Cours d'Architecture
Premi e Ann e

Circuits logiques et Algèbre de Boole

M. Billaud
D partement Informatique
IUT - Universit Bordeaux

1999
revu 24 juillet 2021

Table des matières

Table des figures

Chapitre 1

Codages de l'information

Les ordinateurs manipulent les informations sous forme *binaire* : toutes les données sont représentées par des suites d'éléments d'information (*bits*) qui ne peuvent prendre que deux valeurs, appelées arbitrairement 0 et 1 (ou *vrai* et *faux*).

On remarque qu'une séquence de 2 bits peut prendre 4 valeurs différentes $00, 01, 10, 11$. Une séquence de 3 bits peut prendre 8 valeurs : 000, 001,010,011,100,101,110,111. Plus généralement, il y a 2^n séquences de n bits.

> **Exercice** Combien faut-il de bits d'information pour désigner une carte parmi les 78 d'un jeu de tarot ?

Le codage choisi dépendra de la nature des informations à coder (caractères, nombres entiers ou pas, positifs ou pas etc.) et les traitements que l'on désire effectuer sur ces informations.

1.1 Bases de numération

1.1.1 La notation positionnelle

Dans la vie courante nous utilisons le *système décimal positionnel* pour représenter les nombres ; *décimal* parce qu'il emploie dix symboles (les chiffres 0 à 9), *positionnel* parce que la valeur que l'on attache à ces chiffres dépend de leur position dans le nombre.

> **Exemple** Dans $4305,72$ nous accordons au 4 une valeur de milliers, au 3 des centaines, etc. La valeur de ce nombre se lit donc :
>
> $$(4 \times 1000) + (3 \times 100) + (0 \times 10) + (5 \times 1) + (7/10) + (2 \times 1/100)$$
>
> ou encore
>
> $$4 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 5 \times 10^0 + 7 \times 10^{-1} + 2 \times 10^{-2}$$

Ce système positionnel est utilisable avec d'autres bases de numération. Les informaticiens emploient fréquemment les systèmes suivants :

- *binaire*, ou base 2, qui utilise seulement deux chiffres 0 et 1,
- *octal*, ou base 8, (chiffres 0 à 7)
- *hexadécimal*, ou base 16, qui emploie les chiffres 0 à 9, puis les lettres A (qui représente la valeur 10), B=11, ...F=15

Losqu'il y a risque de confusion, on fait figurer la base de numération en indice ou entre parenthèses après le nombre concerné.

Le principe commun de toutes ces bases est le suivant : un nombre N écrit en base B est représenté par une suite de chiffres adjacents $x_n x_{n-1} \dots x_1 x_0$. Chacun de ces chiffres est un symbole qui a une valeur entière comprise entre 0 et $B-1$. A chaque position est attribué un poids qui

est une puissance de B. Le chiffre le plus à droite a un poids de $1 = B^0$, son voisin de $B = B^1$, le suivant de B^2 etc. La valeur du nombre N s'obtient en faisant la somme des produits des valeurs des chiffres par leur poids respectifs :

$$N = x_n \times B^N + x_{n-1} \times B^{n-1} + \ldots + x_1 \times B^1 + x_0 \times B^0$$

Exemple Expression décimale de $421(8)$

$$4 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 = 4 \times 64 + 2 \times 8 + 1 = \ldots(10)$$

Ceci se généralise facilement aux nombres "à virgule" : les chiffres de la "partie décimale" ont alors des poids successifs de $B^{-1} = 1/B, B^{-2} = 1/B^2, \ldots$.

1.1.2 Conversion vers le système décimal

La méthode usuelle de conversion découle immédiatement de la définition donnée plus haut.

Exemple Expression décimale de $N = 1001011(2)$ On écrit de droite à gauche, sous chacun des chiffres de N, les puissances successives de 2. On multiplie ensuite chaque chiffre par son poids, et on fait la somme.

$N =$	1	0	0	1	0	1	1	
$\times poids$	64	32	16	8	4	2	1	
$=$	64	0	0	8	0	2	1	$total = 75(10)$

1.1.3 De décimal vers base quelconque

Méthode des restes successifs

Diviser le nombre à convertir par la valeur de la base. Le reste de la division fournit le chiffre de droite, et recommencer avec le quotient. S'arrêter à 0.

Exemple Conversion de $1789(10)$ **en hexadécimal**
— $1789 = 111 \times 16 + 13$. On pose $x_0 = D$.
— $111 = 6 \times 16 + 15$. On pose $x_1 = F$.
— $6 = 0 \times 16 + 6$. On pose $x_2 = 6$.
— le résultat est $6FD(16)$

Méthode soustractive

Écrire une liste des puissances successives de la base, de droite à gauche. Chercher le plus grand poids contenu dans le nombre. Diviser le nombre par ce poids, écrire le quotient en dessous, et recommencer avec le reste. On met des 0 dans les autres positions.

On utilise cette méthode surtout avec le système binaire puisque la division ne nécessite en fait qu'une soustraction (le quotient est forcément 1).

Exemple Conversion de $1789(10)$ **en binaire**
La liste des poids est

$$\ldots 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1$$

— Comme $2048 > 1789 \geq 1024$, on met 1 sous 1024, et on soustrait ce poids $1789 - 1024 = 765$
— Comme $1024 > 765 \geq 512$, on met 1 sous 512, et on soustrait ce poids $765 - 512 = 253$
— on met 1 sous 128, et on soustrait $253 - 128 = 125$
— on met 1 sous 64, et on soustrait $125 - 64 = 61$
— on met 1 sous 32, et on soustrait $61 - 32 = 29$

— on met 1 sous 16, et on soustrait $29 - 16 = 13$
— on met 1 sous 8, et on soustrait $13 - 8 = 5$
— on met 1 sous 4, et on soustrait $5 - 4 = 1$
— on met 1 sous 1, et on arrête. Le résultat est 110 1111 1101(2).

1.1.4 Conversions entre bases 2, 8 et 16

Les nombres 8 et 16 étant des puissances entières de 2, il existe un moyen très rapide de convertir un nombre binaire en octal ou hexadécimal (et réciproquement), puisqu'à chaque chiffre octal (resp. hexadécimal) correspond une "tranche" de 3 (resp. 4) bits dans la représentation binaire du nombre.

Exemple Conversion de 11011111101(2) **en octal et hexadécimal** On découpe le nombre en tranches de 3 bits (matérialisées ici par des espaces) en partant des unités. On obtient 11 011 111 101(2). On lit ensuite ce nombre en traduisant chaque tranche par un chiffre octal, et on trouve 3375(8).

Si on découpe par tranches de 4 on obtient 110 1111 1101(2), ce qui donne $6FD(16)$.

Le système hexadécimal permet de coder des séquences binaires de façon 4 fois plus courte, les conversions se faisant sans calcul.

1.2 Arithmétique binaire

Les opérations arithmétiques usuelles peuvent se faire dans toutes les bases, selon des procédés très proches de l'arithmétique décimale. Nous ne présenterons ici que l'arithmétique et la multiplication binaire, les autres opérations (ainsi que la généralisation à d'autres bases) est laissée en exercice.

1.2.1 Addition binaire

Comme il n'y a que deux chiffres, la table d'addition se réduit à peu de choses

+	0	1
0	0	1
1	1	10

Pour additionner deux nombres A et B, on les écrit l'un sous l'autre, et on commence l'addition par la droite. L'addition de deux chiffres 1 engendre une retenue, que l'on propage au rang suivant.

Exemple Somme de 1011101 et 1110001

	1	1	1				1		*retenues*
		1	0	1	1	1	0	1	A
+		1	1	1	0	0	0	1	B
=	1	1	0	0	1	1	1	0	*Somme*

1.2.2 Multiplication binaire

Exemple Produit de 110011 et 1101

					1	1	0	1	1
×						1	1	0	1
					1	1	0	1	1
				0	0	0	0	0	
			1	1	0	1	1		
		1	1	0	1	1			
	1	0	1	0	1	1	1	1	1

A chaque étape, les multiplications partielles par chacun des chiffres du second nombre se ramènent soit à recopier le premier nombre avec un décalage, soit à inscrire des 0.

1.3 Codages des nombres entiers en binaire

Le problème est de coder des nombres avec une séquence binaire de longueur limitée. On sait déjà qu'avec un nombre fini de bits, on ne pourra représenter qu'un nombre fini de valeurs (2^N, où N est le nombre de bits).

1.3.1 Binaire pur

Si les nombres à représenter sont toujours positifs ou nuls, on pourra adopter le codage *binaire pur non-signé*, qui est une application directe de la numération binaire vue ci-dessus.

Sur un octet (8 bits) on écrira un nombre qui sera forcement compris entre 0 et $2^8 - 1 = 255$.

Exemple Le nombre 35(10) sera codé par 0010 0011. Le plus grand nombre représentable (ici 255) sera traduit par une suite de 1 : 1111 1111.

Exercice Quels nombres peut on coder sur 16 bits ? Sur 32 bits ?

1.3.2 Binaire signé

Si l'on veut représenter des nombres positifs ou négatifs, la première idée qui vient est de réserver un bit (par exemple celui de gauche) pour le signe, (0 pour positif, 1 pour négatif) et de coder la valeur absolue du nombre sur les bits restants.

Exemple Sur 8 bits, +35(10) sera codé par 0010 0011, et −35(10) par 1010 0011.

Quoique simple, ce procédé de codage n'est quasiment pas utilisé, car il ne permet pas de faire facilement des opérations sur les nombres ainsi codés (pour additionner deux nombres il faut examiner leurs signes puis additionner les valeurs absolues, ou soustraire la plus petite de la plus grande).

En outre, dans ce système le nombre 0 possède deux codages (+0 et −0). On ne code donc que $2^N - 1$ nombres distincts (de $-(2^{N-1} - 1)$ à $+2^{N-1} - 1$).

1.3.3 Représentation biaisée

L'idée est de décaler les entiers de l'intervalle choisi en leur soustrayant la plus petite valeur de cet intervalle. Cela permet de se ramener au codage des nombres positifs ou nuls.

Exemple On choisit de coder l'intervalle $-127 \ldots + 128$. Le codage du nombre −94 s'obtient en lui ajoutant 127, et en traduisant le résultat $-94 + 127 = 33$ en binaire, soit 0010 0001(2).

Pour additionner deux nombres en représentation biaisée, il faudra additionner leurs représentations biaisées, puis soustraire le biais.

Exercice Détailler le calcul de $2 + 3$ en représentation biaisée sur 4 bits. Que faut-il faire pour une soustraction ?

1.3.4 Binaire complément à deux

Ce procédé permet de coder, sur N bits, les nombres de -2^{N-1} à $+2^{N-1} - 1$:

— les nombres positifs (de 0 à $+2^{N-1} - 1$) sont écrits en binaire pur (on remarque que le bit de gauche sera alors forcément à 0)

— aux nombres négatifs (entre -2^{N-1} et -1), on ajoute 2^N. Le nombre obtenu est alors écrit en binaire pur. Comme il est compris entre 2^{N-1} et $2^N - 1$, son premier bit est à 1.

Exemple Codage de -35 sur un octet. Le nombre à coder étant négatif, on lui ajoute 256, et le problème se ramène donc à coder $256 - 35 = 221$ en binaire pur, soit 1101 1101.

On le voit, ce codage est basé sur l'arithmétique modulo 2^N. Puisque les nombres -positifs ou négatifs- sont représentés par leurs valeurs positives modulo 2^N, l'addition et la soustraction se font sans se préoccuper des signes : les circuits de calcul en seront grandement simplifiés. C'est l'avantage décisif qui a conduit à l'adoption générale de ce système.

Le dénomination de "complément à deux" provient de la remarque suivante : pour calculer l'opposé d'un nombre ainsi codé, il suffit d'ajouter 1 à son *complément à 1*, que l'on obtient en changeant les 1 en 0 et réciproquement.

Exemple Codage de -35 par complémentation à deux. En binaire pur 35(10) s'écrit -sur un octet- 0010 0011. Le complément à 1 est 1101 1100. En ajoutant 1 on trouve 1101 1101, qui est le codage attendu.

Remarque Cette opération est idempotente : si on complémente 1101 1101 on trouve 0010 0010, et en ajoutant 1 on obtient 0010 0011 qui est la représentation de +35. On utilise cette propriété pour le décodage : pour décoder un nombre écrit en binaire complément à deux

— si le premier bit est à 0, le signe est +, et la valeur absolue est obtenue par un décodage binaire non signé du nombre ;

— si le premier bit est à 1, le signe est $-$, et la valeur absolue s'obtient par décodage binaire non signé du complément à 2 de ce nombre.

Exemple Décodage de 1111 0100**.** Le signe est négatif. Le complément à 2 est $0000\ 1011 + 1 = 0000\ 1100$ qui, traduit en décimal vaut 12. Le nombre cherché est donc -12.

1.4 Décimal Codé Binaire

Cette notation hybride est utilisée pour le codage des grands nombres entiers ou à virgule fixe dans les machines destinées aux applications commerciales : chaque chiffre *décimal* est codé par une tranche de 4 bits.

Exemple Codage BCD de 14285739. En mettant des espaces pour bien voir les tranches :

0001 0100 0010 1000 0101 0111 0011 1001

Les procédés de codages et décodages entre décimal et BCD sont très simples, et nous verrons que les circuits d'addition ne sont pas beaucoup plus compliqués que dans le cas du binaire pur.

1.5 La représentation en virgule flottante

La représentation en virgule flottante permet de représenter des nombres réels très petits et très grands, avec une bonne précision.

1.5.1 La représentation flottante

L'idée de base est similaire à la notation des nombres sous la forme d'une mantisse multipliéee par une puissance de 10. Par exemple 12.34×10^{-30}.

Cette notation est très employée en physique. Elle revient à décrire un nombre par un couple (m, e) formé de la mantisse et de l'exposant, ici $(12.34, -30)$. Il y a plusieurs couples qui représentent le même nombre -par exemple $(1.234, -29), (1234, -32), \ldots$ - on peut donc convenir d'une représentation *normalisée*, par exemple en décidant que $1 \leq |m| < 10$. Tous les nombres (sauf 0) auront alors une représentation unique.

En binaire, c'est la même idée : le nombre sera de la forme $m \times 2^e$, avec $0 < |m| \leq 2$.

Exemple Normaliser le nombre -13 en vue d'une représentation binaire.
Comme 13 est compris entre $8 = 2^3$ et $16 = 2^4$, on prend $e = 3$ et $m = -13/8 = -1.625(10) = -1.101(2)$.

1.5.2 Le format flottant standard IEEE

Représentations normalisées

Le format standard IEEE simple précision code les nombres réels sur 32 bits (soit 4 octets).
— le premier bit S code le signe du nombre (0 :positif, 1 :négatif).
— 8 bits pour l'exposant (zone E). La représentation est biaisée : cette zone représente un nombre e compris dans l'intervalle $-127..+128$, codé en binaire non signé avec un décalage de 127 :

$$e = E - 127$$

Attention les "exposants" E=0 et E=127 sont réservés pour le codage de nombres "spéciaux" (voir plus loin).
— 23 bits pour la mantisse. Cette zone M ne contient que la partie décimale de la mantisse m, en effet la convention de normalisation nous assure que la partie entière est toujours 1. (Pour le codatge du 0, voir plus loin).

Exemple Coder -13 dans le format IEEE simple précision.
On sait déjà que $-13 = -1.101(2) \times 2^3$.
— Le nombre étant négatif, le signe S est 1.
— L'exposant est 3, on lui ajoute 127, et on obtient 130 qui, en binaire sur 8 bits s'écrit 10000010.
— La mantisse 1.101(2) privée de sa partie entière est codée sur 23 bits : 101 0000 0000 0000 0000 0000
Le nombre réel -13 est donc codé en binaire :

1100 0001 0101 0000 0000 0000 0000 0000

ou $C1500000_H$, si on préfère l'hexadécimal (plus maniable).

Exercice Décoder le nombre 12340000_H.

Exercice Quel est le plus grand nombre normalisé ? Le plus petit (en valeur absolue) ?

Les zéros, les infinis et les autres

Le nombre zéro est, par convention, représenté avec les zones E et M remplies de zéros, le signe S étant indifférent (il y a donc deux zéros …) :

x000 0000 0000 0000 0000 0000 0000 0000

Les deux infinis $+\infty$ et $-\infty$ ont leur zone E remplies de 1, et la zone M à 0 :

S111 1111 1000 0000 0000 0000 0000 0000

Certains codes spéciaux représentent des erreurs, on les appelle des NaN (Not a Number). On les reconnait à leur zone E remplie de 1, et leur zone M non nulle.

Précision du codage

L'éventail des nombres réels que l'on peut coder ainsi est très large, mais la précision n'est pas illimitée. On peut mesurer cette précision en calculant l'écart qu'il y a entre le nombre 1.0 et le plus petit nombre représentable qui lui soit immédiatement supérieur.

En représentation normalisée le nombre 1 s'écrit $1.00\ldots \times 2^0$. Comme la zone M contient les 23 chiffres "décimaux" de la mantisse, le nombre suivant s'obtient en remplaçant le chiffre le plus

à droite par un 1, ce qui correspond à $1 + 2^{-23}$. La précision est donc de 2^{-23}, soit à peu près 1.2×10^{-7}. On compte donc, au mieux, sur 6 chiffres significatifs.

Attention, la précision se perd très rapidement lors des calculs répétitifs, par exemple lors d'évaluation de séries de Taylor.

Pour avoir une meilleure précision on peut employer des nombres en double précision, la mantisse occupant alors 32 bits supplémentaires.

1.6 Codage des caractères

L'échange de données avec les périphériques d'entrée (clavier) et de sortie (écran, imprimantes) se fait caractère par caractère. Dans les premiers dispositifs utilisés (télétypes) les caractères étaient codés sur 6 bits (26 lettres + 10 chiffres + 28 symboles). La distinction entre majuscules et minuscules a exigé le passage à 7 bits (codes ASCII et EBCDIC), et le passage à 8 bits est rendu nécessaire pour la représentation des caractères accentués des langues européennes basées sur l'alphabet romain (ISO 8859 Latin1). Il existe également un jeu de caractères "universel" sur 16 bits appelé UNICODE, qui regroupe les principaux alphabets utilisés dans le monde.

1.6.1 Code ASCII 7 bits

Le code ASCII (American Standard Code for Information Interchange) utilise une table de 128 positions, que l'on repère par leur numéro en décimal. Si on met cette table sous forme de 8 colonnes, de 16 lignes, les 2 premières colonnes contiennent des caractères dits "non imprimables" suivis par des symboles divers. Les chiffres commencent en quatrième colonne (position 48). Les majuscules sont en cinquième et sixième colonne (à partir de 65), les minuscules dans les deux dernières (à partir de 97).

En pratique les caractères ASCII sont transmis dans des octets, le bit supplémentaire étant ignoré ou utilisé pour effectuer un *contrôle de parité* :

- bit de parité à 0 : l'émetteur met 0 dans le bit de gauche. Si le récepteur y lit un 1, il y a eu une erreur de transmission.
- bit de parité à 1 (semblable au précédent).
- parité paire : l'émetteur met 0 ou 1 dans le bit de gauche de façon à ce que le nombre total de 1 dans l'octet soit pair. Si le récepteur reçoit un octet avec un nombre impair de 1, il y a eu une erreur.
- parité impaire : comme ci-dessus.

Ce code suffit pour la langue anglaise, qui ignore les accents.

1.6.2 Code EBCDIC

Autrefois préconisé par le principal constructeur d'ordinateurs de l'époque (IBM). En désuétude maintenant.

1.6.3 Code ANSI/ISO Latin 8859-1

L'existence du marché européen a rendu nécessaire l'adoption de codes contenant les lettres accentuées de l'alphabet latin. Divers codes 8 bits ont été utilisés, qui sont (à peu de choses près) des extensions du code ASCII 7 bits : ASCII-IBM (pour les PC sous DOS), ANSI (PC sous Windows), ASCII-McIntosh etc. Ces codes sont bien sûr incompatibles entre eux.

Actuellement un consensus se dégage (c'est une conséquence de la profusion des réseaux hétérogènes) pour l'utilisation du code ANSI, normalisé par l'ISO (International Standard Organization) sous la référence ISO-8859-1 (ou ISO-Latin-1). Ce code 8 bits contient le nécessaire pour les alphabets dérivés de l'alphabet romain : langues latines mais aussi anglo-saxonnes, nordiques, polonais, turc, vietnamien etc. (Avec l'oubli fâcheux du caractère Œ).

Exercice Chercher dans les documentations accessibles par le Web si le caractère "euro" a bien été ajouté.

1.6.4 En attendant UNICODE...

Le code 16 bits UNICODE permet de représenter 65536 caractères, ce qui suffit pour les alphabets les plus répandus (romain, cyrillique, arabe, hébreu, grec, hiragana, katagana, coréen ...), mais pas pour les idéogrammes chinois. L'inconvénient d'UNICODE est de nécessiter deux fois plus de place pour représenter un caractère.

Chapitre 2

Eléments de technologie

Nous avons vu comment coder l'information par des suites de valeurs binaires. Nous allons voir maintenant comment représenter ces valeurs binaires par des signaux électriques afin de les manipuler par des circuits électroniques.

2.1 Représentations de la logique binaire

2.1.1 Représentation des signaux

Il existe plusieurs façons de coder les valeurs 0 et 1. La plus simple, que nous adopterons dans la suite, est appelée *logique positive* et consiste à représenter 1 par la présence d'une tension (par exemple +5v) et 0 par une tension nulle. [1]

2.1.2 Génération d'un signal

Il est facile de fournir de tels signaux à un circuit : il suffit d'un interrupteur à deux positions (utilisé pour les va-et-vient) ; dans une position la sortie de l'interrupteur transmettra la tension +V, dans l'autre 0v (voir figure 2.1)

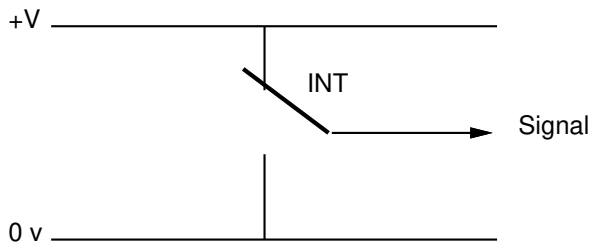

FIGURE 2.1 – Génération d'un signal d'entrée

En pratique, on utilise plutôt (figure 2.2) un interrupteur simple dont la sortie est reliée à une *résistance de rappel* R assez forte. Lorsque l'interrupteur est fermé, la sortie est à +V, lorsqu'il est ouvert la résistance ramène la tension de sortie vers 0. De plus ceci évite d'avoir une entrée "en l'air" quand l'interrupteur est entre deux positions.

1. Une logique *négative* aura des signaux inversés : 0v représente la valeur 1, +5v la valeur 0.

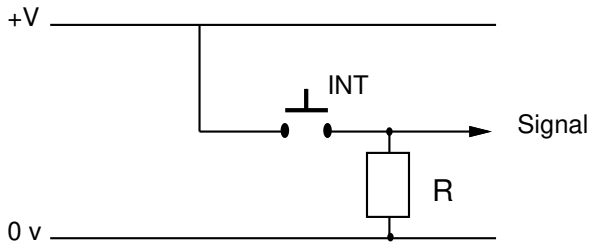

FIGURE 2.2 – Génération d'un signal d'entrée (2)

2.1.3 Observation d'un signal

Pour observer la sortie d'un circuit on peut utiliser (figure 2.3) une ampoule du voltage voulu (5v = lampe de poche).

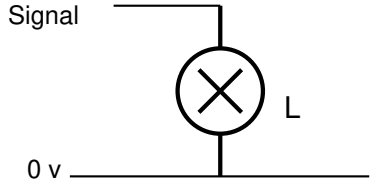

FIGURE 2.3 – Ampoule témoin

On préfèrera généralement employer des diodes électro-luminescentes (LED = Light Emitting Diode), qui coûtent moins cher et nécessitent un courant moindre (voir figure 2.4). 0

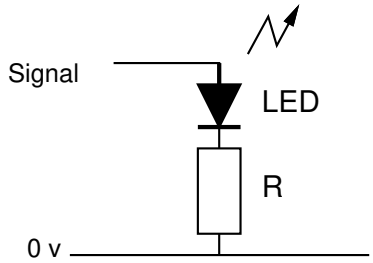

FIGURE 2.4 – LED témoin

Encore mieux, pour éviter de trop "tirer" sur la sortie du circuit observé, on pourra utiliser un transistor en amplification (voir plus loin 2.2.3).

2.2 Notions rudimentaires d'électronique

Quelques notions sommaires d'électronique sont nécessaires pour la compréhension des circuits logiques de base. Le lecteur est supposé connaître la loi d'Ohm et des rudiments d'électricité.

2.2.1 La diode

La diode à semi-conducteurs est un composant électronique muni de deux bornes : l'*anode* et la *cathode* (voir 2.5). La propriété fondamentale de la diode est d'opposer qu'une résistance très

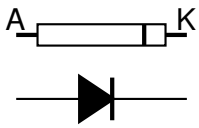

FIGURE 2.5 – La diode : apparence physique et symbole

faible lorsqu'elle est traversée par un courant de l'anode vers la cathode (sens passant), et une résistance très forte dans le sens inverse. Voir figure 2.6.

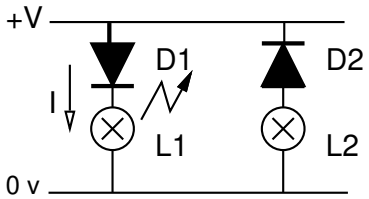

FIGURE 2.6 – Diodes passante (D1) et bloquée (D2)

Pour analyser les circuits logiques nous considérerons que les diodes sont parfaites, c'est-à-dire ayant une résistance nulle dans l'état passant, et infinie dans l'état bloqué.

> **Remarque importante.** Lorsque nous construirons des circuits nous aurons soin d'éviter de faire traverser les diodes par des courants trop forts (risque de claquage). Il faudra donc de les monter en série avec des résistances pour limiter le courant.

2.2.2 La diode électro-luminescente

Les diodes électro-luminescentes (LED = Light Emitting Diod, voir fig. 2.7) émettent de la lumière quand elles sont traversées par un courant de l'ordre de 10 à 20 mA dans le sens passant. Ces diodes offrant une résistance très faible, il conviendra de les monter en série avec une *résistance limitatrice de courant* d'une valeur suffisante ($R = U/I = 5v/10mA = 500\Omega$).

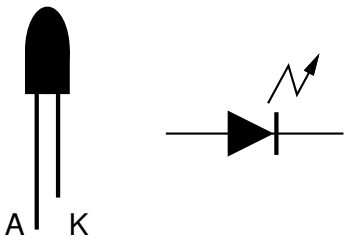

FIGURE 2.7 – LED : apparence physique et schéma

2.2.3 Le transistor

Le transistor[2] est un composant à 3 pattes : l'émetteur, le collecteur et la base (voir figure 2.9).

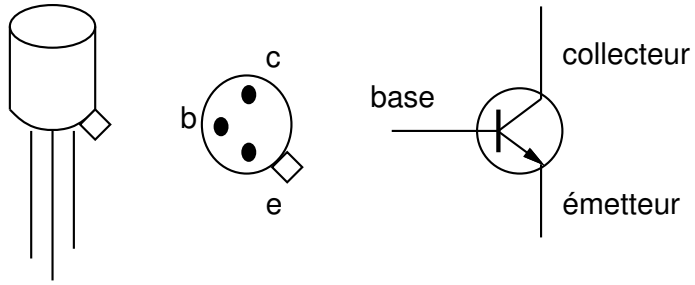

FIGURE 2.8 – Transistor NPN : boitier, broches (vues de dessous), schéma

Dans les montages usuels, le collecteur est relié à la tension d'alimentation $+V$ au travers d'une résistance R_c, l'émetteur étant relié à la masse (0v). Le courant I_b qui traverse la base permet de contrôler l'intensité de celui qui traverse l'émetteur et le collecteur.

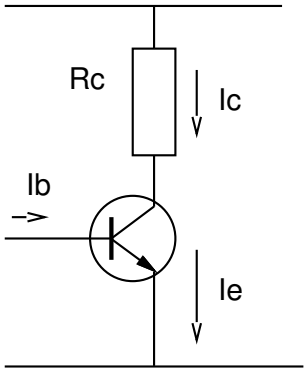

FIGURE 2.9 – Polarisation d'un transistor

Lorsque le transistor est utilisé en amplification, les courants sont liés par les équations

$$\begin{aligned} I_e &= I_c + I_b \\ I_c &= \beta . I_b + I_{ce} \end{aligned}$$

Les deux constantes β et I_{ce} dépendent du transistor : le gain β est de l'ordre de 100, le courant I_{ce} vaut quelques micro-ampères pour les transistors au silicium, et quelques nano-ampères pour les transistors au germanium. En général on le néglige pour les calculs.

Les montages logiques utilisent le mode de fonctionnement *bloqué-saturé*, en débordant largement de la plage de valeurs où les formules ci-dessus sont valides : si on applique une tension assez forte (proche de +V) à la base, l'intensité I_b est maximum et le transistor n'oppose qu'une résistance très faible entre émetteur et collecteur : le transistor est saturé. Si on applique une tension nulle à la base, le courant I_c sera négligeable : le transistor est alors dit bloqué.

Ceci justifie le montage de la figure 2.10 : lorsque l'entrée est à +V, le transistor est saturé, donc il laisse passer le courant à travers la LED qui s'éclaire. Lorsque l'entrée est au niveau bas (0v), le transistor est bloqué et la LED reste éteinte. En raison des propriétés amplificatrices du transistor,

2. Pour simplifier l'exposé nous n'évoquerons que les transistors NPN.

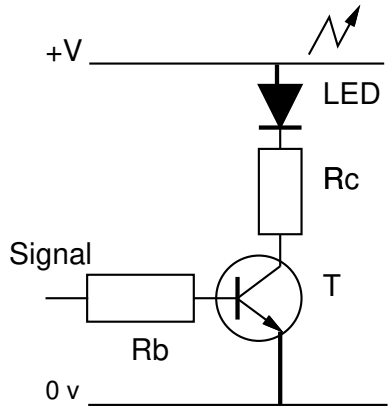

FIGURE 2.10 – LED amplifiée par un transistor

le courant de base n'a pas besoin d'être très élevé (de l'ordre de I_c/β), par conséquent on peut mettre une résistance limitatrice sur la base. Cette résistance évite de trop "tirer de courant" du signal que l'on observe, ce qui risquerait de perturber son fonctionnement.

2.3 Portes logiques

En assemblant ces composants on obtient des *portes logiques* qui combinent les signaux.

2.3.1 Porte OU

Montage à diodes

Deux diodes et une résistance suffisent pour réaliser une *porte OU* selon le schéma de la figure 2.11.

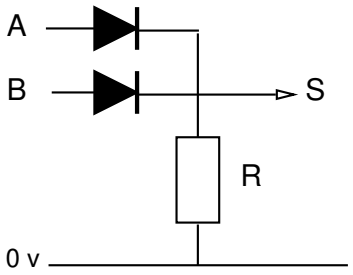

FIGURE 2.11 – Porte logique OU à diodes

Analyse du montage

Supposons que les deux entrées A et B soient portées au potentiel +V. Les deux diodes sont dans le sens passant, et laissent donc passer le courant. Tout se passe alors comme si la sortie S était reliée à la même tension +V.

Si A et B sont reliées à la masse (0v), les diodes ne conduisent pas le courant. La sortie S est donc ramené à 0v par la résistance de rappel.

Si une des entrées est à +V et l'autre à la masse, la diode passante suffit, comme dans le premier cas, à amener la tension +V sur S.

Table de vérité

Si nous prenons la convention de logique positive, la tension +V correspond à la valeur logique "vrai" (notée 1) et 0v à "faux" (0). La table de vérité de l'opération "+" ainsi obtenue est celle de l'opérateur "OU logique" :

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

La sortie est à 1 si au moins une des entrées est à 1.

Porte OU multiple

En reliant plusieurs entrées de la même façon on obtient une porte OU multiple. Son fonctionnement se résume en une phrase : la sortie est à 1 si au moins une des entrées est à 1.

Dans les schémas logiques, on représente les portes logiques par un symbole (figure 2.12)

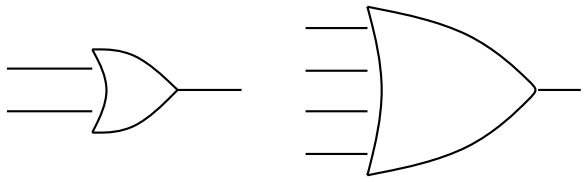

FIGURE 2.12 – Symboles des portes OU

Montage à transistors

Le montage de la figure 2.13 remplit la même fonction. L'analyse en est laissée au lecteur.

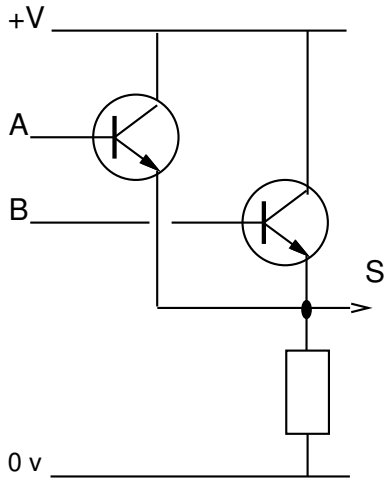

FIGURE 2.13 – Porte OU à transistors en parallèle

2.3.2 Porte ET

Montage à diodes

Le schéma (figure 2.14) n'est pas sans rappeler celui de la porte OU.

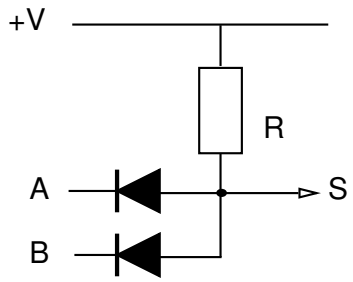

FIGURE 2.14 – Porte ET à diodes

Analyse du montage

Chaque diode ne peut être passante que si l'entrée associée est à 0. Si une des entrées est à 0, la sortie sera donc également à 0. Par contre si les deux entrées sont à +V, la sortie S sera à +V au travers de la résistance de rappel.

Table de vérité

L'opération logique "ET" (noté "$\cdot$") correspondant à ce montage possède donc la table de vérité suivante :

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

La sortie est à 1 si les deux entrées sont à 1.
On représente les portes ET (à deux ou plusieurs entrées) par un symbole (figure 2.15)

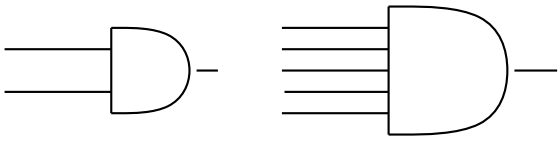

FIGURE 2.15 – Symboles des portes ET

Montage à transistors

Le montage de la figure 2.16 remplit la même fonction. L'analyse en est laissée au lecteur.

2.3.3 Porte NON

Montage

Dans la figure 2.17 on utilise un transistor en mode bloqué/saturé.

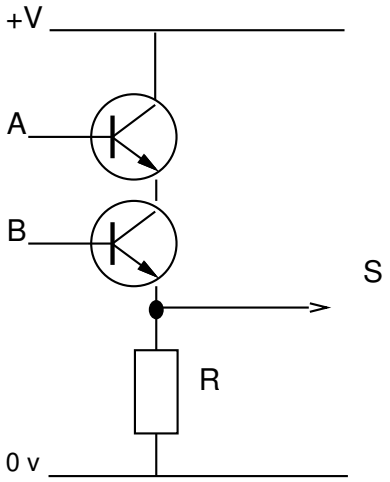

FIGURE 2.16 – Porte ET à transistors en série

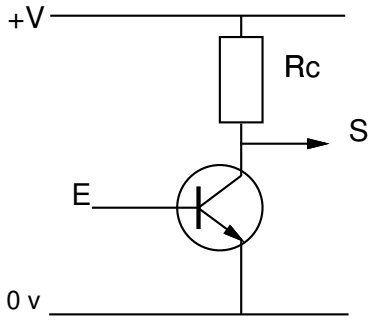

FIGURE 2.17 – Porte NON à transistor

Analyse du montage

Lorsque l'entrée est au niveau haut, le transistor est saturé. La sortie est alors reliée à la masse par l'intermédiaire de la résistance interne très faible du transistor : la sortie S est au niveau bas.

Lorsque l'entrée est au niveau bas, le transistor est bloqué. La résistance de rappel ramène donc la sortie au niveau haut.

Table de vérité

La table de l'opérateur NON "¬" ainsi obtenu est

A	$\neg A$
0	1
1	0

La sortie est à 1 si et seulement si l'entrée est à 0.

On symbolise cet opérateur (figure 2.18) par un triangle (indiquant par convention l'amplification) suivi d'un rond (négation).

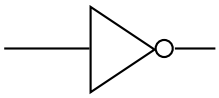

FIGURE 2.18 – Symbole de la Porte NON

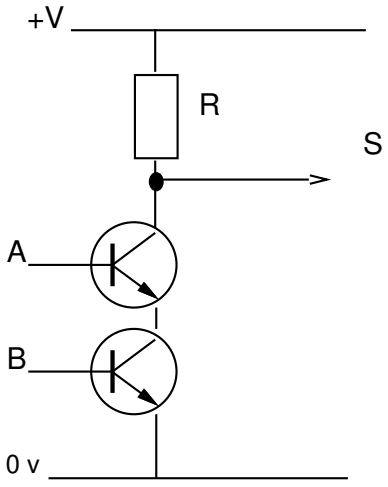

FIGURE 2.19 – Porte NAND

2.3.4 Porte NON-ET (NAND)

Le montage de la figure 2.19 réalise une fonction dont la table de vérité est :

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

On remarque aisément que l'on a $S = \overline{A.B}$, et on appelle cet opérateur le NON-ET (ou "nand").

Exercice Donnez un schéma de cet opérateur utilisant 2 diodes, une résistance et un transistor.

On le représente par un symbole "ET" suivi du rond qui indique la négation (figure 2.20).

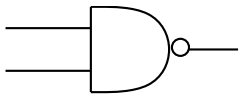

FIGURE 2.20 – Porte NON-ET

La porte NAND a un intérêt pratique évident : elle permet de reconstituer tous les autres types de portes.

— La porte NON, puisque $\overline{A} = A \; nand \; 1$

— La porte ET, puisque $A.B = \overline{(A \; nand \; B)} = (A \; nand \; B) \; nand \; 1$

— La porte OU, puisque $A+B = \overline{\overline{A}.\overline{B}}$ et donc $A+B = ((A \; nand \; 1) \; nand \; (B \; nand \; 1)) \; nand \; 1$.

2.3.5 Porte NON-OU (NOR)

La fonction “non-ou” (symbolisée fig. 2.21) est définie de la même façon, par l'équation

$$nor(A, B) = \overline{A + B}$$

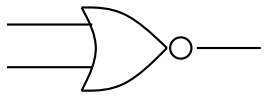

FIGURE 2.21 – Symbole de la porte NOR (non-ou)

Exercice Proposez une porte NOR à transistors.

2.3.6 Porte OU-exclusif (XOR)

La fonction XOR “ou-exclusif” (fig. 2.22) est souvent notée $\oplus$. On la définit par :

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

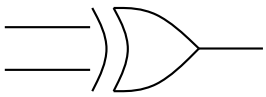

FIGURE 2.22 – Symbole de la porte XOR (ou-exclusif)

Exercice Proposez une porte XOR à transistors.

2.3.7 Le circuit intégré CMOS 4011

Les circuits intégrés logiques sont des boîtiers qui renferment plusieurs portes logiques interconnectées. Il existe une grande quantité de circuits intégrés logiques, renfermant des portes ET, OU, NAND, etc. Pour des petits montages, il est économique d'utiliser un seul type de circuit, comme le circuit CMOS 4011, qui contient 4 portes NAND.

Brochage

Le circuit 4011 se présente sous forme d'un boîtier DIL (Dual in line) à 14 broches (voir figure 2.23). Une encoche sur le côté gauche permet de reconnaître le sens du circuit.

Les broches 14 (en haut à gauche) et 7 (en bas à droite) servent à l'alimentation du circuit (14 : +V, 7 : masse). Les autres sont les entrées et sorties des 4 portes NAND :

— porte 1 : entrées 12 et 13, sortie 11 ;
— porte 2 : entrées 8 et 9, sortie 10 ;
— porte 3 : entrées 1 et 2, sortie 3 ;
— porte 4 : entrées 5 et 6, sortie 4 ;

La figure 2.24 montre comment réaliser une porte OU avec ce circuit.

Exercice Fonctions binaires usuelles Montrez comment réaliser les expressions $A.B$, $A \oplus B$, $A + \overline{B}$ à l'aide d'un circuit 4011.

Exercice Fonction majorité Réalisez la fonction $maj(A, B, C)$, dont le résultat est 1 si au moins deux entrées sont à 1, à l'aide d'un circuit 4011 (ou plusieurs).

Exercice Réalisez la fonction $f(A, B, C) =$ si $A = 1$ alors $B.C$ sinon $B + C$.

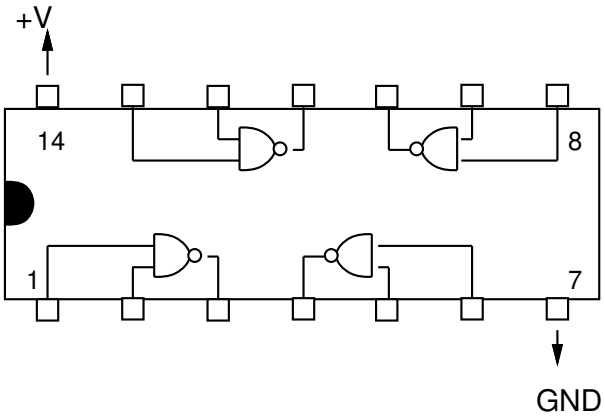

FIGURE 2.23 – Circuit CMOS 4011 vu de dessus

Conditions d'emploi des circuits CMOS

Les circuits de type CMOS présentent certains avantages pour les montages expérimentaux :
- ils acceptent une tension d'alimentation entre 3 et 15 V ;
- leur consommation est très faible (de l'ordre de 0,1 mW par porte) ;
- les entrées des circuits CMOS ont une impédance très élevée : on peut relier de nombreuses entrées sur la sortie d'une porte sans craindre de "tirer" trop de courant de celle-ci.

Par contre, ces circuit sont sensibles à l'électricité statique [3]. De plus, la propagation des signaux de l'entrée à la sortie d'une porte est plus lente (20-40 ns) qu'avec d'autres familles de circuits, comme les TTL.

Emploi des circuits TTL

Les circuits TTL (transistor-transistor-logic) sont très utilisés pour les réalisations professionnelles. A titre d'exemple, les caractéristiques de la série SN74 sont :
- tension nominale d'alimentation de $5v \pm 0.5V$, risque de claquage à partir de 7V
- fonctionnement entre 0 et 70^oC
- puissance par porte de l'ordre de 10 mW
- courant d'entrée de l'ordre de $1.5mA$
- temps de propagation de l'ordre de 10 à 20 ns.

3. En particulier "effet d'antenne" lorsqu'on approche la main d'un circuit dont une des entrées est restée "en l'air"

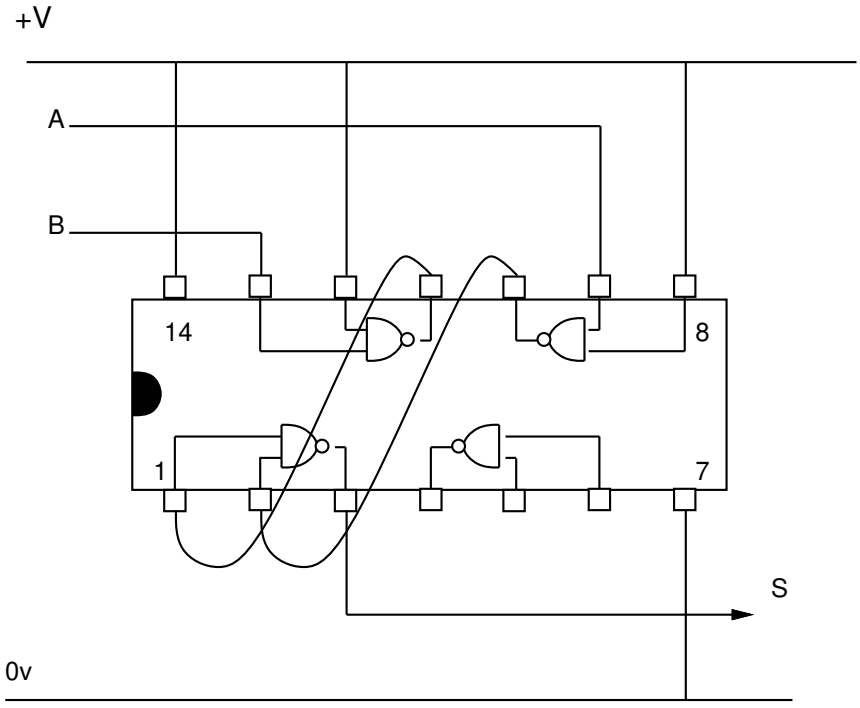

FIGURE 2.24 – Porte OU avec CMOS 4011

Chapitre 3

Algèbre de Boole et circuits logiques

En combinant les portes logiques, et en prenant garde de ne pas faire de boucles dans les circuits [1], on réalise des circuits dont les sorties ne dépendent que des valeurs des entrées. On pourra alors décrire la fonction réalisée par ce circuit par sa table de vérité.

L'étude des "tables de vérité" a été entreprise il y a plus d'un siècle, et constitue ce qu'on appelle l'algèbre de Boole [2].

Les propriétés de cette algèbre nous permettront de *raisonner* sur les circuits, et donc de les construire rigoureusement, plutôt que par la (coûteuse) méthode des essais et erreurs. En particulier nous verrons des méthodes systématiques pour simplifier les circuits.

3.1 Définitions

On appelle *fonction booléenne* toute fonction de $\{0,1\}^n$ dans $\{0,1\}$.

Il y a 2^{2^n} fonctions booléennes à n variables : en effet pour décrire une fonction booléenne à n variables il suffit de remplir les 2^n cases de sa table de vérité, chaque case pouvant contenir 0 ou 1.

Une *expression booléenne* est un terme construit à partir de variables (prises dans un ensemble V), de constantes 0 ou 1, et d'opérateurs ET, OU, et NON. Exemple : $a + \overline{a.\overline{b}} + b$.

Une expression booléenne *représente* une fonction booléenne ; nous dirons que deux expressions sont équivalentes si elles dénotent la même fonction.

Exemple Si nous construisons la table de vérité de l'expression ci-dessus, et de l'expression $\overline{a} + b$:

a	b	$\overline{b}$	$a.\overline{b}$	$\overline{a.\overline{b}}$	$\overline{a.\overline{b}} + b$	$\overline{a}$	$\overline{a} + b$
0	0	1	0	1	1	1	1
0	1	0	0	1	1	1	1
1	0	1	1	0	0	0	0
1	1	0	0	1	1	0	1

nous constatons qu'elles coïncident : les deux expressions représentent la même fonction.

Par commodité, on appelle *somme* toute expression de la forme $t_1 + t_2 + \ldots + t_n$. Elle peut se réduire à un seul terme, ou aucun (dans ce cas c'est 0). Un *produit* est de la forme $t_1.t_2 \ldots t_n$. Le produit vide est 1.

1. les circuits combinatoires (sans boucles) et séquentiels (avec boucles) seront étudiés dans les chapitres suivants
2. L'oeuvre la plus célèbre de George Boole (1815-1864) est intitulée *Une investigation dans les lois de la pensée, sur lesquelles sont fondées les théories mathématiques de la logique et des probabilités* (1854), plus connue sous le titre abrégé *Les lois de la pensée*

Un monôme est un produit de variables ou de négations de variables, chaque variable n'apparaissant qu'au plus une fois.

Exemple de monômes : $x, \overline{y}, x.\overline{y}.z, 1$

Remarque À partir d'un ensemble de n variables on peut construire $C_n^k.2^k$ monômes distincts à k variables : pour choisir un tel monôme il suffit de sélectionner k variables parmi n (d'où le coefficient binomial), et d'attribuer ou non à chacune d'entre elles une barre (2^k possibilités).

Il y a en tout 3^n monômes, puisqu'il suffit de décider, pour chacune des variables, de la faire figurer telle quelle, avec une barre, ou pas du tout. Ceci prouve l'égalité :

$$\sum_{k=0}^{n} C_n^k.2^k = 3^n$$

que l'on peut d'ailleurs retrouver en développant $(1+2)^n$ à l'aide des formules connues.

Un monôme est *complet* par rapport à un ensemble donné de variables si toutes les variables de cet ensemble apparaîssent une fois. Il y a 2^n monômes complets sur un ensemble de n variables, chaque monôme complet correspondant à une des cases d'une table de vérité à n variables.

Une expression est *en forme canonique* si elle est écrite sous forme d'une somme sans répétition de monômes complets.

Proposition Toute fonction booléenne peut être construite par composition d'opérations OU, ET et NON.

Preuve Toute fonction booléenne $f(a, b, c \ldots)$ peut être décrite par sa table de vérité. Chaque case de cette table correspond à un *monôme complet*. Une expression de la fonction f s'obtient en faisant la somme des monômes complets dont la valeur est 1.

Exemple Considérons une fonction maj (majorité) à trois variables a, b, c, qui vaut 1 quand au moins deux des entrées sont à un. On dresse la table de vérité de maj, en faisant figurer dans la marge les monômes correspondants :

a	b	c	$maj(a,b,c)$	monôme
0	0	0	0	$\overline{a}\overline{b}\overline{c}$
0	0	1	0	$\overline{a}\overline{b}c$
0	1	0	0	$\overline{a}b\overline{c}$
0	1	1	1	$\overline{a}bc$
1	0	0	0	$a\overline{b}\overline{c}$
1	0	1	1	$a\overline{b}c$
1	1	0	1	$ab\overline{c}$
1	1	1	1	abc

et on obtient l'expression en forme canonique

$$maj(a, b, c) = \overline{a}bc + a\overline{b}c + ab\overline{c} + abc$$

3.2 Propriétés des algèbres de Boole

Les operations ET, OU, NON definies sur l'ensemble $\{0,1\}$ possèdent les propriétés suivantes :

$\overline{\overline{A}}$	$=$	A					double négation
$\overline{0}$	$=$	1		$\overline{1}$	$=$	0	constantes
$\overline{A+B}$	$=$	$\overline{A}.\overline{B}$		$\overline{A.B}$	$=$	$\overline{A}+\overline{B}$	dualité
$(A+B)+C$	$=$	$A+(B+C)$		$(AB)C$	$=$	$A(BC)$	associativité
$A+B$	$=$	$B+A$		AB	$=$	BA	commutativité
$(A+B)C$	$=$	$AC+BC$		$AB+C$	$=$	$(A+C)(B+C)$	distributivité
$A+A$	$=$	A		AA	$=$	A	idempotence
$A+0$	$=$	A		$A.1$	$=$	A	éléments neutres
$A+1$	$=$	1		$A.0$	$=$	0	absorption
$A+\overline{A}$	$=$	1		$A.\overline{A}$	$=$	0	complémentaires

Remarques

- Chaque égalité peut-être vérifée en construisant les tables de vérité de ses parties gauche et droite.
- La colonne de droite contient des équations *duales* de celles de la colonne de gauche, obtenues en intervertissant $+$ et $\cdot$, 0 et 1. On peut passer d'une équation à l'équation duale en utilisant seulement $\overline{\overline{A}} = A$, $\overline{0} = 1$ et $\overline{A+B} = \overline{A}.\overline{B}$.
- On voit assez facilement que ces équations permettent de développer n'importe quel terme sous forme d'une expression canonique. L'expression canonique d'une fonction étant unique (à la commutation de $+$ et . près), ces équations suffisent donc pour montrer l'équivalence de deux expressions.
- Les lois de dualité sont appelées aussi lois de De Morgan. [3]

Algèbre de Boole

On appelle *algèbre de Boole* tout ensemble qui possède deux éléments 0 et 1 et des opérations $+, \cdot, \neg$ qui satisfont les équations ci-dessus.

> **Exercice** Montrez qu'il n'y a pas d'algèbre de Boole à 3 éléments $\{0,1,2\}$. (Indication : que vaut $\overline{2}$?).

On voit facilement que le produit $A \times B$ de deux algèbres de Boole est également une algèbre de Boole. Les opérations sur le produit sont définies par $\overline{(a,b)} = (\overline{a},\overline{b})$, $(a,b)+(a',b') = (a+a', b+b')$, $(a,b).(a',b') = (a.a', b.b')$. Dans le sens inverse, un théorême important et difficile (Stone) dit que toute algèbre de Boole se ramène en fait à un produit d'algèbres de Boole à 2 éléments [4].

3.3 Simplification des expressions

A partir d'une expression booléenne on pourra facilement fabriquer un circuit. Pour réaliser une même fonction, on aura tout intérêt à avoir des circuits qui utilisent le moins possible de portes logiques, pour des raisons de simplicité, de coût, de taille du circuit et de consommation de courant.

3. Les travaux d'Augustus de Morgan (1806-1871) influencèrent fortement George Boole, et ses vives recommandations permirent à ce dernier, bien qu'autodidacte, d'obtenir la Chaire de Mathématiques du Queen's College de Cork.

4. Ce produit peut être infini, mais c'est une autre histoire

On peut essayer de simplifier les circuits en utilisant les équations vues plus haut. Par exemple, pour la fonction maj :

$$\begin{array}{rcll} maj(a,b,c) & = & \overline{a}bc + a\overline{b}c + ab\overline{c} + abc & \\ & = & \overline{a}bc + a\overline{b}c + ab\overline{c} + abc + abc + abc & \text{(idempotence)} \\ & = & \overline{a}bc + abc + a\overline{b}c + abc + ab\overline{c} + abc & \text{(commutativité)} \\ & = & (\overline{a}+a)bc + a(\overline{b}+b)c + ab(\overline{c}+c) & \text{(distributivité)} \\ & = & 1.bc + a.1.c + ab.1 & \text{(complémentarité)} \\ & = & bc + ac + ab & \text{(éléments neutres)} \end{array}$$

Mais la difficulté est alors de mener le calcul vers une expression simple que l'on ne connait pas a priori.

Problème Donnez une expression aussi simple que possible de chacune des $2^{2^2} = 16$ fonctions à deux variables.

Déterminez celles qui sont croissantes, c'est à dire telles que $x \leq x'$ et $y \leq y'$ entraîne $f(x,y) \leq f(x',y')$. Pour chacune d'elles montrez qu'elle peut s'exprimer (sans utiliser de négation) par une somme de produits de variables.

Réciproquement, montrez que toute fonction dont l'expression contient des ET et des OU mais pas de NON est nécessairement croissante.

Généralisez au cas des fonctions ayant un nombre quelconque de variables.

3.4 Méthode de Karnaugh

La méthode de Karnaugh est une méthode visuelle pour trouver une expression simple d'une fonction booléenne de quelques variables (jusquà 6). [5].

Elle repose sur une présentation particulière de la table de vérité de la fonction étudiée. Voici la disposition adoptée pour les fonctions de 3 et 4 variables :

ab \ c	0	1
00		
01		
11		
10		

ab \ cd	00	01	11	10
00				
01				
11				
10				

On remarque que dans ces tables, deux cases adjacentes ne diffèrent que le changement de valeur d'une seule variable. Comme chaque case correspond à un monôme complet, un groupement de 2 cases adjacentes représentera un monôme à $n-1$ variables.

Exemple Les deux cases du bas de la première table correspondent respectivement à $a\overline{b}\overline{c}$ et $a\overline{b}c$, qui diffèrent par la variable c. En les regroupant on obtient $a\overline{b}\overline{c} + a\overline{b}c = a\overline{b}(\overline{c}+c) = a\overline{b}$

Attention : il faut également considérer les bords opposés comme étant adjacents, par exemple les cases $\overline{a}\overline{b}\overline{c}$ (en haut à gauche) et $a\overline{b}\overline{c}$ (en bas à gauche) se regroupent en $b\overline{c}$.

Les monômes à $n-2$ variables sont visualisés sous forme de carrés, ou de rectangles 1×4.

Exemple On a $bd = (a+\overline{a})b(c+\overline{c})d$. En développant cette expression on trouve une somme de 4 monômes complets qui occupent le milieu de la seconde table. Les 4 coins de cette même table forment (virtuellement) un carré dont l'expression est $\overline{a}\overline{d}$. La colonne de droite correspond à $c\overline{d}$, etc.

De même les produits de $n-3$ variables occupent 8 cases disposées en rectangles 2×4.

La méthode de Karnaugh consiste à écrire la table de vérité de la fonction dans la table, puis procéder à des regroupements que l'on entoure et dont on écrit l'expression au fur et à mesure.

5. Nous ne montrerons ici que la méthode pour 3 et 4 variables, le passage à 5 ou 6 variables nécessite une bonne vision dans l'espace

Algorithme L'algorithme est le suivant :

- Si tous les 1s ont été entourés : arrêter.
- trouver, visuellement, le plus gros regroupement possible contenant au moins un 1 non entouré.
- l'entourer sur le tableau, et écrire son expression
- recommencer

La figure 3.1 illustre le déroulement possible de la méthode sur l'exemple de la fonction majorité. Le résultat obtenu est $BC + AB + AC$.

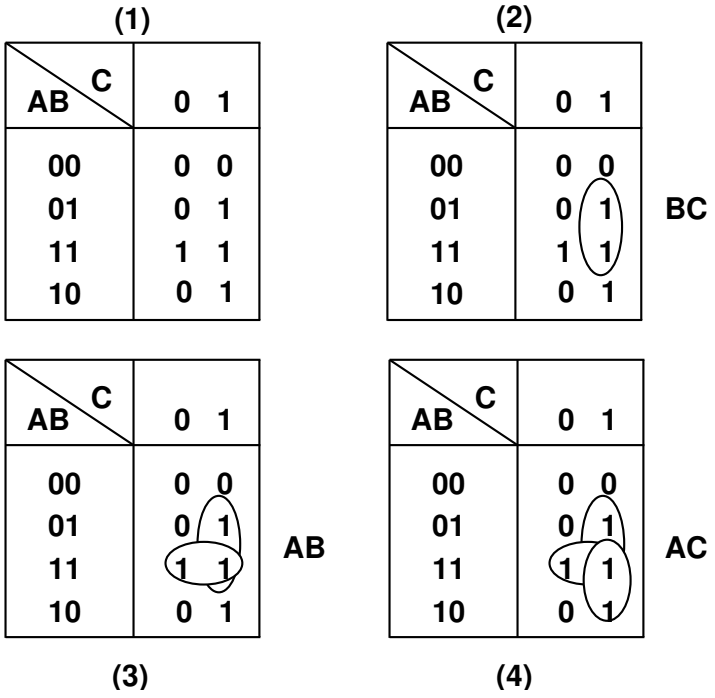

FIGURE 3.1 – Méthode de Karnaugh : exemple

Exercice Le vote Une commission est composé d'un président P et trois membres A,B,C. Cette commission doit se prononcer sur un vote par oui ou non à la majorité. Personne ne peut s'abstenir. En cas d'égalité entre les oui et les non, la voix du président est prépondérante. Em utilisant la méthode de Karnaugh, donnez une expression simple de la fonction $vote(A, B, C, P)$.

Exercice Décodage 7 segments hexadécimal Un afficheur 7 segments est composé de 7 diodes électro-luminescentes notés a,b,…g (voir figure 3.2). Un nombre est fourni, codé en binaire sur 4 bits $x_3x_2x_1x_0$, qui devra être affiché. Donnez l'expression des 7 fonctions $a(x_3, x_2, x_1, x_0), b(x_3, x_2, x_1, x_0), \ldots$

La méthode de Karnaugh est également applicable dans le cas des fonctions incomplètement spécifiées, c'est-à-dire qui ont des "cases vides" dans lesquelles on peut mettre ce que l'on veut, parce que cela correspond à des situations qui ne peuvent pas se produire.

La méthode est alors la suivante : former les plus gros paquets possibles contenant au moins un 1 non entouré et aucun 0.

Exercice Le dé électronique (fig. 3.3) possède trois entrées, par lesquelles arrivent des nombres de 1 à 6 codés en binaire. Donnez une fonction pour chacune des lampes qui indiquera si elle doit être allumée ou éteinte.

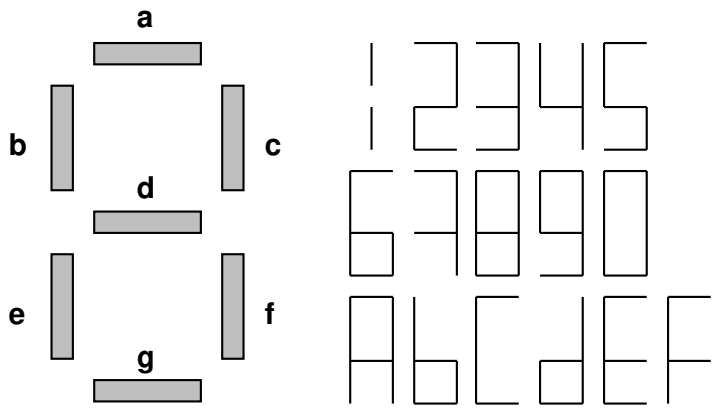

FIGURE 3.2 – Afficheur 7 segments, chiffres

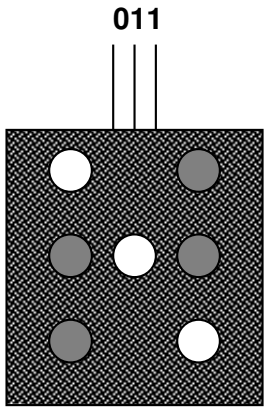

FIGURE 3.3 – Le dé électronique

Exercice Décodage 7 segments décimal Reprendre l'exercice de l'afficheur 7 segments, en supposant que les valeurs d'entrée vont toujours de 0000 à 1001.

Exercice Multiples de 3 Proposez un circuit qui prendra en entrée un nombre N entre 0 et 15 (codé en binaire évidemment) et dont la sortie indiquera si N est un multiple de 3.

Chapitre 4

Quelques circuits combinatoires

Dans ce chapitre nous étudions brièvement quelques circuits combinatoires qui interviennent dans la réalisation des composants logiques d'un ordinateur.

Nous présenterons l'étude de ces circuits sous une forme commune : la *spécification* énonce rapidement le rôle du circuit, la *fonction de tranfert* indique précisément (en général par des tables de vérité) la valeur des sorties pour toutes les entrées possibles, la *réalisation* montre un des circuits possibles.

4.1 Demi-additionneur

Spécification C'est un circuit (voir fig. 4.1) à deux entrées a, b et deux sorties r, s. Les entrées représentent les nombres 0 et 1, et rs est l'expression en binaire de leur somme. (s est la somme, r la retenue).

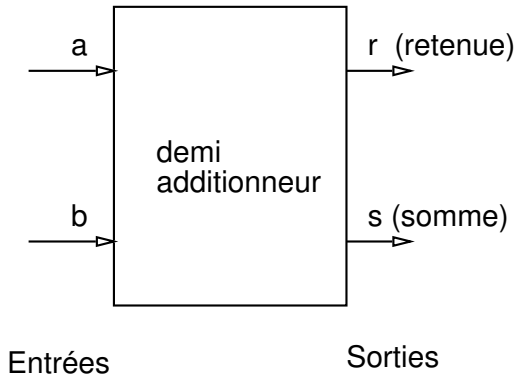

FIGURE 4.1 – Demi-additionneur

Fonction de transfert

a	b	r	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Équations :

$$\begin{aligned} r &= ab \\ s &= a\overline{b} + b\overline{a} \\ &= a \oplus b \end{aligned}$$

Réalisation : Le schéma de la figure 4.2 se déduit immédiatement des équations.

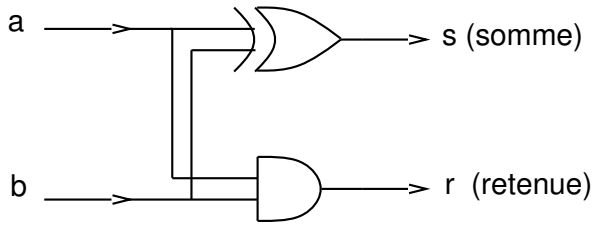

FIGURE 4.2 – Schéma d'un demi-additionneur

4.2 Additionneur élémentaire

Le demi-additionneur que nous venons de voir ne suffit pas pour réaliser des additions "chiffre par chiffre" parce qu'il produit des retenues, mais ne sait pas les utiliser.

Spécification L'*additionneur élémentaire* (Fig. 4.3) est un circuit à 3 entrées a, b, c et deux sorties r, s. Le nombre rs indique, en binaire, le nombre d'entrées qui sont à 1.

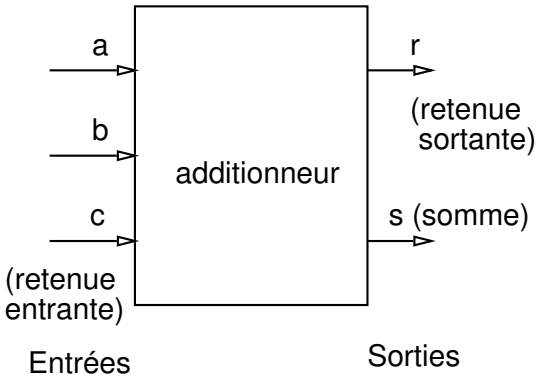

FIGURE 4.3 – Additionneur

Fonction de transfert

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

Equations :

$$\begin{aligned} r &= ab + bc + ac \\ s &= a \oplus b \oplus c \\ &= abc + \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}b\overline{c} \end{aligned}$$

Exercice Montrez qu'un additionneur peut être réalisé à partir de trois demi-additionneurs

Exercice Montrez qu'un additionneur peut être réalisé à partir de deux demi-additionneurs et une porte OU.

4.3 Circuit additionneur $2 \times n$ bits

En mettant côte à côte n additionneurs on peut réaliser un additionneur n bits.

Spécification Circuit à $2n+1$ entrées $(a_0 \dots a_{n-1}, b_0 \dots b_{n-1}, c)$ et $n+1$ sorties $(s_0 \dots s_{n-1}, r)$. Les entrées $a_{n-1}a_{n-2} \dots a_1a_0$ représentent un nombre A codé en binaire (et de même pour B sur les entrées b_j), et c est la retenue entrante. Les sorties $rs_{n-1}sn-2 \dots s_1s_0$ expriment la valeur de $A+B+c$.

Réalisation Une réalisation directe (par étude des tables de vérité et recherche d'expressions minimales) n'est envisageable que pour de très petites valeurs de n (si n = 3 il y a 7 entrées, donc des tables de vérité à 128 cases …).

On procède donc par décomposition du problème : il suffit de mettre en parallèle n additionneurs élémentaires identiques. La figure 4.4 montre un additionneur 2×4 bits.

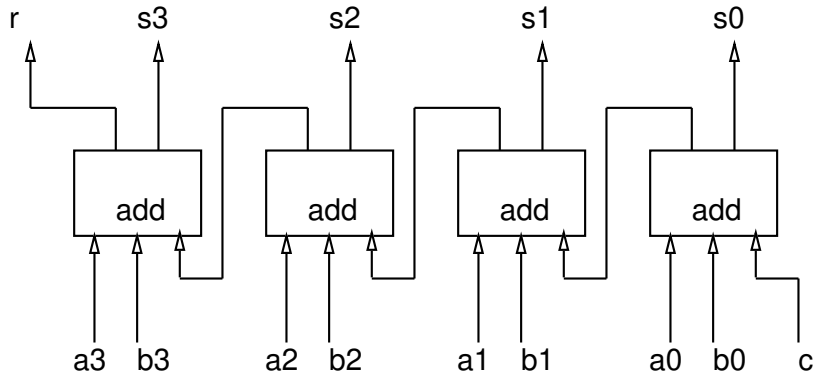

FIGURE 4.4 – Additionneur en tranches

Remarque Ces additionneurs "en tranches" peuvent eux-mêmes être juxtaposés pour former des additionneurs capables de traiter des nombres plus grands : il suffit de relier la retenue sortante de chaque circuit à la retenue entrante du suivant.

4.4 Décodeur

Ce circuit permet d'envoyer un signal à une sortie choisie.

4.4.1 Décodeur simple

Spécification Ce circuit possède n entrées $a_0 \dots a_{n-1}$ et 2^n sorties $s_0 \dots s_{2^n-1}$. Les entrées a_i codent un nombre A en binaire. La sortie s_A est mise à 1, les autres à 0.

Fonction de transfert Pour $n = 2$ on a la table :

a_1	a_0	s_0	s_1	s_2	s_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

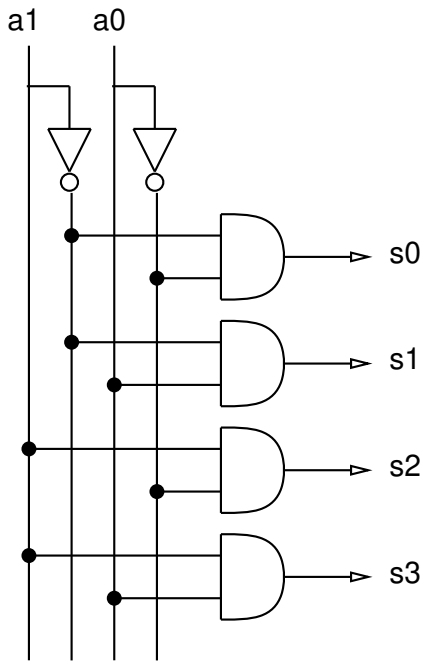

FIGURE 4.5 – Décodeur "2 vers 4"

Réalisation On voit facilement qu'à chaque sortie correspond un monôme complet sur $a_0, a_1 \ldots$, et réciproquement. Voir figure 4.5 pour le schéma d'un décodeur "2 vers 4".

Exercice Dessinez un décodeur "3 vers 8"

4.4.2 Décodeur avec validation

Ce décodeur possède une entrée supplémentaire V, qui sert à "activer" le circuit. Si cette entrée est à 1, le décodeur fonctionne comme précédemment. Si $V = 0$, toutes les sorties sont à 0.

On obtient (fig. 4.6) ce circuit par une simple modification du schéma précédent.

Remarque Les décodeurs avec validation peuvent être assemblés entre eux pour former des décodeurs plus gros. Le montage de la figure 4.7 montre un décodeur "maître" pilotant 4 décodeurs esclaves, le tout formant un décodeur "4 vers 16" avec validation.

Exercice Quelles doivent être les valeurs des entrées pour avoir $s_{13} = 1$?

4.5 Multiplexeur

Autre circuit très utile, le multiplexeur permet de sélectionner une entrée parmi plusieurs. C'est un circuit à une seule sortie S. Il prend comme entrées un nombre N sur k bits, et 2^k entrées e_i. La valeur de la N-ième entrée est copiée sur la sortie S.

La figure 4.8 montre un multiplexeur 4 voies. On remarquera que le multiplexeur contient un décodeur 2 vers 4.

Exercice Multiplexeur 16 voies Montrez comment réaliser un multiplexeur 16 voies par un montage maître-esclaves.

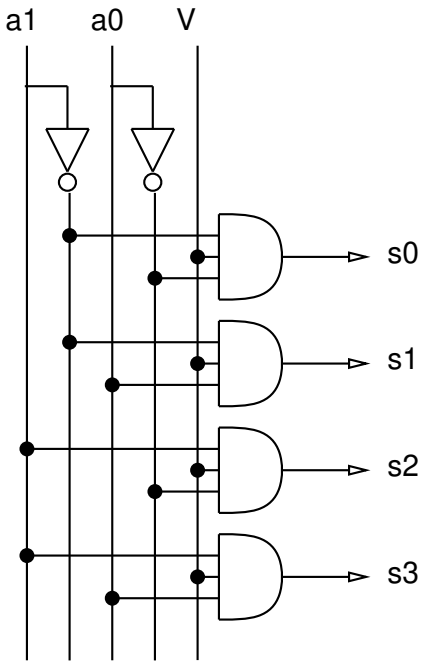

FIGURE 4.6 – Décodeur “2 vers 4” avec validation

4.6 Exercices

4.6.1 Test d’égalité

Concevoir un circuit qui prendra en entrée deux nombres binaires A et B de 4 chiffres, et indiquera si ces deux nombres sont égaux.

4.6.2 Comparateur

Concevoir un circuit qui prendra en entrée deux nombres binaires A et B de 4 chiffres, et indiquera si $A < B$, $A = B$ ou $A > B$. Essayez d’obtenir un circuit “cascadable” pour comparer des nombres plus grands.

4.6.3 Additionneur à retenue anticipée

Dans l’additionneur classique la retenue se propage de chiffre en chiffre, ce qui induit un délai de calcul proportionnel au nombre N de bits des nombres à traiter. Evaluez ce délai en fonction de N et du temps de traversée des portes ET, NON, OU.

La première retenue sortante r_0 vaut $a_0b_0 + a_0c_0 + b_0c_0$ (c_0 est la première retenue entrante). Pour la seconde retenue on a $r_1 = a_1b_1 + a_1c_1 + b_1c_1$. Comme $c_1 = r = 0$, on peut donc exprimer r_1 en fonction de a_0, a_1, b_0, b_1, c_0 (donnez la formule). C’est ce qu’on appelle le *pré-calcul* de r_1. Or cette formule s’exprime avec deux étages de portes (faites le schéma).

En suivant la même technique, donnez une formule pour r_2, puis r_3. Que dire du temps de traversée d’un tel additionneur N bits ? Quelle est la contrepartie de ce gain de temps ?

4.6.4 Compteur

Réaliser un circuit à 3 entrées e_1, e_2, e_3 et 2 sorties s_0, s_1. Le nombre binaire s_1s_0 indiquera le nombre d’entrées qui sont à 1.

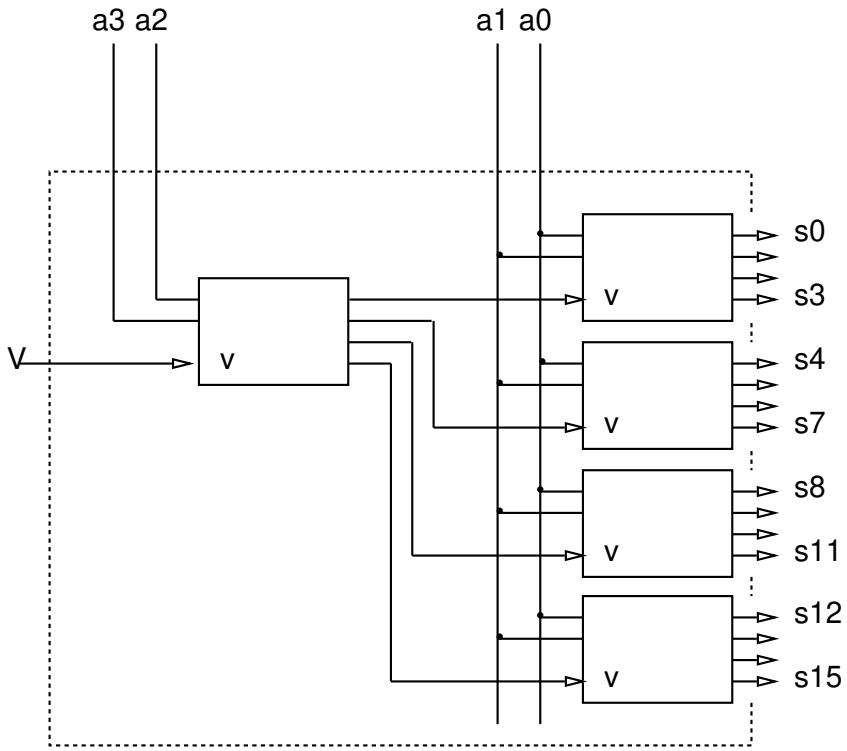

FIGURE 4.7 – Montage de décodeurs en maître-esclaves

4.6.5 Encodeur de priorités

Réaliser un circuit à 3 entrées e_1, e_2, e_3 et 2 sorties s_0, s_1. Le nombre binaire $s_1 s_0$ indiquera le plus grand indice des entrées qui sont à 1, ou 00 si toutes les entrées sont à 0.

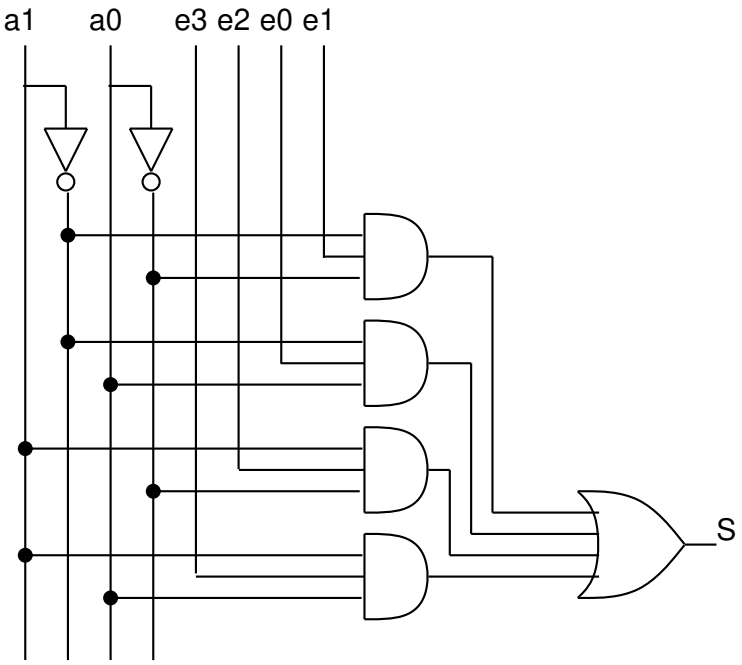

FIGURE 4.8 – Multiplexeur 4 voies

Chapitre 5

Circuits séquentiels

5.1 Définition

Un circuit séquentiel possède des entrées E et des sorties S, mais aussi un état interne Q qui sert à mémoriser de l'information. On peut résumer un tel circuit par deux fonctions. L'une (fonction de sortie f) indique la valeur des sorties en fonction des entrées et de l'état actuel :

$$S = f(E, Q)$$

L'autre (fonction de transition g) indique l'état que le circuit prendra à l'instant suivant :

$$Q' = g(E, Q)$$

5.2 Du bistable à la bascule RS

Une expérience simple consiste à monter (fig. 5.1) deux portes NON tête-bêche, la sortie de l'une servant d'entrée à l'autre. Soient Q_1 et Q_2 les deux sorties.

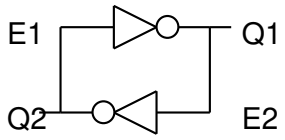

FIGURE 5.1 – Bistable à deux portes NON

On constate que ce circuit prend l'un des deux eétats suivants :
— état (0) : $Q_1 = 0$ et $Q_2 = 1$
— état (1) : $Q_1 = 1$ et $Q_2 = 0$
et qu'il y reste indéfiniment : on dit que le circuit est *stable*. (C'est un *bistable*, car il y a deux *positions de stabilité*).

En revanche si on boucle l'entrée avec la sortie d'une seule porte NON, (fig. 5.2) c'est un circuit *astable* qui oscille continuellement (et très rapidement) entre 0 et 1.

Reprenons notre bistable à 2 portes NON. Supposons qu'il soit dans l'état (0), c'est-à-dire $Q_1 = 0, Q_2 = 1$. Pour le faire basculer dans l'autre état, il faudrait forcer l'entrée E2 à 1. Mais pour l'instant l'entrée E2 n'est reliée qu'à Q_1. On intercale donc (fig. 5.3) une porte OU avec une entrée de commande S (Set).

Dans l'état (0) avec S=0, la situation est stable. L'arrivée d'une impulsion sur S provoque le passage de E2 à 1, puis Q_2 prend la valeur 0 et Q_1 passe à 0. Le bistable est donc dans la situation

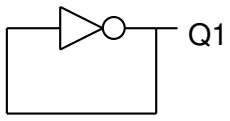

FIGURE 5.2 – Circuit astable

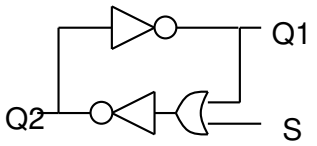

FIGURE 5.3 – Bistable avec commande S (set)

(1) qui est stable, et il y reste désormais quel que soit le signal d'entrée sur S. Ceci est résumé par le chronogramme de la figure5.4.

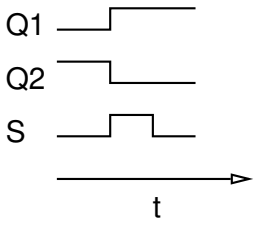

FIGURE 5.4 – Chronogramme

Pour permettre le retour à l'état (0) on peut ajouter -par symétrie- une autre entrée de commande R (reset) et une autre porte OU (fig. 5.5).

Le circuit résultant (que l'on réalise avec des portes NOR) est appelé *bascule RS*. Son fonctionnement normal se résume comme suit :

— 2 entrées R et S
— 2 sorties Q_1 et Q_2 (appelées aussi Q et $\overline{Q}$)
— R et S ne doivent pas être à 1 simultanément.
— une impulsion sur S met Q_1 à 1 et Q_2 à 0
— une impulsion sur R met Q_1 à 0 et Q_2 à 1

La figure 5.6 montre, sous forme de chronogramme, l'effet d'une séquence de "tops" envoyés successivement sur les entrées de commande d'une bascule RS.

Remarque Si R et S sont simultanément à 1 les deux sorties sont à 1. Si S et R repassent simultanément à 0 le résultat est alors imprévisible (dépend des vitesses de réaction des différentes portes).

5.3 Bascules dérivées

5.3.1 Bascule RS à portes NAND

En mettant des portes NAND au lieu des portes NOR, on obtient une bascule RS à signaux de commande inversés. On désigne alors les entrées de commande par $\overline{R}$ et $\overline{S}$ (voir figure 5.7).

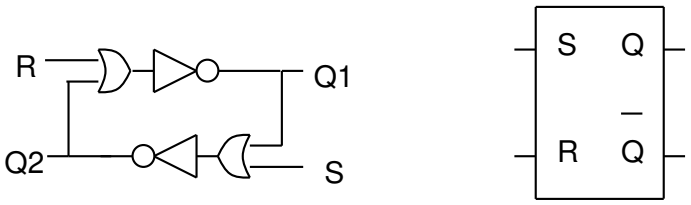

FIGURE 5.5 – Bascule RS : schéma et symbole

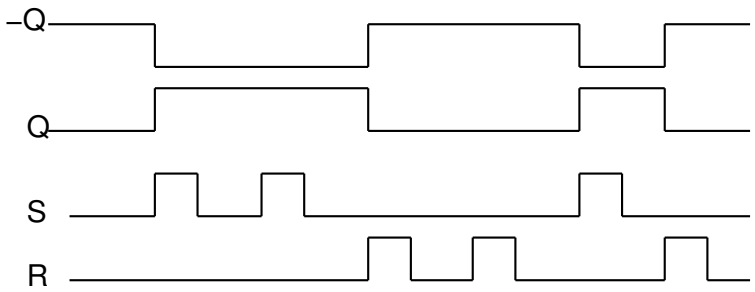

FIGURE 5.6 – Chronogramme d'une bascule RS

5.3.2 Bascule RS avec horloge

Les commandes R et S ne sont actives que lorsque l'entrée d'horloge est à 1. Il suffit (fig. 5.8) de valider les signaux R et S par l'entrée H au moyen de deux portes ET.

5.3.3 Bascule D

On la réalise à partir de la bascule RSH (figure 5.9) Lorsque son entrée H est à 1, elle mémorise la valeur de son entrée D. Quand H=0, elle reste insensible aux changements sur D (voir chronogramme de la figure 5.10)

La bascule D est utilisée pour la constitution des *mémoires*.

5.4 La conception de circuits séquentiels

Considérons par exemple une commande d'éclairage par bouton-poussoir. Il y a une entrée B (l'état du bouton : appuyé ou pas), une sortie L (la lampe allumée ou éteinte), mais il est clair

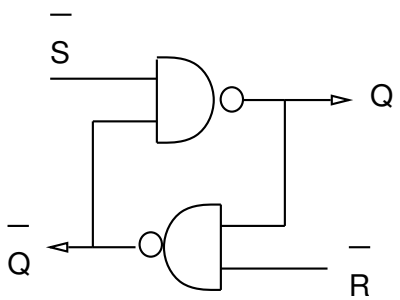

FIGURE 5.7 – Bascule RS à portes NAND

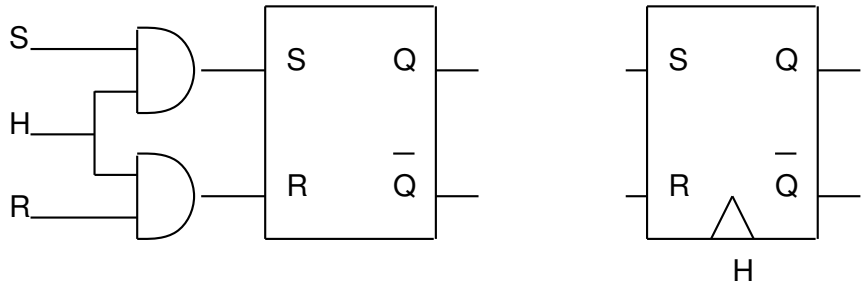

FIGURE 5.8 – Bascule RSH : schéma et symbole

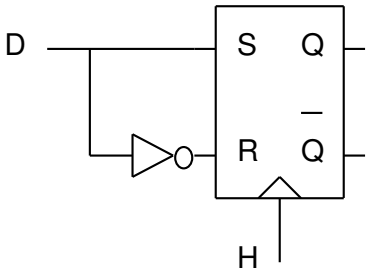

FIGURE 5.9 – Bascule D

que la sortie ne dépend pas seulement de l'entrée, mais aussi de ce qui s'est passé avant, que l'on représente par un état Q.

Ici on pourra décider que Q mémorise simplement l'état de la lampe. Si Q=0, la lampe est éteinte. Si Q=1, la lampe est allumée

La sortie est fonction de l'entrée et de l'état

$$L = f(B, Q)$$

Ici on prendra par exemple $L = Q$.

Une autre fonction indique comment l'état interne évolue au cours du temps. Pour simplifier le raisonnement on découpe le temps en "instants". L'état Q' "à l'instant suivant" dépend des entrées ainsi que de l'ancien état :

$$Q' = g(B, Q)$$

Dans l'exemple on dira que si B=0 l'état reste stable et si B=1 l'état s'inverse, d'où la table :

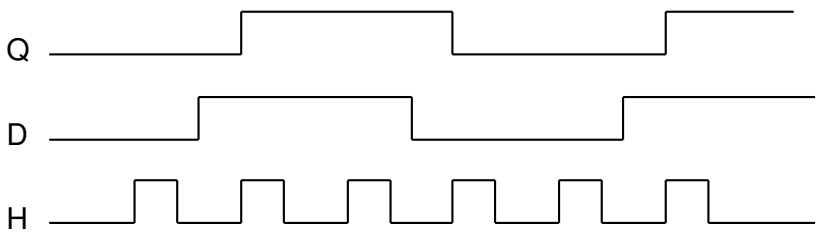

FIGURE 5.10 – Chronogramme d'une Bascule D

B	Q	$Q' = g(B, Q)$
0	0	0
0	1	1
1	0	1
1	1	0

A partir de là on pourrait en conclure qu'il suffit (fig. 5.11) d'une bascule D pour représenter l'état, qui serait activée par B et dont la sortie serait rebouclée sur l'entrée à travers un NOT. Mais attention ça ne marche pas ! En effet l'impulsion sur le bouton dure "plusieurs instants", ce

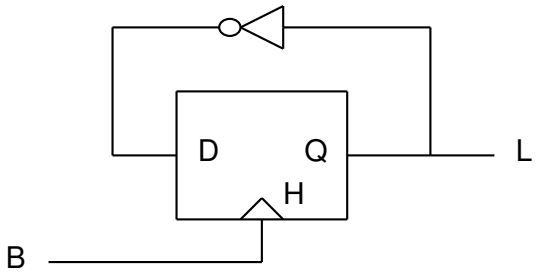

FIGURE 5.11 – Un montage naïf

qui fait clignoter (trop vite pour que l'oeil le perçoive) la sortie pendant l'appui du bouton : c'est un état instable. La vitesse élevée d'oscillation fait alors qu'à la fin de l'impulsion le résultat est imprévisible.

Pour s'en sortir on emploiera le montage de la figure 5.12, (appelé maître-esclave).

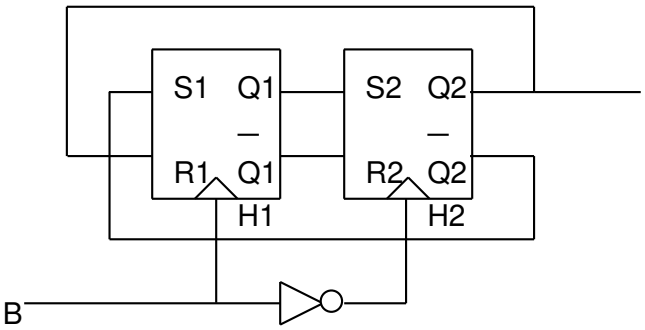

FIGURE 5.12 – Bascule RSH maître-esclave

la bascule 1 (maître) est validée par $H_1 = B$ et la 2 (esclave) par $H_2 = \overline{B}$, mais on utilise (et c'est là toute l'astuce) une porte NON dont le seuil de déclenchement est plus bas que la normale, ce qui fait qu'une impulsion sur B correspond à deux impulsions décalées sur H_1 et H_2, comme le montre le chronogramme de la figure 5.13.

Donc à l'arrivée d'une impulsion (front montant) la bascule esclave se verrouille en mémorisant l'état du maître, puis le maître change d'état. A la fin de l'impulsion (front descendant de l'impulsion sur B) le maître se verrouille, et l'esclave change d'état.

Exercice : vérifier le fonctionnement de ce circuit à l'aide d'un chronogramme (dessiner l'arrivée de 2 impulsions sur B).

Problème Bascule JK En utilisant le même principe (maître-esclave) concevoir une bascule JK. Comme son nom l'indique une bascule JK possède 3 entrées J,K et H et deux sorties Q et $\overline{Q}$. La bascule ne peut changer d'état que pendant une impulsion sur

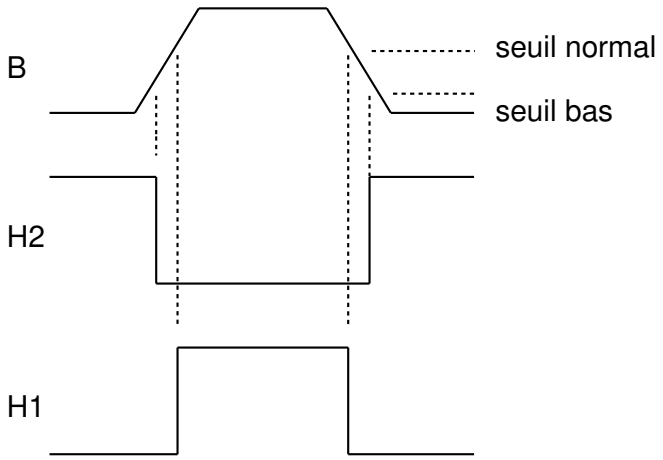

FIGURE 5.13 – Seuils de déclenchement

H. Si J=K=0 l'état reste inchangé. Si J=1 K=0 l'état devient 1. Si J=0 et K=1 l'état devient 0. Si J=K=1 l'état s'inverse.

Circuits Synchrones On appelle *circuit synchrone* un circuit séquentiel dont l'état interne ne change qu'à l'arrivée d'une impulsion sur une entrée spéciale appelée "Horloge". On suppose en général que les autres signaux ne varient pas pendant l'arrivée d'un top d'horloge.

L'intérêt de cette horloge est de donner une définition précise de "l'instant suivant". C'est l'horloge qui rythme les transitions d'état.

5.5 Application à la synthèse de compteurs

On appelle *compteur* un dispositif séquentiel qui passe périodiquement par une suite d'états. Par exemple le compteur binaire sur deux bits suivra la séquence $00, 01, 10, 11, 00, 01, 10, 11, 00$, etc. à chaque top d'horloge.

Nous allons voir comment réaliser, de manière systématique, n'importe quel compteur, d'abord avec des bascules D maître-esclave, puis avec des bascules JK qui permettent d'obtenir des circuits plus simples.

5.5.1 Réalisation à l'aide de bascules D

Exemple compteur binaire 2 bits

Les deux bits de l'état sont stockés dans deux bascules D.

— il n'y a pas de signal en entrée

— les sorties Q_1, Q_0 sont issues directement des bascules D

— pour réaliser la fonction de transition, il suffit de présenter sur les entrées D1 et D0 la valeur du prochain état que l'on calculera à partir de Q_1 et Q_0. Pour cela on construit la table :

ancien état		nouvel état	
Q_1	Q_0	$D1$	$D0$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

D'où on tire facilement :

$$\begin{aligned} D1 &= Q_1 \oplus Q_0 \\ D0 &= \overline{Q_0} \end{aligned}$$

5.5.2 Réalisation à l'aide de bascules JK

Le principe est légèrement différent : à partir de l'ancien état nous ne faisons plus calculer le nouvel état pour l'injecter dans les bascules, mais nous déterminons les commandes à envoyer aux bascules (sur les entrées J et K) pour que l'état change.

Avant de reprendre l'exemple du compteur 2 bits, quelques remarques sur les bascules JK. Regardons la table de transition :

ancien état	J	K	nouvel état
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

En regardant cette table d'une façon différente, on obtient

pour passer de	à	il faut avoir
0	0	$J = 0$
0	1	$J = 1$
1	0	$K = 1$
1	1	$K = 0$

Ceci va nous servir pour construire le compteur 2 bits, à l'aide de 2 bascules JK.

ancien état		nouvel état		commandes à envoyer	
Q_1	Q_0	Q'_1	Q'_0		
0	0	0	1	$J_1 = 0$	$J_0 = 1$
0	1	1	0	$J_1 = 1$	$K_0 = 1$
1	0	1	1	$K_1 = 0$	$J_0 = 1$
1	1	0	0	$K_1 = 1$	$K_0 = 1$

Il ne reste plus qu'à trouver des expressions convenables pour J_0, K_0, J_1 et K_1 en fonction de Q_1 et Q_0. En remplissant les tableaux de Karnaugh :

$J_0(Q_0, Q_1)$	0	1
0	1	.
1	1	1

$K_0(Q_0, Q_1)$	0	1
0	.	1
1	.	1

$J_1(Q_0, Q_1)$	0	1
0	0	1
1	.	.

$K_1(Q_0, Q_1)$	0	1
0	.	.
1	0	1

L'abondance de cas indéterminés nous permet d'obtenir des expressions très simples : $J_0 = 1$, $K_0 = 1$, $J_1 = Q_1$, $K_1 = Q_1$.

Exercice compteur modulo 3 La séquence à effectuer est $00, 01, 10, 00, 01, 10$, etc.

Exercice compteur binaire sur 3 bits $000, 001, 010, 011, ...111, 000, ...$

Exercice Compteur modulo 10 Séquence 0000,0001,....,1000,1001,0000...

Exercice Compteur/décompteur 3 bits Une entrée supplémentaire C indique s'il faut compter (C=1) ou décompter (C=0).

Exercice Compteurs rampants 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, ...

Exercice Compteurs de gray
— sur 1 bit : 0,1,0,1...
— sur 2 bits : 00,01,11,10,00,....
— sur 3 bits : 000,001,011,010,110,111,101,100,...

Bibliographie

[1] P. Cabanis, *Electronique Digitale*, Dunod (1985).

[2] Joël Risori et Lucien Ungaro, Cours d'architecture des ordinateurs, Tome 1 : conception des circuits digitaux, Eyrolles (1991).

[3] Donald D. Givone et Robert P. Roesser, *Microprocessors/Microcomputers : An Introduction*, McGraw-Hill (International Student Edition), 1980.