Cours d'électronique numérique

Camille Diou, Maître de Conférences
Laboratoire Interfaces Capteurs et Microélectronique
Université Paul Verlaine–Metz

Format A5 – Version du 24 février 2009

Notes sur cet ouvrage

Ce document est à la date d'aujourd'hui (24 février 2009) toujours en phase d'écriture. Il est donc nécessairement incomplet et peut même encore comporter des erreurs qui n'auraient pas été détectées.

Ce document doit notamment s'enrichir à l'avenir des points suivants (dans le désordre) :

- la logique mixte
- compléter la simplification des fonctions logiques
  - méthode de Quine/McCluskey → *fait*
  - *ajouté : méthode de Petrick*
  - *ajouté : algorithme Espresso* (à détailler)
  - diagrammes de Venn, Johnston et Caroll
  - familles logiques et spécifications électriques
- étude des systèmes programmables évolués (en complément du chapitre actuel)
- synthèse des systèmes séquentiels synchrones
  - machines d'états (Moore, Huffman, Mealey)
- synthèse des systèmes séquentiels asynchrones
- arithmétique binaire et opérateurs arithmétiques
- compléter les exercices et corrigés

Ce document a été réalisé à l'aide des logiciels TEX et LATEX sous les environnements TEXLive, TeTEX puis MiKTEX. Les diagrammes sont réalisés à l'aide de XY-pic. Une partie des schémas électronique est réalisée à l'aide du paquetage CIRC.

La police utilisée pour le texte principal est Fourier.

Les descriptions bibliographiques/historiques présentes dans les entêtes de chapitres sont composées en DayRoman comportant les ligatures « c t » (ct) et « s t » (st).

Quant à l'extrait du texte de Blaise Pascal en préambule du chapitre 2, il est également composé dans la police DayRoman, mais dotée – notamment – du « s long » (ſ) et des ligatures « ſs » (ß). Les ligatures alternatives « ſſ » (ſſ) et « ffi » (ffi) ne sont pas utilisées dans ce document).

Table des matières

Première partie

Les nombres

Chapitre 1

Les systèmes de numération

Gottfried Wilhelm von Leibnitz
⋆ jui. 1646, Allemagne
† 1716

Ce philosophe d'origine Allemande est l'inventeur d'une machine permettant de calculer directement les 4 opérations de base. Il est aussi celui qui a introduit la notion de binaire en Occident.

1.1 La représentation polynomiale

Si nous manipulons les nombres de manière intuitive, c'est la plupart du temps dans la base décimale, naturelle et universelle. Mais cela ne doit pas masquer la nature même de la numération qui peut prendre plusieurs formes, parmi lesquelles on trouve la théorie des ensembles et la représentation polynomiale. La représentation polynomiale d'un nombre est sa représentation sous la forme suivante, où b est appelée la base :

$$a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_2b^2 + a_1b + a_0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots + a_{-m}b^{-m}$$

Si la base 10 nous est familière, d'autres bases existent et les bases les plus utilisées en informatique sont les bases 10, 2, 8 et 16 appelées respectivement « décimale », « binaire », « octale » et « hexadécimale ».

► **Remarque 1.1**

- *Toute base s'écrit* 10 *(un-zéro) dans son propre système de numérotation*
- *Un décalage à gauche multiplie un nombre par sa base*
- *Un décalage à droite divise un nombre par sa base*

1.2 Le système binaire

1.2.1 Introduction

Le système décimal est malheureusement difficile à adapter aux mécanismes numériques, car il est difficile de concevoir du matériel électronique fonctionnant sur dix plages de tensions différentes.
On lui préférera donc le système binaire :
– base B=2 ;
– 2 symboles : $\{0,1\}$ appelés « éléments binaires » ou « bits » (bit=*Binary digIT*) ;
– le système binaire est pondéré par 2 : les poids sont des puissances de 2 ;

▷ **Exemple 1.1**

2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-3}	2^{-3}
1	0	1	1	0	0	1	,	0	1	1

– les différentes puissances de 2 sont :

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

– un ensemble de 8 bits est appelé « octet » (ou *byte*).

1.2.2 Comptage binaire

On présente les nombres binaires en général avec un nombre fixe de bits, nombre imposé par les circuits mémoires utilisés pour représenter ces nombres.
Suite des nombres binaires à 4 bits :

Poids :	2^3	2^2	2^1	2^0	B10
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9

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

Le bit le plus significatif – le bit le plus à gauche – est appelé « bit de poids fort » ou MSB (*Most Significant Bit*). Le bit le moins significatif – le bit le plus à droite – est appelé « bit de poids faible » ou LSB (*Less Significant Bit*).
Si on utilise N bits, on peut représenter 2^N valeurs différentes de 2^0 à 2^{N-1}

▷ **Exemple 1.2**

$N = 8 : 00000000 \rightarrow 11111111 \leftrightarrow 255$

► **Remarque 1.2**

Comme l'on traite souvent en micro-informatique de nombres à 8 ou 16 éléments binaires (e.b.), on se sert des systèmes :

- *octal : à base* 8 *;*
- *hexadécimal : à base* 16.

1.3 Le système octal

- base B=8 ;
- 8 symboles : $\{0,1,2,3,4,5,6,7\}$;

L'intérêt de ce système est que la base 8 est une puissance de 2 ($8 = 2^3$), donc les poids sont aussi des puissances de 2. Chaque symbole de la base 8 est exprimé sur 3 éléments binaires : $(a_i)_8 = b_{i_2} b_{i_1} b_{i_0}$

▷ **Exemple 1.3**

$(52,3)_8 = 101\,010,011$

1.4 Le système hexadécimal

- base B=16 ;
- 15 symboles : $\{0,1,2,\ldots,9,A,B,C,D,E,F\}$ appelés « digits » ;
- chaque symbole est exprimé en binaire sur 4 bits ;

▷ **Exemple 1.4**

$(F3D,2)_{16} = 1111\,0111\,1101,0010$

1.5 Conversion d'un système de numération à un autre

1.5.1 Base B *vers base 10*

$(a_n \dots a_0)_B = a_n B^n + \dots + a_0 B^0 = (a'_m \dots a'_0)_{10}$

▷ **Exemple 1.5**

$(1001,1)_2 = 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0 + 1.2^{-1} = 8 + 0 + 0 + 1 + 0,5 = 9,5$
$(A12)_{16} = A.16^2 + 1.16^1 + 2.16^0 = 2560 + 16 + 2 = 2578$

1.5.2 Base 10 *vers base* B

1.5.2.a Première méthode, dite « à soustractions successives »

Elle consiste à soustraire successivement la plus grande puissance de B.

▷ **Exemple 1.6**

$$\left.\begin{array}{rcrcr} 100 & = & 1.2^6 & + & 36 \\ 36 & = & 1.2^5 & + & 4 \\ 4 & = & 1.2^2 & + & 0 \end{array}\right\} \rightarrow (100)_{10} = (1100100)_2$$

1.5.2.b Deuxième méthode, à division par B

Elle consiste à diviser par B autant de fois que cela est nécessaire pour obtenir un quotient nul. Ensuite on écrit les restes dans l'ordre inverse de celui dans lequel ils ont été obtenus.
Pour la partie fractionnaire on multiplie par B jusqu'à obtenir un résultat nul ou la précision souhaitée.

▷ **Exemple 1.7**

$(20,4)_{10} = (?)_2$
Partie entière :

20	2							
0	10	2						
	0	5	2					
		1	2	2				
			0	1	2			
				1	0			

Partie fractionnaire :

$$\begin{array}{rl} & 0,4 \quad \times \\ & \quad 2 \\ \hline \mathbf{0} & 0,8 \end{array} \rightarrow \begin{array}{rl} & 0,8 \quad \times \\ & \quad 2 \\ \hline \mathbf{1} & 1,6 \end{array} \rightarrow \begin{array}{rl} & 0,6 \quad \times \\ & \quad 2 \\ \hline \mathbf{1} & 1,2 \end{array}$$

Le résultat est donc 10100,0110.

1.5.3 Base 2^n vers base 2

Chaque symbole de la base $B = 2^n$ peut être représenté par n éléments binaires.

▷ **Exemple 1.8**

$(3A9)_{16} = (0011\ 1010\ 1001)_2$
$(742,5)_8 = (111\ 100\ 010,101)_2$

1.5.4 Base 2 vers base 2^n

Il suffit de regrouper les éléments binaires par paquets de n.

▷ **Exemple 1.9**

$$\begin{array}{rcl} (1011011)_2 & = & (\underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{011}_{3})_2 = (133)_8 \\ & = & (\underbrace{0101}_{5}\ \underbrace{1011}_{B})_2 = (5B)_{16} \end{array}$$

1.5.5 Base i *vers base* j

– si i et j sont des puissances de 2, on utilise la base 2 comme relais ;

▷ **Exemple 1.10**

base 8 → base 2 → base 16

– sinon, on utilise la base 10 comme relais.

▷ **Exemple 1.11**

base 5 → base 10 → base 2

Les nombres

▶Exercice 1.1

Convertir en binaire, octal et hexadécimal les nombres décimaux suivants :
43 ; 154 ; 25740

▶Exercice 1.2

Convertir en décimal et hexadécimal les nombres suivants :
$(1101110)_2$; $(75)_8$; $(587)_8$

▶Exercice 1.3

Convertir en binaire et hexadécimal les nombres suivants :
$(166,25)_{10}$; $(126,34)_8$; $(231,1)_4$

▶Exercice 1.4

Convertir en binaire le nombre décimal suivant : 24537

▶Exercice 1.5

Convertir en décimal les nombres suivants :
$(D9,4)_{\mathrm{H}}$; $(576)_8$

m Chapitre 2 n

Codage des nombres dans les machines numériques

Blaise Pascal
? 19 juin 1623, Clermont, France
† 19 août 1662, Paris, France

Ami le c eur, cet avertiÿement fervira pour te faire favoir que j'expo fe au public une petite machine de mon invention, par le moyen de laquelle feul tu pourras, fans peine quelconque, faire toutes les opérations de l'arithmétique, et te foulager du travail qui t'a fouvent fatigué l'efprit, lor fque tu as opéré par le jeton ou par la plume : je puis, fans préfomption, e fpérer qu'elle ne te déplaira pas, après que Monfeigneur le Chancelier l'a honorée de fon eftime, et que, dans Paris, ceux quifont les mieux verfés aux mathématiques ne l'ont pas jugée indigne de leur approbation. Néanmoins, pour ne pas paraître négligent à lui faire acquérir auÿi la tienne, j'ai cru être obligé de t'éclairer fur toutes les diHcultés que j'ai eftimées capables de choquer ton fens lorfque tu prendras la peine de la confidérer.

(Blaife Pafcal, Avis néceÿaire à ceux qui auront curio fité de voir la machine d'arithmétique, et de f'en fervir, 1645).

Les systèmes logiques sont constitués de mécanismes qui ne permettent de noter que 2 états : « 0 » ou « 1 ». Une mémoire élémentaire est donc une unité contenant « 0 » ou « 1 ». Plusieurs de ces unités sont assemblées pour représenter un nombre binaire.

B Exemple 2.1

mémoire 8 bits :

Ces mémoires sont indissociables et l'ordre d'assemblage donne le poids de chaque bit.

2.2.2 Représentation en complément restreint (CR) ou complément à 1 (C1)

$-A = \overline{A}$: pour prendre l'inverse d'un nombre, il suffit de le complémenter (inversion de tous ses bits). Comme dans le cas précédent, la nature du premier bit donnera le signe : $0 \leftrightarrow +$ et $1 \leftrightarrow -$.

▷ **Exemple 2.3**

$$\text{avec 4 bits : } \begin{cases} +5 & \rightarrow & 0101 \\ -5 & \rightarrow & 1010 \end{cases}$$

Problème : de nouveau, on a deux représentations différentes pour le zéro.

2.2.3 Représentation en complément vrai (CV) ou complément à 2 (C2)

C'est la représentation la plus utilisée. Le bit le plus à gauche est encore le bit de signe : $0 \leftrightarrow +$ et $1 \leftrightarrow -$.

$$\begin{array}{rcl} -A & = & \overline{A}+1 \\ \overline{A} & = & \overline{a_{n-1}}\,\overline{a_{n-2}} \cdots \overline{a_0}\ ^{1} \\ A+\overline{A} & = & 1\,1 \ldots 1 \\ 1+\overline{A}+A & = & \mathbf{1} \leftarrow \underbrace{0\,0 \ldots 0}_{0}\ ^{2} \end{array}$$

$\Rightarrow -A = \overline{A}+1$ est appelé complément à 2

► **Remarque 2.1**

- *pour passer d'une valeur négative à une valeur positive, on applique aussi le complément à 2 ;*
- *une seule représentation pour le zéro ;*
- *avec des mots de n éléments binaires, on obtient 2^n valeurs différentes, de 0 à $2^{n-1}-1$ pour les valeurs positives, et de -1 à -2^{n-1} pour les valeurs négatives ;*

▷ **Exemple 2.4**

$$n = 8 \Rightarrow \begin{cases} \text{nb} > 0 \text{ de } 0 \text{ à } 127 \\ \text{nb} < 0 \text{ de } -1 \text{ à } -128 \end{cases}$$

1. on complémente chaque coefficient
2. car on représente sur n bits seulement

– $nb \geq 0 \rightarrow$ *bit de signe* $= 0$ $nb < 0 \rightarrow$ *bit de signe* $= 1$
– *pour représenter un nombre positif sur une mémoire de taille donnée, on complète les cases vides de gauche par des* 0 *; pour représenter un nombre négatif sur une mémoire de taille donnée, on complète les cases vides de gauche par des* 1 *;*

▷ **Exemple 2.5**

+13 sur 8 bits : 00001101, −13 sur 8 bits : 11110011

► **Remarque 2.2**

*La représentation en complément vrai est en fait une écriture polynomiale (donc pondérée) du nombre, de même que le code binaire naturel, à ceci près que le bit de poids fort à un poids négatif (*cf. *§ 3.1.2 page 38) :*
$-a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_2b^2 + a_1b + a_0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots + a_{-m}b^{-m}$

2.2.4 Représentation en code relatif à 2^{n-1}

Les nombres x sont représentés par $2^{n-1} + x$.

On constate ici que le bit de signe est inversé par rapport aux représentations précédentes : ce code est en fait identique au codage en complément à 2 avec le bit de signe complémenté.

On calcul l'inverse d'un nombre en relatif à 2^{n-1} comme en complément à 2 en complémentant le nombre puis en ajoutant 1.

2.3 Représentation des nombres réels dans un calculateur

Dans un calculateur, un nombre est toujours écrit sous forme d'un bloc de n éléments binaires (considéré comme un entier N). Pour représenter les nombres fractionnaires il est nécessaire de définir la position de la virgule. Pour ce faire, il existe deux méthodes :
– la représentation en virgule fixe ;
– la représentation en virgule flottante.

2.3.1 La représentation en virgule fixe

On décide que la virgule est toujours à une position donnée (un entier peut être représentatif d'un nombre fractionnaire si on connaît la place de la virgule).

▷ **Exemple 2.6**

Virgule au rang K (K chiffres après la virgule) :

La valeur N écrite en mémoire aura les poids suivants :

$N = 2^{N-1-K} \cdots 2^0\, 2^{-1}\, 2^{-K}$ $\qquad 0 \leq N \leq (2^n - 1)2^{-K}$

Virgule au rang 0 :

$N = 2^{N-1} \cdots 2^0$ $\qquad 0 \leq N \leq 2^N - 1$

Inconvénient de la méthode :

– problème de gestion de la virgule notamment dans les multiplications (pour les additions et soustractions pas de problème, la position de la virgule ne change pas) ;

▷ **Exemple 2.7**

Si on décide de définir 2 symboles pour les parties entières et 2 symboles pour les parties fractionnaires, on ne peut plus écrire 256,1.

– utilisation limitée lorsqu'on traite des données de grandeurs différentes, car on doit prendre un grand nombre de bits de part et d'autre de la virgule pour pouvoir représenter des grandeurs très faibles et des grandeurs très importantes.

2.3.2 La représentation en virgule flottante simplifiée

2.3.2.a Introduction [WebMul]

Il arrive dans de nombreux domaines que l'intervalle des valeurs numériques pertinentes soit particulièrement étendu. L'astronomie en est un exemple extrême puisque certains calculs peuvent faire intervenir simultanément la masse du soleil (2.10^{30} kg) et la masse de l'électron (9.10^{-31}kg). Ces deux nombres diffèrent de plus de 60 ordres de grandeur (10^{60}) !

Des calculs faisant intervenir ces nombres pourraient s'effectuer en précision multiple, avec par exemple des nombres de 62 chiffres. Tous les opérandes et tous les résultats seraient représentés par des nombres de 62 chiffres. Cependant, la masse du soleil n'est connue qu'avec une précision de 5 chiffres, et il n'y a en physique pratiquement aucune mesure que l'on puisse réaliser avec une précision de 62 chiffres. Une solution serait alors d'effectuer les calculs avec

une précision de 62 chiffres et de laisser tomber 50 ou 60 d'entre eux avant d'annoncer les résultats, mais ceci est coûteux à la fois en espace mémoire et en temps de calcul.

En fait, ce qu'il faut est un système permettant de représenter des nombres, tel que la taille de l'intervalle des nombres "exprimables" soit indépendante du nombre de chiffres significatifs.

2.3.2.b Principe de la représentation en virgule flottante

Le nombre N est représenté sous la forme :

exposant	mantisse

1ère approche :

Soit $N = a_3\, a_2\, a_1\, a_0\,,\ a_{-1}\, a_{-2}\, a_{-3}$: N peut se noter :

$$(\underbrace{a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0}_{\text{mantisse}}).\underbrace{2^{-3}}_{\text{exp}}$$

$$\Rightarrow \begin{cases} \text{exposant} = & -3 \\ \text{mantisse} = & a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0 \end{cases}$$

Les valeurs de la mantisse et l'exposant seront notées en complément à 2 en mémoire du calculateur

▷ **Exemple 2.8**

Soit la mémoire de taille suivante :

4 bits	12 bits
exposant	mantisse

Coder la valeur 26,75 en virgule flottante.

$(26,75)_{10} = (11010,110)_2$

$(11010,11)_2 = (11010110).2^{-3}$

$$\Rightarrow \begin{cases} \text{exposant} = & -3 \\ \text{mantisse} = & 11010110 \end{cases}$$

1101	0000011010110

exp=−3 mantisse=214

$26,75 = 214.2^{-3}$

2ème approche :

C'est la méthode inverse de la précédente : on considère que le bit le plus à gauche de la mantisse a pour poids 2^{-1}.

Soit : $N = a_3\ a_2\ a_1\ a_0\ ,\ a_{-1}\ a_{-2}\ a_{-3}$
N peut aussi se noter $(0, \underbrace{a_{-1}\ a_{-2}\ a_{-3}\ a_{-4}\ a_{-5}\ a_{-6}\ a_{-7}}_{\text{mantisse}}).\ \underbrace{2^4}_{\text{exp}}$

▷ **Exemple 2.9**

Même exemple que précédemment :
$(26,75)_{10} = (11010,110)_2 \longrightarrow (0,11010110).2^5$

0101	110101100000

► **Remarque 2.3**

Les ordinateurs utilisent cette représentation avec 32 bits pour la mantisse et 8 bits pour l'exposant. En général, on utilise la représentation inverse, avec le bit le plus à gauche égal à 1*, soit une mantisse normalisée* $\Rightarrow 0,5 \leq M < 1$

2.3.3 La représentation IEEE 754

2.3.3.a Présentation [WebMul]

Le standard IEEE 754 définit trois formats : les nombres en simple précision sur 32 bits, les nombres en double précision sur 64 bits, et les nombres en représentation intermédiaire sur 80 bits. La représentation sur 80 bits est principalement utilisée en interne par les processeurs pour minimiser les erreurs d'arrondi.

Un nombre N de 32 bits est représenté sous la forme :

s	exposant	mantisse

où le signe « s » est codé sur 1 bit, l'exposant est codé sur 8 bits en code relatif à 127 (*cf.* § 2.2.4 page 22), et la mantisse sur 23 bits.

Un nombre de 64 bits (double précision) utilise la même représentation à ceci près que la taille de l'exposant est portée à 11 bits en code relatif à 1023, et celle de la mantisse à 52 bits.

Une mantisse normalisée commence toujours par un bit 1, suivi par la virgule, puis par le reste de la mantisse. Le bit initial, toujours présent et toujours à 1 dans une mantisse normalisée est implicite et non représenté. La valeur de la mantisse est appelée « significande » ; le significande a donc une valeur implicite $1 \leq x < 2$.

▷ **Exemple 2.10**

– $1 = 2^0 \times (1 + 0)$

Le bit de signe sera 0, l'exposant, en code relatif à 127 sera représenté par 127 = 01111111, et le significande vaut 1, ce qui résulte en une mantisse dont tous les bits sont à 0. La représentation IEEE simple precision IEEE 754 du nombre 1 est donc :

Code(1) = | 0 | 01111111 | 0000...0 | = $3F800000$

s e m

– $0.5 = 2^{-1} \times (1+0)$

Le bit de signe est 0, l'exposant, en code relatif à 127 est représenté par 127 − 1 = 01111110, et le significande vaut 1, ce qui résulte en une mantisse dont tous les bits sont à 0. La représentation IEEE simple précision IEEE 754 du nombre 0,5 est donc :

Code(0,5) = | 0 | 01111110 | 0000...0 | = $3F000000$

s e m

– $1.5 = 2^{0} \times (1+2^{-1})$

Le bit de signe est 0, l'exposant, en code relatif à 127 est représenté par 127 = 01111111, et le significande vaut 1,1, ce qui résulte en une mantisse dont le premier bit est à 1 et les 22 suivants à 0. La représentation IEEE simple precision IEEE 754 du nombre 1,5 est donc :

Code(1,5) = | 0 | 01111111 | 1000...0 | = $3FC00000$

s e m

2.3.3.b Nombres spéciaux

En arithmétique à virgule flottante on peut obtenir un résultat valable, ou alors rencontrer un problème de dépassement par valeur supérieure (*overflow*) lorsque le résultat est trop grand pour pouvoir être représenté, ou par valeur inférieure (*underflow*) lorsque le résultat est trop petit.

Dépassement par valeur inférieure

Cette situation arrive lorsqu'un résultat est trop petit pour pouvoir être représenté. Le standard IEEE 754 résout partiellement le problème en autorisant dans ce cas une représentation dénormalisée. Une représentation dénormalisée est caractérisée par le fait d'avoir un code d'exposant complètement nul, ce qui est interprété comme une indication du fait que le bit de poids fort de la mantisse, implicite, est cette fois à 0 au lieu d'être à 1. De cette façon, le plus petit nombre « exprimable » est : $2^{-127} \times 2^{-23} = 2^{-150} \sim 10^{-45}$.

Cependant, il faut remarquer que plus le nombre représenté est petit, moins sa mantisse comportera de bits significatifs. Ce schéma permet une approche « douce » du phénomène de dépassement par valeur inférieure, en sacrifiant la précision lorsqu'un résultat est trop petit pour admettre une représentation normalisée.

Zéro

Zéro est représenté sous la forme d'un nombre dénormalisé. Ceci résulte en deux représentations possibles pour zéro : l'une pour +0, l'autre pour −0. Ces représentations sont caractérisées par un bit de signe suivi par 31 zéros.

Dépassement par valeurs supérieures

Le dépassement par valeurs supérieures ne peut pas être traité comme le dépassement par valeurs inférieures, et est indiqué par un code d'exposant dont tous les bits sont à 1, suivi par une mantisse dont tous les bits sont à 0. Ceci est interprété comme représentant l'infini. L'infini peut être positif ou négatif, en fonction de la valeur du bit de signe. L'infini peut être utilisé dans les calculs et les résultats correspondent au sens commun : $\infty + \infty = \infty$; $x/\infty = 0$; $x/0 = \infty$.

***Not a Number* (NaN)**

Cependant, certaines opérations peuvent ne conduire à aucun résultat exprimable, comme $\infty/\infty =?$ ou $0 \times \infty =?$.

Le résultat de telles opération est alors indiqué par un autre code spécial : le code d'exposant a tous les bits à 1, suivi par une mantisse non nulle. Le « nombre » correspondant est appelé NaN (*Not a Number*) : c'est un non-nombre.

2.3.3.c Résumé

Nombre	**Signe**	**Exposant**	**Mantisse**
nombre normalisé	0/1	01 à FE	quelconque
nombre dénormalisé	0/1	00	quelconque
zéro	0/1	00	0
∞	0/1	FF	0
NaN	0/1	FF	tout sauf 0

IEEE 754	**Simple précision**	**Double précision**
exposant	-126 à $+127$	-10^{22} à $+10^{23}$
mantisse	1 à $2-2^{-23}$	1 à $2-2^{-52}$
+ pt # normalisé	2^{-126}	2^{-1022}
+ gd # normalisé	presque 2^{128}	presque 2^{1024}
intervalle utile	$\approx 10^{-38}$ à 10^{38}	$\approx 10^{-308}$ à 10^{308}
+ pt # dénormalisé	$2^{-150} \approx 10^{-45}$	$2^{-1074} \approx 10^{-324}$

2.4 Arithmétique binaire

2.4.1 Addition

L'addition en binaire se fait avec les mêmes règles qu'en décimal : on commence par additionner les bits de poids faibles ; on a des retenues lorsque la somme de deux bits de même poids dépasse la valeur de l'unité la plus grande (dans le cas du binaire : 1) ; cette retenue est reportée sur le bit de poids plus fort suivant.

La table d'addition binaire est la suivante :

A		B		C	retenue	(*carry*)
0	+	0	=	0	0	
0	+	1	=	1	0	
1	+	0	=	1	0	
1	+	1	=	0	1	

▷ **Exemple 2.11**

Addition des nombres de 4 bits :

	0	0	1	1		+3
+	1	0	1	0	+	−6
=	1	1	0	1	=	−3

	0	1	1	1	,	1	1		7,75
+	0	1	0	1	,	0	1	+	5,25
=	1	1	0	1	,	0	0	=	−3,00

La retenue de la deuxième opération indique un dépassement de capacité (*overflow*) : le bit de signe est à 1 alors qu'il aurait dû être à 0 (addition de deux nombres positifs).

Conditions de dépassement lors de l'addition de deux nombres A et B de 16 bits :

a_{15}	b_{15}	r_{15}	opérandes		résultat	R	D
0	0	0	$a>0$	$b>0$	$r>0$	0	non
0	0	1	$a>0$	$b>0$	$r<0$	0	oui
0	1	0	$a>0$	$b<0$	$r>0$	1	non
0	1	1	$a>0$	$b<0$	$r<0$	0	non
1	0	0	$a<0$	$b>0$	$r>0$	1	non
1	0	1	$a<0$	$b>0$	$r<0$	0	non
1	1	0	$a<0$	$b<0$	$r>0$	1	oui
1	1	1	$a<0$	$b<0$	$r<0$	1	non

R : retenue ; D : dépassement

Ce tableau nous permet de déterminer la condition de dépassement (OF : *overflow flag*) : OF = $\overline{a_{15}}.\overline{b_{15}}.r_{15} + a_{15}.b_{15}.\overline{r_{15}}$.

– si OF est à 0, le bit de poids fort (r_{15}) donne le signe du résultat dont la valeur est disponible sur les 15 bits de poids faibles.
– si OF est à 1, l'indicateur de retenue (C) donne le signe du résultat qui est lui-même sur 16 bits. Dans ce dernier cas, le bit de poids fort ne donne pas le signe du résultat !

2.4.2 Soustraction

Dans la soustraction binaire, 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. En binaire, ce 1 ajoute 2 à la quantité dont on soustrait, tandis qu'en décimal il ajoute 10.

La table de soustraction binaire est la suivante :

A		B		C	retenue	(*borrow*)
0	−	0	=	0	0	
0	−	1	=	1	1	
1	−	0	=	1	0	
1	−	1	=	0	0	

▷ **Exemple 2.12**

Soient les nombres non signés représentés en virgule fixe suivants :

		1	0	1	,	0			5
	−	0	1	1	,	1		−	3,5
0	←	0	0	1	,	1		=	1,5

Soient les nombres entiers signés suivants :

		0	0	0	1	1		3
	–	0	1	1	0	0	–	12
1	←	1	0	1	1	1	=	-9

► **Remarque 2.4**

On peut utiliser le complément à 2 *de la valeur à soustraire puis on additionne. Dans ce cas, il faut complémenter la retenue (*carry*) pour obtenir la retenue soustractive (*borrow*). Cela se passe de cette manière dans certains calculateurs.*

▷ **Exemple 2.13**

$7-2$:

$7=$	0	0	1	1	1			0	0	1	1	1
$2=$	0	0	0	1	0		+	1	1	1	1	0
$-2=$	1	1	1	1	0	1	←	0	0	1	0	1

On ne tient pas compte de la retenue.

2.4.3 Multiplication

La table de multiplication en binaire est très simple :

A		B		C
0	×	0	=	0
0	×	1	=	0
1	×	0	=	0
1	×	1	=	1

La multiplication se fait en formant un produit partiel pour chaque chiffre du multiplieur (seul les bits non nuls donneront un résultat non nul). Lorsque le bit du multiplieur est nul, le produit partiel est nul, lorsqu'il vaut un, le produit

partiel est constitué du multiplicande décalé du nombre de positions égal au poids du bit du multiplieur.

▷ **Exemple 2.14**

			0	1	0	1	multiplicande			5
×			0	0	1	0	multiplieur	×		2
			0	0	0	0				
		0	1	0	1	=				
	0	0	0	0	=	=				
=		0	1	0	1	0		=		10

► **Remarque 2.5**

La multiplication binaire par 2^N, se résume à un décalage de N bits vers la gauche. On introduira donc à droite N zéro.

▷ **Exemple 2.15**

8×4 sur 8 bits :

0	0	0	0	1	0	0	0

$\Longrightarrow$

0	0	1	0	0	0	0	0	$\leftarrow 0$

-16×4 sur 8 bits :

1	1	1	1	0	0	0	0

$\Longrightarrow$

1	1	0	0	0	0	0	0	$\leftarrow 0$

2.4.4 Division

La table de division binaire est la suivante :

A		B		C
0	÷	0	=	impossible
0	÷	1	=	0
1	÷	0	=	impossible
1	÷	1	=	1

La division binaire s'effectue à l'aide de soustractions et de décalages, comme la division décimale, sauf que les chiffres du quotient ne peuvent être que 1 ou 0. Le bit du quotient est 1 si on peut soustraire le diviseur, sinon il est 0.

▷ **Exemple 2.16**

Division du nombre $(10010000111)_2$ par $(1011)_2 = (1101001)_2$ reste $(100)_2$,

c'est-à-dire 1159/11 = 105, reste 4.

1	0	0	1	0	0	0	0	1	1	1	1	0	1	1					
−	1	0	1	1							0	1	1	0	1	0	0	1	
	0	1	1	1	0														
	−	1	0	1	1														
		0	0	1	1	0	0												
			−	1	0	1	1												
				0	0	0	1	1	1	1									
						−	1	0	1	1									
							0	1	0	0									

► **Remarque 2.6**

La division binaire par 2^N, se résume à un décalage de N bits vers la droite. En arithmétique signée, il faut penser à recopier à gauche le bit de signe autant de fois que nécessaire.

▷ **Exemple 2.17**

8/4 sur 8 bits :

0	0	0	0	1	0	0	0

$\Longrightarrow$ **0→**

0	0	0	0	0	0	1	0

−16/4 sur 8 bits :

1	1	1	1	0	0	0	0

$\Longrightarrow$ **1→**

1	1	1	1	1	1	0	0

2.5 En résumé

- La valeur d'un nombre est indépendante de la base dans laquelle il est noté.
- Un nombre binaire peut avoir plusieurs valeurs différentes selon le système de représentation. Soit le nombre binaire $a_n a_{n-1} \ldots a_1 a_0$. Ce nombre vaut :
  - ☞ $a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
    en représentation non signée
  - ☞ $-a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
    en représentation signée complément à 2
  - ☞ $1 - a_n.2^n + a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0$
    en représentation signée complément à 1
  - ☞ $-1^{a_n} \times (a_{n-1}.2^{n-1} + \ldots + a_1.2 + a_0)$

en représentation module et signe

– Les opérations arithmétiques obéissent en binaire aux mêmes règles qu'en décimal, il suffit juste de se rappeler que la base de numération est 2 et non plus 10.

L'arithmétique binaire

▶Exercice 2.1

Que peuvent représenter les octets suivants ?
01111001 ; 10100100 ; 01101010 ; 10010111

▶Exercice 2.2

On effectue les opérations suivantes sur des octets signés (représentation en complément à 2). Donner les résultats en discutant leur validité. Vérifier en prenant les équivalents décimaux.
$5F+6D$; $E8+C7$; $9A-17$; $5B-C4$; $A4-62$

▶Exercice 2.3

Une mémoire contient des octets stockés entre les adresses $(9400)_{\mathrm{H}}$ et $(B3FF)_{\mathrm{H}}$. Combien d'octets contient-elle ? Quelle est la capacité totale en kbits ?

▶Exercice 2.4

Une mémoire contient 2 kilo-octets stockés à partir de l'adresse $(700)_{\mathrm{H}}$. Quelle est la dernière adresse ?

Chapitre 3

Les codes numériques

Richard Wesley Hamming
⋆ 11 fév. 1915 à Chicago, E.-U.
† 7 jan. 1998 à Monterey, E.-U.

« Indeed, one of my major complaints about the computer field is that whereas Newton could say, "If I have seen a little farther than others, it is because I have stood on the shoulders of giants," I am forced to say, "Today we stand on each other's feet." Perhaps the central problem we face in all of computer science is how we are to get to the situation where we build on top of the work of others rather than redoing so much of it in a trivially different way. Science is supposed to be cumulative, not almost endless duplication of the same kind of things. »
(Richard W. Hamming,
One Man's View of Computer Science, 1968, Turing Award Lecture)

Codage : opération qui établit une correspondance entre un ensemble source (nombre, caractère, symbole) vers un ensemble destination contenant des combinaisons de 0 et de 1.

3.1 Codes numériques pondérés

3.1.1 Code binaire pur

Le code binaire pur est un code pondéré par des puissances de 2, utilisé en arithmétique binaire. Ses dérivées sont le code octal et le code hexadécimal.

▷ **Exemple 3.1**

$$01100101 = 2^6 + 2^5 + 2^2 + 2^0 = 64 + 32 + 4 + 1 = (101)_{10}$$

3.1.2 Code binaire en complément vrai (Complément à 2)

Tout comme le code binaire pur, c'est un code pondéré par des puissances de 2 dont le bit de poids fort a un poids négatif. C'est le code le plus utilisé en arithmétique binaire.

▷ **Exemple 3.2**

$$10011011 = -2^7 + 2^4 + 2^3 + 2^1 + 2^0 = -128 + 16 + 8 + 2 + 1 = (-101)_{10}$$

3.1.3 Code DCB (décimal codé binaire)

Dans le code DCB, chaque chiffre décimal $(0,1,\ldots,9)$ est codé en binaire avec 4 éléments binaires. C'est un code pondéré avec les poids $1,2,4,8,10,20,40,80,100,\ldots$ Plus facile pour coder des grands nombre, il est surtout utilisé pour l'affichage des nombres.

► **Remarque 3.1**

Il ne faut pas confondre le code DCB et le code binaire pur : quand on code selon le code binaire pur on prend le nombre dans son intégralité et on le convertit ; par contre, quand on code en DCB on code chaque chiffre indépendamment les uns des autres.

▷ **Exemple 3.3**

$$\begin{aligned}(137)_{10} &= (010001001)_2 \\ &= (000100110111)_{\text{DCB}}\end{aligned}$$

Comme tous les systèmes de codage pondérés, il est possible d'appliquer des opérations arithmétiques aux nombres codés en DCB. L'arithmétique DCB est en fait une arithmétique modulo 6.

3.1.3.a Addition en DCB

L'addition de deux nombres codés en DCB ne pose pas de problème tant que le résultat est inférieur ou égal à 9 :

	0000 0010
+	0000 0101
	0000 0111

	02
+	05
	07

Par contre, dès que le résultat est supérieur à 9, il faut apporter une correction en additionnant 6, de manière à obtenir une réponse valide. Ceci est dû au fait que l'on représente un nombre modulo 10 avec un code modulo 16 : $16-10=6$.

	0000 0110		06	
+	0000 0100	+	04	
=	0000 1010	=	0?	erreur !
+	0000 0110	+	06	
=	0001 0000	=	10	

La correction est ici évidente, puisque la valeur obtenue est invalide en codage DCB. L'exemple suivant est moins évident :

	0000 1001		09	
+	0000 1000	+	08	
=	0001 0001	=	11	erreur !
+	0000 0110	+	06	
=	0001 0111	=	17	

Dans ce dernier exemple, la correction est due au fait qu'il a eu débordement sur les 4 bits de poids faible du mot DCB : il faut donc apporter une correction sur ces 4 bits de poids faible.

▶Note 3.1

- *lorsque le résultat de l'addition DCB est inférieur à* 9*, on ne change pas le résultat ;*
- *lorsque le résultat de l'addition DCB est supérieur à* 9*, on ajoute* 6 *au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue auxiliaire (ou décimale) (*auxiliary *ou* decimal carry*), on ajoute également* 6 *au résultat obtenu, même si la valeur est inférieure à* 9*.*

Les propriétés énoncées ci-dessus pour les chiffres des unités sont évidemment valables pour les dizaines, les centaines, etc. La correction à apporter sera alors – selon les circonstances – +6, +60, +66, etc.

3.1.3.b Soustraction en DCB

La soustraction en DCB se comporte exactement comme l'addition, au signe près.

▶**Note 3.2**

- *lorsque le résultat de la soustraction DCB est inférieur à* 9, *on ne change pas le résultat ;*
- *lorsque le résultat de la soustraction DCB est supérieur à* 9, *on soustrait* 6 *au résultat pour obtenir la valeur exacte ;*
- *lorsqu'il y a une retenue soustractive (*borrow*), on soustrait également* 6 *au résultat obtenu, même si la valeur est inférieure à* 9.

3.1.4 Code binaire de Aiken

Pondéré par 2421, le code Aiken est un code autocomplémentaire (les représentations de 2 chiffres dont la somme est 9 sont complémentaires l'une de l'autre.

Il peut être constitué par les règles suivantes :
- de 0 à 4 on code en binaire pur ;
- de 5 à 9 on ajoute 6 et on code en binaire pur. (c.à.d. $5 \rightarrow 5+6=11, 6 \rightarrow 6+6=12, \ldots$)

▷**Exemple 3.4**

décimal	Aiken			
	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

décimal	Aiken			
	2	4	2	1
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Application pratique 3.1 : Arithmétique

Ce code est utilisé dans certains calculateurs pour effectuer des soustractions par additions de la forme complémentaire.

3.2 Codes numériques non pondérés

3.2.1 Code majoré de trois (excédant de neuf)

Le code majoré de trois consiste à prendre chaque chiffre décimal, à lui additionner 3, puis à convertir le résultat obtenu en binaire. On a parfois recours à ce code en raison de la facilité avec laquelle on peut faire certains calculs arithmétiques. La valeur d'un mot en code majoré de trois est en fait égale au code DCB auquel on a ajouté 3.

▷ **Exemple 3.5**

$(48)_{10}$

4	8
+3	+3
7	11
↓	↓
0111	1011

3.2.2 Code de Gray (binaire réfléchi)

Le code Gray, ou code binaire réfléchi, est à un code dit « à distance minimale » ou « à termes adjacents », c'est-à-dire un code pour lequel un seul bit change entre deux nombres consécutifs. Ce code fait donc apparître la notion d'adjacence entre deux termes.

Ce code est utilisé dans les tableaux de Karnaugh (*cf.* section 5.1.1.c page 73), dans des circuits d'entrée/sortie, notamment dans les codeurs optiques et dans certains convertisseurs analogique/numérique.

Il ne convient pas pour l'arithmétique binaire.

Décimal	Gray			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0

Décimal	Gray			
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

Le code présente 4 symétries miroir. Il est cyclique : il se referme sur lui-même. C'est cette particularité qui est exploité dans les codeurs optiques (*cf.* Application 3.2 de la présente page).

Pour convertir un nombre en code binaire naturel (CBN) vers un nombre en code binaire réfléchi (CBR), il faut ajouter le CBN trouvé à lui-même décalé d'un rang vers la gauche, sans tenir compte de l'éventuelle retenue et en abandonnant dans le résultat le bit de poids faible.

▷ **Exemple 3.6**

Soit le nombre décimal 87 ; sa valeur binaire est 1010111. Donc :

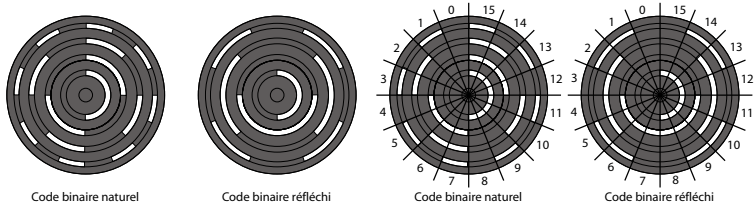
$$\begin{array}{r} 1010111 \\ +1010111\color{white}{0} \\ \hline 11111001 \end{array}$$

L'équivalent en code binaire réfléchi de $(87)_{10}$ est 1111100

Application pratique 3.2 : Les codeurs optiques

Le code Gray est souvent utilisé comme système de représentation des nombres dans les codeurs optiques, préférentiellement au code binaire naturel. Ces systèmes permettent de connaître avec une certaine précision la position absolue d'un moteur. Cette précision dépend du nombre de segments de la roue codeuse, lui même dépendant du nombre de bits codés.

La figure ci-dessous à gauche présente deux roues codeuses 4 bits, l'une en code binaire naturel, l'autre en code binaire réfléchi.



On constate que chaque segment de ces disques contient 4 zones qui peuvent être ouvertes (1 logique), ou fermées (0 logique). La figure ci-dessus à droite fait apparaître plus clairement chaque segment et la valeur décimale associée.

Comme on le constate sur ces exemples, le bit de poids fort est au plus proche de l'axe du disque, et le bit de poids faible à la périphérie. Ceci est dû au fait que la couronne des bits de poids faible est susceptible de comporter un nombre très important d'alternances pour les codeurs optiques à grande précision, comme on peut le constater sur le codeur optique 12 bits ci-contre.

3.2.2.a Conversion du code binaire naturel vers binaire réfléchi

	Binaire naturel				Binaire réfléchi			
	a_3	a_2	a_1	a_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Les équation logiques pour un mot de 4 bits sont :

- ☞ $g_0 = a_1 \oplus a_0$
- ☞ $g_1 = a_2 \oplus a_1$
- ☞ $g_2 = a_3 \oplus a_2$
- ☞ $g_3 = a_3$

Pour un mot binaire de format n on a donc :

- ☞ $g_i = a_{i+1} \oplus a_i$, pour $n-2 \geq i \geq 0$

- ☞ $g_{n-1} = a_{n-1}$

On peut également exprimer g_n de manière récursive :

- ☞ $g_0 = g_3 \oplus g_2 \oplus g_1 \oplus a_0$
- ☞ $g_1 = g_3 \oplus g_2 \oplus a_1$
- ☞ $g_2 = g_3 \oplus a_2$
- ☞ $g_3 = a_3$

3.2.2.b Conversion du code binaire réfléchi vers binaire naturel

	Binaire réfléchi				Binaire naturel			
	g_3	g_2	g_1	g_0	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

Les équation logiques pour un mot de 4 bits sont :

- ☞ $a_3 = g_3$
- ☞ $a_2 = g_3 \oplus g_2$
- ☞ $a_1 = g_3 \oplus g_2 \oplus g_1$
- ☞ $a_0 = g_3 \oplus g_2 \oplus g_1 \oplus g_0$

Pour un mot binaire de format n on a donc :

- ☞ $a_{n-1} = g_{n-1}$

☞ $a_i = \oplus \sum_{j=1}^{n-1} g_i = a_{i+1} \oplus g_i$, pour $n-2 \geq i \geq 0$

3.3 Codes détecteurs d'erreurs et autocorrecteurs

Ces codes sont utilisés pour contrôler la transmission des données.

Souvent, on utilise un nombre de bits supérieur à celui strictement nécessaire pour coder l'information elle-même.

3.3.1 Les codes p *parmi* n

Ce sont des codes autovérificateurs (détecteurs d'erreurs mais pas autocorrecteurs). Ces codes possèdent n éléments binaires dont p sont à 1 ; la position des « 1 » permet de reconnaître un élément codé. Le nombre de combinaisons répondant à cette définition est :

$$C_n^p = \frac{n!}{p!(n-p)!}$$

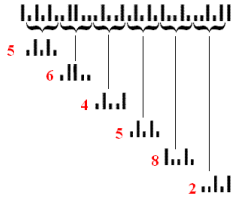
3.3.1.a Les codes 2 parmi 5

Pour transmettre l'information numérique dans les centraux téléphoniques (*cross bar*), on utilise un code 2 parmi 5 (ou code 01236) pour représenter les chiffres décimaux.

Les codes 2 parmi 5 possèdent 10 combinaisons possibles :

Déc.	Télécoms					PostNET				
	0	1	2	3	6	7	4	2	1	0
1	1	1	0	0	0	0	0	0	1	1
2	1	0	1	0	0	0	0	1	0	1
3	1	0	0	1	0	0	0	1	1	0
4	0	1	0	1	0	0	1	0	0	1
5	0	0	1	1	0	0	1	0	1	0
6	1	0	0	0	1	0	1	1	0	0
7	0	1	0	0	1	1	0	0	0	1
8	0	0	1	0	1	1	0	0	1	0
9	0	0	0	1	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	0	0

Application pratique 3.3 : Le code PostNET



Le système PostNET (*Postal Numeric Encoding Technique*) est utilisé aux États-Unis (United States Postal Service) pour faciliter l'aiguillage du courrier. Le code postal américain, appelé ZIP (*Zoning Improvment Plan*), est converti en barres de hauteur entière ou de demi-hauteur selon le code 74210 (un « 0 » est représenté par une barre en demi-hauteur, et un « 1 » par une barre entière).

Application pratique 3.4 : Les codes barres

Les codes 3 parmi 9 (codes 39 et 93) et 2 parmi 5 (code 2 parmi 5 entrelacé) sont notamment utilisés pour coder les caractères numériques (code 2 parmi 5) ou alphanumériques (codes 3 parmi 9) dans les codes barres.

Le code à barres le plus répandu est cependant le code EAN (*European Article Numbering*) qui utilise un codage plus complexe des caractères numériques uniquement.

3.3.2 *Le code biquinaire*

C'est un code composé d'un groupe de n bits (en général 5) dont un seul parmi n progresse à la fois, et d'un groupe de m bits (1 à 2) assurant la distinction entre $n < 5$ et $n \geq 5$.

▷ **Exemple 3.7**

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

Chaque combinaison a un nombre pair de 1, ce qui introduit une sécurité lors de la transmission de ces valeurs. Ce code permet donc de détecter deux erreurs à condition qu'elle ne soit pas dans le même groupe ; par contre, il ne permet ni de les localiser ni de les corriger.

Ce code est utilisé dans les calculatrices.

3.3.3 Les codes à contrôle de parité

Dans ces codes, on ajoute un éléments binaire de sorte que l'ensemble des bits à transmettre (ou le mot) ait un nombre pair (parité paire) ou impair (parité impaire) de « 1 ».

▷ **Exemple 3.8**

0101 ⟶ 00101
0111 ⟶ 10111

► **Remarque 3.2**

Dans l'application de la méthode de la parité, l'émetteur et le récepteur se mettent d'accord à l'avance sur la parité à surveiller (paire ou impaire).

► **Remarque 3.3**

Pour détecter la place d'un élément binaire faux, il faut coder dans deux dimensions selon les lignes et les colonnes.

▷ **Exemple 3.9**

0	1	0	0	1	Transmission ⟶	→	0	1	0	0	1
1	0	0	1	0			1	0	0	0	0
0	0	0	1	1			0	0	0	1	1
1	1	1	0	1			1	1	1	0	1
0	0	1	0	1			0	0	1	0	1
										↑	

Ce code détecte les erreurs simples à condition que l'élément binaire de parité ne soit pas erroné.

3.3.4 Code de Hamming

Ce code est utilisé dans les transmissions de données. Il localise et corrige les chiffres erronnés (en ajoutant des éléments binaires supplémentaires aux bits de l'information).

Le nombre binaire d'information effective est : $N = ABCD = 4$

Le nombre binaire d'information transmise est : $N = abcdefg = 7$

avec $a = A \oplus B \oplus C \oplus D$
$b = A \oplus C \oplus D$
$c = A$
$d = B \oplus C \oplus D$
$e = B$
$f = C$
$g = D$

3.4 Les codes alphanumériques

Ils servent à coder des chiffres, des lettres, des signes de ponctuations et des caractères spéciaux (26 caractères minuscules, 26 caractères majuscules, 7 signes, 20 à 40 caractères spéciaux comme +,|,≠,%,...)

*3.4.1 Le code ASCII (*American Standard Code for Information Interchange*)*

C'est le plus répandu. On le retrouve pratiquement dans tous les ordinateurs et leurs organes périphériques, pour leurs dialogues et la représentation des textes en mémoire.

Chaque symbole (caractère d'imprimerie) est codé par 7 éléments binaires (un $8^{ème}$ bit peut servir de parité) : $2^7 = 128$ combinaisons différentes.

Les codes

Exercice 3.1

Convertir en décimal et hexadécimal les nombres suivants :
$(10010101)_{\text{DCB}}$; $(10010101)_{\text{DCB}}$

Exercice 3.2

En parité impaire, quel est le bit de parité à associer aux octets suivants ?
EC ; $F1$; 69 ; $A3$

Exercice 3.3

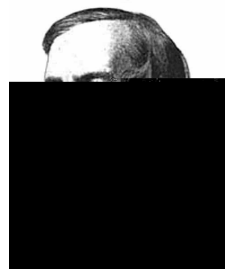
En parité paire, quel est le bit de parité à associer aux octets suivants ?
CD ; $6E$; $B8$; $A4$

Deuxième partie

La logique combinatoire

Chapitre 4

Algèbre booléenne et opérateurs logiques



George Boole
⋆ 2 nov. 1815, Lincoln, R.-U.
† 8 déc. 1864, Ballintemple, Irlande

« Une proposition peut être vraie ou fausse, mais ne peut pas être vraie et fausse. »
(Aristote ⋆ 384, † 322 av. J.-C.)

4.1 Introduction

Les sytèmes logiques fonctionnent en mode binaire ⟶ les variables d'entrée et de sortie ne prennent que deux valeurs : « 0 » ou « 1 ». Ces valeurs (états) « 0 » et « 1 » correspondent à des plages définies à l'avance.

▷ **Exemple 4.1**

- Technologie électrique TTL :
« 1 » ↔ 2,4 à 5 V
« 0 » ↔ 0 à 0,8 V
- Technologie pneumatique :
« 1 » ↔ présence de pression
« 0 » ↔ absence de pression

Les valeurs « 0 » et « 1 » ne représentent pas des nombres réels mais plutôt l'état d'une variable (logique) ⟶ on les appelle donc « niveaux logiques ».

4.1.1 Convention de nommage des synonymes des « 0 » et « 1 » :

Ces deux valeurs peuvent être nommées de différentes façons :

- Niveau logique « 1 » : Vrai, Fermé, Marche, Haut, Allumé, Oui ;
- Niveau logique « 0 » : Faux, Ouvert, Arrêt, Bas, Éteint, Non.

4.1.2 Types de logiques

On définit deux types de logiques :
- Logique positive :
  - niveau haut ⟶ état logique « 1 » (5V)
  - niveau bas ⟶ état logique « 0 » (0V)
- Logique négative :
  - niveau haut ⟶ état logique « 0 » (0V)
  - niveau bas ⟶ état logique « 1 » (5V)

La logique binaire basée sur l'algèbre de Boole permet de décrire dans un modèle mathématique les manipulations et traitement des informations binaires, et d'analyser les systèmes numériques.

Il existe 3 fonctions élémentaires dans l'algèbre de Boole :
- addition logique : appelée `OU`, symbolisée par un plus : « + » ;
- multiplication logique : appelée `ET`, symbolisée par un point : « . » ;
- complémentation : appelée `NON`, symbolisée par un surlignement : « $\overline{a}$ »

Tout circuit numérique peut être défini à l'aide d'une fonction logique (ou expression logique) qui représente la variable de la sortie en fonction des variables d'entrée.

4.1.3 Variables logiques (ou variables binaires)

Ce sont des variables ne pouvant prendre que deux valeurs distinctes : « 0 » ou « 1 ». Une variable binaire peut représenter n'importe quel dispositif binaire (contact, lampe, électro-vanne...)

4.1.4 Convention :

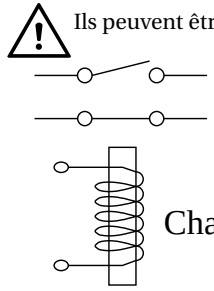
Tout appareil est schématisé à l'état de repos. Dans tous les cas, l'action sur un appareil sera notée a, b, ... et la non action $\overline{a}$, $\overline{b}$, ...

▷ **Exemple 4.2**

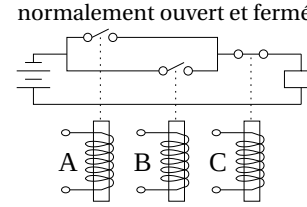
Bouton poussoir ⟶ contact repos et contact travail.
1er cas : schéma d'un contact ouvert au repos dit « contact travail ».
2^{è} cas : schéma d'un contact fermé au repos dit « contact repos ».

▷ **Exemple 4.3**

Relais : c'est un interrupteur opérant de façon électromagnétique ; lorsqu'un courant approprié passe dans le charbon, une force magnétique déplace les armatures imposant l'ouverture ou la fermeture des contacts. Il est présenté dans sa position non alimentéee (au repos).



Symbole d'un relais double



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

4.2 Propriétés de l'algèbre booléenne

4.2.1 Présentation

L'algèbre booléenne définit un cadre mathématique d'étude de propositions logiques portant sur des ensembles E d'éléments.

Définition 4.1

Algèbre booléenne : *un ensemble E d'éléments $(a,b,c,...)$ associé à deux opérations binaires $+$ et $.$ constitue une algèbre booléenne si et seulement si les postulats suivants sont satisfaits :*

- ☞ ***P1*** *Les opérations sont commutatives ;*
- ☞ ***P2*** *Chacune des opérations est distributive sur l'autre ;*
- ☞ ***P3*** *Il existe les éléments identité 0 et 1 respectivement pour $+$ et $.$;*
- ☞ ***P4*** *Pour chaque élément $a \in E$, il existe un élément $\overline{a} \in E$ tel que : $a + \overline{a} = 1$ et $a.\overline{a} = 0$.*

À partir de ces postulats, il est possible de démontrer les théorèmes d'idempotence (*cf.* §4.4.3), de l'élément nul, d'involution (*cf.* §4.4.5), d'absorption (*cf.* §4.5.6), d'associativité ainsi que la loi de De Morgan (*cf.* §4.7). Tous ces théorèmes seront présentés plus loin.

Le lecteur attentif aura remarqué après la lecture des quatre postulats ci-dessus qu'il n'est jamais fait mention du nombre d'éléments dans l'ensemble E, ni encore moins que ce nombre d'éléments est limité à deux !

L'algèbre booléenne n'est pas restreinte aux ensembles binaires.

En fait, le nombre d'éléments dans E peut être infini, mais doit au moins comporter les éléments 0 et 1. Ainsi l'algèbre binaire, qui ne contient que les éléments 0 et 1, constitue l'algèbre booléenne la plus simple.

▷ **Exemple 4.4**

Algèbre booléenne portant sur 4 éléments : $E = \{0, a, b, 1\}$

+	**0**	**a**	**b**	**1**
0	0	a	b	1
a	a	a	1	1
b	b	1	b	1
1	1	1	1	1

.	**0**	**a**	**b**	**1**
0	0	0	0	0
a	0	a	0	a
b	0	0	b	b
1	0	a	b	1

4.3 Algèbre binaire ou algèbre de commutation

4.3.1 Postulats de base

Le domaine de définition B_2 de l'algèbre de commutation comprend donc deux éléments 0 et 1 ($B_2 = \{0, 1\}$).

Si a est une variable logique on a :

☞ **P1** $a = 0$ si et seulement si $a \neq 1$

☞ **P1*** $a = 1$ si et seulement si $a \neq 0$

L'opération `NON`(ou complément), notée « $\overline{a}$ » est définie par :

☞ **P2** $0 = \overline{1}$

☞ **P2*** $1 = \overline{0}$

L'opération `OU`(ou disjonction), notée « + » est définie par :

☞ **P3** $1 + 1 = 1 + 0 = 0 + 1 = 1$

☞ **P4** $0+0=0$

L'opération ET(ou intersection), notée « . » est définie par :

☞ **P3*** $0.0=0.1=1.0=0$

☞ **P4*** $1.1=1$

L'algèbre de commutation est le système algébrique constitué de l'ensemble $\{0,1\}$ et des opérateurs ET, OU, NON.

À partir de ces quatre postulats, on peut construire les différents théorèmes présentés dans les sections § 4.4 de la présente page et § 4.5 page suivante.

4.3.2 Hiérarchie des opérations

Dans une expression sans parenthèses, on effectue d'abord les opérations ET et, par la suite, les OU.

4.3.3 Induction parfaite

Dans le domaine linéaire, il n'est pas possible de prouver une équation en la vérifiant pour toutes les valeurs des variables.

En logique binaire, puisque les variables sont limitées à deux états, on peut prouver une relation en la vérifiant pour toutes les combinaisons de valeurs pour les variables d'entrée. Ainsi, toutes les propriétés présentés dans les sections § 4.4 de la présente page et § 4.5 page suivante peuvent être démontrées par induction parfaite.

On notera qu'il n'est pas évident de démontrer ces relations par induction parfaite en algèbre booléenne de plus de deux variables. La preuve de de ces théorèmes peut être consultée notamment dans [LivWhi61].

4.4 Théorèmes monovariables

4.4.1 Identité

À chaque opérateur correspond un élément neutre qui, lorsqu'il est opéré avec une variable quelconque A, donne un résultat identique à cette variable.

$A+0=A$ $\qquad$ $A.1=A$

4.4.2 Élément nul

À chaque opérateur correspond un élément nul qui, lorsqu'il est opéré avec une variable quelconque A, donne un résultat identique à cet élément nul.

$A+1=1$ $\qquad$ $A.0=0$

4.4.3 Idempotence

Le résultat d'une opération entre une variable A et elle-même est égal à cette variable.

$A+A=A$ $\qquad$ $A.A=A$

4.4.4 Complémentation

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

4.4.5 Involution

Le complément du complément d'une variable A est égal à cette variable.

$\overline{\overline{A}}=A$

4.5 Théorèmes multivariables

4.5.1 Équivalence

Deux fonctions sont équivalentes si on peut leur faire correspondre la même table de vérité.

Si $F=\overline{\mathrm{A}.\mathrm{B}}$ et $G=\overline{\mathrm{A}}+\overline{\mathrm{B}}$, alors $F=G$, et on dit que F est équivalente à G.

4.5.2 Complémentarité

Deux fonctions sont dites complémentaires si l'une est l'inverse de l'autre pour toutes les combinaisons d'entrées possibles.

Si $F=\overline{\mathrm{A}}.\overline{\mathrm{B}}$ et $G=A+B$, alors $F=\overline{\mathrm{G}}$, et on dit que F et G sont complémentaires.

4.5.3 Associativité

Les opérations +, ., et $\oplus$ sont associatives :

$A+B+C=(A+B)+C=A+(B+C)$

$A.B.C=(A.B).C=A.(B.C)$

$A\oplus B\oplus C=(A\oplus B)\oplus C=A\oplus(B\oplus C)$

4.5.4 Commutativité

Les opérations +, ., et $\oplus$ sont commutatives :

$A+B=B+A$ $\qquad$ $A.B=B.A$ $\qquad$ $A\oplus B=B\oplus A$

4.5.5 Distributivité

Chacune des opérations + et . est distributive sur l'autre :

$A.(B+C)=A.B+A.C$ $\qquad$ $A+B.C=(A+B).(A+C)$

On peut remarquer que ce théorème est particulier dans l'algèbre booléenne puisqu'ici les deux expressions sont vraies, alors que seule la première l'est dans l'algèbre ordinaire.

4.5.6 Absorption

Absorption 1 : $\qquad A+(A.B)=A$ $\qquad$ $A.(A+B)=A$

Absorption 2 : $\qquad (A+\overline{B}).B=AB$ $\qquad$ $(A.\overline{B})+B=A+B$

Ce théorème est particulièrement intéressant pour la conception de circuits numériques puisqu'il permet d'éliminer les termes inutiles et par là-même de réduire la complexité du circuit.

4.5.7 Dualité

Deux expressions sont dites duales si l'on obtient l'une en changeant dans l'autre, les `ET` par des `OU`, les `OU` par des `ET`, les « 1 » par des « 0 » et les « 0 » par des « 1 ».

Si on sait que $\overline{A.B}=\overline{A}+\overline{B}$, alors, on saura que $\overline{A+B}=\overline{A}.\overline{B}$ par dualité.

4.5.8 Théorème de De Morgan

Le théorème de De Morgan est une expression du principe de dualité.

Première forme : $\overline{\mathrm{A+B+C+\cdots}} = \overline{\mathrm{A}}.\overline{\mathrm{B}}.\overline{\mathrm{C}}.\cdots$

Deuxième forme : $\overline{\mathrm{A.B.C.\cdots}} = \overline{\mathrm{A}} + \overline{\mathrm{B}} + \overline{\mathrm{C}} + \cdots$

Cf. § 4.7 page 67 pour plus de précisions.

4.5.9 Sommes de produits, produits de sommes et forme canonique

Les expressions booléennes peuvent être manipulées sous différentes formes, certaines d'entre elles étant nécessaires pour simplifier ces expressions :

– somme de produits ; ex. : $F(A,B,C,D) = A.B + A.\overline{C}.D + B.D$

– produit de sommes ; ex. : $F(A,B,C,D) = (A+B).(A+\overline{C}+D).(B+D)$

Une expression est sous sa forme canonique si tous les symboles qui représentent les variables apparaissent dans tous les termes qui la constitue. Lorsqu'une équation est écrite à partir de sa table de vérité, elle est dans sa forme canonique.

4.5.9.a Forme disjonctive et sommes de mintermes

Si une fonction est une somme de produits, on a une somme canonique ou forme disjonctive.

Exemple : $F = \overline{\mathrm{A}}.B.C + A.B.C + A.\overline{\mathrm{B}}.\overline{\mathrm{C}} + \overline{\mathrm{A}}.\overline{\mathrm{B}}.\mathrm{C}$

Une fonction booléenne peut être représentée sous forme d'une somme de produits utilisant les mintermes. Les mintermes sont représentés par des « 1 » dans une table de vérité.

La table suivante donne les mintermes d'une fonction de trois variables :

			m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
A	B	C	$\overline{A}.\overline{B}.\overline{C}$	$\overline{A}.\overline{B}.C$	$\overline{A}.B.\overline{C}$	$\overline{A}.B.C$	$A.\overline{B}.\overline{C}$	$A.\overline{B}.C$	$A.B.\overline{C}$	$A.B.C$
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

4.5.9.b Forme conjonctive et produits de maxtermes

Si une fonction est un produit de somme, on a un produit canonique ou forme conjonctive.

Exemple : $G = (\overline{A} + B + C).(A + B + C).(A + \overline{B} + \overline{C}).(\overline{A} + \overline{B} + \overline{C})$

Une fonction booléenne peut être représentée sous forme d'un produit de sommes utilisant les maxtermes. Les maxtermes sont représentés par des « 0 » dans une table de vérité.

La table suivante donne les maxtermes d'une fonction de trois variables :

			M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
A	B	C	$A+B+C$	$A+B+\overline{C}$	$A+\overline{B}+C$	$A+\overline{B}+\overline{C}$	$\overline{A}+B+C$	$\overline{A}+B+\overline{C}$	$\overline{A}+\overline{B}+C$	$\overline{A}+\overline{B}+\overline{C}$
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

4.5.9.c Représentations d'une fonction sous forme de mintermes et maxtermes

Soit la fonction $\mathcal{F}$ telle que $\mathcal{F}(A,B,C) = A.B + \overline{B}.(\overline{A} + \overline{C})$.

Cette fonction peut être représentée sous sa :

– *première forme canonique* (somme de mintermes) : on developpe la fonction sous la forme d'une somme de produits puis on prend chaque terme avec pour variable manquante X et on applique un `ET` logique avec $X + \overline{X}$;
– *deuxième forme canonique* (produit de maxtermes) : on développe la fonction sous la forme d'un produit de sommes puis on prend chaque terme avec pour variable manquante X et on applique un `OU` logique avec $X.\overline{X}$;

▷ **Exemple 4.5**

Représentation sous forme de somme de mintermes :

$$\begin{aligned}
\mathcal{F}(A,B,C) &= A.B + \overline{B}.(\overline{A} + \overline{C}) \\
&= A.B + \overline{A}.\overline{B} + \overline{B}.\overline{C} \\
&= A.B.(C + \overline{C}) + \overline{A}.\overline{B}.(C + \overline{C}) + \overline{B}.\overline{C}.(A + \overline{A}) \\
&= \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.B.\overline{C} + A.B.C \\
&= \sum m(0,1,4,6,7)
\end{aligned}$$

▷ **Exemple 4.6**

Représentation sous forme de produit de maxtermes

$$\begin{aligned}
\mathcal{F}(A,B,C) &= A.B + \overline{B}.(\overline{A} + \overline{C}) \\
&= A.B + \overline{A}.\overline{B} + \overline{B}.\overline{C} \\
&= (A + \overline{B}).(A + \overline{B} + \overline{C}).(\overline{A} + B + \overline{C}) \text{ par distributivité} \\
&= (A + \overline{B} + C.\overline{C}).(A + \overline{B} + \overline{C}).(\overline{A} + B + \overline{C}) \\
&= (A + \overline{B} + C).(A + \overline{B} + \overline{C}).(\overline{A} + B + \overline{C}) \\
&= \prod M(2,3,5)
\end{aligned}$$

4.5.10 Résumé des propriétés des opérateurs `OU` et `ET`

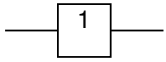
Propriété	`OU`	`ET`
Identité	$a + 0 = a$	$a.1 = a$
Élément neutre	$a + 0 = a$	$a.1 = a$
Élément absorbant	$a + 1 = 1$	$a.0 = 0$
Idempotence	$a + a = a$	$a.a = a$
Complémentation	$a + \overline{a} = 1$	$a.\overline{a} = 0$
Involution	$\overline{\overline{a}} = a$	$\overline{\overline{a}} = a$
Commutativité	$a + b = b + a$	$a.b = b.a$
Associativité	$a + (b + c)$ $= (a + b) + c$	$a.(b.c)$ $= (a.b).c$

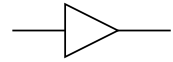
Distributivité	$a+(b.c)$ $=(a+b).(a+c)$	$a.(b+c)$ $=(a.b)+(a.c)$
Absorption 1	$a+a.b=a$	$a.(a+b)=a$
Absorption 2	$a+\overline{a}.b=a+b$	$a.(\overline{a}+b)=a.b$
Consensus	$a.b+\overline{a}.c+bc$ $=a.b+\overline{a}c$	$(a+b).(\overline{a}+c).(b+c)$ $=(a+b).(\overline{a}+c)$
	$(a+b).(\overline{a}+b)=(a.b)+(\overline{a}.b)$	
De Morgan	$\overline{a+b}=\overline{a}.\overline{b}$	$\overline{a.b}=\overline{a}+\overline{b}$

4.6 Opérateurs logiques élémentaires et composés

Les fonctions logiques sont conçues à partir d'un groupe d'opérateurs élémentaires appelés « portes ». Chaque opérateur est représenté par un symbole et sa fonction est définie par une table de vérité.

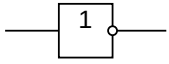
4.6.1 `OUI` *: identité ou transfert*

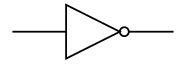




A	$S=A$
0	0
1	1

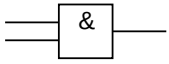
4.6.2 `NON (NOT)` *: complément «* $\overline{a}$ *»*





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

4.6.3 `ET (AND)` *: produit logique «* . *»*

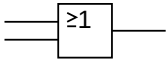


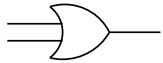
A	B	$S=A.B$
0	0	0
0	1	0
1	0	0
1	1	1

Propriétés du `ET` :

$a.1 = a \quad a.\overline{a} = 0 \quad a.0 = 0 \quad a.a = a$
Élément neutre : 1
Élément absorbant : 0

4.6.4 `OU (OR)` : somme logique « + »





A	B	$S = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Propriétés du `OU` :

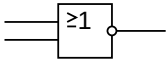
$a + 1 = 1 \quad a + \overline{a} = 1 \quad a + 0 = a \quad a + a = a$
Élément neutre : 0
Élément absorbant : 1

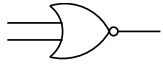
► **Remarque 4.1**

Les opérateurs {`ET,OU,NON`} permettent à eux trois de réaliser n'importe quelle fonction logique : on dit qu'ils forment un groupe complet.
Le théorème de De Morgan permet de dire que les groupes {`ET,NON`} et {`OU,NON`} sont également des groupes complets.

4.6.5 `NON-OU (NOR)` « ↓ »

Les deux opérateurs `OU` et `NON` peuvent être combinés en un seul opérateur `NON-OU` : `NON-OU` est donc un opérateur complet.

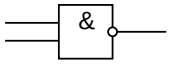




A	B	$S = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

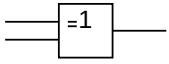
4.6.6 `NON-ET (NAND)` « ↑ »

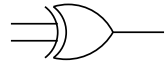
Les deux opérateurs `ET` et `NON` peuvent être combinés en un seul opérateur `NON-ET` : `NON-ET` est donc un opérateur complet.



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

4.6.7 OUX (XOR) : ou exclusif ou dilemme « ⊕ »





A	B	$S = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Propriétés du OUX :

Le ou exclusif est commutatif et associatif
$a \oplus 0 = a \quad a \oplus 1 = \overline{a} \quad a \oplus \overline{a} = 1 \quad a \oplus a = 0$
Élément neutre : 0
Élément absorbant : $a, \overline{a}$

► **Remarque 4.2**

Le ou exclusif est souvent utilisé dans les circuits numériques du fait de ses propriétés :

- *le ou exclusif est l'opérateur somme modulo 2, on le retrouve donc dans les additionneurs ou la sortie $S = a \oplus b \oplus r$;*
- *il est également largement utilisé dans les circuits de correction d'erreurs (calcul de parité) : $b_0 \oplus b_1 \oplus b_2 \oplus \cdots \oplus b_n$ est égal à 0 si le nombre de bits à 1 est pair, à 0 sinon ;*
- *$a \oplus 1 = \overline{a}$ et $a \oplus 0 = a$: le OU exclusif peut être utilisé comme inverseur commandé.*

Le ou exclusif n'est pas un opérateur complet, mais comme il peut être utilisé pour réaliser la complémentation, les groupes {OUX, ET} et {OUX, OU} sont des groupes complets.

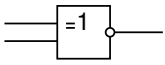
► **Remarque 4.3**

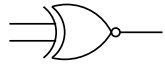
Relations d'identité utilisables avec l'opérateur ou exclusif :

1. $a \oplus b = a\overline{b} + \overline{a}b = (a + b).(\overline{a} + \overline{b})$

2. $\overline{(a \oplus b)} = a \oplus \overline{b} = \overline{a} \oplus b = ab + \overline{ab} = (\overline{a} + b)(a + \overline{b})$
3. $a \oplus a = 0 \quad et \quad a \oplus \overline{a} = 1$
4. $a \oplus 1 = \overline{a} \quad et \quad a \oplus 0 = a$
5. $a(b \oplus z) = ab \oplus az$
6. $a + b = a \oplus b \oplus ab = a \oplus \overline{a}b$
7. $a + b = a \oplus b \quad si \ ab = 0$
8. $a \oplus b = c \Rightarrow c \oplus b = a, \quad c \oplus a = b, \quad a \oplus b \oplus c = 0$
9. $a \oplus (a + b) = \overline{a}b$
10. $a \oplus ab = a\overline{b}$

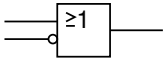
4.6.8 NON-OUX (XNOR) : coïncidence ou équivalence « ⊙ »

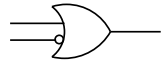




A	B	$S = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

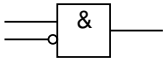
4.6.9 IMP (IMP) : implication « ⊂ » ou « ⊃ »

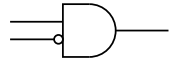




A	B	$S = A + \overline{B}$
0	0	1
0	1	0
1	0	1
1	1	1

4.6.10 INH (INIB) : inhibition « / »





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

4.6.11 Résumé : les différents opérateurs

Nom	Symbole	Valeur de xy 00	01	10	11	Expression algébrique
Zéro		0	0	0	0	$F_0 = 0$
Et	$x.y$	0	0	0	1	$F_1 = x.y$
Inhibition	x/y	0	0	1	0	$F_2 = x.\overline{y}$
Transfert		0	0	1	1	$F_3 = x$
Inhibition	y/x	0	1	0	0	$F_4 = \overline{x}.y$
Transfert		0	1	0	1	$F_5 = y$
Ou exclusif	$x \oplus y$	0	1	1	0	$F_6 = x\overline{y} + \overline{x}y$
Ou	$x + y$	0	1	1	1	$F_7 = x + y$
Non-ou	$x \downarrow y$	1	0	0	0	$F_8 = \overline{x + y}$
Équivalence	$x \odot y$	1	0	0	1	$F_9 = x\overline{y} + \overline{x}\overline{y}$
Complément	$\overline{y}$	1	0	1	0	$F_{10} = \overline{y}$
Implication	$x \subset y$	1	0	1	1	$F_{11} = x + \overline{y}$
Complément	$\overline{x}$	1	1	0	0	$F_{12} = \overline{x}$
Implication	$x \supset y$	1	1	0	1	$F_{13} = \overline{x} + y$
Non-et	$x \uparrow y$	1	1	1	0	$F_{14} = \overline{x.y}$
Un		1	1	1	1	$F_{15} = 1$

4.7 Universalité des portes NON-ET et NON-OU

▶**Note 4.1**

Théorème de De Morgan :

1. *Le complément d'un produit est égal à la somme des compléments des termes du produit :* $\overline{S} = \overline{a.b} = \overline{a} + \overline{b}$
2. *Le complément d'une somme est égal au produit des compléments des termes de la somme :* $\overline{S} = \overline{a+b} = \overline{a}.\overline{b}$

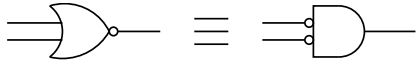
Non seulement le théorème de De Morgan et ses conséquences est très utile pour simplifier des expressions, mais il est également valable si a ou b sont des expressions contenant plusieurs variables.

▷ **Exemple 4.7**

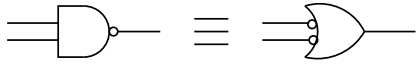
$$\overline{(A.\overline{B}+C)} = \overline{A}.\overline{\overline{B}}.\overline{C} = \overline{A}+B+C$$

En conséquence du théorème de De Morgan, on peut affirmer notamment :

1. une porte NON-OU est une porte ET avec ses entrées inversées :



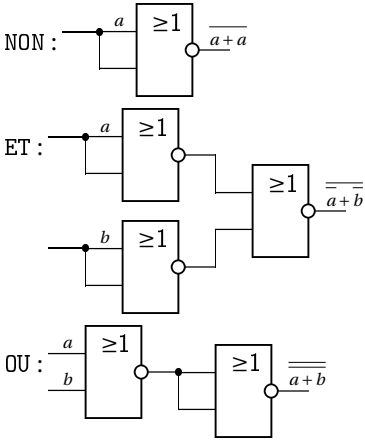
2. une porte NON-ET est une porte OU avec ses entrées inversées :



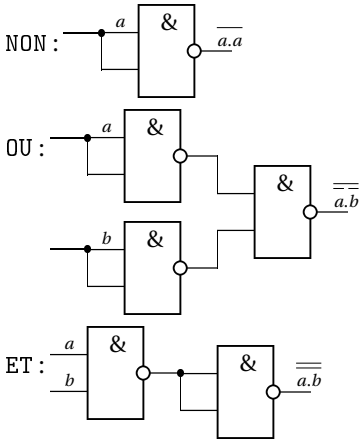
▶**Note 4.2**

Universalité des portes NON-ET et des portes NON-OU :
Toutes les portes logiques élémentaires (ET, OU, NON) peuvent être réalisées avec des portes NON-OU ou NON-ET.

4.7.1 Universalité des portes NON-OU

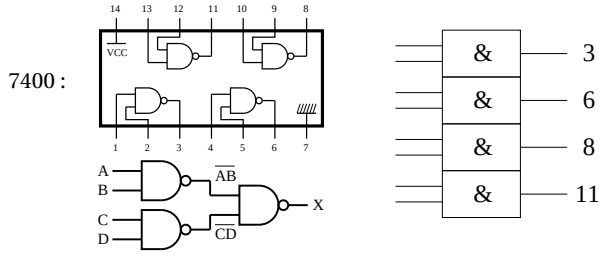


4.7.2 *Universalité des portes NON-ET*



▷ **Exemple 4.8**

Réaliser la fonction X=AB+CD à l'aide du CI (circuit intégré) suivant :



► **Remarque 4.4**

- *le groupe d'opérateurs {ET,OU,NON} permet de réaliser toutes les fonctions logiques : on dit que c'est un « groupe complet », ainsi que les groupes {ET,NON} et {OU,NON} ;*
- *de même, les opérateurs NON-ET, NON-OU, sont appelés des « opérateurs complets » ;*
- *comme l'opérateur OUX peut être utilisé pour réaliser un inverseur, les groupes {ET,OUX} et {OU,OUX} sont également des groupes complets ; le groupe {ET,OUX} est un anneau booléen appelé corps de Galois.*

Chapitre 5

Représentation et simplification des fonctions logiques

Maurice Karnaugh
⋆ 4 oct. 1924, New York, É.-U.

Ph.D. de Physique, Université de Yale - 1952. Chercheur aux laboratoires Bell Telephone de 1952 à 1966, puis au centre de recherche d'IBM à New York de 1966 à 1993. Professeur d'informatique à l'Institut Polytechnique de New York de 1980 à 1999. Élu membre de l'IEEE (IEEEfellow) en 1976, pour ses travaux sur l'utilisation des techniques numériques en télécommunications. Inventeur du diagramme de Karnaugh en logique en 1953. Co-inventeur des premiers circuits logiques (Essex).
Article de référence : Maurice Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits", Trans. AIEE. pt I, 72(9) :593-599, November 1953.

Tout circuit logique peut être décrit par des fonctions logiques et/ou une table de vérité, et être réalisé à partir des opérateurs logiques élémentaires.

5.1 Méthodes de représentation des fonctions logiques

En dehors de la représentation algébrique que nous avons utilisée jusqu'à présent, d'autres méthodes permettent de représenter les fonctions logiques. Les plus couramment employées sont les représentation tabulaires, implicites, et graphiques.

5.1.1 Représentations tabulaires

5.1.1.a Table de vérité

La table de vérité nous fait connaître la réaction d'un circuit logique aux diverses combinaisons de niveaux logiques appliquées à ses entrées. Chaque

ligne présente la combinaison des variables d'entrée ainsi que la ou les sorties correspondante(s).

▷ **Exemple 5.1**

La table de vérité d'un additionneur complet est la suivante :

A	B	C	S	R
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

$$\begin{cases} S = A \oplus B \oplus C \\ R = A.B + A.C + B.C \end{cases}$$

▷ **Exemple 5.2**

Donner la table de vérité d'un circuit à 3 entrées A,B,C et 2 sorties X,Y tel que :

$\begin{cases}$ X=1 si les 3 entrées ont le même niveau
Y=1 si A=B $\end{cases}$

Le principal inconvénient de la table de vérité est qu'elle devient rapidement très encombrante lorsque le nombre de variables d'entrée augmente.

5.1.1.b Diagramme de Veitch

Le diagramme de Veitch est une table sur laquelle on représente les n variables d'entrée selon les deux axes vertical et horizontal. En général, pour $n = p + q$ on porte sur les colonnes p variables où p est la partie entière de $n/2$, et les q variables restantes sur les lignes. Les colonnes et les lignes sont numérotées selon l'ordre binaire naturel.

Le diagramme de Veitch de l'exemple précédent est le suivant :

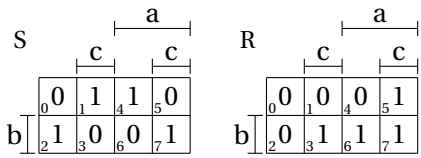
S

b \ a c	00	01	11	10
0	0	1	1	0
1	1	0	0	1

R

b \ a c	00	01	11	10
0	0	0	0	1
1	0	1	1	1

On peut également numéroter les cases du diagramme de Veitch selon l'image décimale de la fonction représentée. Chaque case correspond à une ligne de la table de vérité, et peut donc être représentée par son image décimale (*cf.* § 5.1.2.b) :



5.1.1.c Diagramme de Karnaugh[ArtKar53] et termes adjacents

Le diagramme de Karnaugh est un outil graphique, méthodique. Il permet d'obtenir une solution optimale à la simplification logique (*cf.* § 5.2.3 page 77).

Comme la table de vérité, le diagramme de Karnaugh met en évidence le rapport entre les entrées et les sorties (chaque ligne de la table de vérité correspond à une case du diagramme de Karnaugh).

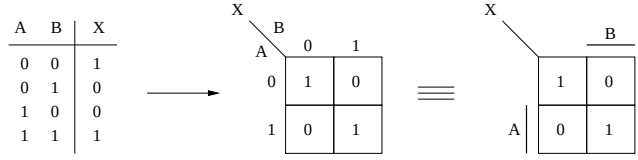
Deux termes sont adjacents quand ils ne diffèrent l'un de l'autre que par une seule variable. $A\overline{B}C$ et ABC sont adjacents. Un diagramme – ou tableau – de Karnaugh est une table d'implication logique disposée de telle manière que deux termes logiquement adjacents soient également adjacents géométriquement.

Le diagramme de Karnaugh est très proche du diagramme de Veitch présenté § 5.1.1.b, mais afin d'exploiter la notion d'adjacence ente les termes, les cases sont ordonnées selon le code binaire réfléchi, au lieu du code binaire naturel.

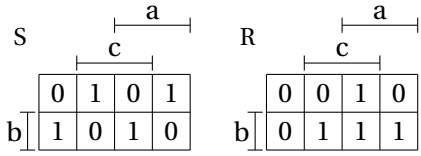
► Remarque 5.1

Les tableaux de Karnaugh se présentent comme des cylindres fermés dans les deux sens.

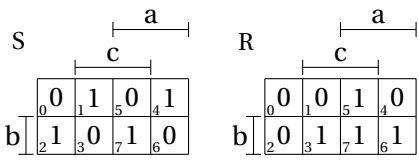
▷ Exemple 5.3



Le diagramme de Karnaugh de l'exemple précédent est le suivant :



Comme pour le diagramme de Veitch, on peut numéroter les cases du diagramme de Karnaugh selon l'image décimale de la fonction représentée :



► **Remarque 5.2**

- *il peut exister des états indifférents (notés « X »). Ces états correspondent à des combinaisons d'entrée impossibles. On les remplacera par 1 ou 0 de façon à avoir la simplification la plus optimale ;*
- *on peut utiliser une meme case plusieurs fois, puisque* $x+x+x+\cdots+x=x$.

Chaque case du tableau représente une combinaison et une seule des variables de la fonction. Dans cette case, on inscrit « 0 » ou « 1 » selon la valeur prise par la fonction. Cette combinaison exclusive de variables peut être notée par un ET entre les variables représentées.

Par exemple, la case pour laquelle $a=0$, $b=1$, $c=0$ et $d=1$ sera notée $\overline{a}b\overline{c}d$: c'est un « minterme ».

La représentation de la fonction sera alors la somme logique (OU) de toutes les combinaisons pour lesquelles la fonction vaut « 1 ».

Quelquefois, on peut préférer considérer la seconde forme canonique. La combinaison exclusive de variables sera alors notée par un OU entre les variables représentées.

Par exemple, la case pour laquelle $a=0$, $b=1$, $c=0$ et $d=1$ sera notée $\overline{a}+b+\overline{c}+d$: c'est un « maxterme ».

5.1.1.d Diagramme de Venn

À venir ...

5.1.1.e Diagramme de Johnston

À venir ...

5.1.1.f Diagramme de Caroll

À venir ...

5.1.2 Représentations implicites

5.1.2.a Image caractéristique

L'image caractéristique d'une fonction $\mathcal{F}$ à n entrée est constituée des 2^n valeurs de cette fonction, ordonnées selon l'ordre binaire naturel.

Ainsi, soit la fonction $\mathcal{F}(x_0, x_1)$ suivante, définie par sa table de vérité :

x_0	x_1	$\mathcal{F}$
0	0	0
0	1	1
1	0	0
1	1	0

On peut représenter $\mathcal{F}$ par son image caractéristique, soit $I_c[\mathcal{F}(x_0, x_1)] = 0100$.

Reprenons la table de vérité d'un additionneur complet :

A	B	C	S	R
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

L'image caractéristique de S est $I_c[S(A,B,C)] = 01101001$.

L'image caractéristique de R est $I_c[R(A,B,C)] = 00010111$.

5.1.2.b Image décimale

Nous avons vu que toute fonction logique $\mathcal{F}$ peut s'exprimer par ses formes canoniques, soit comme somme de produits, soit comme produit de sommes.

On notera donc la fonction $\mathcal{F}$ comme :

- ☞ la somme des états pour lesquels elle vaut « 1 » que l'on notera : $\mathcal{F}_1 = \sum(d_1, \ldots, d_p)$
- ☞ le produit des états pour lesquels elle vaut « 0 » que l'on notera : $\mathcal{F}_0 = \prod(d_1, \ldots, d_p)$

où d_1 à d_p représentent les valeurs décimales des nombres binaires représentés par les variables de la fonction

Reprenons comme exemple la fonction $\mathcal{F}(x_0, x_1)$ présentée § 5.1.2.a. On numérote les différents états de cette fonction en attribuant des poids aux variables selon l'ordre binaire naturel ; notons N la valeur décimale de ces états :

N	x_0	x_1	$\mathcal{F}$
0	0	0	0
1	0	1	1
2	1	0	0
3	1	1	0

La fonction $\mathcal{F}(x_0, x_1)$ peut s'écrire :

$\mathcal{F}_1 = \overline{x_0}.x_1 = \sum(1)$

$\mathcal{F}_0 = (\overline{x_0} + \overline{x_1}).(x_0 + \overline{x_1}).(x_0 + x_1) = \prod(0,2,3)$

Le principal avantage de la notation décimale est le risque d'erreur très faible lors de son écriture. En effet, il est plus difficile de remplacer un 3 par un 1 que d'oublier une barre de complémentation sur une variable.

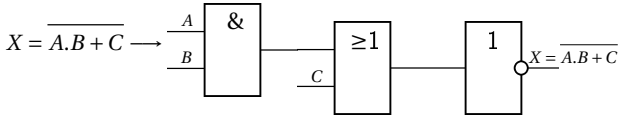
De plus, on a vu dans les sections 5.1.1.b et 5.1.1.c que cette notation est utilisée pour numéroter les cases des diagrammes de Veitch et de Karnaugh, et faciliter ainsi la représentation d'une fonction sous forme de diagramme.

5.1.3 Représentations graphiques

5.1.3.a Logigramme

Un logigramme est un schéma illustrant l'expression d'une fonction logique sans tenir compte des constituants technologiques.

▷ **Exemple 5.4**



► **Remarque 5.3**

Notation : *Par convention, une entrée ou une sortie d'opérateur logique active à un niveau haut sera notée a, b, sel, etc.*

Une entrée ou une sortie d'opérateur logique active à un niveau bas sera notée $\overline{c}$*,* $\overline{d}$*,* $\overline{MEM}$*, etc.*

5.2 Simplification d'expressions logiques

À venir ...

5.2.1 Formes canoniques d'une fonction logique

À venir ...

5.2.2 Méthode algébrique

Il n'est pas facile de trouver le résultat minimal → application des théorèmes de De Morgan, factorisation, astuce, ...

▷ **Exemple 5.5**

$x+\overline{x}y = x(1+y)+\overline{x}y = x+xy+\overline{x}y = x+y$ (théorème d'allègement)

$x.(x+y) = x+xy = x$ (absorption)

$ABC + A\overline{B}C + AB\overline{C} + \overline{A}BC = AC + AB + BC$

5.2.3 Simplification par diagramme de Karnaugh

La méthode de simplification d'une fonction par diagramme de Karnaugh s'appuie sur l'adjacence entre les termes de la fonction pour en extraire la représentation la plus simple possible.

Les diagrammes de Karnaugh contiennent des ensembles de termes (« 0 » ou « 1 ») nommés implicants. Ces ensembles sont des :
- implicants simples lorsqu'il s'agit de termes isolés ;
- implicants majeurs lorsqu'il s'agit d'ensembles contenant 2^n termes aussi grands que possible ;
- implicants majeurs essentiels lorsque les termes considérés ne sont présents dans aucun autre implicant ;
- implicants majeurs non essentiels lorsqu'un terme est présent dans plusieurs implicants.

5.2.3.a Simplification par extraction des sommes de produits

La méthode est la suivante :

1. dessiner la table de Karnaugh correspondant à la fonction ;

- on entame les « 1 » isolés ;
- on réunit les octets de « 1 » adjacents ;
- on réunit les quartets de « 1 » adjacents ;
- on réunit les doublets de « 1 » adjacents pour réunir tous les « 1 » du tableau ;

2. identifier tous les implicants majeurs essentiels pour les « 1 » ;
3. identifier tous les implicants majeurs non essentiels pour les « 1 » ;
4. pour tous les implicants majeurs essentiels et un des implicants majeurs non essentiels sélectionné dans chaque ensemble, déterminer les termes de produits correspondant ;
5. effectuer l'addition logique de tous les termes précédents, sachant que :
   - un octet de « 1 » permet d'éliminer les 3 variables qui se trouvent sous les deux formes (complémenté et non complémenté) ;
   - un quartet de « 1 » permet d'éliminer les 2 variables qui se trouvent sous les deux formes (complémenté et non complémenté) ;
   - un doublet de « 1 » permet d'éliminer la variable qui se trouve sous les deux formes (complémenté et non complémenté) ;

▷ Exemple 5.6

Simplifier la fonction :

$\mathcal{F}(A,B,C) = \sum m(0,1,4,5) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$

Solution :

$\mathcal{F}$	A			
	C			
	1 (0)	1 (1)	1 (5)	1 (4)
B	0 (2)	0 (3)	0 (7)	0 (6)

$\begin{cases} \mathcal{F}_1 = (0,1,4,5) \\ \mathcal{F}_0 = (2,3,6,7) \end{cases}$

- l'implicant majeur essentiel est $\overline{B}$
- il n'y a aucun implicant majeur non essentiel

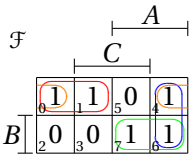
La solution est $\mathcal{F}(A,B,C) = \overline{B}$

▷ Exemple 5.7

Simplifier la fonction :

$\mathcal{F}(A,B,C) = \sum m(0,1,4,6,7) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.B.\overline{C} + A.B.C$

Solution :



$$\begin{cases} \mathcal{F}_1 = (0,1,4,6,7) \\ \mathcal{F}_0 = (2,3,5) \end{cases}$$

– les implicants majeurs essentiels sont $\overline{A}.\overline{B}$ et $A.B$

– les implicants majeurs non essentiels sont $\overline{B}.\overline{C}$ ou $A.\overline{C}$

La solution est $\mathcal{F}(A,B,C) = A.B + \overline{A}.\overline{B} + \overline{B}.\overline{C}$

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

Normalement, l'utilisation des tableaux de Karnaugh pour la simplification par extraction des sommes de produits exploite l'adjacence entre les « 1 » pour représenter la fonction à simplifier. Cependant, il est possible d'utiliser les « 0 » en procédant exactement de la même manière : on obtiendra alors une représentation de la fonction complémentée.

5.2.3.b Simplification par extraction des produits de sommes

La méthode est la suivante :

1. dessiner la table de Karnaugh correspondant à la fonction ;
   – on entame les 0 isolés ;
   – on réunit les octets de 0 adjacents ;
   – on réunit les quartets de 0 adjacents ;
   – on réunit les doublets de 0 adjacents pour réunir tous les 1 du tableau ;
2. identifier tous les implicants majeurs essentiels pour les « 0 » ;
3. identifier tous les implicants majeurs non essentiels pour les « 0 » ;
4. pour tous les implicants majeurs essentiels et un des implicants majeurs non essentiels sélectionné dans chaque ensemble, déterminer les termes de sommes correspondant ;
5. effectuer l'addition logique de tous les termes précédents, sachant que :
   – un octet de 0 permet d'éliminer les 3 variables qui se trouvent sous les deux formes (complémenté et non complémenté) ;
   – un quartet de 0 permet d'éliminer les 2 variables qui se trouvent sous les deux formes (complémenté et non complémenté) ;

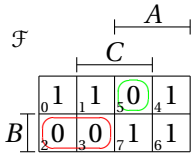
– un doublet de 0 permet d'éliminer la variable qui se trouve sous les deux formes (complémenté et non complémenté) ;

▷ Exemple 5.8

Simplifier la fonction :

$\mathcal{F}(A,B,C) = \prod M(2,3,5) = A.\overline{B}.C + A.\overline{B}.\overline{C} + \overline{A}.\overline{B}.\overline{C}$

Solution :



$\begin{cases} \mathcal{F}_1 = (0,1,4,6,7) \\ \mathcal{F}_0 = (2,3,5) \end{cases}$

– les implicants majeurs essentiels sont $A + \overline{B}$ et $\overline{A} + B + \overline{C}$
– il n'y a aucun implicant majeur non essentiel

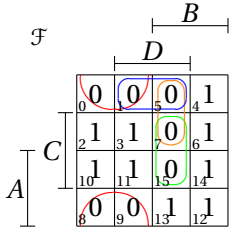
La solution est $\mathcal{F}(A,B,C) = (A + \overline{B}).(\overline{A} + B + \overline{C})$

▷ Exemple 5.9

Simplifier la fonction :

$\mathcal{F}(A,B,C) = \prod M(0,1,5,7,8,9,15)$

Solution :



$\begin{cases} \mathcal{F}_1 = (2,3,4,6,10,11,12,13,14) \\ \mathcal{F}_0 = (0,1,5,7,8,9,15) \end{cases}$

– les implicants majeurs essentiels sont $B + C$ et $\overline{B} + \overline{C} + \overline{D}$
– les implicants majeurs non essentiels sont $A + \overline{B} + \overline{D}$ ou $A + C + \overline{D}$

La solution est $\mathcal{F}(A,B,C) = (B + C).(\overline{B} + \overline{C} + \overline{D}).(A + \overline{B} + \overline{D})$

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

Normalement, l'utilisation des tableaux de Karnaugh pour la simplification par extraction des produits de sommes exploite l'adjacence entre les « 0 » pour représenter la fonction à simplifier. Cependant, il est possible d'utiliser les « 1 » en procédant exactement de la même manière : on obtiendra alors une représentation de la fonction complémentée.

5.2.3.c Cas des états indéterminés ou indifférents

Dans le cas général, l'utilisation des « 1 » ou des « 0 » doit conduire à des fonctions équivalentes (l'une étant la complémentaire de l'autre), même si les écritures peuvent être différentes. Cependant, il faut considérer avec attention le cas particulier des fonctions non complètement définies.

Certaines fonctions logiques sont données comme étant incomplètes (avec des états indéterminés) ou avec des états indifférents (combinaisons de variables d'entrées n'influençant pas le résultat). Ces conditions permettent de simplifier le tableau de Karnaugh, et par là-même, l'implantation de la fonction sous forme matérielle.

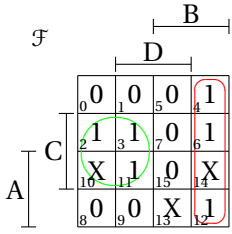
En plus des ensembles de « 0 » et des ensembles de « 1 », il y a donc également des ensembles de « X » ou « - » qui représentent les états indéterminés/indifférents de la fonction à minimiser. Ces états « X » ou « - » peuvent être rassemblés indifféremment avec des « 0 » ou « 1 » pour simplifier la minimisation logique dans les tableaux de Karnaugh.

Ainsi :

- les cases non définies d'un diagramme de Karnaugh peuvent être exploitées dans une simplification par les « 1 » comme dans une simplification par les « 0 » ;
- en conséquence, une même case pourra avoir été utilisée à la fois dans la représentation directe de la fonction, et dans sa représentation complémentée ;
- ainsi, si les deux représentations obtenues sont toutes deux justes, elles ne sont en aucun cas identiques, ni même équivalente : les fonctions sont différentes, bien que correspondant toutes deux au même diagramme de Karnaugh.

▷ **Exemple 5.10**

Soit le tableau de Karnaugh suivant à simplifier sous forme de somme de produit :



$$\begin{cases} \mathcal{F}_1 = (2,3,4,6,11,12) \\ \mathcal{F}_0 = (0,1,5,7,8,9,15) \\ \mathcal{F}_X = (10,13,14) \end{cases}$$

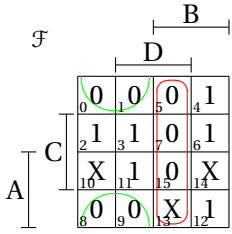
Solution :

- les implicants majeurs essentiels sont $B.\overline{D}$ et $\overline{B}.C$
- il n'y a aucun implicant majeur non essentiel

La solution $\sum\prod$ est $\mathcal{F}(A,B,C) = B.\overline{D} + \overline{B}.C$

▷ Exemple 5.11

Soit le tableau de Karnaugh suivant à simplifier sous forme de produit de somme :



$$\begin{cases} \mathcal{F}_1 = (2,3,4,6,11,12) \\ \mathcal{F}_0 = (0,1,5,7,8,9,15) \\ \mathcal{F}_X = (10,13,14) \end{cases}$$

Solution :

- les implicants majeurs essentiels sont $B + C$ et $\overline{B} + \overline{D}$
- il n'y a aucun implicant majeur non essentiel

La solution $\prod\sum$ est $\mathcal{F}(A,B,C) = (B + C).(\overline{B} + \overline{D})$

5.3 Simplifications par méthodes algorithmiques

Au delà de 6 variables, on utilise des méthodes algorithmiques.

5.3.1 Algorithme de Quine–McCluskey [Wiki01]

L'algorithme de Quine–McCluskey (ou méthode des implicants majeurs) est une méthode utilisée pour la minimisation de fonctions booléennes développée par Willard Van Orman Quine et Edward J. McCluskey.

Il est fonctionnellement identique à la méthode du tableau de Karnaugh, mais sa forme tabulaire le rend plus efficace lors d'une utilisation dans des algorithmes informatiques, et il fournit également un moyen déterministe de vérifier que la forme minimale d'une fonction booléenne a été atteinte.

La méthode comporte deux étapes :

1. trouver tous les implicants majeurs de la fonction ;
2. utiliser ces implicants majeurs dans un tableau pour trouver les implicants majeurs essentiels de la fonction, ainsi que les autres implicants majeurs nécessaires pour couvrir la fonction.

5.3.1.a Complexité

Bien qu'il soit plus pratique que les tableaux de Karnaugh pour manipuler des fonctions comportant plus de quatre variables, l'algorithme de Quine–McCluskey offre une étendue d'utilisation limitée puisque le problème qu'il résout est NP-complet : le temps de calcul de l'algorithme de Quine–McCluskey croît exponentiellement avec le nombre d'entrées.

Il peut être montré que pour une fonction de n variables, la valeur limite supérieure pour le nombre d'implicants majeurs est $3n/n$. Si $n = 32$ il peut y avoir plus de $6,5 \times 10^{15}$ implicants majeurs. Les fonctions possédants un nombre de variables important doivent être minimisées à l'aide de méthodes heuristiques potentiellement non-optimales, pour lesquelles l'heuristique de minimisation logique Espresso constitue le standard mondial mondial *de facto*.

▷ **Exemple 5.12**

Étape 1 : trouver les implicants majeurs

On veut minimiser la fonction arbitraire suivante :

$\mathscr{F}(A,B,C,D) = \sum m(4,8,10,11,12,15) + d(9,14)$

	A	B	C	D	$\mathscr{F}$
m0	0	0	0	0	0
m1	0	0	0	1	0
m2	0	0	1	0	0
m3	0	0	1	1	0
m4	0	1	0	0	1
m5	0	1	0	1	0
m6	0	1	1	0	0
m7	0	1	1	1	0
m8	1	0	0	0	1
m9	1	0	0	1	x
m10	1	0	1	0	1
m11	1	0	1	1	1
m12	1	1	0	0	1
m13	1	1	0	1	0
m14	1	1	1	0	x
m15	1	1	1	1	1

On peut facilement former l'expression de la somme canonique à partir de cette table simplement en additionnant les mintermes (en ignorant les termes X, *don't care*) là où la fonction est égal à 1.

$\mathscr{F}_{A,B,C,D} = \overline{A}.B.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D} + A.B.C.D$

Bien sûr, ce n'est certainement pas minimal. Ainsi, pour l'optimiser, tous les mintermes qui valent 1 sont d'abord placés dans une table de mintermes. Les termes *don't care* sont également ajoutés à cette table, de façon à pouvoir être combinés aux mintermes :

Nb 1	min-t.	Représentation binaire
1	m4	0100
	m8	1000
2	m9	1001
	m10	1010
	m12	1100
3	m11	1011
	m14	1110
4	m15	1111

À ce point, on peut commencer à combiner les mintermes entre eux. Si deux termes diffèrent d'un seul bit, ce bit peut être remplacé par un tiret indiquant que le bit est quelconque. Les termes qui ne peuvent être combinés avec aucun autre sont précisés par une astérisque (*). Lorsque l'on passe de la taille 2 à la taille 4, le tiret '-' est traité comme étant une troisième valeur de bit.

Par exemple, -110 et -100 peuvent être combinés, mais pas -110 et 011-. L'astuce est de faire correspondre d'abord les '-'.

Nb 1	min-t.	0-Cube
1	m4	0 1 0 0
	m8	1 0 0 0
2	m9	1 0 0 1
	m10	1 0 1 0
	m12	1 1 0 0
3	m11	1 0 1 1
	m14	1 1 1 0
4	m15	1 1 1 1

Implicants de taille 2	
m(4,12)	- 1 0 0 *
m(8,9)	1 0 0 -
m(8,10)	1 0 - 0
m(8,12)	1 - 0 0
m(9,11)	1 0 - 1
m(10,11)	1 0 1 -
m(10,14)	1 - 1 0
m(12,14)	1 1 - 0
m(11,15)	1 - 1 1
m(14,15)	1 1 1 -

Implicants de taille 4	
m(8,9,10,11)	1 0 - - *
m(8,10,12,14)	1 - - 0 *
m(10,11,14,15)	1 - 1 - *

Étape 2 : table des implicants majeurs

Aucun des termes de peut plus être combiné à aucun autre, ainsi devons nous construire une table des implicants majeurs essentiels. Les lignes représentent les implicants majeurs générés précédements, et les colonnes les mintermes spécifiés plus haut. Les termes *don't care* ne sont pas placés parmi les implicants – ils sont omis de cette section parce qu'ils ne constituent pas des entrées nécessaires.

	4	8	10	11	12	15	
m(4,12)*	X				X		- 1 0 0
m(8,9,10,11)		X	X	X			1 0 - -
m(8,10,12,14)		X	X		X		1 - - 0
m(10,11,14,15)*			X	X		X	1 - 1 -

Ici, chacun des implicants majeurs essentiels a été marqué d'une astérisque – le second implicant majeur peut être « couvert » par le troisième et le quatrième, et le troisième implicant majeur peut être « couvert » par le second et le premier, et ne sont donc plus essentiels. Si un implicant majeur est essentiel, comme ce à quoi on s'attend, il est alors nécessaire de l'inclure dans l'équation booléenne minimisée. Dans certains cas, les implicants majeurs essentiels ne couvrent pas tous les mintermes, auquel cas des procédures additionnelles peuvent être employées pour réduire la table. La procédure la plus simple serait de procéder par tests et erreurs, mais un moyen plus systématique est la méthode de Petricks. Dans cet exemple, les implicants majeurs essentiels ne prennent pas en compte tous les mintermes, et on peut donc dans ce cas combiner les implicants essentiels avec l'un des deux non-essentiels pour obtenir l'une de ces deux équations :

$\mathscr{F}_{A,B,C,D} = B.\overline{C}.\overline{D} + A.\overline{B} + A.C$

$\mathscr{F}_{A,B,C,D} = B.\overline{C}.\overline{D} + A.\overline{D} + A.C$

Ces deux équations finales sont fonctionnellement équivalentes à l'originale beaucoup plus coûteuse en surface :

$\mathscr{F}_{A,B,C,D} = \overline{A}B.\overline{C}.\overline{D}+A.\overline{B}.\overline{C}.\overline{D}+A.\overline{B}.\overline{C}D+A.\overline{B}.C.\overline{D}+A.\overline{B}C.D+A.B.\overline{C}.\overline{D}+A.B.C.\overline{D}+A.B.C.D$

5.3.2 Méthode de Petrick [Wiki02]

En algèbre booléenne, la méthode de Petrick est une technique permettant de déterminer toutes les solutions minimales de sommes de produits pour une table d'implicants majeurs. Cette méthode est très pénible pour des grands tableaux, mais elle est simple à implanter de façon informatique.

1. Réduire la table des implicants majeurs en éliminant les lignes d'implicants majeurs essentiels et les colonnes correspondantes.
2. Numéroter les lignes de la table réduite des implicants majeurs P_1, P_2, P_3, P_4, etc.
3. Construire une fonction logique P qui est vraie si toutes les colonnes sont couvertes. P consiste en un produit de sommes où chaque terme de somme a la forme $(P_{i0} + P_{i1} + \cdots + P_{iN})$, où chaque P_{ij} représente une ligne qui couvre la colonne i.
4. Réduire P à une somme de produits minimale en multipliant les implicants majeurs et en appliquant $X + X.Y = X$.
5. Chaque terme du résultat représente une solution, c'est-à-dire un ensemble de lignes qui couvrent tous les mintermes de la table. Pour déterminer les solutions minimales, trouver les termes qui contiennent un nombre minimum d'implicants majeurs.
6. Pour chacun des termes trouvés à l'étape précédente, compter le nombre de litéraux dans chaque implicant majeur et trouver le nombre total de litéraux.
7. Choisir le ou les termes composés du nombre total minimum de litéraux, puis écrire les sommes d'implicants majeurs correspondantes.

▷ **Exemple 5.13**

Nous voulons réduire la fonction suivante :

$\mathscr{F}(A,B,C,D) = \sum m(0,1,2,5,6,7)$

La table d'implicants majeurs obtenue par l'algorithme de Quine–McCluskey est la suivante :

		0	1	2	5	6	7
K (0,1)	$\overline{a}.\overline{b}$	X	X				
L (0,2)	$\overline{a}.\overline{c}$	X		X			
M (1,5)	$\overline{b}.c$		X		X		
N (2,6)	$b.\overline{c}$			X		X	
P (5,7)	$a.c$				X		X
Q (6,7)	$a.b$					X	X

En se basant sur les X de la table ci-dessus, construire un produit de sommes des lignes où chaque ligne est additionnée et les colonnes multipliées :

$(K+L)(K+M)(L+N)(M+P)(N+Q)(P+Q)$

Utiliser la règle de distributivité pour transformer cette expression en une somme de produits. Utiliser également les équivalence suivantes pour simplifier l'expression finale : $X+X.Y=X$ et $X.X=X$ et $X+X=X$

$=(K+L)(K+M)(L+N)(M+P)(N+Q)(P+Q)$
$=(K+LM)(N+LQ)(P+MQ)$
$=(KN+KLQ+LMN+LMQ)(P+MQ)$
$=KNP+KMNP+LMNP+LMPQ+KMNQ+KLMQ+LMNQ+LMQ$

Utiliser l'équivalence suivante pour réduire encore l'équation : $X+X.Y=X$

$=KNP+LMNP+LMQ+KMNQ$

Choisir les produits avec le moins de termes ; dans cet exemple, il y a deux produits avec trois termes : KNP et LMQ

Choisir le ou les termes avec le plus petit nombre de littéraux. Dans cet exemple, les deux produits se développent en un total de 6 littéraux chacun :

KNP se développe en $\overline{a}.\overline{b}+\underline{b}.\overline{c}+a.c$
LMQ se développe en $\overline{a}.\overline{c}+\overline{b}.c+a.b$

Ainsi, l'un comme l'autre peut être utilisé.

5.3.3 Heuristique de minimisation logique Espresso [Wiki03]

Le minimiseur logique Espresso est un programme informatique largement répandu utilisant une heuristique et des algorithmes spécifiques pour réduire efficacement la complexité des circuits logiques. Espresso a été développé chez IBM par Richard L. Rudell. Rudell publia plus tard la variante Espresso-MV (1986) sous le titre « *Multiple-Valued Logic Minimization for PLA Synthesis* » (Minimisation de logique multivaluée pour la synthèse de PLA).

5.3.3.a Introduction

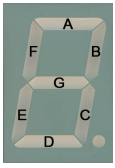
Conception des circuits logiques

Tous les systèmes numériques sont composés de deux fonctions élémentaires : des éléments mémoire pour stocker l'information et des circuits à portes logiques combinatoires pour traiter cette information. Les machines d'état, comme les compteurs, ne sont rien d'autre qu'une combinaison d'éléments mémoire et de circuits combinatoires. Puisque les circuits mémoire sont des composants standards devant être sélectionnés parmi un ensemble limité, la conception de fonctions numériques passe par l'implantation des circuits à porte combinatoire pour les blocs de base ainsi que l'interconnexion de tous ces blocs de base.

En général, l'implantation de circuits à portes logiques est nommée « synthèse logique », qui peut être faite manuellement, mais pour laquelle des méthodes informatiques formelles sont généralement appliquées. Les méthodes de conception de circuits combinatoires sont brièvement résumées ici.

Le point de départ de la conception des circuits logiques est la fonctionnalité souhaitée, obtenue à partir de l'analyse du système comme un tout, le circuit logique en étant une partie. La description peut être faite sous forme algorithmique ou sous forme d'équations logiques, mais peut également être résumée sous la forme d'une table. L'exemple ci-dessous montre le début d'une telle table représentant un convertisseur binaire → 7 segments :

Digit	Code	Segments A-G
0	0000	1 1 1 1 1 1 0
1	0001	0 1 1 0 0 0 0
2	0010	1 1 0 1 1 0 1
3	0011	1 1 1 1 0 0 1
.	....	.....



Le processus d'implantation commence avec la phase de minimisation logique, décrite ci-dessous, afin de simplifier la table de vérité en combinant les termes séparés en termes plus important contenant moins de variables.

Ensuite, le résultat minimisé peut être séparé en parties plus petites par une procédure de factorisation puis est finalement *mappé* sur les cellules logiques disponibles de la technologie cible. Cette opération est nommée « optimisation logique ».

Méthodes de minimisation classiques

Minimiser les fonctions booléennes à la main en utilisant les classiques diagrammes de Karnaugh est un processus laborieux, pénible et source d'erreurs. Ce n'est pas adapté pour des systèmes à plus de 6 variables d'entrée et utili-

sable en pratique jusqu'à seulement 4 variables, alors que les fonctions à plusieurs sorties sont encore plus complexes à gérer. De plus, cette méthode n'est pas adaptée à une implantation sous forme de programme informatique. Or, puisque les fonctions logiques actuelles ne sont généralement pas limitées à un si petit nombre de variables alors que le coût et le risque de faire des erreurs est prohibitif pour une implantation manuelle, l'utilisation d'ordinateurs devient indispensable.

La première méthode alternative à devenir populaire fut la méthode tabulaire développé par Quine et McCluskey. Partant de la table de vérité pour un ensemble de fonctions logiques, en combinant les mintermes pour lesquelles les fonctions sont actives – *ON-cover* – ou pour lesquelles la valeur de la fonction est sans objet – *DC-cover* – un ensemble d'implicants majeurs est composé. Enfin, une procédure systématique suit pour trouver le plus petit ensemble d'implicants majeurs avec lesquels la fonction peut-être réalisée.

Bien que l'algorithme Quine–McCluskey soit très bien adapté à une implantation informatique, le résultat est cependant loin d'être efficace en termes de temps de calcul et d'utilisation mémoire. L'ajout d'une variable à la fonction double chacun d'eux, parce que la longueur de la table de vérité augmente exponentiellement avec le nombre de variables. Un problème similaire se pose lorsque l'on augmente le nombre de fonctions de sortie d'un bloc combinatoire. Finalement, la méthode Quine–McCluskey est utilisable en pratique uniquement pour les fonctions comportant un nombre limité de variables d'entrée et de fonctions de sortie.

5.3.3.b Algorithme Espresso

Une approche radicalement différente de ce problème est suivie par l'algorithme Espresso, développé par Brayton à Berkeley, université de Californie. Plutôt que d'étendre la fonction logique en mintermes, le programme manipule des « cubes » représentant les termes de produits couvrant les 1 (*ON*), les X (*DC*) et les 0 (*OFF*) itérativement. Bien que le résultat de la minimisation ne soit pas garanti comme étant le minimum global, il en est en pratique une approximation très proche, alors que la solution est toujours sans redondance. Comparée aux autres méthodes, celle-ci est essentiellement plus efficace, réduisant l'utilisation mémoire et le temps de calcul par plusieurs ordres de magnitude. Son nom fait référence au fait de préparer instantanément une tasse de café. Il y a difficilement des restrictions au nombre de variables, de fonctions de sortie, et de termes de produits d'un bloc combinatoire. En général,

cela signifie que des dizaines de variables avec des dizaines de fonctions de sortie peuvent être traitées.

L'entrée d'Espresso est une table de fonction de la fonctionnalité désirée ; le résultat est une table minimisée, décrivant soit les 1, soit les 0 de la fonction, selon les options sélectionnées. Par défaut les termes de produits seront partagés autant que possible par les différentes fonctions de sortie mais le programme peut être configuré pour traiter chacune des fonctions de sortie séparément. Ceci permet une implantation efficace sur des réseaux logiques à deux dimensions tels que des PLA (*Programmable Logic Array*) ou des PAL (*Programmable Array Logic*).

L'algorithme Espresso s'est révélé si efficace qu'il est incorporé comme étape standard de minimisation des fonctions logiques dans virtuellement tout outil de synthèse logique actuel. Pour implanter une fonction en logique à plusieurs niveaux, le résultat de la minimisation est optimisé par factorisation puis *mappé* sur les cellules logiques disponibles de la technologie cible, que ce soit un FPGA (*Field Programmable Gate Array*) ou un ASIC (*Application Specific Integrated Circuit*).

Chapitre 6

Les circuits combinatoires

Augustus De Morgan
⋆ 27 juin 1806, Madura, Indes
† 18 mars 1871, Londres, R.-U.

$comp\left(\bigcap_j A_j\right) = \bigcup_j comp(A_j)$ i.e. le complément de l'intersection d'un nombre quelconque d'ensembles est égal à l'union de leurs compléments.

$comp\left(\bigcup_j A_j\right) = \bigcap_j comp(A_j)$ i.e. le complément de l'union d'un nombre quelconque d'ensembles est égal à l'intersection de leurs compléments.

6.1 Circuits logiques combinatoires usuels

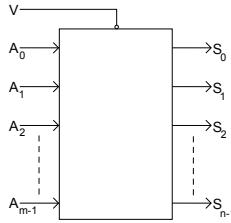
Un **Circuit combinatoire** est un circuit dont les sorties dépendent uniquement de la combinaison des états des entrées à l'instant de l'observation.

6.1.1 Circuits de transcodage (codeurs, décodeurs, convertisseurs)

6.1.1.a Codeur (encodeur)

Un codeur est un circuit à $M=2^N$ entrées et N sorties qui code en binaire le rang de la seule entrée active.

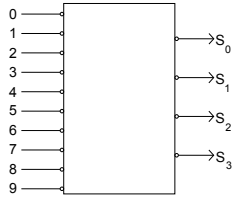
une seule entrée active à la fois



représentation en sortie sur N bits

▷ Exemple 6.1

Soit le codeur décimal-DCB à 10 entrées et 4 sorties suivant :

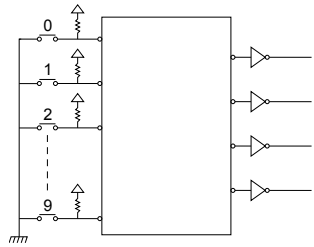


La table de vérité de ce codeur est la suivante :

$\overline{A_9}$	$\overline{A_8}$	$\overline{A_7}$	$\overline{A_6}$	$\overline{A_5}$	$\overline{A_4}$	$\overline{A_3}$	$\overline{A_2}$	$\overline{A_1}$	$\overline{A_0}$	$\overline{S_3}$	$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$
1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	0	1	0	0	0	1
1	1	1	1	1	1	1	0	1	1	0	0	1	0
1	1	1	1	1	1	0	1	1	1	0	0	1	1
1	1	1	1	1	0	1	1	1	1	0	1	0	0
1	1	1	1	0	1	1	1	1	1	0	1	0	1
1	1	1	0	1	1	1	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	0	0	1

On notera que ce codeur fonctionne en logique négative : l'unique entrée active est au niveau logique « 0 ».

Application pratique 6.1 : Codeur de clavier numérique



représentation DCB

► **Remarque 6.1**

Les codeurs de priorités sont une version modifiée du codeur : quand deux entrées sont actives, c'est l'entrée correspondant au nombre le plus haut qui est choisi.

6.1.1.b Décodeur

Le décodeur est un circuit qui établit la correspondance entre un code d'entrée sur N bits et M lignes de sortie ($M \leq 2^N$).

Pour chacune des combinaisons d'entrée, une seule ligne de sortie est validée.

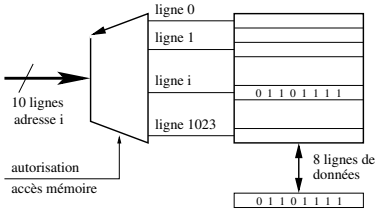
▷ **Exemple 6.2**

Décodeur DCB-décimal : 4 entrées, 10 sorties.

► **Remarque 6.2**

La plupart des décodeurs sont dotés d'une ou plusieurs entrées de validation qui commandent son fonctionnement.

Application pratique 6.2 : Adressage d'une mémoire



– une mémoire est un tableau d'éléments binaires (divisés en lignes et colonnes) ;
– pour lire un mot mémoire, il faut lui envoyer le numéro de ligne souhaité (adresse) ;
– souvent, le décodeur est interne à la mémoire.

Application pratique 6.3 : Génération de fonction

Toute fonction logique peut être réalisée à partir d'une combinaison de décodeur.

▷ **Exemple 6.3**

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

► **Remarque 6.3**

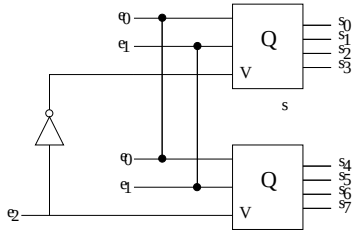
Il n'est pas nécessaire de simplifier la fonction avant la réalisation.

6.1.1.c Mise en cascade des décodeurs

Utilisation de l'entrée de validation.

▷ **Exemple 6.4**

Réaliser un décodeur à 3 entrées en utilisant 2 décodeurs à 2 entrées.



Réaliser un décodeur à 16 sorties à l'aide de décodeurs à 4 sorties.

6.1.1.d Transcodeurs (convertisseurs)

Circuit à p entrées et k sorties qui convertit un nombre écrit dans un code C1 en un nombre écrit dans un code C2.

▷ **Exemple 6.5**

Code binaire → code Gray
Code DCB → code affichage chiffre (décodeur 7 segments)

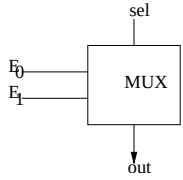
6.1.2 Multiplexeurs–démultiplexeurs

6.1.2.a Multiplexeurs (MUX)

Circuit à 2^n entrées d'informations, n entrées de sélection, et une sortie. Il permet l'aiguillage de l'une de ces entrées vers la sortie.

▷ **Exemple 6.6**

MUX à 2 entrées de données



E_1	E_0	sel	out
X	X	0	E_0
X	X	1	E_1

→ $S=\overline{sel}.E_0 + sel.E_1$

► **Remarque 6.4**

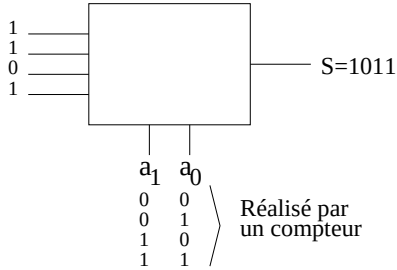
La table de vérité devient rapidement très importante (à partir de 4 entrées). On exprime alors la fonction de sortie directement

▷ **Exemple 6.7**

MUX à 4 entrées (→ 2 entrées de sélection a_1a_0) $S=\overline{a_1}.\overline{a_0}.E_0 + \overline{a_1}.a_0.E_1 + \cdots$

Application pratique 6.4 : Conversion parallèle–série

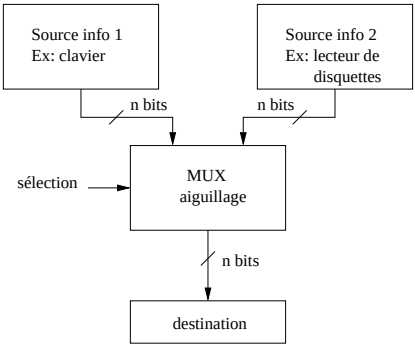
On place successivement les valeurs 00, 01, 10, 11 sur a_1a_0.



Application pratique 6.5 : Générateur de fonctions

Toute fonction logique peut être réalisée à partir des MUX. Les entrées de sélection (commande) sont alors les variables de la fonction.

Application pratique 6.6 : Sélection de mots



$$S = ABCE_0 + AB\overline{C}E_1 + \cdots + \overline{ABC}E_4 + \cdots$$

Le MUX est réalisé à partir de n MUX à 2 entrées travaillant avec la même commande de sélection.

► **Remarque 6.5**

Intérêt : il n'est pas nécessaire de simplifier la fonction avant de la réaliser.

▷ **Exemple 6.8**

$F = ABC + \overline{ABC}$
Utilisation de MUX 8 vers 1.

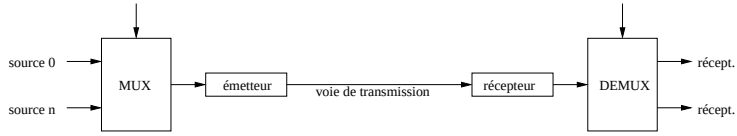
6.1.2.b Démultiplexeurs (DEMUX)

Circuit à 2^n sorties, 1 entrée d'information, n entrées de commande. Il permet l'aguillage d'information de l'entrée vers l'une des sorties.

► **Remarque 6.6**

Le MUX–DEMUX est un circuit programmable : les relations entre entrées et sorties sont modifiables.

Application pratique 6.7 : Transmission avec MUX/DEMUX



6.1.3 Le comparateur

Il détecte l'égalité entre deux nombres A et B. Certains circuits permettent également de détecter si A est supérieur ou bien inférieur à B.

6.1.3.a Comparateur de 2 éléments binaires

a_i	b_i	E_i	S_i	I_i
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

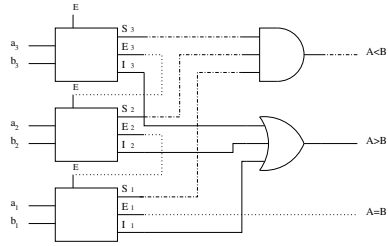
$E_i = \quad a_i = b_i \quad = \overline{a \oplus b}$

$S_i = \quad a_i > b_i \quad = a.\overline{b}$

$I_i = \quad a_i < b_i \quad = \overline{a}.b$

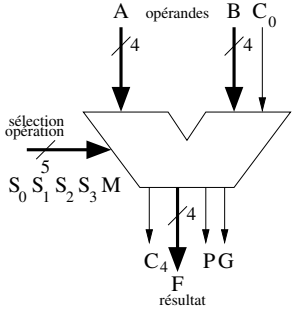
$D_i = \quad a_i \neq b_i \quad = a \oplus b$

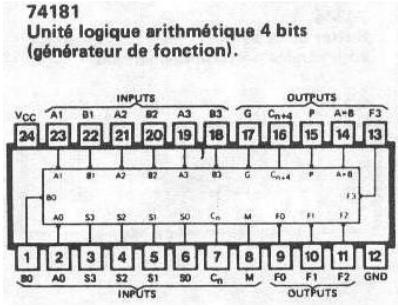
6.1.3.b Comparateur de 2 nombres



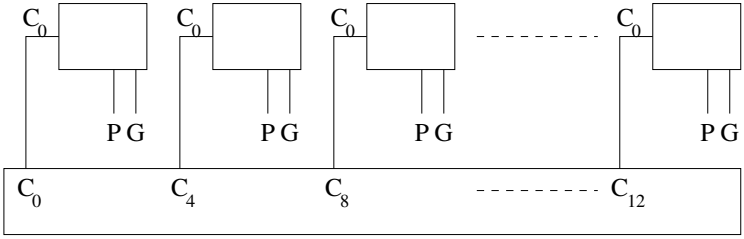
6.1.4 L'unité arithmétique et logique (UAL)

Utilisée dans pratiquement tous les systèmes informatiques, elle réalise des opérations arithmétiques (addition, soustraction, etc.) et logiques (`ET`, `OU`, etc.). C'est un circuit programmable : les relations entre les données en sortie et les données en entrée sont modifiables.





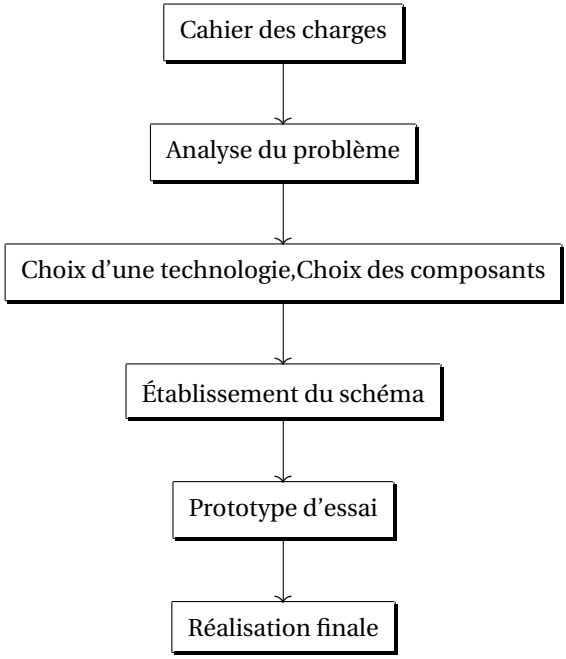
Les sorties P et G servent à la mise en cascade des ALUs, et donc au calcul de retenue anticipée.



Générateur de retenue anticipée

6.2 Synthèse des circuits combinatoires

6.2.1 Présentation



Si le nombre de variables mises en œuvre est faible (typiquement inférieur à 10), les circuits sont réalisés directement à l'aide de la table de vérité, éventuellement après simplification de la fonction logique. Dans le cas contraire, la fonction est décomposée en différents blocs fonctionnels analysés séparément.

Le choix des composants utilisés est basé sur différents critères : nombre de boitiers, coût, disponibilité, points test, complexité des connexions, etc.

Les différents choix sont :

a) utilisation de portes simples (`OU`, `ET`, `NON`) ou des portes `NON-OU` et `NON-ET` ;

b) développement de circuits intégrés (CI) spécialisés. Le problème du coût de développement et de fabrication impose une production en très grandes séries ;

c) utilisation de circuits intégrés combinatoires :
   – MUX, DEMUX ;
   – décodeurs ;
   – circuits logiques programmables : PROM, PAL, etc.

6.2.2 Circuits logiques programmables

6.2.2.a Introduction

La réalisation pratique d'un système logique dit « câblé » consiste à utiliser les composants CI disponibles sur le marché. Cela oblige le concepteur à décomposer un système donné en blocs fonctionnels proposés par les constructeurs et à optimiser son choix.

L'apparition des circuits adaptables dits « programmables » par le constructeur ou l'utilisateur apporte une solution à ce problème.

6.2.2.b Structure des circuits logiques programmables

Toute fonction logique de n variables peut se mettre sous la forme d'une somme de produits. Cela implique que toute fonction logique peut être réalisée par l'utilisation d'une structure comportant deux ensembles fonctionnels :
– un ensemble d'opérateurs `ET` organisés sous forme de matrice permet de générer les produits des variables d'entrée ;
– un ensemble d'opérateurs `OU` permet de sommer les produits.
La programmation de ces circuits est possible grâce à des fusibles placés à chaque noeud, et consiste à griller les fusibles de manière à supprimer le contact entre les lignes.

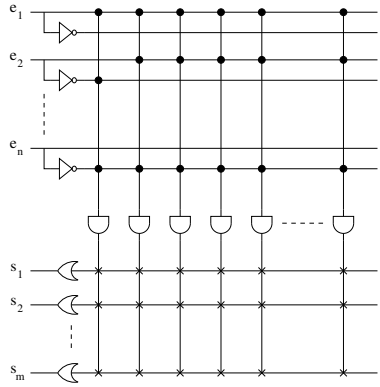
1. PROM (*Programmable Read-Only Memory*) ou PLE (*Programmable Logic Element*)

   Contrairement au FPLA dont les deux matrices sont programmables (*cf.* ¶ 3 page 102), les structures de type PROM voient leur matrice `ET` figée en usine, formant les 2^n fonctions possibles des n entrées. La matrice `OU` reste quant à elle entièrement programmable.
   – chaque sortie de la mémoire correspond à une fonction (sortie 3 états) ;

– la matrice ET correspond en fait à un décodeur $n \rightarrow 2^n$ (décodeur d'adresse) ;
– une fonction est réalisée en programmant sa table de vérité, c'est-à-dire en mettant en mémoire la valeur de f pour l'ensemble des combinaisons des entrées.

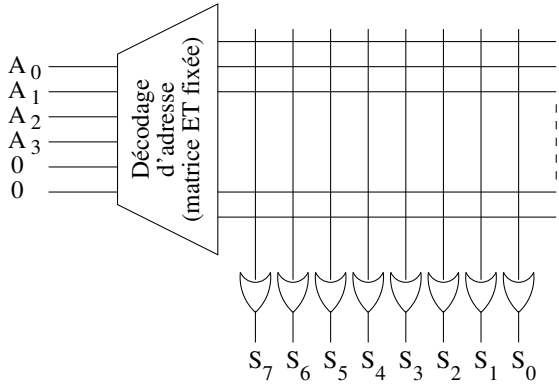

: interconnexion non programmée


: interconnexion programmée



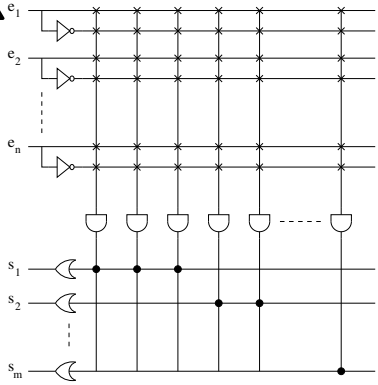
▷ **Exemple 6.9**

Réaliser le circuit $N \rightarrow N^2$ (N : nombre codé en DCB sur 4 bits) à l'aide de la PROM suivante (PROM à 6 entrées et 8 sorties → capacité de $2^6 = 64$ octets) :

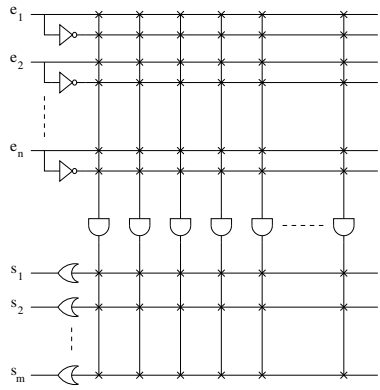


2. PAL (*Programmable Array Logic*)
   La structure des PAL est opposée à celle des PROM : la matrice `OU` est figée alors que la matrice `ET` est programmable.

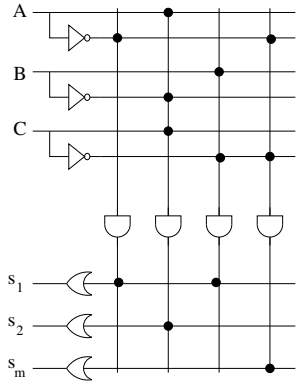
Les circuits PAL existent également en logique séquentielle.



3. FPLA (*Field Programmable Logic Array*) : matrice `OU` et `ET` programmable
   La structure des FPLA autorise une très grande souplesse dans la programmation. Par conséquent, c'est le circuit le plus souvent proposé pour la réalisation des fonctions logiques.



▷ **Exemple 6.10**



- $S_1 = \overline{A} + B.\overline{C}$
- $S_2 = A.B.\overline{C}$
- $S_m = \overline{A}.\overline{C}$

6.2.3 Programmation des circuits logiques programmables

- Les PROMs et PALs se programment assez facilement avec des « programmateurs universels » standards dans lesquels est incorporé un module spécifique pour chaque constructeur ;
- les FPLAs nécessitent des programmateurs plus sophistiqués à cause des doubles matrices à programmer.

Chapitre 7

Fonctions et opérateurs arithmétiques

Charles Babbage
⋆ 26 déc. 1791, Teignmouth, R.-U.
† 1871, London, R.-U.

«... I was sitting in the rooms of the Analytical Society, at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep, called out, "Well, Babbage, what are you dreaming about" to which I replied "I am thinking that all these tables" (pointing to the logarithms) "might be calculated by machinery." »
«... J'étais assis dans la salle de l'Analytical Society, à Cambridge, ma tête penchée vers la table dans une sorte d'humeur pensive, avec une table de logarithmes ouverte devant moi. Un autre membre, entrant dans la pièce et me voyant à moitié endormi, me demanda "Et bien, Babbage, à quoi rêvez-vous" ce à quoi je répondis "je pense que toutes ces tables" (montrant les logarithmes) "pourraient être calculées par une machine." »

(Charles Babbage)

Les systèmes combinatoires

▶Exercice 7.1

Développer et simplifier algébriquement les expressions booléennes suivantes :

- $F_1 = \overline{(x+\overline{y})}.(x+z)$
- $F_2 = (x.\overline{y}+z).(x+\overline{y}).z$
- $F_3 = (x+y).z+\overline{x}.(\overline{y}+z)+\overline{y}$
- $F_4 = bd+cd+\overline{c}d+\overline{a}\overline{b}cd+ab\overline{c}$
- $F_5 = ab\overline{c}+b.(a+\overline{c})+\overline{\overline{a}+b+\overline{a}c}$

▶Exercice 7.2

Faire le schéma des fonctions suivantes avec les portes indiquées :

- $x = \overline{abc}+\overline{c}d$ (3 portes NOR)
- $y = \overline{a(b+c)}$ (3 portes NAND)
- $z = \overline{abc}$ (3 NAND à 2 entrées)
- $f = a \oplus b$ (4 NAND à 2 entrées)

▶Exercice 7.3

Simplifier les expressions logiques suivantes :

- $F_1 = ab \oplus abcd$
- $F_2 = a \oplus (a+b)$
- $F_3 = a+(a \oplus b)$
- $F_4 = (a \oplus b) \oplus (\overline{a} \oplus \overline{c})$
- $F_5 = (a \oplus b) \oplus (\overline{a} \oplus \overline{b})$

▶Exercice 7.4

Chercher les formes canoniques des expressions suivantes :
- $F_1 = a \oplus (\overline{b} + c)$
- $F_2 = (a + \overline{c}).\overline{b} + (\overline{a} + c).b$

▶Exercice 7.5

Montrer algébriquement que $\overline{\overline{ab}} + bc + a\overline{c} = ab + \overline{\overline{bc}} + \overline{a}c$. Vérifier à l'aide d'un diagramme de Karnaugh.

▶Exercice 7.6

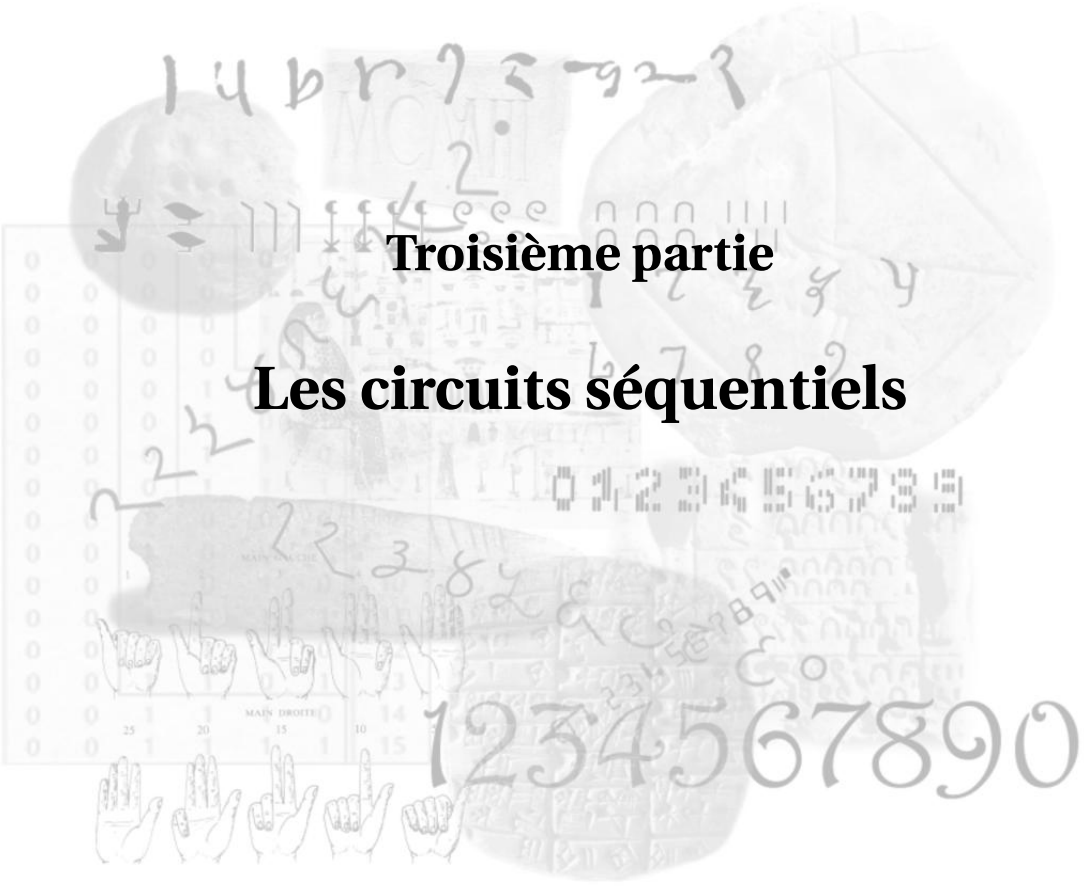
Simplifier cette expression à l'aide d'un diagramme de Karnaugh :
$F = \overline{a}(b \oplus c) + \overline{a}\,\overline{c}\,\overline{d} + \overline{a}d(b \oplus \overline{c}) + (a \oplus d)\overline{bc} + \overline{a}\,\overline{c}\,\overline{b \oplus d}$
Faire le schéma avec 2 portes dont un `XOR`.

▶Exercice 7.7

Une fonction $f(a,b,c,d)$ est incomplètement définie. On code ses états sur le mot binaire $abcd$,a représentant le poids fort. La fonction est vraie pour les états 0, 1, 3, 4, 6, A, B ; elle est fausse pour les états 7, 8, D, E. Tracer le diagramme de Karnaugh. Simplifier la fonction en vue d'une réalisation en portes NAND. Même question avec des portes NOR. Quelle est la meilleure solution ?

Troisième partie

Les circuits séquentiels



Chapitre 8

Les bascules

Alan Mathison Turing
⋆ 23 juin 1912, Londres, R.-U.
† 8 juin 1954, R.-U.

« [A universal machine] ... which can be made to do the work of any special-purpose machine, that is to say to carry out any piece of computing, if a tape bearing suitable "instructions" is inserted into it. »
« [Une machine universelle] ... qui peut être conçue pour faire le travail de n'importe quelle machine spécialisée, c'est-à-dire de procéder à n'importe quel fragment de calcul, si une bande comportant les "instructions" adaptées y est insérée. »
(Alan M. Turing, 1936, à propos de la « machine de Turing »)

8.1 Introduction

Circuit séquentiel : circuit dont l'état des sorties dépend non seulement des entrées mais également de l'état antérieur des sorties. Ces circuits doivent donc être capables de mémoriser.

▷ **Exemple 8.1**

$$1\begin{cases} M=0 \\ A=0 \end{cases} \rightarrow L=0 \qquad 2\begin{cases} M=1 \\ A=0 \end{cases} \rightarrow L=1$$

$$3\begin{cases} M=0 \\ A=0 \end{cases} \rightarrow L=1 \qquad 4\begin{cases} M=0 \\ A=1 \end{cases} \rightarrow L=0$$

$$5\begin{cases} M=0 \\ A=0 \end{cases} \rightarrow L=0$$

Dans un tel système, à une même combinaison des variables d'entrée ne correspond pas toujours la même valeur à la sortie (3 et 5). La fonctionnalité dépend de l'ordre des opérations (ordre de déroulement des séquence) → système séquentiel.

Les fonctions séquentielles de base sont :

– mémorisation ;
– comptage ;
– décalage.

Les circuits séquentiels fondamentaux sont :
– bascules (3 types) ;
– compteurs ;
– registres ;
– RAM (Random Access Memory).

Ces circuits peuvent travailler soit en mode synchrone, soit en mode asynchrone :
– mode asynchrone : À tout moment, les signaux d'entrée peuvent provoquer le changement d'état des sorties (après un certain retard qu'on appelle « temps de réponse ». Ces systèmes sont difficiles à concevoir et à dépanner.
– mode synchrone : Le moment exact où les sorties peuvent changer d'état est commandé par un signal d'horloge (train d'ondes carrées ou rectangulaires). Les changements d'état s'effectuent tous pendant une transition appelée « front » (montant ou descendant).

La majorité des systèmes numériques séquentiels sont synchrones même si certaines parties peuvent être asynchrone (ex. : reset).

Les avantages principaux du mode synchrone sont :
– préparer les entrées sans perturber les sorties ;
– protéger des parasites survenant en entrée.

Les bascules que l'on peut considérer comme des mémoires élémentaires, sont les briques de base des circuits séquentiels.

Ce sont les circuits de mémorisation les plus répandus dans les systèmes numériques en raison de leur rapidité de fonctionnement, de la facilité d'écriture et de lecture d'information, et de la grande simplicité de leur interconnexion avec des portes logiques.

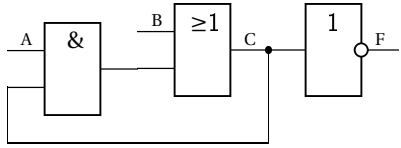
On trouve deux grandes familles de bascules :
– bascules de mémorisation : elles possèdent les commandes de mise à zéro, mise à un, mémorisation ;
– bascules de comptage : elles possèdent en outre une commande de changement d'état.

8.2 Point mémoire

La principale différence entre un système séquentiel et un système combinatoire est que lorsque l'on présente plusieurs fois de suite un même vecteur d'entrée à un système séquentiel, celui-ci – contrairement au système combinatoire – ne délivre pas nécessairement un le même vecteur de sortie à chaque fois.

En d'autres termes, l'état de la sortie d'un système séquentiel dépend non seulement de l'état des variables d'entrée, mais également du paramètre « temps », lequel paramètre est la plupart du temps concrétisé par « l'état antérieur » du système.

Soient le circuit et sa table de vérité associée suivants :



A	B	F_t
0	0	1
0	1	0
1	0	C_{t-1}
1	1	0

La sortie de la fonction F ci-dessus est dépendante d'une variable interne C. On peut en effet constater que l'état de la variable C dépend de l'état des entrées A Et B, mais également de son état antérieur : C mémorise donc liée aux entrées appliquées antérieurement au circuit. On constate sur ce circuit que l'effet de mémorisation est dû à la boucle de rétroaction présente entre la sortie du `OU` et l'entrée du `ET`. À cette boucle est associée la variable C qui constitue le point mémoire.

Définition 8.1

Circuit séquentiel : *un circuit séquentiel est un système bouclé permettant la conservation d'un état dépendant de la valeur des variables d'entrée ainsi que de l'état antérieur du système.*

La bascule constitue le système séquentiel de base et permet de mémoriser un élément d'information élémentaire appelé bit.

▶Exercice 8.1

Quel sera l'état de sortie du système F à l'issue des deux séquences (00, 10) et (01,10) ?

Nous avons brièvement présenté en introduction de ce chapitre ce qu'étaient les systèmes séquentiels synchrones et asynchrones. Une autre façon de décrire ces systèmes est donnée par les définitions 8.2 et 8.3 suivantes :

Définition 8.2

Système asynchrone : *un système séquentiel est asynchrone si à partir de l'instant où on applique un vecteur d'entrée, son évolution est incontrôlable de l'extérieur.*

Définition 8.3

Système synchrone : *un système séquentiel est synchrone si son évolution est contrôlable de l'extérieur par un signal d'horloge.*

8.3 Bascule RS

La bascule RS est le circuit séquentiel le plus simple. C'est une bascule asynchrone, et toutes les autres bascules, synchrones ou asynchrones, reposent sur cette bascule.

Son rôle consiste à mémoriser une information fugitive, selon le fonctionnement suivant : une apparition, même fugitive, de S entraîne un état stable Q=1, et une apparition, même fugitive, de R entraîne un état stable Q=0.

Symbole

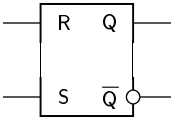


Tableau de Karnaugh

Q^+ / Qt	R=0, S=0	R=0, S=1	R=1, S=1	R=1, S=0
Qt=0	0 ($_0$)	1 ($_1$)	X ($_5$)	0 ($_4$)
Qt=1	1 ($_2$)	1 ($_3$)	X ($_7$)	0 ($_6$)

Diagramme temporel

Quand une impulsion est appliquée à 1 entrée pour imposer un certain état à la bascule, celle-ci demeure dans cet état, même après que l'impulsion ait disparu. Q garde son état lorsque S passe de 1 à 0 et lorsque R passe de 1 à 0.

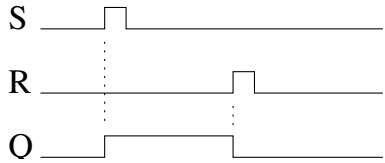


Table de vérité

S	R	Qt	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	X
1	1	1	X

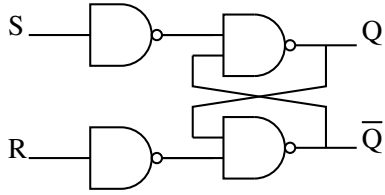
⟶

S	R	Q^+	
0	0	Q	→ mémorisation
0	1	0	→ mise à 0
1	0	1	→ mise à 1
1	1	X	→ interdit

Réalisation

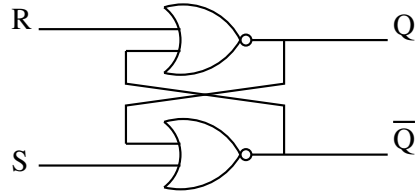
Si X= 1 → Q=S+$\overline{R}$.Q Les états indéterminés sont forcés à 1 : la bascule est dite à enclenchement prioritaire.

⟹ somme de produit ⇒ réalisation à l'aide de portes NAND.



Si X= 0 → Q= $\overline{R}$.(S+Q). Les états indéterminés sont forcés à 0 : la bascule est dite à déclenchement prioritaire.

⟹ produit de sommes ⇒ réalisation à l'aide de portes NOR.



► **Remarque 8.1**

Dans les deux cas, lorsqu'on passe de l'état (R,S)=(1,1) à (R,S)=(0,0) en passant soit par l'état stable correspondant à (R,S)=(1,0), soit par l'état stable correspon-

dant à (R,S)=(0,1), selon la rapidité relative des passages 0→1 de chacun des signaux, alors la sortie peut prendre aussi bien l'état $Q = 1$ *que* $Q = 0$.

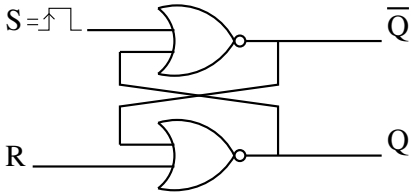
⇒ *il faut donc interdire la combinaison* $R = S = 1$ *afin de lever l'ambiguïté pour un état* $R = S = 0$ *venant après un état* $R = S = 1$.

Fonctionnement de la bascule avec des NOR

– quand R = S = 0, il y a deux possibilités et nous verrons que l'état pris par la bascule dépend des valeurs appliquées précédemment aux entrées :

$$\left.\begin{array}{l} -\text{ si } Q = 0 \xrightarrow{S=0} \overline{Q} = 1 \text{ et } Q = 0 \\ -\text{ si } Q = 1 \xrightarrow{S=0} \overline{Q} = 0 \text{ et } Q = 1 \end{array}\right\} \rightarrow \text{memorisation}$$

– Examinons si S = ⎍ et R = 0
Si Q = 0 à l'arrivée de l'impulsion sur S, alors S = 1 → $\overline{Q} = 0$ → Q = 1



Si Q = 1 à l'arrivée de l'impulsion sur S, alors S = 1 → $\overline{Q} = 0$ → Q reste à 1
⇒ l'application d'une impulsion de niveau haut sur S place la bascule dans l'état Q = 1.
→ opération de mise à 1 → SET

– Si on applique R= ⎍ et S= 0

$$\left.\begin{array}{l} \text{Si } Q = 0 \xrightarrow{R=⎍} Q = 0 \rightarrow \overline{Q} = 1 \\ \text{Si } Q = 1 \xrightarrow{R=⎍} Q = 0 \rightarrow \overline{Q} = 1 \end{array}\right\} \rightarrow R = ⎍$$

⇒ l'application d'une impulsion de niveau haut sur R place la bascule dans l'état Q = 0.
→ opération de mise à 0 → RESET

– R = S = 1
⇒ $Q = \overline{Q} = 0$
→ condition indésirable, puisque $\overline{Q}$ et Q doivent être l'inverse l'un de l'autre

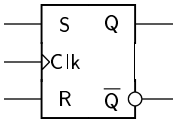
→ de plus, incertitude lorsque S et R reviennent à 0
→ R = S = 1 ne doit pas servir

L'avantage principal (unique ?) de la bascule RS est sa simplicité. Ses principaux inconvénients sont le fait qu'elle soit asynchrone, sa sensibilité aux parasites (tout bruit présent sur l'une des entrées de la bascule RS peut modifier l'état de la sortie), et le fait qu'il existe un état interdit pour R=S=1.

8.4 Bascule RS synchrone ou bascule RSH

La bascule RSH [1] est un bascule RS synchronisée par un signal d'horloge H. Lorsque H est au niveau bas, la bascule fonctionne comme une mémoire, et lorsque H est au niveau haut, la bascule fonctionne comme une bascule RS classique, et conserve donc les états interdits pour R=S=1.

Symbole



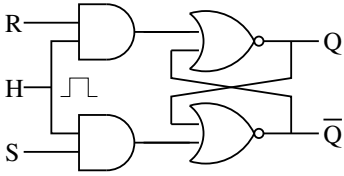
La sortie est indicée est vaut Q_N avant le front de l'horloge et Q_{N+1} après le front de l'horloge.

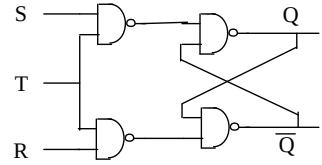
S et R n'influencent Q que lorsque l'horloge est au niveau haut.

Table de vérité

S	R	H	Q_{N+1}
X	X	0	Q_N
0	0	1	Q_N
0	1	1	1
1	0	1	0
1	1	X	X

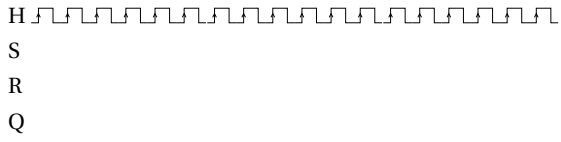
Réalisation





1. La bascule RSH est également appelée bascule RST ; on préférera néanmoins le terme RSH, plus explicite.

▷ **Exemple 8.2**



L'avantage de la bascule RSH par rapport à la bascule RS est sa sensibilité moindre aux parasites. Comme la bascule n'est sensible au bruit que lorsque l'horloge est au niveau haut, plus les états haut de l'horloge seront brefs, moins la bascule sera sensible.

8.5 Bascule à verrouillage (D-*latch*)

La D-*latch* est une bascule RSH pour laquelle on n'a conservé que les deux combinaisons RS=(0,1) et RS=(1,0). La D-*latch* a une seule entrée, nommée D.

Symbole

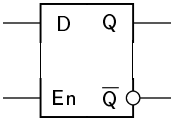
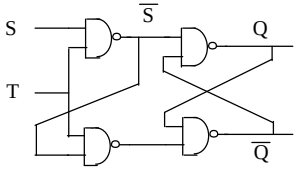


Table de vérité

D_N	En	Q_{N+1}	Mode
X	0	Q_N	verrouillé
0	1	0	transparent
1	1	1	transparent

Réalisation



Fonctionnement

La D-*latch* n'a pas d'état interdit et est transparente sur le niveau haut de l'horloge.

D = S et En = T.

- quand En=0 → l'entrée D n'a aucun effet et la bascule mémorise la valeur de la sortie (donc de l'entrée) au moment du passage de l'état 1 à l'état 0 de En → la bascule est verrouillée ;
- quand En=1 → Q suit les changements de D → la bascule est transparente.

► **Remarque 8.2**

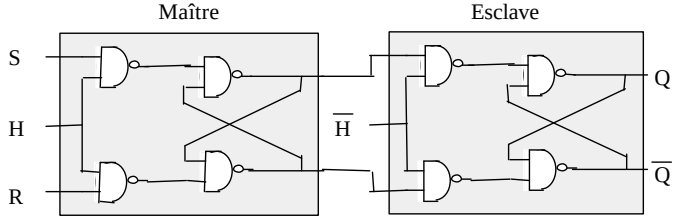
Notez l'absence du symbole ▷ sur l'entrée d'horloge.

8.6 Bascules maître-esclave

Les bascules maître-esclaves permettent de diminuer la sensibilité aux parasites en minimisant la période de transparence. La nature des bascules maître-esclave vient du fait que deux bascules RST montées en cascade et commandées par deux horloges en opposition de phase réalisent la même fonction qu'une seule bascule. La différence tient seulement au fait que la bascule ne fonctionne plus sur le niveau haut de l'horloge, mais sur son front descendant

- ☞ sur le niveau bas de l'horloge, le premier étage (maître) fonctionne en mode « mémorisation », et le deuxième étage (esclave) est en mode RS ;
- ☞ sur le niveau haut de l'horloge, le maître fonctionne en mode RS, et l'esclave est dans l'état « mémorisation »

La période pendant laquelle la bascule est sensible aux parasites se résume donc à la durée de commutation de l'horloge du niveau haut au niveau bas (front descendant).



8.7 Bascule JK

Les bascules JK sont des bascules maître-esclave fonctionnant seulement en mode synchrone. Elles sont plus polyvalentes que les bascules RS, car elles n'ont pas d'état ambigu et $R = S = 1 \rightarrow Q_{N+1} = \overline{Q_N}$.

Symbole

$$Q_{N+1} = J.\overline{Q_N} + \overline{K}.Q_N$$

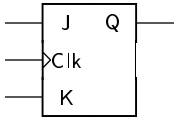


Tableau de Karnaugh

Q^+ / Qt \ J K	0	1
0	0 (0)	1 (5)
1	1 (2)	0 (7)

Q^+ / Qt \ J K	0	1
0	0 (0)	1 (4)
1	1 (3)	0 (6)

Sachant que les sorties sont toujours complémentaires, leur rebouclage sur les entrées élimine l'état interdit. Il n'y a pas d'inconvénient à ce rebouclage car les sorties de l'esclave ne changent d'état que lorsque le maître est bloqué. Les bascules JK sont très courantes dans les systèmes numériques

Cette bascule fonctionne toujours sur les fronts descendant.

Réalisation

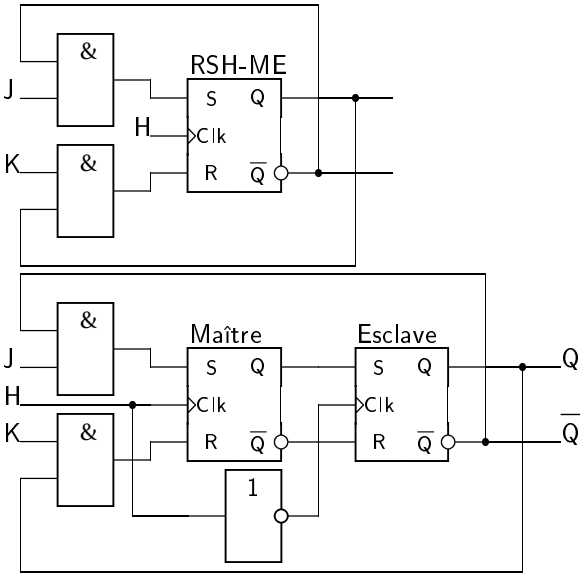


Table de vérité

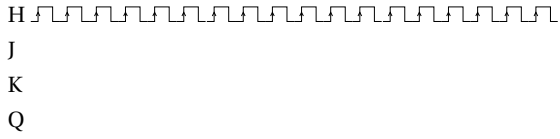
J	K	H	Q_{N+1}	
X	X	–	Q_N	
0	0	↓	Q_N	mémorisation
1	0	↓	1	forçage à 1
0	1	↓	0	forçage à 0
1	1	↓	$\overline{Q_N}$	commutation

J	K	Q_N	H	Q_{N+1}
X	X	X	–	Q_N
X	0	1	↓	1
X	1	1	↓	0
0	X	0	↓	0
1	X	0	↓	1

► **Remarque 8.3**

Pour que le basculement fonctionne, il faut avoir H très étroite, autrement il y a rebasculement.

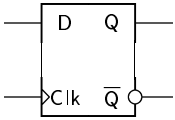
▷ **Exemple 8.3**



8.8 Bascule D synchrone

La bascule D est une bascule maître-esclave conçue sur le même principe que la JK. La bascule D est une bascule n'ayant qu'une seule entrée nommée D.

Symbole



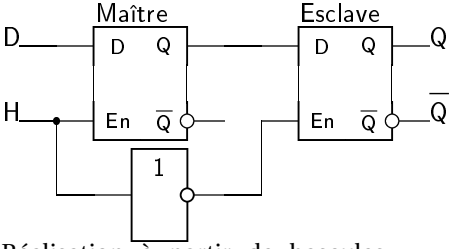
Le symbole de la bascule D est identique à celui de la D-*latch* à ceci près que l'entrée d'activation est remplacée par une entrée d'horloge, qui dispose donc du symbole associé.

Table de vérité

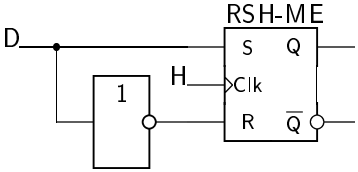
H	D_N	Q_{N+1}
↓	1	1
↓	0	0

Q_{N+1} prend la valeur de D_N après le front actif : $Q_{N+1} = D_N$
C'est une bascule de recopie : on l'emploie seulement en synchrone.

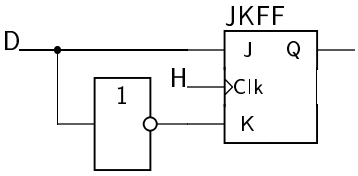
Réalisation



Réalisation à partir de bascules RSH maître-esclave :



Idem pour la réalisation à partir de bascules JK :



► **Remarque 8.4**

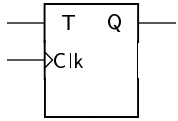
La sortie Q n'est égale à l'entrée D qu'à des moments bien précis → le signal Q est différent du signal D.

La bascule D fonctionne sur fronts d'horloge. En fait, la donnée d'entrée D est transférée à travers le maître lors du front montant et à travers l'esclave lors du front descendant. Pour fonctionner, cette bascule nécessite donc les deux front d'horloge. Différentes structures de bascules D existent, certaines pouvant transférer une donnée en ne recevant qu'un seul front d'horloge.

8.9 Bascule T

La bascule T s'obtient par exemple à partir d'une bascule JK dont on a relié les entrées J et K entre elles. Elle est utilisable uniquement en mode synchrone, et ne fonctionne qu'en commutation.

Symbole



Réalisation

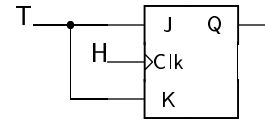
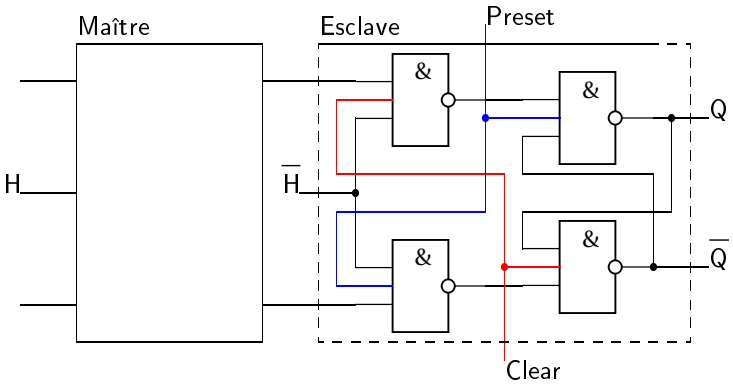


Table de vérité

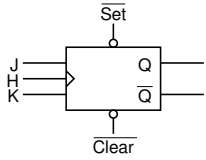
T	Q_{N+1}
0	Q_N
1	$\overline{Q_N}$

8.10 Entrées prioritaires asynchrones des bascules

La plupart des bascules synchrones possèdent des entrées prioritaires asynchrones. Elles agissent indépendamment de l'horloge et des entrées synchrones des bascules. Elles servent à forcer, à tout moment, la mise à 1 ou à 0 de la bascule, quelles que soient les conditions d'entrée. Elle agissent sur l'étage esclave des bascules.



▷ **Exemple 8.4**



$\overline{Set}$	$\overline{Clear}$	Q
1	1	fonctionnement normal
0	1	1
1	0	0
0	0	ambigu, interdit

Les entrées asynchrones peuvent être vraies à l'état bas (cas le plus fréquent) ou à l'état haut. En général, on applique juste une impulsion à ces entrées pour faire une initialisation.

Désignations synonymes :

Clear	RAZ	[c]et	DC clear
Preset	RAU	Set	DC set

► **Remarque 8.5**

Les entrées synchrones sont des niveaux de tension continue

8.11 Paramètres temporels des bascules

Pour qu'une bascule fonctionne correctement, il est nécessaire que le signal présent sur les entrées de la bascule (D ou JK) soit stabilisé depuis un certain temps lorsque le front d'horloge actif intervient (temps de « *setup* ») et reste stable pendant un certain temps après ce front d'horloge (temps de « *hold* » ou de maintien).

D'autre part, la commutation des sorties dune bascule se fait avec un certain temps de retard par rapport au signal qui a produit cette commutation (Hor-

loge, Reset ou Preset). Ces retards peuvent être différents selon le signal qui a produit la commutation, mais également selon que la commutation du signal de sortie est montante ou descendante. Ces retards seront notés TpLH et TpHL pour « Temps de Propagation *Low High* » et « Temps de Propagation *High Low* ».

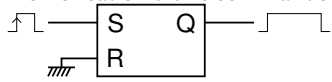
8.12 Applications des bascules

Application pratique 8.1 : Mémoire

→ mémorisation d'une information fugitive

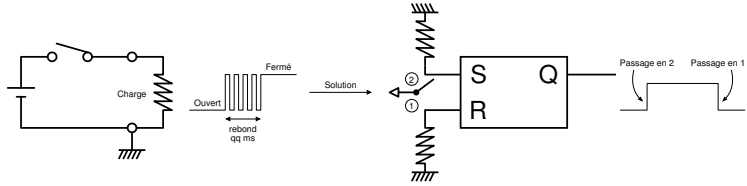
▷ **Exemple 8.5**

Mémorisation d'une commande de marche



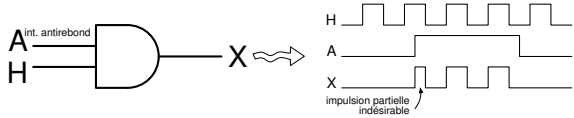
Application pratique 8.2 : Antirebond pour commutateur

▷ **Exemple 8.6**

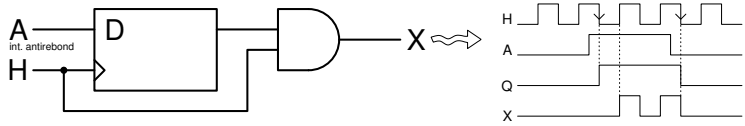


Application pratique 8.3 : Synchronisation

▷ **Exemple 8.7**

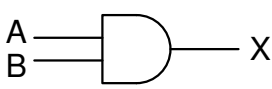


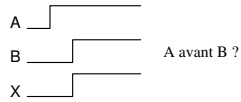
Solution :



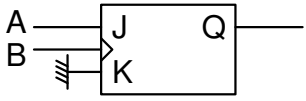
Application pratique 8.4 : Détection d'une séquence d'entrée

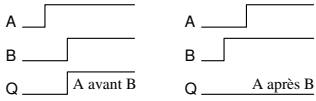
▷ Exemple 8.8





Solution :





→ détection du sens de rotation d'un moteur.

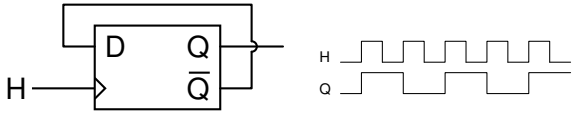
Application pratique 8.5 : Division de fréquence

La division de fréquence par 2 (et donc 2^N) peut être réalisée facilement à l'aide des différents registres.

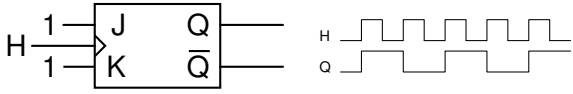
Bascule D

$D_N=Q_{N+1}$.

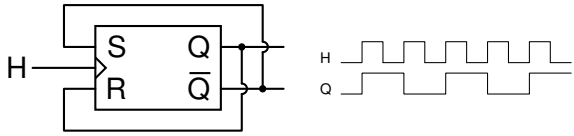
On veut $Q_{N+1}=\overline{Q}_N \Rightarrow D_N = \overline{Q}_N$



Bascule JK



Bascule RS



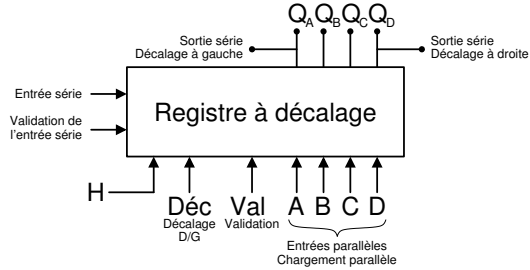
Chapitre 9

Registres : stockage et transfert de données

Howard Hathaway Aiken
⋆ 9 mars 1900, Hoboken, E.-U.
† 14 mars 1973, St Louis, E.-U.

[En 1964 Aiken reçoit le Harry M Goode Memorial Award, une médaille et $2,000 offert par la Computer Society] for his original contribution to the development of the automatic computer, leading to the first large-scale general purpose automatic digital computer.

Registre : ensemble de n bascules synchronisées permettant de stocker momentanément une information sur n bits.



9.1 Définition

Un registre est un circuit constitué de n bascules synchronisées permettant de stocker temporairement un mot binaire de n bits en vue de son transfert dans un autre circuit (pour traitement, affichage, mémorisation, etc.)

Le schéma d'un tel système comporte autant de bascules (de type D) que d'éléments binaires à mémoriser. Toutes les bascules sont commandées par le même signal d'horloge.

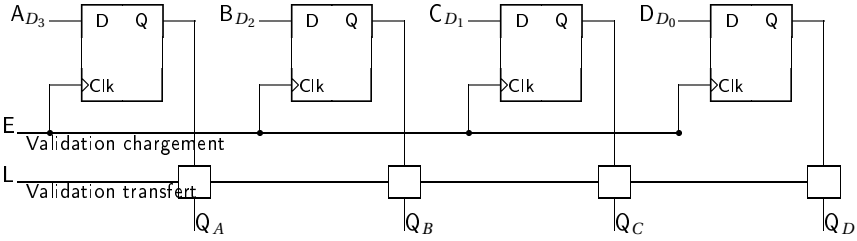
Moyennant une interconnexion entre les cellules (les bascules D), un registre est capable d'opérer une translation des chiffres du nombre initialement stocké. Le déplacement s'effectue soit vers la droite soit vers la gauche. Le registre est alors appelé « registre à décalage ».

Applications :
– conversion série-parallèle d'une information numérique ;
– opérations de multiplications et divisions par deux ;
– ligne à retard numérique ;
– mémoires à accès séquentiel

« Registre universel » : il résume les différentes entrées et sorties d'un registre à décalage procurant tous les modes de fonctionnement possibles.

9.2 Registre de mémorisation : écriture et lecture parallèles

Tous les bits du mot à traiter sont écrits (entrée écriture E=1), ou lus, (entrée lecture L=1), simultanément.



→ stockage en parallèle et transfert en parallèle d'un mot de 4 bits.

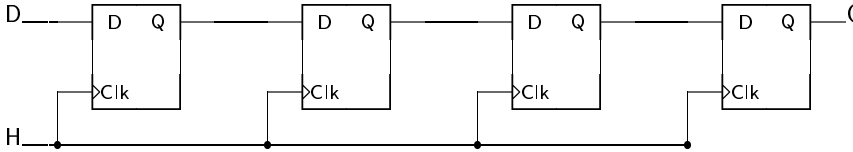
9.3 Registres à décalage

Comme son nom l'indique, un registre à décalage consiste à décaler bit par bit un mot binaire soit vers la gauche, soit vers la droite. Le registre à décalage peut être à écriture et à lecture série ou parallèle.

► **Remarque 9.1**

Un registre à décalage à droite peut être utilisé comme un diviseur par 2 alors qu'un registre à décalage à gauche peut être utilisé comme un multiplieur par 2.

9.3.1 Registre à écriture série et lecture série

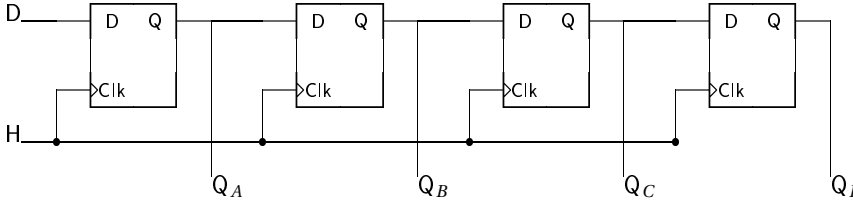


Après 4 pulsations de CLK, les 4 bits sont entrés dans le registre.

Après 4 autres cycles d'horloge, les 4 bits sont déplacés vers la sortie.

Leur application est essentiellement le calcul arithmétique binaire. CLK est alors l'entrée de décalage.

9.3.2 Registre à écriture série et lecture parallèle

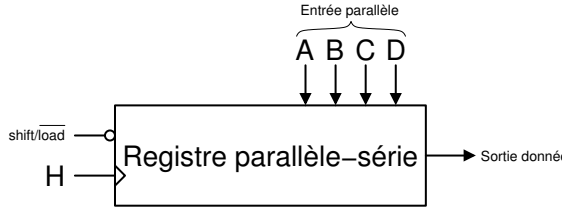


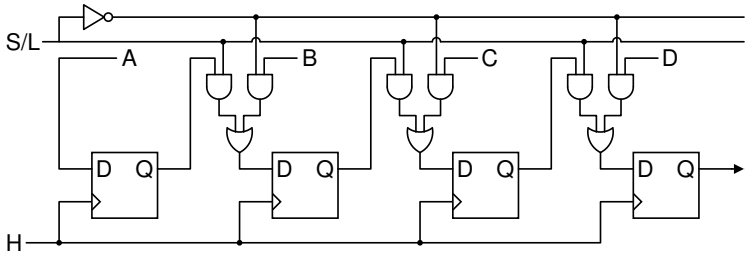
Lorsque l'entrée est stockée, chaque bit apparaît simultanément sur les lignes de sortie.

Le registre à décalage est utilisé comme convertisseur série-parallèle. Il est nécessaire à la réception lors d'une transmission série.

9.3.3 Registre à écriture parallèle et lecture série

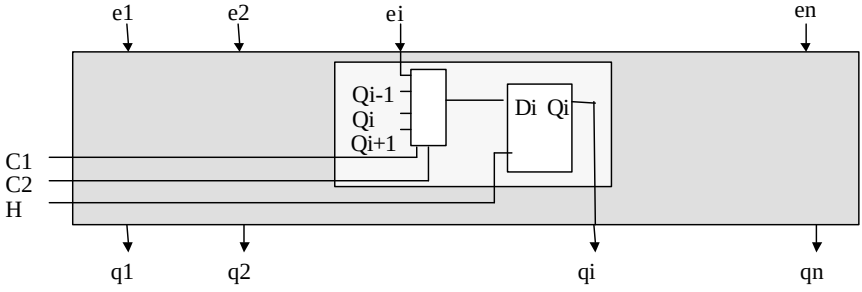
Utilisé comme convertisseur parallèle-série, il est nécessaire à l'émission lors d'une transmission série.





9.4 Registre universel

Le registre universel permet quatre modes de fonctionnement commandés par deux variables S_1 et S_2.



$$Di = \overline{C1}.\overline{C2}\ .ei + \overline{C1}.C2.Qi\text{-}1 + C1.\overline{C2}.Qi{+}1 + C1.C2.Qi$$

Ces entrées de sélection S_1 et S_2 sont en fait les entrées de sélection de multiplexeurs connectés aux entrées des bascules. Ces multiplexeurs à quatre entrées permettent donc quatre modes de fonctionnement :l'entrée D de chaque bascule est ainsi fonction du mode de fonctionnement désiré.

S_1	S_2	Mode
0	0	Chargement parallèle
0	1	Décalage à droite
1	0	Décalage à gauche
1	1	Inhibition de l'horloge

Chapitre 10

Les compteurs

Claude Elwood Shannon
⋆ 30 avr. 1916, Gaylord, E.-U.
† 24 fév. 2001, Medford, E.-U.

« The most important results [mostly given in the form of theorems with proofs] deal with conditions under which functions of one or more variables can be generated, and conditions under which ordinary differential equations can be solved. Some attention is given to approximation of functions (which cannot be generated exactly), approximation of gear ratios and automatic speed control. »

(Claude E. Shannon, Mathematical theory of the differential analyzer, 1941)

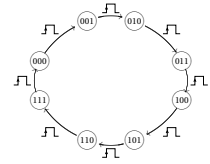
Définition 10.1

Compteur : *un compteur est un circuit séquentiel comportant n bascules décrivant au rythme d'une horloge un cycle de comptage régulier ou quelconque d'un maximum de 2^n combinaisons.*

Définition 10.2

État, Modulo : *la combinaison de sortie d'un compteur est appelé état, et le nombre d'états possibles d'un compteur est appelé modulo.*

Un compteur modulo N passera donc successivement par N états. Un compteur binaire naturel comptera donc de 0 à $N-1$. Le graphe suivant présente les différents états parcourus par un compteur modulo 8.



10.1 Compteur asynchrone (à propagation)

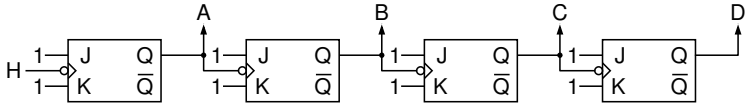
Nous avons vu dans la section § 8.5 page 126 comment réaliser une division par deux à l'aide de bascules JK. En cascadant des bascules JK montées en diviseurs de fréquence, on peut donc réaliser un compteur dont le modulo dépendra du nombre de bascules.

10.1.1 Compteur asynchrone à cycle régulier

▷ **Exemple 10.1**

Compteur asynchrone à 4 bits (compte de 0 à 15).

10.1.1.a Réalisation à l'aide de bascules JK



La sortie de chaque bascule agit comme le signal d'horloge de la suivante.

Fonctionnement

- J=K=1 ; toutes les bascules commutent sur des fronts descendants ;
- la bascule A commute à chaque front descendant du signal d'horloge ;
- la sortie de la bascule 1 sert d'horloge pour la bascule 2 → B commute chaque fois que A passe de 1 à 0 ;
- de la même manière, C commute lorsque B passe de 1 à 0, et D commute lorsque C passe de 1 à 0.

10.1.1.b Table d'implication séquentielle

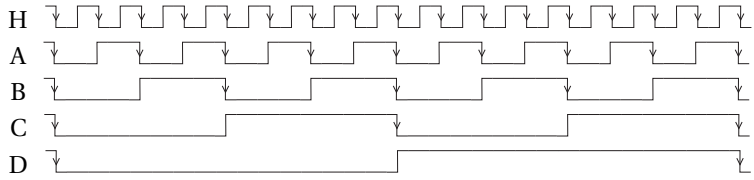
Elle montre les états binaires pris par les bascules après chaque front descendant.

N°	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
⋮	⋮	⋮	⋮	⋮
15	1	1	1	1
16	0	0	0	0
⋮	⋮	⋮	⋮	⋮

Si on imagine que DCBA représente un nombre binaire, le compteur réalise la suite des nombres binaires allant de 0000 à 1111 (soit de 0 à 15).

A près la 15ème impulsion, les bascules sont dans la condition 1111. Quand la 16ème impulsion arrive, le compteur affiche 0000 : un nouveau cycle commence.

10.1.1.c Chronogramme



→ chaque bascule divise par deux la fréquence d'horloge qui alimente son entrée CLK : $f_D = \frac{f_{initale}}{16}$.

Application : avec ce genre de circuit, on peut diviser la fréquence initiale par n'importe quelle puissance de 2.

10.1.1.d Modulo

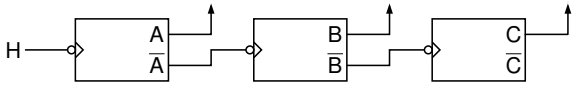
– c'est le nombre d'état occupés par le compteur pendant un cycle complet ;
– le modulo maximal d'un compteur à n bits (n bascules) est 2^n ;
– ex. : compteur 4 bits → 16 états distincts → modulo 16.

10.1.2 Décompteurs asynchrones

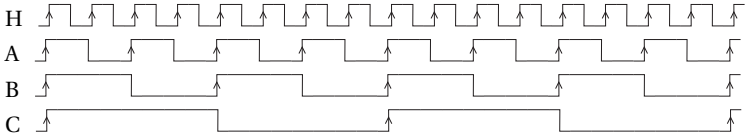
Il suffit de piloter chaque entrée CLK des bascules au moyen de la sortie complémentée de la bascule précédente.

▷ **Exemple 10.2**

Décompteur modulo 8



Chronogramme :



10.1.3 Compteur asynchrone à modulo $N < 2^n$ (à cycle régulier)

10.1.3.a Méthode

Pour réaliser un compteur ou un décompteur dont le cycle n'est pas une puissance de 2, la seule solution est d'agir sur l'entrée « Clear » lorsque la combinaison correspondant au modulo du compteur se produit sur les sorties de celui-ci.

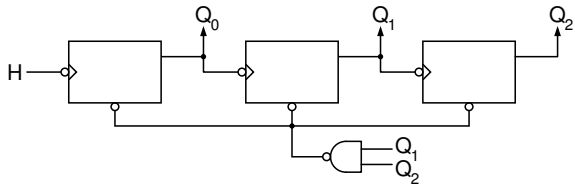
Ainsi, pour $2^{N-1} < N < 2^N$, on réalise un compteur modulo 2^n (avec n bascules), puis on raccourcit le cycle en jouant sur les entrées RAZ des bascules.

▷ **Exemple 10.3**

Compteur asynchrone modulo 6 : $2^2 < 6 < 2^3$ → on réalise un compteur modulo 8 avec 3 bascules, et on ramène le compteur à 000 dès que $Q_2Q_1Q_0 = 110$.

→ dès que la sortie de la porte NAND passe à 0, les bascules sont forcées à 0 : le compteur se remet à compter à partir de 0.

⇒ le compteur réalisé compte de 000 à 101 (de 0 à 5) puis recommence un nouveau cycle → modulo 6

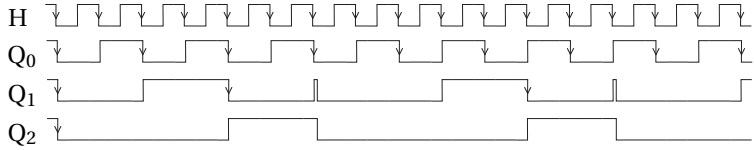


10.1.3.b Table d'implication séquentielle

N°	Q_2	Q_1	Q_0				
0	0	0	0				
1	0	0	1				
2	0	1	0				
3	0	1	1				
4	1	0	0				
5	1	0	1				
6	1	1	0	→	0	0	0

$Q_2Q_1Q_0 = 110$ est un état temporaire. Il existe mais pendant une dure très courte. C'est un état indésirable que l'on nomme parfois *glitch*.

10.1.3.c Chronogramme



► **Remarque 10.1**

Les sorties Q_2 et Q_1 ne sont pas des ondes carrées.

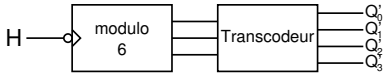
10.1.4 Comptage asynchrone dans un ordre quelconque (cycle irrégulier)

1ère méthode

On réalise un compteur de même modulo, puis on transcode ses sorties pour obtenir le cycle demandé.

▷ **Exemple 10.4**

Cycle 2, 5, 6, 8, 4, 10



N°	Q_2	Q_1	Q_0	Q'_3	Q'_2	Q'_1	Q'_0		
0	0	0	0	0	0	1	0	→	2
1	0	0	1	0	1	0	1	→	5
2	0	1	0	0	1	1	0	→	6
3	0	1	1	1	0	0	0	→	8
4	1	0	0	0	1	0	0	→	4
5	1	0	1	1	0	1	0	→	10

2ème méthode

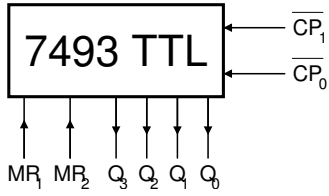
Utilisation des entrées RAZ et RAU.

▷ **Exemple 10.5**

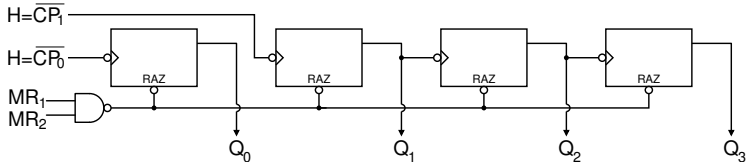
Cycle 0, 1, 2, 3, 5, 6, 8, 9, 11, 12, 15 : on réalise un compteur modulo 16 et on agit sur les RAU pour sauter les étapes.

10.1.5 Exemple de CI

Il existe de nombreuses puces en technologies TTL et CMOS. Parmi les plus populaires on trouve en TTL le 7493 qui est un compteur 4 bits, et en CMOS le 4024 qui est un compteur 7 bits.



Circuit interne



MR → Master Reset.

10.1.6 Inconvénients des compteurs asynchrones

Chaque bascule introduit un retard de D_p (D_p=25ns). Comme les bascules ne commutent pas sur le même signal d'horloge, les retards s'additionnent : à la $n^{\text{ième}}$ bascule, on a un retard T_m de $n \times D_p$.

Ainsi, la fréquence maximum de fonctionnement F_H d'un compteur modulo n, constitué de n bascules de délai de propagation D_p dépend du nombre de bascules du compteur et donc du modulo du compteur. Cette fréquence peut être établie comme suit :

$T_m = D_p \times n$: Délai de propagation du compteur

$T_H = 2 \times T_m$: Période min de l'horloge

$F_H = 1/(2 \times T_m)$: Fréquence max de l'horloge = $1/(2 \times n \times D_p)$

L'accumulation des retards des bascules implique une utilisation du compteur limitée en fréquence, particulièrement lorsque le nombre de bits est élevé, puisque le nombre de bascules augmente en même temps que le nombre de bits.

Les fronts des signaux appliqués sur les entrées d'horloge des bascules n'ayant pas lieu au même instant à cause des retards différents, les sorties ne changent pas d'état en même temps, ce qui implique un problème d'interface avec des circuits rapides (temps de lecture inférieur au retard entre plusieurs bits).

D'autre part, ces retards de commutation introduisent des états transitoires relativement conséquents, particulièrement lorsque le nombre de bascules traversées est important.

Mais l'inconvénient le plus important est lié au fait que cette structure nécessite de la logique sur des signaux asynchrones (l'horloge est générée par une bascule et le signal `Clear` est généré par une structure combinatoire). Cette logique combinatoire peut donc engendrer (ou propager) des états transitoires qui peuvent entraîner des dysfonctionnements du compteur.

10.2 Compteur synchrone (parallèle)

Toutes les bascules sont déclenchées en même temps par le même signal d'horloge. Ceci évite le problème du retard de propagation.

10.2.1 Réalisation

Elle est possible avec des bascules JK, D ou T.

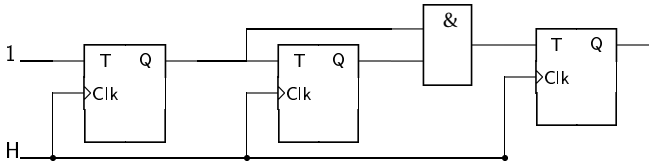
▷ **Exemple 10.6**

Réalisation d'un compteur modulo 8 (à cycle complet) à l'aide de bascules T

Table d'excitation

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	T_2	T_1	T_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

On constate que : $T_0 = 1$ et $T_1 = Q_0$ et $T_2 = Q_1 Q_0$



▷ **Exemple 10.7**

Réalisation d'un compteur synchrone décrivant le cycle 4, 9, 1, 3, 2.

a) À l'aide de bascules JK :

b) À l'aide de bascules D :

c) À l'aide de bascules T :

10.2.2 Exemples de circuit intégré

Compteur pré-réglable 74160 (74161, 74162, 74163)

– l'état initial du compteur est réglable à l'aide des entrées D_1, D_2, D_3, D_4 ;
– validation : elle permet de verrouiller le compteur.

Circuit	Comptage	Chargement	RAZ
74160	synchr. DCB	synchrone	asynchrone
74161	synchr. bin.	synchrone	asynchrone
74162	synchr. DCB	synchrone	synchrone
74163	synchr. bin.	synchrone	synchrone

– RAZ synchrone : indépendant de l'horloge.
– RAZ asynchrone : 000 est obtenu au coup d'horloge suivant l'instant ou clear est porté à l'état actif 0.

Compteur réversible pré-réglable 74193

– MR : entrée de réinitialisation asynchrone
– $Q_0 \ldots Q_3$: sorties des bascules
– $\overline{P_0} \ldots P_3$: entrée des données parallèles
– $\overline{PL}$: entrée de chargement asynchrone
– CP_U : entrée du signal d'horloge de comptage
– CP_D : entrée du signal d'horloge de décomptage
– E_C : valide le comptage

– E_D : valide le décomptage

► **Remarque 10.2**

Toutes les commandes agissant sur le comptage sont regroupées sur la figure ci-dessous :

10.2.3 Applications

10.2.3.a Compteur de fréquence

Circuit qui mesure et affiche la fréquence d'un signal impulsionnel (mesure de fréquence inconnue).

Principe d'un compteur de fréquence

: durée pendant laquelle les impulsions sont comptées

: RAZ met le compteur à zéro

: $f = \frac{\text{contenu}}{t_2 - t_1}$

► **Remarque 10.3**

Le compteur est un montage en cascade de compteurs DCB, chacun ayant une unité décodeur/afficheur (affichage décimal).

La précision de cette méthode est fonction de l'intervalle d'échantillonnage.

10.2.3.b Horloge numérique

10.3 Résumé sur les méthodes de conception des compteurs

On a vu que la méthode à employer pour concevoir un compteur dépend de la catégorie du compteur.

On distingue déjà les compteurs synchrones des compteurs asynchrones. Si à première vue la réalisation d'un compteur asynchrone semble plus simple, il s'avère que la méthode de conception des compteurs synchrones permet une plus grande souplesse dans la réalisation de cycles évolués : ceci est dû au fait que les entrées des bascules constituant un compteur synchrone sont configurables, ce qui n'est pas le cas avec un compteur asynchrone.

Au sein d'une catégorie de compteurs, on trouve également des différences dans la démarche de conception selon la nature du cycle de comptage (modulo n ou 2^n, cycle régulier ou non, etc.).

10.3.1 Compteur asynchrone

Un compteur asynchrone est constitué par des diviseurs de fréquence par deux, connectés en cascade : la sortie d'un diviseur de fréquence est utilisé comme signal d'horloge pour le diviseur suivant.

Les diviseurs de fréquence doivent être sensibles au front descendant des signaux d'horloge ; dans le cas contraire, le système constitue un décompteur.

1. **compteur asynchrone à cycle régulier complet modulo 2^n ;**
   Ce type de compteur est le plus simple à réaliser : comme le modulo de ce compteur est une puissance de 2, le compteur reprend le cycle à 0 après avoir parcouru les 2^n états (de 0 à $2^n - 1$).
   Il n'y a donc aucun contrôle à effectuer.
2. **compteur asynchrone à cycle régulier complet modulo n ;**
   Ce type de compteur se conçoit comme un compteur asynchrone modulo 2^n auquel on ajoute de la logique permettant de détecter le premier état indésiré $n+1$. Dès que cet état est détecté, toutes les bascules sont alors forcées à l'état bas à l'aide de l'entrée de mise à zéro (clear, reset, etc.)
   Une variante consiste à détecter le dernier état valide n puis à forcer immédiatement les sorties de toutes les bascules à un à l'aide des entrées de mise à un (set, preset, etc.). L'inconvénient de cette méthode est qu'elle conduit à un cycle irrégulier, et que l'état (111…11) doit être transcodé pour donner un état valide.
3. **compteur asynchrone à cycle régulier incomplet ;**
   Ce type de compteur est conçu comme un compteur modulo n à ceci près que la logique de détection des états invalides doit prendre en compte non seulement le dernier état, mais également tous les états intermédiaires qui n'appartiennent pas aux états valides du compteur. Ceci implique que l'état du compteur peut être forcé non seulement à zéro, mais également à un état $k < n$: les entrées de mise à un seront donc utilisées en plus des entrées de mise à zéro.

▷ **Exemple 10.8**

Un compteur parcourant le cycle 0–1–4–6 sera conçu comme un modulo 7, mais les états 2 et 5 seront détectés en plus de l'état 7. Pour l'état 2, le compteur est forcé à l'état 4, et pour l'état 5, il est forcé à l'état 6. Il sera forcé à l'état 0 pour l'état 7, comme pour un compteur modulo 7 classique.

4. **compteur asynchrone à cycle irrégulier ;**

Ce type de compteur doit obligatoirement être conçu à l'aide d'un compteur à cycle régulier dont les états seront ensuite transcodés pour fournir les états désirés.

▷ **Exemple 10.9**

Un compteur, un compteur qui parcoure le cycle 0–3–1–7–5 sera conçu comme un compteur modulo 5, mais les états 0, 1, 2, 3, 4 seront convertis respectivement en 0, 3, 1, 7, 5.

Cette technique fonctionne également si le compteur traverse plusieurs fois le même état

▷ **Exemple 10.10**

0–3–1–7–3 sera conçu de la même manière que précédemment, mais les états 1 et 4 seront tous les deux transcodés en 3.

Notez que, du fait de la structure d'un compteur asynchrone, cette technique est la seule méthode permettant de faire passer le compteur plusieurs fois par le même cycle : il est en effet impossible de définir le signal d'entrée des bascules, car celles-ci sont configurées en diviseurs de fréquence par deux. La seule autre solution (dans le cadre du comptage asynchrone) serait d'ajouter des bascules permettant de sauvegarder l'état précédent du compteur).

► **Remarque 10.4**

Utilisation des entrées asynchrones prioritaires $\overline{\texttt{Clear}}$ ***et*** $\overline{\texttt{Preset}}$ ***:*** *Le propre d'un compteur est de compter ! L'utilisation des entrées asynchrones prioritaires a donc ceci de dangereux que ces entrées sont justement prioritaires est ignorent tout autre signal ! En conséquence, lorsque l'on définit les entrées* $\overline{\texttt{Clear}}$ *et* $\overline{\texttt{Preset}}$*, il faut veiller à ce que celles-ci soient activées (mises à zéro)* uniquement *pour les états indésirables !*

Ainsi, on ne peut pas définir les entrées prioritaires pour les états que l'on souhaite conserver, puisque ces entrées invalideraient instantanément l'état courant du compteur. La règle est donc de forcer l'entrée $\overline{\texttt{Clear}}$ *ou* $\overline{\texttt{Preset}}$ *dès que l'état indésirable est atteint, mais de la laisser à un le reste du temps.*

10.3.2 Compteur synchrone

1. **compteur synchrone modulo** n **;**
2. **compteur synchrone à cycle irrégulier.**

Chapitre 11

Méthodes d'étude des circuits séquentiels

Charles Lutwidge Dodgson
ou Lewis Carroll
⋆ 1832 – † 1898

« Can you do addition? the White Queen asked. What's one and one and one and one and one and one and one and one and one and one?
– I don't know, said Alice I lost count. » (Lewis Carroll, Through the Looking Glass)

De nombreux outils permettent d'analyser le fonctionnement et/ou de prévoir l'évolution d'un système séquentiel :

1°) Méthodes descriptives :

a) les tables d'état : elles donnent l'état futur des sorties pour les éléments de mémoire inclus dans les systèmes et l'état des sorties :

A	B	S	S^+

;

b) les diagrammes des temps (chronogrammes) : ils décrivent la succession des signaux d'entrée, des états des éléments de mémoire. Ils représentent la succession des états logiques en fonction du temps.

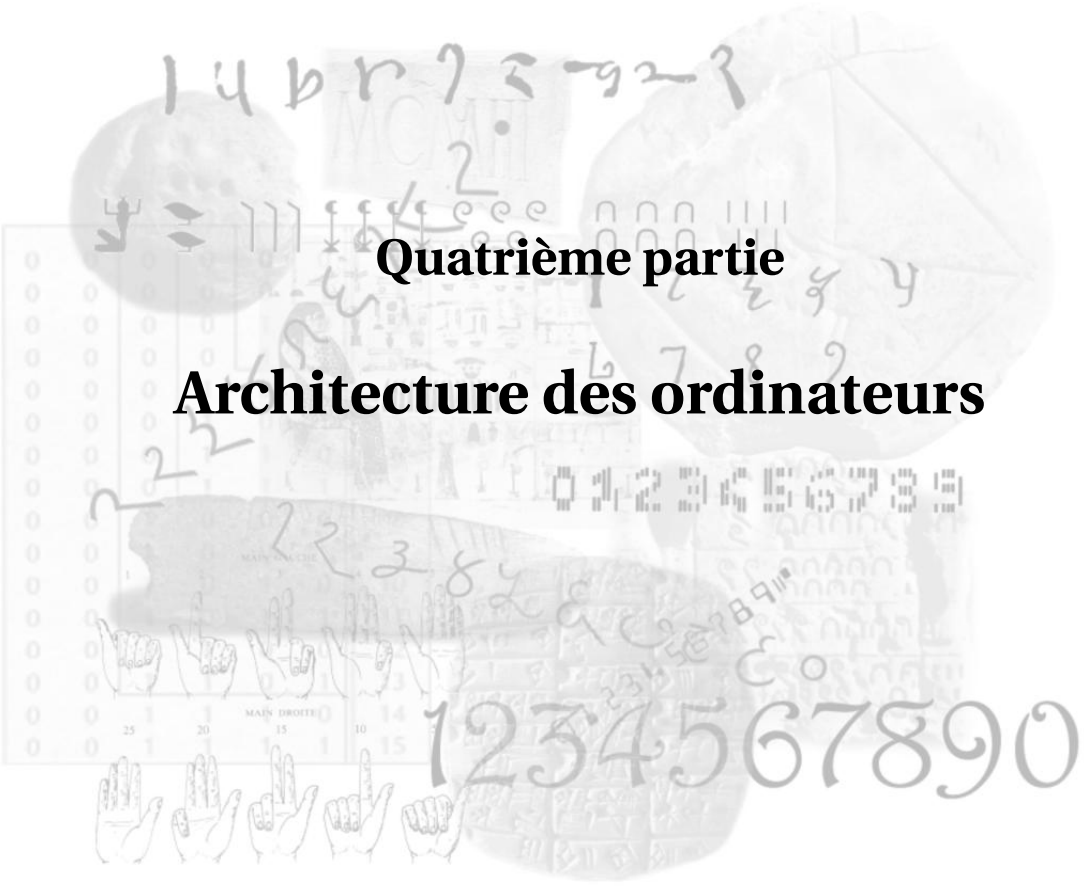
2°) Les diagrammes d'états ou graphes : ce sont des représentations formelles avec nœuds et flèches pour représenter les états stables et les transitions. Le graphe donne une image géométrique d'une table de vérité.

3°) Le grafcet : automatismes industriels : étape → transition → étape.

4°) Les théories formelles : équations qui représentent l'action à effectuer et l'état futur d'un élément de mémoire en fonction des entrées et de l'état présent des mémoires.

Quatrième partie

Architecture des ordinateurs



Chapitre 12

Concepts de base des processeurs

John von Neumann
⋆ 28 déc. 1903, Budapest, Hongrie
† 8 fév. 1957, Washington D.C., E.-U.

« Si quelqu'un croit que les mathématiques sont difficiles, c'est simplement qu'il ne réalise pas comme la vie est complexe ! » (John von Neumann)

Cinquième partie

Technologie des portes logiques

Chapitre 13

Famille des circuits logiques

William Bradford Shockley, ⋆ 13 fév. 1910, London, R.-U. ; † 12 août 1989, London, R.-U.
John Bardeen, ⋆ 23 mai 1908, Madison, Wisconsin, E.-U. ; † 30 jan. 1991
Walter Houser Brattain, ⋆ 10 fév. 1902, Amoy, Chine ; † 13 oct. 1987

Lauréats du Prix Nobel de Physique de 1956 pour l'invention du transistor. John Bardeen obtiendra un second Prix Nobel de Physique en 1972 pour ses travaux sur la supraconductivité. La qualité des travaux de Shockley sur le transistor ne doit cependant pas crédibiliser ses théories eugéniques d'un autre âge ...

Les technologies de portes logiques ont évolué à partir de la technologie *Diode-Logic* (DL). Celle-ci a évolué en *Resistor-Transistor-Logic* (RTL) puis *Diode-Transistor-Logic* (DTL), avant d'aboutir à la famille la plus populaire en son temps, la famille *Transistor-Transistor Logic* (TTL, CI 54xx ou 74xx).

D'autres technologies existent qui présentent chacunes des avantages et des inconvénients propres :

- ECL (*Emitter-Coupled Logic*) → pour des circuits rapides (10xxx) ;
- MOS (*Metal Oxyd Semiconductor*) → haute intégration ;
- CMOS (*Complementary Metal Oxyd Semiconductor*) → faible consom. (40xx) ;
- I^2L (*Integrated Injection Logic*) → haute intégration.

Certaines fonctionnent en logique positive, d'autres en logique négative.

13.1 Caractéristiques d'une famille de circuits numériques

Une famille de circuits logiques est définie par les caractéristiques dont les définitions suivent. Ces définitions sont nécessaires pour bien comprendre les différentes notions dont il est question dans ce chapitre.

Définition 13.1

Amplification : *L'amplification représente la capacité d'une porte logique d'amplifier la tension ou le courant présent à son entrée de manière à ce que le signal ne soit pas dégradé après avoir traversé plusieurs portes.*

Définition 13.2

Gain en courant : *Le gain en courant d'une porte représente le rapport du courant à la sortie de cette porte sur celui à son entrée pour le même niveau de tension. Il y a donc deux valeurs de ce gain une pour chaque niveau logique.*

Définition 13.3

Sortance : *La sortance ou* fan-out *est le nombre d'unités de « charge logique » disponibles à la sortie d'une porte ; cette unité correspond à la valeur du courant nécessaire pour commander une entrée de circuit logique*

Définition 13.4

Entrance : *L'entrance ou* fan-in *est le nombre d'unités de « charge logique » nécessaire en entrée pour faire fonctionner la porte.*

Définition 13.5

Temps de traversée : *Le temps de traversée est l'intervalle de temps qui sépare le signal d'entrée du signal de sortie qui en est la conséquence.*

Définition 13.6

Temps de montée, temps de descente : *Les temps de montée (respectivement de descente) d'un signal et l'intervalle de temps nécessaire au signal pour passer de 10% à 90% (respectivement de 90% à 10%) de sa valeur nominale.*

D'autres paramètres caractéristiques des circuits logiques peuvent avoir un impact lors du choix technologique, mais n'entrent pas dans le cadre de ce chapitre : dissipation d'énergie, tolérance, fiabilité, immunité aux parasites, coût, packaging, etc.

13.2 Évolution des différentes familles logiques

Pourquoi cette évolution ?

Le principal moteur de l'évolution des technologies de circuits intégrés est, comme bien souvent, le prix.

En 1960, une diode coûtait autant que 10 résistances, et un transistor autant que 50 résistances (ou 5 diodes). En 1966, une diode valait autant que 2 résistances, et un transistor autant que 10 résistance (toujours 5 diodes).

Cet état de fait a influé sur les premières technologies de circuits à composants discrets utilisant peu de transistors est beaucoup de résistances et de diodes : DL, DTL, DCTL, RTL, etc.

Aujourd'hui un transistor ne coûte pas plus cher qu'une diode, mais une résistance vaut autant que de nombreuses diodes pour plusieurs raisons : une raison technologique d'abord qui tient compte de la nature du matériaux utilisé pour réaliser la résistance, mais aussi et surtout par la place importante qu'occupe une résistance sur le silicium. De plus, une résistance, par définition, dissipe beaucoup d'énergie et entraîne une élévation de la température du circuit.

Toutes ces raisons font que les technologies actuelles utilisent principalement des transistors et peu de résistances (TTL, ECL) voire pas du tout de résistances (I^2L, CMOS).

État de l'art actuel

Les technologies à transistors bipolaires :

DTL : diode transistor logic (abandonné)

DCTL : direct coupled transistor logique

RTL : resistor transistor logic

RCTL : resistor capacitor transistor logic (abandonné)

ECL : emitter coupled logic
CML : current mode logic
TTL : transistor transistor logic
CTL : complementary transistor logic (abandonné)
I^2L : integrated injection logic

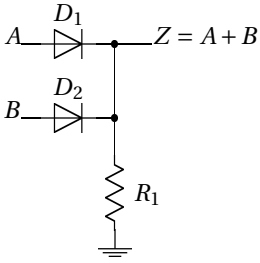
Les technologies à transistors MOS :

MOSP : metal oxyd semiconductor à canal P
MOSN : metal oxyd semiconductor à canal N
CMOS : complementary metal oxyd semiconductor
SOS : silicon on sapphir or spinel (MOS déposé sur saphir ou spinelle)
SOI : silicon on insulator (MOS déposé sur isolant SiO_2)

13.3 Présentation des différentes familles logiques

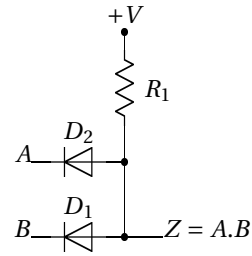
13.3.1 Diode Logic (DL)

La logique DL tire parti du fait que la diode est un composant électronique qui ne conduit le courant électrique que dans une seule direction. Ainsi, la diode agit comme un interrupteur électronique.



Le schéma ci-contre illustre une porte `OU` conçue en technologie DL. On considère que le 1 logique est représenté par +5 V, et que le 0 logique est représenté par la masse, ou 0 V. Si les deux entrées sont laissées non connectées, ou sont toutes les deux à l'état 0, la sortie Z sera également forcée à la masse par la résistance, et donc forcée à l'état 0. Si l'une ou l'autre des entrées est forcée à +5 V, la diode correspondante devient alors passante, ce qui force la sortie à l'état logic 1. Si les deux entrées sont à 1, la sortie sera toujours à 1 également.

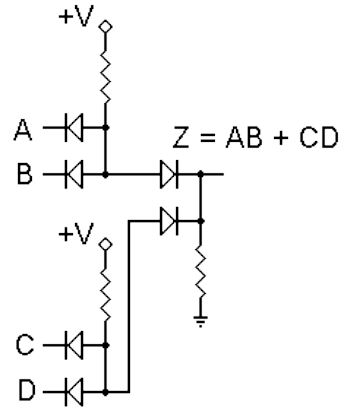
Le schéma ci-après illustre une porte `ET` en DL. Les mêmes niveaux logiques sont utilisés, mais les diodes sont inversées et la résistance est configurée pour rappeler la tension de sortie à l'état logique 1.



Si les deux entrées sont non connectées ou si elles sont toutes les deux à l'état logique 1, la sortie Z sera également à l'état logique 1. Si l'une ou l'autre des entrées est connectée à la masse (état logique 0), la diode conduit et ramène la masse vers la sortie, qui est donc forcée à l'état logique 0 également.

Dans les exemples précédents, nous avons considéré que les diodes n'introduisaient aucune erreur ni aucune perte dans le circuit, ce qui n'est pas vraiment le cas : une diode va entraîner une perte de tension d'environ 0,65–0,7 volts lorsqu'elle conduit. Nous pouvons cependant nous affranchir de ce problème en définissant un état logique 1 comme une tension *supérieure* à 3,5 V, et un état logique 0 comme une tension *inférieure* à 1,5 V. Tout niveau compris entre 1,5 et 3,5 V sera considéré comme illégal : c'est la région pour laquelle le niveau logique est *non défini*.

L'utilisation de portes individuelles reposant sur cette technologie ne pose pas de problème, du moment que l'on respecte les contraintes sur les niveaux de tension.



Cependant, si nous cascadons plusieurs portes DL, des problèmes peuvent apparaître. Dans l'exemple ci-contre, nous avons deux portes `ET` dont les sorties sont connectées aux entrées d'une porte `OU`. Ce schéma est très simple et ne semble pas poser de problème. En pratique, il en va différemment !

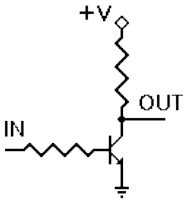
Si nous forçons les entrées à l'état bas 0, la sortie sera également forcée à 0 ; aucun problème donc.

Cependant, si les deux entrées de l'une ou l'autre des portes `ET` sont à +5 V, les diodes de la porte `OU` seront alors passantes (niveau haut ramené sur les anodes), et le courant circulera alors à travers la résistance de la porte `ET`, à travers la diode, et à travers la résistance de la porte `OU`.

Si on considère que les résistances sont d'égale valeur (ce qui est typiquement le cas), elles vont se comporter comme un diviseur de tension et ainsi partager le +5 V en deux parties égales ; la diode de la porte OU va également introduire une légère perte de tension, et la tension de sortie du système sera alors d'environ 2,1 ou 2,2 V. Si les deux portes ET voient leurs deux entrées au niveau logique 1, la tension de sortie peut monter à 2,8 ou 2,9 V. En tout état de cause, la tension de sortie de la porte OU sera dans la « zone interdite », région de la tension pour laquelle le niveau logique n'est pas défini.

En poursuivant plus avant, si on connecte les sorties de deux ou plus de ces structures à une autre porte OU, nous perdons tout contrôle sur la tension de sortie : il va se trouver quelquepart une diode polarisée en inverse qui va bloquer le signal´ d'entrée, empêchant le circuit de fonctionner correctement. C'est pourquoi la logique DL ne peut être utilisée que pour des portes uniques, et dans des circonstance spécifiques.

13.3.2 Resistor Transistor Logic (RTL)



Considérons le circuit à transistor le plus simple qui soit, comme celui-ci ci-contre à gauche. Nous appliquerons uniquement l'une des deux tensions suivantes à l'entrée I : 0 V (0 logique) ou +V volts (1 logique). La valeur exacte de la tension +V dépend des paramètres du circuit ; dans les circuits intégrés RTL, la tension habituellement utilisée est +3,6 V. Considérons que le transistor utilisé ici est un transistor NPN avec un gain en courant raisonnable, une tension émetteur-base de 0,65 V, et une tension de saturation collecteur-émetteur inférieure à 0,3 V. Dans les circuits intégrés RTL standards, la résistance de base est de 470 Ω, et la résistance de collecteur est de 640 Ω.

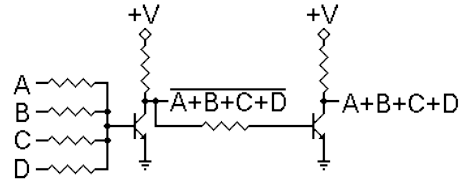
Lorsque la tension d'entrée est zéro volt (en pratique, n'importe quelle tension inférieure à 0,5 V), il n'y a pas de courant émetteur-base et le transistor est bloqué. Ainsi, aucun courant ne circule à travers la résistance de collecteur, et la tension de sortie est de +V volts. En d'autres termes, un 0 logique en entrée résulte en un 1 logique en sortie.

Lorsque la tension d'entrée est de +V volts, la jonction émetteur-base est polarisée est le transistor passant. La tension de sortie sera donc de $3,6-0,65 = 2,95$ volts appliqué au travers d'une combinaison de résistances en série de 640 Ω

pour la résistance de sortie et de 470 Ω pour la résistance d'entrée. Ceci nous donne un courant de base de $2,95/1110 = 0,0026576577 = 2,66$ mA.

La logique RTL est une technologie relativement ancienne, et les transistors utilisés dans les circuits intégrés RTL ont un gain d'environ 60 à 100. Si on considère un gain de 60, un courant de base de 2,66 mA supporte un courant de collecteur maximal de 159,6 mA. Si la chute de tension aux bornes de la résistance de collecteur de 640 Ω est de 3,3 V $(3,6 - 0,3)$, le courant sera alors de 5,1 mA. Ainsi, le transistor sera complètement saturé.

Avec un 1 logique en entrée, ce circuit produit un 0 logique en sortie, et nous vu qu'un 0 logique en entrée, il produit un 1 logique en circuit : ce circuit est un inverseur.



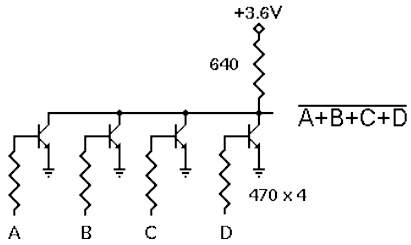
Comme nous pouvons le constater à l'issue des calculs précédents, la quantité de courant fournie à la base du transistor est beaucoup plus importante que ce qui est nécessaire pour faire commuter le transistor vers la saturation. Ainsi, il est possible d'utiliser une seule sortie pour commander plusieurs entrées d'autres portes, ainsi que d'avoir des portes comportant plusieurs résistances d'entrée. Un tel circuit est représenté ci-dessus.

Dans ce circuit, nous avons quatre résistances d'entrée. Porter l'une des entrées à 3,6 V est suffisant pour saturer le transistor, et appliquer d'autre 1 logique additionnels en entrée n'aura pas réellement d'effet sur la tension de sortie. Rappelons que la tension de polarisation sur la base du transistor n'excédera pas 0,65 V, ainsi le courant à travers une résistance d'entrée reliée à la masse ne dépassera pas $0,65/470 = 1,383$ mA. Ceci nous donne une limite pratique pour le nombre de résistances d'entrées pour un seul transistor, mais ne génère aucun problème sérieux à l'intérieur de cette limite.

La porte RTL décrite précédemment fonctionne, mais elle pose problème à cause d'une possible interaction des signaux d'entrée à travers les multiples résistances d'entrée. Une meilleure façon d'implanter une fonction `NON-OU` est montrée sur le schéma suivant.

Ici, chaque transistor a seulement une résistance d'entrée, de manière à ce qu'il n'y ait aucune interaction entre les entrées. La fonction `NON-OU` est réalisée à

la connexion du collecteur commun de tous les transistors qui partagent une seule résistance de charge du collecteur.



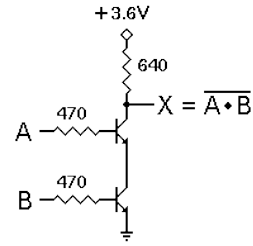
Ceci est en fait la structure utilisée pour tous les circuits intégrés RTL. Le circuit μL914, très répandu, est un double porte `NON-OU` à deux entrées, où chaque porte est une version à deux transistors du circuit ci-dessus à droite. Il consomme 12 mA lorsque toutes les sorties sont au niveau logique 0. Ceci correspond parfaitement aux calculs que nous avons effectués précédemment.

Le *fan-out* standard pour les portes RTL est de 16. Cependant, le *fan-in* pour une porte RTL standard est de 3. Ainsi, une porte produit 16 unités de courant en sortie, mais nécessite 3 unités pour commander une entrée. Il existe des version basse consommation (*low-power*) de ces portes qui augmentent les valeurs des résistances de base et de collecteur à 1,5 kΩ et 3,6 kΩ respectivement. De telles portes demandent moins de courant, et ont typiquement un *fan-in* de 1 et un *fan-out* de 2 ou 3. Elles ont également une réponse en fréquence réduite, de sorte qu'elles ne peuvent fonctionner aussi rapidement que les portes standards. Pour obtenir une plus grande capacité de commande en sortie (*fan-out* plus élevé), on utilise des *buffers* : ce sont des inverseurs conçu de manière à avoir un *fan-out* de 80. Ils ont également un *fan-in* de 6, puisqu'ils utilisent des paires de transistors pour obtenir cette capacité de fournir plus de courant.

On peut obtenir une fonction `NON-ET` de deux façons : on peut d'inverser les entrées d'une porte `NON-OU/OU`, la transformant ainsi en porte `ET/NON-ET`, ou on peut utiliser le circuit présenté ci-contre.

Dans ce circuit, chaque transistor possède sa propre résistance d'entrée, ainsi chacun est contrôlé par un différent signal d'entrée. Cependant, la seule façon dont la sortie peut être ramenée au niveau logique 0 est que les deux transistors soient activés par des entrées au niveau logique 1. Si l'une ou l'autre des entrées est au niveau logique 0 le transistor correspondant ne peut conduire, ainsi aucun courant ne circule dans aucun des transistors. La sortie est donc au niveau logique 1. C'est le comportement d'une porte `NON-ET`. Il est possible d'inclure un inverseur pour réaliser une sortie `ET` par la même occasion.

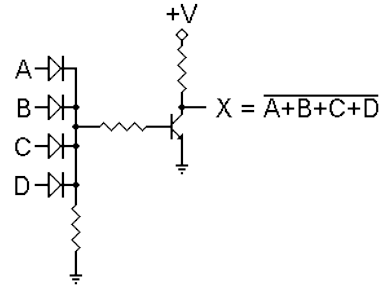
Le problème avec ce circuit NON-ET tient au fait que les transistors ne sont pas parfaits. La tension de collecteur de 0,3 V lorsque le transistor est saturé devrait idéalement être de 0 V. Comme elle ne l'est pas, il faut examiner ce qu'il se passe lorsque l'on « empile » les transistors de cette façon. Avec deux transistors, la tension de collecteur en saturation cumulée est de 0,6 V, c'est-à-dire seulement très peu en deçà de la tension de base de 0,65 V qui sature un transistor.



Si l'on empile trois transistors pour réaliser une porte NON-ET à trois entrées, la tension de collecteur en saturation cumulée est de 0,9 V : ceci est trop élevé, et provoquera la conduction dans le transistor suivant quelquesoit le niveau logique appliqué en entrée. De plus, la charge que constitue le transistor le plus haut à la porte qui le pilote sera différente de la charge que constitue le transistor le plus bas. Ce genre d'irrégularité peut causer l'apparition de problèmes, plus particulièrement lorsque la fréquence des opérations augmente. À cause de ces problèmes, cette approche n'est pas utilisée avec les circuits intégrés RTL standards.

13.3.3 Diode Transistor Logic (DTL)

Nous l'avons vu §13.3.1, le principal problème avec les portes DL est qu'elles détériorent rapidement le signal logique. Cependant, elles fonctionnent pour un étage à la fois si le signal est ré-amplifié entre deux portes : c'est le but de la technologie *Diode Transistor Logic*.

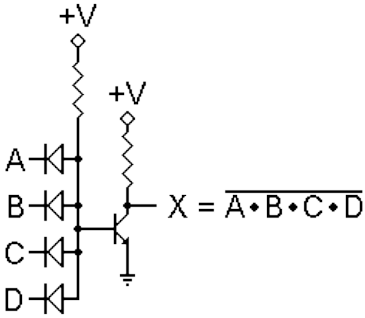


La porte à droite est une porte OU DL suivie par un inverseur tel que celui présenté §13.3.2.

La fonction OU est toujours réalisée par les diodes. Cependant, quelquesoit le nombre d'entrées au niveau logique 1, il est certain qu'il y aura une tension d'entrée suffisante pour faire passer le

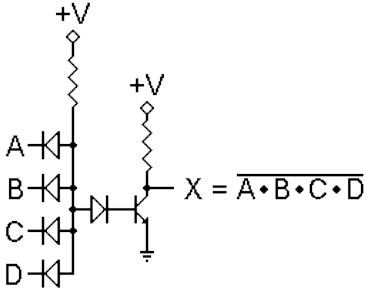
transistor en saturation. Le transistor restera bloqué uniquement si toutes les entrées sont au niveau logique 0. Ainsi le circuit réalise la fonction NON-OU.

L'avantage de ce circuit sur son équivalent RTL est que le OU logique est réalisé par les diodes, et non par les résistances. Ainsi, il n'y a aucune interaction entre les différentes entrées, et un nombre quelconque de diodes peut être utilisé (donc un nombre quelconque d'entrées). Un inconvénient de ce circuit est la résistance d'entrée du transistor. Sa présence a tendance à ralentir le circuit, et limite ainsi la vitesse à laquelle le transistor est capable de changer d'état.



En première lecture, la version NON-ET ci-contre devrait éliminer ce problème. Un niveau logique 0 devrait ramener immédiatement la masse à la base du transistor et ainsi bloquer ce dernier...

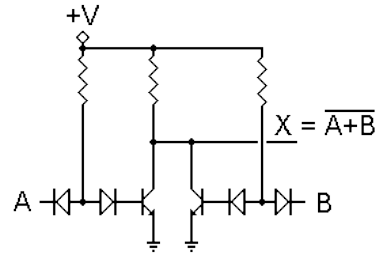
En fait, ça ne se passe pas réellement ainsi. Rappelons que la tension de seuil de la diode lorsqu'elle conduit est très proche de la tension présente à la base du transistor (0,65 V). Ainsi, même lorsque toutes les entrées sont reliées à la masse, la base du transistor sera à un potentiel d'environ 0,65 V, et le transistor peut conduire ...



Pour résoudre ce problème, il est possible d'ajouter une diode en série avec le transistor comme montré sur le schéma ci-contre. Maintenant, la tension nécessaire pour faire commuter le transistor est de 1,3 V. Pour plus de sécurité, on pourrait ajouter une seconde diode en série, ce qui nécessiterait 1,95 V pour saturer le transistor. De plus, on peut ainsi être sûr que des changements de température n'affecteront pas de manière significative le fonctionnement du circuit.

En tout état de cause, ce circuit fonctionne comme une porte NON-ET. De plus, comme pour la porte NON-OU, on peut utiliser autant de diodes d'entrées que l'on veut sans augmenter la tension de seuil. De plus, en l'absence de résistance en série dans le circuit d'entrée, il y a moins d'effet de ralentissement, et le transistor peut commuter plus rapidement et donc gérer des fréquences plus élevées.

Ceci étant, est-il possible d'appliquer la même raisonnement à la porte NON-OU et éliminer la résistance pour permettre une commutation plus rapide ?



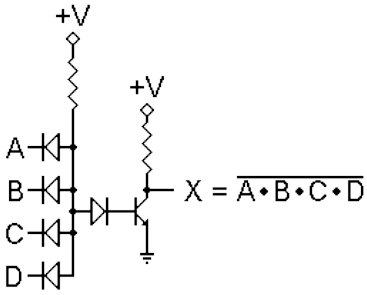
La réponse est oui. Considérons le circuit ci-contre. On utilise ici des transistors séparés connectés ensembles. Chacun a une entrée unique, et fonctionne donc comme un inverseur. Cependant, si les collecteurs sont connectés ensembles, un 1 logique appliqué à l'une des entrées forcera la sortie au niveau logique 0. C'est une porte NON-OU.

La même approche peut être utilisée pour les portes NON-OU/OU RTL, de manière à ce que l'opération NON-OU soit réalisée au niveau des collecteurs plutôt qu'en utilisant des résistances. Cette approche élimine également la limite sur le nombre d'entrées pouvant être utilisées, puisqu'il n'y a aucune interaction entre les entrées.

13.3.4 Transistor Transistor Logic (TTL)

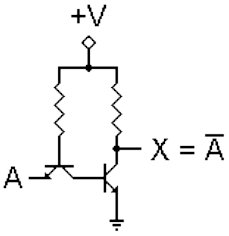
Avec le développement rapide des circuits intégrés, des nouveaux problèmes sont apparus, et des nouvelles solutions furent développées pour y remédier. L'un des problèmes avec les circuits DTL était qu'il fallait autant de place sur le circuit pour réaliser une diode que pour un transistor. Il était donc souhaitable de ne pas avoir à nécessiter autant de diodes. La question est donc de savoir par quoi remplacer ces diodes ...

En étudiant la porte DTL ci-contre, on peut constater que les diodes montées en opposition ressemblent énormément aux deux jonctions d'un transistor. En fait, si nous avions un inverseur, il n'aurait qu'une seule diode d'entrée, et il aurait été possible de remplacer ces deux diodes opposées par un transistor NPN qui jouerait le même rôle.



En pratique, ceci fonctionne parfaitement. La figure suivante illustre l'inverseur résultant de cette transformation.

De plus, il est possible d'ajouter plusieurs émetteurs au transistor d'entrée sans accroître énormément l'espace nécessaire sur le circuit. Ceci nous permet de réaliser une porte à plusieurs entrées dans pratiquement le même espace qu'un inverseur.



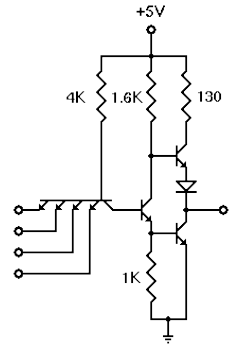
Les économies d'espaces réalisées se traduisent en une économie significative sur les coûts de fabrication, ce qui réduit les coûts au niveau de l'utilisateur final.

Un problème partagé par toutes les portes logiques avec un seul transistor de sortie et une résistance de rappelle à +V (*pull-up*) sur le collecteur est la vitesse de commutation. Le transistor tire la sortie vers le niveau logique 0 de manière active, mais la résistance n'est pas active lorsqu'elle tire la sortie vers le niveau logique 1. À cause de facteurs inévitables comme les capacitances du circuit ainsi qu'à une caractéristique des transistors bipolaire appelée « stockage de charge », cela prendrait un certain temps aux transistor pour se bloquer complètement et à la sortie pour atteindre le niveau logique 1. Ceci limite la fréquence à laquelle la porte peut fonctionner.

Les concepteurs de circuits TTL commerciaux réduisent ce problème en modifiant le circuit de sortie. Le résultat est le circuit de sortie « totem pole » utilisé dans la plupart des circuits intégrés TTL des séries 7400/5400. Le circuit final utilisé dans la plupart des circuits intégrés commerciaux standards est présenté sur la figure de gauche. Le nombre d'entrées peut varier – un CI commercial peut avoir 6 inverseurs, quatre portes à deux entrées, trois portes à trois entrées, ou deux portes à quatre entrées. Une porte à 8 entrée dans un seul boitier est également disponible. Dans tous les cas la structure du circuit reste la même.

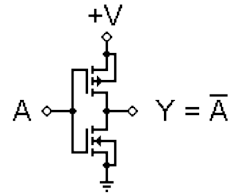
13.3.5 Complementary Metal Oxyd Semiconductor Logic (CMOS)

La technologie CMOS est une technologie plus récente basée sur l'utilisation de transistors MOS complémentaires pour réaliser les fonctions logiques et qui nécessite un courant pratiquement nul pour fonctionner. Ceci rend cette technologie particulièrement intéressante dans les applications alimentées par batteries. De plus, elles peuvent fonctionner avec des tensions variant de 3 V (voire moins pour les dernières technologies : 1,3 V) à 15 V.



Les portes CMOS sont toutes basées sur l'inverseur présenté sur la figure de droite. Les deux transistors sont des MOSFETs en mode étendu ; un canal N avec sa source reliée à la masse, et un canal P avec sa source connectée à +V. Leurs grilles sont reliées pour former l'entrée, et leurs drains sont reliés pour former la sortie.

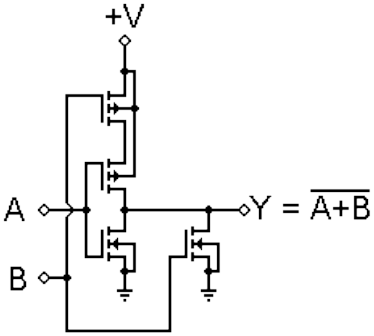
Les deux MOSFETs sont conçus pour avoir des caractéristiques crorrespondantes ; ainsi, ils sont complémentaires l'un de l'autre. Lorsqu'ils sont bloqués, leur résistance est infinie, et lorsqu'il sont passants, la résistance de leur canal est d'environ 200 Ω. Puisque la porte est essentiellement un circuit ouvert, elle ne consomme aucun courant, et la tension de sortie sera égale soit à la masse, soit à la tension d'alimentation, selon le transistor en train de conduire.



Lorsque l'entrée A est mise à la masse (0 logique), le MOSFET à canal N n'est pas polarisé, il est bloqué. C'est un circuit ouvert, et donc il laisse la ligne de sortie déconnectée de la masse. En même temps, le MOSFET à canal P est polarisé ; il devient passant et son canal a une résistance d'environ 200 Ω, connectant ainsi la ligne de sortie à l'alimentation. Ceci ramène donc la tension +V à la sortie (1 logique).

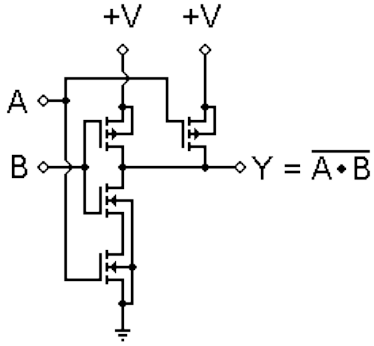
Lorsque l'entrée A est à +V (1 logique), le MOSFET à canal P est bloqué et le MOSFET à canal N est passant, ramenant la masse vers la sortie (0 logique).

Ainsi, le circuit réalise bien l'inversion logique en même temps qu'il génère des rappels actifs à +V (*pull-up*) ou à la masse (*pull-down*), selon l'état de la sortie.



Ce concept peut être étendu aux structures `NON-OU` et `NON-ET` en combinant des inverseurs dans une structure partiellement série, partiellement parallèle. Le circuit présenté ci-dessus est un exemple de porte `NON-OU` CMOS à deux entrées.

Dans ce circuit, si les deux entrées sont à l'état bas, les deux MOSFETs à canal P seront passant, induisant une connexion à +V. Les deux MOSFETs à canal N seront bloqués, il n'y aura donc pas de connexion à la masse. Cependant, si l'une des entrées passe à l'état haut, le MOSFET à canal P correspondant se bloquera et déconnectera par là même la sortie du +V, alors que le MOSFET à canal N correspondant deviendra passant, ramenant la sortie à la masse.

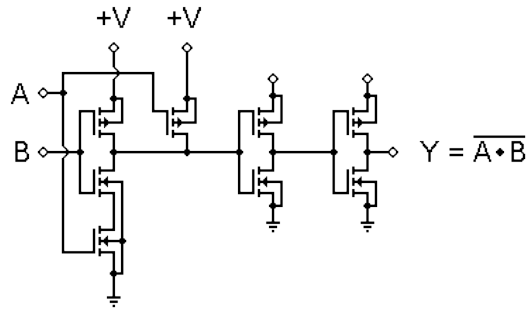


La structure peut être inversée, comme montré sur la figure précédente. Ici, nous avons une porte `NON-ET` à deux entrées, pour laquelle un 0 logique sur l'une ou l'autre des entrées forcera la sortie à l'état logique 1. Il faudra par contre que les deux entrées soient au niveau logique 1 pour autoriser la sortie à passer à l'état logique 0.

Cette structure est moins limitée que son équivalent bipolaire, mais il existe tout de même des limites pratiques. L'une de ces limites est la résistance cumulée des MOSFETs en série. Ainsi, les totem-pôles CMOS ne contiennent pas plus de quatre entrées. Les portes avec plus de quatre entrées sont construites en cascadant les structures plutôt que comme des structures uniques.

Même avec cette limite, la structure totem-pôle cause encore problème dans certaines applications. Les résistances de rappel à +V et à la masse présentes en sortie ne sont jamais les mêmes, et peuvent changer de manière significative lorsque les entrées changent d'état, même si la sortie ne change pas d'état logique.

Le résultat est des temps de montée et de descente irréguliers et imprédictibles pour le signal de sortie. Ce problème a été résolu à l'aide des versions bufferisées (série B) des portes CMOS.



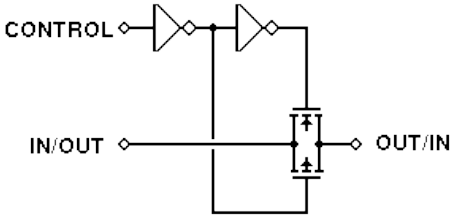
La technique utilisée ici est de faire suivre la porte `NON-ET` par une paire d'inverseurs. Ainsi, la sortie sera toujours pilotée par un seul transistor, soit à canal P, soir à canal N. Puisque les transistor sont choisis pour être aussi appairés que possible, la résistance de sortie de la porte sera toujours la même, le comportement du signal en est plus prédictible.

L'un des principaux problèmes avec les portes CMOS est leur vitesse. Elles ne peuvent pas fonctionner très rapidement, à cause de leur capacitance d'entrée inhérente. Les portes de la série B permettent de résoudre en partie ces limitations en fournissant un courant de sortie uniforme et commutant les états en sortie plus rapidement, même si le signal d'entrée change plus lentement.

Un type de porte, illustré ci-après, est unique à la technologie CMOS : il s'agit du « switch bilatéral », plus couramment appelé « porte de transmission ». Cette porte fait un usage approfondi du fait que les TEC individuels dans un circuit intégré CMOS sont construits de manière à être symétriques. Et en pratique le drain et la source de n'importe quel transistor peuvent être interchangés sans affecter les performances ni du transistor lui-même, ni du circuit dans son ensemble.

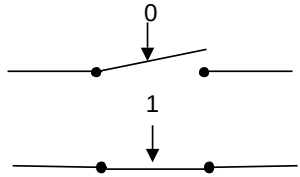
Lorsque les TEC de type N et P sont connectés comme montré dans ce schéma et que leurs grilles sont pilotées par des signaux de contrôle complémentaires, les deux transistors seront passants ou bloqués ensemble, au lieu de l'être alter-

nativement. S'ils sont tous les deux bloqués, le chemin parcouru par le signal est un circuit ouvert : il n'y a aucune connexion entre l'entrée et la sortie. S'ils sont tous les deux passants, il y a une connexion de très faible résistance entre l'entrée et la sortie, et un signal pourra circuler.

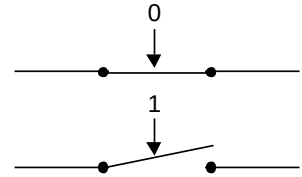


Ce qui est vraiment intéressant dans cette structure est que le signal contrôlé de cette manière n'a pas besoin d'être un signal numérique. Aussi longtemps que la tension du signal ne dépassera pas la tension d'alimentation, même un signal analogique peut être contrôlé par ce type de porte.

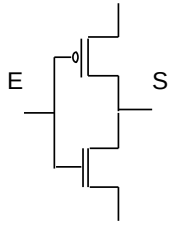
13.4 Implantation des opérateurs en technologie CMOS

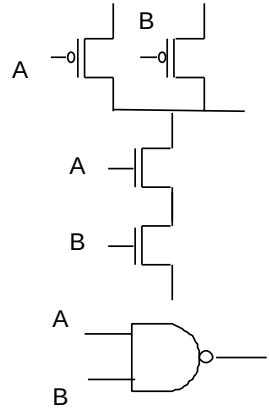


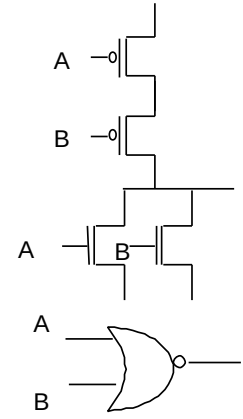
Transistor N :
ouvert si grille =0
fermé si grille =1

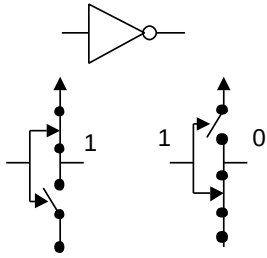


Transistor P :
ouvert si grille =1
fermé si grille =0







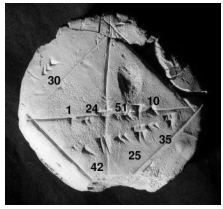


Sixième partie

Annexes

Annexe A

Examen sur les systèmes de numération sumérien et babylonien



Tablet YBC 7289 (c. 1800–1600 BCE) : Babylonian approximation to $\sqrt{2}$ in the context of Pythagoras' Theorem for an isosceles triangle. ***(Bill Casselman)***

La numération mésopotamienne utilise essentiellement deux systèmes de numération de position : l'une sexagésimal stricte avec les clous et chevrons, l'autre mélangeant système décimal et sexagésimal. Cette numération est partagée par les Babyloniens et les Akkadiens et provient de celle utilisée par les Sumériens

A.1 Numération

Le système numérique sumérien primitif est un système mixte de bases 60 (sexagésimale) et 10 (décimale). Un nombre dans ce système est composé des 6 symboles représentés dans le tableau ci-dessous avec leurs poids respectifs et la notation que nous utiliserons par convention.

Symbole						
Poids	$10^0.60^0$	$10^1.60^0$	$10^0.60^1$	$10^1.60^1$	$10^0.60^2$	$10^1.60^2$
Valeur	*1*	*10*	*60*	*600*	*3600*	*36000*
Notation	*T*	*V*	*W*	*X*	*Y*	*Z*
Modulo	*10*	*6*	*10*	*6*	*10*	—

On peut donc l'écrire :

$N = z \times$  $+ y \times$  $+ x \times$  $+ w \times$  $+ v \times$  $+ t \times$  *, soit*

$N = z.Z + y.Y + x.X + w.W + v.V + t.T$, *soit*

$N = z \times 10 \times 60^2 + y \times 60^2 + x \times 10 \times 60 + w \times 60 + v \times 10 + t$ *, soit*

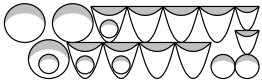
$N = (10.z + y) \times 60^2 + (10.x + w) \times 60 + (10.v + t)$

où l'on retrouve bien la base 60 qui multiplie des nombres exprimés en base 10. Les coefficients t, v, w, x, y, et z représentent le nombre d'occurences des symboles respectifs T,V,W,X,Y, et Z.

Exemple :

$$(45322)_{10} = 1\times 36000 + 2\times 3600 + 3\times 600 + 5\times 60 + 2\times 10 + 2\times 1$$

$1\times$  $+2\times$  $+3\times$  $+5\times$  $+2\times$  $+2\times$ 

Soit : 

Question 1

On désire coder en binaire le système de numération sumérien : on nommera ce code SCB (Sumérien Codé Binaire). Le nombre z.Z+y.Y+x.X+w.W+v.V+t.T sera alors noté $(\text{zyxwvt})_{\text{SCD}}$.

Exemple :

$$(45322)_{10} = 1.Z + 2.Y + 3.X + 5.W + 2.V + 2.U = (123522)_{\text{SCB}}$$

a) En fixant la taille des mots SCB à 24 bits, et en tenant compte des caractéristiques de chaque symbole (modulo), quel est le nombre minimal de bits à assigner à chaque symbole ? On considèrera Z non borné.

b) Quel est, en binaire et en décimal, le plus grand nombre entier positif qu'il est possible de représenter dans ce système (toujours sur 24 bits) ? À titre de comparaison, quel est, toujours en binaire et en décimal, le plus grand nombre qu'il est possible de représenter sur 24 bits en DCB ainsi qu'en binaire pur ? Tracez un tableau comparatif.

A.2 Arithmétique

Rappel de cours :
L'addition de deux nombres codés en DCB se passe de la manière suivante : on additionne deux à deux les chiffres de même rang (unités, dizaines, etc.). Si le résultat est inférieur à $(10)_{10}$, on conserve la valeur obtenue, sinon on ajoute 6 et on propage la retenue. Ceci est dû au fait que les nombres sont codés modulo 16 (4 bits) mais représentent une valeur modulo 10 : comme 16-10=6, on ajoute 6 pour « compléter » le modulo.

Exemple :

	7	*6*			*7*	*6*
+	*1*	*1*		+	*1*	*7*
=	*8*	*7*		=	*8*	*D*
				+	0_1	*6*
				=	*9*	*3*

Question 2

En considérant que tous les symboles du code SCB sont codés avec un nombre minimal de bits, expliquez comment appliquer cette méthode à l'addition de nombres SCB. Détaillez et faites un schéma.

A.3 Conversion

Le système numérique babylonien a hérité du système sumérien et conserve notamment la base sexagésimale. Cependant, la représentation des nombres repose maintenant sur 59 symboles réalisés à partir de deux graphes uniques :  *et dont les valeurs respectives sont 1 et 10.*

Les 59 symboles du système babylonien sont les suivants :

e) Convertissez  = $(1,25,47)_{60}$ vers la base 10.

f) Convertissez  = $(1,25,47)_{60}$ vers la base SCB. Que constatez-vous ?

A.4 Comptage

*On désire réaliser un compteur **synchrone** modulo 60 à fonctionnement sur front. En se basant sur l'écriture des nombres babyloniens, et bien que le zéro n'existe pas dans ce système on notera les nombres en base 10 : le chiffre des dizaines*  *varie de 0 à 5, et celui des unités UN de 0 à 9. On réalisera donc le compteur modulo 60 comme un compteur modulo 10 en cascade avec un compteur modulo 6.*

Question 4

a) Réalisez les deux compteurs en logique synchrone à l'aide :
– de bascules D pour le compteur modulo 10 ;
– de bascules JK pour le compteur modulo 6 ;
– de portes NON-ET **ou** NON-OU, en expliquant votre choix.

b) De manière à faciliter la mise en cascade de ces compteurs synchrones, chacun des deux compteurs devra comporter une entrée de validation E_n, ainsi qu'une sortie de dépassement C_{out} positionnée à 1 lorsque le compteur atteint son état final, et à 0 sinon. La table de vérité du compteur sera alors la suivante :

E_n	Mode
0	Mémorisation
1	Comptage

Vous devez impérativement donner les tables d'implication séquentielle, ainsi que les tableaux de Karnaugh si nécessaire, et commenter tous vos choix technologiques.

Sans refaire les calculs, et pour chacun des compteurs, donnez les nouvelles équations des entrées J,K ou D en fonction de E_n, ainsi que l'équation de C_{out}.

Question 5

Sans entrer dans les détails, réalisez un vrai compteur modulo 60, et comparez les coûts respectifs des deux méthodes en termes de bascules et de logique

combinatoire. Aidez-vous de la structure d'un compteur synchrone à cycle complet, sans refaire la table d'implication séquentielle.

Rappel de cours :
Pour un compteur modulo 8 (cycle complet sur 3 bits), on obtient les équations suivantes : $J_0=K_0=1$, $J_1=K_1=Q_0$, *et* $J_2=K_2=Q_0.Q_1$

Le système sexagésimal, inventé il y a plus de 4000 ans, reste très utilisé aujourd'hui, notamment dans la représentation des heures et des angles.

On désire réaliser la commande d'un moteur de télescope avec une précision d'une seconde d'arc. Le moteur à contrôler avance par pas de 1" à chaque impulsion de commande, et peut faire un tour complet. Le déplacement à effectuer pour atteindre la position souhaitée est stocké dans trois registres (deg, mn, sec).

Le principe consiste donc à appliquer au moteur un nombre d'impulsions fixe de manière à ce que le parcours effectué soit identique au parcours stocké sous forme d'angle dans les registres. Les comparateurs C_0, C_1 *et* C_2 *délivrent un niveau haut en sortie tant que les valeurs en entrées sont différentes l'une de l'autre. Le générateur d'impulsions fabrique ces impulsions tant qu'une au moins des sorties* r_0, r_1 *et* r_2 *des comparateurs est au niveau logique 1. À partir du moment où toutes les sorties des comparateurs sont à zéro, les sorties des compteurs et du générateur d'impulsions sont forcées à 0 de manière asynchrone, jusqu'à l'apparition d'un nouveau parcours se caractérisant par un basculement vers le niveau haut d'un ou plusieurs comparateurs.*

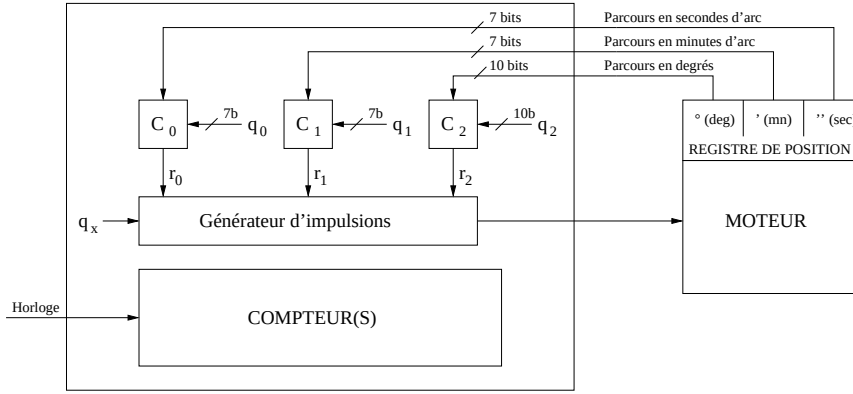
Question 6

Sachant que 1° = 60' et 1' = 60", proposez, à partir d'un ou plusieurs compteurs, un système permettant de positionner le téléscope ; pour cela :

a) Dessinez le schéma du compteur en y indiquant les sorties q_0, q_1, q_2 et q_x. Précisez notamment à quoi correspondent q_2 et q_x.

b) Dessinez le schéma logique du générateur d'impulsions à l'aide des portes logiques de votre choix.

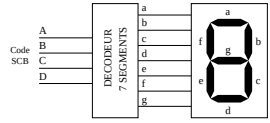
Note : *N'utilisez que des compteurs parmi ceux réalisés précédemment, sans refaire les calculs, et en vous aidant du schéma suivant et de la question 4.*



A.5 Codage

Question 7

On veut afficher la valeur du déplacement angulaire à l'aide de 7 afficheurs à 7 segments composés de 7 diodes électroluminescentes commandées par les bits a,b,c,d,e,f.



En vous aidant du schéma précédent, réalisez le décodeur SCB vers 7 segments :
– donnez les équations des variables, a,b,c,d,e et f ;
– réalisez le schéma logique de chacune de ces variables.

Annexe B

Correction des exercices

Index

Bibliographie

Livres

[LivWhi61] J.E. Whitesitt, “Boolean algebra and its applications”, 1961, Addison-Wesley.

[LivLaf] J.-C. Lafont & J.-P. Vabre, « Cours et Problèmes d’Électronique Numérique », Éditions Ellipses.

[LivToc] R. J. TOCCI, « Circuits Numériques, Théorie et Applications », Éditions Dunod.

Articles de revues

[ArtKar53] Maurice Karnaugh, “The Map Method for Synthesis of Combinational Logic Circuits”, *Trans. AIEE. pt I*, 72(9) :593-599, November 1953.

Sites Web

[WebMul] Daniel Muller, « Systèmes de numération », http://tic01.tic.ec-lyon.fr/~muller/trotek/cours/numeration/index.html.fr

[WebBig] Ken Bigelow, “Digital Logic” (Site Web sur la technologie des portes logiques), http://www.play-hookey.com/digital/

[Wiki01] Wikipédia, Article sur l’algorithme Quine–McCluskey : http://en.wikipedia.org/wiki/Quine%E2%80%93McCluskey_algorithm

[Wiki02] Wikipédia, Article sur la méthode de Petrick, http://en.wikipedia.org/wiki/Petrick%27s_method

[Wiki03] Wikipédia, Article sur Espresso, http://en.wikipedia.org/wiki/Espresso