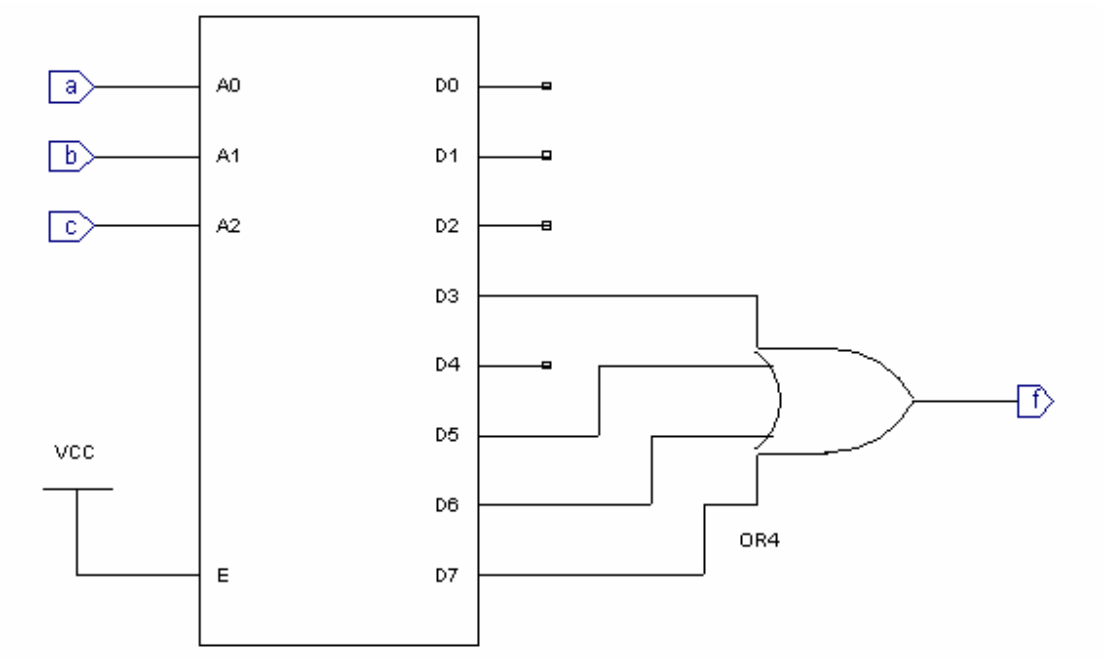


Figure 14 Réalisation d'une fonction logique avec un décodeur/démultiplexeur.

3.5. *Exercices.*

Exercice 1

On considère le montage de la Figure 15.

1. Quelle est la fonction logique F réalisée par ce montage ?

2. Simplifier la fonction F (on peut utiliser indifféremment des tables de Karnaugh ou le théorème de De Morgan).

3. Proposer un montage plus simple permettant de réaliser la fonction F.

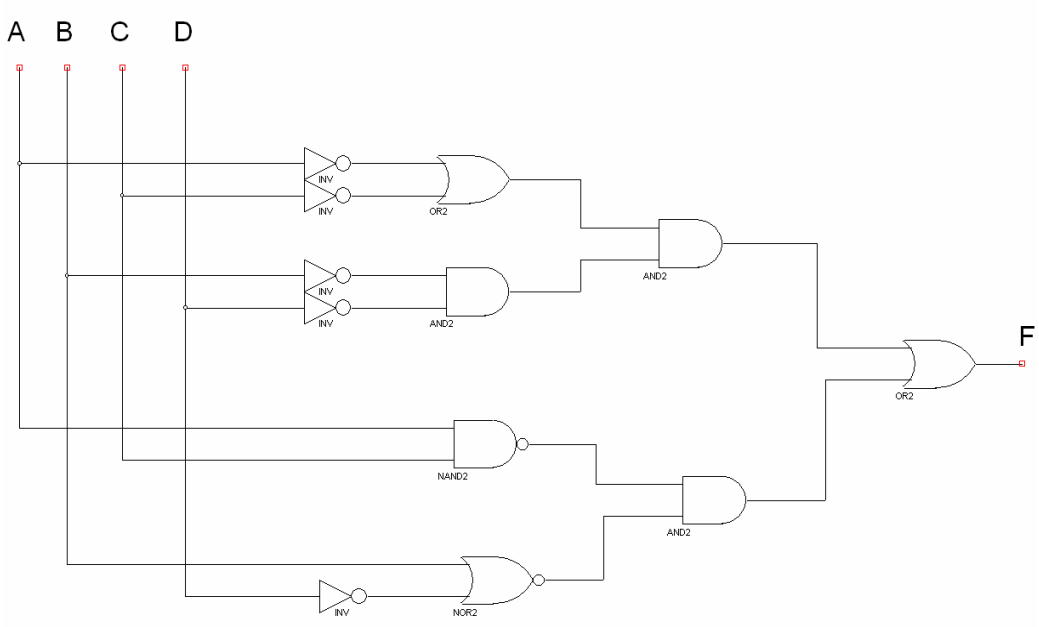


Figure 15

Autocorrection

Exercice 2

Réaliser la fonction équivalence $f = \overline{x \oplus y} = \overline{x} \bullet \overline{y} + x \bullet y$ à l'aide de 4 portes logiques **NOR** uniquement.

Exercice 3

1 . Donner l'équation de la fonction logique F réalisée par le multiplexeur représenté sur la Figure 16.

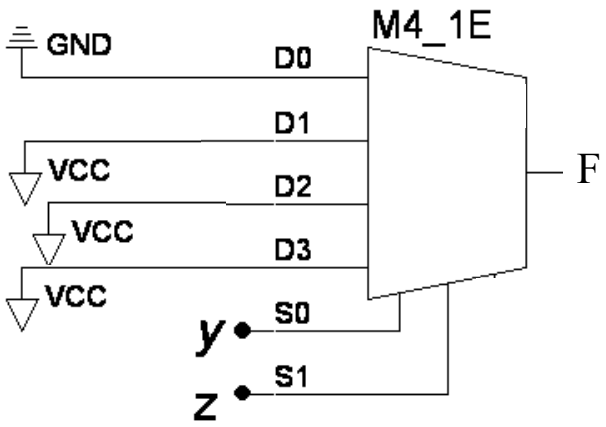


Figure 16

2. A l'aide d'un multiplexeur à 2 entrées d'adresses, réaliser la fonction :

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

3. A l'aide d'un multiplexeur à 3 entrées d'adresses, réaliser la fonction

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

Exercice 4

En binaire, un chiffre décimal (compris entre 0 et 9) est codé sur 4 bits $a\ b\ c\ d$ dans l'ordre des poids décroissants. Ce chiffre est visualisé sur un afficheur 7 segments représenté sur la Figure 17. Chaque segment est représenté par une lettre allant de A à G. Lors de l'affichage du chiffre 6 (respectivement 9) le segment A (respectivement D) est allumé.

1. Donner les expressions logiques, en fonction de $a\ b\ c\ d$, des fonctions logiques f_A et f_D valant 1 lorsque les segments A et D de l'afficheur sont allumés.
2. Simplifier les fonctions précédentes en utilisant des tables de Karnaugh.
3. Donner le schéma de la fonction f_A avec un minimum de portes **NOR** et **NAND**.
4. Donner le schéma de la fonction f_D en utilisant un multiplexeur à 3 entrées d'adresse.

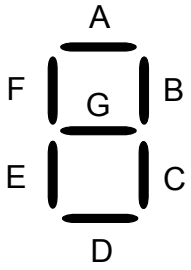


Figure 17 Afficheur 7 segments.

3.6. *Correction des exercices*

Exercice 1

1. On lit directement la fonction à partir du logigramme

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

2. On simplifie en appliquant le théorème de Morgan puis les règles de calcul de l'algèbre de Boole

$$\begin{aligned} F &= (\overline{C} + \overline{A}) \bullet \overline{B} \bullet \overline{D} + \overline{A \bullet C} \bullet \overline{\overline{D} + B} \\ &= \overline{A \bullet C} \bullet \overline{B} \bullet \overline{D} + \overline{A \bullet C} \bullet D \bullet \overline{B} \\ &= \overline{A \bullet C} \bullet \overline{B} \bullet (D + \overline{D}) = \overline{A \bullet C} \bullet \overline{B} = \overline{A \bullet C + B} \end{aligned}$$

3. Le logiramme de la fonction F peut donc se réécrire

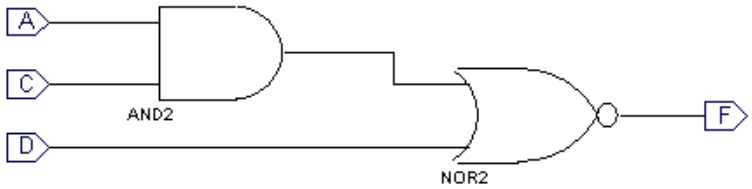


Figure 18

Exercice 2

On procède comme pour la fonction **XOR** (OU EXCLUSIF) à la page 49. L'idée est d'introduire des termes supplémentaires qui en apparence compliquent l'écriture de la fonction logique mais qui en pratique simplifient la réalisation.

Autocorrection

$$y \bullet \overline{x} = \overline{\overline{x} + \overline{y}} = \overline{x + \overline{y} \bullet (x + \overline{x})} = \overline{x + \overline{y} \bullet \overline{x} + y \bullet x} = \overline{x + \overline{y} \bullet \overline{x}}$$

$$x \bullet \overline{y} = \overline{\overline{y} + \overline{x}} = \overline{y + \overline{x} \bullet (y + \overline{y})} = \overline{y + \overline{x} \bullet \overline{y} + y \bullet x} = \overline{y + \overline{y} \bullet \overline{x}}$$

La fonction f se réécrit donc $f = \overline{x \oplus y} = \overline{x} \bullet \overline{y} + x \bullet y = \overline{\overline{x + (x + y)} + \overline{y + (x + y)}}$ ce qui aboutit au schéma suivant :

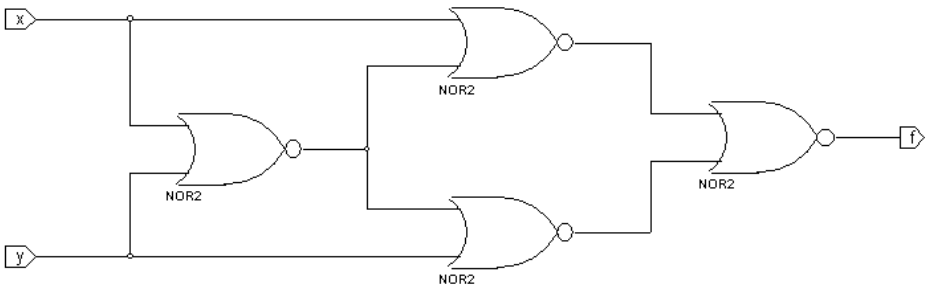


Figure 19

Exercice 3

1. L'adresse est écrite sous la forme $(z\ y)$. Les entrées de données d'adresses 1 $(z\ y = 0\ 1)$, 2 $((z\ y = 1\ 0)$ et 3 $(z\ y = 1\ 1)$ sont au niveau haut (1 logique). La fonction logique réalisée par le multiplexeur s'écrit donc $F = z \bullet \overline{y} + \overline{z} \bullet y + z \bullet y$

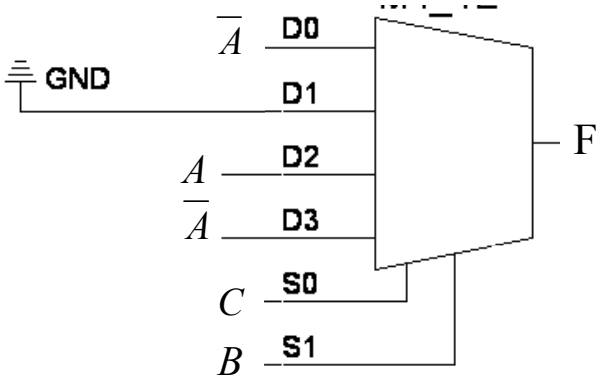
2. Il y a plusieurs solutions possibles, toutes correctes, selon les choix que l'on effectue pour les bits d'adressage. Décidons par exemple d'écrire les adresses sous la forme $S_1 S_0 = B\,C$ [5]. Avec ce choix le bit A doit être relié aux entrées de données selon le tableau

[5] B est donc le bit de poids fort de l'adresse et C celui de poids faible.

Autocorrection

	B C	F	entrée
adresse	0 0	$\overline{A}$	D_0
adresse	0 1	0	D_1
adresse	1 0	A	D_2
adresse	1 1	$\overline{A}$	D_3

Le schéma du multiplexeur est donc



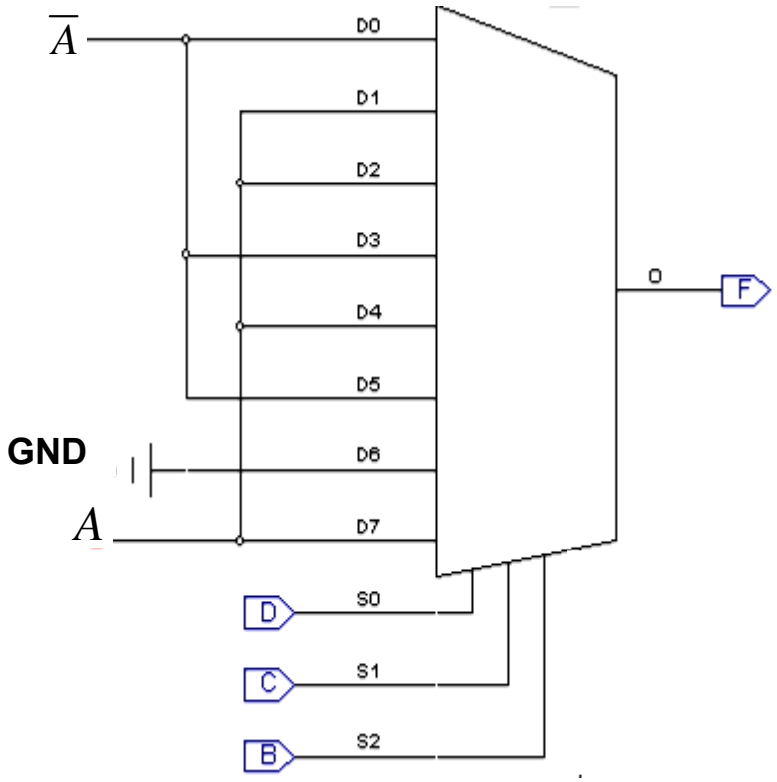
3. On applique exactement la même démarche. Si on choisit d'écrire l'adresse sous la forme $S_2\, S_1\, S_0 = B\ C\ D$. Le bit A doit être relié aux entrées de données selon le tableau

Autocorrection

	B	C	D	F	entrée
adresse	0	0	0	$\overline{A}$	D_0
adresse	0	0	1	A	D_1
adresse	0	1	0	A	D_2
adresse	0	1	1	$\overline{A}$	D_3
adresse	1	0	0	A	D_4
adresse	1	0	1	$\overline{A}$	D_5
adresse	1	1	0	0	D_6
adresse	1	1	1	A	D_7

Le schéma du multiplexeur est alors

Autocorrection



Exercice 4

1. On dresse la table de vérité des fonctions f_A et f_D

a	b	c	d	f_A	f_D
0	0	0	0	1	1
0	0	0	1	0	0
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1

Cette table de vérité ne concerne évidemment que les dix combinaisons de a, b, c et d correspondant aux chiffres de 0 à 9. Les six autres combinaisons ne sont pas spécifiées.

Les expressions des fonctions logiques f_A et f_B s'obtiennent directement à partir de la table:

$$f_A = \bar{a} \bullet \bar{b} \bullet \bar{c} \bullet \bar{d} + \bar{a} \bullet \bar{b} \bullet c \bullet \bar{d} + \bar{a} \bullet \bar{b} \bullet c \bullet d + \bar{a} \bullet b \bullet \bar{c} \bullet d + \bar{a} \bullet b \bullet c \bullet \bar{d} + \bar{a} \bullet b \bullet c \bullet d + a \bullet \bar{b} \bullet \bar{c} \bullet \bar{d} + a \bullet \bar{b} \bullet \bar{c} \bullet d$$

$$f_D = \bar{a} \bullet \bar{b} \bullet \bar{c} \bullet \bar{d} + \bar{a} \bullet \bar{b} \bullet c \bullet \bar{d} + \bar{a} \bullet \bar{b} \bullet c \bullet d + \bar{a} \bullet b \bullet \bar{c} \bullet d + \bar{a} \bullet b \bullet c \bullet \bar{d} + a \bullet \bar{b} \bullet \bar{c} \bullet \bar{d} + a \bullet \bar{b} \bullet \bar{c} \bullet d$$

2. Pour simplifier les fonctions précédentes le plus simple est d'utiliser la méthode des

Autocorrection

tables de Karnaugh, d'autant plus que les deux fonctions sont incomplètement spécifiées et que des regroupements supplémentaires vont éventuellement être possibles.

		a	**0**	**0**	**1**	**1**
c	**d**	**b**	**0**	**1**	**1**	**0**
0	**0**		1	0	φ	1
0	**1**		0	1	φ	1
1	**1**		1	1	φ	φ
1	**0**		1	1	φ	φ

f_A

Soit $f_A = c + a + b \bullet d + \overline{b} \bullet \overline{d}$

		a	**0**	**0**	**1**	**1**
c	**d**	**b**	**0**	**1**	**1**	**0**
0	**0**		1	0	φ	1
0	**1**		0	1	φ	1
1	**1**		1	0	φ	φ
1	**0**		1	1	φ	φ

f_D

Soit $f_D = a + \overline{b} \bullet \overline{d} + c \bullet \overline{b} + c \bullet \overline{d} + b \bullet \overline{c} \bullet d$

Dans les deux simplifications précédentes toutes les situations indéterminées

Autocorrection

(correspondant à des φ) dans les tables de Karnaugh ont été prises en compte. Rappelons que ceci n'est pas une obligation et que le choix doit se faire au cas par cas (voir le paragraphe 1.3.2)

3. On utilise les règles de l'algèbre de Boole pour modifier l'expression de f_A en ne faisant apparaître que des fonctions **NOR** et **NAND.**

$$\begin{aligned} f_A &= c + a + b \bullet d + \overline{b} \bullet \overline{d} = c + a + b \bullet d + \overline{b + d} \\ &= c + a + \overline{(\overline{b} + \overline{d}) \bullet (b + d)} \\ &= c + a + \overline{b \oplus d} \\ &= \overline{\overline{c + a} \bullet (b \oplus d)} \end{aligned}$$

On aboutit ainsi à un circuit logique ne comportant que 5 portes **NAND** et une porte **NOR** (**Figure 20**)

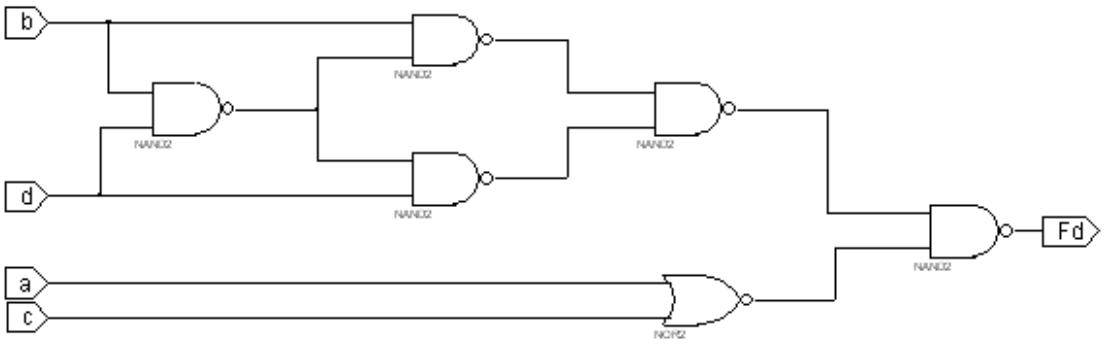


Figure 20

4. Pour réaliser la fonction f_D avec un multiplexeur on peut utiliser aussi bien la forme simplifiée que la forme non simplifiée. Utilisons l'expression non simplifiée

$$f_D = \overline{a} \bullet \overline{b} \bullet \overline{c} \bullet \overline{d} + \overline{a} \bullet \overline{b} \bullet c \bullet \overline{d} + \overline{a} \bullet \overline{b} \bullet c \bullet d + \overline{a} \bullet b \bullet \overline{c} \bullet d + \overline{a} \bullet b \bullet c \bullet \overline{d} + a \bullet \overline{b} \bullet \overline{c} \bullet \overline{d} + a \bullet \overline{b} \bullet \overline{c} \bullet d$$

et choisissons d'écrire l'adresse sous la forme $S_2\, S_1\, S_0 = c\ b\ a$. Le bit d doit être relié aux entrées de données selon le tableau

Autocorrection

	c	*b*	*a*	f_D	**entrée**
adresse	0	0	0	$\overline{d}$	D_0
adresse	0	0	1	1	D_1
adresse	0	1	0	d	D_2
adresse	0	1	1	0	D_3
adresse	1	0	0	1	D_4
adresse	1	0	1	0	D_5
adresse	1	1	0	$\overline{d}$	D_6
adresse	1	1	1	0	D_7

Le schéma du multiplexeur est donné sur la Figure 21

Autocorrection

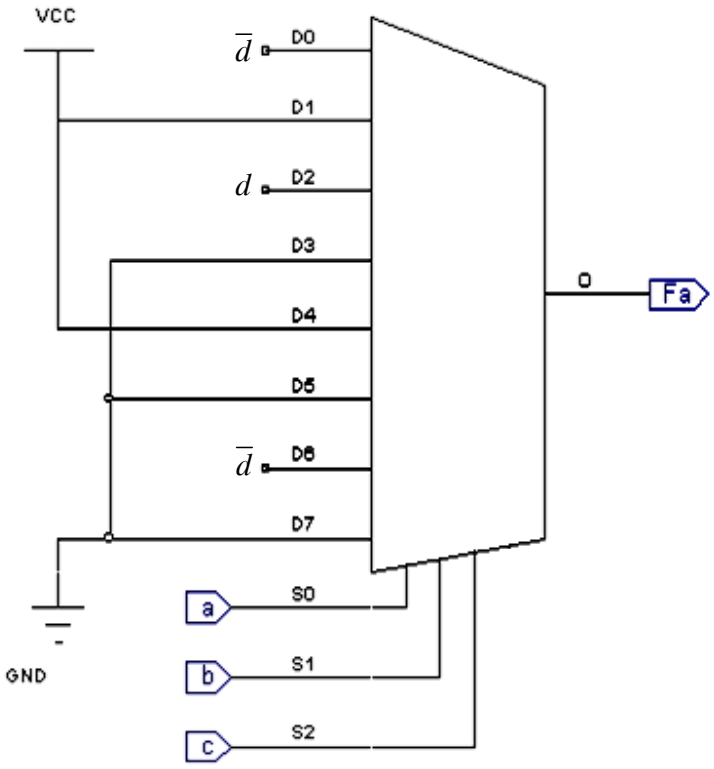


Figure 21

Chapitre 4

Logique séquentielle.

4.1. *Introduction*

Tous les circuits que nous avons étudiés jusqu'a présent ont été réalisés en agençant, de façon plus ou moins astucieuse, des portes logiques les unes derrière les autres. A aucun moment la sortie d'une porte logique n'a été rebouclée, plus ou moins directement, vers son entrée (Figure 22.a). Ce type de circuit, pour le quel on peut toujours calculer de proche en proche la valeur des sorties lorsque l'on connaît les entrées, est décrit dans le cadre de la logique combinatoire. Pour ces circuits une même combinaison des entrées donnera toujours la même valeur des sorties.

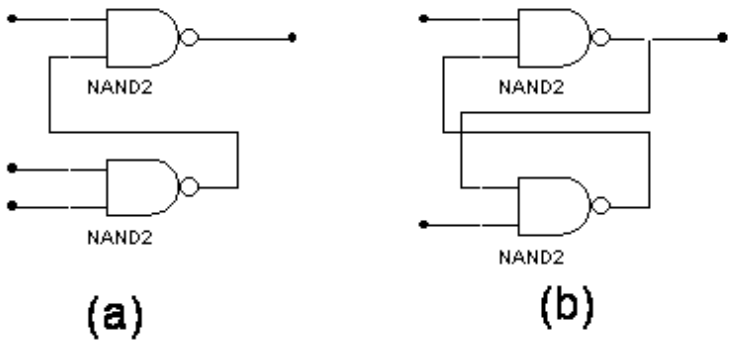


Figure 22 (a) circuit décrit par la logique combinatoire. (b) circuit décrit par la logique séquentielle

La situation est complètement différente lorsque la sortie d'une porte est rebouclée sur son entrée (Figure 22.b). Dans cette situation l'état de sortie du circuit à un instant donné dépend :

- De la valeur des entrées à cet instant

- De la valeur de la (ou des) sortie(s) aux instants antérieurs

Ces circuits sont évidemment parfaitement déterministes mais leur état présent est fixé par toute la séquence des entrées, c'est-à-dire par les valeurs qui ont précédées les valeurs actuelles. Une étude temporelle est donc indispensable pour en comprendre le fonctionnement. Ces circuits n'obéissent plus aux règles de la logique combinatoire. Il doivent être décrit et étudiés dans le cadre de la logique séquentielle. Le terme "séquentielle" fait ici référence à une succession d'événements dans le temps et s'applique aux systèmes où le temps joue un rôle à part entière pour la détermination de l'état de sortie.

Bien entendu, toutes les règles d'algèbre de Boole que nous avons vues précédemment s'appliqueront en logique séquentielle.

4.2. *Les bascules.*

Les bascules sont les éléments de base de la logique séquentielle, un peu comme l'étaient les portes logiques en logique combinatoire. Nous verrons par la suite qu'elles permettent de réaliser de nombreux systèmes (compteurs, registres, mémoires …) d'où leur importance.

De façon générale une bascule se caractérise par :

- L'existence de deux états de sortie stables. C'est à dire deux états dans lesquels la bascule peut se maintenir indéfiniment sans action extérieure.
- Des entrées de commande permettant de passer au choix d'un état à un autre.

Nous allons tout de suite donner l'exemple de la bascule la plus simple très connue sous le nom de bascule *RS*.

4.2.1. La bascule RS.

a. Schéma.

La bascule *RS* est la plus simple des bascules. Elle est réalisée à partir de deux portes NOR ou de deux portes NAND. Nous allons étudier en détail le fonctionnement de la bascule *RS* réalisée avec des portes NOR. Le schéma de cette bascule est donné sur la Figure 23. La bascule possède deux entrées notées *R* et *S* ainsi que deux sorties conventionnellement notées Q et $\overline{Q}$.

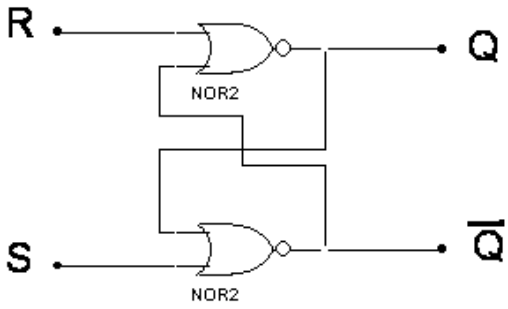


Figure 23 Bascule RS réalisée avec des portes NOR

b. Fonctionnement

La bascule étant constituée sur la base de portes NOR, dès que l'une de ses entrées est au niveau 0, les sorties Q et $\overline{Q}$ sont nécessairement complémentées. En effet, compte tenu de la table de vérité de la fonction NOR, on remarque que :

- la combinaison *S* = 1 et *R* = 0 impose $Q=1$ et $\overline{Q}=0$: L'entrée *S* = 1 place la sortie Q au niveau haut. Cette combinaison des entrées correspond au mode de fonctionnement SET.

- la combinaison *S* = 0 et R = 1 impose $Q=0$ et $\overline{Q}=1$: L'entrée *R* = 1 place la sortie *Q* au niveau bas. Cette combinaison des entrées correspond au mode de fonctionnement RESET.

La situation la plus intéressante est certainement *S* = 0, *R* = 0. Cette situation autorise en sortie aussi bien la combinaison $Q=1,\ \overline{Q}=0$ que la combinaison $Q=0,\ \overline{Q}=1$. Ces deux combinaisons correspondent aux deux états stables de la bascule, c'est-à-dire aux deux états dans lesquels la bascule se maintient indéfiniment tant que l'on ne change pas la valeur de ses entrées. La combinaison effectivement présente sur les sorties dépend de la combinaison des entrées qui a précédé la situation *S* = 0, *R* = 0. Cette situation antérieur était nécessairement du type S = 1, R = 0 ou S = 0, R = 1[6] et c'est la dernière entrée qui valait 1 qui impose l'état de la bascule selon les règles énoncées plus haut.

[6] La combinaison des entrées précédent S = 0, R = 0 ne peut pas être S = 1, R = 1. En pratique les deux entrées ne peuvent pas changer rigoureusement au même instant, l'une commute nécessairement avant l'autre.

Ainsi :

- $Q=1$, $\overline{Q}=0$ si les entrées étaient S = 1 et R = 0
- $Q=0$, $\overline{Q}=1$ si les entrées étaient S = 0 et R = 1

Lorsque S = 0, R =0 la bascule mémorise donc l'état associé à la précédente combinaison des entrées. La combinaison *S* = 0, *R* = 0 est ainsi appelée mode MEMOIRE.

Remarquons pour finir que la combinaison des entrées *S* = 1 et *R* = 1 ne donne pas des sorties complémentées. On trouve en effet $Q=\overline{Q}=0$. Bien que cette situation n'entraîne aucun dommage pour les circuits et qu'elle soit parfaitement réalisable en pratique, on a coutume de la désigner comme interdite pour une bascule RS.

c. Chronogramme

Le comportement de la bascule *RS* lors de changements successifs des entrées *R* et *S* est illustré sur le chronogramme suivant (Figure 24). Les flèches sur le dessin indiquent les relations de cause à effet entre les signaux. En pratique on trouvera souvent ce type d'informations dans les DATA BOOKS des constructeurs.

- A $t=t_0$ les entrées sont *R* = 0 et *S* = 0, et les sorties $Q=0$ et $\overline{Q}=1.$ La bascule est en mode MEMOIRE.
- A $t=t_1$: *R* = 0 et S effectue la transition $0\rightarrow1$. La bascule passe en mode SET et les sorties commutent en conséquence $Q:0\rightarrow1$ et $\overline{Q}:1\rightarrow0$.
- A $t=t_2$: *R* = 0 et S retourne au niveau bas $S:1\rightarrow0.$ Les sorties ne changent pas et restent $Q=1$ et $\overline{Q}=0$. La bascule à donc mémorisé le passage de l'entrée *S* au niveau 1.
- A $t=t_3$: l'entrée R effectue la transition $0\rightarrow1$ alors que *S* = 0. La bascule passe donc mode mémoire au mode RESET et les sorties commutent $Q:1\rightarrow0$ et $\overline{Q}:0\rightarrow1$.
- A $t=t_4$: L'entrée R revient au niveau bas $R:1\rightarrow0$ et $S=0$. La bascule revient alors en mode mémoire, mais cette fois la sortie mémorisée est $Q=0$ et $\overline{Q}=1.$

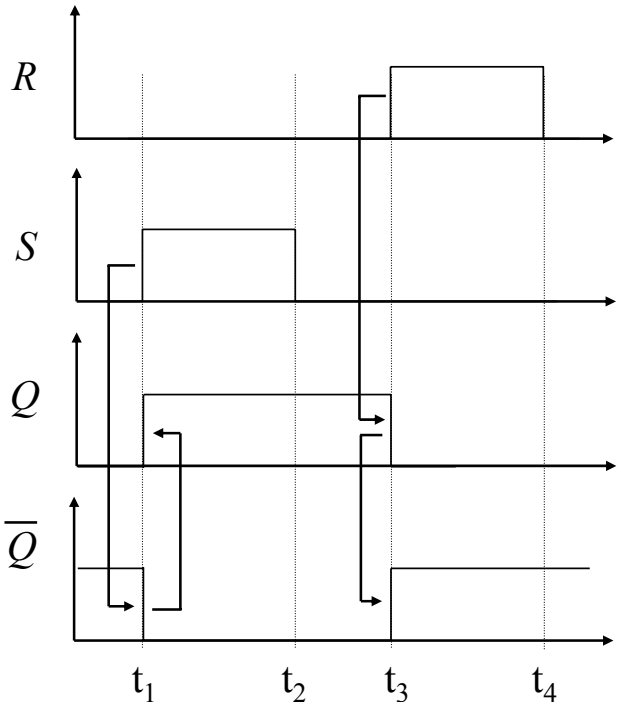


Figure 24 Fonctionnement de la bascule RS à base de portes NOR

d. Table de vérité et équation caractéristique.

On peut écrire une table de vérité pour cette bascule en faisant explicitement intervenir le temps. Comme dans les DATA BOOKS, on notera Q_0 la valeur de la sortie Q juste avant que la condition *S* = 0, *R* = 0 ne soit réalisée.

S	R	Q	$\overline{Q}$	
0	0	Q_0	$\overline{Q_0}$	Mémoire
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	« Interdit »

Cette table de vérité donne la valeur présente de la sortie Q en fonction de la valeur précédente Q_0.

L'équation caractéristique s'obtient à partir de la table de vérité précédente en

cherchant l'expression de Q en fonction de *R*, *S* et Q_0. Pour cela à partir de la table de Karnaugh :

	S	**0**	**0**	**1**	**1**
Q_0	***R***	**0**	**1**	**1**	**0**
0		0	0	0	1
1		1	0	0	1

Q

On obtient :

$$Q = S \bullet \overline{R} + Q_0 \bullet \overline{R} = \overline{R} \bullet (Q_0 + S)$$

e. Conclusion.

Plusieurs conclusions peuvent être tirées de l'étude précédente.

1. La bascule *RS* à base de portes NOR possède un mode mémoire *S* = *R* = 0 au quel sont associés deux états stables en sortie, soit $Q = 1$ et $\overline{Q} = 0$, soit $Q = 0$ et $\overline{Q} = 1$.

2. La transition $R : 0 \to 1$ avec *S* = 0 entraîne $Q \to 0$. Pour cette raison l'entrée *R* est appelée l'entrée RESET (de l'anglais to reset).

3. La transition $S \to 1$ avec *R* = 0 entraîne Q→1. Pour cette raison l'entrée *S* est appelée l'entrée SET (de l'anglais to set).

4. Les entrées *R* et *S* sont des entrées de commande permettant de passer d'un état stable à un autre.

5. Ce type de bascule présente au moins deux inconvénients:

a) Une faible immunité aux parasites, puisque toute variation des entrées (même non désirée) est prise en compte et est susceptible de modifier l'état des sorties.

b) La difficulté de connecter entre elles un grand nombre de bascules. En effet les bascules commutant chacune à leur rythme il est impossible de les synchroniser pour réaliser un système complexe.

Pour remédier, au moins en partie, à ces problèmes d'autres types de bascules ont été

développés.

4.2.2. La bascule RS avec validation (RS latch).

a. Schéma

Le schéma de principe d'une bascule *RS* avec validation est représenté sur la Figure 25. On reconnaît aisément la bascule *RS* dont les entrées sont maintenant notées *R'* et *S'* et sur lesquelles on a ajouté deux portes AND. L'entrée de validation *E* (pour Enable en anglais) permet de contrôler l'ouverture des 2 portes AND.

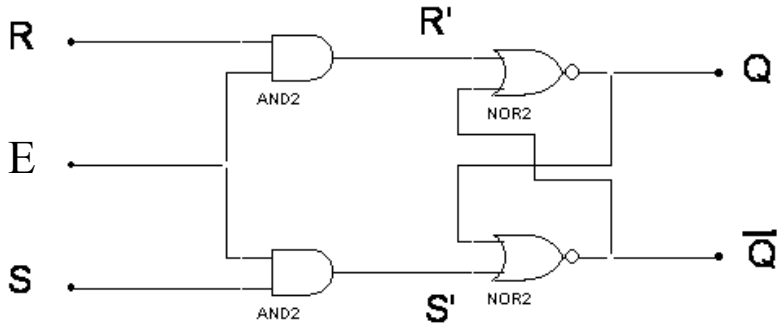


Figure 25 Schéma d'une bascule RS avec entrée de validation

b. Fonctionnement

Raisonnons en considérant les deux valeurs possibles du signal de validation.

- Lorsque *E* = 0 alors $R' = S' = 0$: La bascule RS est en mode mémoire et elle restera dans cet état quelles que soient les valeurs présentes sur les entrées *R* et *S*.
- Lorsque *E* = 1 on distingue deux cas
  - Si *R* = 1, *S* = 0 alors $R' = 1$, $S' = 0$ ce qui impose en sortie $Q = 0$ et $\overline{Q} = 1$ (mode reset).
  - Si *R* = 0, *S* = 1 alors $R' = 0$, $S' = 1$ ce qui impose en sortie $Q = 1$ et $\overline{Q} = 0$ (mode set).

On voit nettement sur cet exemple que le fonctionnement de la bascule *RS* avec validation est identique à celui de la bascule *RS* simple lorsque *E* = 1. En revanche,

lorsque *E* = 0 la bascule est bloquée dans l'état imposé par les valeurs de *R* et *S* au moment où *E* passe à 0. Dans cette situation les variations des entrées *R* et *S* n'affectent plus les sorties.

4.2.3. La bascule D.

a. Schéma.

Le logigramme de la bascule *D* est donné sur la Figure 26. Il s'agit en réalité d'une bascule *RS* avec validation (voir le paragraphe 4.2.2) sur laquelle on a ajouté un inverseur. La présence de cet inverseur entraîne obligatoirement $R' = \overline{S'}$ lorsque *E* = 1 si bien que dans cette situation la bascule n'est jamais en mode mémoire.

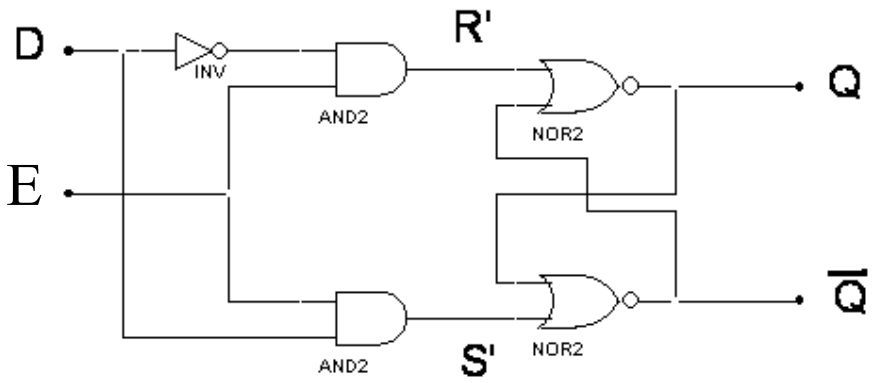


Figure 26 Schéma d'un bascule D

b. Fonctionnement.

Comme pour la bascule RS avec validation, nous allons raisonner sur les deux valeurs possibles de l'entrée de validation *E*.

- Lorsque *E* = 0 alors $R' = S' = 0.$ La bascule est en mode mémoire et les variations de l'entrée *D* n'affectent pas les sorties.
- Lorsque *E* = 1 alors $R' = \overline{D}$ et $S' = D$. Dans ces conditions la sortie Q recopie l'entrée *D*.

Le fonctionnement de la bascule *D* est illustré sur le chronogramme suivant (Figure

27). On suppose que l'état initial est caractérisé par *E* = 0 et $Q=0$.

- Pour $t < t_1$ Les variations de l'entrée *D* ne sont par répercutées sur les sorties. La bascule reste dans l'état d'équilibre $Q=0,\ \overline{Q}=1$.
- A l'instant $t = t_1$ l'entrée de validation *E* passe au niveau haut alors que *D* = 1. On a donc $S^{'} = 1$ et $R^{'} = 0$ et par conséquent $Q=1$ (mode set).
- A l'instant $t = t_2$ *D* passe au niveau bas 0, en conséquence $R^{'}$ passe à 1 et $S^{'}$ passe à 0 ce qui entraîne la commutation de la sortie Q vers le niveau bas $Q:1\rightarrow 0$ (mode reset).
- A l'instant $t = t_3$ *D* repasse à l'état haut, donc $R^{'}$ retourne à 0 et $S^{'}$ à 1. Par conséquent la sortie Q effectue la transition $Q:0\rightarrow 1$ (mode set). En analysant le chronogramme on constate que depuis l'instant $t = t_1$ la sortie Q a recopié l'entrée *D*.
- A l'instant $t = t_4$ l'entrée de validation *E* repasse à 0 ce qui entraîne de nouveau $R^{'} = S^{'} = 0.$ La bascule est en mode mémoire et elle mémorise la valeur présente sur la sortie Q au moment de la transition *E :1→ 0*, en l'occurrence $Q=1$.

c. Equation caractéristique.

L'équation caractéristique de la bascule *D* s'obtient facilement à partir de celle de la bascule *RS* en écrivant que $R^{'} = \overline{D}$ et $S^{'} = D.$ En effet :

$$Q = \overline{R^{'}} \bullet (S^{'} + Q_0) = D \bullet (D + Q_0)$$
$$Q = D \bullet Q_0 + D \bullet D = D \bullet (Q_0 + 1)$$
$$Q = D$$

4.2.4. Bascules synchrones / bascules asynchrones.

En logique séquentielle, on est amené à définir deux grands types de circuits : les circuits asynchrones et les circuits synchrones

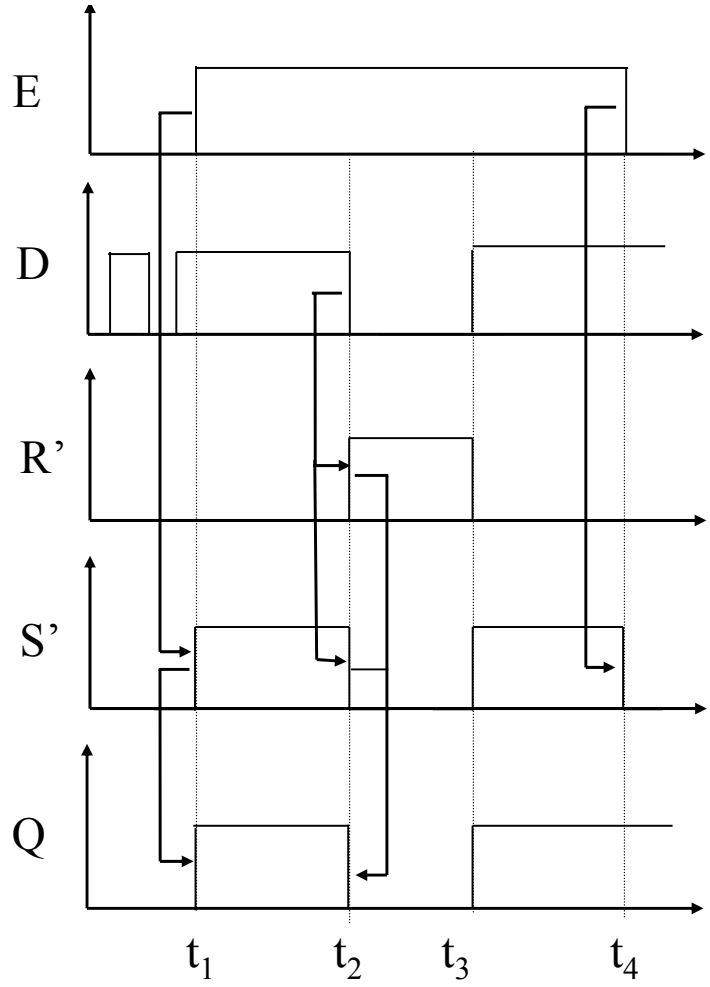


Figure 27 Fonctionnement d'une bascule D

- ***Les circuits asynchrones :***

Un système asynchrone est un système dont les entrées sont sensibles à des niveaux de tension (0 ou 1, bas ou haut …). Après un changement des entrées le circuit évolue librement jusqu'à ce qu'il atteigne un état stable. Les transitions d'un état à un autre se produisent donc à des instants quelconques que l'on ne peut pas contrôler.

Toutes les bascules que nous avons étudiées jusqu'à présent sont des systèmes asynchrones.

- ***Les circuits synchrones :***

Les circuits synchrones ont une de leurs entrées, généralement appelée entrée d'horloge (ou *CLK* pour clock), sensible à des impulsions[7], les autres entrées restant sensibles à des niveaux. Pour ces systèmes, quelle que soit la valeur des entrées, le passage d'un état à un autre ne se fait qu'au moment où l'entrée d'horloge reçoit une impulsion.

Les circuits synchrones avec horloge sont très répandus. En effet, dans de nombreux systèmes, les ordinateurs par exemple, les opérations doivent être parfaitement cadencées de manière à se produire dans un ordre bien déterminé. La synchronisation par une horloge commune est alors indispensable.

4.2.5. La structure maître-esclave.

Les bascules synchrones présentent, en outre, l'avantage d'être peu sensibles aux parasites. En effet, puisque seule compte la valeur des entrées au moment de l'impulsion d'horloge, toute variation intempestive des entrées entre deux impulsions d'horloge sera sans effet sur la bascule. Ces bascules sont basées sur une structure maître-esclave dont nous allons donner le principe.

Nous avons vu, dans le paragraphe 4.2.2, que le verrouillage d'une bascule, à l'aide d'une entrée de validation *E*, permettait de s'affranchir des parasites sur les entrées lorsque *E* = 0. Le problème subsiste néanmoins lorsque l'entrée de validation est active ($E = 1$). Pour résoudre le problème il faudrait, lorsque *E* = 1, que la sortie soit verrouillée, mais que les informations sur les entrées soient prises en compte, donc que les entrées soient ouvertes. A l'inverse, lorsque *E* = 0 les entrées devraient être verrouillées et les sorties ouvertes. La solution consiste à utiliser deux bascules à verrouillage, l'une pour les entrées l'autre pour la sortie. Lorsque l'enregistrement est commandé, le système de sortie doit être bloqué ; lorsque la commande d'affichage est active, le système d'entrée doit être verrouillé. Les commandes de validation pourront ainsi être des signaux complémentaires. Ce type de structure est schématisé sur la Figure 28.

Une telle structure est dite maître-esclave ; la bascule d'entrée est le maître, la bascule de sortie, qui recopie l'état du maître, est l'esclave. La plupart des bascules synchrones sont basées sur ce type de structure.

[7] En électronique numérique le terme impulsion désigne le front de montée ou de descente, d'un signal rectangulaire "très raide" mais pas nécessairement très court.

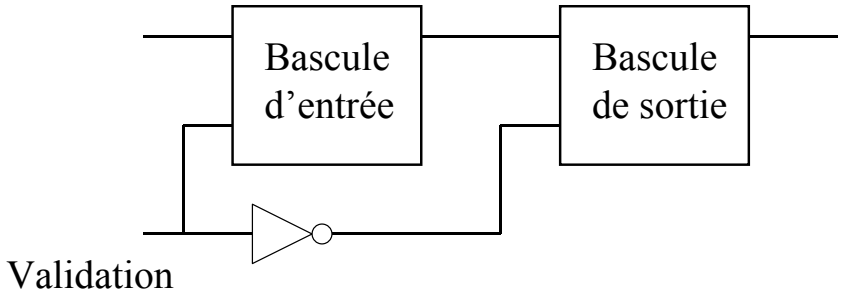


Figure 28 Principe de la structure maître-esclave

4.2.6. Un exemple détaillé de bascule synchrone : la bascule D

Avant de voir les différents types de bascules synchrones disponibles dans les catalogues, nous allons analyser en détail la réalisation d'une bascule D synchrone a partir de deux bascules *RS* asynchrones. La bascule D synchrone est une bascule dont la sortie Q recopie la valeur de l'entrée D à chaque impulsion d'horloge.

a. Schéma.

Le schéma de principe d'une bascule D synchrone réalisée à partir de deux bascules *RS* asynchrone à base de portes NOR est représenté sur la Figure 29. Pour alléger le schéma les deux bascules RS asynchrones sont représentées sous forme de blocs. Chacun de ces blocs est identique à la bascules RS décrite dans le paragraphe 4.2.1

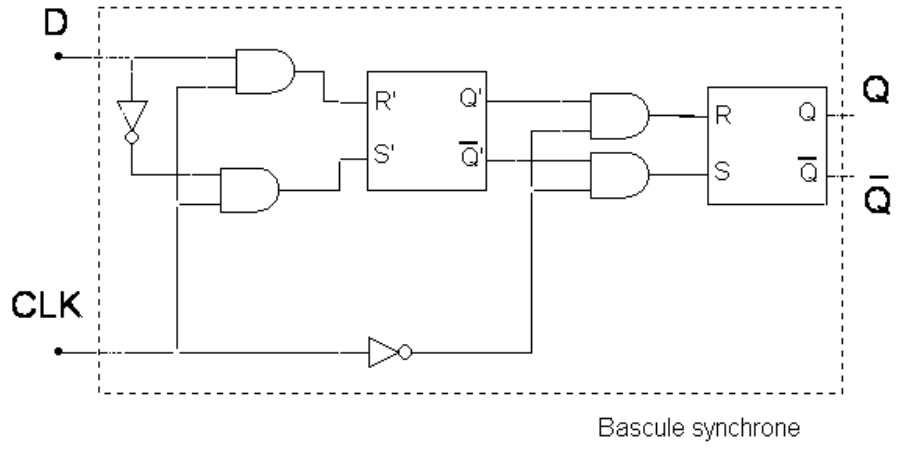


Figure 29 Schéma dune bascule D synchrone basée sur deux bascules RS asynchrones

b. Fonctionnement.

A partir du schéma précédent, on établit facilement les expressions logiques donnant la valeur des entrées des deux bascules:

$$R' = CLK \bullet D$$
$$S' = CLK \bullet \overline{D}$$
$$R = \overline{CLK} \bullet Q'$$
$$S = \overline{CLK} \bullet \overline{Q'}$$

On constate ainsi que si *CLK* = 0, la première bascule est en mode mémoire ($R' = S' = 0$) tandis que la seconde, qui est en mode d'écriture (set ou reset $R = Q'$ et $S = \overline{Q'}$), recopie les sorties de la première bascule ($Q = S = \overline{Q'}$).

On a bien entendu la situation inverse lorsque *CLK* = 1. La première bascule est en mode d'écriture (set ou reset avec $R' = D$ et $S' = \overline{D}$) et recopie l'entrée *D*, alors que la seconde bascule est en mode mémoire ($R = S = 0$). Ce fonctionnement est illustré sur le chronogramme représenté sur la Figure 30

- Lorsque $t_0 < t < t_1.$ l'entrée CLK est au niveau haut (*CLK* = 1) et la bascule d'entrée est active ($R' = D, \quad S' = \overline{D}$) et elle recopie la valeur de l'entrée *D* ($Q' = \overline{D}, \quad \overline{Q'} = D$). A l'inverse la bascule de sortie est en mode mémoire ($R = S = 0$).

- Dans l'intervalle $t_1 < t < t_2$ l'entrée *CLK* est au niveau bas (*CLK* = *0*) la bascule d'entrée passe en mode mémoire. Les sorties Q' et $\overline{Q'}$ gardent en mémoire les valeurs de $\overline{D}$ et *D* présentent au moment de la transition $CLK : 1 \to 0$ (front descendant sur *CLK*). Ainsi $Q' = \overline{D(t_1)}$ et $\overline{Q'} = D(t_1)$). En revanche la seconde bascule est en mode d'écriture avec $R = Q'$ et $S = \overline{Q'}$. Elle recopie l'état de la bascule maître ($Q = \overline{Q'} = D(t_1)$).

Lorsque $t_2 < t < t_3.$ alors *CLK* = 1, la bascule d'entrée est active avec $R' = D$ et $S' = \overline{D}$. Elle recopie la valeur présente sur l'entrée *D* ($Q' = \overline{D}$, $\overline{Q'} = D$). La bascule

esclave est en mode mémoire et ses sorties ne changent pas ($Q = D(t_1)$).

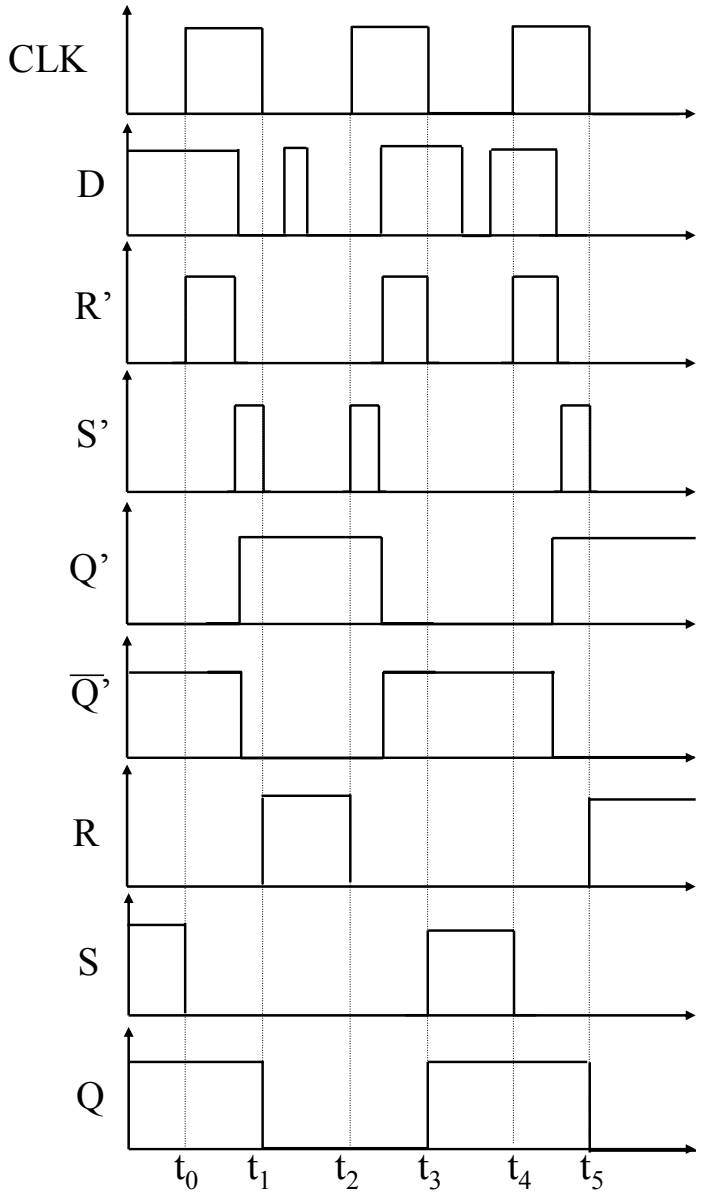


Figure 30 Chronogramme illustrant le fonctionnement d'une bascule D synchrone active sur front descendant.

- Dans l'intervalle $t_3 < t < t_4$. *CLK* = 0. La bascule maître repasse en mode mémoire. En revanche la seconde bascule commute en mode d'écriture et recopie

l'état de la bascule maître ($Q = \overline{Q'} = D(t_3)$).

Lorsque $t_4 < t < t_5$. l'entrée *CLK* est au niveau haut (*CLK*=1) la bascule maître est active et recopie la valeur de l'entrée *D*. La bascule esclave est en mode mémoire et ses sorties ne changent pas ($Q = D(t_3)$).

A chaque impulsion d'horloge (ici chaque front descendant) la sortie Q recopie la valeur de l'entrée *D* à cet instant. Le fonctionnement de cette bascule *D* synchrone est donc identique à celui de sa consoeur asynchrone si ce n'est qu'elle ne prend en compte les valeurs de *D* qu'au moment de l'impulsion d'horloge. Toutes les variations de *D* entre deux impulsions d'horloge sont donc sans effet sur la sortie.

4.2.7. Représentations des bascules synchrones.

Les principales bascules synchrones disponibles dans le commerce sont :

- Les bascules *RS* de table de vérité identique à celle de leurs homologues asynchrone à base de NOR.
- Les bascules *D*
- Les bascules *JK*, identiques aux bascules *RS* pour les entrées autres que 11 (l'entrée *J* correspond à *S* et l'entrée *K* à *R*). Pour les entrées 11 ces bascules fonctionnent en mode Toggle à savoir que leur sortie Q change de valeur à chaque impulsion d'horloge. Par rapport aux bascules *RS*, les bascules *JK* permettent d'utiliser toutes les combinaisons des entrées.

On a représenté sur la Figure 31 les symboles utilisés pour chacune de ces bascules.

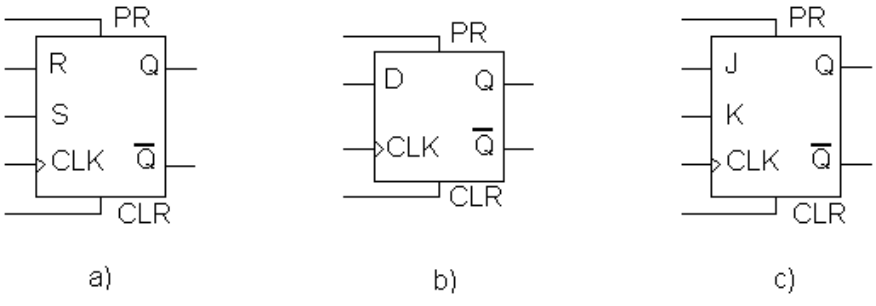


Figure 31 Représentation des principales bascules synchrones. a) la bascule RS, b) la bascule D, c) la bascule JK

On note la présence de deux entrées *PR* (Preset) et *CLR* (Clear) qui imposent respectivement $Q=1$ et $Q=0$ lorsque elles sont actives. Ces deux entrées sont prioritaires sur toutes les autres entrées. Selon les bascules, ces deux entrées peuvent de nature asynchrone, leur effet est "immédiat", ou synchrone, leur action sur les sorties est répercutée au moment de l'impulsion d'horloge. Lorsque les entrées *PR* et *CLR* sont actives au niveau bas, elles sont notées $\overline{PR}$ et $\overline{CLR}$ dans les tables de vérité et sur les symboles des bascules.

Signalons enfin qu'il existe également des bascules synchrones actives sur le front descendant du signal d'horloge. Elles se distinguent de leur consoeurs actives sur front montant par un cercle placé sur l'entrée d'horloge.

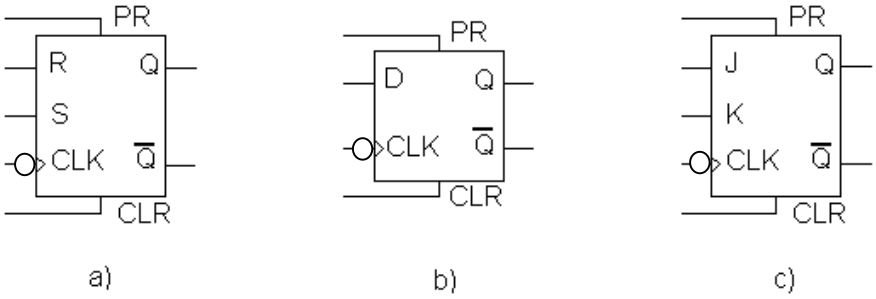


Figure 32 Représentation des principales bascules synchrones actives sur front descendant. a) la bascule *RS*, b) la bascule *D*, c) la bascule *JK*

4.2.8. Tables de vérités et tables des commandes.

Nous donnons ci-dessous les tables de vérité des bascules *RS*, *D* et *JK*, avec pour chacune d'entre elle la table des commandes[8] pour une transition donnée. La table des commandes est en fait une autre représentation de la table de vérité, plus facile à utiliser lors de la synthèse d'une fonction séquentielle. Elle indique quelles doivent être les valeurs des entrées pour obtenir, au moment de l'impulsion d'horloge une transition donnée sur la sortie.

Pour écrire les tables de vérité nous adoptons les notations suivantes :

[8] Dans la littérature ces tables sont parfois appelées table de fonctionnement ou tables d'excitation.

- R_n, S_n, D_n, J_n, K_n, Q_n et $\overline{Q}_n$ représentent les valeurs des entrées et des sorties juste avant le n^{éme} front actif de l'horloge CLK. Selon les bascules il peut s'agir soit d'un front montant ↑, soit d'un front descendant ↓

- Q_{n+1} et $\overline{Q}_{n+1}$ représentent les valeurs des sorties juste après le n^{éme} front actif de l'horloge *CLK*.

Ces tables de vérité ne font pas apparaître les entrées asynchrones *PR* et *CLR* qui, rappelons le, peuvent être asynchrones et prioritaires sur les autres entrées. On notera également que l'entrée horloge ne figure pas explicitement dans les tables de vérité. Il ne faut cependant pas oublier que dans un circuit synchrone les transitions s'effectuent uniquement au moment ou l'entrée d'horloge (CLK) reçoit une impulsion. Toute variation des entrées entre deux impulsions est sans effet sur les sorties

a. Bascule RS.

Les tables de vérité et de commandes s'écrivent respectivement :

S_n	R_n	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	Q_n	$\overline{Q_n}$	Mémoire
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	« Interdit »

et

$Q_n \to Q_{n+1}$	S_n	R_n
$0 \to 0$	0	φ
$0 \to 1$	1	0
$1 \to 0$	0	1
$1 \to 1$	φ	0

La table de transition se lit de la façon suivante. La ligne 1 indique que la sortie Q restera à 0 après l'impulsion d'horloge si $S_n = R_n = 0$ ou si $S_n = 0$ et $R_n = 1$ avant l'impulsion d'horloge. La ligne 2 indique que la sortie Q passera de 0 à 1 si $S_n = 1$ et $R_n = 0$ juste avant l'impulsion d'horloge.

L'équation caractéristique s'obtient facilement en écrivant la table de Karnaugh :

	S_n	0	0	1	1
Q_n	R_n	0	1	1	0
0		0	0	0	1
1		1	0	0	1

Q_{n+1}

Il vient ainsi

$$Q_{n+1} = S_n \bullet \overline{R_n} + \overline{R_n} \bullet Q_n .$$

- ***Bascule D.***

Les tables de vérité et de commandes s'écrivent respectivement :

D	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	1
1	1	0

et

$Q_n \to Q_{n+1}$	D_n
$0 \to 0$	0
$0 \to 1$	1
$1 \to 0$	0
$1 \to 1$	1

Ces deux tables sont bien entendu évidentes pour une bascule *D* et on retrouve sans peine l'équation caractéristique de ce type de bascule :

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

Bascule JK.

Les tables de vérité et de commandes s'écrivent respectivement :

J_n	K_n	Q_{n+1}	$\overline{Q_{n+1}}$	
0	0	Q_n	$\overline{Q_n}$	Mémoire
0	1	0	1	Reset
1	0	1	0	Set
1	1	$\overline{Q_n}$	Q_n	Toggle

et

$Q_n \rightarrow Q_{n+1}$	J_n	K_n
$0 \rightarrow 0$	0	φ
$0 \rightarrow 1$	1	φ
$1 \rightarrow 0$	φ	1
$1 \rightarrow 1$	φ	0

Par rapport à la bascule *RS* l'entrée *J* joue le rôle de S (mode Set) et l'entrée *K* celui de *R* (mode Reset). On remarque également qu'il n'y a plus de ligne indéterminée. Lorsque les entrées sont toutes les deux au niveau haut la bascule fonctionne en mode Toggle. La table des commandes de la bascule *JK* contient une valeur indéterminée à chaque ligne. Il est donc possible, en principe, d'imposer *J* = *K* et de se ramener à une seule variable. On perd alors le bénéfice de simplifications supplémentaires lors de la phase de synthèse d'un système séquentiel.

L'équation caractéristique de la bascule *JK* s'obtient en écrivant la table de Karnaugh:

	J_n	**0**	**0**	**1**	**1**
Q_n	K_n	**0**	**1**	**1**	**0**
0		0	0	1	1
1		1	0	0	1

Q_{n+1}

Ce qui donne après simplification:

$$Q_{n+1} = J_n \bullet \overline{Q_n} + \overline{K_n} \bullet Q_n$$

Paramètres dynamiques des bascules.

En complément de toutes les informations concernant la fonction logique des différentes bascules (table de vérité …), les documentations techniques contiennent également de nombreuses informations sur les paramètres dynamiques des bascules (fréquence maximum des impulsions, temps de commutation, temps de propagation de l'information

…). Ces paramètres ne doivent pas être ignorés lorsque l'on souhaite réaliser un circuit séquentiel complexe nécessitant plusieurs bascules. Nous reviendrons sur ce point dans le polycopié de 2eme année.

4.3. *Exercices.*

Exercice 1

Le montage de la Figure 33 représente une bascule réalisée à partir de portes NAND.

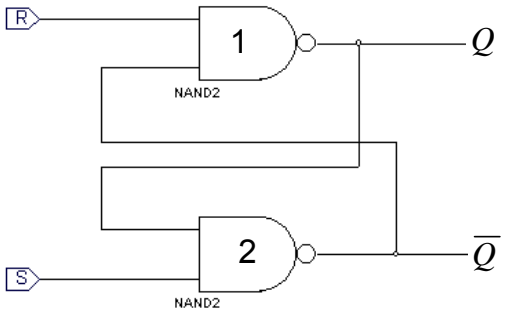


Figure 33

1. Que valent les sorties Q et $\overline{Q}$ dans les deux situations suivantes: R = 1, S = 0 et R = 0, S = 1?

2. Le circuit est dans un état définit par R = 1, S = 0, $Q = 0$, $\overline{Q} = 1$. Quelle est l'évolution des sorties lorsque l'entrée S commute vers le niveau haut (R = 1, $S : 0 \rightarrow 1$)?

3. Le circuit est dans un état définit par R = 0, S = 1, $Q = 1$, $\overline{Q} = 0$. Quelle est l'évolution des sorties lorsque l'entrée R commute vers le niveau haut (S = 1, $R : 0 \rightarrow 1$)?

4. donner la table de vérité décrivant le fonctionnement de ce circuit.

Autocorrection

Exercice 2

On considère le montage de la Figure 34 réalisé avec une bascule D synchrone active sur les fronts montants du signal d'horloge.

1. Donner les expressions des sorties A et B.

2. Compléter le chronogramme de la Figure 35.

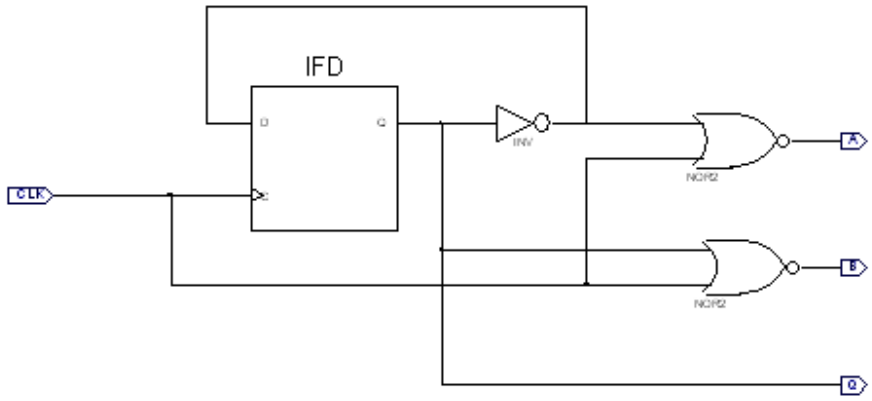


Figure 34

Figure 35

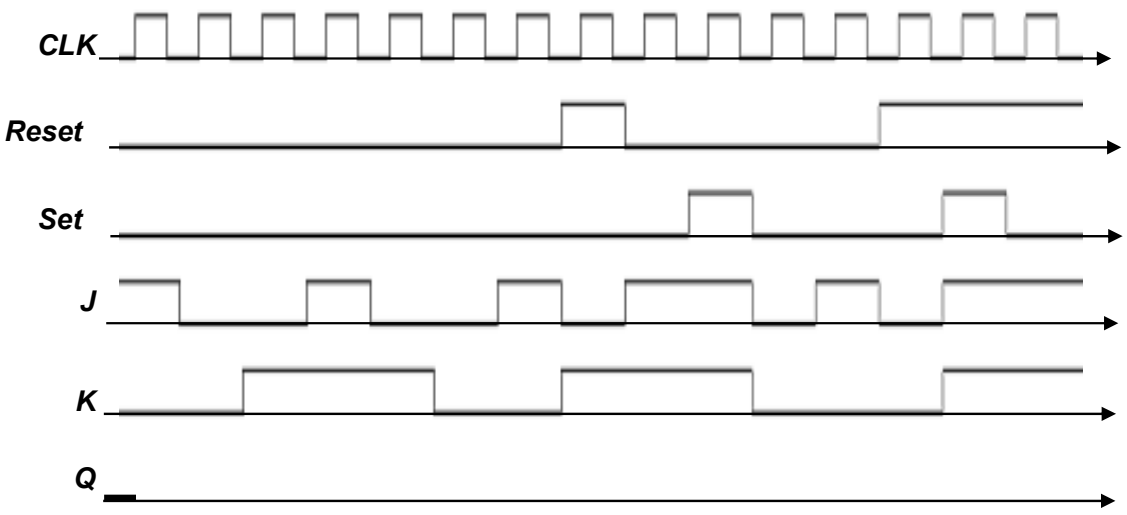
Autocorrection

Exercice 3

Soit une bascule JK dont la table de vérité est donnée ci-dessous. Les entrées Clear et Preset sont synchrones et notées respectivement R et S

Inputs						Outputs
R	S	CE	J	K	C	Q
1	X	X	X	X	↑	0
0	1	X	X	X	↑	1
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	1	↑	Toggle
0	0	1	1	0	↑	1

Compléter le chronogramme suivant (pour CE =1) :



Autocorrection

Exercice 4

On considère le schéma de la figure ci-dessous réalisé avec des bascules RS asynchrones à base de portes **NOR**.

1. Donner les expressions logiques des entrées R1, S1, R2 et S2 des deux bascules.
2. Rappeler la table de vérité et le fonctionnement d'une bascule RS asynchrone à base de portes **NOR**.
3. Que peut-on dire des bascules 1 et 2 lorsque E = 0?
4. Même question lorsque E = 1.
5. Compléter le chronogramme de la Figure 37.
6. Quelle est la fonction réalisée par ce montage?

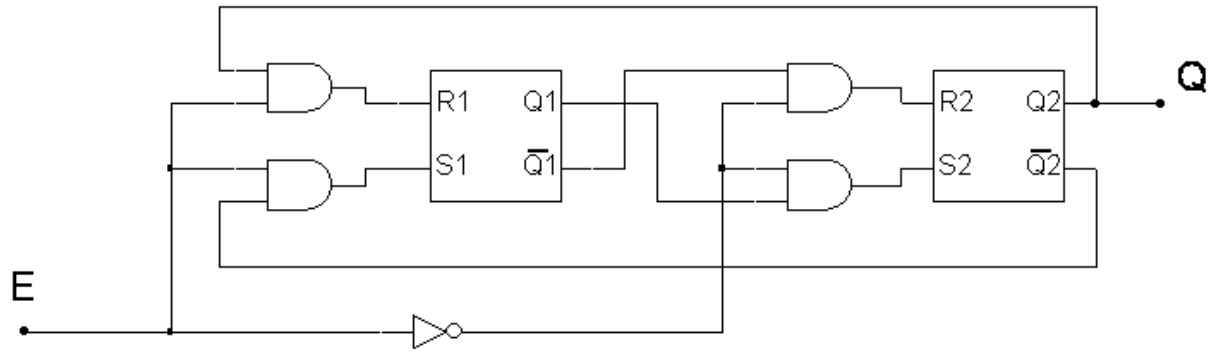


Figure 36

Autocorrection

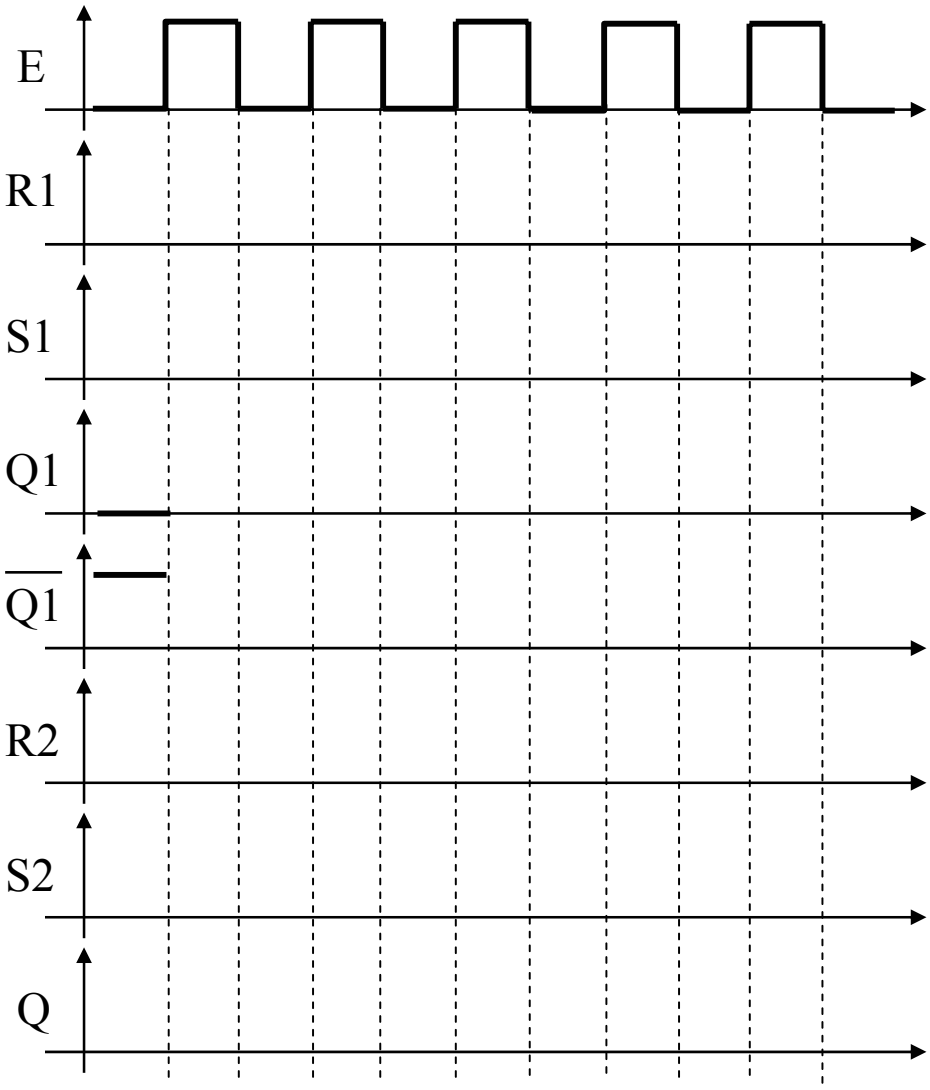


Figure 37

4.4. *Correction des exercices*

Exercice 1

1. L'entrée S = 0 sur la porte NAND n°2 impose $\overline{Q}=1$. Les entrées de la porte NAND n°1 étant toutes les deux au niveau haut on a $Q=0$. Cette situation correspond au mode Set de la bascule RS

L'entrée R = 0 sur la porte NAND n°1 impose $Q=1$. Les entrées de la porte NAND n°2 étant toutes les deux au niveau haut on a $\overline{Q}=0$. Cette situation correspond au mode Reset de la bascule RS

2. Le circuit est dans un état définit par R = 1, S = 0, $Q=0$, $\overline{Q}=1$. Lorsque l'entrée S commute vers le niveau haut $S:0\to 1$ la sortie de la porte NAND n°2 ne change pas puisque l'une des deux entrées reste a 0. Les sorties restent donc .

3. Le circuit est dans un état définit par R = 0, S = 1, $Q=1$, $\overline{Q}=0$. Lorsque l'entrée R commute vers le niveau haut $R:0\to 1$ la sortie de la porte NAND n°1 ne change pas puisque la seconde entrée reste a 0. Les sorties restent donc $Q=1,\ \overline{Q}=0$. Comme précédemment la bascule à garder en mémoire les sorties correspondant à la configuration antérieure des entrées.

4.

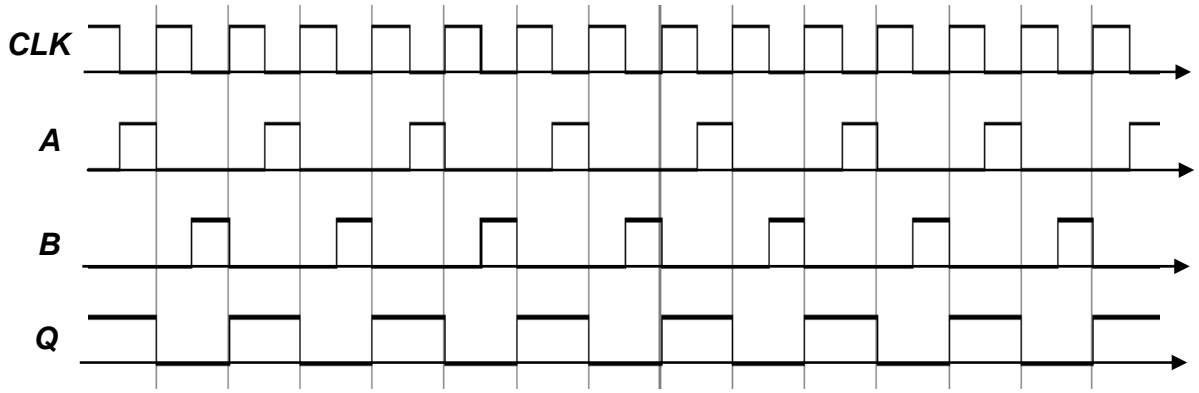
S	R	Q	$\overline{Q}$	
1	1	Q_0	$\overline{Q_0}$	Mémoire
0	1	0	1	Reset
1	0	1	0	Set
0	0	1	1	

Autocorrection

Exercice 2

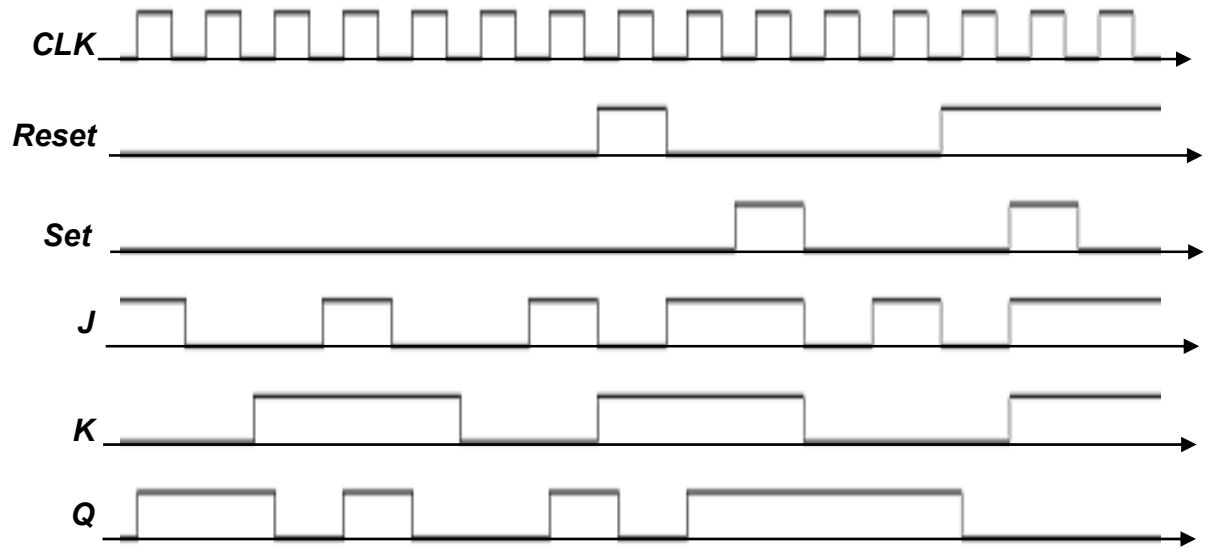
1. Directement à partir du schéma on lit $A=\overline{\overline{Q}+CLK}$ et $B=\overline{Q+CLK}$

2. La bascule D synchrone recopie sur sa sortie la valeur présente sur son entrée au moment du front montant sur le signal d'horloge *CLK*



Exercice 3

La bascule est active sur fronts montants. L'entrée R (Reset) est prioritaire sur toutes les autres. Le chronogramme s'obtient directement à partir de la table de vérité.



Exercice 4

1. Il s'agit d'une bascule maître esclave. Sur le schéma on lit directement

$$R_1 = E \bullet Q_2 \qquad R_2 = \overline{E} \bullet \overline{Q_1}$$

$$S_1 = E \bullet \overline{Q}_2 \qquad S_2 = \overline{E} \bullet Q_1$$

2. La table de vérité est donnée page 77

S	R	Q	$\overline{Q}$	
0	0	Q_0	$\overline{Q_0}$	Mémoire
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	« Interdit »

3. Lorsque E = 0 on a $R_1 = S_1 = 0$ et $R_2 = \overline{Q_1}$, $S_2 = Q_1$ la bascule 1 (maître) est en mode mémoire alors que la bascule 2 (esclave) est en mode écriture (set ou reset). La bascule esclave recopie les sorties de la bascule maître.

4. Lorsque E = 1 on a $R_2 = S_2 = 0$ et $R_1 = Q_2$, $S_1 = \overline{Q_2}$ la bascule 2 (l'esclave) est en mode mémoire alors que la bascule 1 (le maître) est en mode écriture (set ou reset). La bascule maître enregistre les variations de l'entrée E sans que les sorties de la bascule esclave changent.

5.

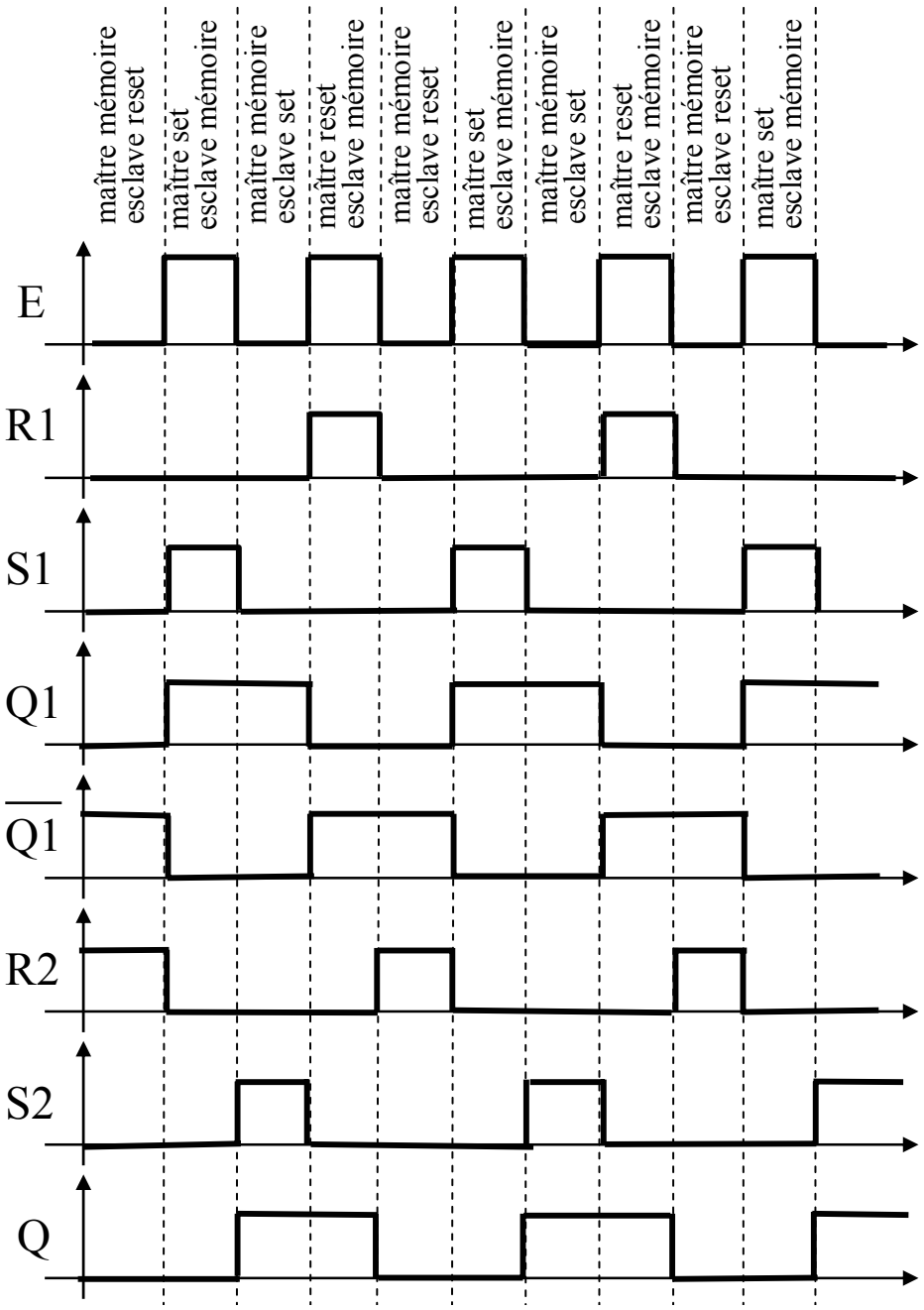


Figure 38

6. La sortie Q change de valeur à chaque transition $1 \rightarrow 0$ sur l'entrée E: Il s'agit d'une bascule T active sur front descendant. On peut également voir ce système comme un diviseur de fréquence par 2 (la fréquence du signal sur la sortie Q est deux fois plus petite que sur l'entrée E).

Chapitre 5

Compteurs, registres et mémoires.

Dans ce chapitre nous allons combiner entre elles plusieurs bascules synchrones pour réaliser des fonctions logiques plus évoluées. Nous nous attarderons sur trois types de circuits numériques extrèmement répandus : les compteurs, les registres et les mémoires.

5.1. *Généralités sur les compteurs.*

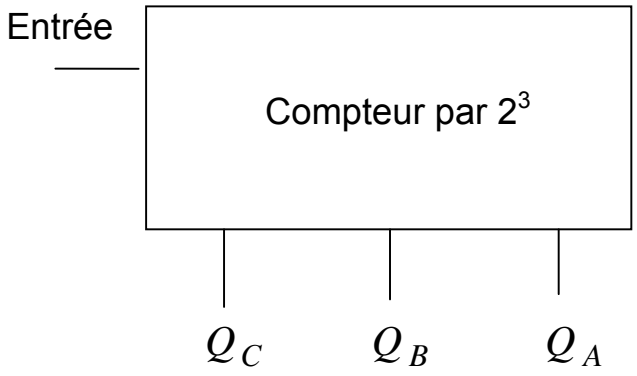
Les compteurs sont des systèmes séquentiels très utilisés. Leur fonction est de coder, dans un système de numérotation approprié, le nombre d'impulsions en un point d'un circuit.

5.1.1. Compteurs binaires.

Un compteur binaire par 2^n compte modulo 2^n les impulsions arrivant sur son entrée, et présente le résultat en binaire naturel sur ses n sorties. Un tel compteur possède donc au minimum une entrée synchrone, qui reçoit les impulsions, et n sorties notées $Q_{n-1}\,Q_{n-2}\,...Q_0$. Sa capacité de comptage est ainsi comprise entre 0 et $2^n - 1$. Considéré comme un système synchrone, un compteur binaire est, par définition, un circuit séquentiel à 2^n états. Chaque état correspond à une combinaison particulière des sorties et le compteur évolue cycliquement d'un état à un autre au fur et à mesure des impulsions qu'il reçoit.

Le tableau suivant représente la table de transition d'un compteur binaire par 8 ($n=3$). Comme précédemment l'état n est l'état juste avant l'impulsion (état présent) et l'état $n + 1$ celui juste après l'impulsion (état futur). Le compteur possède trois bits de sortie Q_A , Q_B et Q_C affectés respectivement des poids 2^0, 2^1 et 2^2. Le bit Q_A est donc bit de poids

faible (LSB) et Q_C celui de poids fort (MSB).



Etat n			Etat n+1		
Q_C	Q_B	Q_A	Q_C	Q_B	Q_A
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

5.1.2. Réalisation d'un compteur binaire.

La réalisation d'un compteur binaire repose sur les constations suivantes que l'on peut facilement vérifiées sur la table de transition ci dessus:

1. Le bit de poids faible commute à chaque impulsion d'horloge

2. Un bit de sortie Q_i commute ($0 \rightarrow 1$ou $1 \rightarrow 0$) si, et seulement si, tous les bits de poids plus faible sont au niveau 1.

3. Un bit de sortie Q_i commute ($0 \rightarrow 1$ ou $1 \rightarrow 0$) si, et seulement si, le bit de poids immédiatement inférieur passe de 1 à 0.

Les points 2 et 3 sont équivalents mais conduisent à la réalisation de deux types de compteurs différents ayant chacun leurs avantages et leurs inconvénients.

Quelle que soit l'hypothèse retenue pour sa réalisation, un compteur est constitué de cellules élémentaires commutant, ou pas, à chaque impulsion d'horloge. Chaque bit de comptage correspond en pratique à une bascule *T* réalisée soit avec une bascule *JK* dont les deux entrées sont à 1 (*J* = *K* = 1), soit avec une bascule *D* synchrone dont la sortie $\overline{Q}$ est rebouclée sur l'entrée *D* (Figure 39).

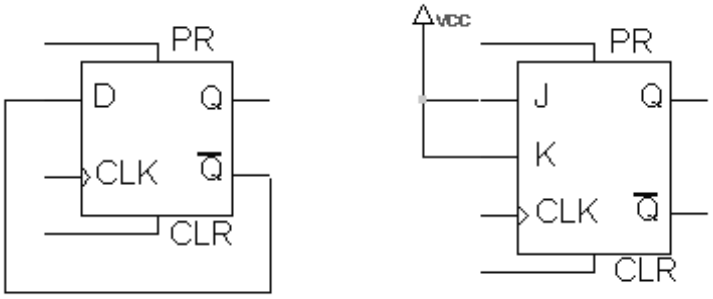


Figure 39 Bascules T

5.1.3. Compteur synchrone / compteur asynchrone.

Précisons immédiatement que, au sens ou nous l'avons entendu jusqu'à présent, tous les compteurs sont des systèmes synchrones puisqu'ils ne changent d'état qu'au moment ou ils reçoivent une impulsion. Pour les compteurs les termes synchrone et asynchrone ont des significations différentes que nous allons maintenant expliquer.

- Lorsqu'un compteur est basé sur le principe qu'un bit de sortie commute ($0 \rightarrow 1$ ou $1 \rightarrow 0$) si, et seulement si, tous les bits de poids plus faible sont au niveau 1, il est qualifié de synchrone. En effet, supposons que les *p* premiers bits de poids faible soient à 1 et que le bit de rang *p*+1 soit à 0. Au moment de l'impulsion d'horloge les *p* premiers bits passent simultanément à 0 et le bit *p*+1 à 1. Les p+1 premiers bits du compteur commutent donc au même moment[9]. D'où le qualificatif de synchrone.

- Lorsqu'un compteur est basé sur l'idée qu'un bit de sortie commute ($0 \rightarrow 1$ ou $1 \rightarrow 0$) si, et seulement si, le bit de poids immédiatement inférieur passe de 1 à 0, il est qualifié d'asynchrone. Supposons toujours que les *p* premiers bits soient à 1 et le bit suivant à 0. Au moment de l'impulsion d'horloge le premier bit commute, passant de 1 à 0 ; il entraîne alors la commutation du second bit et ainsi de suite. Cette cascade s'arrête au bit de rang *p*+1 qui passe de 0 à 1 avec un retard, par rapport à l'impulsion initiale, égal à *p*+1 fois le temps de commutation d'une bascule élémentaire. Dans cette approche les bascules commutent les unes après les autres d'où le qualificatif de compteur asynchrone.

[9] Aux différences de temps de commutation des bascules près.

5.1.4. Compteurs à cycle incomplet ou non binaire.

En pratique on a souvent besoin d'un compteur par *N* avec *N* différent de 2^n, par exemple pour compter jusqu'à 10! Il arrive également que l'on ait besoin de compteur affichant le résultat dans d'autres codes que le code binaire naturel. Dans ces situations les solutions envisageables sont en autres:

- Une synthèse complète du compteur si le code utilisé est différent du code binaire et que les règles de commutation énoncées au paragraphe 5.1.2 ne sont plus applicables.
- Le décodage de la valeur $N-1$ (la valeur la plus élevée affichée par le compteur) puis la rétroaction sur les entrées CLEAR des bascules à l'impulsion d'horloge suivante. Cette solution n'est bien sûr valable que pour le code binaire naturel.

5.2. *Les compteurs asynchrones.*

5.2.1. Les compteurs binaires.

Dans un compteur asynchrone le bit de poids *p* commute ($0\rightarrow1$ ou $1\rightarrow0$) si, et seulement si, le bit de poids *p – 1* passe du niveau haut au niveau bas ($1\rightarrow0$). a partir de ce constat on aboutit à un schéma très simple. Le compteur est réalisé avec de bascules T déclenchées sur fronts descendants (negative edge triggered). L'entrée d'horloge de chaque cellule est alimentée par la sortie *Q* de la bascule précédente sauf pour la première qui est reliée directement à l'entrée du compteur.

On a représenté sur la Figure 40 le schéma d'un compteur par 16 standard (référence 7493). Ces compteurs sont simples et leur mise en série est immédiate. Ils sont constitués d'un compteur par 2 (entrée *A*, sortie Q_A) et d'un compteur par 8 (entrée *B*, sorties $Q_B\ Q_C\ Q_D$) dont la mise en série ($B = Q_A$) donne un compteur par 16. De la même façon, on obtient un compteur par 256 en connectant la sortie Q_D sur l'entrée *A* d'un second compteur par 16 identique.

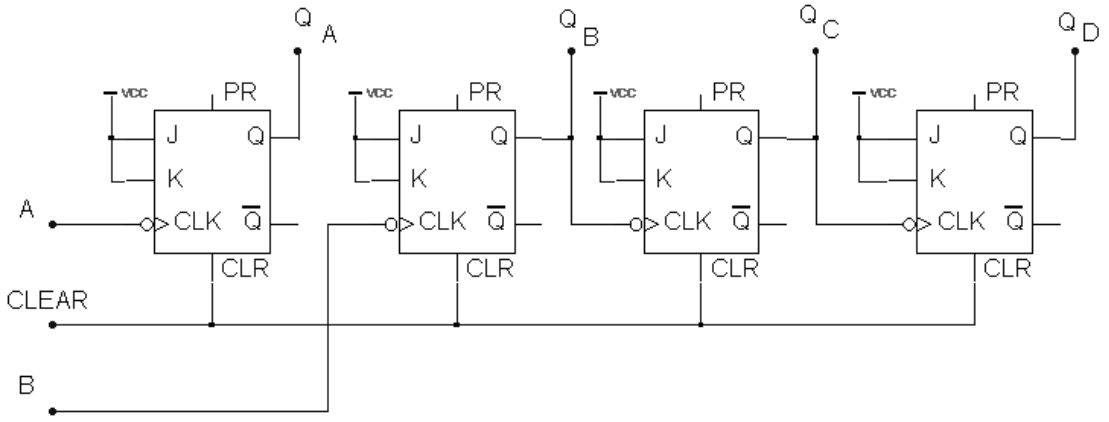


Figure 40 Compteur asynchrone par 16 de référence 7493. Le comptage par 16 est obtenu en connectant la sortie du compteur par 2 (Q_A) avec l'entrée *B* du compteur par 8.

Les compteurs binaires asynchrones présentent certains avantages. Ils sont en particulier faciles à mettre en série et permettent aisément d'atteindre de grandes capacités de comptage. Ils présentent néanmoins plusieurs inconvénients :

- Les bits commutant les uns après les autres, sur un compteur de grande capacité le bit de poids fort peut commuter "bien longtemps" après le bit de poids faible. Ce décalage peut être source de problèmes.
- L'inconvénient principal réside dans la succession d'états transitoires lors du basculement en cascade des cellules élémentaires. Par exemple le passage de 7 (0111) à 8 (1000) se fait par l'intermédiaire de 6 (0110) puis de 4 (0100). Bien que ces états transitoires ne soient présents qu'un faible laps de temps, ils peuvent entraîner des aléas dans le fonctionnement du système et limitent la fréquence maximum à laquelle le compteur peut compter les impulsions (typiquement $f_{max} \sim 16\,MHz$ dans le cas du compteur 7493 en technologie TTL).

5.2.2. Les compteurs asynchrones par 10.

Pour illustrer le fonctionnement des compteurs par 10 nous allons étudier en détail le compteur par 10 de référence 7490.

a. Fonctionnement.

Ces compteurs, très utilisés à cause du code *BCD*, sont en général constitués d'un compteur par 2 (entrée *A,* sortie Q_A) et d'un compteur par 5 (entrée *B*, sorties $Q_D Q_C Q_B$) dont le cycle de comptage est tronqué[10]. Ce dernier compteur compte suivant le cycle 0, 1, 2, 3, 4, 0, 1… Le premier étage Q_B (LSB) et le dernier étage Q_D (MSB) ont un donc un fonctionnement particulier. En effet, après l'état $Q_D Q_C Q_B$=100, c'est à dire 4, le bit Q_D doit revenir à 0 tandis que le bit Q_C reste à 0 comme cela est indiqué sur la table de transitions ci dessous.

Etat n			Etat n+1		
Q_D	Q_C	Q_B	Q_D	Q_C	Q_B
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0

Les bits Q_B (LSB) et Q_D (MSB) ne suivent donc pas les règles relatives aux compteurs binaires énoncées dans le paragraphe 5.2.1. Il en résulte que l'horloge du dernier étage ne peut pas être directement commandée par la sortie Q_C. Les transitions $0 \to 1$ et $1 \to 0$ de la bascule représentant le MSB (Q_D) se faisant dans des conditions particulières, on n'utilisera pas une bascule *JK*, montée en bascule *T* comme pour les étages précédents, mais plutôt une bascule synchrone *RS* ou *JK*.

b. Réalisation.

Dans le compteur 7490 le bit de poids fort est réalisé avec une bascule *RS*. Nous allons déterminer quel doit être le « câblage » des entrées *R* et *S* pour assurer un

[10] Le cycle de comptage compte 5 états au lieu de 8 habituellement pour un compteur binaire sur 3 bits.

comportement du bit Q_D conforme à l'évolution décrite dans la table précédente. Pour cela nous analyserons l'évolution de chacun des états du cycle de comptage et nous utiliserons la table de commande de la bascule RS donnée dans le paragraphe YYY.

- La transition $0 \to 1$ du bit Q_D se produit lorsque $Q_D Q_C Q_B = 011$, donc tout à fait normalement, et nécessite d'après la table de commande de la bascule *RS* : $S = 1$ et $R = 0$.
- La transition $1 \to 0$ a lieu lorsque $Q_D Q_C Q_B = 100$. Elle impose donc d'avoir *S* = 0 et *R* = 1 au moment de l'impulsion.
- Pour les trois autres états $Q_D Q_C Q_B = 000, 001, 010$ le bit de poids fort doit rester à 0 ($Q_D : 0 \to 0$). On doit donc avoir *S* = 0, *R* = φ.

On déduit facilement des remarques précédentes les tables de Karnaugh de *S* et *R*, ainsi que les expressions logiques correspondantes.

Q_B	0	0	1	1
Q_D Q_C	0	1	1	0
0	0	0	1	0
1	0	φ	φ	φ

S

Q_B	0	0	1	1
Q_D Q_C	0	1	1	0
0	φ	φ	0	φ
1	1	φ	φ	φ

R

On obtient $S = Q_B \bullet Q_C$ et $R = Q_D$.

Le bit de poids faible Q_B, qui est matérialisé par une bascule *JK*, a également un comportement atypique par rapport au fonctionnement des compteurs binaires. Il commute à chaque impulsion d'horloge sauf au moment de la transition de 4 à 0 au cours laquelle il doit rester à 0. Pour les états $Q_D Q_C Q_B = 000, 001, 010, 011$ on peut donc avoir $J_B = K_B = 1$. En revanche pour l'état $Q_D Q_C Q_B = 100$ il est nécessaire que $J_B = 0$ et $K_B = 1$. La solution est donc de câbler la bascule *JK* représentant le bit de poids faible avec $J_B = \overline{Q_D}$ et $K_B = 1$

La bascule représentant le bit Q_B fonctionne donc toujours en bascule *T* sauf lorsque

$Q_D = 1$ c'est à dire uniquement lors du passage de 4 à 0.

c. Schéma

Le schéma de ce compteur est représenté sur la Figure 41. Il correspond au compteur *TTL* de référence 7490. Notons que ne figurent pas sur ce schéma les entrées de remise à 0 et de remise à 9 présentent dans le compteur réel. Toutes les entrées *J* et *K* non connectées sur le schéma sont au niveau 1.

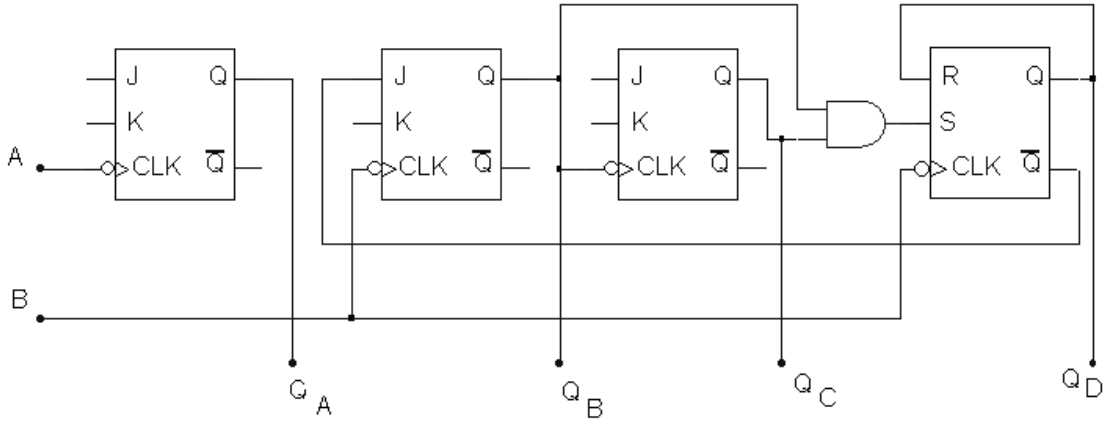


Figure 41 Compteur asynchrone par 10 (type 7490). Toutes les entrées *J* et *K* non connectées sur le dessin sont en réalité au niveau haut.

d. Evolution des états non compris dans le cycle de comptage.

Le fonctionnement d'un compteur peut être représenté sous la forme d'un cycle sur lequel on indique les différents états (ici $Q_D Q_C Q_B$) et les transitions entre ces états. La Figure 42.a) représente le cycle de comptage du compteur par 5 que nous venons d'étudier. On remarque aisément que trois des huit combinaisons possibles des bits $Q_D Q_C Q_B$ ne sont pas utilisées dans le cycle de comptage ; ce sont les combinaisons $Q_D Q_C Q_B$=101, 110 et 111. La question se pose alors de savoir comment évolue le compteur lorsqu'il se trouve dans l'un de ces états[11]? Peut-il y rester bloqué ou va-t-il rejoindre le cycle "normal" comptage après une ou plusieurs impulsions?

[11] Cette situation peut se produire à la mise sous tension lorsque le compteur se retouve aléatoirement dans l'un des huit états possibles.

Pour répondre il faut, pour chaque état, établir les valeurs des entrées des bascules. On en déduit alors facilement l'évolution de l'état.

- Pour l'état $Q_DQ_CQ_B$=101 les entrées des bascules sont $J_B = 0$, $K_B = 1$, $J_C = 1$, $K_C = 1$ et $S = 0, R = 1$. Les valeurs futures de $Q_DQ_CQ_B$ seront donc 010. L'état $Q_DQ_CQ_B = 101$ évolue ainsi vers l'état $Q_DQ_CQ_B = 010$ qui appartient au cycle de comptage.
- Pour l'état $Q_DQ_CQ_B = 110$ les entrées des bascules sont J_B=0, K_B=1, J_C=1, K_C=1, et S = 0, R = 1. Les valeurs futures de $Q_DQ_CQ_B$ seront alors 010. En effet, la sortie Q_B restant inchangée, **il n'y a pas d'impulsion sur l'entrée horloge de la bascule représentant le bit** Q_C. En conséquent bien que J_C=1, K_C=1 le bit Q_C ne change pas !
- Pour l'état $Q_DQ_CQ_B = 111$ les entrées des bascules sont J_B=0, K_B=1, J_C=1, K_C=1, et S = 1, R = 1. Les valeurs futures de $Q_DQ_CQ_B$ seront donc 000.

Les trois états en dehors du cycle évoluent donc, en une seule impulsion, vers un état du cycle de comptage. La Figure 42.b) schématise le fonctionnement complet du compteur avec l'évolution des états hors cycle.

5.3. *Les compteurs synchrones.*

5.3.1. Les compteurs binaires à retenue série.

Rappelons que dans un compteur synchrone chaque bit de sortie Q_i commute ($0 \to 1$ ou $1 \to 0$) si, et seulement si, tous les bits de poids inférieurs sont au niveau haut. Par exemple la bascule représentant le bit de poids fort Q_D commute si $Q_A \bullet Q_B \bullet Q_C = 1$. Cette quantité s'appelle la retenue. Si la retenue est différente de 1 la bascule doit être en mode mémoire pour ne pas commuter. On réalise généralement ces compteurs en utilisant des bascules *JK* soit en mode mémoire (J = K = 0), soit en mode toggle (J = K = 1).

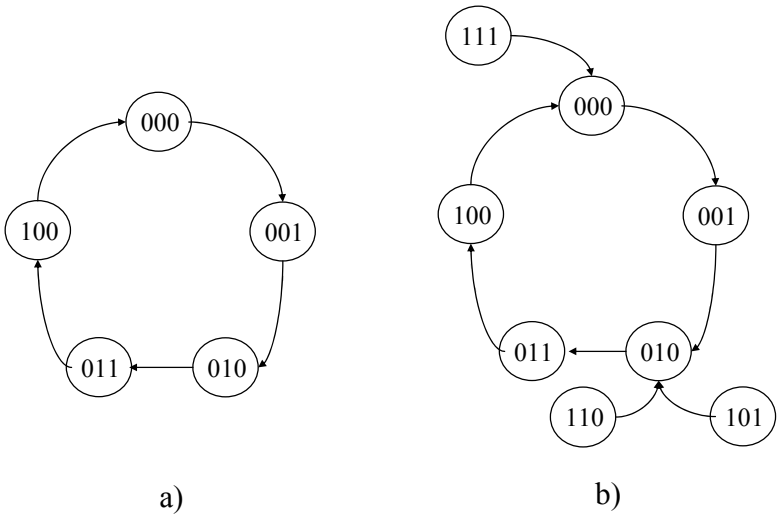


Figure 42Cycle de comptage du compteur par 5. Les transitions entre états se produisent sur les fronts descendants du signal présent sur l'entrée B.

La manière la plus simple de calculer la retenue est d'effectuer le double produit $(Q_A \bullet Q_B) \bullet Q_C$, et ainsi de suite de proche en proche, en n'utilisant que des portes AND à deux entrées. On aboutit ainsi au schéma de la Figure 43 qui est finalement assez peu utilisé en pratique. En effet la retenue se propage d'un bout à l'autre du circuit avec un retard croissant dû à l'accumulation des portes AND. Comme dans le cas des compteurs asynchrones, ces retards limitent la fréquence maximum de comptage. A l'inverse il faut signaler que les états transitoires n'existent quasiment plus!

5.3.2. Les compteurs binaires à retenue parallèle (ou anticipée).

Pour chaque bit, par exemple le bit *i*+1, on calcule la retenue $Q_0 \bullet Q_1 ... \bullet Q_i$ avec des portes AND à entrées multiples. Ces portes ne coûtent pas beaucoup plus cher que les portes à deux entrées et elles ont le même temps de propagation. On atteint de cette façon des fréquences de comptage plus élevées. On abouti de cette façon au schéma de la Figure 44 sur lequel on distingue bien les AND multiples qui calculent directement la retenue pour chaque étage. On remarque également l'entrée *R* pour la retenue initiale ainsi que la sortie R' pour une éventuelle retenue propagée (ripple carry) vers le compteur suivant dans le cas d'une mise en série.

Ces compteurs sont parfois appelés compteurs à propagation de retenue.

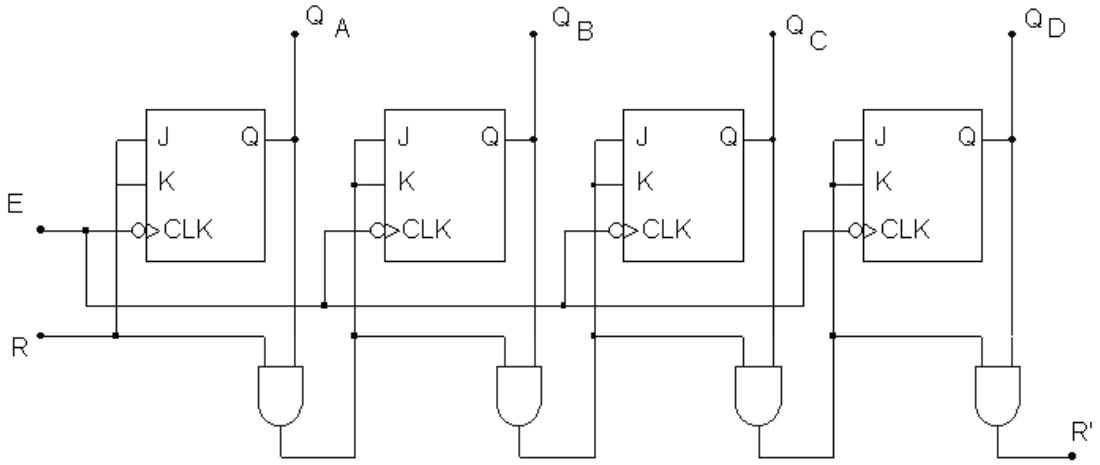


Figure 43 Compteur synchrone série. R correspond à la retenue initiale (entrée de validation) et R' à la retenue finale (à transmettre, éventuellement au compteur suivant)

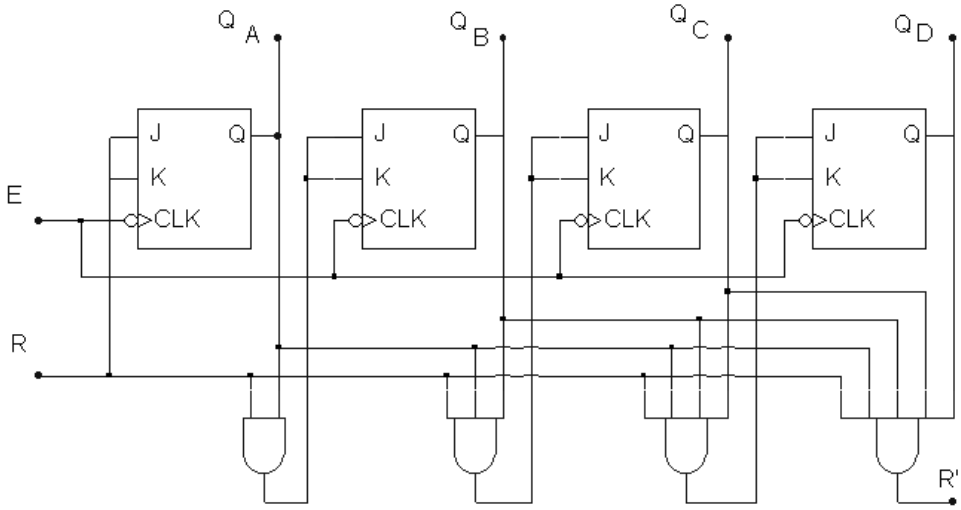


Figure 44 Schéma de principe d'un compteur synchrone à retenue parallèle

5.3.3. Les compteurs synchrones par 10.

Il est toujours possible, en utilisant l'entrée CLEAR présente sur chaque compteur, de remette à zéro d'un compteur en cours de comptage. Selon les compteurs cette remise à zéro peut se faire de deux façons. Soit en agissant sur l'entrée CLR de chaque bascule, soit en imposant $J = 0$, $K = 1$ sur chacune des bascules. Dans le premier cas la remise à

zéro est immédiate et indépendante des impulsions arrivant sur le compteur: on parle alors de remise à zéro asynchrone. Dans le second cas la remise à zéro ne sera effective que au moment de la prochaine impulsion arrivant sur le compteur: La remise à zéro est alors synchrone.

Les compteurs disposant d'une remise à zéro synchrone permettent de réaliser facilement des cycles de comptage par $N \neq 2^n$. Par exemple, pour réaliser un compteur par 10 il suffit de décoder en sortie la valeur 9 ($Q_D Q_C Q_B Q_A = 1001$) et de rétro-agir sur l'entrée CLEAR synchrone du compteur. L'impulsion suivante entraînera alors la remise à zéro du compteur. On réalise de la même façon des compteurs par 7, 5, 11…. La Figure 45 propose un exemple de compteur par 10 réalisé à partir d'un compteur synchrone par 16 de référence 74163.

Comme dans le cas des compteurs asynchrones, la réalisation de cycles de comptage dans des codes autres que le code binaire standard (CBN) nécessite une synthèse complète du compteur, ce qui sort du cadre du cours de 1ere année.

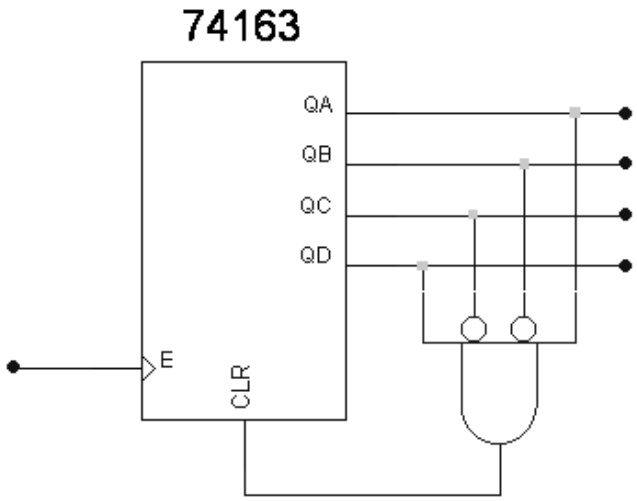


Figure 45 Réalisation d'un compteur synchrone par 10 en utilisant une contre réaction sur l'entrée CLEAR synchrone.

5.4. *Les registres.*

5.4.1. Définitions.

Les registres sont les éléments de base des mémoires réalisées avec des semi-conducteurs. On peut se représenter un registre comme un ensemble de mémoires élémentaires susceptibles de stocker chacune un bit. L'entrée des informations dans un registre peut se faire soit en série (les unes après les autres) soit en parallèle (toutes au même moment). De la même façon la présentation des informations sur les sorties peut se faire soit en série soit parallèle. On aboutit ainsi à 4 types de fonctionnements différents pour les registres (parallèle-série, parallèle-parallèle, série-parallèle et série-série).

5.4.2. Les registres tampon.

Les registres tampon sont des registres de type parallèle-parallèle constitués de *n* bascules de type *D* commandées par une même horloge (Figure 46).

Au signal d'horloge (impulsion sur CLK) les entrées D_i sont recopiées sur les sorties Q_i. Une entrée asynchrone *CLR* permet, de façon prioritaire, d'effacer le contenu du registre et d'écrire $Q_i = 0.$ Entre deux impulsions les sorties sont parfaitement isolées des entrées

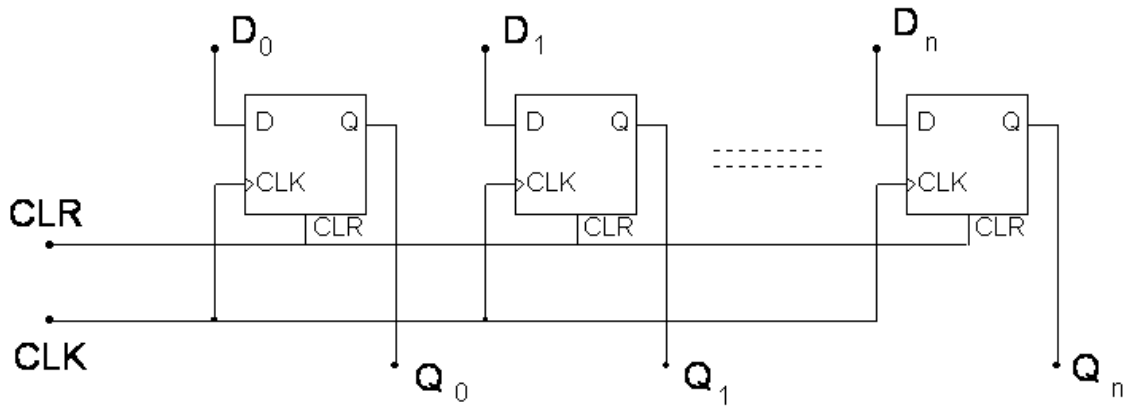


Figure 46 Registre tampon à base de bascules D

5.4.3. Les registres à décalage.

Les registres à décalage sont des registres de type série-série ou série parallèle, dans lesquels les informations sont décalées d'une bascule vers la suivante au rythme des impulsions d'une horloge. Ils sont généralement réalisés avec des bascules *RS* de type maître-esclave.

Le schéma de principe d'un registre à décalage (vers la droite) avec entrée série est présenté sur la Figure 47. Au fur et à mesure des impulsions d'horloge les données présentes sur l'entrée série *E* sont transférées sur les différentes bascules. La présence de l'inverseur entre *R* et *S* assure toujours $\overline{S} = R.$ Les bascules ne sont donc jamais en mode mémoire mais toujours en mode SET ou RESET. Dans ces conditions la sortie Q_i recopie, au moment de l'impulsion d'horloge, la valeur présente sur l'entrée S_i.

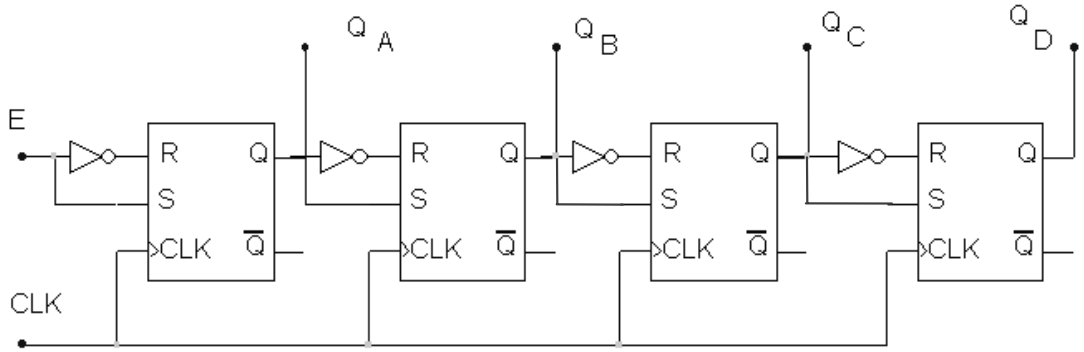


Figure 47 Schéma de principe d'un registre à décalage avec une entrée série (*E*). Le décalage se fait de Q_A vers Q_B.

Donnons un exemple de fonctionnement pour fixer les idées. On suppose que la situation de départ est la suivante: *E* = 0 et seule la sortie Q_A de la première bascule est au niveau 1 les autres étant au niveau 0 ($Q_A Q_B Q_C Q_D = 1000$).

– 1ére impulsion : Chaque bascule recopie sur sa sortie Q la valeur présente sur son entrée *S*. On obtient donc après l'impulsion $Q_A Q_B Q_C Q_D = 0100.$

– 2eme impulsion : Seule l'entrée S de la troisième bascule est au niveau haut. Après l'impulsion on a donc $Q_A Q_B Q_C Q_D = 0010.$

On raisonne de la même façon pour la troisième impulsion et l'on aboutit à

$Q_A Q_B Q_C Q_D = 0001.$ Ainsi au fur et à mesure des impulsions sur l'entrée d'horloge (CLK) le 1 présent initialement sur Q_A a été progressivement décalé vers la droite.

5.4.4. Les registres universels.

Dans la pratique il est inutile d'effectuer la synthèse des registres à décalage : un choix très vaste est offert par les constructeurs. A titre d'exemple citons les registres universels de type 194 dont le schéma est présenté sur la page suivante. Ce sont des registres 4 bits à chargement parallèle ou série. Outre une remise à zéro asynchrone (CLEAR) ils offrent la possibilité de déplacer l'information vers la droite (de Q_A vers Q_D) ou vers la gauche (de Q_D vers Q_A). Le mode de fonctionnement est choisi avec les entrées synchrones S_0 et S_1.

5.4.5. Applications des registres à décalage

Les registres à décalage sont utilisés dans de nombreuses applications comme par exemple :

- La conversion parallèle série (registre à entrée parallèle et sortie série).
- La mise en mémoire, avant affichage, des sorties d'un compteur (registre à entrée et sortie parallèles).
- La génération de séquences 011001… pseudo aléatoire (registre à entrée série et sortie parallèle).
- La réalisation de compteurs dans des codes autres que le *CBN* (registre à entrée série et sortie parallèle).
- La multiplication d'un nombre codé $a_n \ldots a_0$ par 2^p se traduit par le décalage de tous les bits de *p* cases vers la gauche.

74AC11194
4–BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER

SCAS093 – NOVEMBER 1989 – REVISED APRIL 1993

Function Table

INPUTS										OUTPUTS			
CLEAR	MODE		CLOCK	SERIAL		PARALLEL				Q_A	Q_B	Q_C	Q_D
	S1	S0		LEFT	RIGHT	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{AO}	Q_{BO}	Q_{CO}	Q_{DO}

H = high level (steady state)
L = low level (steady state)
X = irrelevant (any input, including transitions)
↑ = transition from low to high level
a,b,c,d = the level of steady-state input at inputs A, B, C, or D, respectively.
Q_{AO}, Q_{BO}, Q_{CO}, Q_{DO} = the level of Q_A, Q_B, Q_C, or Q_D, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn} = the level of Q_A, Q_B, Q_C, or Q_D respectively, before the most-recent ↑ transition of the clock.

logic diagram (positive logic)

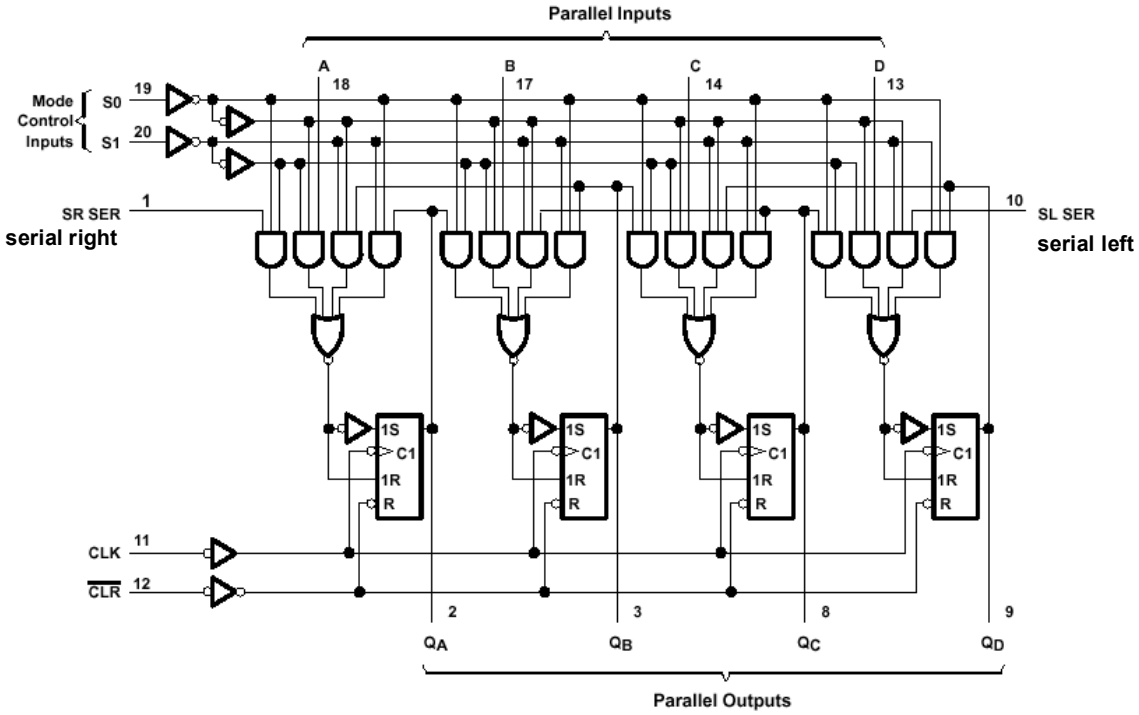


Figure 48 Schéma d'un registre à décalage universel type 194

5.5. *Les mémoires à semi-conducteur.*

Contrairement aux systèmes analogiques, les systèmes numériques permettent de mémoriser des informations avant de les traiter. Actuellement, on distingue trois grands types de mémoires (Figure 49) :

– Les mémoires magnétiques comme par exemple les disques durs ou les disquettes.

– Les mémoires à semi-conducteur telles que les RAM ou les ROM.

– Les mémoires optiques comme les CD et les DVD.

Dans ce cours nous ne décrirons que les mémoires à semi-conducteurs et les mémoires optiques.

5.5.1. Les mémoires vives.

Les mémoires vives ou RAM (Random Access Memory) peuvent être lues ou écrites quasi-instantanément en fonction des besoins de l'utilisateur (les temps d'accès sont de l'ordre de 10 à 50 ns). Historiquement, le nom de RAM est apparu pour signifier que, contrairement aux bandes magnétiques, qui étaient alors les seuls supports d'information existants, il n'est pas nécessaire de faire défiler toutes les données situées avant l'information que l'on cherche. Ce type de mémoire est largement utilisé dans tous les systèmes ayant besoin de stocker temporairement de l'information, comme par exemple les micro-ordinateurs. Leur inconvénient majeur est la perte de toute l'information stockée en cas de coupure de l'alimentation électrique. En pratique, il existe deux sortes de mémoires vives.

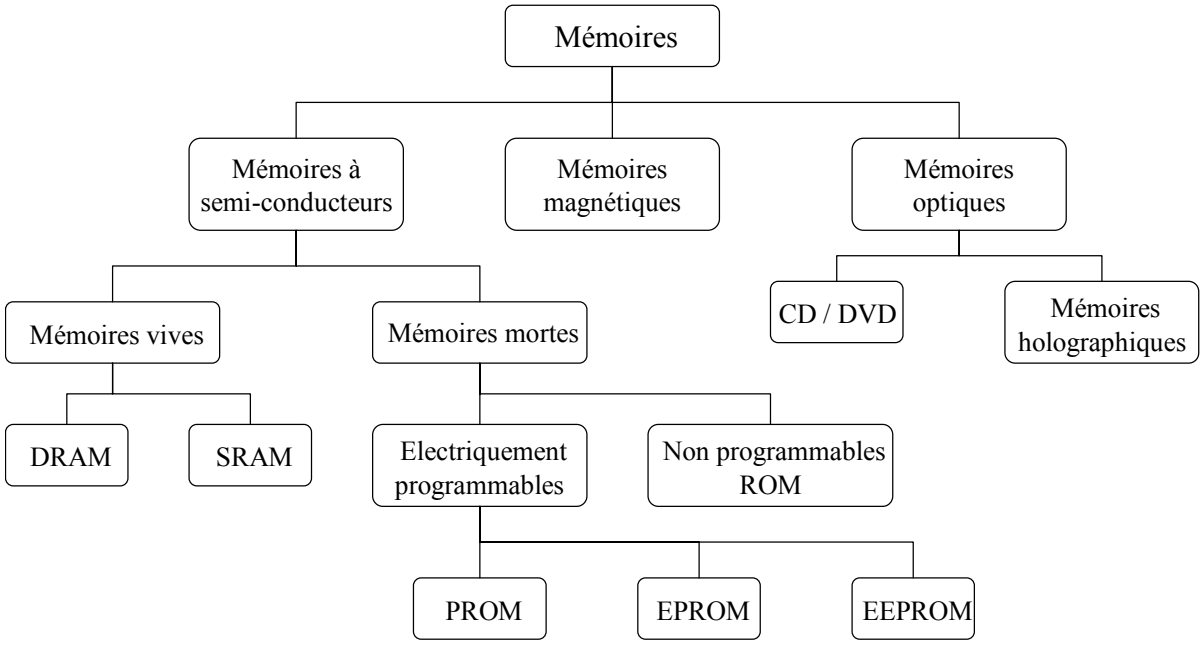


Figure 49 Les différents types de mémoires

a. Les RAM statiques (SRAM).

L'unité de base d'une SRAM est typiquement une bascule D (Figure 50). Dans la phase d'écriture la donnée présente sur D est recopiée sur la sortie Q au moment d'un front montant sur l'entrée d'horloge. La lecture consiste à forcer la ligne de sélection au niveau haut pour présenter la sortie de la bascule D sur le bus de données.

b. Les RAM dynamiques (DRAM).

Le principe de fonctionnement d'une cellule mémoire est assez simple puisqu'il s'agit, pour écrire 1 ou 0, de charger ou de décharger un condensateur[12]. Cependant, en raison des courants de fuite des condensateurs, l'information stockée tend à se dégrader au cours du temps. Ces mémoires doivent donc être périodiquement rafraîchies (typiquement toutes les 20 ms). Malgré cet inconvénient les DRAM sont très fréquemment utilisées car leur simplicité permet de les intégrer en plus grand nombre sur une puce de silicium que leurs concurrentes statiques (SRAM).

[12] En pratique les condensateurs sont les capacités parasites des grilles de transistors MOS.

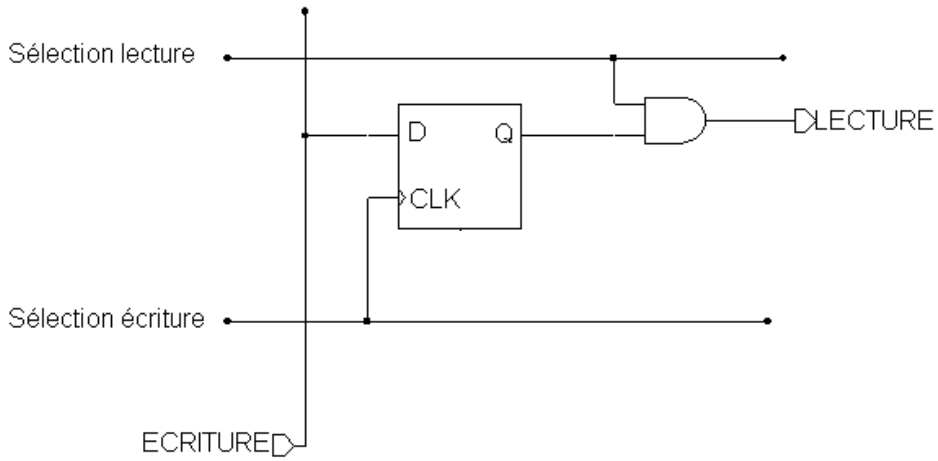


Figure 50 Cellule élémentaire d'une mémoire RAM statique (SRAM).

5.5.2. Les mémoires mortes.

Les mémoires mortes ou ROM (Read Only Memory) ne sont accessibles que en lecture. Elles doivent être programmées en dehors du système où elles sont exploitées. Selon le mode d'écriture et son caractère définitif ou pas, on distingue différents types de mémoires mortes.

a. Les ROM.

Les ROM sont écrites une fois pour toute en usine chez le fabriquant de la mémoire. Cette programmation est faite directement sur le "wafer" (galette de silicium) à l'aide de masques de programmation. Bien évidement la fabrication de ROM ne se conçoit que pour des séries importantes (> 10 000 unités).

b. Les PROM.

Si l'écriture est réalisée par un programmateur après la fabrication de la mémoire on parle de PROM (Programmable ROM). En pratique les PROM sont constituées par un réseau de fusibles et la programmation est réalisée en détruisant certains fusibles du réseau (pour cela il suffit, pendant une centaine de *ms*, d'appliquer une tension de 10-15 V à l'adresse désirée). Considérons, par exemple, la PROM de 4×5=20 bits (4 mots de 5 bits) représentée sur la Figure 51. Un décodeur 2 vers 4 avec sorties actives à l'état bas permet de sélectionner une ligne parmi les 4. Ainsi à la sortie d'adresse $a_1 a_0 = 01$, la ligne notée 01 est forcée à 0 et les autres lignes restent à 1 ; les 5 bits de sortie valent alors 0 (Figure 51.a). Avant programmation toutes les sorties de la PROM sont au niveau bas.

Après programmation, c'est à dire après la destruction de certains fusibles ont peut obtenir la situation représentée sur la Figure 51.b . Une ligne de sortie vaut 1 en l'absence de diode (liaison détruite) entre elle et la ligne d'adresse sélectionnée. A l'inverse si une diode est présente, elle ramène le potentiel de la ligne à 0. Le contenu de cette mémoire 20 bits est le suivant :

Adresse a_1 a_0	*Sorties* $S_4 S_3 S_2 S_1 S_0$
0 0	0 0 1 0 1
0 1	1 0 1 0 1
1 0	0 1 1 0 1
1 1	1 1 1 1 0

Les PROM sont plutôt utilisées pour des systèmes fabriqués en séries restreintes ou souvent renouvelées

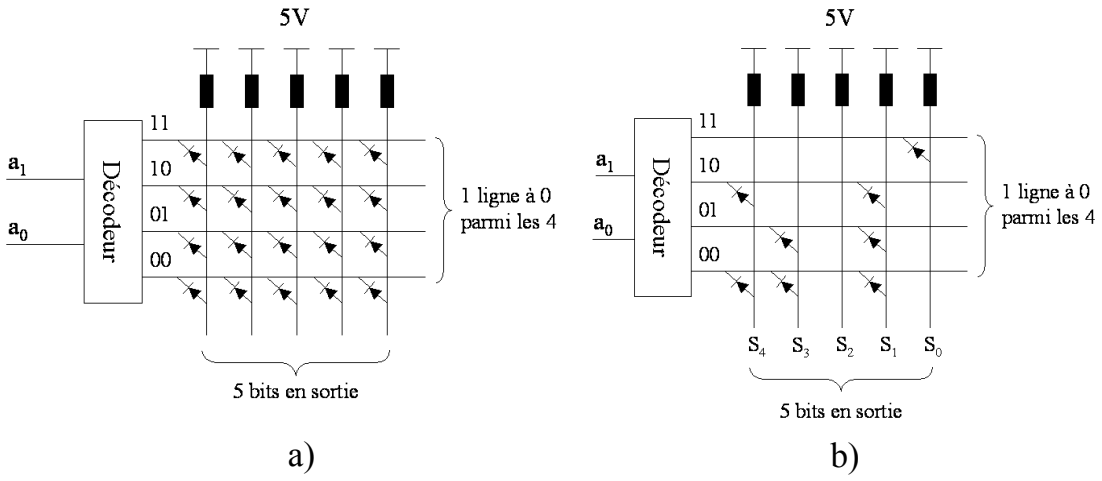


Figure 51 Principe de la programmation d'une mémoire PROM. a) avant la programmation. b) après la programmation.

c. Les PROM effaçables.

Il existe des PROM que l'on peut effacer avec des rayons UV, ce sont les EPROM (Erasable PROM). On enfin trouve des mémoires mortes effaçables électriquement ce sont les EEPROM (Electrical Erasable PROM). Il est bien entendu possible de réécrire dans ces mémoires effaçables. Les EPROM et les EEPROM sont surtout utilisées dans

les phases de développement ou dans les systèmes fabriqués en très petites séries.

5.5.3. Organisation d'une mémoire.

D'un point de vue logique, une mémoire est constituée d'un ensemble de registres dont l'un est en relation avec l'extérieur soit pour sa lecture (ROM) soit pour sa lecture ou son écriture (RAM). La Figure 52 représente l'organisation d'une RAM composée 64 registres de 8 bits. La taille des registres définit le format de la mémoire (ici 8 bits soit 1 octet) et le nombre de registres détermine la capacité de la mémoire (ici 64 octets).

a. Les données.

Pour limiter au maximum l'encombrement du composant, on utilise généralement les mêmes pattes pour les entrées (Input) et les sorties (Output) de la RAM. On gagne ainsi un facteur 2 sur le nombre de broches dédiées aux données, mais il faut bien sûr ajouter une entrée de contrôle supplémentaire pour spécifier si la RAM fonctionne en mode lecture ou mode écriture (entrée $\overline{OE}$ pour Output Enable[13]). Pour la RAM considérée ici le gain est de 16/2 – 1 = 7 broches.

b. Les adresses.

Le choix du registre en contact avec l'extérieur est réalisé en décodant l'adresse écrite sur les 6 bits $A_5A_4A_3A_2A_1A_0$ ($A_5 = MSB$ et $A_0 = LSB$). L'adresse $A_5A_4A_3A_2A_1A_0 = 001101$ désigne ainsi le registre n°13. Là encore, des "astuces de câblage" permettent de réduire le nombre de connexions pour les mémoires de grande capacité. Par exemple, une mémoire de 1 Méga-Octet nécessite 20 bits d'adresse donc 20 broches. On peut réduire ce nombre par 2 en transmettant l'adresse en deux paquets consécutifs de 10 bits.

c. Gestion des cycles d'écriture et de lecture.

En plus de l'entrée $\overline{OE}$ décrite précédemment, les RAM possèdent souvent au moins

[13] En électronique les broches des composants ont souvent des noms mnémotechniques en rapport avec leur fonction. Par convention, si le nom est surmonté d'une barre la fonction est active au niveau bas. Ainsi $\overline{OE} = 0$ place la RAM en mode lecture et $\overline{OE} = 1$ en mode écriture.

deux autres entrées de contrôle. D'une part l'entrée $\overline{CS}$ (Chip Select) qui permet de mettre la RAM en mode actif ou en mode veille (en mode veille la consommation électrique de la RAM est nettement réduite mais aucune action de lecture ou d'écriture n'est possible). D'autre part l'entrée $\overline{WE}$ (Write Enable) qui n'autorise l'écriture que si elle est au niveau bas.

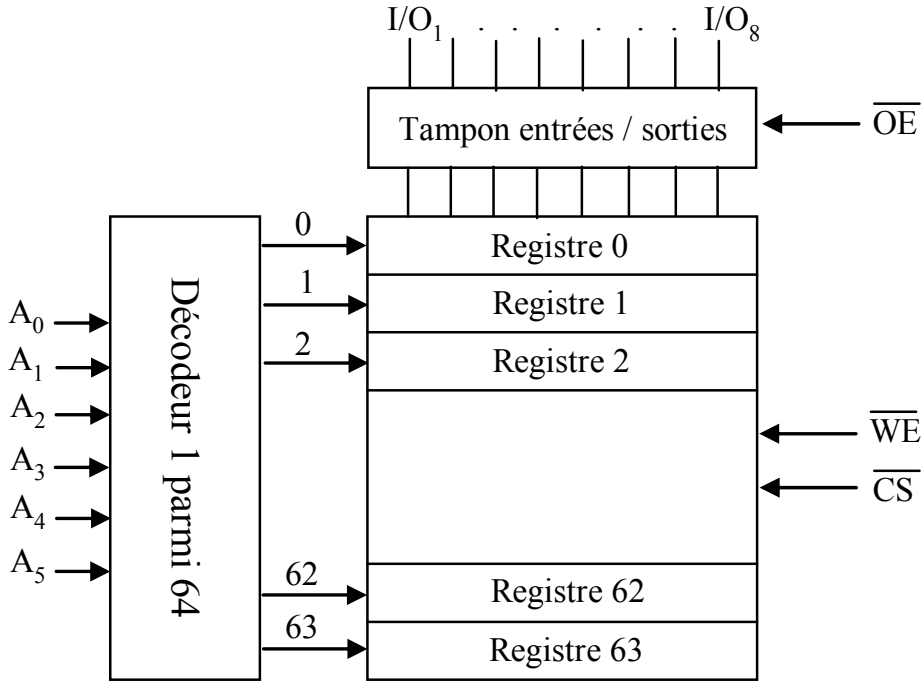


Figure 52 Schéma de l'organisation d'une mémoire RAM 64x8 bits.

Tout cela fait beaucoup d'entrées pour un seul composant et il convient, lorsque l'on veut utiliser une RAM, de respecter une certaine chronologie. En général, dans un cycle d'écriture, il faut tout d'abord sélectionner le composant ($\overline{CS} = 0$), le mettre en mode lecture ($\overline{OE} = 0$), puis fixer les adresses et les données, et ensuite seulement donner l'ordre d'écriture ($\overline{WE} = 0$). Cette chronologie est illustrée sur la Figure 53.

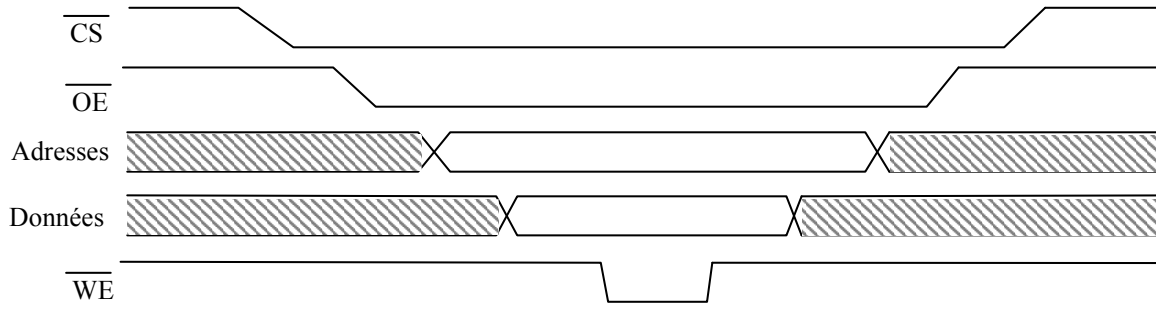


Figure 53 Cycle d'écriture dans une RAM.

En plus des mémoires à base de semiconducteurs, il existe aujourd'hui d'autres types de mémoires basées sur le stockage optique de l'information. Même si ces mémoires sortent, à priori, du cadre de ce chapitre, il n'est pas inutile d'avoir quelques idées sur leur fonctionnement, d'autant plus que elles équipent la totalité des ordinateurs actuels.

5.6. *Les mémoires optiques CD et DVD.*

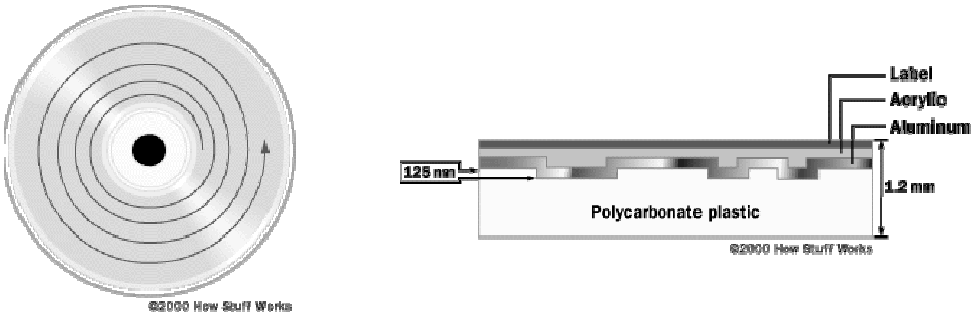
Les CD (Compact Disk) et DVD (Digital Versatil Disc) sont extrêmement répandus aujourd'hui. Ils sont utilisés pour stocker de la musique, des données informatiques ou de la vidéo. Du fait de leur "facilité" de fabrication et d'utilisation, ils sont devenus un système standard de diffusion de l'information. Les principes de stockage de l'information sur ces deux types de supports sont semblables. Nous commencerons par étudier les CD.

5.6.1. Les CD préenregistrés.

a. Stockage de l'information.

Un CD est un simple morceau de plastique (polycarbonate) de 1,2 mm d'épaisseur, sur lequel l'information est écrite le long d'une spirale qui se déroule de l'intérieur vers l'extérieur du disque (Figure 54.a). Au cours de la fabrication, l'information est stockée sous forme de microcuvettes gravées sur une des faces du disque ; cette face est ensuite recouverte d'une couche métallique (aluminium, argent ou or) puis d'un revêtement acrylique pour protéger le dépôt métallique(Figure 54.b). Ce qui est réellement impressionnant sur un CD ce sont les dimensions de ces microcuvettes (Figure 55). Leur profondeur est de 125 nm, leur largeur de 500 nm et leur longueur de 830 nm au minimum

($1nm = 10^{-9}m$ = 1milliardième de mètre). Les sillons de la spirale étant séparés de 1,6 μm, on peut calculer que l'information est stockée sur 5 kilomètres de long!



a) b)

Figure 54 a) Sur un disque CD les données sont écrites le long d'une spirale qui part du centre vers l'extérieur du disque. b) Coupe transversale d'un disque CD.

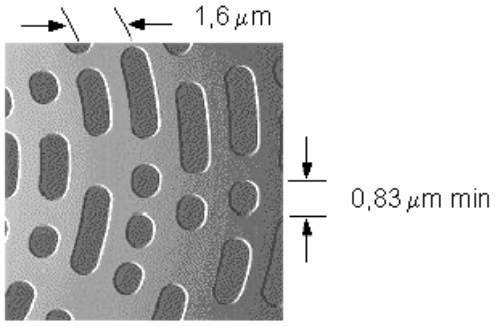


Figure 55 Agrandissement de la surface d'un disque CD préenregistré.

b. Lecture.

Le rôle principal d'un lecteur de CD est de focaliser un laser le long de la spirale sur laquelle sont gravées les microcuvettes. Le faisceau laser passe à travers le polycarbonnate et se réfléchi sur la couche métallique. Selon que le laser est réfléchi dans une cuvette ou sur une partie plate, une photodiode détecte, ou pas, le rayonnement. Cette modulation de la réflexion permet de reconstituer le signal inscrit sur le disque (Figure 56). En fait c'est le délai entre deux transitions consécutives qui constitue l'information. Ce type de fonctionnement impose des contraintes très fortes sur la fiabilité et la précision des parties mécaniques d'un lecteur de CD.

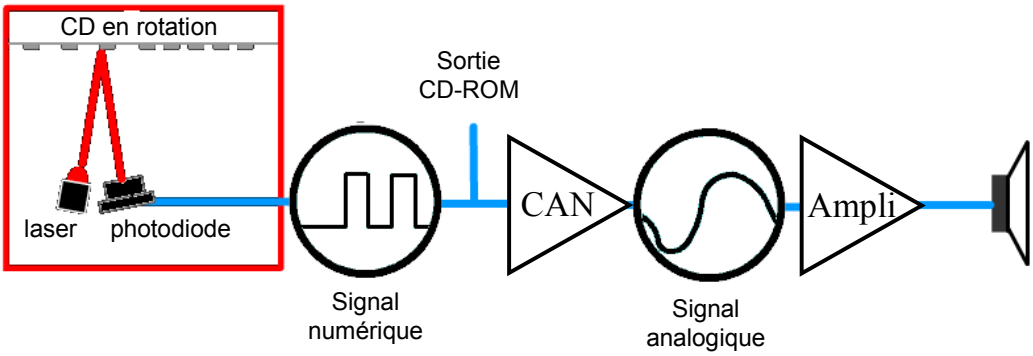


Figure 56 Principe de la lecture des données sur un disque CD. Lorsque le laser arrive sur un creux, le rayon n'est pas réfléchi vers la photodiode et n'est donc pas détecté (0 logique).

Un lecteur de CD est composé de 3 parties importantes (Figure 57) :

- Un moteur qui fait tourner le disque et dont la vitesse de rotation est très précisément contrôlée entre 200 et 500 tours/min en fonction de la partie du disque qui est lue.
- Un système mobile comprenant, entre autre, une diode laser, une lentille et une photodiode.
- Un second moteur qui déplace le système laser le long du disque afin que le laser suive précisément la spirale. Ce système de déplacement doit avoir une précision de l'ordre du micron.

5.6.2. Les CD enregistrables CD-R.

Les disques CD-R sont constitués d'un support en plastique revêtu d'une couche de colorant et d'une couche métallique. Il n'y a bien sûr plus de microcuvettes à la surface du disque puisque celui ci ne contient pas de données. En revanche, la spirale sur la quelle seront enregistrées les informations est matérialisée par un sillon creusé dans la surface du plastique.

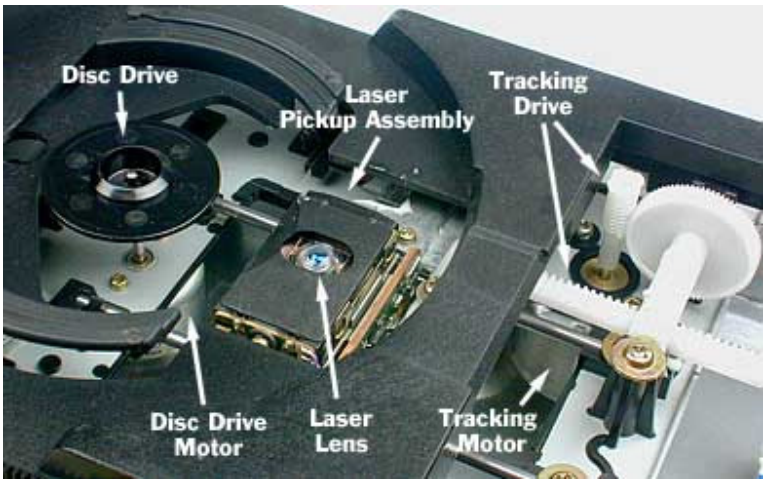


Figure 57 Les différentes parties d'un lecteur de CD.

L'enregistrement des données se fait à l'aide d'un laser focalisé modulé dans le temps entre deux niveaux de puissance: la puissance de lecture (inférieure au mW) et la puissance d'écriture (comprise entre 6 et 12 mW selon les disques). Le colorant absorbe la lumière et induit un échauffement local proportionnel à la puissance du laser. A la puissance d'écriture le colorant est dégradé et le substrat en plastique subit une déformation locale. Ces deux effets se traduisent par une baisse de la réflexion. La modulation de puissance est évidemment imposée par le signal que l'on veut écrire.

Les CD-R ne sont enregistrables qu'une seule fois car les effets du laser sur le colorant et le support plastique sont irréversibles. Une autre catégorie de CD a donc été développée afin de permettre l'effacement des données inscrites.

5.6.3. Les CD réengistrables CD-RW.

Les CD-RW utilisent des matériaux à changement de phase. Ce sont des matériaux dans lesquels peuvent coexister, à température ambiante, deux phases différentes ; en l'occurrence une phase cristalline *et* la phase amorphe. Ces deux états du matériau ont des propriétés optiques très différentes. Cette différence est mise à profit pour enregistrer l'information sous forme d'une variation locale de la réflexion. L'état initial, associé à la réflexion haute, est l'état cristallin. Une donnée est inscrite sous la forme d'une marque amorphe ayant une réflexion basse. Pour effacer les données, il suffit de recristalliser les marques amorphes (Figure 58).

a. Ecriture des données.

Pour écrire une marque amorphe, il faut fondre localement le matériau et le refroidir

suffisamment vite pour qu'il se solidifie en restant dans un état proche de l'état liquide: l'état amorphe. Typiquement, la fusion est obtenue autour de $600^{O}C$ avec une impulsion laser de 10 mW et d'une durée de 10 *ns*. Si le refroidissement n'est pas suffisamment rapide, les atomes peuvent se déplacer et former un édifice cristallin. Cette recristallisation se traduit par une baisse voir une disparition du contraste.

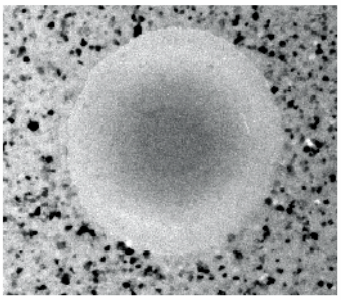


Figure 58 Image en microscopie électronique d'une marque amorphe dans une matrice cristalline.

b. Effacement des données.

Pour effacer les données on chauffe la marque amorphe a une température inférieure à la température de fusion. Ceci est toujours obtenu avec le faisceau laser focalisé, mais à une puissance intermédiaire entre la puissance d'écriture et la puissance de lecture.

Dans le domaine des disques optiques deux grandes catégories de matériaux à changement de phase sont utilisées :

- Les alliages ternaires de germanium, d'antimoine et de tellure GeSbTe
- Les alliages d'antimoine et de tellure dopés à l'argent et à l'indium AgInSbTe

5.6.4. Les DVD.

Bien que semblable d'aspect à un disque CD, un disque DVD standard permet de stocker 7 fois plus d'informations (4,7 Go au lieu de 700 Mo). Le principe des DVD, quel qu'en soit le type (préenregistré, enregistrable ou réenregistrable) est similaire à celui des CD ; la seule différence est la taille des structures inscrites sur le disque. Par exemple, sur un DVD préenregisté les microcuvettes sont à peu près deux fois plus petites que sur les CD (400 nm au lieu de 830 nm) et la spirale est beaucoup plus serrée puisque les sillons ne sont séparés que de 0,74 μm au lieu de 1,6 μm (Figure 59) L'information est ainsi stockée sur une spirale de 11 kilomètres de long!

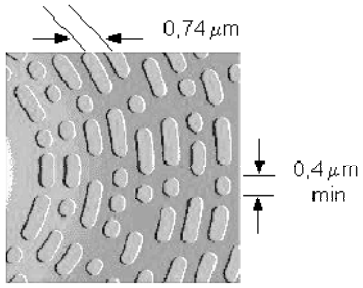


Figure 59 Agrandissement de la surface d'un DVD.

Les chiffres précédents posent un problème. Sur un DVD les microcuvettes étant deux fois plus petites et la spirale deux fois plus longue que sur un CD, on devrait, à priori, stocker 4 fois plus d'informations. D'ou vient le facteur 7 observé en pratique sur la capacité de stockage? La réponse est, en partie, dans le codage des informations. Pour garantir une bonne restitution, les informations sont écrites dans des codes capables de corriger les erreurs, cela entraîne une augmentation importante du nombre de bits nécessaires pour coder une information. La méthode de correction utilisée sur les CD est plutôt "ancienne" et beaucoup moins efficace que celle des DVD pour lesquels on obtient les mêmes résultats en utilisant moins de bits que dans les CD.

Pour conclure signalons enfin que les DVD préenregistrés peuvent contenir des informations sur les deux faces et que chaque face peut avoir deux niveaux de données (Figure 60). Pour les DVD multicouches la piste extérieure est recouverte d'or (couche semi transparente) et la piste interne d'aluminium. La lecture de l'un ou l'autre des niveaux, se fait on focalisant le laser sur la piste désirée.

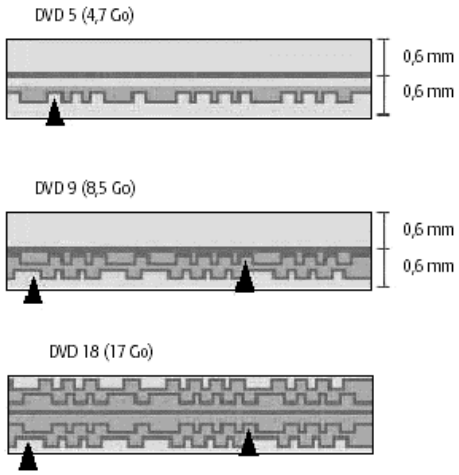


Figure 60 Les différents types de DVD vidéo et leur capacité de stockage.

5.7. *Exercices*

Exercice 1 Compteurs

1. Réaliser un diviseur de fréquence par 2 (autrement dit une bascule T) avec une bascule D.

2. Réaliser un compteur par 16 asynchrone avec des bascules D.

3. Comment connecter des blocs de compteurs par 16 asynchrones pour obtenir un compteur asynchrone à 12 bits ?

Exercice 2 Registres

On considère le registre à décalage 74AC11194 dont la table de vérité et le schéma logique sont donnés page 117.

1. Quelles doivent être les valeurs des entrées pour avoir un fonctionnement du registre en mode parallèle - parallèle ?

2. Quelles doivent être les valeurs des entrées pour avoir un fonctionnement du registre en mode série avec un décalage vers la droite ?

3. Même question pour avoir un décalage vers la gauche.

4. Compléter le chronogramme de la Figure 61 en considérant que toutes les sorties Q_i sont au niveau bas à l'instant initial.

Exercice 3 Mémoires

On considère une mémoire de format 1 octet et de capacité 4 kilo-octets

1. Combien cette mémoire a-t-elle d'entrées d'adresse et de données?

2. Les entrées d'adresse de la mémoire sont reliées à un compteur qui est lui même piloté par un horloge à 10 Mhz. Combien faudra-t-il de temps au minimum pour lire intégralement la mémoire?

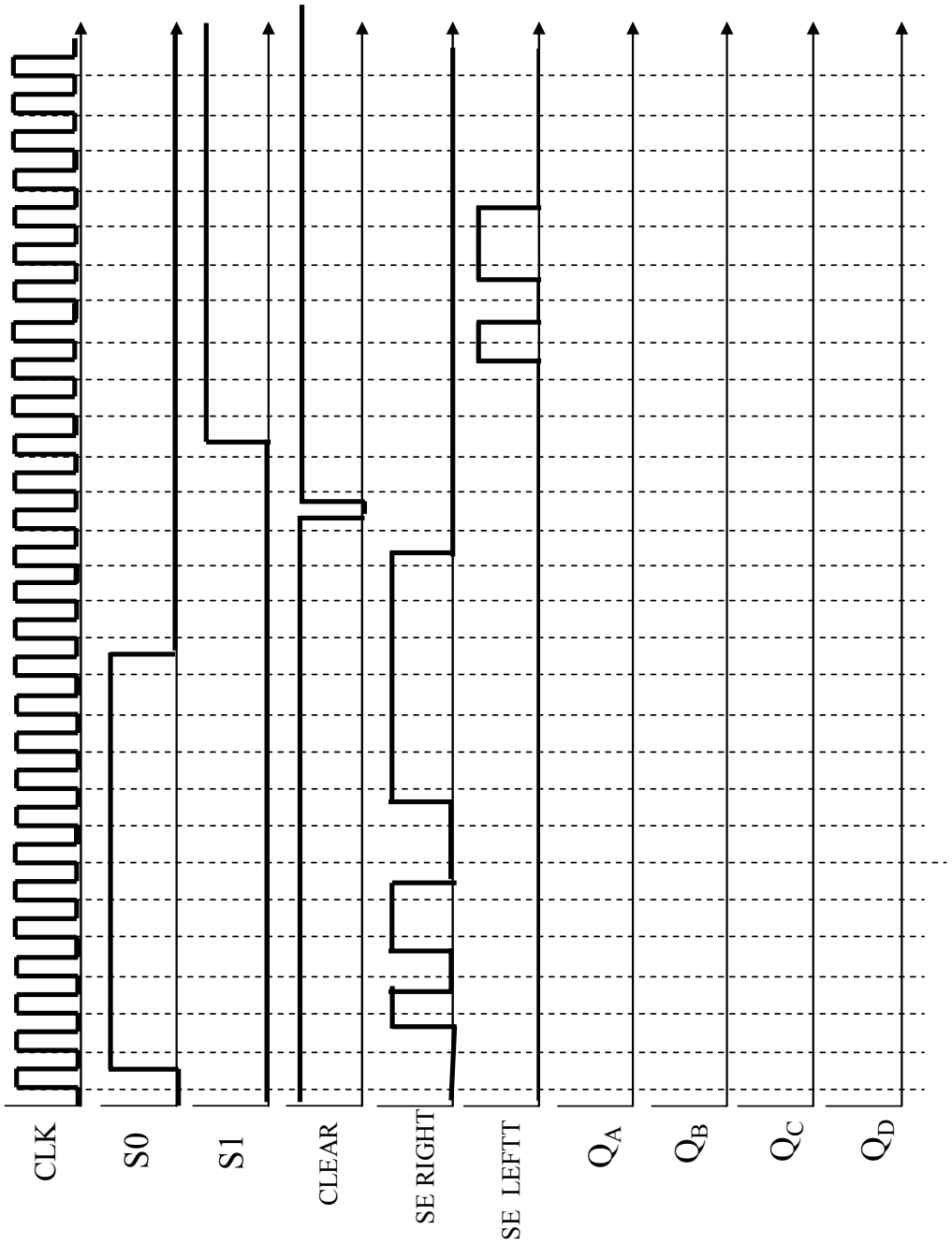
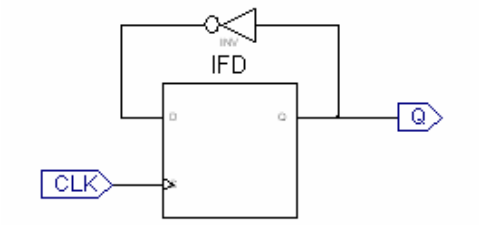


Figure 61

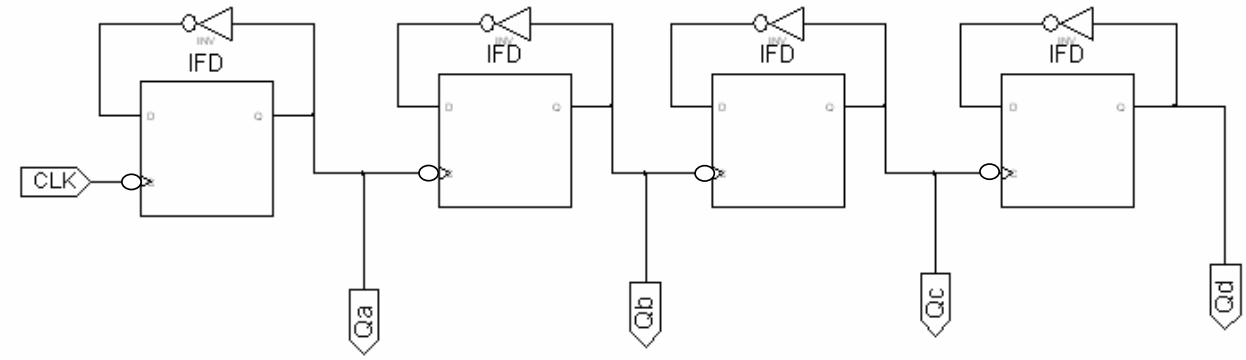
5.8. *Correction des exercices.*

Exercice 1

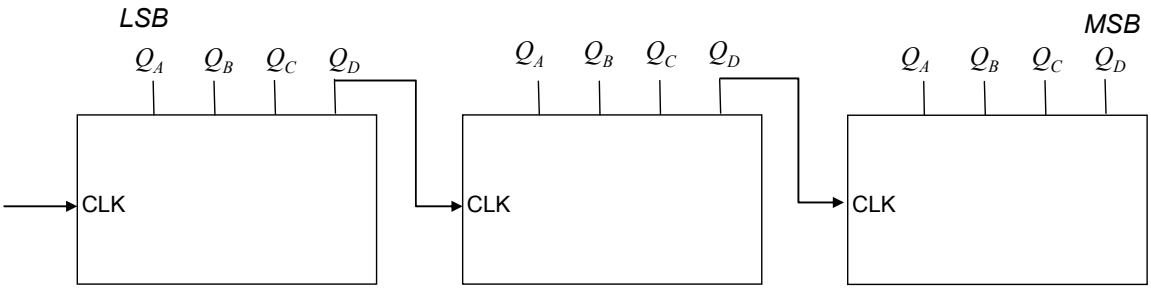
1. Il suffit e reboucler la sortie inversée sur l'entrée



2. L'architecture est la même que pour le compteur par 16 asynchrone réalisé avec des bascules *JK*.



3. On doit connecter en série trois compteurs par 16 asynchrones. On obtient de cette façon un compteur par 4096 (capacité de comptage de 0 à 4095)



Exercice 2

1. Le mode de fonctionnement parallèle-parallèle correspond à la 3éme ligne de la table de vérité. On doit donc avoir $S_1 = S_0 = 1$.

2. Le mode de fonctionnement série avec un décalage vers la droite est décrit sur les lignes 4 et 5 de la table de vérité. Lorsque $S_1 = 0$ et $S_0 = 1$, la valeur présente sur l'entrée serial right est recopiée sur la sortie Q_A au moment du front montant sur l'entrée horloge. Dans le même temps la valeur précédemment présente sur Q_A est décalée sur Q_B et ainsi de suite …

3. Le décalage vers la gauche est obtenu pour $S_1 = 1$ et $S_0 = 0$. C'est maintenant la valeur présente sur l'entrée serial left qui est transférée sur Q_D.

4. Le chronogramme est représenté sur la Figure 62. On notera en particulier l'effet de l'entrée asynchrone CLEAR qui remet toute les sorties au niveau bas.

Exercice 3

1. La mémoire est constituée de 4096 registres de 1 bit. Il faut donc 12 bits pour coder les adresses (2^{12} = 4096). La mémoire a donc une entrée de donnée et 12 entrées pour l'adresse.

2. Pour engendrer les adresses correspondant à tous les registres le compteur doit compter entre 0 et 4095. Excité par une horloge à 10 MHz il lui faudra le temps $t = 4096 / 10^5 = 41 ms$.

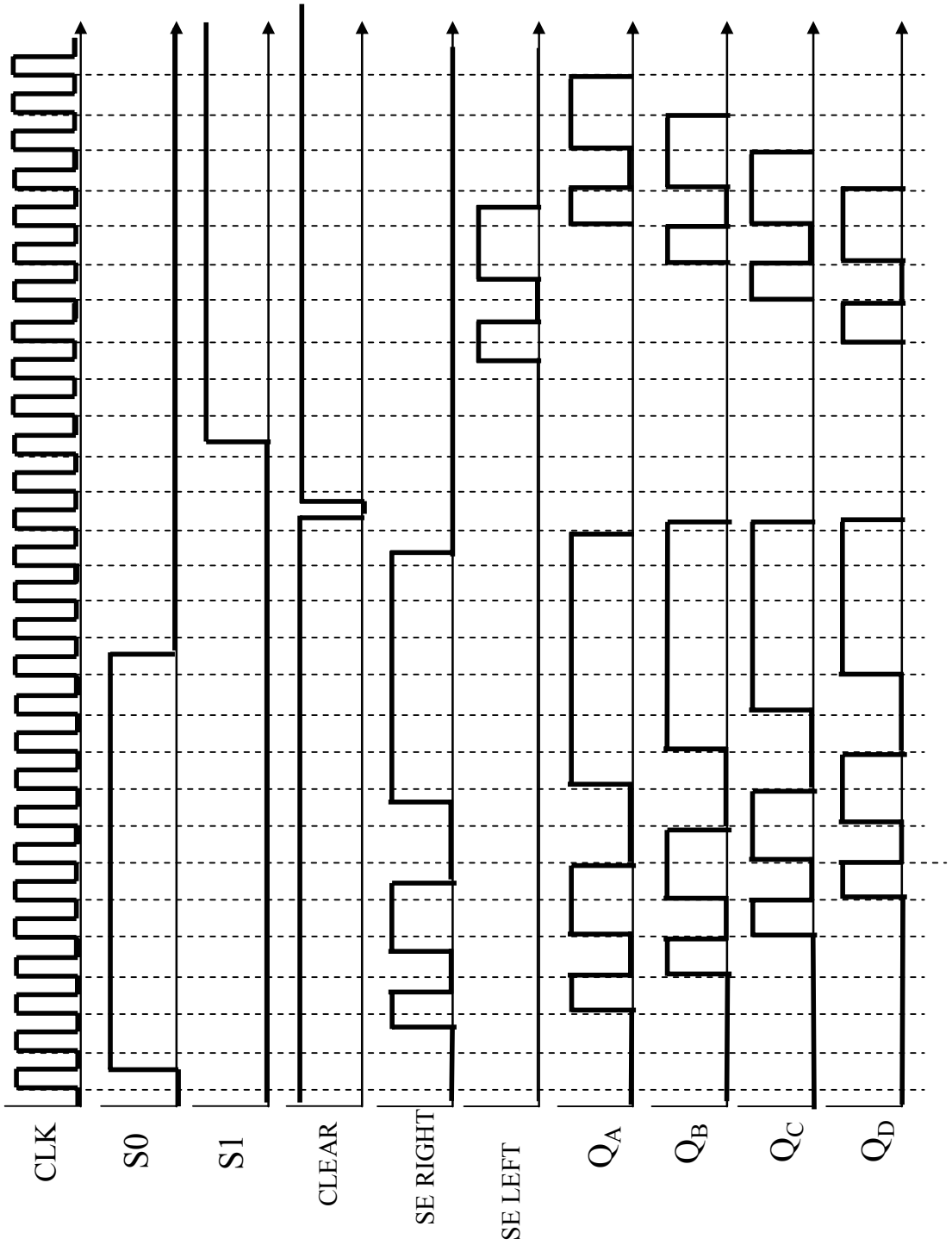


Figure 62