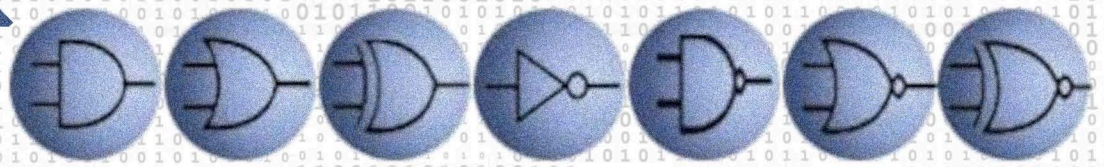


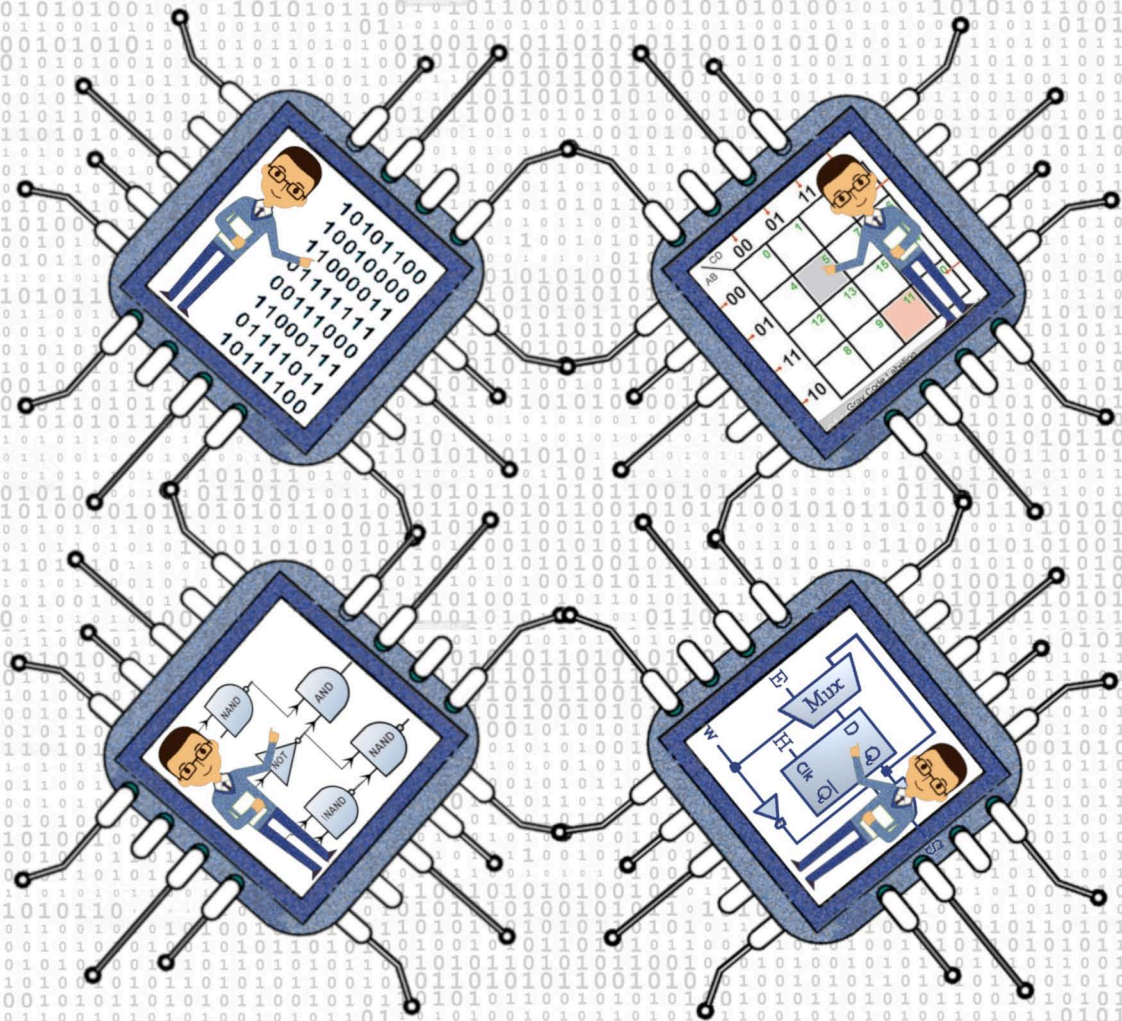
MI



STRUCTURE MACHINE

Cours avec exemples explicatifs

Support de Cours



Déstiné aux étudiants en première année licence (LMD)

Mathématiques et informatique (MI)

2020
2021

Elaboré par :

Dr. BOULAICHE AMMAR

Maitre de conférences-Département d'informatique
Faculté des sciences exactes et informatique
Université de Jijel

Avant-propos

Ce support de cours représente le fruit d'un enseignement délivré depuis de nombreuses années aux étudiants en graduation. Il est destiné principalement aux étudiants de la première année tronc commun Mathématiques et Informatique (MI) de l'université de Jijel. Mais il peut également être utilisé par toute personne ayant besoin d'acquérir les notions de base des systèmes de numération, de l'algèbre de Boole et des circuits numériques.

Le contenu de ce support recouvre le programme du module structure machine I et II selon le dernier syllabus délivré par le ministère de l'enseignement supérieur et de la recherche scientifique. Les chapitres de ce support sont présentés d'une façon simple et facile, dans lesquels l'étudiant trouvera une partie cours suffisamment détaillée expliquant les notions fondamentales abordées avec des exemples permettant d'appliquer directement ces notions.

Pour terminer, nous espérons que le présent support répondra aux attentes des étudiants quelque soit leur niveau, et qu'il les aidera à progresser dans leurs études et les mettra sur la voie du succès dans leur cursus.

Table des matières

Avant-propos	I
1 Représentation des données	1
1.1 Introduction	1
1.2 Changements de bases	1
1.2.1 Conversion du décimal en une base b quelconque	2
1.2.2 Conversion d'une base b quelconque en décimal	3
1.2.3 Conversion d'une base $b = 2^n$ quelconque en binaire	3
1.2.4 Conversion du binaire en une base $b = 2^n$ quelconque	3
1.2.5 Conversion d'une base b_1 en une base b_2 ($b_1 \neq b_2 \neq 10$)	3
1.3 Opérations arithmétiques en base b quelconque	4
1.3.1 Addition	4
1.3.2 Soustraction	4
1.3.3 Multiplication	5
1.3.4 Division	5
1.4 Représentation des données	5
1.4.1 Représentation des nombres entiers	6
1.4.1.1 Entiers naturels	6
1.4.1.2 Entiers relatifs	6
1.4.2 Représentation des nombres réels	7
1.4.2.1 Représentation en virgule fixe	7
1.4.2.2 Représentation en virgule flottante	8
1.4.2.2.1 Passage du décimal vers la virgule flottante	8
1.4.2.2.2 Passage de la virgule flottante vers le décimal	9
1.4.3 Représentation des données non numériques (textes)	9
1.4.3.1 Code ASCII	10
1.4.3.2 Code ASCII étendu	12
1.4.3.3 Encodage UNICODE	12
1.4.4 Autres codes particuliers	13
1.4.4.1 Code BCD	13
1.4.4.1.1 Addition en BCD	13
1.4.4.1.2 Soustraction en BCD	13
1.4.4.2 Code Gray	14
1.4.4.2.1 Transcodage Binaire - Gray	14
1.4.4.2.2 Transcodage Gray - Binaire	14
1.4.4.3 Codes détecteurs d'erreur	15
2 Algèbre de Boole	16
2.1 Introduction	16
2.2 Définition de l'algèbre de Boole	16

2.3	Théorèmes fondamentaux de l'algèbre de Boole	17
2.4	Fonctions de base de l'algèbre de Boole	17
2.5	Représentation d'une fonction logique	18
2.5.1	Table de vérité d'une fonction logique	18
2.5.2	Logigramme d'une fonction logique	18
2.5.3	Chronogramme d'une fonction logique	19
2.6	Forme canonique d'une fonction logique	19
2.6.1	Définitions	19
2.6.2	Mettre une fonction en première forme canonique	20
2.6.2.1	La méthode algébrique	20
2.6.2.2	La méthode de la table de vérité	20
2.6.3	Mettre une fonction en deuxième forme canonique	20
2.6.3.1	La méthode algébrique	21
2.6.3.2	La méthode de la table de vérité	21
2.6.4	Représentation décimale des deux formes canoniques	21
2.6.5	Fonctions logiques avec uniquement des portes NAND et NOR .	22
2.6.5.1	Expression d'une fonction logique avec des portes NAND	22
2.6.5.2	Expression d'une fonction logique avec des portes NOR	22
2.7	Simplification des fonctions logiques	23
2.7.1	Méthode algébrique	23
2.7.2	Méthode de karnaugh	24
2.7.2.1	Table de karnaugh	24
2.7.2.1.1	Caractéristiques de la table de karnaugh . . .	24
2.7.2.1.2	Table de Karnaugh à partir de la table de vérité	24
2.7.2.1.3	Table de Karnaugh à partir de la forme canonique	25
2.7.2.2	Méthode de simplification	25
2.7.2.3	Tables de karnaugh avec des valeurs inconnues	25
2.7.2.4	Tables de karnaugh à cinq variables	26
2.7.2.5	Tables de karnaugh à six variables	27
2.7.3	Méthode de Quine Mc Cluskey	28
3	Circuits logiques combinatoires	30
3.1	Introduction	30
3.2	Synthèse des systèmes combinatoires	30
3.3	Synthèse des principaux circuits combinatoires	31
3.3.1	L'additionneur	31
3.3.1.1	Le semi-additionneur	31
3.3.1.2	L'additionneur complet	31
3.3.1.3	L'additionneur n bits	33
3.3.2	Le comparateur 2 bits	33
3.3.3	Le décodeur	35
3.3.4	L'encodeur	36
3.3.5	Le multiplexeur	36
3.3.6	Le démultiplexeur	37
3.3.7	Génération des fonctions logiques via des multiplexeurs	38
3.3.8	Génération des fonctions logiques via des décodeurs	39

4	Circuits logiques séquentiels	41
4.1	Introduction	41
4.2	Modèles des circuits séquentiels	41
4.2.1	Machines de Mealy	41
4.2.2	Machines de Moore	42
4.3	Automates à états finis (graphes de transitions)	42
4.3.1	Types de graphes de transitions	42
4.3.1.1	Graphes de Moore	42
4.3.1.2	Graphes de Mealy	43
4.3.2	Passage du graphe de Moore au graphe de Mealy	43
4.3.3	Passage du graphe de Mealy au graphe de Moore	44
4.4	Circuits séquentiels à base de bascules	44
4.4.1	Types de bascules	44
4.4.1.1	Bascules asynchrones	44
4.4.1.2	Bascules synchrones	45
4.4.2	Bascules de base des circuits séquentiels	45
4.4.2.1	La bascule RS	45
4.4.2.2	La bascule RSH	46
4.4.2.3	La bascule D	48
4.4.2.4	La bascule JK	48
4.4.2.5	La bascule T	49
4.4.3	Analyse des circuits séquentiels à bascules	50
4.4.4	Synthèse des circuits séquentiels à bascules	51
4.5	Application des circuits séquentiels à bascules	53
4.5.1	Les registres	53
4.5.1.1	Définition	53
4.5.1.2	Types de registres	54
4.5.1.2.1	Registres à entrées parallèles et sorties parallèles	54
4.5.1.2.2	Registres à entrées parallèles et sorties série	55
4.5.1.2.3	Registres à entrées série et sorties parallèles	55
4.5.1.2.4	Registres à entrées série et sorties série	56
4.5.2	Les compteurs	57
4.5.2.1	Les compteurs asynchrones	57
4.5.2.1.1	Compteurs asynchrones à cycle complet . . .	57
4.5.2.1.2	Compteurs asynchrones à cycle incomplet . .	58
4.5.2.1.3	Inconvénients des compteurs asynchrones . . .	58
4.5.2.2	Les compteurs synchrones	59
4.5.3	Les mémoires à semi-conducteurs	60
5	Circuits intégrés	63
5.1	Introduction	63
5.2	Définition	63
5.3	Construction des circuits intégrés	64
5.4	Familles des circuits intégrés	66
5.4.1	Circuits intégrés TTL	66
5.4.2	Circuits intégrés CMOS	66
5.5	Circuits intégrés TTL des portes logiques de base	67
5.6	Montage d'un circuit combinatoire via des circuits intégrés	68

Chapitre 1

Représentation des données

1.1 Introduction

Les données traitées par un ordinateur sont généralement de différents types. Elles peuvent être des textes, des nombres, des images, des vidéos, etc. Cependant, quel que soit leur type, ces données sont toujours stockées et manipulées par l'ordinateur sous forme binaire. Elles seront donc traitées comme une suite de 0 et de 1. Dans une telle représentation, l'unité d'information est le chiffre binaire (0 ou 1), que l'on appelle bit (pour binary digit, chiffre binaire).

Pour pouvoir traiter l'information dans l'ordinateur, il faut donc que cette dernière soit codée en binaire. Le codage d'une information consiste à établir une correspondance entre la représentation externe de cette information (le mot *Bien* ou le nombre 317 par exemple), et sa représentation interne dans l'ordinateur, qui est une suite de 0 et de 1.

On utilise la représentation binaire car elle est simple, facile à réaliser techniquement à l'aide de bistables (système à deux états réalisés à l'aide de transistors, voir le chapitre 4). Enfin, les opérations arithmétiques de base (addition, multiplication, etc.) sont faciles à exprimer en base 2 (noter que la table de multiplication se résume à $0 \times 0 = 0$, $1 \times 0 = 0$ et $1 \times 1 = 1$).

1.2 Changements de bases

Dans cette première partie de ce chapitre et avant d'aborder la représentation des différents types de données, nous devons tout d'abord nous familiariser avec la représentation d'un nombre dans une base b quelconque.

Dans nos calculs quotidiens, on utilise le système décimal (base 10) pour représenter les nombres, c'est-à-dire que l'on écrit à l'aide de 10 chiffres distincts. Ce sont les chiffres de 0 à 9.

Ainsi, en base b , on utilise b chiffres distincts. Ce sont les chiffres de 0 à $b-1$. Notons a_i la suite des chiffres utilisés pour écrire un nombre, chacun de ces chiffres est placé dans une position dans ce nombre. C'est ce que l'on appelle la notation positionnelle.

– Cas des nombres entiers

$$x = a_n a_{n-1} \dots + a_1 a_0 = \sum a_i \times b^i$$

a_0 est le chiffre de poids faible, et a_n le chiffre de poids fort.

Exemples : $(1996)_{10} = 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 6 \times 10^0$
 $(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = (5)_{10}$

La notation $()_b$ indique que le nombre est écrit en base b .

- **Cas des nombres fractionnaires** (Les nombres fractionnaires sont ceux qui comportent des chiffres après la virgule).

$$x = a_n \dots a_1 a_0, a_{-1} \dots a_{-p} = a_n \times b^n + \dots + a_0 \times b^0 + a_{-1} \times b^{-1} + \dots + a_{-p} \times b^{-p}$$

Exemple : $(12,346)_{10} = 1 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2} + 6 \times 10^{-3}$

Remarques :

1. En décimal, $b = 10$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
2. En binaire, $b = 2$, $a_i \in \{0, 1\}$; 2 chiffres binaires, ou bits;
3. En hexadécimal, $b = 16$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
 (on utilise les 6 premières lettres de l'alphabet comme des chiffres).

1.2.1 Conversion du décimal en une base b quelconque

- **Cas des nombres entiers :**

Dans le cas des nombres entiers (les nombres qui ne comportent pas des chiffres après la virgule), la conversion s'effectue par des divisions entières successives par la base b . On divise le nombre par la base, puis le quotient obtenu par la base, et ainsi de suite jusqu'à ce que l'on obtienne un quotient nul. Le nombre en base b résultant est obtenu en lisant les restes du dernier vers le premier.

Exemple : Conversion du nombre décimal 13 en binaire ($b = 2$) :

$$\begin{array}{r|l} 13 & 2 \\ \hline 1 & 6 \\ \hline 0 & 3 \\ \hline 1 & 1 \\ \hline 1 & 0 \end{array} \Rightarrow (13)_{10} = (1101)_2$$

- **Cas des nombres fractionnaires :**

Dans le cas des nombres fractionnaires (les nombres qui comportent des chiffres après la virgule, notés : $a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} \dots a_{-p}$), la partie entière est convertie par des divisions entières successives par b comme pour un nombre entier, alors que la partie fractionnaire est convertie par une multiplication successive par la base en répétant l'opération sur la partie fractionnaire du produit jusqu'à ce qu'elle soit nulle (ou que la précision voulue soit atteinte).

Exemple : Conversion du nombre décimal 13,125 en binaire ($b = 2$) :

- **Partie entière :** $(13)_{10} = (1101)_2$ par des divisions entières successives.

- **Partie fractionnaire :**

$$\begin{array}{llll} 0,125 \times 2 = 0,25 & \Rightarrow & a_{-1} = 0 & \text{Poids binaire } 0 \times 2^{-1} \\ 0,25 \times 2 = 0,50 & \Rightarrow & a_{-2} = 0 & \text{Poids binaire } 0 \times 2^{-2} \\ 0,5 \times 2 = 1,00 & \Rightarrow & a_{-3} = 1 & \text{Poids binaire } 1 \times 2^{-3} \end{array}$$

Condition d'arrêt (partie fractionnaire du résultat égale à 0).

Donc, $(13,125)_{10} = (1101,001)_2$

1.2.2 Conversion d'une base b quelconque en décimal

La conversion se fait simplement en additionnant les produits des chiffres du nombre à convertir par les puissances de la base b correspondantes.

$$(a_n \dots a_0, a_{-1} \dots a_{-p})_b = (a_n \times b^n + \dots + a_0 \times b^0 + a_{-1} \times b^{-1} + \dots + a_{-p} \times b^{-p})_{10}$$

Exemples :

$$(10101, 01)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (21, 25)_{10}$$

$$(35, 4)_8 = 3 \times 8^1 + 5 \times 8^0 + 4 \times 8^{-1} = (29, 5)_{10}$$

$$(2F, 8)_{16} = 2 \times 16^1 + 15 \times 16^0 + 8 \times 16^{-1} = (47, 5)_{10}$$

1.2.3 Conversion d'une base $b = 2^n$ quelconque en binaire

La conversion consiste tout simplement à remplacer chaque chiffre par son équivalent binaire sur n bits.

Exemples :

$$(17, 5)_8 = \underbrace{001}_1 \underbrace{111}_7, \underbrace{101}_5 = (1111, 101)_2,$$

$$(17, 5)_{16} = \underbrace{0001}_1 \underbrace{0111}_7, \underbrace{0101}_5 = (10111, 0101)_2$$

1.2.4 Conversion du binaire en une base $b = 2^n$ quelconque

La conversion consiste à découper la séquence des chiffres binaires en un ensemble de groupes de n bits chacun. Le découpage doit commencer de la virgule vers la droite et vers la gauche. Si le nombre de bits de la partie entière (partie gauche) n'est pas un multiple de n , on doit compléter par des zéros à gauche de la séquence. De même, si le nombre de bits de la partie fractionnaire (partie droite) n'est pas un multiple de n , on doit compléter par des zéros à droite de la séquence. Après avoir découpé la séquence des chiffres binaire en groupes de n bits, on remplace chaque groupe par le chiffre de la base b correspondant.

Exemples :

$$(11011, 01)_2 = \underbrace{011}_3 \underbrace{011}_3, \underbrace{010}_2 = (33, 2)_8,$$

$$(11011, 01)_2 = \underbrace{0001}_1 \underbrace{1011}_B, \underbrace{0100}_4 = (1B, 4)_{16}$$

1.2.5 Conversion d'une base b_1 en une base b_2 ($b_1 \neq b_2 \neq 10$)

Pour réaliser une telle conversion, on doit passer par la base décimale comme base intermédiaire. C'est à dire, on doit d'abord convertir le nombre de la base b_1 en décimal avant de convertir, ensuite, le nombre décimal résultant en base b_2 .

Exemple : $(25, 3)_6 = (?)_8 \Rightarrow (25, 3)_6 = (?)_{10} = (?)_8$

$$(25, 3)_6 = 2 \times 6^1 + 5 \times 6^0 + 3 \times 6^{-1} = (17, 5)_{10},$$

$$(17, 5)_{10} = \boxed{2} \times 8^1 + \boxed{1} \times 8^0 + \boxed{4} \times 8^{-1} = (21, 4)_8,$$

$$\text{Donc, } (25, 3)_6 = (21, 4)_8$$

- **Cas particulier :** Lorsque $b_1 = 2^n$ et $b_2 = 2^m$ ($n \neq m$)

Dans le cas où b_1 et b_2 sont toutes les deux des puissances de 2, il est plus facile de passer par le binaire comme base intermédiaire que de passer par le décimal.

Exemple : $(65, 7)_8 = (?)_{16} \Rightarrow (65, 7)_8 = (?)_2 = (?)_{16}$

$$(65, 7)_8 = \underbrace{110}_6 \underbrace{101}_5 \underbrace{111}_7 = (110101, 111)_2$$

$$(110101, 111)_2 = \underbrace{0011}_3 \underbrace{0101}_5 \underbrace{1110}_E = (35, E)_{16}$$

Donc, $(65, 7)_8 = (35, E)_{16}$

1.3 Opérations arithmétiques en base b quelconque

Les opérations arithmétiques de base, telles que l'addition, la soustraction, la multiplication et la division dans une base b quelconque sont effectuées de la même façon qu'en décimal.

1.3.1 Addition

L'addition en base b se fait généralement en suivant les mêmes règles qu'en décimal. C'est à dire, on commence à additionner les chiffres de poids faibles (les chiffres de droite) jusqu'au chiffre de poids le plus fort (celui tout à gauche). Lorsque la somme de deux chiffres de poids égaux dépasse la valeur de la base, on divise la somme par la base et on reporte le quotient aux chiffres du poids suivant (à gauche) et on inscrit le reste comme somme des deux chiffres du poids en cours.

Exemple : $(15, 4)_6 + (12, 5)_6 = (?)_6 \Rightarrow \left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_6$

1.3.2 Soustraction

Pour la soustraction en base b , on procède comme en décimal. Quand la quantité à soustraire est supérieure à la quantité dont on soustrait, on emprunte 1 au voisin de gauche. Ce 1 emprunté ajoute la valeur de la base b à la quantité dont on soustrait afin de pouvoir réaliser la soustraction en cours. Ensuite, on retranche cet emprunt du bit de gauche et on fait la même chose pour les bits qui restent.

Exemple 1 :

$$(20)_3 - (2)_3 \Rightarrow \left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_3$$

Dans cet exemple, on doit soustraire $0 - 2$ pour les deux bits de poids faible, ce qui n'est pas possible car 0 est inférieur à 2. On emprunte donc 1 au bit de gauche ce qui va donner $(3 + 0) - 2 = 1$. Ensuite, on retranche cet emprunt du bit de gauche et on aura donc $2 - 0 - 1 = 1$.

Exemple 2 : $(1010)_2 - (0111)_2 = (?)_2$

$$\left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_2 \Rightarrow \left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_2 \Rightarrow \left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_2 \Rightarrow \left(\begin{array}{r} \\ \\ \\ \hline \end{array} \right)_2$$

1.3.3 Multiplication

La multiplication en base b est effectuée de la même façon qu'en décimal, en formant un produit partiel pour chaque chiffre du multiplieur (sauf que la table du produit change d'une base à une autre). Pour finir, on additionne les résultats des différents produits partiels.

Exemples :

$$(15, 4)_6 \times (2, 5)_6 \Rightarrow \left(\begin{array}{r} 13, 4 \\ \times 2, 5 \\ \hline 13 2 \\ 3 2 \\ \hline 5, 2 \end{array} \right)_6 ; (1101)_2 \times (101)_2 \Rightarrow \left(\begin{array}{r} 10 \\ \times 11 \\ \hline 10 \\ 00 \\ 10 \\ \hline 1001 \end{array} \right)_2$$

1.3.4 Division

La division en base b s'effectue comme en décimal à l'aide de soustractions et de décalages, sauf que la table de division change d'une base à une autre ce qui rend l'opération un peu compliquée. Cependant, cette opération est plus facile à effectuer en binaire. On prend le nombre de chiffres nécessaires pour la première division, puis on pose un 1 en quotient et on fait la soustraction, par la suite on abaisse un chiffre et on pose un 1 en quotient si l'on peut soustraire le diviseur, ou un 0 sinon et on recommence la même opération tant qu'il y a des chiffres qui ne sont pas encore abaissés.

Exemple 1 :

$$\left(\begin{array}{l} 1\ 0\ 1\ 1\ 0 \mid 1\ 0 \\ - \\ \hline 0\ 0 \end{array} \right)_2 \Rightarrow \left(\begin{array}{l} 1\ 0\ 1\ 1\ 0 \mid 1\ 0 \\ - \\ \hline 0\ 0\ 1\ 1 \\ - \\ \hline 0\ 1 \end{array} \right)_2 \Rightarrow \left(\begin{array}{l} 1\ 0\ 1\ 1\ 0 \mid 1\ 0 \\ - \\ \hline 0\ 0\ 1\ 1 \\ - \\ \hline 0\ 1\ 0 \\ - \\ \hline 0\ 0 \end{array} \right)_2$$

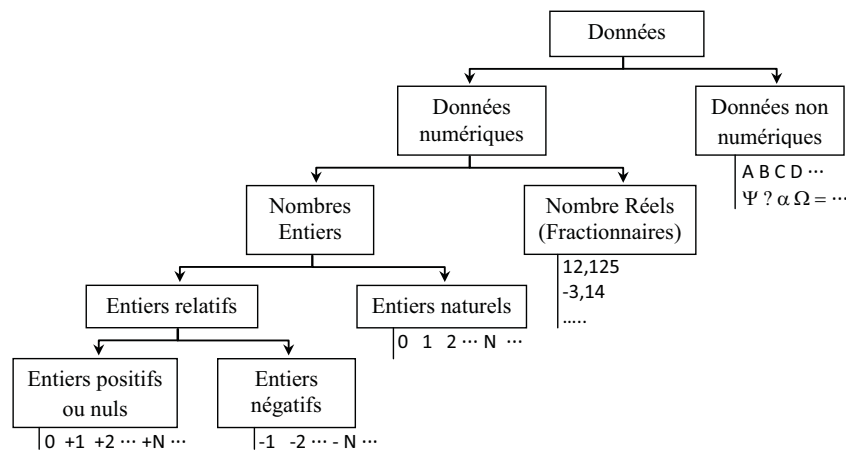
Exemple 2 :

$$(1133, 12)_4 \div (1, 3)_4 \Rightarrow \left(\begin{array}{l} 1\ 1\ 3\ 3\ 1, 2 \mid 1\ 3 \\ - \\ \hline 0\ 0\ 2\ 3 \\ - \\ \hline 1\ 1\ 0\ 1 \\ - \\ \hline 0\ 0\ 3\ 2 \\ - \\ \hline 0\ 0 \end{array} \right)_4$$

$$\begin{aligned} (0)_4 \times (13)_4 &= (0)_4 \\ (1)_4 \times (13)_4 &= (13)_4 \\ (2)_4 \times (13)_4 &= (32)_4 \\ (3)_4 \times (13)_4 &= (111)_4 \end{aligned}$$

1.4 Représentation des données

Comme indiqué plus haut, les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, images, vidéos, etc.) mais elles sont toujours représentées et manipulées par l'ordinateur sous forme binaire. Dans cette partie, on va se focaliser sur la représentation interne, dans la machine, des données numériques (celles qui peuvent être l'objet d'une opération arithmétique) et des données non numériques (par exemple, les symboles et les caractères constituant un texte). Les données traitées dans cette partie du chapitre ainsi que leur classification sont résumées par le graphe suivant :



1.4.1 Représentation des nombres entiers

1.4.1.1 Entiers naturels

Dans un ordinateur, les données sont codées sur un nombre de bits bien fixé, n bits par exemple. On rencontre habituellement des codages sur 8, 16, 32 ou 64 bits. Ainsi, les nombres entiers naturels sont représentés à l'intérieur de l'ordinateur par une simple conversion de ceux-ci en binaire de sorte que le résultat soit sur n bits, comme exigé par le système. Par conséquent, les résultats qui n'atteignent pas les n bits sont complétés à gauche par des zéros. Par ailleurs, si le nombre de bits du nombre converti dépasse les n bits exigés, celui-ci est considéré comme irréprésentable. En fait, un codage sur n bits ne permet de représenter que les nombres naturels compris entre 0 et 2^{n-1} . Par exemple sur 1 octet, on pourra coder les nombres naturels de 0 à $255 = 2^{8-1}$.

Exemple :

Représentation du nombre naturel 21 sur 8 bits : $(21)_{10} = (10101)_2 \Rightarrow (00010101)_2$

Compléter par des zéros pour atteindre les 8 bits exigés

1.4.1.2 Entiers relatifs

Un entier relatif est un entier pouvant être positif ou négatif. Il possède en plus des chiffres qui le composent un autre symbole qui précise son signe. Il faut donc une méthode pour représenter ce type de nombres de sorte que l'on puisse les coder avec leur signe, tout en conservant les règles des opérations arithmétiques. Comme solution, plusieurs représentations ont été proposées, les plus connues sont : la représentation en signe avec valeur absolue (S+VA), la représentation en complément logique (complément restreint ou complément à 1) et la représentation en complément arithmétique (complément vrai ou complément à 2).

1. **Représentation en signe et valeur absolue (S+VA) :** Dans cette méthode, on réserve le bit de poids fort (celui étant situé le plus à gauche) pour représenter le signe, les autres bits représentent la valeur absolue du nombre. On met le bit de poids fort à 0 pour repérer un nombre positif et on met ce même bit à 1 pour repérer un nombre négatif.

Exemple : Sur 8 bits : $(00011011)_{S+VA}$ représente $(+27)_{10}$
 $(10011011)_{S+VA}$ représente $(-27)_{10}$

2. Représentation en complément à 1 (complément logique ou restreint) :

Dans cette méthode, les entiers positifs ou nuls sont représentés comme pour les entiers naturels, sauf que le bit de poids fort est toujours à 0 : on n'utilise donc que $n - 1$ bits pour représenter les valeurs de ces nombres. Alors que les entiers négatifs sont représentés en inversant les valeurs des n bits constituant la valeur binaire de leurs opposés (leurs équivalents en positif), c'est à dire chaque bit à 0 de ces derniers est remplacé par un 1 et vice-versa.

En d'autre terme, pour obtenir le codage en complément à 1 d'un nombre négatif, on code en binaire son équivalent positif (sa valeur absolue) sur n bits, puis on complémente (ou inverse) tous les bits (c'est à dire, remplacer les 0 par des 1 et les 1 par des 0).

Exemple : Sur 8 bits : $(00011011)_{C1}$ représente $(+27)_{10}$
 $(11100100)_{C1}$ représente $(-27)_{10}$

3. Représentation en complément à 2 (complément arithmétique ou vrai) :

Dans cette méthode, les entiers positifs ou nuls sont toujours représentés comme pour les entiers naturels (le bit de poids fort est toujours à 0, comme en complément à 1) et les entiers négatifs sont représentés en ajoutant un 1 à leur représentation en complément à 1.

En d'autre terme, pour obtenir le codage en complément à 2 d'un nombre négatif, on code en binaire son équivalent positif (sa valeur absolue) sur n bits, puis on complémente (ou inverse) tous les bits et on ajoute 1 au codage obtenu.

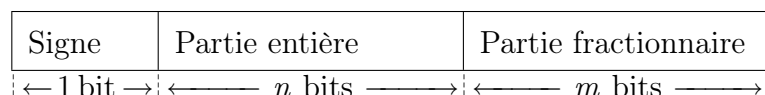
Exemple : Sur 8 bits : $(00011011)_{C2}$ représente $(+27)_{10}$
 $(11100101)_{C2}$ représente $(-27)_{10}$

1.4.2 Représentation des nombres réels

Deux méthodes principales sont utilisées pour représenter les nombres réels à l'intérieur d'un ordinateur : la représentation en virgule fixe et celle en virgule flottante.

1.4.2.1 Représentation en virgule fixe

La représentation en virgule fixe est la méthode la plus simple et la plus utilisée sur les ordinateurs des premières générations. Le principe de cette représentation consiste, comme le montre la figure suivante, à réserver une partie de n bits pour représenter la partie entière du réel et une partie de m bits pour représenter la partie fractionnaire du réel, les deux parties sont précédées par un bit indiquant le signe du réel.



Exemple :

Sur 16 bits : $(1|0101011|10100000)_{VFixe}$ représente $(-101011, 101)_2 = (-43, 625)_{10}$

En effet, la représentation en virgule fixe est une méthode très simple et permet d'effectuer les conversions très rapidement. Toutefois, cette structure de codage est trop rigide. Elle ne permet pas de représenter des nombres très grands, comme par exemple le nombre $1,5 \times 10^{15}$, ou ceux qui sont très proche de zéro, comme par exemple le

nombre $1,5 \times 10^{-15}$. Comme solution à ces problèmes, une autre méthode, à base de virgule flottante, a été introduite.

1.4.2.2 Représentation en virgule flottante

La représentation des nombres réels à base de virgule flottante est la méthode la plus répandue dans la plupart des ordinateurs d'aujourd'hui, en particulier la représentation en virgule flottante avec la norme IEEE 754. Dans cette représentation, le nombre fractionnaire N est représenté sous la forme suivante :

Signe (S)	Exposant décalé (ED)	Mantisse (M)
← 1 bit →	← n bits →	← m bits →

Les champs de cette représentation sont extraits de la notation dite scientifique du nombre à représenter, qui est donnée par la formule suivante :

$$N = (-1)^S \times 1, M \times 2^{E=ED-D}$$

L'exposant décalé ED est donc calculé par la formule : $ED = E + D$. Où E est l'exposant non décalé, alors que D est le décalage (biais) de l'exposant, c'est une valeur fixe donnée par la formule suivante (selon le format utilisé) :

$$D = 2^{\text{Taille de ED}-1} - 1$$

La taille des trois champs (S , ED et M) ainsi que la valeur de D sont liées de manière directe au format de représentation utilisé. Il y a trois formats pour la norme IEEE 754 :

- **Format simple précision (sur 32 bits)** : 1 bit pour le signe, 8 bits pour l'exposant décalé et 23 bits pour la mantisse. Dans ce format, la valeur de décalage D est égale à 127.
- **Format double précision (sur 64 bits)** : 1 bit pour le signe, 11 bits pour l'exposant décalé et 52 bits pour la mantisse. Dans ce format, la valeur de décalage D est égale à 1023.
- **Format précision étendue (sur 80 bits)** : 1 bit pour le signe, 15 bits pour l'exposant décalé et 64 bits pour la mantisse. Dans ce format, la valeur de décalage D est égale à 16383.

1.4.2.2.1 Passage du décimal vers la représentation en virgule flottante

Pour représenter un nombre réel en virgule flottante selon la norme IEEE 754, on va suivre les étapes suivantes :

1. Convertir le nombre en binaire.
2. Normaliser le nombre en base 2, en l'écrivant sous la forme : $(-1)^S \times 1, M \times 2^E$.
3. Calculer l'exposant décalé : $ED = E + D$, et convertir le résultat en binaire.
4. Remplir les trois champs de la représentation (S , ED et M).

Exemple : $(-21,625)_{10} = (?)_{IEEE754(SP)}$

1. $(-21,625)_{10} = (-10101,101)_2$.

2. $(-10101, 101)_2 = (-1)^1 \times 1, 0101101 \times 2^4$.
3. $ED = E + D = 4 + 127 = (131)_{10} = (10000011)_2$
4.

1	10000011	010110100000000000000000
---	----------	--------------------------

Donc, $(-21, 625)_{10} = (11000001101011010000000000000000)_{IEEE754(SP)}$

1.4.2.2.2 Passage de la virgule flottante vers le décimal

Pour extraire la valeur décimale représentée par un code en virgule flottante selon la norme IEEE 754, on va suivre les étapes suivantes :

1. Extraire les trois champs de la représentation (S, ED et M).
2. Convertir l'exposant décalé en décimal, et calculer l'exposant non décalé :
 $E = ED - D$.
3. Ecrire le nombre sous la forme $(-1)^S \times 1, M \times 2^E$. et dénormaliser le résultat.
4. Convertir le résultat en décimal.

Exemple : $(11000001011011000000000000000000)_{IEEE754(SP)} = (?)_{10}$

1.

1	10000010	110110000000000000000000
---	----------	--------------------------
2. $ED = (10000010)_2 = (130)_{10} \Rightarrow E = ED - D = 130 - 127 = (3)_{10}$.
3. $N = (-1)^S \times 1, M \times 2^E = (-1)^1 \times 1, 11011 \times 2^3 = (-1110, 11)_2$
4. $(-1110, 11)_2 = (-14, 75)_{10}$

Donc, $(11000001011011000000000000000000)_{IEEE754(SP)} = (-14, 75)_{10}$

Remarque : Dans la norme IEEE 754, les exposants $00 \dots 0$ et $11 \dots 1$ sont interdits. Ils sont réservés pour représenter les cas particuliers suivants :

- La représentation :

x	$00 \dots 0$	$00 \dots 0$
-----	--------------	--------------

 code la valeur zéro ($x = 0$ ou 1).
- La représentation :

0	$11 \dots 1$	$00 \dots 0$
---	--------------	--------------

 code la valeur plus infini ($+\infty$).
- La représentation :

1	$11 \dots 1$	$00 \dots 0$
---	--------------	--------------

 code la valeur moins infini ($-\infty$).
- La représentation :

x	$00 \dots 0$	$M \neq 00 \dots 0$
-----	--------------	---------------------

 code un nombre dénormalisé (un nombre très proche de zéro).
- La représentation :

x	$11 \dots 1$	$M \neq 00 \dots 0$
-----	--------------	---------------------

 code un nombre indéterminé. On note cette configuration "NaN : Not a Number", et on l'utilise pour signaler des erreurs de calculs, comme par exemple une division par 0.

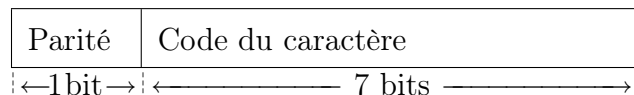
1.4.3 Représentation des données non numériques (textes)

En plus des données numériques (entiers naturels, entiers relatifs et réels) abordées dans les sections précédentes de ce chapitre, les ordinateurs peuvent également traiter des données non numériques, telles que les données *alphanumériques* qui incluent des lettres alphabétiques majuscules et minuscules, des symboles de ponctuation (& @ , . ; # ", etc.), et des chiffres.

La représentation (codage) des données *alphanumériques* est souvent faite par une table de correspondance indiquant le code binaire représentant chaque caractère. Les deux codes les plus connus et les plus utilisés actuellement sont le code ASCII (American Standard Code for Information Interchange) et l'UNICODE (UNiversal CODE).

1.4.3.1 Code ASCII

Le code ASCII est la première norme adoptée par l'*American National Standards Institute* (ANSI) dans les années 1960 pour représenter les données alphanumériques. A cette époque, les ordinateurs fonctionnaient en 8 bits, ce qui était largement suffisant pour coder tous les caractères de l'anglais, majuscules et minuscules, les chiffres de 0 à 9 et certains caractères de contrôle (fin de ligne, tabulateur, etc) et de ponctuation (& @ , . ; # ", etc.).



Pour des raisons de fiabilité, les 8 bits du code ASCII sont divisés, comme le montre la figure ci-dessus, en deux parties. Les 7 premiers bits de poids faible représentent le code du caractère à représenter, alors que le 8ième bit (bit de poids fort), appelé bit de parité, est réservé pour contrôler et détecter les erreurs de transmission, car à l'époque les ordinateurs étaient liés à des télétypes qui servaient de point d'entrée des données, ce qui nécessitait un mécanisme de contrôle permettant de détecter les erreurs produites lors du transmission des données de ce derniers vers l'ordinateur. La valeur du bit de parité est donnée en fonction du nombre de bits à 1 dans le code du caractère à représenter. Si ce dernier est pair, le bit de parité va avoir la valeur 1, sinon il va prendre la valeur 0. La table suivante représente la codification ASCII en binaire, en hexadécimal et en décimal des différents caractères alphanumériques.

Décimal	Hexa	Binaire	Caractère	Décimal	Hexa	Binaire	Caractère
0	0	0000000	NUL	32	20	0100000	ESPACE
1	1	0000001	SOH	33	21	0100001	!
2	2	0000010	STX	34	22	0100010	"
3	3	0000011	ETX	35	23	0100011	#
4	4	0000100	EOT	36	24	0100100	\$
5	5	0000101	ENQ	37	25	0100101	%
6	6	0000110	ACK	38	26	0100110	&
7	7	0000111	BEL	39	27	0100111	,
8	8	0001000	BS	40	28	0101000	(
9	9	0001001	HT	41	29	0101001)
10	A	0001010	LF	42	2A	0101010	*
11	B	0001011	VT	43	2B	0101011	+
12	C	0001100	FF	44	2C	0101100	,
13	D	0001101	CR	45	2D	0101101	-
14	E	0001110	SO	46	2E	0101110	.
15	F	0001111	SI	47	2F	0101111	/
16	10	0010000	DLE	48	30	0110000	0
17	11	0010001	DC1	49	31	0110001	1
18	12	0010010	DC2	50	32	0110010	2
19	13	0010011	DC3	51	33	0110011	3
20	14	0010100	DC4	52	34	0110100	4
21	15	0010101	NAK	53	35	0110101	5
22	16	0010110	SYN	54	36	0110110	6
23	17	0010111	ETB	55	37	0110111	7
24	18	0011000	CAN	56	38	0111000	8
25	19	0011001	EM	57	39	0111001	9
26	1A	0011010	SUB	58	3A	0111010	:
27	1B	0011011	ESC	59	3B	0111011	;
28	1C	0011100	FS	60	3C	0111100	<
29	1D	0011101	GS	61	3D	0111101	=
30	1E	0011110	RS	62	3E	0111110	>
31	1F	0011111	US	63	3F	0111111	?

Décimal	Hexa	Binaire	Caractère	Décimal	Hexa	Binaire	Caractère
64	40	1000000	@	96	60	1100000	‘
65	41	1000001	A	97	61	1100001	a
66	42	1000010	B	98	62	1100010	b
67	43	1000011	C	99	63	1100011	c
68	44	1000100	D	100	64	1100100	d
69	45	1000101	E	101	65	1100101	e
70	46	1000110	F	102	66	1100110	f
71	47	1000111	G	103	67	1100111	g
72	48	1001000	H	104	68	1101000	h
73	49	1001001	I	105	69	1101001	i
74	4A	1001010	J	106	6A	1101010	j
75	4B	1001011	K	107	6B	1101011	k
76	4C	1001100	L	108	6C	1101100	l
77	4D	1001101	M	109	6D	1101101	m
78	4E	1001110	N	110	6E	1101110	n
79	4F	1001111	O	111	6F	1101111	o
80	50	1010000	P	112	70	1110000	p
81	51	1010001	Q	113	71	1110001	q
82	52	1010010	R	114	72	1110010	r
83	53	1010011	S	115	73	1110011	s
84	54	1010100	T	116	74	1110100	t
85	55	1010101	U	117	75	1110101	u
86	56	1010110	V	118	76	1110110	v
87	57	1010111	W	119	77	1110111	w
88	58	1011000	X	120	78	1111000	x
89	59	1011001	Y	121	79	1111001	y
90	5A	1011010	Z	122	7A	1111010	z
91	5B	1011011	[123	7B	1111011	
92	5C	1011100	\	124	7C	1111100	
93	5D	1011101]	125	7D	1111101	
94	5E	1011110	^	126	7E	1111110	~
95	5F	1011111	_	127	7F	1111111	

Comme le montre la table des codes ASCII, les codes compris entre 0 et 31 ne représentent pas des caractères, ils ne sont pas affichables. Ces codes, souvent nommés *caractères de contrôles*, étaient utilisés pour contrôler la transmission des données du télétype (point d'entrée) vers l'unité centrale (ordinateur). Ils indiquent des actions et des directives dont la signification de leurs codes est résumée dans la table suivante.

Code	Signification	Code	Signification
NUL	Absence de caractère	DLE	Changement de signific (DataLink Escape)
SOH	Début d'entête (Start Of Heading)	DC1	Contrôle de périphérique 1 (Device Ctrl 1)
STX	Début de texte (Start of Text)	DC2	Contrôle de périphérique 2 (Device Ctrl 2)
ETX	Fin de texte (End of Text)	DC3	Contrôle de périphérique 3 (Device Ctrl 3)
EOT	Fin de transmission (End of Trans)	DC4	Contrôle de périphérique 4 (Device Ctrl 4)
ENQ	Demande de réponse (Enquiry)	NAK	Accusé de reception négatif (Neg Ack)
ACK	Accusé de reception (Acknowledge)	SYN	Attente synchronisée (Synchronous idle)
BEL	Signal sonore ou 'bip' (Bell)	ETB	Fin transmission bloc (End Trans Block)
BS	Retour arrière (Backspace)	CAN	Annuler transmission précédente (Cancel)
HT	Tabulation horizontale (Horizontal Tab)	EM	Fin de média (End of Medium)
LF	Saut de ligne (Line Feed)	SUB	Remplacement (Substitute)
VT	Tabulation verticale (Vertical Tab)	ESC	Contrôle d'extension (Escape)
FF	Saut de page (Form Feed)	FS	Séparateur de fichier (File Separator)
CR	Retour chariot (Carriage Return)	GS	Séparateur de groupe (Group Separator)
SO	Sortir des caractères réguliers (Shift Out)	RS	Séparateur d'enregistrement (Record Sep)
SI	Retour aux caractères réguliers (Shift In)	US	Séparateur d'unité (Unit Separator)

En code ASCII, les lettres se suivent dans l'ordre alphabétique (codes 65 à 90 pour les majuscules, 97 à 122 pour les minuscules), ce qui simplifie la comparaison entre les différents caractères. De plus, les codes des caractères majuscules et minuscules sont donnés de sorte que le passage du majuscule au minuscule, ou l'inverse, soit plus facile. Par exemple, pour passer du code d'un caractère majuscule à son code en minuscule, il suffit de modifier la valeur du 6^{ème} bit du code à la valeur 1, ce qui revient à ajouter la valeur 32 à son code ASCII décimal.

1.4.3.2 Code ASCII étendu

Afin de permettre l'utilisation des caractères accentués ainsi que d'autres caractères spéciaux, absents de la table ASCII basique, le huitième bit qui servait pour le contrôle de parité, contrôle rendu de plus en plus inutile face aux améliorations de la fiabilité des transmissions, a été utilisé par la suite pour fournir un jeu de caractères "étendu" pour les valeurs entre 128 et 255. Cette extension a permis l'émergence de plusieurs normes différentes. On trouve par exemple la norme ISO Latin-1 ou ISO 8859-1 pour l'alphabet de l'europe occidentale, ISO Latin-2 ou ISO 8859-2 pour l'alphabet de l'europe centrale, ISO Latin-3 ou ISO 8859-3 pour l'alphabet de l'europe du sud (turc, maltais, etc), ISO Latin-4 ou ISO 8859-4 pour l'alphabet de l'europe du nord (estonien, letton, lituanien, groenlandais et sami), ISO Latin-5 ou ISO 8859-5 pour l'alphabet cyrillique comme le bulgare, le biélorusse, le russe, le serbe et le macédonien, etc.

1.4.3.3 Encodage UNICODE

En effet, le code ASCII étendu a conduit à l'apparition de plusieurs normes différentes, ce qui a rendu difficile le passage d'un document d'une plate-forme à une autre. Pour résoudre ce problème et afin de satisfaire à tous les besoins mondiaux, les organismes de normalisation ont proposé en 1991 l'encodage UNICODE (pour UNiVersal CODE en anglais). Ce dernier code les caractères sur 16 bits, offrant donc 65 536 (2^{16}) codes différents, ce qui permet d'incorporer presque tous les alphabets existants (arabe, arménien, latin de base, chinois, japonais, cyrillique, bengali, etc.) dans une seule table de codage. Il est compatible avec le code ASCII basique (les 128 premiers caractères Unicode sont les mêmes que ceux de la table ASCII basique) ainsi que le code ASCII étendu ISO-8859-1 (les 256 premiers caractères Unicode sont les mêmes que ceux de l'encodage ISO-8859-1). Par exemple le caractère A est codé $(41)_{16}$ en ASCII et $(0041)_{16}$ en Unicode. La table suivante montre les plages (en hexadécimal) de codes de quelques alphabets en Unicode.

Alphabet	Plage des codes	Alphabet	Plage des codes
Latin	[U+0000 à U+024F]	Hébreu	[U+0590 à U+05FF]
Grec et Copte	[U+0370 à U+03FF]	Arabe	[U+0600 à U+06FF]
Cyrillique	[U+0400 à U+052F]	Maldivien	[U+0780 à U+07BF]
Arménien	[U+0530 à U+058F]	Éthiopien	[U+1200 à U+137F]

En Unicode, un caractère prend 2 octets. Par conséquent, un texte en Unicode prendra deux fois plus de place qu'en ASCII. Ce qui représente un gaspillage, notamment pour les textes anglais et latins (français, espagnol, allemand, etc.). Car la grande majorité des caractères de ces langues utilisent seulement le code ASCII. Seuls quelques rares caractères nécessitent l'Unicode.

Pour résoudre ce problème, le format UTF-8 (Universal Transformation Format 8 bis) a été introduit. Un texte en UTF-8 est écrit en ASCII (sur 8 bits), et dès qu'on a besoin d'un caractère qui n'appartient pas au code ASCII, on utilise l'Unicode en précédant le code de ce dernier par un caractère spécial signalant que le caractère suivant est en Unicode.

1.4.4 Autres codes particuliers

1.4.4.1 Code BCD

Le code BCD (Binary Coded Decimal) est un code particulier qui permet de coder un nombre décimal en binaire. Dans ce code, chaque chiffre du nombre décimal est codé, comme le montre la table suivante, en binaire naturel sur 4 bits. Ainsi, pour un nombre décimal à n chiffres, on aura une représentation BCD de $4 \times n$ chiffres binaires.

Décimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Exemples :

$$\begin{aligned}(258)_{10} &= (0010\ 0101\ 1000)_{\text{BCD}} \\ (32)_{10} &= (0011\ 0010)_{\text{BCD}} \\ (69)_{10} &= (0110\ 1001)_{\text{BCD}} \\ (4096)_{10} &= (0100\ 0000\ 1010\ 0110)_{\text{BCD}}\end{aligned}$$

1.4.4.1.1 Addition en BCD

Pour additionner deux nombres en BCD, il suffit de :

1. Procéder à l'addition en binaire naturel ;
2. Tester le résultat chiffre par chiffre, c'est à dire 4 bits par 4 bits, en commençant par les poids faibles et ajouter la valeur $(6)_{10}$, c'est à dire $(0110)_2$, à tout quartet incorrect dépassant la valeur $(9)_{10}$ (c'est à dire supérieure à $(1001)_2$) ou ayant une retenue sur l'opération de son dernier bit.

Exemple :

$$\left(\begin{array}{r} 69537 \\ + 18383 \\ \hline 87920 \end{array} \right)_{10} \Leftrightarrow \left(\begin{array}{r} \overset{1}{\curvearrowright} \\ + \begin{array}{cccccc} 0110 & 1001 & 0101 & 0011 & 0111 \\ 0001 & 1000 & 0011 & 1000 & 0011 \\ \hline 1000 & 0001 & 1000 & 1011 & 1010 \\ + & & 0110 & & 0110 & 0110 \\ \hline 1000 & 0111 & 1001 & 0010 & 0000 \end{array} \end{array} \right)_{\text{BCD}}$$

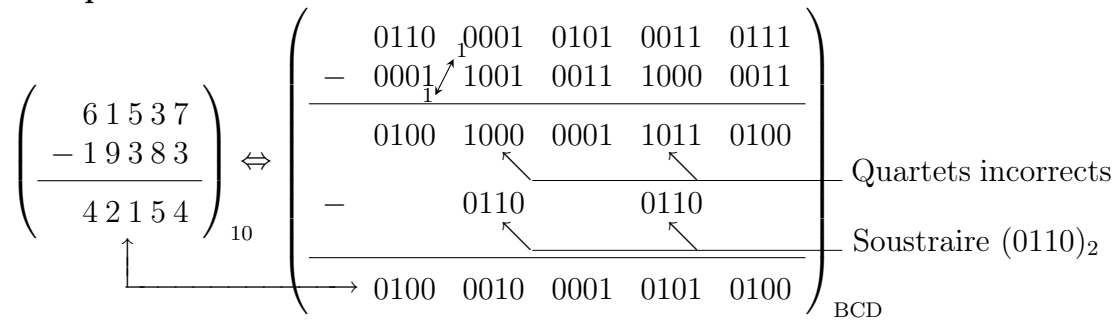
Quartets incorrects
Rajout de $(0110)_2$

1.4.4.1.2 Soustraction en BCD

Pour soustraire deux nombres en BCD, il suffit de :

1. Procéder à la soustraction en binaire naturel ;
2. Tester le résultat chiffre par chiffre, c'est à dire 4 bits par 4 bits, en commençant par les poids faibles et soustraire la valeur $(6)_{10}$, c'est à dire $(0110)_2$, de tout quartet incorrect dépassant la valeur $(9)_{10}$ (c'est à dire supérieure à $(1001)_2$) ou ayant emprunté un 1 pour son dernier bit.

Exemple :



1.4.4.2 Code Gray

Le code Gray, appelé aussi code binaire réfléchi, est un code adjacent dans lequel un seul bit change quand on passe d'un nombre entier au nombre entier suivant ou au nombre entier précédent (c'est à dire, d'un nombre n au nombre $n + 1$ ou au nombre $n - 1$). La table suivante montre le codage des chiffres décimaux en code gray.

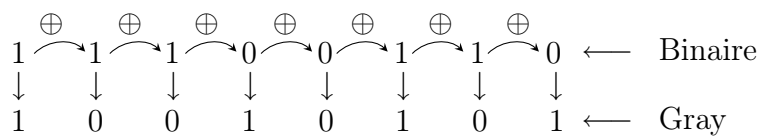
Décimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

1.4.4.2.1 Transcodage Binaire - Gray

Pour convertir un nombre $b = b_n b_{n-1} \dots b_1 b_0$ du binaire naturel au binaire réfléchi ou gray $g = g_n g_{n-1} \dots g_1 g_0$, il suffit de changer le bit qui précède directement un bit 1 ou bien de calculer les valeurs $g_i, 0 \leq i \leq n$ du code gray comme suit :

$$\begin{cases} g_n = b_n \\ g_i = b_i \oplus b_{i+1} & 0 \leq i \leq n-1 \end{cases}$$

Exemple : $(11100110)_2 = (10010101)_{\text{gray}}$

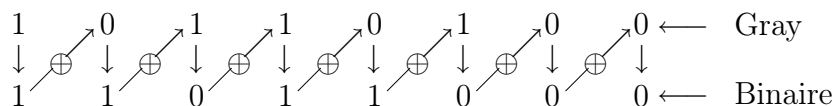


1.4.4.2.2 Transcodage Gray - Binaire

Pour convertir un nombre $g = g_n g_{n-1} \dots g_1 g_0$ du binaire réfléchi ou gray au binaire naturel $b = b_n b_{n-1} \dots b_1 b_0$, il suffit de calculer les valeurs $b_i, 0 \leq i \leq n$ du code binaire naturel comme suit :

$$\begin{cases} b_n = g_n \\ b_i = b_{i+1} \oplus g_i & 0 \leq i \leq n-1 \end{cases}$$

Exemple : $(10110100)_{\text{gray}} = (11011000)_2$



1.4.4.3 Codes détecteurs d'erreur

Les données traitées par un ordinateur peuvent subir des modifications involontaires lors de leur transmission ou lors de leur stockage en mémoire. Donc, il est parfois important d'utiliser des codes permettant de détecter les erreurs dues à ces modifications. Dans cette optique, plusieurs variantes du code BCD ont été introduites. Parmi celles-ci, on trouve le code *2 dans 5*, le code *Johnson* et le code *biquinaire*. Dans le code *2 dans 5*, un chiffre décimal est codé sur 5 bits, avec 2 et seulement 2 bits ayant la valeur 1. Le code *Johnson* est un code BCD adjacent à 5 bits dont les 1 constituant le code sont toujours placés en un seul groupe qui se déplace d'un chiffre décimal au chiffre suivant. De son côté, le code *biquinaire* est une variante du code BCD où les chiffres décimaux sont codés sur 7 bits, avec toujours un et un seul bit à 1 dans les deux bits de gauche et un et un seul bit 1 dans les cinq bits de droite. La table suivante montre le codage des différents chiffres décimaux pour chacun de ces trois codes.

Décimal	0	1	2	3	4	5	6	7	8	9
2 dans 5	00011	00101	00110	01001	01010	01100	10001	10010	10100	11000
Johnson	00000	00001	00011	00111	01111	11111	11110	11100	11000	10000
biquinaire	0100001	0100010	0100100	0101000	0110000	1000001	1000010	1000100	1001000	1010000

Exemples :

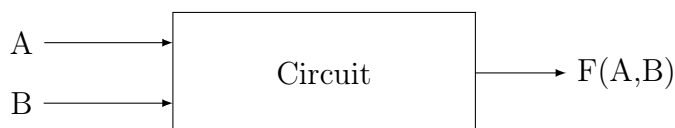
$$\begin{aligned}
 (2058)_{10} &= (00110 \ 00011 \ 01100 \ 10100)_{2 \text{ dans } 5} \\
 &= (00011 \ 00000 \ 11111 \ 11000)_{\text{Johnson}} \\
 &= (0100100 \ 0100001 \ 1000001 \ 1001000)_{\text{biquinaire}}
 \end{aligned}$$

Chapitre 2

Algèbre de Boole

2.1 Introduction

Les machines numériques sont constituées d'un ensemble de circuits électroniques. Chaque circuit fournit une fonction bien déterminée (addition, comparaison, etc.).



Pour concevoir et réaliser un circuit numérique, on doit avoir un modèle mathématique de la fonction réalisée par ce circuit. La réalisation de ce modèle est basé sur l'algèbre de Boole. Ainsi dans ce chapitre, et avant d'aborder la notion des circuits combinatoires et des circuits séquentiels qui seront détaillées dans les chapitres ultérieurs, nous devons tout d'abord se familiariser avec la notion de l'algèbre de boole.

2.2 Définition de l'algèbre de Boole

L'algèbre de Boole est une algèbre définie sur un ensemble comprenant deux éléments (vrai ou faux) que nous noterons 0 et 1 ($\{0,1\}$). Les variables de cette algèbre sont appelées variables logiques et elles peuvent prendre l'une de ces deux valeurs. Il s'agit donc d'un outil mathématique qui permet d'établir la relation entre un ensemble de variables logiques via des fonctions appelées fonctions logiques. La sortie de celles-ci ne peut, elle aussi, prendre que deux valeurs (0 et 1).

Les opérateurs de base utilisés par une fonction logique pour relier ses variables d'entrée sont :

- La somme booléenne, appelé aussi le OU logique (notation $+$)
- Le produit booléen, appelé aussi le ET logique (notation \cdot)
- La négation (complément) booléenne, appelé aussi le NON logique (notation $-$)

Voici un exemple d'une fonction logique :

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

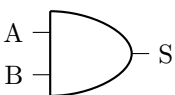
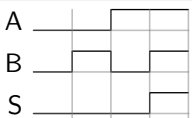
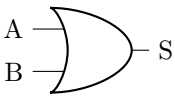
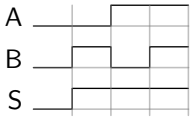
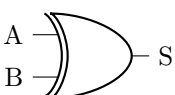
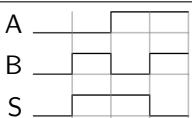
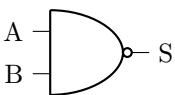
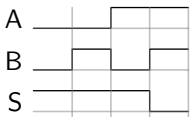
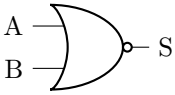
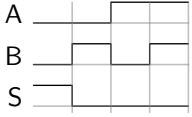
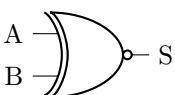
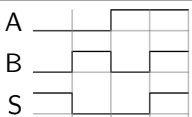
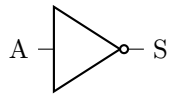
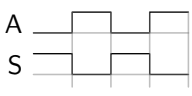
En électronique, l'algèbre de Boole est définie comme étant l'outil mathématique qui permet d'établir la relation entre les entrées et les sorties du système à réaliser (synthèse du système). Réciproquement, cet outil est utilisé également pour déterminer les règles de fonctionnement d'un système logique existant (analyse du système).

2.3 Théorèmes fondamentaux de l'algèbre de Boole

Une algèbre de Boole doit vérifier les axiomes et les théorèmes suivants :

axiome ou théorème	Opérateur OU (+)	Opérateur ET (·)
Commutativité	$A+B = B+A$	$A \cdot B = B \cdot A$
Distributivité	$A+(B \cdot C) = (A+B) \cdot (A+C)$	$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$
Associativité	$(A+B)+C = A+(B+C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
Idempotence	$A+A = A$	$A \cdot A = A$
Complémentation	$A+\bar{A} = 1$	$A \cdot \bar{A} = 0$
Involution	$\bar{\bar{A}} = A$	
Elément neutre	$A+0 = A$	$A \cdot 1 = A$
Elément dominant	$A+1 = 1$	$A \cdot 0 = 0$
Absorption	$A+(A \cdot B) = A$	$A \cdot (A+B) = A$
Allègement	$A+(\bar{A} \cdot B) = A+B$	$A \cdot (\bar{A}+B) = A \cdot B$
Inclusion	$(A+B) \cdot (A+\bar{B}) = A$	$(A \cdot B) + (A \cdot \bar{B}) = A$
De Morgan	$\overline{A+B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$

2.4 Fonctions de base de l'algèbre de Boole

Fonction	Symbole	Equation	Table de vérité	Chronogramme															
ET AND		$S = A \cdot B$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	S	0	0	0	0	1	0	1	0	0	1	1	1	
A	B	S																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OU OR		$S = A + B$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	S	0	0	0	0	1	1	1	0	1	1	1	1	
A	B	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
OU Exclusif XOR		$S = A \oplus B$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S	0	0	0	0	1	1	1	0	1	1	1	0	
A	B	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
NON-ET NAND		$S = \overline{A \cdot B}$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S	0	0	1	0	1	1	1	0	1	1	1	0	
A	B	S																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NON-OU NOR		$S = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	S	0	0	1	0	1	0	1	0	0	1	1	0	
A	B	S																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
NON-OU Exclusif XNOR		$S = \overline{A \oplus B}$	<table><tr><th>A</th><th>B</th><th>S</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	S	0	0	1	0	1	0	1	0	0	1	1	1	
A	B	S																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	
NON NOT		$S = \overline{A}$	<table><tr><th>A</th><th>S</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	S	0	1	1	0										
A	S																		
0	1																		
1	0																		

2.5 Représentation d'une fonction logique

En plus de sa représentation algébrique, une fonction logique peut être aussi représentée par une table de vérité, par un logigramme et par chronogramme.

2.5.1 Table de vérité d'une fonction logique

Une table de vérité est un tableau qui définit la valeur de sortie de la fonction logique en fonction de toutes les combinaisons de valeurs (0 ou 1) que peuvent prendre les variables d'entrées.

La taille de la table de vérité dépend du nombre de variables d'entrée. Par exemple, pour une fonction de n variables, il y aura 2^n combinaisons possibles, il y aura donc 2^n lignes dans la table de vérité.

Pour trouver la table de vérité d'une fonction logique, il faut trouver la valeur de sa sortie pour chaque combinaison de ses variables d'entrée. Lors de l'évaluation des différentes combinaisons de la fonction, on doit respecter l'ordre de priorité entre les différents opérateurs logiques, qui est :

- | | |
|--------------------------|------------------------------|
| (1) Les parenthèses, | (3) Le produit logique (ET), |
| (2) Le complément (NON), | (4) La somme logique (OU). |

Exemple : La table de vérité de la fonction logique : $F(A, B, C) = A \cdot B + \bar{B} \cdot C + A \cdot \bar{C}$

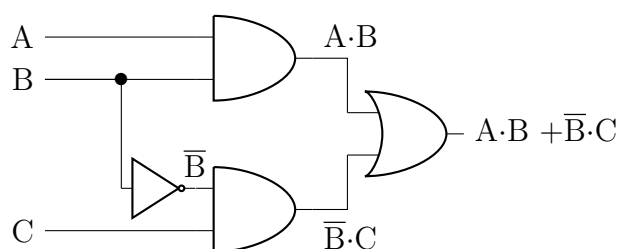
A	B	C	\bar{B}	\bar{C}	$A \cdot B$	$\bar{B} \cdot C$	$A \cdot \bar{C}$	$F(A, B, C)$
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	1	0	1
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	1	0	1
1	1	0	0	1	1	0	1	1
1	1	1	0	0	1	0	0	1

Remarque : On dira que deux fonctions sont équivalentes si et seulement si elles possèdent la même table de vérité.

2.5.2 Logigramme d'une fonction logique

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

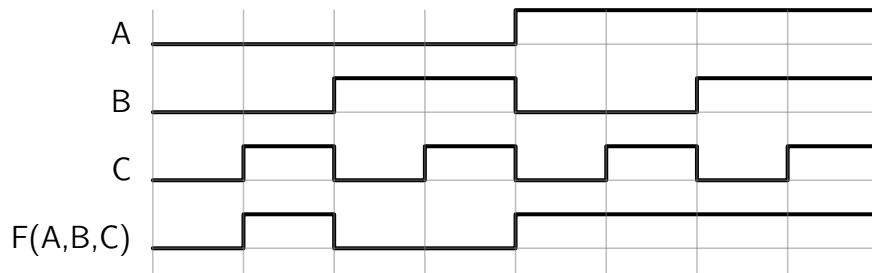
Exemple : Le logigramme de la fonction : $F(A, B, C) = A \cdot B + \bar{B} \cdot C$ est :



2.5.3 Chronogramme d'une fonction logique

Un chronogramme est une représentation graphique qui permet de visualiser, en fonction du temps, l'état de la sortie correspondant aux différentes combinaisons d'états logiques des entrées. Les variables binaires sont représentées par une tension haute (+5V par exemple) lorsqu'elles sont à un niveau logique haut "1" et par une tension nulle (0V) lorsqu'elles sont à un niveau logique bas "0".

Exemple : Le chronogramme de la fonction : $F(A, B, C) = A \cdot B + \overline{B} \cdot C + A \cdot \overline{C}$ est :



2.6 Forme canonique d'une fonction logique

La forme canonique d'une fonction logique est la forme où la fonction est écrite sous forme d'une somme de mintermes ou sous forme d'un produit de maxtermes.

2.6.1 Définitions

1. **Définition des mintermes :** Un minterme de N variables est un produit de ces N variables ou de leurs complémentaires.

Exemple : Pour les quatre variables A, B, C et D :

- $A \cdot B \cdot C \cdot D, A \cdot \overline{B} \cdot C \cdot D, \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$: sont des mintermes.
- $A \cdot B \cdot C, A \cdot \overline{B} \cdot \overline{C} \cdot D, \overline{A} + \overline{B} + C + \overline{D}$: ne sont pas des mintermes.

2. **Définition des maxtermes :** Un maxterme de N variables est une somme de ces N variables ou de leurs complémentaires.

Exemple : Pour les quatre variables A, B, C et D :

- $A + B + C + D, A + \overline{B} + C + D, \overline{A} + \overline{B} + C + \overline{D}$: sont des maxtermes.
- $A + B + C, A + \overline{B} + \overline{C} + D, \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$: ne sont pas des maxtermes.

3. **Définition de la première forme canonique :** La première forme canonique d'une fonction logique est la forme où la fonction est écrite sous forme d'une somme de mintermes.

Exemple : $F(A, B, C) = A \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$

4. **Définition de la deuxième forme canonique :** La deuxième forme canonique d'une fonction logique est la forme où la fonction est écrite sous forme d'un produit de maxtermes.

Exemple : $F(A, B, C) = (A + B + C) \cdot (A + \overline{B} + C) \cdot (A + B + \overline{C})$

2.6.2 Mettre une fonction en première forme canonique

Pour mettre une fonction en première forme canonique, on peut utiliser deux méthodes, la méthode algébrique et la méthode de la table de vérité.

2.6.2.1 La méthode algébrique

Dans cette méthode, on applique les règles de l'algèbre de Boole, la Complémentation et la distributivité, en passant par les trois étapes suivantes :

Etape 1. Ecrire la fonction sous forme de somme de produits (distribution).

Etape 2. Multiplier les termes par les compléments des variables manquantes.

Etape 3. Distribuer les termes sur les compléments insérés.

Exemple :

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

2.6.2.2 La méthode de la table de vérité

Dans cette méthode, on extrait la première forme canonique directement à partir de la table de vérité, en passant par les trois étapes suivantes :

Etape 1. Extraire toutes les combinaisons qui donnent la valeur "1" en résultat.

Etape 2. Ecrire chacune de ces combinaisons sous forme d'un minterme.

Etape 3. Faire la somme des mintermes.

Exemple :

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

→ 0 0 1 ⇒ $\overline{A} \cdot \overline{B} \cdot C$ (minterme 1)

→ 0 1 1 ⇒ $\overline{A} \cdot B \cdot C$ (minterme 2)

→ 1 0 1 ⇒ $A \cdot \overline{B} \cdot C$ (minterme 3)

→ 1 1 0 ⇒ $A \cdot B \cdot \overline{C}$ (minterme 4)

→ 1 1 1 ⇒ $A \cdot B \cdot C$ (minterme 5)

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

2.6.3 Mettre une fonction en deuxième forme canonique

Tout comme pour la première forme canonique, on peut utiliser deux méthodes différentes pour mettre une fonction logique en deuxième forme canonique.

2.6.3.1 La méthode algébrique

Dans cette méthode, on applique les règles de l'algèbre de Boole, la Complémentation et la distributivité, en passant par les trois étapes suivantes :

- Etape 1.** Ecrire la fonction sous forme de produit de sommes (distribution).
- Etape 2.** Additionner les termes avec les compléments des variables manquantes.
- Etape 3.** Distribuer les termes sur les compléments insérés.

Exemple :

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

2.6.3.2 La méthode de la table de vérité

Dans cette méthode, on extrait la deuxième forme canonique directement à partir de la table de vérité, en passant par les trois étapes suivantes :

- Etape 1.** Extraire toutes les combinaisons qui donnent la valeur "0" en résultat.
- Etape 2.** Ecrire chacune de ces combinaisons sous forme d'un maxterme.
- Etape 3.** Faire le produit des maxtermes.

Exemple :

A	B	C	F(A,B,C)	
0	0	0	0	→ 0 0 0 ⇒ $A + B + C$ (maxterme 1)
0	0	1	1	
0	1	0	0	→ 0 1 0 ⇒ $A + \overline{B} + C$ (maxterme 2)
0	1	1	1	
1	0	0	0	→ 1 0 0 ⇒ $\overline{A} + B + C$ (maxterme 3)
1	0	1	1	
1	1	0	1	
1	1	1	1	

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

2.6.4 Représentation décimale des deux formes canoniques

En plus de leur représentation algébrique, les deux formes canoniques peuvent également être représentées en décimal comme suit :

1. Remplacer chaque terme par la valeur décimale correspondante.
2. Mettre les valeurs obtenues entre parenthèses.
3. Précéder les parenthèses par le symbole de somme (\sum) dans le cas de la première forme canonique, et par le symbole de produit (\prod) dans le cas de la deuxième forme canonique.

Exemple :

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

$$\Rightarrow F(A, B, C) = \sum(1, 3, 5, 6, 7)$$

$$F(A, B, C) = (A + B + C).(A + \overline{B} + C).(\overline{A} + B + C) \Rightarrow F(A, B, C) = \prod(0, 2, 4)$$

2.6.5 Expression d'une fonction logique avec des portes NAND et NOR exclusivement

Bien que les trois portes logiques de base (ET, OU et NON) sont les portes les plus utilisées dans la réalisation des circuits logiques, les portes NAND et NOR ont aussi leur importance dans la réalisation des circuits logiques, car elles permettent de réaliser n'importe quelle fonction logique combinatoire. Ceci est très intéressant puisque cela permet de réduire le nombre de transistors qui interviennent dans le circuit à réaliser et d'optimiser le nombre de boîtiers de circuits intégrés sur la plaque de circuit imprimé (voir le dernier chapitre).

2.6.5.1 Expression d'une fonction logique avec des portes NAND

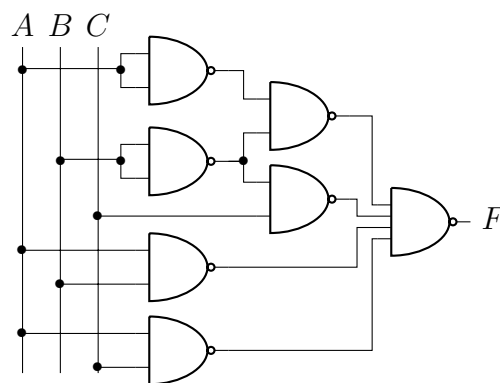
Pour réaliser une fonction logique à l'aide uniquement de portes NAND, il suffit de suivre les étapes suivantes :

Etape 1. Ecrire la fonction logique sous forme de somme de produits.

Etape 2. Appliquer deux négations consécutives sur la fonction, dont la première sera distribuée sur les termes de la fonction, alors que la deuxième sera restée au dessus de tous les termes.

Exemple : Réaliser la fonction suivante avec uniquement des portes NAND.

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



2.6.5.2 Expression d'une fonction logique avec des portes NOR

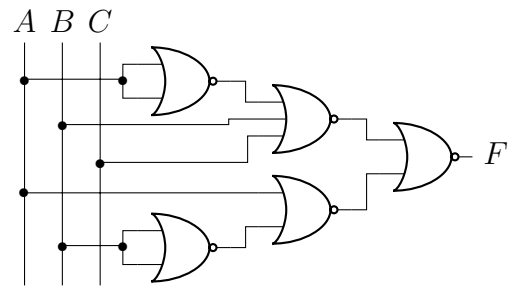
Tout comme pour les portes NAND, pour réaliser une fonction logique à l'aide uniquement de portes NOR, il suffit de suivre les étapes suivantes :

Etape 1. Ecrire la fonction logique sous forme de produit de sommes.

Etape 2. Appliquer deux négations consécutives sur la fonction, dont la première sera distribuée sur les termes de la fonction, alors que la deuxième sera restée au dessus de tous les termes.

Exemple : Réaliser la fonction suivante avec uniquement des portes NOR.

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



2.7 Simplification des fonctions logiques

En pratique, les fonctions logiques sont réalisées avec des circuits électroniques dont le nombre de composants (transistors) qui les constituent dépend essentiellement du nombre de termes et du nombre de variables constituant les termes de la fonction à réaliser. Ainsi, pour minimiser le coût de fabrication de ces circuits, il faut que la fonction logique soit écrite avec un minimum de termes qui eux-mêmes doivent contenir le minimum de variables. En plus de minimisation du coût, une fonction simplifiée permet aussi d'accélérer le temps d'exécution et de diminuer le taux d'échauffement du circuit.

Il est donc nécessaire de disposer d'outils et de techniques permettant de déterminer la forme minimale d'une fonction logique. On parlera alors de simplification ou de minimisation de fonctions logiques. Parmi les méthodes les plus utilisées pour simplifier une fonction logique, on trouve la méthode algébrique, la méthode de karnaugh et la méthode de Quine Mc Cluskey.

2.7.1 Méthode algébrique

Le principe de cette méthode est d'appliquer la règle de la mise en facteur commun, ainsi que les règles de l'algèbre de Boole afin d'éliminer des variables ou des termes. Malheureusement, il n'y a pas une démarche bien spécifique. Elle consiste essentiellement à utiliser intuitivement des règles de l'algèbre de Boole qui peuvent conduire à la simplification de la fonction, telles que :

- Mettre en facteur des variables afin d'éliminer un ou plusieurs termes.

Exemple :

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

- Ajouter un terme déjà existant à la fonction afin de pouvoir le mettre en facteur avec plusieurs termes.

Exemple :

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

- Augmenter le nombre de variables de certains termes pour pouvoir les éliminer.

Exemple :

$$\begin{aligned}
 F(A, B, C) &= A.B + \overline{B}.C + A.C \\
 &= A.B + \overline{B}.C + A.C.(B + \overline{B}) \\
 &= A.B + \overline{B}.C + A.B.C + A.\overline{B}.C \\
 &= A.B.(1 + C) + \overline{B}.C.(1 + A) \\
 &= A.B + \overline{B}.C
 \end{aligned}$$

2.7.2 Méthode de karnaugh

La méthode de karnaugh est une méthode graphique qui permet de simplifier une fonction logique à partir d'une table appelée table de karnaugh.

2.7.2.1 Table de karnaugh

La table de Karnaugh est une variante des tables de vérité. Elle est organisée de telle façon que les termes adjacents (qui ne diffèrent que par une variable) soient systématiquement regroupés dans des cases voisines.

2.7.2.1.1 Caractéristiques de la table de karnaugh

Une table de karnaugh est généralement caractérisée par les caractéristiques suivantes :

- C'est une table à deux entrées (lignes-colonnes).
- Les variables sont équilibrées sur les lignes et les colonnes pour s'approcher d'un tableau carré.
- Les bords de la table sont adjacents.
- Les lignes et les colonnes sont codées en code GRAY (00, 01, 11, 10).
- Chaque case contient l'état de la sortie (0 ou 1) pour les entrées correspondantes.

2.7.2.1.2 Table de Karnaugh à partir de la table de vérité

Pour remplir une table de karnaugh à partir d'une table de vérité, on procède comme suite :

- Pour chaque combinaison qui donne un 1 en résultat dans la table de vérité lui correspond une case à 1 dans la table de karnaugh.
- Les cases qui restent sont mises à zéro.

Exemple :

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

⇒

		AB			
		00	01	11	10
C	0	0 ⁰ 0	2 ⁰ 0	6 ¹ 1	4 ⁰ 0
	1	1 ¹ 1	3 ¹ 1	7 ¹ 1	5 ¹ 1

2.7.2.1.3 Table de Karnaugh à partir de la forme canonique

Pour remplir une table de karnaugh à partir de la forme canonique de sa fonction, on procède comme suite :

- Si la fonction logique est donnée sous la première forme canonique alors : pour chaque terme, on lui correspond une case à 1.
- Si la fonction logique est donnée sous la deuxième forme canonique alors : pour chaque terme, on lui correspond une case à 0.

Exemple :

$$F(A, B, C) = \underbrace{\overline{A}.\overline{B}.\overline{C}}_0 + \underbrace{\overline{A}.B.\overline{C}}_2 + \underbrace{\overline{A}.B.C}_3 \quad \left| \quad F(A, B, C) = \underbrace{(\overline{A} + \overline{B} + \overline{C})}_7 \underbrace{(\overline{A} + B + \overline{C})}_5 \underbrace{(\overline{A} + B + C)}_4$$

	AB	00	01	11	10
C		0	1	2	3
0		1	1	0	0
1		0	1	0	0

	AB	00	01	11	10
C		0	1	6	4
0		1	1	1	0
1		1	1	0	0

2.7.2.2 Méthode de simplification

Pour simplifier une fonction par la méthode de karnaugh, nous devons suivre les étapes suivantes :

- Remplir le tableau à partir de la table de vérité ou à partir de la forme canonique,
- Faire des regroupements de 64, 32, 16, 8, 4, 2, 1 cases (les mêmes termes peuvent participer à plusieurs regroupements, c'est à dire les intersections sont autorisées),
- Eliminer les regroupements inutiles, c'est à dire ceux qui sont inclus dans d'autres regroupements,
- Pour chaque regroupement, on élimine les variables qui changent d'état,
- L'expression simplifiée est la somme des termes correspondant aux différents regroupements après simplification et élimination des variables qui changent d'état.

Exemple : Simplifier la fonction $F(A, B, C) = A.B.C + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C}$

	AB	00	01	11	10
C		0	1	2	3
0		0	0	1	0
1		0	1	1	1

→ AB

→ AC

→ BC

La fonction simplifiée est :

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

2.7.2.3 Tables de karnaugh avec des valeurs inconnues

Lorsque la fonction à réaliser n'est pas complètement définie, c'est-à-dire pour certaines combinaisons des variables, la valeur logique que prend la fonction n'est pas connue. Les cases correspondant à ces combinaisons sont appelées des cases inconnues,

on les note par x . Lors de la simplification, on attribuera à chaque case x la valeur (0 ou 1) qui permet d'obtenir le plus grand nombre de 1 regroupables ensemble.

Exemple :

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

L'expression simplifiée de :

$$F(A, B, C, D) = \sum(1, 3, 5, 12) + X(4, 7, 11, 13, 14, 15)$$

est :

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

2.7.2.4 Tables de karnaugh à cinq variables

Pour cinq variables A, B, C, D et E, la table de karnaugh va contenir 32 cases (2^5 combinaisons), pour garder l'adjacence des différentes combinaisons, on utilise une représentation spatiale (deux faces opposées d'un cube).

		$B.C$				$D.E$			
		00	01	11	10	00	01	11	10
$B.C$	00	0	4	12	8	20	28	24	
	01	1	5	13	9	21	29	25	
	11	3	7	15	11	23	31	27	
	10	2	6	14	10	22	30	26	

$A = 1$
 $A = 0$

Cette représentation spatiale peut être remplacée par deux tables de 4 variables. La cinquième variable (variable de poids fort) change de valeur lorsque l'on passe d'une table à l'autre. Les cases qui occupent la même position dans les deux tables sont adjacentes (0 est adjacente avec 16, 5 est adjacente avec 21 et ainsi de suite).

		BC			
		00	01	11	10
DE	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

$A = 0$

		BC			
		00	01	11	10
DE	00	16	20	28	24
	01	17	21	29	25
	11	19	23	31	27
	10	18	22	30	26

$A = 1$

Pour simplifier l'expression d'une table de karnaugh à 5 variables, on passe par les mêmes étapes que celles des tables à 4 variables, sauf qu'ici il faut respecter l'adjacence entre les deux tables.

Exemple : Simplifier la fonction logique suivante :

$$F(A, B, C, D, E) = \sum(0, 1, 2, 4, 5, 6, 10, 13, 14, 18, 21, 22, 24, 26, 29, 30)$$

		BC							BC						
		00	01	11	10				00	01	11	10			
DE	00	1	1	0	0	$\overline{A}.\overline{B}.\overline{D}$ $C.\overline{D}.E$ $D.\overline{E}$ $A.B.\overline{C}.\overline{E}$			DE	00	0	0	0	1	
	01	1	1	1	0					01	0	1	1	0	
	11	0	0	0	0					11	0	0	0	0	
	10	1	1	1	1					10	1	1	1	1	
$A = 0$										$A = 1$					

La fonction simplifiée est : $F(A, B, C, D, E) = D.\overline{E} + C.\overline{D}.E + \overline{A}.\overline{B}.\overline{D} + A.B.\overline{C}.\overline{E}$

2.7.2.5 Tables de karnaugh à six variables

De la même façon, la table de karnaugh à six variables A, B, C, D, E et F (2^6 combinaisons = 64 cases), est représentée par quatre tables de 4 variables. La cinquième et la sixième variable (les deux variables de poids fort) changent de valeur lorsque l'on passe d'une table à l'autre. Les cases qui occupent la même position dans les tables adjacentes sont adjacentes (la case 0 est adjacente avec les cases 16 et 32, la case 31 est adjacente avec les cases 15 et 63, et ainsi de suite).

		CD					CD					
		00	01	11	10		00	01	11	10		
EF	00	0	4	12	8	$AB = 00$	EF	00	16	20	28	24
	01	1	5	13	9			01	17	21	29	25
	11	3	7	15	11			11	19	23	31	27
	10	2	6	14	10			10	18	22	30	26
						$AB = 01$						
		CD					CD					
		00	01	11	10		00	01	11	10		
EF	00	32	36	44	40	$AB = 10$	EF	00	48	52	60	56
	01	33	37	45	41			01	49	53	61	57
	11	35	39	47	43			11	51	55	63	59
	10	34	38	46	42			10	50	54	62	58
						$AB = 11$						

Pour simplifier, on suit le même principe que celui avec 5 variables.

Exemple : Simplifier la fonction logique suivante :

$$F(A, B, C, D, E, F) = \sum(4, 5, 6, 7, 8, 10, 13, 15, 18, 20, 21, 22, 23, 26, 29, 30, 31, 33, 36, 37, 38, 39, 40, 42, 49, 52, 53, 54, 55, 60, 61)$$

		CD			
		00	01	11	10
EF	00	0	1	0	1
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	0	1

$AB = 00$

		CD			
		00	01	11	10
EF	00	0	1	0	0
	01	0	1	1	0
	11	0	1	1	0
	10	1	1	1	1

$AB = 01$

		CD			
		00	01	11	10
EF	00	0	1	0	1
	01	1	1	0	0
	11	0	1	0	0
	10	0	1	0	1

$AB = 10$

		CD			
		00	01	11	10
EF	00	0	1	1	0
	01	1	1	1	0
	11	0	1	0	0
	10	0	1	0	0

$AB = 11$

La fonction simplifiée est : $F(A, B, C, D, E, F) = \overline{C}.D + \overline{A}.D.F + \overline{A}.B.E.\overline{F} + \overline{B}.C.\overline{D}.\overline{F} + A.\overline{C}.\overline{E}.F + A.B.D.\overline{E}$

Remarque : Au-delà de 6 variables aucune représentation de karnaugh n'est possible, il faudra donc faire appel à d'autres méthodes, comme celle de *Quine Mc Cluskey*.

2.7.3 Méthode de Quine Mc Cluskey

La méthode de Quine Mc Cluskey est une méthode de simplification des fonctions logiques qui est fonctionnellement identique à la méthode de Karnaugh, mais sous une forme plus adaptée à la programmation informatique.

Pour simplifier via cette méthode, on doit suivre les étapes suivantes :

- Etape 1.** Exprimer la fonction sous la première forme canonique.
- Etape 2.** Exprimer les mintermes sous forme binaire.
- Etape 3.** Grouper les termes selon leurs poids (nombre de bits à 1).
- Etape 4.** Unir les termes des groupes adjacents deux à deux (répéter cette étape autant de fois que nécessaire).
- Etape 5.** Identifier les impliquants premiers.
- Etape 6.** Identifier les impliquants premiers essentiels.

Etape 7. Vérifier si la fonction est entièrement exprimée par ses impliquants essentiels, auquel cas arrêter.

Etape 8. Si on n'a pas fini à l'étape (8), choisir les impliquants premiers appropriés.

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

Etape 1. Exprimer la fonction sous la première forme canonique.

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

Etape 2. Exprimer les mintermes sous forme binaire.

$$F(A, B, C) = 101 + 100 + 011 + 010 + 001 + 111$$

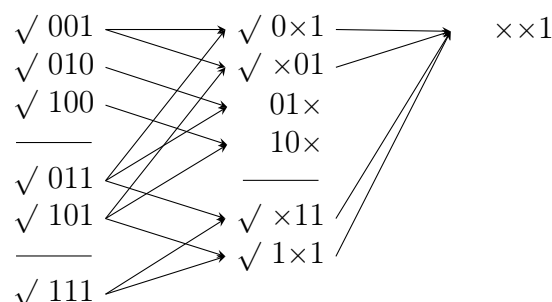
Etape 3. Grouper les termes selon leurs poids (nombre de bits à 1).

Poids 1 : 001, 010, 100

Poids 2 : 011, 101

Poids 3 : 111

Etape 4. Unir les termes des groupes adjacents deux à deux (répéter cette étape autant de fois que nécessaire).



Etape 5. Identifier les impliquants premiers (ceux qui n'ont aucune flèche sortante). 01x, 10x, $\times \times 1$

Etape 6. Identifier les impliquants premiers essentiels.

	001	010	100	011	101	111
01x		\checkmark		\checkmark		
10x			\checkmark		\checkmark	
$\times \times 1$	\checkmark			\checkmark	\checkmark	\checkmark

On cherche les colonnes où se retrouve un seul signe. Les lignes correspondantes sont des impliquants premiers essentiels. Ici, tous les impliquants premiers sont essentiels.

Etape 7. Vérifier si la fonction est entièrement exprimée par ses impliquants essentiels, auquel cas arrêter. Ici, tous les impliquants de la fonction $F(A, B, C)$ sont essentiels, donc la condition est forcément remplie.

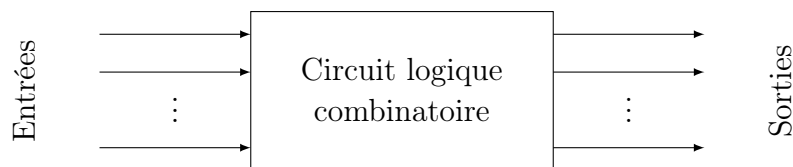
Etape 8. La fonction simplifiée est la somme des impliquants premiers essentiels : $F(A, B, C) = \bar{A}.B + A.\bar{B} + C$

Chapitre 3

Circuits logiques combinatoires

3.1 Introduction

Un circuit logique combinatoire est un circuit numérique qui possède un certain nombre d'entrées et un certain nombre de sorties, où les sorties dépendent uniquement des entrées. Les sorties sont reliées aux entrées par des fonctions logiques. A une combinaison d'entrées (l'entrée) ne correspond qu'une seule combinaison de sorties (la sortie).



Ces circuits sont établis à partir d'une opération appelée synthèse combinatoire.

3.2 Synthèse des systèmes combinatoires

La synthèse combinatoire est la traduction d'une fonction logique, à partir d'un cahier des charges, en un schéma logique (logigramme). L'objectif est d'obtenir les équations booléennes des sorties en fonction des entrées.

La synthèse des circuits logiques combinatoires passe généralement par les étapes suivantes :

Etape 1. Identifier les entrées et les sorties du circuit logique.

Etape 2. Construire la table de vérité des sorties en fonction des entrées.

E_1	E_2	\dots	E_N	S_1	S_2	\dots	S_M
.	.	\dots	.	.	.	\dots	.
\vdots	\vdots	\dots	\vdots	\vdots	\vdots	\dots	\vdots
.	.	\dots	.	.	.	\dots	.

Etape 3. Identifier les fonctions logiques des différentes sorties à partir de la table de vérité.

Etape 4. Simplifier les fonctions logiques obtenues en utilisant une des trois méthodes de simplification (algébrique, karnaugh, Quine McCluskey).

Etape 5. Dessiner le schéma du circuit (logigramme).

3.3 Synthèse des principaux circuits combinatoires

3.3.1 L'additionneur

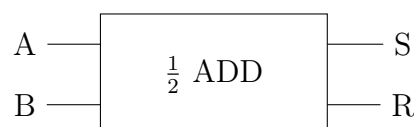
3.3.1.1 Le semi-additionneur

Définition : Un semi-additionneur est un circuit permettant d'additionner 2 bits d'entrée, et d'obtenir comme sortie le résultat de l'addition et la retenue.

Synthèse du circuit :

Etape 1. Identification des entrées et des sorties.

D'après la définition précédente, le semi-additionneur a deux entrées et deux sorties.



Etape 2. La table de vérité du circuit.

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

Etape 3. Identification des fonctions logiques des deux sorties S et R.

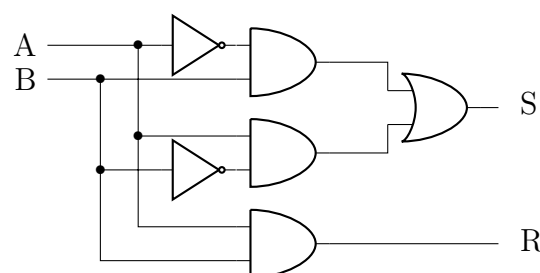
$$S(A, B) = \overline{A}.B + A.\overline{B}$$

$$R(A, B) = A.B$$

Etape 4. Simplification des fonctions obtenues.

Les fonctions obtenues ne peuvent pas être simplifiées encore plus.

Etape 5. Le schéma du circuit (logigramme).



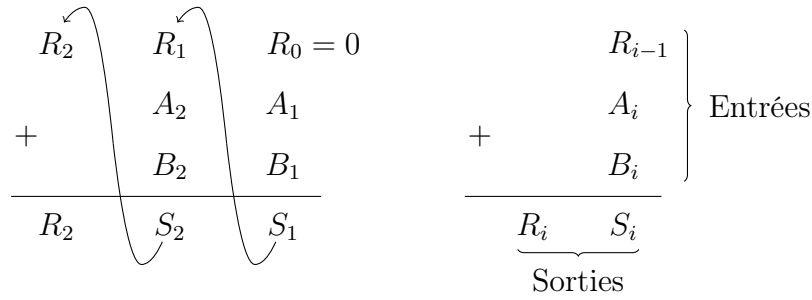
3.3.1.2 L'additionneur complet

Définition : L'additionneur complet est un additionneur à deux bits qui tient compte non seulement des deux bits à additionner, mais aussi de la retenue obtenue lors de l'addition des deux bits de la position précédente.

Synthèse du circuit :

Etape 1. Identification des entrées et des sorties.

En binaire lorsque on fait une addition il faut tenir en compte de la retenue entrante.



L'additionneur complet possède donc 3 entrées et deux sorties.



Etape 2. La table de vérité du circuit.

A_i	B_i	R_{i-1}	S_i	R_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Etape 3. Identification des fonctions logiques des deux sorties S et R.

$$S_i = \overline{A_i} \cdot \overline{B_i} \cdot R_{i-1} + \overline{A_i} \cdot B_i \cdot \overline{R_{i-1}} + A_i \cdot \overline{B_i} \cdot \overline{R_{i-1}} + A_i \cdot B_i \cdot R_{i-1}$$

$$R_i = \overline{A_i} \cdot B_i \cdot R_{i-1} + A_i \cdot \overline{B_i} \cdot R_{i-1} + A_i \cdot B_i \cdot \overline{R_{i-1}} + A_i \cdot B_i \cdot R_{i-1}$$

Etape 4. Simplification des fonctions obtenues.

$$S_i = \overline{A_i} \cdot \overline{B_i} \cdot R_{i-1} + \overline{A_i} \cdot B_i \cdot \overline{R_{i-1}} + A_i \cdot \overline{B_i} \cdot \overline{R_{i-1}} + A_i \cdot B_i \cdot R_{i-1}$$

$$= \overline{A_i} \cdot (\overline{B_i} \cdot R_{i-1} + B_i \cdot \overline{R_{i-1}}) + A_i \cdot (\overline{B_i} \cdot \overline{R_{i-1}} + B_i \cdot R_{i-1})$$

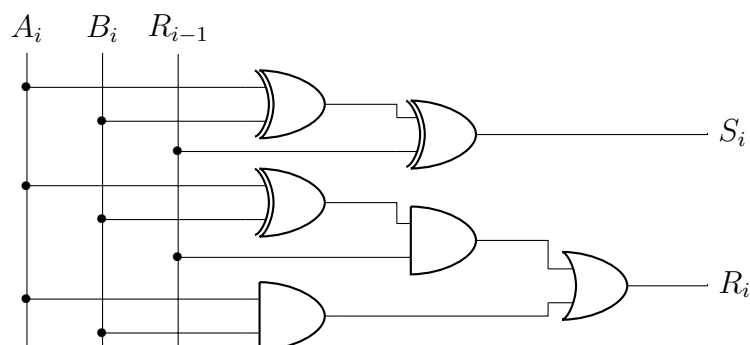
$$= \overline{A_i} \cdot (B_i \oplus R_{i-1}) + A_i \cdot (\overline{B_i} \oplus \overline{R_{i-1}}) = A_i \oplus B_i \oplus R_{i-1}$$

$$R_i = \overline{A_i} \cdot B_i \cdot R_{i-1} + A_i \cdot \overline{B_i} \cdot R_{i-1} + A_i \cdot B_i \cdot \overline{R_{i-1}} + A_i \cdot B_i \cdot R_{i-1}$$

$$= R_{i-1} \cdot (\overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}) + A_i \cdot B_i \cdot (\overline{R_{i-1}} + R_{i-1})$$

$$= R_{i-1} \cdot (A_i \oplus B_i) + A_i \cdot B_i$$

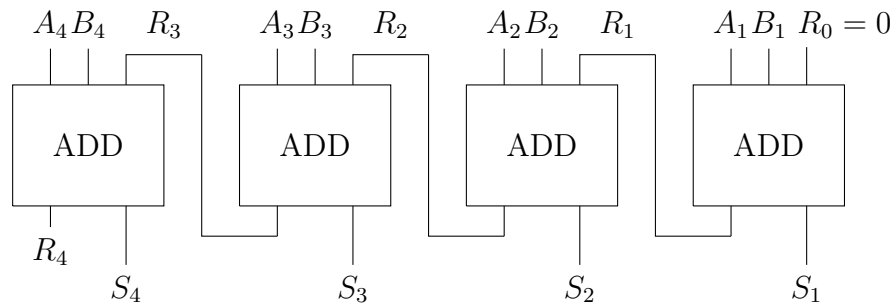
Etape 5. Le schéma du circuit (logigramme).



3.3.1.3 L'additionneur n bits

Pour additionner deux nombres binaires sur plusieurs bits (supérieur ou égale à 2), on peut utiliser des additionneurs complets élémentaires montés en cascade.

Exemple : Additionneur 4 bits à partir d'additionneurs complets élémentaires.
 $A(A_4A_3A_2A_1) + B(B_4B_3B_2B_1) = S(S_4S_3S_2S_1)$ avec une retenue R_4 .

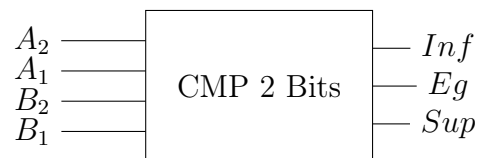


3.3.2 Le comparateur 2 bits

Définition : Le comparateur 2 bits est un circuit combinatoire qui permet de comparer entre deux nombres binaires $A(A_2A_1)$ et $B(B_2B_1)$ chacun sur 2 bits.

Synthèse du circuit :

Etape 1. Identification des entrées et des sorties.



Etape 2. La table de vérité du circuit.

A_2	A_1	B_2	B_1	Inf	Eg	Sup
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Etape 3. Identification des fonctions logiques des deux sorties S et R.

$$Inf = \overline{A_2}.\overline{A_1}.\overline{B_2}.B_1 + \overline{A_2}.\overline{A_1}.B_2.\overline{B_1} + \overline{A_2}.\overline{A_1}.B_2.B_1 + \overline{A_2}.A_1.B_2.\overline{B_1} + \overline{A_2}.A_1.B_2.B_1 + A_2.\overline{A_1}.B_2.B_1$$

$$Eg = \overline{A_2}.\overline{A_1}.\overline{B_2}.\overline{B_1} + \overline{A_2}.A_1.\overline{B_2}.B_1 + A_2.\overline{A_1}.B_2.\overline{B_1} + A_2.A_1.B_2.B_1$$

$$Sup = \overline{A_2}.A_1.\overline{B_2}.\overline{B_1} + A_2.\overline{A_1}.\overline{B_2}.\overline{B_1} + A_2.\overline{A_1}.\overline{B_2}.B_1 + A_2.A_1.\overline{B_2}.\overline{B_1} + A_2.A_1.\overline{B_2}.B_1 + A_2.A_1.B_2.\overline{B_1}$$

Etape 4. Simplification des fonctions obtenues.

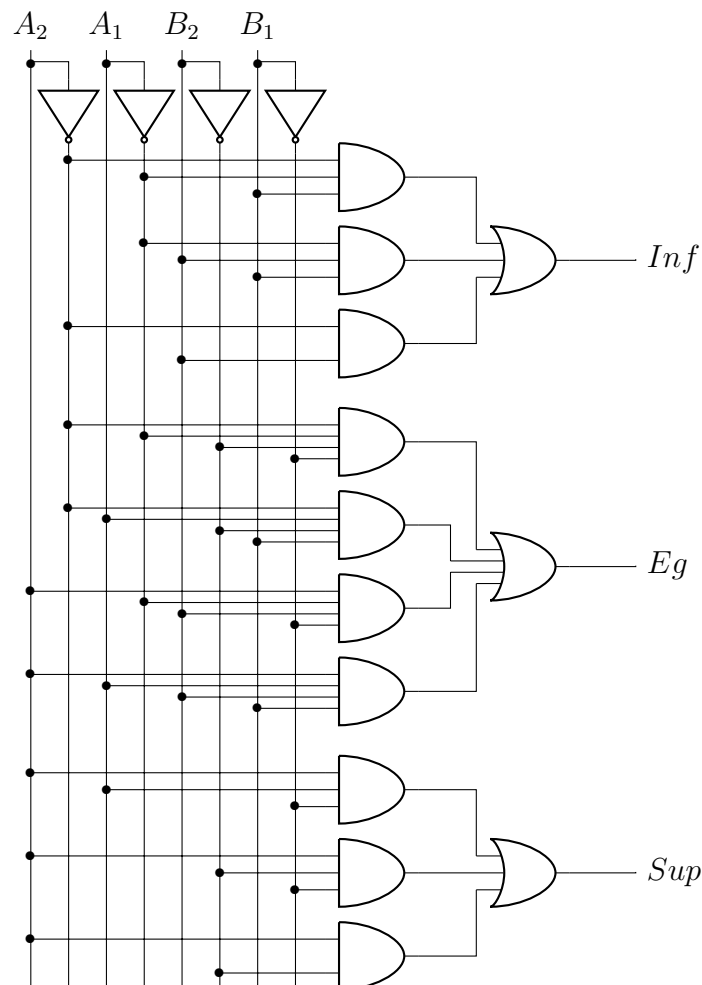
		<i>Inf</i>						<i>Sup</i>			
B_2B_1	A_2A_1	00	01	11	10	B_2B_1	A_2A_1	00	01	11	10
00		0	0	0	0	00		0	1	1	1
01		1	0	0	0	01		0	0	1	1
11		1	1	0	1	11		0	0	0	0
10		1	1	0	0	10		0	0	1	0

$$Inf = \overline{A_2}.\overline{A_1}.B_1 + \overline{A_1}.B_2.B_1 + \overline{A_2}.B_2$$

$$Eg = \overline{A_2}.\overline{A_1}.\overline{B_2}.\overline{B_1} + \overline{A_2}.A_1.\overline{B_2}.B_1 + A_2.\overline{A_1}.B_2.\overline{B_1} + A_2.A_1.B_2.B_1$$

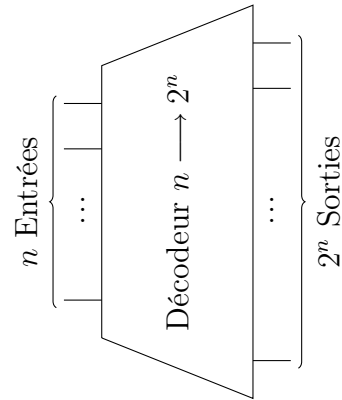
$$Sup = A_2.A_1.\overline{B_1} + A_1.\overline{B_2}.\overline{B_1} + A_2.\overline{B_2}$$

Etape 5. Le schéma du circuit (logigramme).



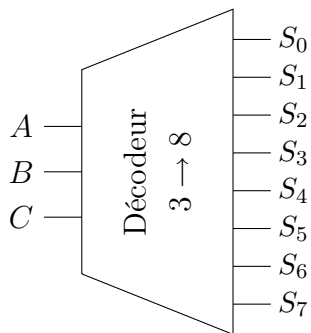
3.3.3 Le décodeur

Définition : Un décodeur k bits est un circuit logique combinatoire à k entrées et 2^k sorties. Pour chaque combinaison en entrée, la sortie qui porte le numéro correspondant à cette combinaison passe à 1 (elle est activée) tandis que toutes les autres sont à 0 (inactives). Les sorties du décodeur sont donc mutuellement exclusives, c'est à dire, pour une combinaison binaire de n entrées, une seule sortie sera mise à 1.



Synthèse du circuit (Décodeur à 3 entrées) :

1. Les entrées/sorties.
2. La table de vérité.



A	B	C	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

3. Les fonctions logiques.
4. Le schéma du circuit (le logigramme).

$$S_0 = \overline{A}.\overline{B}.\overline{C}$$

$$S_1 = \overline{A}.\overline{B}.C$$

$$S_2 = \overline{A}.B.\overline{C}$$

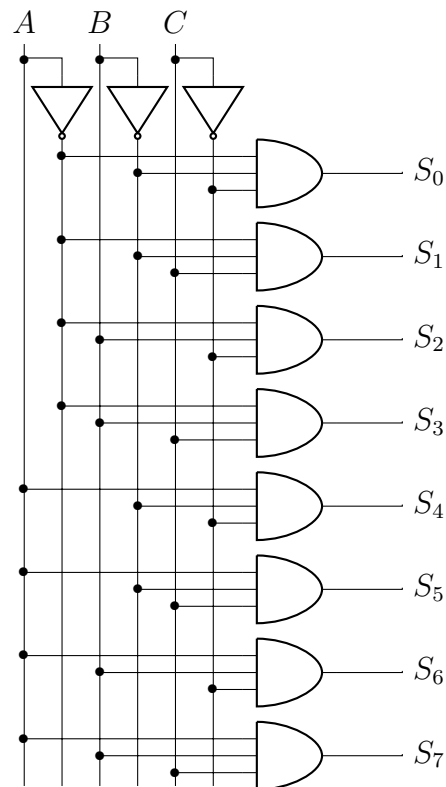
$$S_3 = \overline{A}.B.C$$

$$S_4 = A.\overline{B}.\overline{C}$$

$$S_5 = A.\overline{B}.C$$

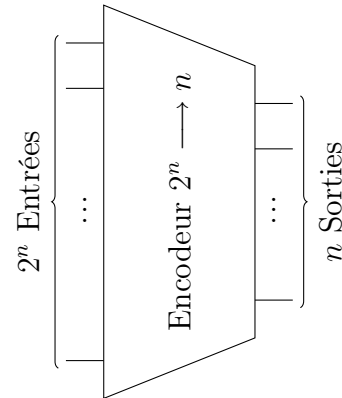
$$S_6 = A.B.\overline{C}$$

$$S_7 = A.B.C$$



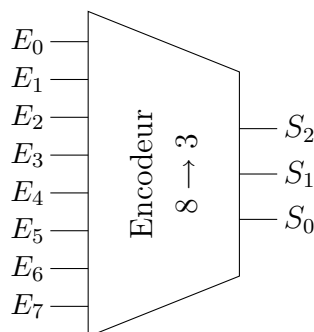
3.3.4 L'encodeur

Définition : Un encodeur k bits est un circuit logique combinatoire à 2^k entrées et k sorties. C'est un circuit qui joue le rôle inverse d'un décodeur. Pour chaque entrée active, la combinaison correspondant à cette entrée est affichée dans les bits de sortie. Tout comme pour les sorties du décodeur, les entrées de l'encodeur sont mutuellement exclusives, c'est à dire, une seule entrée sera active à la fois.



Synthèse du circuit (Encodeur à 8 entrées) :

1. Les entrées/sorties.
2. La table de vérité.



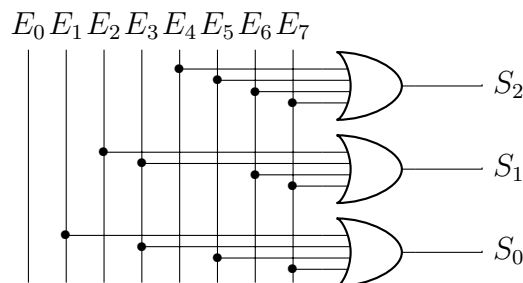
E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	S_2	S_1	S_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

3. Les fonctions logiques.
4. Le schéma du circuit (le logigramme).

$$S_2 = E_4 + E_5 + E_6 + E_7$$

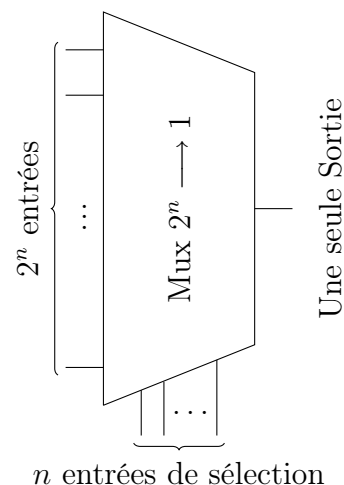
$$S_1 = E_2 + E_3 + E_6 + E_7$$

$$S_0 = E_1 + E_3 + E_5 + E_7$$



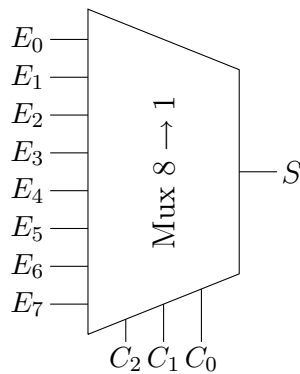
3.3.5 Le multiplexeur

Définition : Le multiplexeur est un circuit logique combinatoire qui permet de transmettre sur une seule ligne de sortie des informations en provenance de plusieurs entrées. Il sélectionne une entrée (1 bit) parmi 2^n entrées possibles, en introduisant le numéro de cette dernière dans les n entrées de sélection, et transmet l'information portée par cette ligne à un seul canal de sortie. Il possède, comme le montre la figure ci-contre, 2^n entrées d'information, n entrées de sélection (commandes) et une seule sortie.



Synthèse du circuit (Multiplexeur à 8 entrées) :

1. Les entrées/sorties.



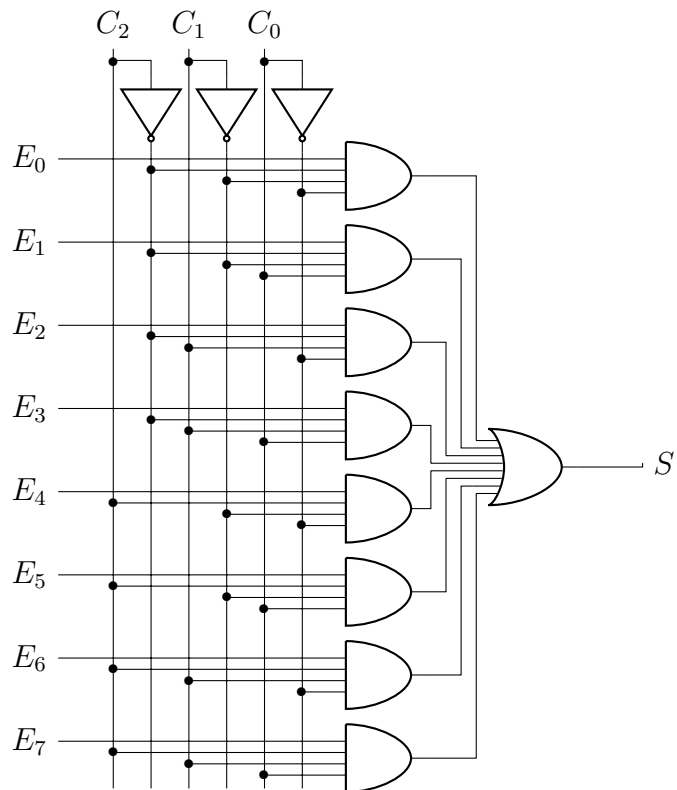
2. La table de vérité.

C_2	C_1	C_0	S
0	0	0	E_0
0	0	1	E_1
0	1	0	E_2
0	1	1	E_3
1	0	0	E_4
1	0	1	E_5
1	1	0	E_6
1	1	1	E_7

3. Les fonctions logiques.

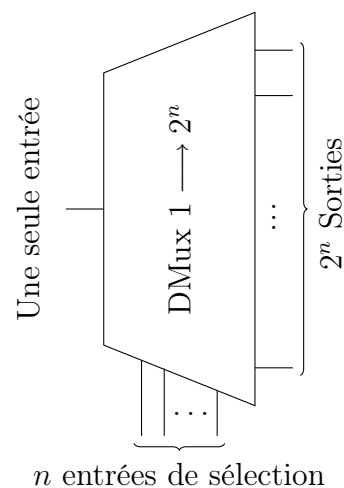
$$\begin{aligned}
 S = & \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0} \cdot E_0 + \\
 & \overline{C_2} \cdot \overline{C_1} \cdot C_0 \cdot E_1 + \\
 & \overline{C_2} \cdot C_1 \cdot \overline{C_0} \cdot E_2 + \\
 & \overline{C_2} \cdot C_1 \cdot C_0 \cdot E_3 + \\
 & C_2 \cdot \overline{C_1} \cdot \overline{C_0} \cdot E_4 + \\
 & C_2 \cdot \overline{C_1} \cdot C_0 \cdot E_5 + \\
 & C_2 \cdot C_1 \cdot \overline{C_0} \cdot E_6 + \\
 & C_2 \cdot C_1 \cdot C_0 \cdot E_7
 \end{aligned}$$

4. Le schéma du circuit (le logigramme).



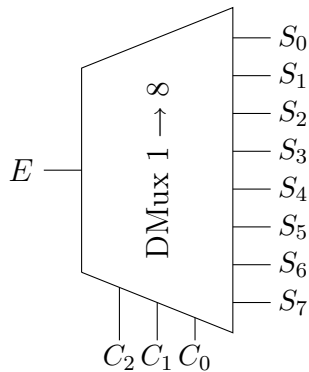
3.3.6 Le démultiplexeur

Définition : Le démultiplexeur est un circuit logique combinatoire qui joue le rôle inverse du multiplexeur. Il permet de transmettre à partir d'une seule ligne d'entrée des informations à destination de plusieurs sorties. Il sélectionne une sortie parmi 2^n sorties possibles, via les n entrées de sélection, et transmet l'information portée par la seule ligne d'entrée à travers le canal de la sortie sélectionnée. Il possède, comme le montre la figure ci-contre, une seule entrée d'information, n entrées de sélection (commandes) et 2^n sorties.



Synthèse du circuit (Démultiplexeur à 8 sorties) :

1. Les entrées/sorties.
2. La table de vérité.



C_2	C_1	C_0	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	E	0	0	0	0	0	0	0
0	0	1	0	E	0	0	0	0	0	0
0	1	0	0	0	E	0	0	0	0	0
0	1	1	0	0	0	E	0	0	0	0
1	0	0	0	0	0	0	E	0	0	0
1	0	1	0	0	0	0	0	E	0	0
1	1	0	0	0	0	0	0	0	E	0
1	1	1	0	0	0	0	0	0	0	E

3. Les fonctions logiques.
4. Le schéma du circuit (le logigramme).

$$S_0 = \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0} \cdot E$$

$$S_1 = \overline{C_2} \cdot \overline{C_1} \cdot C_0 \cdot E$$

$$S_2 = \overline{C_2} \cdot C_1 \cdot \overline{C_0} \cdot E$$

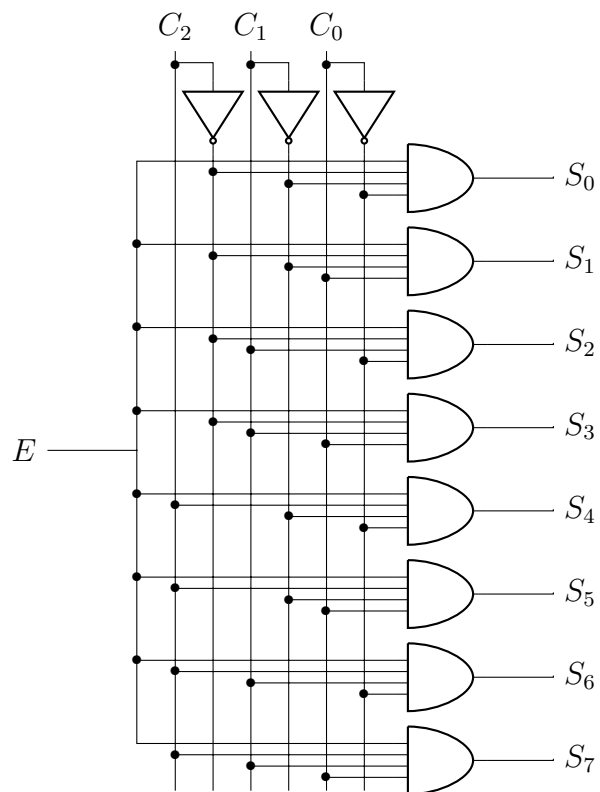
$$S_3 = \overline{C_2} \cdot C_1 \cdot C_0 \cdot E +$$

$$S_4 = C_2 \cdot \overline{C_1} \cdot \overline{C_0} \cdot E$$

$$S_5 = C_2 \cdot \overline{C_1} \cdot C_0 \cdot E$$

$$S_6 = C_2 \cdot C_1 \cdot \overline{C_0} \cdot E$$

$$S_7 = C_2 \cdot C_1 \cdot C_0 \cdot E$$



Remarque : Tout comme pour les additionneurs complets, on peut cascader plusieurs démultiplexeurs pour obtenir un démultiplexeur d'ordre supérieur. La même chose pour les décodeurs, les encodeurs et les multiplexeurs (voir la série de TD).

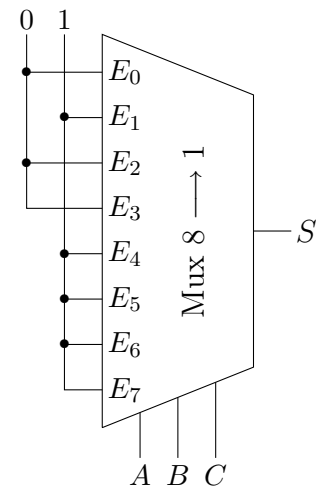
3.3.7 Génération des fonctions logiques via des multiplexeurs

Un des intérêts importants d'un multiplexeur est sa capacité d'être utilisé pour générer une fonction logique quelconque à partir de sa table de vérité. Il suffit d'utiliser les variables de la fonction à générer comme entrées d'adresses (commandes) du multiplexeur, et de relier l'entrée sélectionnée par chaque adresse à 0 ou à 1, selon que la fonction vaut 0 ou 1 dans la table de vérité.

Exemple : réaliser la fonction F suivante en utilisant un multiplexeur à 8 entrées.

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

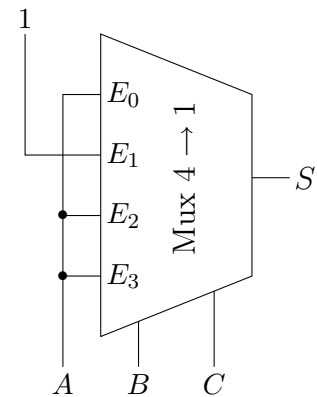
A	B	C	F(A, B, C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



On peut également réaliser cette même fonction avec un multiplexeur à 4 entrées. Pour ce faire, il suffit de choisir deux variables pour être utilisées comme entrées de commande du multiplexeur et la variable qui reste sera utilisée comme entrée standard. Ensuite, il faut écrire la fonction sous la première forme canonique avant de mettre toutes les combinaisons possibles des variables choisies en facteur commun.

Exemple : réaliser la fonction F suivante en utilisant un multiplexeur à 4 entrées.

$$\begin{aligned}
 F(A, B, C) &= A.B + A.\overline{C} + \overline{B}.C \\
 &= A.B.(C + \overline{C}) + A.\overline{C}.(B + \overline{B}) + \overline{B}.C.(A + \overline{A}) \\
 &= A.B.C + A.B.\overline{C} + A.\overline{B}.C + A.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C \\
 &= (A).\overline{B}.C + (A + \overline{A}).\overline{B}.C + (A).\overline{B}.\overline{C} + (A).B.C \\
 &= (A).\underbrace{\overline{B}.C}_{E_0} + (1).\underbrace{\overline{B}.C}_{E_1} + (A).\underbrace{\overline{B}.\overline{C}}_{E_2} + (A).\underbrace{B.C}_{E_3}
 \end{aligned}$$



3.3.8 Génération des fonctions logiques via des décodeurs

Tout comme pour les multiplexeurs, il est aussi possible d'utiliser les décodeurs pour générer une fonction logique quelconque. Il suffit d'établir la table de vérité du décodeur et de la fonction à générer, puis à partir de cette table on va écrire la fonction à générer en fonction des sorties du décodeur.

Exemple : réaliser la fonction F suivante en utilisant un décodeur à 3 entrées.

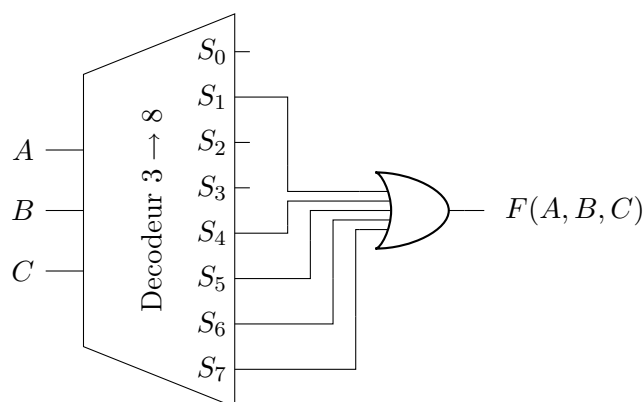
$$F(A, B, C) = \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

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

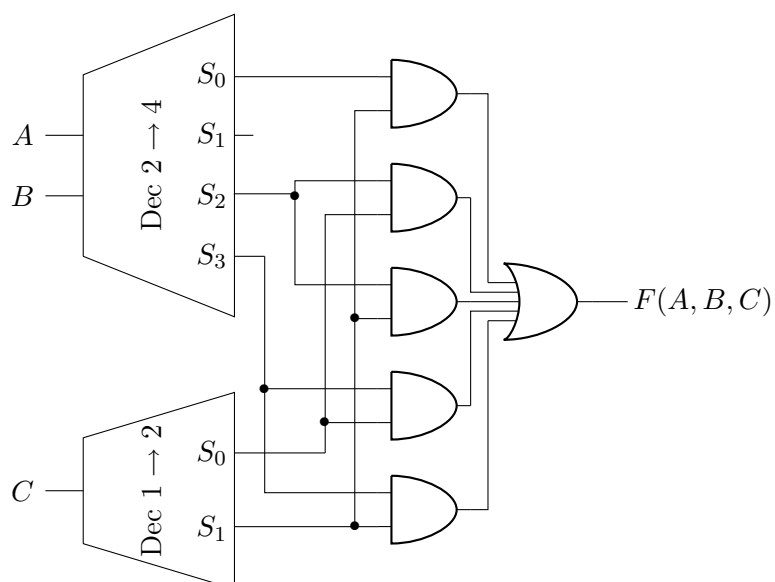
On en déduit alors l'expression de F en fonction des sorties du décodeur :

$$F(A, B, C) = S_1 + S_4 + S_5 + S_6 + S_7$$

On peut alors en déduire le schéma correspondant :



On peut également réaliser cette même fonction avec un décodeur à 2 entrées et un décodeur à 1 seule entrée, comme suit :

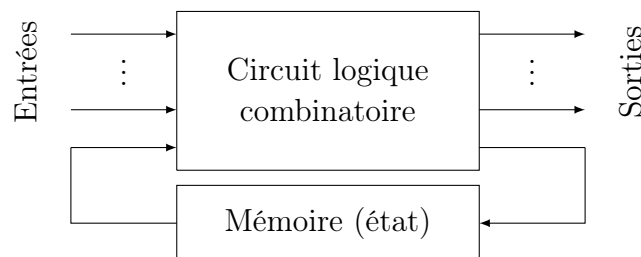


Chapitre 4

Circuits logiques séquentiels

4.1 Introduction

A l'inverse des circuits combinatoires, les circuits séquentiels sont des circuits logiques où les sorties ne dépendent pas uniquement des entrées, mais également de l'état actuel du système. Ces circuits sont donc dotés de dispositifs de mémorisation d'états logiques, comme le montre la figure suivante.



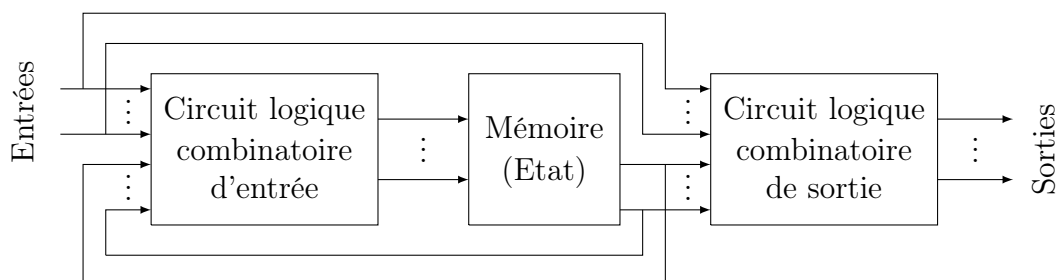
Tout comme pour les circuits combinatoires, l'étude des circuits séquentiels repose essentiellement sur l'algèbre de Boole, mais aussi sur la théorie des automates finis (ou graphes de transitions).

4.2 Modèles des circuits séquentiels

Suivant la façon dont les sorties dépendent des entrées et des états, un système séquentiel peut se représenter sous forme d'une machine de Mealy ou d'une machine de Moore.

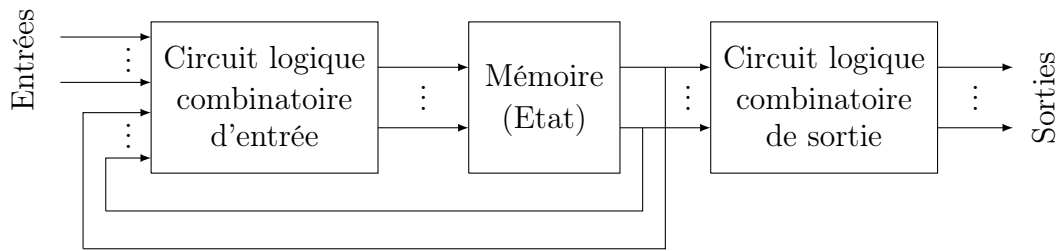
4.2.1 Machines de Mealy

Dans les machines de Mealy, les sorties dépendent de l'état actuel de la machine et des valeurs d'entrées.



4.2.2 Machines de Moore

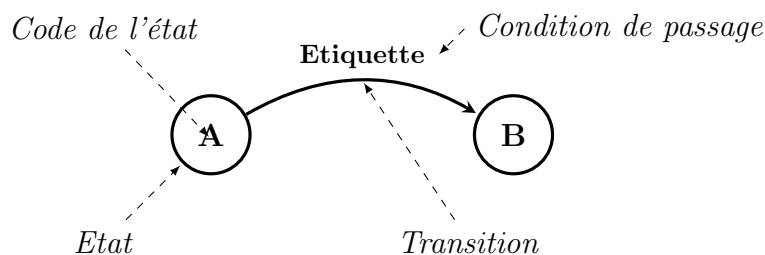
Dans les machines de Moore, les sorties ne dépendent que de l'état actuel de la machine.



4.3 Automates à états finis (graphes de transitions)

Dans les circuits séquentiels, on utilise souvent les termes *état présent* et *état futur* pour distinguer les états internes du circuit d'un instant à un autre. Le passage d'un état à un autre est souvent schématisé par un graphe appelé *graphe de transitions*.

Dans ce graphe, chaque état est représenté par un cercle et codé par un numéro ou un code, les états du circuit sont reliés entre eux par des arcs étiquetés appelés transitions dont les étiquettes représentent la condition de passage de l'état source à l'état destination. Cette condition est souvent représentée par la combinaison des valeurs des variables d'entrée qui provoque la transition correspondante.

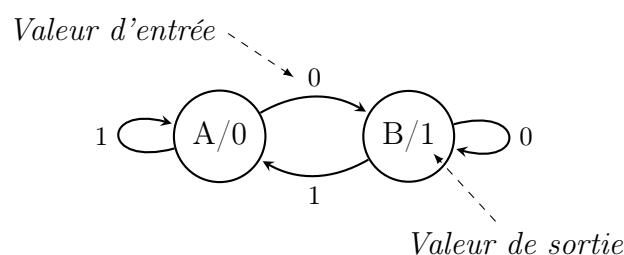


4.3.1 Types de graphes de transitions

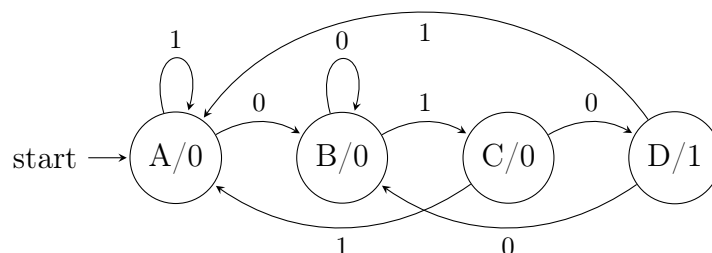
Suivant la position dont les sorties sont placées dans les états ou dans les transitions, on distingue deux types de graphes de transitions : les graphes de transitions de Moore et les graphes de transitions de Mealy.

4.3.1.1 Graphes de Moore

Dans les graphes de Moore, les valeurs de sortie sont placées dans les états du graphe, juste à côté du code de l'état.

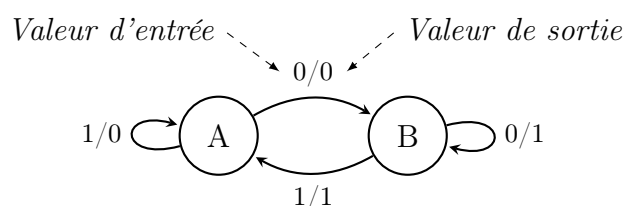


Exemple : Graphe de Moore correspondant au circuit qui détecte la séquence 010.

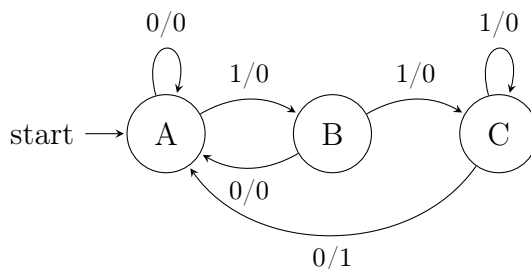


4.3.1.2 Graphes de Mealy

Dans les graphes de Mealy, les valeurs de sortie sont placées dans les étiquettes des transitions du graphe, juste à côté des valeurs d'entrée.



Exemple : Graphe de Mealy correspondant au circuit qui détecte la séquence 110.

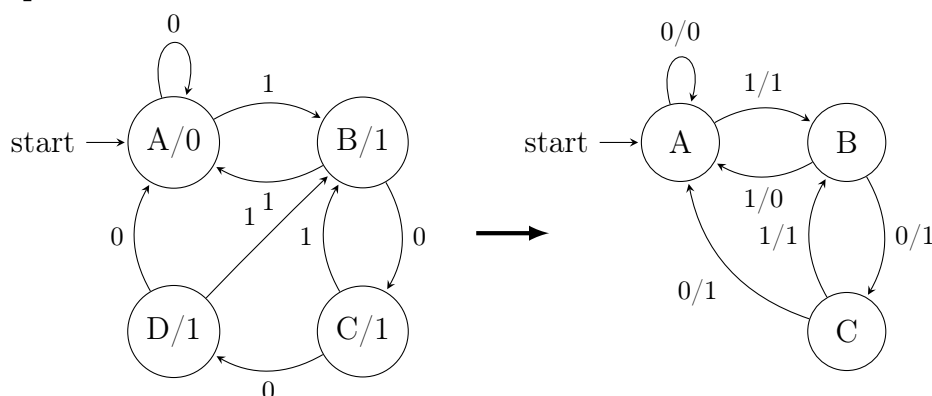


4.3.2 Passage du graphe de Moore au graphe de Mealy

Pour passer d'un graphe de Moore à un graphe de Mealy, il suffit de suivre les deux étapes suivantes :

- Etape 1.** Faire sortir la valeur de sortie de chaque état vers les transitions entrantes.
- Etape 2.** Eliminer les états dupliqués qui ont les mêmes transitions de sortie avec les mêmes valeurs d'entrée et de sortie.

Exemple :

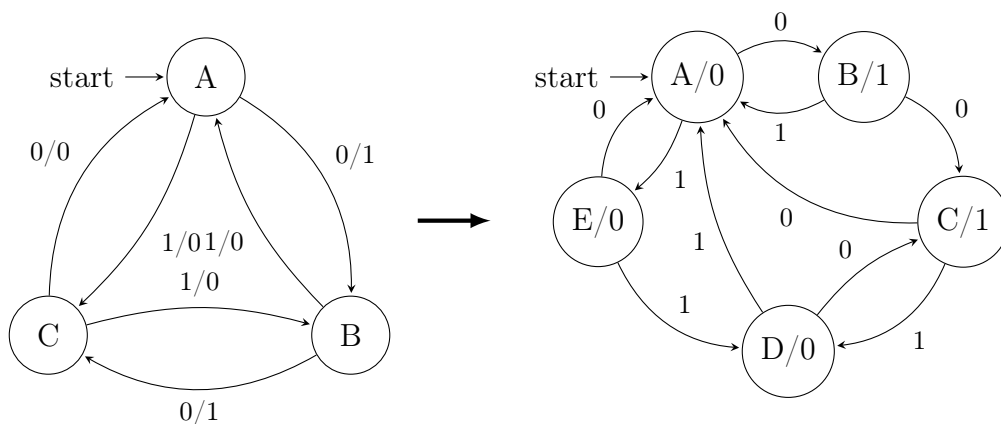


4.3.3 Passage du graphe de Mealy au graphe de Moore

Pour passer d'un graphe de Mealy à un graphe de Moore, on doit passer par les deux étapes suivantes :

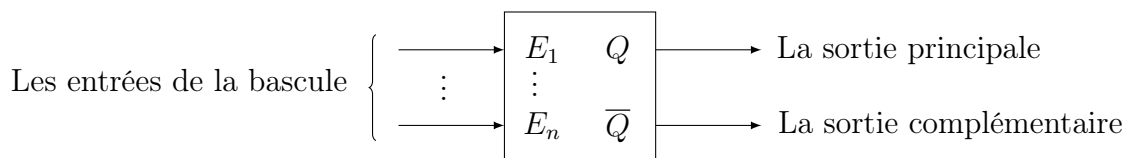
- Etape 1.** Faire entrer les valeurs de sortie des transitions entrant de chaque état à l'intérieur de ce dernier. S'il y a plusieurs valeurs différentes, créer autant d'états que le nombre de ces valeurs.
- Etape 2.** Créer, pour chaque nouvel état, des transitions sortantes identiques à celles de l'état original.

Exemple :



4.4 Circuits séquentiels à base de bascules

Les bascules (ou les bistables) sont les éléments de base de la logique séquentiel moderne, ils permettent de mémoriser une information élémentaire 0 ou 1 (un bit) dans leur sortie principale, et de conserver cette information jusqu'à l'apparition d'une impulsion qui demande le changement de cette dernière. Comme le montre la figure suivante, une bascule possède généralement une ou plusieurs entrées, une sortie principale notée Q et éventuellement une sortie complémentaire notée \bar{Q} . Si $Q = 0$, on dit que la bascule est à l'état 0, sinon (si $Q = 1$) on dit que la bascule est à l'état 1.



4.4.1 Types de bascules

Les circuits séquentiels à base de bascules peuvent être différenciés en fonction de la nature et du mode de fonctionnement de leurs bascules qui peuvent être synchrones ou asynchrones :

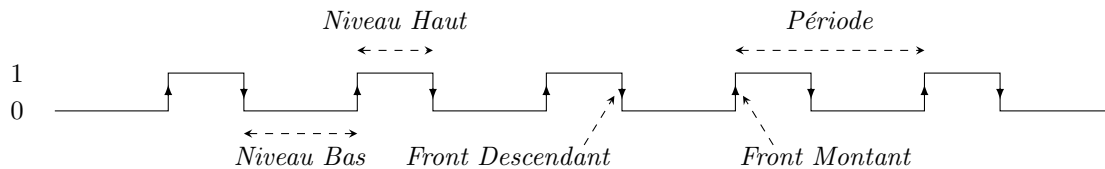
4.4.1.1 Bascules asynchrones

Les bascules asynchrones sont des circuits logiques pour lesquels les valeurs de sortie changent spontanément à la suite d'un changement des valeurs d'entrées.

4.4.1.2 Bascules synchrones

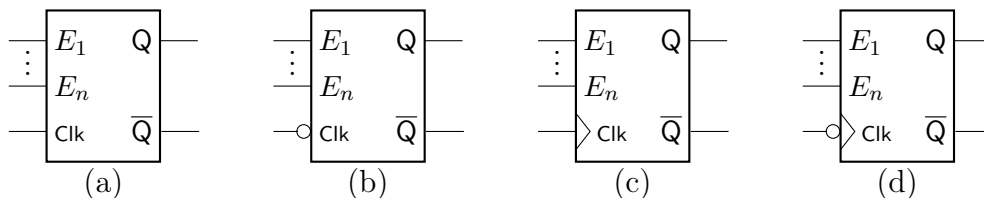
Les bascules synchrones sont des circuits logiques pour lesquels les valeurs de sortie ne changent qu'à la réception d'un signal de commande spécifique appelé *signal d'horloge*.

Le signal d'horloge est un signal périodique dont sa période se divise en deux parties (niveaux) : une partie haute et une partie basse, comme le montre la figure suivante.



Une bascule synchrone peut être synchronisée par niveau d'impulsion de l'horloge (haut ou bas) ou par front d'impulsion de l'horloge (montant ou descendant). Dans la synchronisation par niveau d'impulsion, les valeurs des sorties changent lorsque l'horloge est à un niveau bien défini (haut ou bas). Tandis que, dans la synchronisation par front d'impulsion, les valeurs des sorties changent quand le signal d'horloge effectue un front montant (signal passant de 0 à 1) ou descendant (signal passant de 1 à 0).

Les bascules synchrones disposent donc, en plus des entrées et de la sortie principale et complémentaire, d'une entrée d'horloge Clk (ou H) permettant de contrôler le changement des valeurs de sortie. La représentation électronique de cette entrée dépend principalement du mode d'activation du signal d'horloge comme le montre la figure suivante.



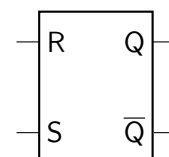
- (a) Bascule active sur le niveau haut de l'horloge.
- (b) Bascule active sur le niveau bas de l'horloge.
- (c) Bascule active sur le front montant de l'horloge (*positive edge triggered*).
- (d) Bascule active sur le front descendant de l'horloge (*negative edge triggered*).

4.4.2 Bascules de base des circuits séquentiels

Dans la logique séquentiel, on distingue cinq types différents de bascules : La bascule RS, la bascule RSH, la bascule D, la bascule JK et la bascule T.

4.4.2.1 La bascule RS

La bascule RS est la bascule la plus simple qui constitue la base de toutes les autres bascules. Ces dernières ne sont en fait que des évolutions de cette bascule. Elle est considérée comme étant la seule bascule asynchrone (sans entrée d'horloge) parmi toutes les autres bascules. Elle dispose de deux entrées R et S, R pour la remise à 0 (Reset) de la sortie Q, et S pour la mise à 1 (Set) de la sortie Q.



1. Table de transitions

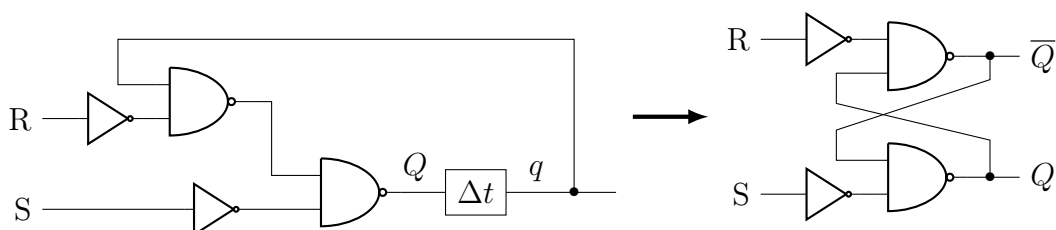
Entrées		Etat actuel	Etat futur	Description
R	S	q	Q	
0	0	0	0	Mémoire
0	0	1	1	
0	1	0	1	Mise à 1
0	1	1	1	
1	0	0	0	Remise à 0
1	0	1	0	
1	1	0	x	Interdit
1	1	1	x	

2. Expression simplifiée de Q .

RS		00	01	11	10
q					
0		0	1	x	0
1		1	1	x	0

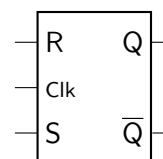
$$Q = S + \overline{R}.q = \overline{\overline{S}. \overline{\overline{R}.q}}$$

3. Schéma logique de la bascule RS.



4.4.2.2 La bascule RSH

La bascule RSH est une bascule RS synchronisée par un signal d'horloge H (ou Clk). Lorsque H est inactive ($H=0$), la bascule est dans l'état mémoire (rien ne va changer, même si R et S changent). Lorsque H est active ($H=1$), la bascule fonctionne comme une bascule RS.



1. Table de transitions

Entrées			Etat actuel	Etat futur	Description
H	R	S	q	Q	
0	0	0	0	0	Mémoire
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	1	
1	0	1	0	1	Mise à 1
1	0	1	1	1	Remise à 0
1	1	0	0	0	
1	1	0	1	0	Interdit
1	1	1	0	x	
1	1	1	1	x	

2. Expression simplifiée de Q .

Hq	RS			
	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	x	0
10	0	1	x	0

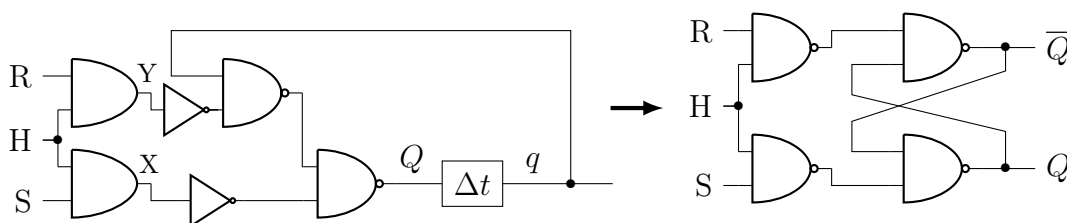
$$\begin{aligned}
 Q &= \overline{H}.q + \overline{R}.q + S.H \\
 &= (\overline{H} + \overline{R}).q + S.H \\
 &= S.H + (\overline{H}.\overline{R}).q \\
 &= X + \overline{Y}.q
 \end{aligned}$$

Tel que : $X = S.H$ et $Y = R.H$

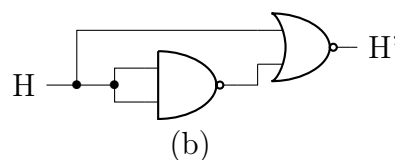
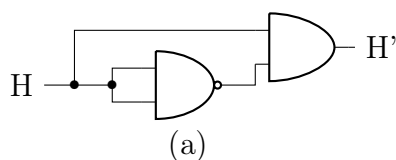
Donc,

- Pour $H = 0$: $Q = q$
- Pour $H = 1$: $Q = S + \overline{R}.q$

3. Schéma logique de la bascule RSH.

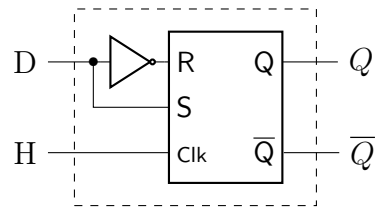


Remarque : La synchronisation sur front d'horloge (montant ou descendant) peut être effectuée en intercalant un circuit supplémentaire entre l'entrée d'horloge et le signal d'horloge original. Les deux figures ci-contre montrent un exemple du circuit supplémentaire pour obtenir une synchronisation sur front montant (*circuit (a)*) ou sur front descendant (*circuit (b)*).



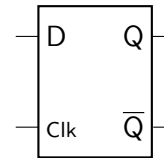
4.4.2.3 La bascule D

La bascule D est une bascule synchrone à une seule entrée de donnée D. Elle est conçue sur le même principe que celui de la bascule RSH en éliminant la combinaison interdite (R,S)=(1,1) par la liaison des deux entrées R et S, comme le montre la figure ci-contre, en ajoutant un inverseur à l'entrée R à partir de l'entrée S.



La bascule D ne garde donc que les trois combinaisons (R,S) = (0,0), (0,1) et (1,0).

- La combinaison (R,S) = (0,0) est assurée par H=0,
- La combinaison (R,S) = (0,1) est assurée par D=1,
- La combinaison (R,S) = (1,0) est assurée par D=0.



Le fonctionnement de la bascule D peut donc se résumer comme suit :

- Quand l'entrée d'horloge est inactive (H=0) : $Q = q$
- Quand l'entrée d'horloge est active (H=1) : $Q = D$

La bascule D peut donc être obtenue à partir de la structure de la bascule RSH, mais également directement en repartant du cahier des charges comme suit.

1. Table de transitions

Entrées		Etat actuel	Etat futur	Description
H	D	q	Q	
0	0	0	0	Mémorisation
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	Remise à 0
1	0	1	0	
1	1	0	1	Mise à 1
1	1	1	1	

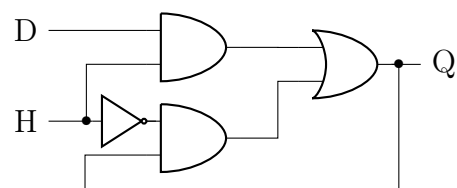
2. Expression simplifiée de Q.

$$\begin{aligned}
 Q &= \overline{H} \cdot \overline{D} \cdot q + \overline{H} \cdot D \cdot q + H \cdot D \cdot \overline{q} + H \cdot D \cdot q \\
 &= \overline{H} \cdot q \cdot (\overline{D} + D) + H \cdot D \cdot (\overline{q} + q) \\
 &= \overline{H} \cdot q + H \cdot D
 \end{aligned}$$

Donc : - Pour $H = 0$: $Q = q$

- Pour $H = 1$: $Q = D$

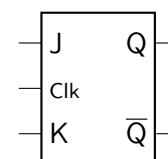
3. Schéma logique de la bascule D.



Remarque : La bascule D existe sous deux formes différentes : la forme *D-latch* qui est active par niveau d'horloge (haut ou bas) et la forme *edge triggered D flip-flop* qui est active sur front d'horloge (montant ou descendant).

4.4.2.4 La bascule JK

La bascule JK est une bascule synchrone disposant de deux entrées J et K, J pour la mise à 1 et K pour la remise à 0. Elle a un fonctionnement identique à celui de la bascule RSH, sauf que dans la bascule JK, il n'y a plus de combinaison interdite. La combinaison (J,K)=(1,1) est autorisée et conduit à l'inversion de l'état courant de la bascule.



1. Table de transitions

Entrées			Etat actuel	Etat futur	Description
H	J	K	q	Q	
0	0	0	0	0	Mémoire
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	Mémoire
1	0	0	1	1	
1	0	1	0	0	Remise à 0
1	0	1	1	0	
1	1	0	0	1	Mise à 1
1	1	0	1	1	
1	1	1	0	1	Inversion
1	1	1	1	0	

2. Expression simplifiée de Q .

Hq \ JK		00	01	11	10
		00	01	11	10
00	00	0	0	0	0
01	01	1	1	1	1
11	11	1	0	0	1
10	10	0	0	1	1

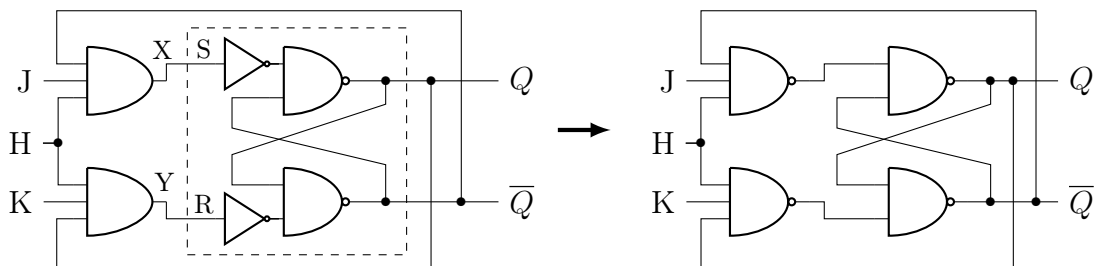
$$\begin{aligned}
 Q &= \overline{H}.q + \overline{K}.q + J.H.\overline{q} \\
 &= \overline{H}.q + \overline{K}.q + \overline{q}.q + J.H.\overline{q} \\
 &= (\overline{H} + \overline{K} + \overline{q}).q + J.H.\overline{q} \\
 &= J.H.\overline{q} + (\overline{H.K.q}).q \\
 &= X + \overline{Y}.q
 \end{aligned}$$

Où : $X = J.H.\overline{q}$ et $Y = H.K.q$

Donc :

- Pour $H = 0$: $Q = q$
- Pour $H = 1$: $Q = J.\overline{q} + \overline{K}.q$

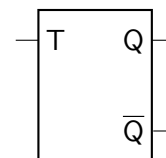
3. Schéma logique de la bascule JK.



Remarque : La bascule JK est généralement commandée par le front descendant.

4.4.2.5 La bascule T

La bascule T (T pour Toggle, c'est à dire bascule à déclenchement) est une bascule qui a une seule entrée T. A chaque impulsion arrivée à son entrée T, la bascule change d'état en inversant les valeurs de ses sorties.



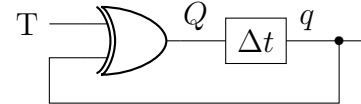
1. Table de transitions

Entrées	Etat actuel	Etat futur	Description
T	q	Q	Mémorisation
0	0	0	
0	1	1	
1	0	1	Inversion
1	1	0	

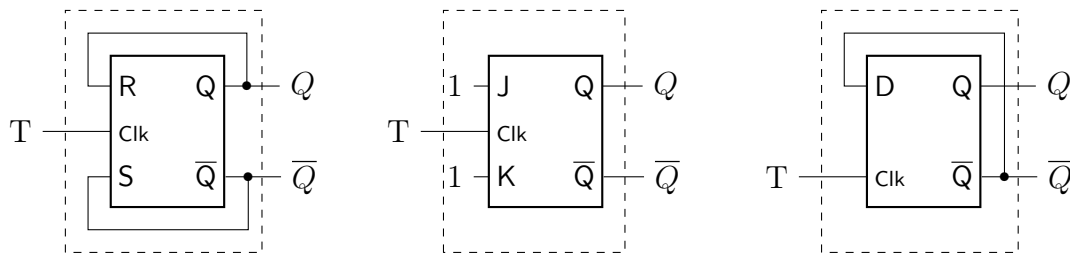
2. Expression simplifiée de Q .

$$Q = \bar{T}.q + T.\bar{q} = T \oplus q$$

3. Schéma logique.



La bascule T peut également être réalisée à partir des trois bascules précédentes (RSH, D et JK) comme suit :

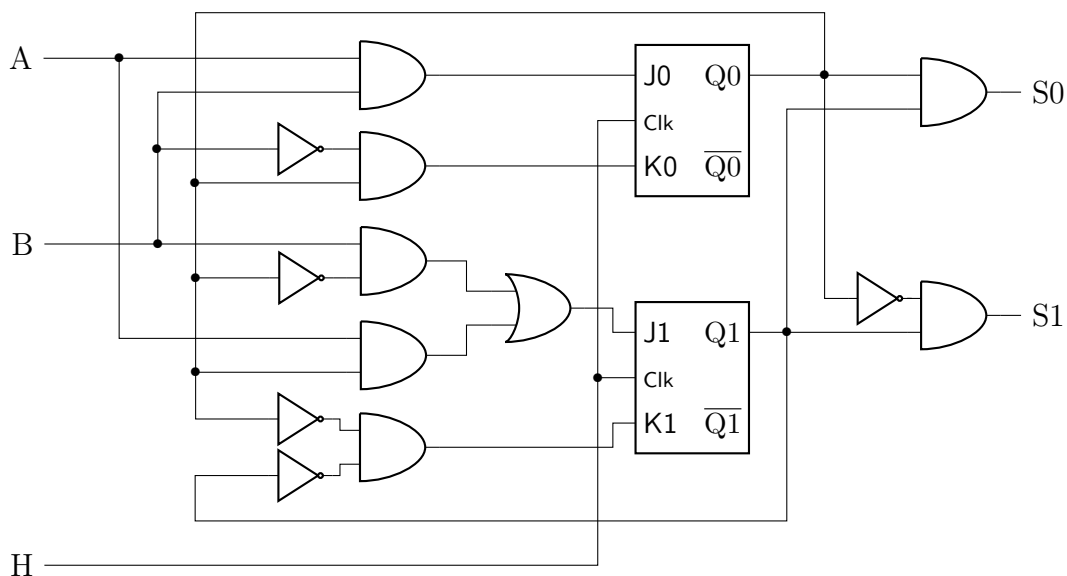


Remarque : La bascule T est essentiellement conçue pour être utilisée dans certaines applications et notamment pour la réalisation de compteurs, ainsi que pour diviser par 2 (2^n si n bascules T en série) la fréquence d'horloge.

4.4.3 Analyse des circuits séquentiels à bascules

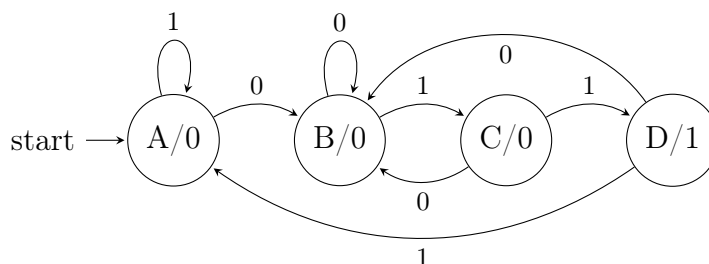
Pour analyser un circuit séquentiel à base de bascules, on passe par les mêmes étapes que celles des circuits séquentiels vues dans les sections précédentes, sauf qu'ici, les expressions d'excitation (c'est à dire, les expressions des sorties Q des différentes bascules constituant le circuit) sont écrites en fonction des expressions des entrées de la bascule. Par conséquent, la première étape consiste à extraire les expressions des entrées des différentes bascules, alors que la deuxième étape consiste à écrire les expressions des sorties Q des différentes bascules en fonction des expressions de leurs entrées. Les étapes qui restent sont les mêmes que celles des circuits séquentiels classiques.

Exemple : analyser le circuit séquentiel suivant :



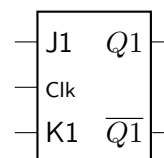
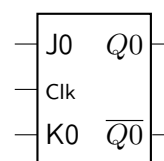
Exemple : synthèse du circuit séquentiel d'un détecteur de la séquence 011, en utilisant des bascules JK.

1. Graphe de transitions de Moore.



2. Table de transitions de Moore.

Etat actuel		Entrée	Etat futur		Sortie
$q1$	$q0$	E	$Q1$	$Q0$	S
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	0	0



3. Détermination des valeurs d'entrée des bascules.

J	K	q	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table réduite de la bascule JK.

q	Q	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Table des valeurs d'entrée des bascules.

Etat actuel		Entrée	Etat futur		Bascule 1		Bascule 0		Sortie
$q1$	$q0$	E	$Q1$	$Q0$	$J1$	$K1$	$J0$	$K0$	S
0	0	0	0	1	0	x	1	x	0
0	0	1	0	0	0	x	0	x	0
0	1	0	0	1	0	x	x	0	0
0	1	1	1	0	1	x	x	1	0
1	0	0	0	1	x	1	1	x	0
1	0	1	1	1	x	0	1	x	1
1	1	0	0	1	x	1	x	0	0
1	1	1	0	0	x	1	x	1	0

4. Expressions simplifiées des entrées des bascules et des sorties du circuit.

$q_1 q_0$					
		00	01	11	10
E	0	0	0	x	x
	1	0	1	x	x

J1

$q_1 q_0$					
		00	01	11	10
E	0	x	x	1	1
	1	x	x	1	0

K1

$q_1 q_0$					
		00	01	11	10
E	0	1	x	x	1
	1	0	x	x	1

J0

$q_1 q_0$					
		00	01	11	10
E	0	x	0	0	x
	1	x	1	1	x

K0

$$J1 = E.q0$$

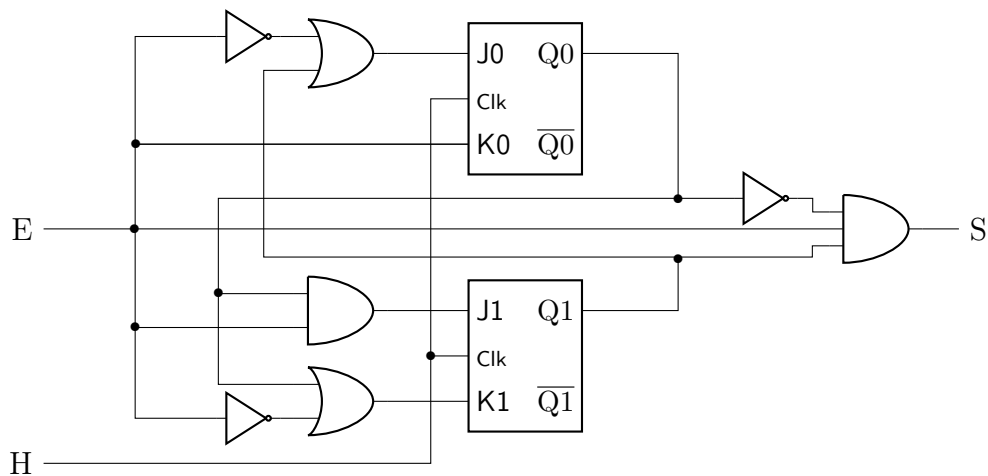
$$K1 = \bar{E} + q0$$

$$J0 = \bar{E} + q1$$

$$K0 = E$$

$$S = E.q1.\bar{q0}$$

5. Le schéma logique.

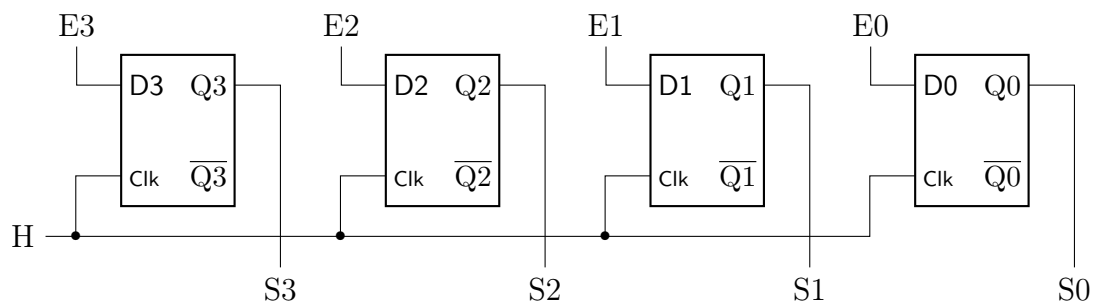


4.5 Application des circuits séquentiels à bascules

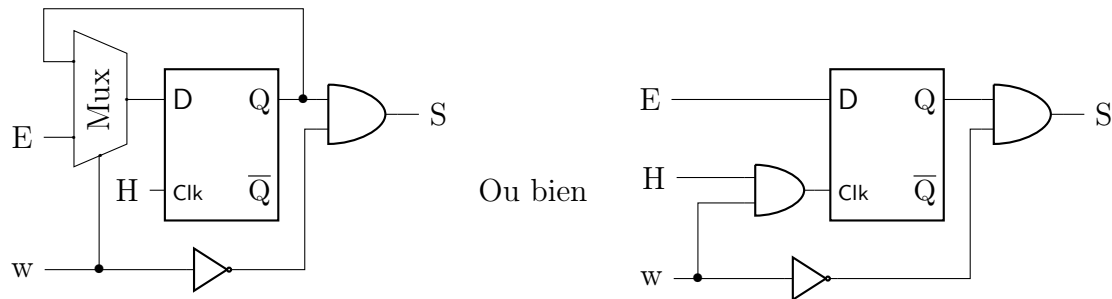
4.5.1 Les registres

4.5.1.1 Définition

Un registre est un ensemble de bascules mises en cascade afin de permettre de garder en mémoire une information binaire de façon provisoire. Le registre comporte donc autant de bascules que le nombre de bits à mémoriser.

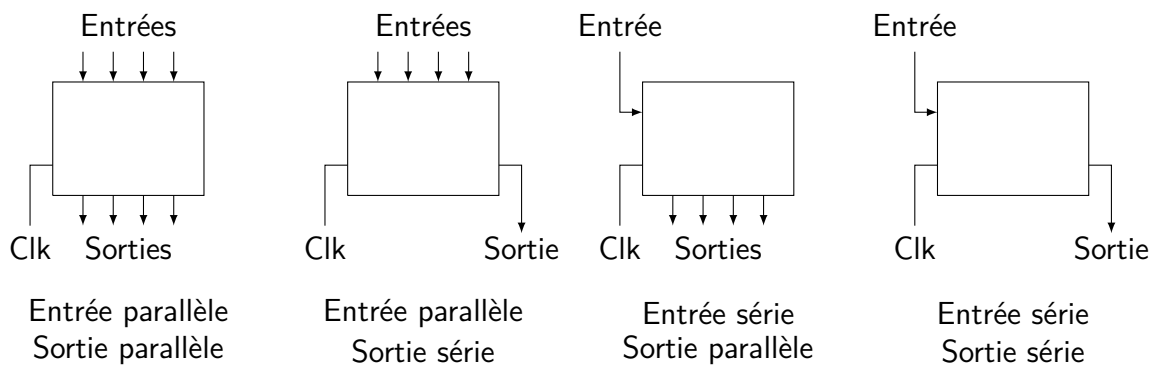


En plus des bascules qui le constituent, un registre dispose généralement d'une entrée supplémentaire de contrôle appelée entrée de *lecture/écriture* et noté w ($w=1$ pour l'écriture et $w=0$ pour la lecture). Avec cette entrée de contrôle, chaque bascule du registre va avoir la structure suivante :



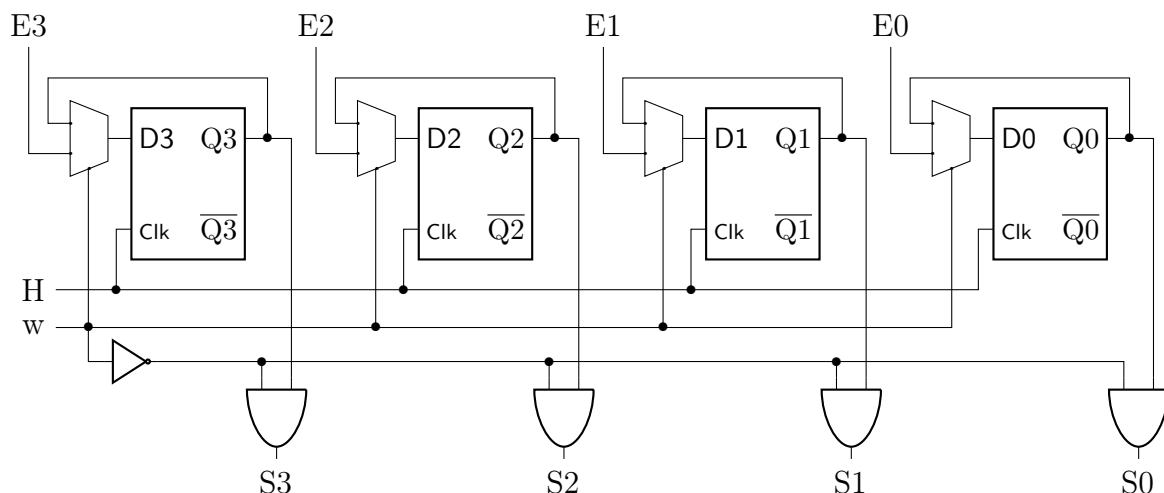
4.5.1.2 Types de registres

Selon la façon dont sont utilisées les entrées et les sorties des différentes bascules constituant le registre (en parallèle ou en série), nous pouvons distinguer quatre types différents de registres.

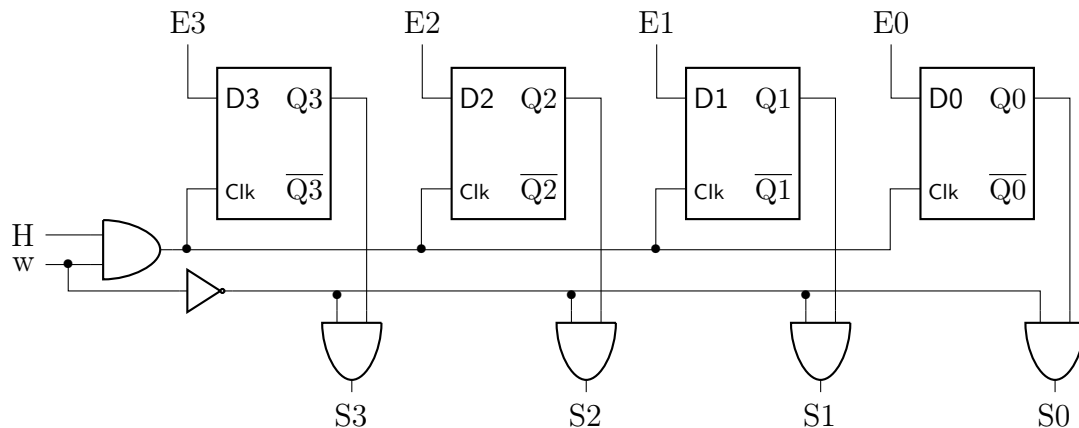


4.5.1.2.1 Registres à entrées parallèles et sorties parallèles

Dans ce type de registre, l'écriture est effectuée par la présentation des bits à écrire sur toutes les entrées de ce dernier. Les entrées du registre sont donc chargées simultanément (en parallèle) dans les bascules qui le constituent lors de l'activation du signal d'horloge H. De même, la lecture du mot stocké dans le registre s'effectue en parallèle sur les sorties des bascules qui le constituent.



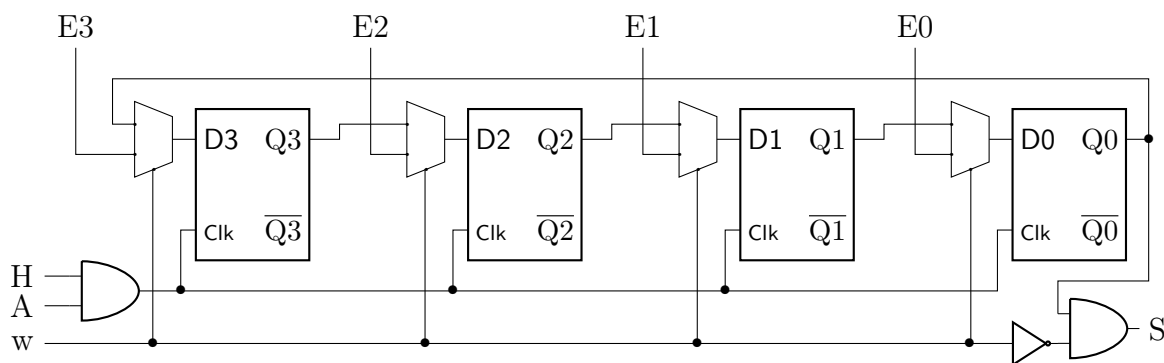
Ce registre peut également être réalisé en intercalant l'entrée w de *lecture/écriture* avec l'entrée d'horloge par une porte ET logique comme le montre la figure suivante.



Ce type de registre est appelé aussi registre de mémorisation ou registre tampon. Il est généralement utilisé pour mémoriser momentanément une information avant de la transmettre vers sa destination finale.

4.5.1.2.2 Registres à entrées parallèles et sorties série

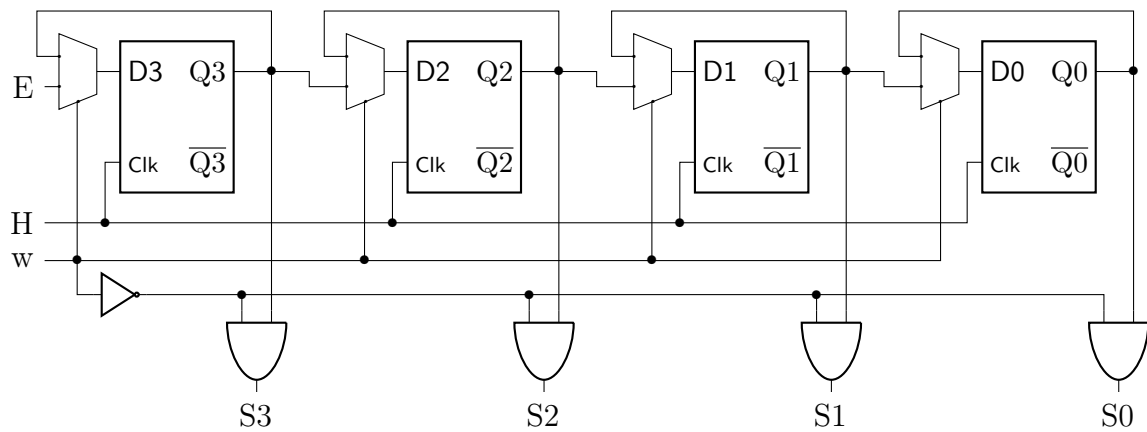
Dans ce type de registre, l'écriture est effectuée par la présentation de tous les bits à écrire sur les n entrées du registre simultanément. Alors que la lecture des bits stockés dans le registre s'effectue sur une seule sortie S , où les bits sont lus les uns après les autres au rythme de l'horloge. Afin d'éviter un décalage permanent, ce type de registre peut également avoir une entrée d'activation notée A permettant d'activer ($A = 1$) ou de désactiver ($A = 0$) le registre. Cette dernière sera intercalée avec l'entrée d'horloge par une porte ET logique.



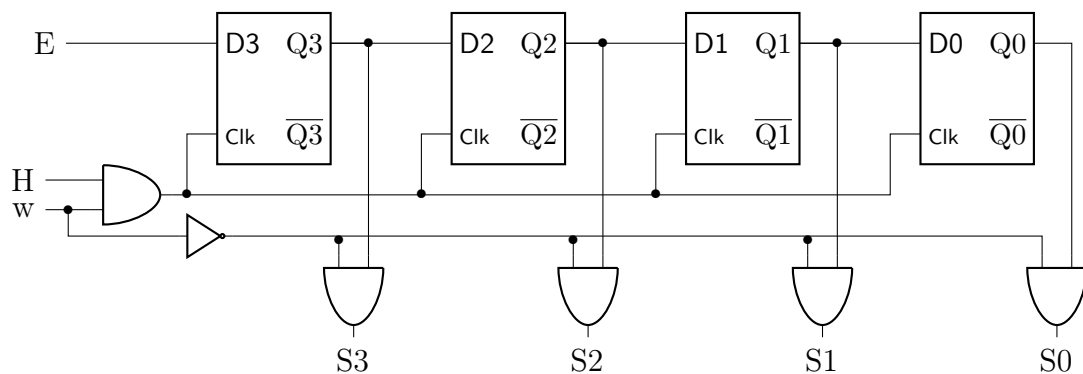
Ce type de registre est généralement utilisé comme convertisseur parallèle-série, il est surtout nécessaire à l'émission des données lors d'une transmission série.

4.5.1.2.3 Registres à entrées série et sorties parallèles

Dans ce type de registre, les bits à écrire sont présentés séquentiellement bit après bit à l'entrée de la première bascule. A chaque signal d'horloge un nouveau bit est introduit pendant que ceux déjà mémorisés sont décalés d'un niveau dans le registre. La lecture du mot stocké dans ce type de registre est effectué en prenant les valeurs de toutes les sorties en même temps (en parallèle).



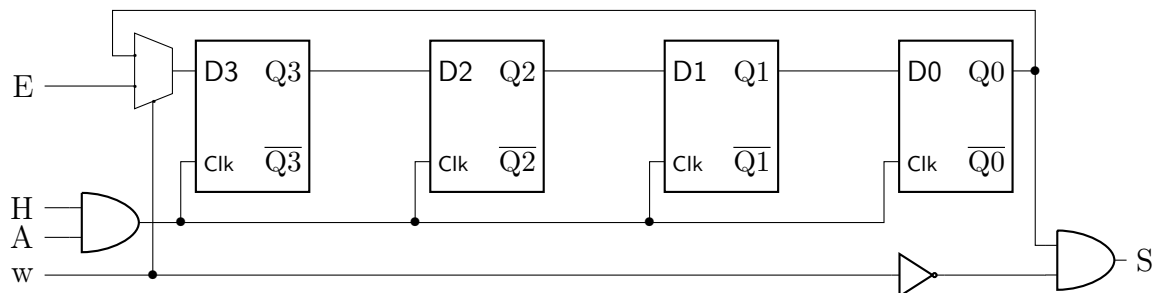
Ce registre peut également être réalisé en intercalant l'entrée w de *lecture/écriture* avec l'entrée d'horloge par une porte ET logique comme le montre la figure suivante.



Ce type de registre est généralement utilisé comme convertisseur série- parallèle, il est surtout nécessaire à la réception des données lors d'une transmission série.

4.5.1.2.4 Registres à entrées série et sorties série

Dans ce type de registres, les bits à écrire sont présentés bit après bit à l'entrée de la première bascule et se propagent à travers le registre à chaque impulsion d'horloge. Les bits chargés dans ce type de registre peuvent être récupérés les uns après les autres en se propageant bit après bit, au rythme de l'horloge, à la sortie de la dernière bascule du registre. Ce type de registre peut également avoir une entrée d'activation A afin d'éviter des décalages permanents.



Ce type de registre est généralement utilisé pour effectuer des opérations de décalage. Notamment pour effectuer des calculs arithmétiques binaires, comme par exemple la multiplication et la division à base de décalage.

4.5.2 Les compteurs

Un compteur est un ensemble de bascules mises en cascade de manière à compter des impulsions mises à leurs entrées d'horloge. Il s'agit donc d'un élément essentiel de la logique séquentielle qui permet de décrire, au rythme d'une horloge, une séquence déterminée d'événements.

Selon le mode de connexion des bascules, nous pouvons distinguer deux types de compteurs : les compteurs asynchrones et les compteurs synchrones.

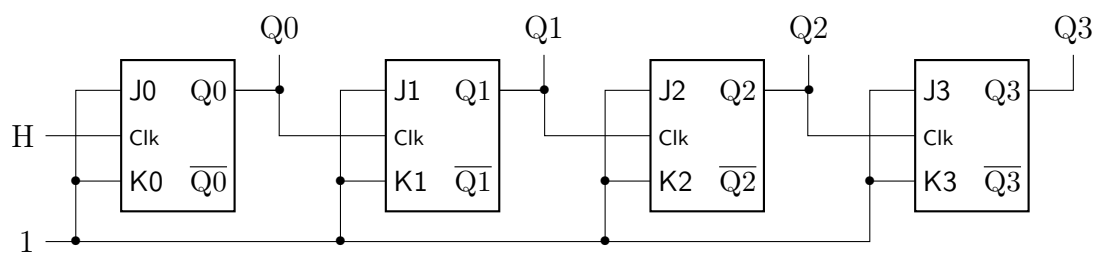
4.5.2.1 Les compteurs asynchrones

Dans ce type de compteur, les bascules qui le composent ne reçoivent pas le même signal d'horloge. Ceci est dû au fait que l'entrée d'horloge de chacune des bascules est reliée à la sortie de la bascule qui la précède. Ainsi, le signal d'horloge est branché sur l'entrée d'horloge de la première bascule ; la sortie de la première bascule sert d'horloge pour la 2ème bascule, et ainsi de suite. Par conséquent, les bascules de ce type de compteur sont asynchrones c'est-à-dire, elles ne changent pas d'état toutes en même temps à la transition du signal d'horloge d'entrée.

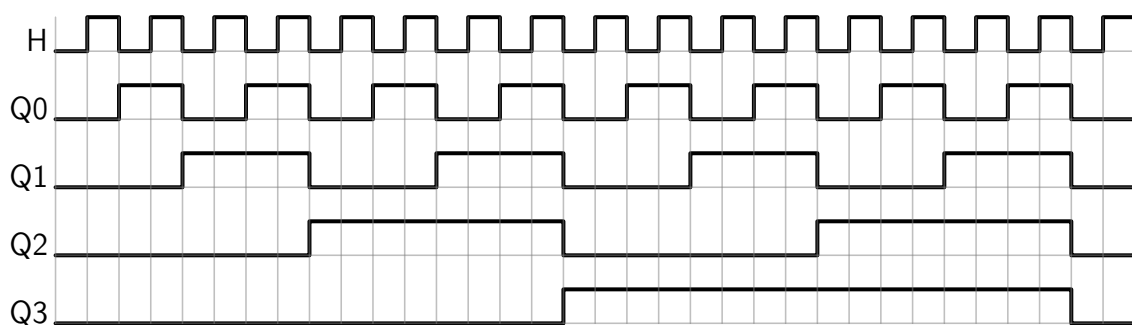
Pour réaliser ce type de compteur, il suffit de mettre en cascade des bascules fonctionnant en mode Toggle (déclenchement), c'est-à-dire des bascules T ou leurs équivalentes (bascules JK dont les entrées J et K sont maintenues à 1, bascules D dont la sortie complémentaire est reliée à l'entrée D ou bascules RS dont la sortie Q est reliée à l'entrée R et la sortie complémentaire est reliée à l'entrée S, voir *réalisation de la bascule T via des bascules JK, D et RS* page 50).

4.5.2.1.1 Compteurs asynchrones à cycle complet

Pour obtenir un compteur à cycle complet, c'est-à-dire un compteur modulo $N = 2^n$, on montera en cascade n bascules dont la sortie Q de chaque bascule est reliée à l'entrée H de la bascule suivante. Ainsi, un compteur binaire modulo 16 nécessite 4 bascules, dont la sortie de la dernière bascule repasse à zéro dès la 16ième impulsion.



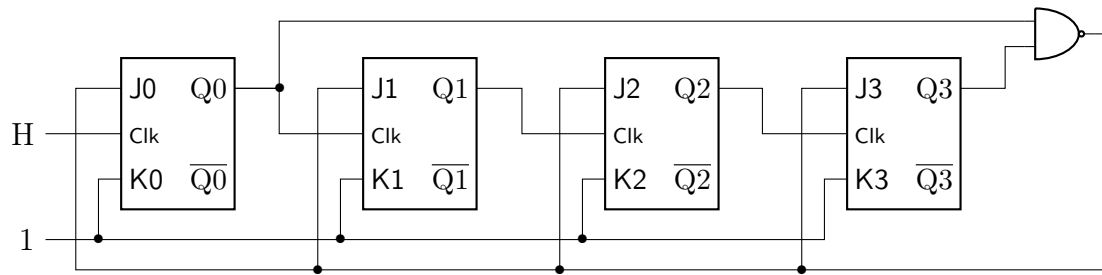
Chaque bascule de ce compteur fonctionne sur le front descendant de la précédente, on aura donc les chronogrammes suivants.



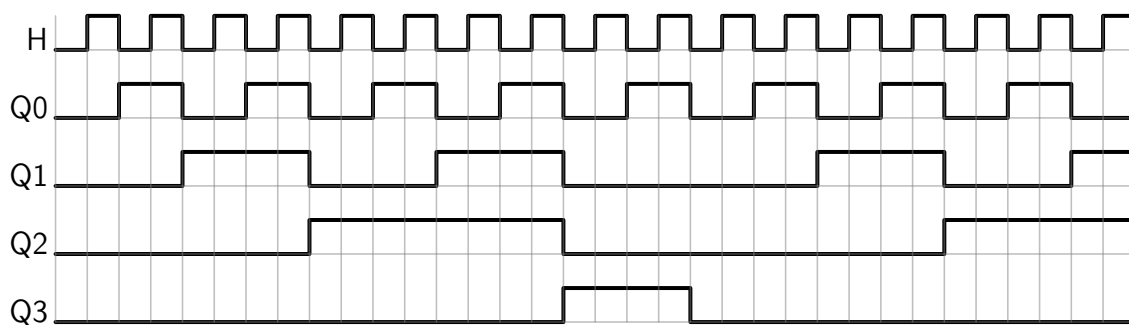
4.5.2.1.2 Compteurs asynchrones à cycle incomplet

Pour réaliser un compteur à cycle incomplet, c'est-à-dire un compteur modulo N où N n'est pas une puissance de 2, on réalise la même structure que précédemment mais il faut d'abord trouver la combinaison appropriée pour forcer le compteur à repasser à zéro avant que le nombre N soit affiché dans les sortie des bascules.

Si l'on désire, par exemple réaliser un compteur modulo 10, il faut utiliser quatre bascules (3 ne permettent de compter que jusqu'à 8) et les remettre toutes à zéro lorsque le nombre 9, soit 1001 en binaire, est affiché en sortie. Pour ce faire, on appliquera un NON ET logique entre la sortie de la première bascule et celle de la dernière bascule dont la sortie est reliée aux entrées J des 4 bascules.



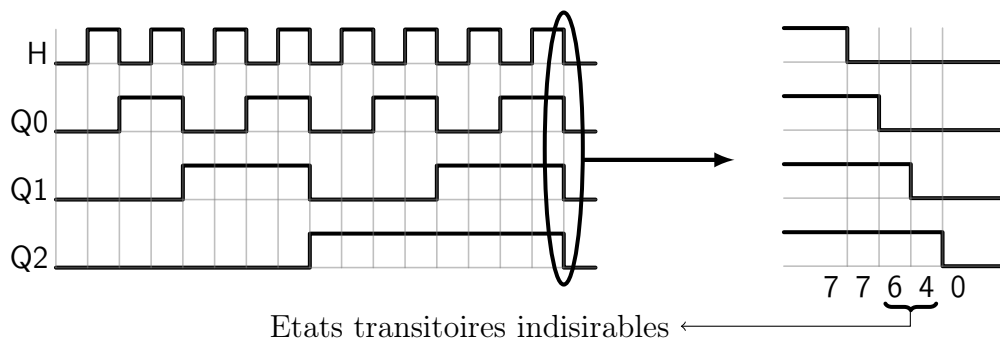
On aura donc les chronogrammes suivants :



Comme le montre les chronogrammes, le compteur est remis à zéro dès la dixième impulsion.

4.5.2.1.3 Inconvénients des compteurs asynchrones

Dans un compteur asynchrone, le signal d'horloge que l'on applique à l'entrée du compteur doit traverser la première bascule avant de pouvoir commander la seconde et ainsi de suite jusqu'à la dernière bascule. Par conséquent, le changement de la valeur du compteur n'est donc pas instantané, celui-ci s'effectue avec un temps de retard appelé temps de propagation (tdp). Ce dernier, dépend essentiellement du nombre de bascules constituant le compteur ainsi que le temps de réponse de chaque bascule (plus le nombre de bascules est grand, plus le temps de propagation est grand). Cela, malheureusement, rend ce type de compteurs limité face aux horloges de hautes fréquences, notamment pour les compteurs de grande taille. De plus, la propagation du signal d'entrée sur les différentes bascules conduira à l'apparition des états transitoires indésirables (voir la figure suivante) qui peuvent poser de vrais problèmes pour les systèmes sensibles. Un autre inconvénient de ce type de compteurs est qu'il n'existe pas de méthode fiable pour réaliser des compteurs à cycle incomplet ou d'autres énumérations que le code binaire naturel.



4.5.2.2 Les compteurs synchrones

Contrairement aux compteurs asynchrones, les compteurs synchrones sont des compteurs dont les bascules qui les constituent sont toutes commandées par le même signal d'horloge. Par conséquent, les bascules de ce type de compteur changent d'état en même temps, ce qui permet d'éliminer le problème des états transitoires indésirables.

Pour réaliser un compteur synchrone, il suffit de chercher les équations des entrées des différentes bascules constituant le compteur, c'est à dire, il faut déterminer comment positionner les entrées des bascules pour que leurs sorties passent d'un nombre (000 par exemple) au nombre suivant (001 pour l'exemple précédent) lors de la prochaine impulsion de l'horloge. Pour ce faire, nous allons suivre la démarche suivante :

1. Dresser la table de l'évolution des sorties du compteur avec les valeurs d'entrée de ses bascules en se basant sur la table réduite de ces dernières.
2. Déterminer les équations simplifiées des entrées des bascules.
3. Dessiner le schéma logique du compteur.

Exemple : Synthèse d'un compteur synchrone modulo 7 en utilisant des bascules JK.

1. Table de l'évolution des sorties du compteur.

Etat du cycle	Sorties			Bascule 2		Bascule 1		Bascule 0	
	Q2	Q1	Q0	J2	K2	J1	K1	J0	K0
0	0	0	0	0	x	0	x	1	x
1	0	0	1	0	x	1	x	x	1
2	0	1	0	0	x	x	0	1	x
3	0	1	1	1	x	x	1	x	1
4	1	0	0	x	0	0	x	1	x
5	1	0	1	x	0	1	x	x	1
6	1	1	0	x	1	x	1	0	x

2. Equations simplifiées des entrées des bascules.

Q ₀	Q ₂ Q ₁			
	00	01	11	10
0	0	0	x	x
1	0	1	x	x

$J2 = Q1 \cdot Q0$

Q ₀	Q ₂ Q ₁			
	00	01	11	10
0	x	x	1	0
1	x	x	x	0

$K2 = Q1$

	Q_2Q_1	00	01	11	10
Q_0	0	0	x	x	0
	1	1	x	x	1

$J1 = Q0$

	Q_2Q_1	00	01	11	10
Q_0	0	x	0	1	x
	1	x	1	x	x

$K1 = Q2 + Q0$

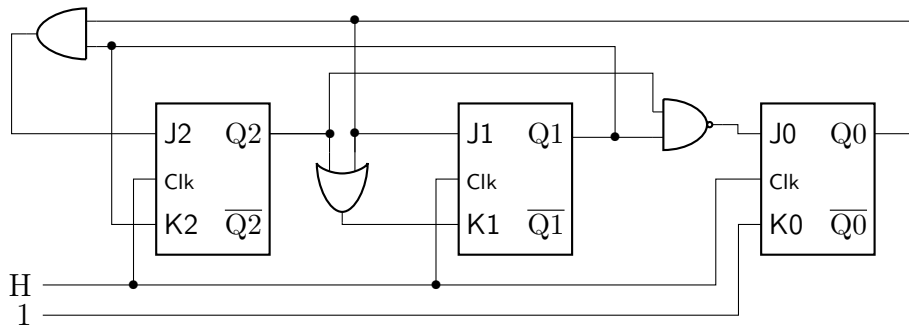
	Q_2Q_1	00	01	11	10
Q_0	0	1	1	0	1
	1	x	x	x	x

$J0 = \overline{Q2} + \overline{Q1} = \overline{Q2.Q1}$

	Q_2Q_1	00	01	11	10
Q_0	0	x	x	x	x
	1	1	1	x	1

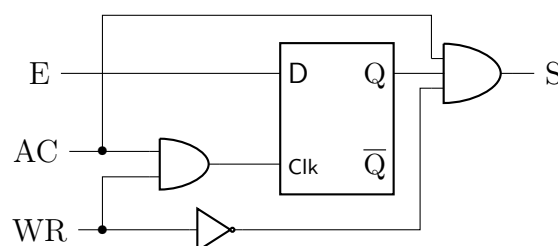
$K0 = 1$

3. Le schéma logique du compteur.

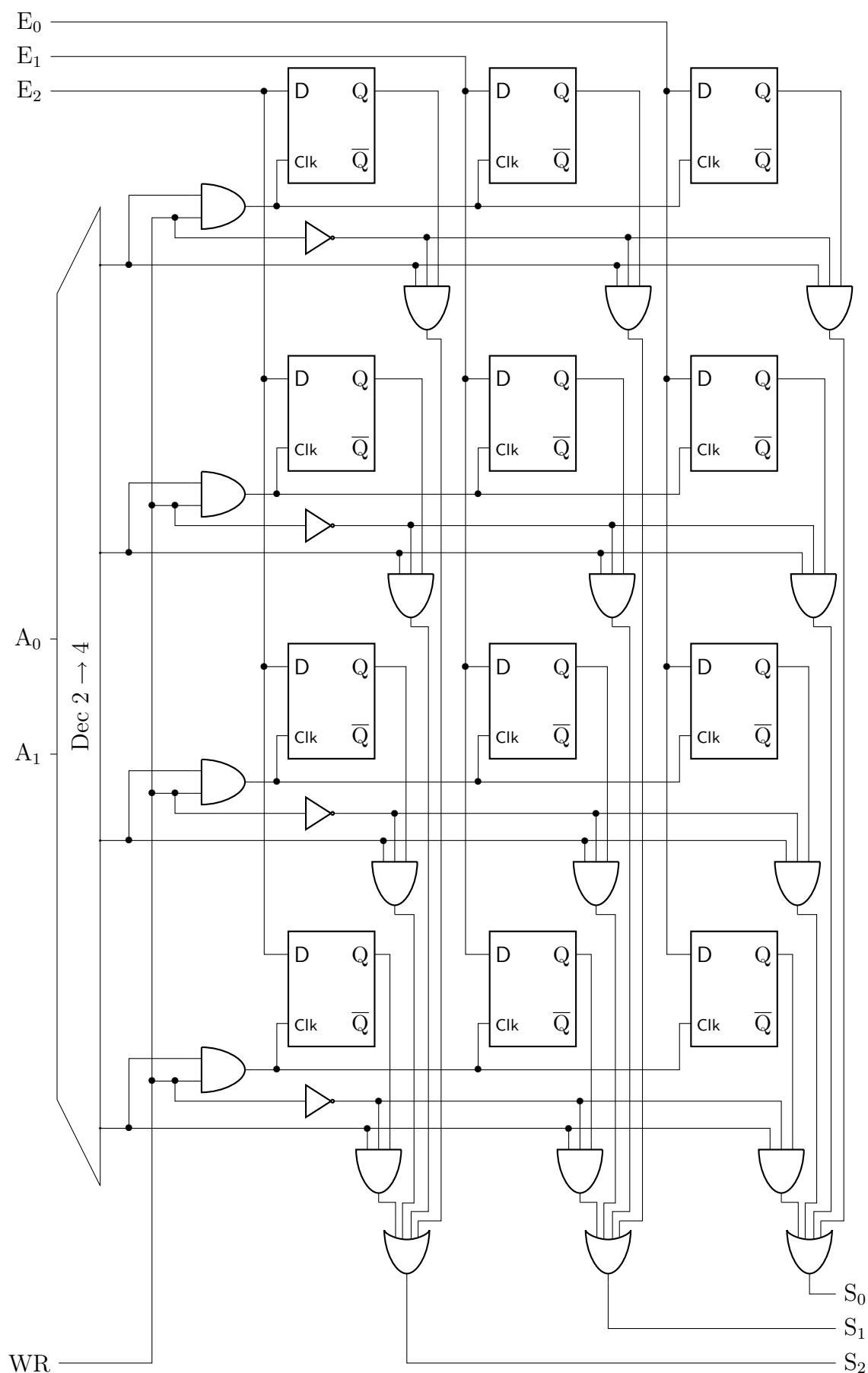


4.5.3 Les mémoires à semi-conducteurs

Une mémoire à semi-conducteur est un ensemble de cases dont chaque case est constituée d'un ensemble de bascules permettant de stocker une information binaire. Dans une telle mémoire, nous pouvons lire ou écrire une case bien précise en activant les bascules qui la constituent et en précisant l'opération à effectuer (lecture/écriture). Pour ce faire, les mémoires disposent généralement de deux entrées de commandes ou de contrôle. D'une part l'entrée d'activation AC (*Activate*) qui permet de mettre les bascules constituant la case en question en mode actif. D'autre part l'entrée de lecture/écriture notée WR (*Write*, WR=1 pour l'écriture et WR=0 pour la lecture) qui permet de préciser le type d'opération à effectuer sur la case active. Les mémoires peuvent également avoir une entrée d'activation du boîtier notée CS (*Chip Select*) qui permet de sélectionner un boîtier mémoire parmi l'ensemble des boîtiers constituant la mémoire. Évidemment, cette entrée de commande est utilisée lorsque plusieurs boîtiers mémoires sont placés en parallèle. Une cellule mémoire va donc avoir la structure suivante :



Ainsi, la réalisation d'une mémoire 4×3 (4 cases de 3 bits chacune) à partir de bascules D est donnée par le schéma logique suivant.



La mémoire de cet exemple dispose de 3 lignes d'entrée ($E_2E_1E_0$) contenant la valeur binaire à écrire dans la case sélectionnée, 3 lignes de sorties ($S_2S_1S_0$) contenant la valeur binaire lue à partir des bascules de la case sélectionnée, 2 lignes d'entrée d'adresse (A_1A_0) contenant l'adresse de la case à sélectionner et une entrée de commande WR permettant de préciser le type d'opération à réaliser (lecture ou écriture). Grace au décodeur lié aux entrées d'adresse, une seule case sera active à la fois, il s'agit de la case liée à la sortie du décodeur correspondant à la valeur fournie par les entrées d'adresse (A_1A_0).

Chapitre 5

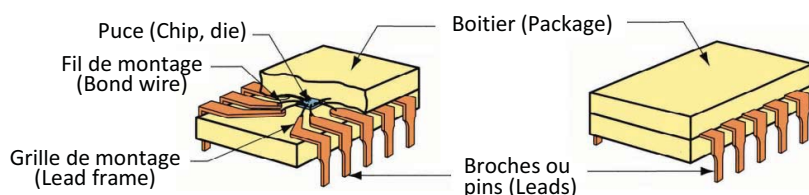
Circuits intégrés

5.1 Introduction

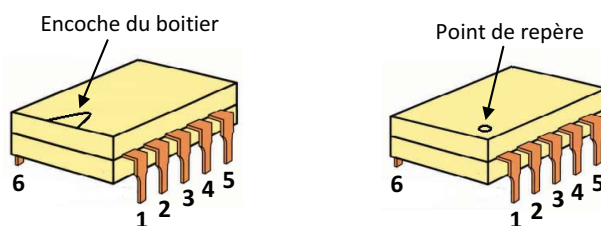
Les circuits combinatoires ainsi que les circuits séquentiels, tels que ceux que l'on a étudiés dans les chapitres précédents, sont construits sur la base d'un ensemble de composants numériques appelés circuits intégrés, dont chacun contient un nombre limité de portes logiques. Dans ce chapitre, nous allons voir les principales notions liées aux circuits intégrés numériques, leur démarche de fabrication, ainsi que les différentes familles de circuits intégrés standards existantes.

5.2 Définition

Un circuit intégré est un composant électronique regroupant plusieurs types de composants électroniques de base (résistances, diodes et transistors.) dans une plaquette très réduite de silicium. Cette dernière est protégée par un boîtier de plastique ou de céramique où sortent des broches qui serviront de connexion (pins), comme le montre la figure suivante.



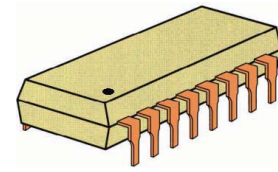
Les broches ou les connexions des circuits intégrés sont généralement numérotées dans le sens trigonométrique à partir de l'encoche du boîtier ou du point de repère, comme le montre la figure suivante.



Les boîtiers encapsulant les circuits intégrés sont de différents types. On peut distinguer cinq types différents, selon le nombre et la position de broches externes, ainsi que la méthode de montage sur la carte de circuits.

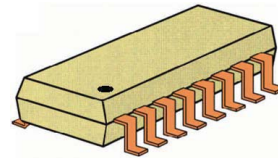
1. Les boîtiers DIL ou DIP (Dual In Line Package)

Ce sont les boîtiers les plus fréquemment rencontrés aujourd'hui. Ils se distinguent par leur forme rectangulaire avec un double rangé de connexions (broches) droites. Leur montage est effectué en soudant leurs broches dans des trous réalisés dans la carte imprimée.



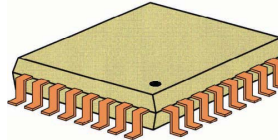
2. Les boîtiers SOP (Small Outline Package)

Ce sont des boîtiers semblables aux boîtiers DIP dont l'espace entre deux connexions est divisé par 2 et dont les connexions ne sont plus droites mais coudées afin de permettre une soudure sans trous.



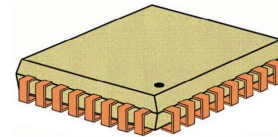
3. Les boîtiers QFP (Quad Flat Package)

Ce sont des boîtiers carrés ou rectangulaires (en plastique ou en céramique) dotés de broches de connexion sur leurs 4 côtés (jusqu'à 200 broches). Les broches sont généralement extérieures au boîtier et sont semblables à celles des boîtiers SOP (coudées).



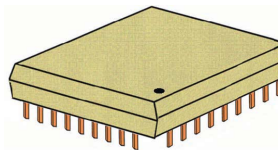
4. Les boîtiers CCP (Chip Carrier Package)

Ce sont des boîtiers semblables aux boîtiers QFP dont les connexions ne sont plus tendues vers l'extérieur mais encastrées sur leurs 4 côtés afin de permettre un montage facile sans soudure (boîtiers enfichables dans la carte imprimée via des supports spéciaux).



5. Les boîtiers PGA (Pin Grid Array)

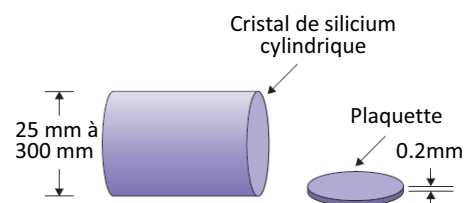
Ce sont des boîtiers carrés ou rectangulaires dont les connexions (broches) sont agencées selon un réseau régulier sur la face inférieure du boîtier, ce qui leur permet de supporter un très grand nombre de connexions. Ce type de boîtiers est généralement facile à monter dans la carte imprimée en les plaçant dans des supports spéciaux à trous traversant.



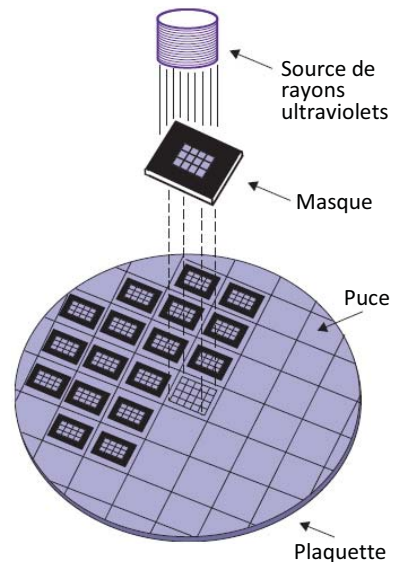
5.3 Construction des circuits intégrés

La construction de circuits intégrés est l'un des processus de production les plus exigeants et les plus compliqués. Elle passe généralement par les étapes suivantes.

1. Construction d'un monocristal de silicium pur (sans impuretés) sous la forme d'un cylindre d'un diamètre pouvant aller jusqu'à 300 mm et découpage de celui-ci en tranches minces appelées plaquettes (wafers), d'une épaisseur approximative de 0,2 mm.

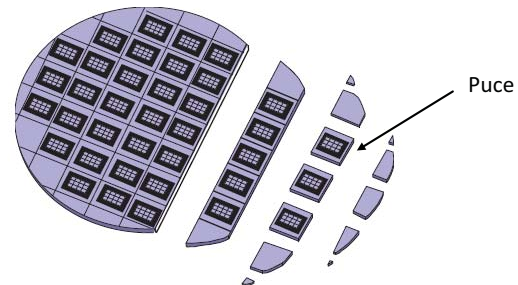


2. Implantation sur chaque tranche (plaquette) de silicium une centaine de puces (chips) représentant chacune un circuit intégré contenant des centaines de milliers, voire des millions de transistors. Pour ce faire, un processus de la lithographie optique est appliqué, dans lequel des rayons ultraviolets passent à travers un masque portant des motifs formés par des zones transparentes ou opaques aux fréquences ultraviolettes et l'image résultante est projetée sur la surface de la tranche. Une fois que la zone correspondant à un circuit intégré a été exposée, la plaquette est déplacée et le processus est répété jusqu'à ce que le même motif ait été reproduit sur toute la surface de la plaquette.

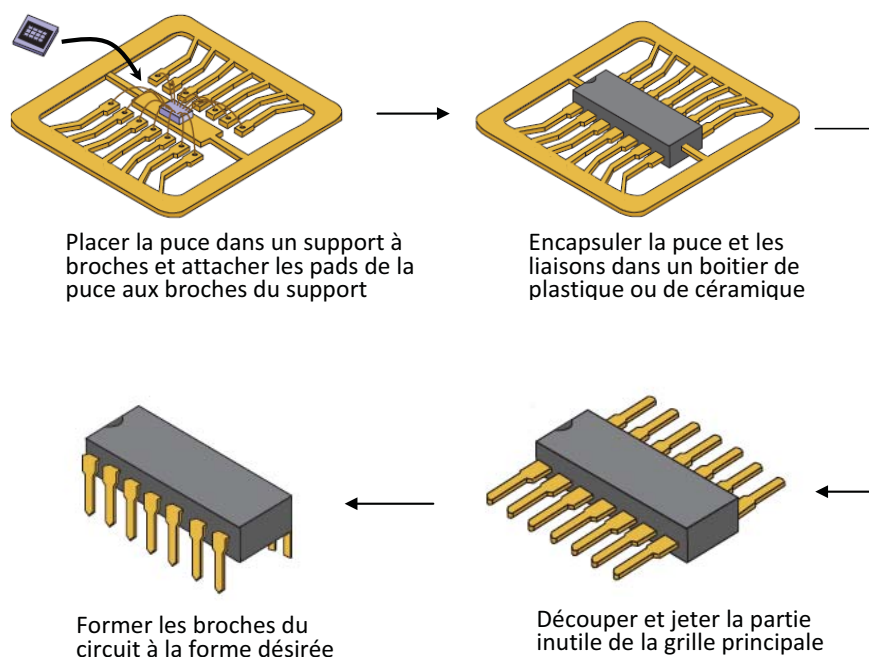


3. Traitement de la surface de la plaquette pour construire les transistors. C'est la phase la plus délicate, elle nécessite de 20 à 40 traitements chimiques successifs pour construire les transistors nécessaires au fonctionnement des circuits implantés.

4. Découpage des plaquettes en puces et test de celles-ci afin d'éliminer celles qui ne répondent pas bien aux spécifications. Plus le circuit réalisé est complexe, plus sa taille augmente et plus le nombre de circuits défectueux est grand.



5. Insertion des puces dans des boîtiers en plastique ou en céramique.



5.4 Familles des circuits intégrés

Selon le type de transistors utilisés, les circuits intégrés se divisent en deux grandes familles :

5.4.1 Circuits intégrés TTL (Transistor Transistor Logique)

La famille TTL (Transistor Transistor Logic), construite autour de transistors bipolaires, est la famille des circuits intégrés la plus utilisée et la plus populaire à cause de son coût faible, sa vitesse élevée, sa possibilité d'interconnexion, son immunité acceptable aux bruits, ainsi que la disponibilité de centaines de fonctions différentes introduites sous forme de circuits intégrés référencés, dont la référence commence, selon la série du circuit, par :

- **74xx** pour les circuits TTL Standards.
- **74Lxx** pour les circuits *Low power TTL* (circuits TTL à faible consommation).
- **74Sxx** pour les circuits *Schottky TTL* (circuits TTL réalisés avec des transistors schottky).
- **74LSxx** pour les circuits *Low power Schottky TTL* (circuits TTL à faible consommation réalisés avec des transistors schottky).
- **74ASxx** pour les circuits *Advanced Schottky TTL* (circuits TTL réalisés avec des transistors schottky avancés).
- **74ALSxx** pour les circuits *Advanced Low power Schottky TTL* (circuits TTL à faible consommation réalisés avec des transistors schottky avancés).
- **74Fxx** pour les circuits *Fast TTL* (circuits TTL rapides).

5.4.2 Circuits intégrés CMOS (Complementary Metal Oxide Semi-conductor)

La famille CMOS (Complementary Metal Oxyde Semiconductor), construite autour de transistors unipolaires, est une famille relativement récente qui a commencé à s'imposer de plus en plus dans le marché des circuits intégrés à cause de son procédé de fabrication qui est plus simple que celui des circuits TTL et de sa densité d'intégration qui est plus élevée, ainsi que de sa consommation électrique qui est très faible par rapport à celle des circuits TTL (Les CMOS ne consomment qu'une fraction de l'énergie dissipée par la série TTL faible consommation). Tout comme pour la famille TTL, la famille CMOS dispose de plusieurs fonctions différentes introduites sous forme de circuits intégrés de référence commençant par :

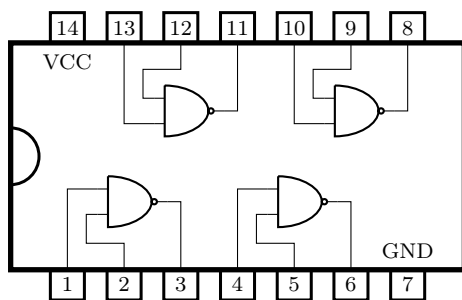
- **4xxx** pour les circuits *CMOS* classiques.
- **74Cxx** pour les circuits *CMOS* compatibles avec leurs homologues de la famille TTL de même numéro, en termes de fonction réalisée et de brochage.
- **74HCxx** pour les circuits *High speed CMOS* (CMOS rapide), c'est une amélioration de la série 74Cxx en termes de vitesse de commutation (la série 74HCxx est dix fois plus rapide que la série 74Cxx). La vitesse de cette série est comparable à celle de la série TTL 74LSxx.
- **74HCTxx** pour les circuits *High speed CMOS* totalement compatibles avec leurs homologues de la famille TTL de même numéro, y compris en termes de tension.

- **74ACxx** pour les circuits *Advanced CMOS* qui sont encore plus rapides que la série 74HCxx.
- **74ACTxx** pour les circuits *Advanced CMOS* totalement compatibles avec leurs homologues de la famille TTL de même numéro, y compris en termes de tension.

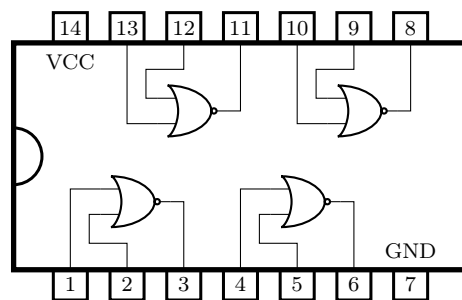
5.5 Circuits intégrés TTL des portes logiques de base

On s'intéresse ici uniquement aux circuits intégrés qui contiennent quatre portes logiques indépendantes à 2 entrées chacune, dont la première porte a pour entrées les broches 1 et 2, et pour sortie la broche 3, la deuxième a pour entrée les broches 4 et 5, et pour sortie la broche 6, la troisième a pour entrée les broches 9 et 10, et pour sortie la broche 11 et la quatrième porte a pour entrées les broches 12 et 13, et pour sorties la broche 11. Les broches 7 et 14 sont réservées pour l'alimentation du circuit, le premier est relié à la masse (côté négatif GND) alors que le deuxième est relié au potentiel le plus haut (côté positif VCC).

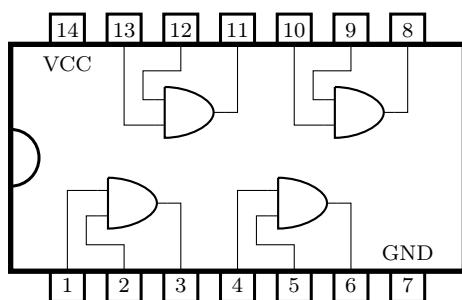
1. Circuit intégré NAND 7400



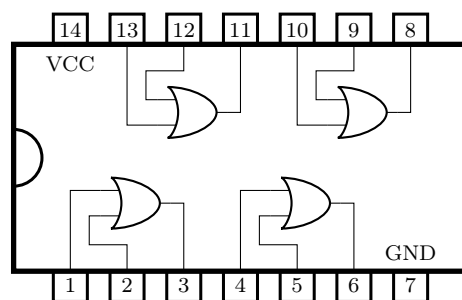
2. Circuit intégré NOR 7402



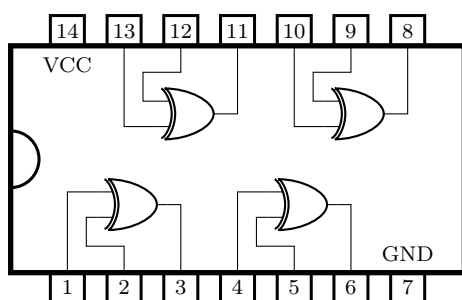
3. Circuit intégré AND 7408



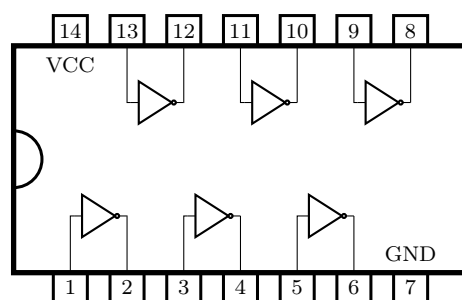
4. Circuit intégré OR 7432



5. Circuit intégré XOR 7486



6. Circuit intégré NOT 7404

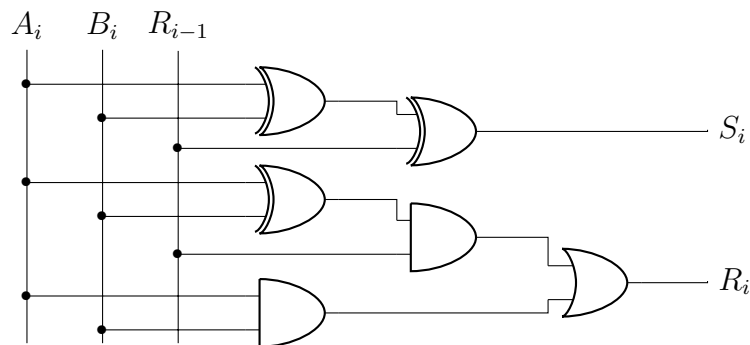


5.6 Montage d'un circuit combinatoire via des circuits intégrés

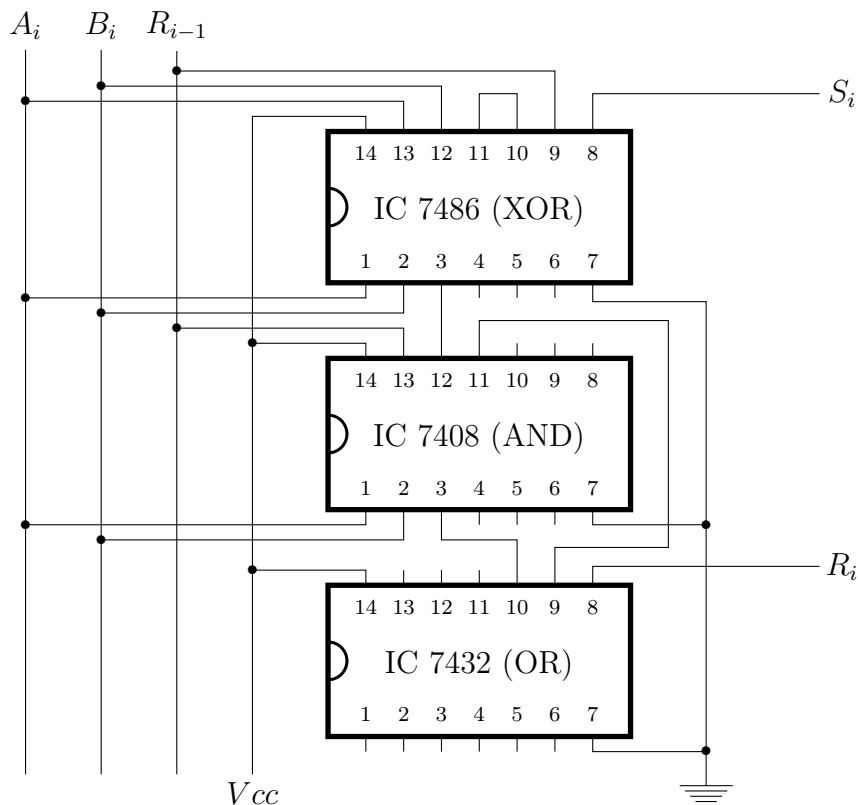
En fait, nous pouvons toujours réaliser n'importe quel circuit combinatoire ou séquentiel en se basant sur le montage d'un ou plusieurs circuits intégrés. Pour ce faire, nous suivrons les mêmes étapes que celles de la synthèse des circuits combinatoires ou séquentiels que nous avons déjà vu dans les chapitre précédents jusqu'à la dernière étape où nous allons remplacer les portes logiques de base par des circuits intégrés correspondants.

Exemple : Montage d'un additionneur complet via des circuits intégrés TTL.

En suivant les étapes de la synthèse d'un additionneur complet (voir la page 31), nous aurons le logigramme suivant :



La réalisation d'un additionneur complet nécessite donc trois portes XOR, deux portes AND et une porte OR. Nous devons donc utiliser 3 circuits intégrés : un circuit TTL 7486 pour les portes XOR, un circuit TTL 7408 pour les portes AND et un circuit TTL 7432 pour la porte OR. Le montage de ce circuit sera donc donné comme suit :



Bibliographie

- [Mercouroff, 1990] Wladimir Mercouroff. *Architecture matérielle et logicielle des ordinateurs et des microprocesseurs*. Armand Colin, 1990. ISBN : 2-200-42007-2. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :002/07.**
- [Ristori, 1991] Joel Ristori, Lucien Ungaro. *Cours d'architecture des ordinateurs*. Eyrolles, 1991. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :002/17.**
- [Goupille, 1993] Pierre-Alain Goupille. *Technologie des ordinateurs pour les I.U.T et B.T.S informatique avec exercices*. Masson, 1993. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :002/01.**
- [Poinsot, 1994] André Poinsot. *Problèmes d'électronique logique*. Masson, 1994. ISSN : 2225844518. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :621/270.**
- [Zanella, 1998] Paolo Zanella, Yves Ligier. *Architecture et technologie des ordinateurs*. Dunod, 1998. ISBN : 210003801X. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :004/258.**
- [Zerkaoui, 2006] Habiba Drias-Zerkaoui. *Introduction à l'architecture des ordinateurs*. Office des publications universitaires, 2006. **Disponible dans la Bibliothèque centrale, Université de Jijel, Cote :002/08.**
- [Prosser, 1987] Franklin P. Prosser and David E. Winkel. *The art of digital design, An Introduction to Top-Down Design, second edition*. Prentice-Hall, 1987. ISBN 0-13-046780-4.
- [Nelson, 1995] Victor Peter Nelson, Bill D. Carroll, H. Troy Nagle, David Irwin. *Digital Logic Circuit Analysis and Design*. Prentice-Hall, 1995. ISBN : 0-13-463894-8.
- [Roth, 2010] Charles H. Roth, Jr. and Larry L. Kinney. *Fundamentals of Logic Design, Sixth Edition*. Cengage Learning, 2010. ISBN-10 : 0-495-66804-4, ISBN-13 : 978-0-495-66804-6.
- [Mano, 2015] M. Morris Mano, Charles R. Kime and Tom Martin. *Logic and Computer Design Fundamentals, Fifth Edition*. Pearson Higher Education, 2015. ISBN-10 : 0-13-376063-4, ISBN-13 : 978-0-13-376063-7.
- [Floyd, 2015] Thomas L. Floyd. *Digital Fundamentals, 11th Edition*. Pearson Higher Education, 2015. ISBN-10 : 9-35-394249-7, ISBN-13 : 978-9-35-394249-6.