Logique combinatoire et séquentielle

Pierre Héroux

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Présentation du cours

1.1.1 Évolution de l'informatique

L'informatique est apparue aux alentours de 1950. Elle n'a cessé depuis cette date d'évoluer rapidement. La loi dite de Moore énonce que les performances matérielles sont doublées tous les 18 mois.

L'informatique fut d'abord utilisée à des fins de calcul par les grands centres de recherche. Puis, elle se répandit dans l'industrie pour être utilisée dans des processus d'automatisation, de contrôle et de commande. L'apparition de la micro-informatique et des bases de données introduisit l'informatique dans le secteur tertiaire avec pour tâches principales la communication et le partage d'information. La baisse des coûts permit à l'informatique d'investir les foyers, pour finalement devenir le moyen principal de diffusion des informations multi-média.

L'informatique consista d'abord en des systèmes centraux, puis vinrent les stations de travail, les micro-ordinateurs (PC) reliés en réseaux tout d'abord locaux puis raccordés sur la "toile" et enfin les terminaux mobiles.

Les évolutions techniques rapides entraînèrent des durées de vie de plus en plus courtes du matériel. Le marché informatique fût souvent instable avec un remise en cause fréquente de la position des constructeurs.

On vit d'abord apparaître des systèmes propriétaires issus de constructeurs en faible nombre. Puis les besoins de normalisation entraînèrent l'apparition de systèmes ouverts.

Du point de vue logiciel, le besoin de réutilisabilité fit naître les concepts du génie logiciel et des technologies objets. La généricité des composants objets est un contrepoids aux performances moindres, des dernières étant de plus masquées par les évolutions de performances du matériel.

Bon nombre des concepts utilisés à l'apparition de l'informatique perdurent actuellement.

Ce cours de science du numérique s'attachera à décrire les concepts qui sont à la base de l'architecture des ordinateurs également présents dans de nombreux appareils de la vie courante (électroménager, automobile, distributeurs de billets, smartphone, consoles de jeux, tablettes multimedia…).

1.1.2 Notion de bloc fonctionnel

Initialement calculateur numérique, un ordinateur est maintenant une machine de traitement de l'information.

Les traitements sont effectués par un *processeur* en réponse à des *instructions* . Les *instructions* et les *données* qu'elles traitent sont stockées dans la *mémoire*.

Instructions et *données* sont codées par un ensemble de valeurs logiques (vraie ou faux) représentées par deux niveaux de tension électrique.

La *mémoire* et le *processeur* sont reliés par un ensemble de câbles nommé *bus* permettant de véhiculer ces niveaux de tension et les valeurs logiques correspondantes.

L'utilisateur interagit avec l'ordinateur par l'intermédiaire de *dispositifs d'entrée/sortie*. Parmi les dispositifs d'entrée/sortie, on trouve :

— ceux qui n'ont vocation qu'à être des dispositifs de sorties (écran, imprimantes, haut-parleurs,...) ;
— ceux qui n'ont vocation qu'à être des dispositifs d'entrée (clavier, souris, scanner, webcam, microphone,...)
— ceux qui servent à la fois en entrée et en sortie avec en particuliers tous les dispositifs d'archivage des données (disque dur, lecteur/enregistreur de bande magnétique de disquette, lecteur/graveur de disque,...) mais également les interfaces permettant la communication par réseau.

Chaque bloc fonctionnel peut être décrit pas des unités fonctionnelles (architecture du processeur, architecture de la mémoire,...).

Chaque objet peut être décrit à plusieurs niveaux de description comme un ensemble de blocs interconnectés (niveau des transistors, niveau des portes logiques, niveau des fonctions logiques,...).

Dans ce cours, nous décrirons la notion de porte logique et construirons petit à petit des blocs fonctionnels de granularité supérieure.

Lors de la conception d'un ordinateur ou autre dispositif à base de processeur, le choix de l'architecture est un compromis entre performances d'une part et coût, facilité de mise en œuvre d'autre part.

L'imbrication entre ce qui est matériel et ce qui est logiciel se fait au niveau du jeu d'instructions du processeur.

1.2 Historique

1.2.1 Calcul mécanique

Le mot *calcul* vient du latin *calculus* signifiant *caillou* ce qui servait sous l'antiquité à dénombrer.

L'utilisation de la numérotation décimale est liée au nombre de doigts.

On a codé les nombres depuis l'antiquité. Tout d'abord, on utilisait des traits, chacun valant 1. Puis un symbole représentant 10 apparaît en Égypte en 3400 av. JC, puis d'autres symboles pour 100, 1000 et 10000.

La numérotation cunéiforme (base 60) fut utilisée à Babylone.

Le premier système binaire date de 3000 av. JC. Il est basé sur de Yin et le Yang (octogone à trigramme de Fou-Hi).

Les outils de calcul mécaniques dans l'antiquité étaient le boulier dans le monde chinois et les abaques dans le monde méditerranéen.

La notation positionnelle apparut ensuite. Elle consiste à donner une valeur différente pour le même symbole selon sa position dans le nombre. Cette notation n'a été rendue

possible qu'avec un symbole supplémentaire représentant le zéro. La notation positionnelle a été développée en Inde aux alentours de 300 av. JC. Puis, elle s'est répandue en Europe par l'intermédiaire des Arabes suite à la traduction en 820 des ouvrages du mathématicien Bagdad Al-Khuwarizmi, auteur d'un livre intitulé Al Jabr (ce qui donnera Algèbre). En effet, à cette époque, l'Islam s'étend de la Chine à l'Espagne.

La notation dite arabe ne remplace la notation romaine qu'au cours du XIV^{e}siècle. Cette notation est plus compacte et permet des calculs simplifiés.

Les premiers systèmes de calcul mécanique apparaissent au cours du XVII^{e}siècle. Ils sont construits sur des systèmes de roues dentées, comme les horloges et les automates.

En 1614, John Napier (Neper) invente les logarithmes. Grâce à eux, les multiplications se ramènent à des additions et les divisions à des soustractions. Il développe des machines à multiplier basées sur le déplacement de tiges, appelées bâtons ou os de Napier. Napier fut également un des premiers à utiliser le point décimal.

En 1622, William Oughtred développe une règle à calcul utilisant les logarithmes. Sa précision fut suffisante pour les calculs scientifiques jusqu'à la première moitié du XX^{e}siècle, mais inadaptée aux calculs de comptabilité.

En 1623, Wilhelm Schickard invente pour Kepler une "horloge calculante" pour le calcul d'éphéméride. Cette horloge utilise le système de roues dentées et introduit le concept de report de retenue. Disparue en 1624 lors de la guerre de Trente Ans en Allemagne, elle est reconstruite en 1960 d'après les plans originaux.

En 1642, le jeune Blaise Pascal, qui a alors 19 ans, construit pour son père commissaire aux impôts à Rouen une machine arithmétique (la Pascaline) permettant les additions et soustractions à 6 chiffres. Elle est composée d'une roue à 10 crans pour chaque chiffre qui lorsqu'elle fait un tour décale la roue suivante d'un cran. Les multiplications sont réalisées par additions successives.

En 1673, Gootfried Leibniz améliore la Pascaline en ajoutant un chariot et un tambour. Une manivelle permet l'automatisation des additions et soustractions successives. Leibniz invente également le système binaire et montre la simplicité de l'arithmétique binaire.

En 1728, Falcon, un mécanicien français, construit le premier métier à tisser commandé avec une planchette de bois percée de trous pour la description des motifs. Cette machine est par la suite perfectionnée par Joseph-Marie Jaquard en 1805. Cette version utilise des cartes cartonnées et articulées et permet la reproduction avec une qualité égale de motifs compliqués. Cette machine permet de se "dispenser" de 5 ouvriers par métier à tisser. Cette machine est en grande partie à l'origine de la Révolte des Canuts à Lyon, premier grand mouvement social qui a eu un retentissement international.

En 1820, Charles Xavier-Thomas de Colmar réalise, sur la base de la machine de Leibniz, le premier arithmomètre qui permet d'effectuer les additions, soustractions multiplications et divisions. D'abord à 3 chiffres pour les opérandes et 6 pour le totaliseur, elle évolua pour permettre le traitement des nombres à 30 chiffres. Cette machine fut utilisée jusqu'à la première guerre mondiale.

Aux alentours de 1820-1830, Charles Babbage, un mathématicien anglais, rapproche les machines à calculer et les machines de Jacquard pour effectuer des calculs complexes (séquences de plusieurs opérations arithmétiques). Un prototype, partiel à cause de difficulté techniques, de la machine à différences fut utilisé pour établir des tables de navigation ou de tir. Puis, la machine analytique permis d'effectuer des opérations arithmétiques en fonction d'instructions données par l'utilisateur. Cette machine contient une unité de traitement (le moulin), une unité de contrôle, une mémoire (magasin), une unité d'entrée permettant de lire des cartes perforées et une unité de

sortie.

Babbage fut aidé par Ada Lovelace (fille de Lord Byron, dont le prénom est le nom d'un langage informatique). Cette dernière définit le concept d'algorithme, du nom du mathématicien arabe Al-Khuwarizmi, comme l'enchaînement d'opérations successives pour la réalisation d'opérations.

En 1854, George Boole propose la formulation mathématique des propositions logiques qui appliquée au système binaire est à la base du fonctionnement des ordinateurs.

1.2.2 Calcul électro-mécanique

En 1890, Hermann Hollerith construit un calculateur de statistiques utilisé dans le recensement américain. Il développe la carte perforée et le système de codage des informations qui porte son nom. La détection d'un trou dans la carte se fait au moyen d'aiguilles qui ferment un circuit électrique trempant dans un godet de mercure. En 1896, Hollerith fonde la Tabulating Machine Company produisant des cartes et des machines électromécaniques. Cette société devient en 1924, l'International Business Machine Corporation, plus connue aujourd'hui sous l'acronyme IBM.

En 1936, Alan Turing conduit des travaux théoriques et énonce le principe d'une machine virtuelle pouvant réaliser tous les calculs mathématiques dont les instructions conditionnelles.

En 1938, Claude Shannon, à partir des travaux de Leibniz et de Boole, met en évidence l'analogie entre algèbre binaire et circuits électriques. Plus tard, il montre que tout calcul logique ou arithmétique peut être réalisé à partir de trois opérations logiques de base : et, ou, non. Il invente également le terme *bit*, contraction de *binary digit* (chiffre binaire).

En 1938, Konrad Zuse crée avec des moyens modestes le Z1, un ordinateur binaire programmable mécanique. En 1939, il le perfectionne (Z2) en remplaçant certaines pièces mécaniques par des relais électromécaniques. En 1941, le Z3 et le Z4 sont utilisés pour effectuer des calculs aéronautiques.

En 1939, John Atanasoff et Cliford Berry réalisent un additionneur 16 bits avec des tubes à vide.

En 1941, le Mark 1 ou ASCC (Automatic Sequence Control Calculator) est développé par IBM et Harvard sous la direction de Howard Aiken. Quelques chiffres :

— 5 tonnes ;
— $25m^2$;
— 25 kW ;
— 3000 relais ;
— 760 000 pièces mécaniques.

Les programmes sont lus sur une bande de papier et les données sur une autre. La machine est dotée de plusieurs lecteurs pour permettre les sauts conditionnels et les sous-programmes. En 1945, un insecte coince un relais provoquant un dysfonctionnement. C'est le premier *bug*.

1.2.3 L'ère de l'électronique

En 1941, Atanasoff et Berry réalisent le premier ordinateur binaire à lampes : l'ABC (Atanasoff Berry Computer). Il est considéré comme le premier ordinateur disposant d'une mémoire de 60 mots de 50 bits et d'une unité arithmétique et logique.

En 1945, mise en service d'ENIAC (Electronic Numerical Intergrator And Calculator) proposé en 1942 par Prosper Eckert et Maughly. Quelques chiffres :

— 19000 tubes ;
— 1500 relais ;
— 170 kW ;
— 30 tonnes ;
— $72m^2$;
— 330 multiplications par seconde (500 fois plus rapide que Mark 1).

Sa programmation se fait par des fiches à brancher sur un tableau et prend plusieurs jours.

En 1945, John Von Neumann, qui a travaillé comme consultant pour l'ENIAC, propose de coder le programme sous forme numérique et de l'enregistrer en mémoire, ce qui permet plus de souplesse et de rapidité.

Des querelles de paternité et de brevets entre Eckert, Maughly et Von Neumann retardent le projet EDVAC (Electronic Discret VAriable Computer), si bien que ce projet est devancé par Manchester Mark 1 en 1948 et par l'EDSAC (Electronic Delay Storage Automatic Computer) de Maurice Wilkes en 1949.

En 1948, William Shockley, John Bardeen et Walter Brattain, inventent le transistor bipolaire qui remplacera les lampes. Ils sont plus fiables, plus rapides. Leur utilisation donne naissance aux ordinateurs de seconde génération. L'encombrement et la consommation diminuent également.

Le premier ordinateur utilisant des transistors, le TRADIC, voit le jour en 1955.

IBM commercialise à cette époque le premier disque dur composé de 5 disques de 61cm de diamètre et d'une capacité de 5Mo. Il existe également des mémoires à torres de ferrite.

DEC commercialise en grande série (50000 exemplaires) le premier mini-ordinateur, le PDP-8.

Les années 1970 voient l'apparition des circuits intégrés donnant lieu aux ordinateurs de troisième génération.

En 1971, l'Intel 4004 est le premier microprocesseur à 4 bits et le premier circuit intégré comprenant une unité de calcul, de la mémoire et une gestion des entrées/sorties. Il comporte 2300 transistors. Un an plus tard, Intel commercialise le 8008.

En 1973, R2E, une entreprise française, commercialise le Micral N.

Depuis les années 80, on parle de machine de quatrième génération avec l'augmentation du niveau d'intégration des puces (densité et surface).

Les années 2000 sont celles de l'informatique embarquée (automobile, smartphone, électroménager, tablette...).

1.3 Premières notions d'architecture des ordinateurs

1.3.1 Les composants essentiels d'un ordinateur

Les composants essentiels d'un ordinateur sont :

— le processeur ;
— la mémoire ;
— le bus ;
— les dispositifs d'entrée/sortie.

Le processeur Le processeur est le composant opératif de l'ordinateur. C'est lui qui en réponse aux instructions d'un programme réalise des opérations arithmétiques et logiques.

De façon simplifiée, le processeur exécute le cycle suivant :

1. lecture (dans la mémoire) de l'instruction à exécuter ;
2. décodage de l'instruction ;
3. lecture (dans la mémoire) des données manipulées par l'instruction (opérandes) ;
4. exécution de l'instruction ;
5. sauvegarde (dans la mémoire) du résultat de l'instruction ;
6. passage à l'instruction suivante.

Le processeur est lui-même composé de :

— l'unité de commande qui contient un dispositif de décodage des instructions (*décodeur*) et un *séquenceur*. L'unité de commande contrôle l'exécution d'une instruction ;
— l'unité arithmétique et logique qui est la partie réellement opérative. Elle exécute des opérations simples arithmétiques, logiques, de comparaison...
— des registres qui sont des unités de mémoire internes au microprocesseur. D'accès très rapide, les registres permettent de stocker des résultats intermédiaires lors de l'exécution d'instruction, ainsi que des informations plus spécifique telles que l'adresse (l'emplacement) de la prochaine instruction, un registre d'état précisant la validité des résultats...
— des chemins de données permettant de véhiculer les informations.

Les différents composants du processeur sont cadencés au rythme d'une horloge. À chaque cycle se produisent des ouvertures ou des fermetures de portes pour déplacer, lire, écrire, comparer, additionner...

La fréquence de l'horloge s'exprime en hertz

Fréquence	Préfixe	Hz	Cycle	Préfixe	s
$1kHz$	kilo	10^3	$1ms$	mili	10^{-3}
$1MHz$	méga	10^6	$1\mu s$	micro	10^{-6}
$1GHz$	giga	10^9	$1ns$	nano	10^{-9}

TABLE 1.1 – Correspondance entre la fréquence et la durée du cycle

La mémoire La mémoire est un dispositif permettant de stocker des informations (données ou instructions d'un programme).

Une écriture en mémoire correspond à une mémorisation de l'information écrite. Une lecture destructive ou non correspond à une restitution de l'information mémorisée.

Une mémoire est caractérisée par :

— sa capacité, c'est-à-dire le volume d'informations qu'elle peut contenir au maximum ;
— son temps d'accès, c'est-à-dire le temps entre la demande de lecture ou d'écriture et la lecture ou l'écriture effective ;
— son temps de cycle, le temps minimum entre deux demandes de lecture/écriture consécutives. Le temps de cycle est supérieur ou égal au temps d'accès.

La cadence de transfert ou débit de la mémoire est le volume d'informations disponible par unité de temps.

Une mémoire est composée de cases ou cellules, chacune étant identifiée par un numéro, son adresse.

Une adresse exprimée sur n bits permet de référencer 2^n cases mémoire différentes au maximum.

La capacité de la mémoire s'exprime en puissances de 2 ou multiple de $2^{10} = 1024$ en bits, octets (byte) ou en nombre de mots mémoire (words).

Symbole	préfixe	approximation décimale	puissances de deux
1ko	kilo	10^3	$2^{10} = 1024$
1Mo	Méga	10^6	$2^{20} = 1024^2 = 1048576$
1Go	Giga	10^9	$2^{30} = 1024^3 = 1073741824$
1To	Tera	10^{12}	$2^{40} = 1024^4 = 1099511627776$

TABLE 1.2 – Capacité des mémoires

Les mémoires se caractérisent également par la façon dont on accède à l'information :

accès aléatoire, direct ou sélectif : le temps d'accès ne dépend pas de l'adresse (RAM).

accès séquentiel : Lecture jusqu'à l'enregistrement recherché (bande magnétique).

accès semi-séquentiel : disque dur (aléatoire pour la piste et séquentiel pour le secteur).

Les mémoires qui sont altérées lorsqu'il y a coupure de l'alimentation électrique sont dites volatiles.

En réalité, il existe une grande variété de dispositifs de mémoire. Le débit est conditionné par le temps d'accès, mais généralement les mémoires à temps d'accès rapide sont chères (elles sont en nombre restreint) et réciproquement.

les registres : Ils sont d'accès très rapides puisque faisant partie du microprocesseur. Ils sont également en nombre restreint.

La mémoire cache ou anté-mémoire : Il s'agit d'une petite mémoire incluse dans le processeur accessible en 1 ou 2 cycles.

La mémoire centrale : Il s'agit de la mémoire qui est accessible au processeur par l'intermédiaire du bus.

Les mémoires de masse : Il s'agit de mémoire accessible par l'intermédiaire de l'interface d'entrée/sortie. On trouve :
- Les disques magnétiques : disques durs et disquettes.
- Les bandes magnétiques essentiellement utilisées pour la sauvegarde en raison de leur grande capacité et de leur temps d'accès très long.
- les disques optiques numériques : CD, DVD.

La bus Le bus est le moyen de communication entre les différents composants de l'ordinateur. Il est composé de fils, chacun véhiculant une information binaire.

On trouve également sur le bus un ensemble de connecteurs (slot) permettant de recevoir une carte électronique dédiée à un dispositif d'entrée/sortie. À chaque carte

est allouée une zone d'adressage et est vue de ce fait comme de la mémoire du point de vue du processeur.

Le bus est séparé en plusieurs partie :

bus d'adresse : sert à véhiculer les adresses.

le bus de données : sert à véhiculer les données.

le bus de contrôle : véhicule des informations permettant de contrôler les échanges d'un composant à l'autre.

Entrée/sortie Les dispositifs d'entrée/sortie permettent à l'ordinateur de communiquer avec son environnement extérieur :

avec l'utilisateur : par l'intermédiaire de terminaux, écran, clavier, souris, imprimante, lecteur...

avec d'autres ordinateurs : par l'intermédiaire d'interface réseau ;

avec d'autres appareils : appareils de mesure, des actionneurs.

1.3.2 Architecture de Von Neumann

Dans l'architecture de Von Neuman, un *processeur* et une *mémoire* sont reliés par un *bus*.

La mémoire contient aussi bien les *instructions* à exécuter que les *données* sur lesquelles doivent être exécutées ces instructions ou leur résultat.

Cycle de lecture simplifié Le processeur affiche sur le bus l'adresse de l'information demandée (instruction ou donnée). En réponse, la mémoire met sur le bus l'information demandée. Le processeur lit, après un certain temps, l'information mise à disposition sur le bus.

Cycle d'écriture simplifié Le processeur écrit sur le bus l'information à mémoriser, ainsi que l'emplacement où la donnée doit être écrite en mémoire. La mémoire lit ces informations sur le bus et stocke l'information à l'emplacement spécifié.

La vitesse d'exécution est limitée par la vitesse de l'élément le plus lent.

Il y a trois possibilités pour gérer les entrées/sorties :

— L'interface d'entrée/sortie peut être directement liée au processeur qui utilise alors des instructions particulières.

— L'interface d'entrée/sortie peut être connectée au bus. Un fil du bus indique si le processeur s'adresse à la mémoire ou au dispositif d'entrée/sortie.

— L'interface d'entrée/sortie est vue par le processeur comme une partie de la mémoire. Un espace d'adressage distinct est alloué au dispositif.

1.3.3 Architecture de Harvard

Dans l'architecture de Harvard, la mémoire est séparée en deux zones dédiées, une aux instructions et une aux données. Chaque zone de la mémoire est accessible par un bus différent. Cela permet un accès simultané aux instructions et aux données entraînant une augmentation du débit.

Cette architecture a été abandonnée pendant une vingtaine d'année puis a de nouveau été utilisée en particulier sur les architectures à base de microprocesseur Motorola 68030.

L'interface d'entrée/sortie est connectée à la mémoire qui est accessible simultanément par moitié par le dispositif d'entrée/sortie et par le processeur.

Chapitre 2

Codage de l'information

2.1 Définitions

Soit I un ensemble d'informations.

Soit $A = \{a_1, a_2, \ldots, a_b\}$ un ensemble fini de b symboles. A est appelé *alphabet*. Les symboles a_i sont appelés les *caractères* de A.

Une suite ordonnée de symboles a_i (eventuellement avec répétition) est appelée *mot* de A.

On appelle *base* b, le cardinal (nombre d'éléments) de A.

Coder l'ensemble d'informations I à l'aide de l'alphabet A consiste à faire correspondre à chaque élément de I un mot construit sur l'alphabet A. Réciproquement, afin de pouvoir décoder I, à chaque mot de A doit correspondre au plus un élément de I.

Dans un codage de longueur n fixe, tous les mots ont la même longueur.
Il y a b possibilités pour choisir le premier caractère du mot.
Il y a b possibilités pour choisir le second caractère du mot.
...
Il y a b possibilités pour choisir le n^{e} caractère du mot.

En conséquence, un codage de longueur n à partir d'un alphabet de b symboles permet de construire b^n mots différents.

2.2 Représentation des nombres

2.2.1 Codage des entiers naturels

Notation positionnelle

La représentation d'un nombre est un codage d'une information quantitative. Un nombre entier naturel peut être représenté dans un base b.

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_b = \sum_{i=0}^{n-1} a_i b^i \tag{2.1}$$

Sur cette formule, les symboles a_i sont des chiffres dont la valeur est comprise entre 0 et $(b-1)$. Leur indice représente le rang du chiffre dans la représentation du nombre.

Les chiffres de gauche sont dits de poids fort puisqu'ils correspondent aux grandes puissances de b. Par opposition, les chiffres de droite sont dits de poids faible.

La formule (2.1) permet la conversion d'un nombre exprimé en base b en décimal.

Exemple : Soit à convertir $(524)_7$ en décimal.

$$\begin{aligned}(524)_7 &= 5 \times 7^2 + 2 \times 7^1 + 4 \times 7^0 \\ &= 5 \times 49 + 2 \times 7 + 4 \times 1 \\ &= 245 + 14 + 4 \\ &= (263)_{10}\end{aligned}$$

Réciproquement, un nombre exprimé en décimal peut se décomposer en base b selon le schéma de Horner.

$$(A)_{10} = \sum_{i=0}^{n} a_i b^i \tag{2.2}$$

$$= a_n \times b^n + a_{n-1} b^{n-1} + \cdots + a_1 \times b^1 + a_0 \times b^0 \tag{2.3}$$

$$= ((\cdots (a_n \times b + a_{n-1}) \times b + \cdots) \times b + a_1) \times b + a_0 \tag{2.4}$$

Les a_i sont les chiffres de la représentation en base b du nombre décimal A.

La formule 2.4 est utilisée pour convertir un nombre décimal en base b. On procède par divisions entières successives du nombre à convertir par b. Le reste de la i$^{\text{e}}$ division entière est le chiffre a_i de la représentation en base b.

Exemple : Soit à trouver la représentation en base 5 du nombre décimal 178.

$$\begin{aligned}178 &= 35 \times 5 + 3 \\ 35 &= 7 \times 5 + 0 \\ 7 &= 1 \times 5 + 2 \\ 1 &= 0 \times 5 + 1\end{aligned}$$

$$(178)_{10} = (1203)_5$$

Codage binaire naturel

Le codage binaire est souvent utilisé car c'est le codage utilisé de façon native par les ordinateurs. Dans le codage binaire naturel b vaut 2. On en déduit $A = \{0, 1\}$.

Un codage de longueur n permet de coder 2^n informations, les entiers de 0 à $2^n - 1$ en utilisant la formule

$$(a_{n-1} a_{n-2} \cdots a_1 a_0)_2 = \sum_{i=0}^{n-1} a_i 2^i$$

Par exemple, un codage de longueur 3 permet de coder les entiers de 0 à 7

Un mot binaire de longueur 8 est appelé *octet*.

entier représenté	a_2	a_1	a_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
7	1	1	1

TABLE 2.1 – Codage binaire naturel de longueur 3

Codage hexadécimal

Le codage hexadécimal est également souvent utilisé car la conversion vers et depuis le binaire est presque immédiate. Il permet égelement une représentation plus compacte des nombres.

Dans le codage hexadécimal, la base b vaut 16.

L'alphabet est composé des chiffres de 0 à 9 et des lettres de A à F valant respectivement de 10 à 15.

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

L'entier représenté est obtenu par la formule

$$(a_{n-1}a_{n-2}\cdots a_1a_0)_{16} = \sum_{i=0}^{n-1} a_i 16^i$$

entier représenté	code	entier représenté	code
0	0	16	10
1	1	17	11
2	2	18	12
3	3	...	...
4	4	25	19
5	5	26	1A
6	6	27	1B
7	7	...	...
8	8	31	1F
9	9	32	20
10	A	33	21
11	B	...	...
12	C	100	64
13	D	...	...
14	E	256	100
15	F	...	...

TABLE 2.2 – Codage hexadécimal

Si le codage est de longueur n les entiers représentés vont de 0 à $16^n - 1$.

La représentation hexadécimale est utilisée pour condenser la représentation binaire naturelle en exploitant

$$2^4 = 16^1$$

De ce fait, tout mot de 4 bits peut être représenté par un chiffre hexadécimal.

mot de 4 bits	chiffre hexadécimal	mot de 4 bits	chiffre hexadécimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

TABLE 2.3 – Correspondance entre mot de 4 bits et chiffre hexadécimal

La conversion binaire naturel vers hexadécimal se réalise simplement en regroupant par 4 les bits de la représentation binaire à partir des bits de poids faible et en les remplaçant par leur correspondant hexadécimaux du tableau précédent.

La conversion hexadécimal vers binaire naturel se fait de façon symétrique.

Exemple

$$\begin{aligned}(1101101001001)_2 &= (0001\,1011\,0100\,1001)_2 \\ &= (1B49)_{16}\end{aligned}$$

Décimal codé binaire (DCB)

Le codage DCB, pour décimal codé binaire, est un code binaire dédié à la représentation des nombres exprimé en décimal.

Les chiffres du codage décimal sont les chiffres de 0 à 9. Ces chiffres peuvent être représentés par un mots de 4 bits allant de 0000 à 1001. La représentation décimale d'un nombre est composée de plusieurs chiffres décimaux.

Le codage DCB consiste à remplacer chaque chiffre de la représentation décimale par le mot de 4 bits équivalent.

Ce codage est en particulier utilisé pour l'affichage des entiers décimaux sur des afficheurs 7 segments.

Les combinaisons allant de 1010 à 1111 ne sont pas utilisées dans le codage DCB.

Code de Gray

Le code de Gray est un codage construit de telle sorte que la représentation de deux entiers consécutifs ne diffère que par un seul bit.

Entier décimal	code DCB sur 8 bits
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
⋮	⋮
9	0000 1001
10	0001 0000
11	0001 0001
⋮	⋮
19	0000 1001
20	0010 0000
21	0010 0001
⋮	⋮
99	1001 1001

TABLE 2.4 – Exemples de codage DCB sur 8 bits

Entier décimal	code de Gray sur 4 bits
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1100
13	1101
14	1001
15	1000

TABLE 2.5 – Code Gray sur 4 bits

2.2.2 Représentation des entiers relatifs

Bit de signe

Dans ce codage sur n bits, on utilise 1 bit pour coder le signe de l'entier relatif et $n-1$ bits pour coder la valeur absolue.

Par convention, on utilise le bit de gauche pour coder le signe. Lorsque le bit de signe vaut 0, l'entier représenté est positif. Lorsque le bit de signe vaut 1, l'entier représenté est négatif.

Dans ce codage, la valeur absolue du nombre est codée en binaire naturel sur $n-1$ bits. Il est donc possible de coder les valeurs absolues de 0 à $2^{n-1}-1$.

En faisant précéder ces $n-1$ bits codant la valeur absolue par le bit codant le signe, on peut coder des nombres entiers relatifs de $-2^{n-1}+1$ à $+2^{n-1}-1$, soit au total 2^n-1 nombres. Or avec n bits, il est en théorie possible de coder 2^n nombres. Toute la capacité du codage n'est pas exploitée.

Le problème vient du fait qu'il existe deux représentations du zéro, une avec le bit de signe à 0 et l'autre avec le bit de signe à 1.

représentation binaire	entier relatif codé
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

TABLE 2.6 – Codage des entiers relatifs avec bit de signe (sur 4 bits)

Convention du complément à un

Dans la notation en complément à 1, sur un codage de n bits, comme dans la notation avec bit de signe, les $n-1$ bits de droite sont utilisés pour coder la valeur absolue du nombre et le bit le plus à gauche est utilisé pour coder le signe avec la même convention que précédemment (0 pour un nombre positif et 1 pour un nombre négatif). Cependant le codage de la valeur absolue est différent selon que le nombre est positif ou négatif.

Pour coder un nombre en utilisant la convention du complément à un, on commence par coder sa valeur absolue en binaire naturel (en fixant donc son bit de signe à 0). Si le nombre est négatif, on inverse tous les bits.

Pour décoder un nombre exprimé selon la convention du complément à un, on commence par en déterminer le signe en regardant le bit de signe qui vaut 1 pour les

nombres négatifs. Si le nombre est négatif, on inverse tous les bits pour obtenir la valeur absolue exprimée en binaire naturel.

représentation binaire	entier relatif codé
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

TABLE 2.7 – Codage des entiers relatifs selon la convention du complément à un (sur 4 bits)

Cette représentation présente l'avantage d'avoir une arithmétique binaire juste contrairement au codage par bit de signe. Cependant, on retrouve l'inconvénient de la double représentation du zéro. Toute la capacité du codage n'est pas exploitée.

Convention du complément à deux

Le plus souvent, nous utiliserons, pour coder les entiers relatifs, la convention du complément à deux. De même que précédemment, le premier bit du codage est utilisé pour indiquer le signe (0 pour les nombres positifs et 1 pour les nombres négatifs) et les $n - 1$ bits restant pour coder la valeur absolue. Le codage utilisé pour coder les nombres positifs est le codage binaire naturel.

Pour les nombres négatifs, on obtient la notation en complément à deux en ajoutant 1 à la représentation en complément à 1.

Exemple : Soit à coder -5 sur un mot de 4 bits en utilsiant la convention du complément à deux.

Nombre à coder	-5
Valeur absolue	5
Representation binaire naturelle de la valeur absolue	0101
Complément à 1	1010
Complément à 2	1011

Pour décoder la représentation binaire en complément à deux, on commence par regarder le bit de signe. Si celui si vaut 0, on obtient directement le nombre en le décodant selon la représentation binaire naturelle. Si le bit de signe vaut 1, l'entier est

négatif, il reste à obtenir sa valeur absolue. En donnant le complément à 2 du nombre, on obtient sa valeur absolue en représentation binaire naturelle.

Exemple : Soit à décoder sur le mot de 4 bits 1001 représenté selon la convention du complément à deux.

Nombre à décoder	1001	
Bit de signe	1	le nombre est négatif
Complément à 1	0110	
Complément à 2	0111	
Valeur absolue	7	
Nombre codé	-7	

Représentation binaire	Entier relatif
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

TABLE 2.8 – Codage des entiers relatifs selon la convention du complément à 2 (sur 4 bits)

On observe que la représentation sur 4 bits permet de coder les entiers de -8 à +7, soit au total 16 entiers. Ceci se généralise au cas des mots de n bits qui permettent de coder les entiers relatifs de -2^{n-1} à $2^{n-1}-1$, soit au total 2^n entiers différents. La capacité du codage est complètement exploitée. De plus, l'artihmétique binaire est juste lorsqu'on utilise ce codage.

Notation excédentaire

Dans la notation excédentaire, pour coder un entier relatif a, on ajoute à a une quantité constante c et on code le résultat en binaire naturel.

La constante est choisie de telle sorte que pour tous les entiers relatifs a, la quatité $a+c$ soit positive ou nulle.

Le plus souvent, un codage sur n bits est utilisé pour coder les nombres de -2^{n-1} à $2^{n-1}-1$ soit au total 2^n combinaisons. Dans ce cas, la constante c doit valoir 2^{n-1}.

Pour décoder un mot de n bits représentant un entier relatif exprimé selon la notation excédentaire, on le décode tout d'abord selon la représentation binaire naturelle, puis on soustrait 2^{n-1} à la quantité obtenue.

Représentation binaire	Entier relatif
0000	-8
0001	-7
0010	-6
0011	-5
0100	-4
0101	-3
0110	-2
0111	-1
1000	0
1001	1
1010	2
1011	3
1100	4
1101	5
1110	6
1111	7

TABLE 2.9 – Codage des entiers relatifs selon la notation excédentaire de 8 (sur 4 bits)

On remarque qu'avec ce codage les entiers positifs ou nuls ont un bit de poids fort valant 1, alors qu'il vaut 0 pour les entiers négatifs.

2.2.3 Représentation des nombres réels

Le codage des nombres réels s'avère un peu plus compliqué que celui des entiers, voire même impossible. En effet, quelque soit deux réels, aussi proches soient ils, il existe encore une infinité de réels entre les deux.

La solution consiste à adopter un codage permettant de coder un certain nombre de réels de façon exacte et à approximer les autres.

Plusieurs réprésentations sont possibles.

Représentation en vigule fixe

La représentation des réels en virgule fixe est similaire à la représentation des entiers, mais une partie du codage est dévolue à la partie entière du nombre (avec éventuellement utilisation du bit de signe) et l'autre à sa partie fractionnaire.

Exemple : Soit un codage sur 8 bits dont les trois bits de poids faible codent la partie fractionnaire.

Un réel exprimé selon ce codage peut s'écrire $(a_4a_3a_2a_1a_0a_{-1}a_{-2}a_{-3})_2$. Le réel codé vaut alors $\sum_{i=-3}^{4} a_i 2^i$.

Par exemple, sur ce codage le mot 00101011 vaut :

$$(00101011) \quad = \quad 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

$$\begin{aligned} &= 4 + 1 + 0,25 + 0,125 \\ &= 5,375 \end{aligned}$$

Réciproquement, pour trouver le codage d'un réel, on code d'une part en binaire naturel sa partie entière, puis sa partie fractionnaire.

Soit à coder 2,69 selon le codage par virgule fixe avec 2 bits pour la partie entière et 6 bits pour la partie fractionnaire. Sa partie entière exprimée sur 2 bits se code 10. Il reste à coder sa partie fractionnaire. Cela se fait en multipliant par 2 de façon successive la partie fractionnaire du nombre et en inscrivant dans le codage la partie entière du nombre obtenu.

poids (i)	nombre	$\times 2$	a_i
-1	0,69	1,38	1
-2	0,38	0,76	0
-3	0,76	1,52	1
-4	0,52	1,04	1
-5	0,04	0,08	0
-6	0,08	0,16	0

TABLE 2.10 – Exemple de codage d'une partie fractionnaire

0,69 se code donc 101100 sur 6 bits, le nombre 2,69 se code alors 10101100. Or en décodant se nombre on obtient

$$\begin{aligned} (10101100) &= 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 0 \times 2^{-6} \\ &= 2 + 0,5 + 0,125 + 0,0625 \\ &= 2,6875 \end{aligned}$$

Le codage a réalisé une approximation vers le réel le plus proche pouvant être représenté exactement.

Un des problèmes majeurs de ce codage est la précision. En effet, la différence entre deux réels consécutifs est fonction de la puissance de 2 associée au bit de poids faible. La précision est constante pour l'ensemble du domaine de représentation, si bien que l'erreur relative d'approximation est importante pour les réels proches de zéro et très faible pour les grands réels.

Par ailleurs, le domaine de représentation dépend de l'endroit où on place la virgule dans le codage. Pour un codage réalisé sur un nombre de bits constant, une extension du domaine de représentation se fait au détriment de la précision du codage.

Codage par couple (numérateur, dénominateur)

Une autre possibilité pour coder les réels consiste à découper le codage en deux parties codant chacune un entier en binaire naturel. Ces deux entiers sont respectivement les numérateur et dénominateur d'un nombre rationnel approximant le réel à coder.

Par définition, ce codage ne permet de coder qu'un sous-ensemble des réels, les rationnels. En réalité, il s'agit même du sous-ensemble des rationnels tel que les numérateur et dénominateur peuvent être codés en binaire naturel sur les bits qui leur sont dévolus.

En pratique, ce codage est très peu utilisé car il engendre une grande complexité dans la mise en œuvre des opérations arithmétiques.

Pour illustrer ce problème, on peut tenter de répondre à la question : " Quel est le meilleur couple d'entiers exprimés sur 8 bits qui approxime au mieux Π ?"

Représentation en virgule flottante

La représentation la plus utilisée pour coder les réels est la représentation en virgule flottante qui présente l'avantage d'avoir une précision qui dépend de l'ordre de grandeur du réel à coder.

Le principe de ce codage est le suivant. Soit X un réel à coder. Pour une base b donnée, il est possible de trouver une infinité de couples (M, E) tels que $X = M \times b^E$ avec E entier relatif.

Par exemple, dans le cas du codage binaire qui nous intéresse plus particulièrement :

$$\begin{aligned} 3,6 &= 3,6 \times 2^0 \\ &= 1,8 \times 2^1 \\ &= 0,9 \times 2^2 \\ &= 0,45^3 \\ &\vdots \\ &= 7,2 \times 2^{-1} \\ &= 14,4 \times 2^{-2} \\ &= 28,8 \times 2^{-3} \\ &\vdots \end{aligned}$$

M est appelée *mantisse* et E est appelé exposant.

Une partie du codage est consacrée au codage de la mantisse (un réel représenté en virgule fixe) et l'autre au codage de l'exposant (entier relatif). Il reste alors à choisir pour un réel donné à choisir quel couple (M, E) utiliser pour le codage.

Un choix pertinent est de choisir le codage qui approxime au mieux le réel, c'est-à-dire le couple (M, E) tel que la partie du codage dévolue à la mantisse soit exploitée au mieux.

Ceci est normalisée par la norme IEEE754.

Norme IEEE754

La norme IEEE754 définit la façon de coder les réels sur 32 bits en simple précision et sur 64 bits en double précision.

Dans cette norme, le couple (M, E) choisi est celui pour lequel la valeur absolue de M est comprise entre 1 inclus et 2 exclus.

Une telle mantisse a toujours une partie entière valant 1 en valeur absolue. De ce fait, et pour améliorer la précision du codage seule la partie fractionnaire m de la mantisse est codée. Comme la partie entière de M vaut 1, on en déduit que $M = 1+m$.

Par ailleurs, pour éviter la manipulation d'un bit de signe sur l'exposant, celui-ci est codé en notation excédentaire et on intègrera au codage e tel que $e = E + offset$.

De ce fait, un réel X peut s'écrire :

$$X = (-1)^s(1 + m) \times 2^{e-offset}$$

Les réels en simple précision sont codés sur 32 bits. Le premier bit code s. m est codé sur 23 bits dont les poids vont de -1 à -23. e est codé sur 8 bits et offset vaut 127.

Exemple : Soit le mot de 32 bits suivant à décoder selon la norme IEEE754 1100 0001 1010 1100 0

s : 1 (le nombre codé est négatif)

e : $(10000011)_2 = (131)_{10}$

m : 0101 1000 0000 0000 0000 000

$$\begin{aligned} m &= 2^{-2} + 2^{-4} + 2^{-5} \\ &= 0,25 + 0,0625 + 0,3125 \\ &= 0,34375 \end{aligned}$$

Le réel codé vaut donc

$$(-1)^1(1 + 0,34375) \times 2^{131-127} = -21,5$$

Soit le réel 51,7 à coder en simple précision selon la norme IEEE754.

$$\begin{aligned} 51,7 &= 1,615625 \times 2^5 \\ &= (-1)^0(1 + 0,615625) \times 2^{132-127} \end{aligned}$$

s vaut 0. Reste à coder 132 en binaire naturel sur 8 bits et 0,615625 en puissances négatives de 2 sur 23 bits.

$$\begin{aligned} 132 &= 66 \times 2 + 0 \\ 66 &= 33 \times 2 + 0 \\ 33 &= 16 \times 2 + 1 \\ 16 &= 8 \times 2 + 0 \\ 8 &= 4 \times 2 + 0 \\ 4 &= 2 \times 2 + 0 \\ 2 &= 1 \times 2 + 0 \\ 1 &= 0 \times 2 + 1 \Rightarrow (132)_{10} = (1000\,0100)_2 \end{aligned}$$

$$\begin{aligned} 0,615625 \times 2 &= 0,23125 + 1 \\ 0,23125 \times 2 &= 0,4625 + 0 \\ 0,4624 \times 2 &= 0,925 + 0 \\ 0,925 \times 2 &= 0,85 + 1 \\ 0,85 \times 2 &= 0,7 + 1 \\ 0,7 \times 2 &= 0,4 + 1 \\ 0,4 \times 2 &= 0,8 + 0 \\ 0,8 \times 2 &= 0,6 + 1 \\ 0,6 \times 2 &= 0,2 + 1 \\ 0,2 \times 2 &= 0,4 + 0 \end{aligned}$$

Donc, la représentation de 0,615625 en puissances négatives de 2 sur 23 bits est 1001 1101 1001 1001 1001 100.

Au final, la représentation en simple précision de 51,7 selon la norme IEEE754 est 0100 0010 0100 1110 1100 1100 1100 1100.

2.3 Codage des informations alphanumériques

Le codage des informations alphanumériques est réalisé par des tables de correspondance entre un mot binaire et le caractère équivalent.

Historiquement, la première table de correspondance utilisé fut la table ASCII (American Standard Code for Information Interchange). Un mot de 7 bits (128 combinaisons) permet de coder les caractères alphabétiques minuscules et majuscules, les chiffres, les signes de ponctuation et certains caractères spéciaux.

Poids fort (octal) Poids faible	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	TAB	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	¡	L	\	l	—
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	¿	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

TABLE 2.11 – Table ASCII

Cette table s’est vite révélée insuffisante pour coder bon nombre de caractères tels que les caractères accentués, entre autres. On a donc adjoint à la table ASCII 128 autres caractères pour obtenir le codage ASCII étendu dans lequel les caractères sont codés sur 8 bits.

Mais les besoins de tous n’étaient pas assouvis, d’où l’apparition dans les années 90 du codage unicode (compatible ASCII) qui code les caractères sur 16 bits et donc la possibilité de coder 65536 caractères. Ce codage permet de coder outre les caractères latins d’autres alphabets :

— Arménien ;
— Cyrillique ;
— Géorgien ;
— Grec ;
— Alphabet phonétique ;
— Ethiopien ;

— Arabe ;
— Hébreu ;
— Plusieurs alphabets indiens (Bengali, Devanagari,...) ;
— Des idéogrammes chinois, japonais, coréens, mongols, tibétains,...

Chapitre 3

Fonctions élémentaires de l'algèbre binaire

3.1 Fonctions binaires et algèbre de Boole

Un système binaire est un système qui ne peut exister que dans deux états.
Selon les domaines d'applications ces états peuvent prendre différentes dénominations.

numérique : 0 et 1 (bit : binary digit)

logique : — vrai et faux
— oui et non

électricité / électronique : — circuit ouvert ou fermé
— tension au niveau haut ou au niveau bas

3.2 Fonctions logiques de base

3.2.1 La porte OU

La fonction OU, également appelée addition logique, possède au minimum deux entrées. Sa sortie vaut 0 si toutes ses entrée valent 0. Par opposition, si au moins un de ses entrées vaut 1, sa sortie vaut 1.

L'opérateur OU est noté $+$.

$$y = a + b$$

a	b	$y = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 3.1 – Table de vérité de la fonction OU (deux entrées)

Il existe deux catégories de symboles permettant de décrire les logigrammes des fonctions logiques : Les symboles de la norme anglo-saxonne et les symboles de la norme européenne.

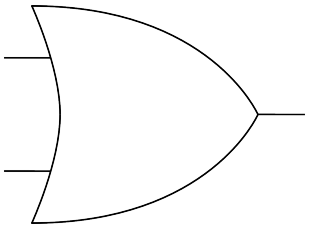

FIGURE 3.1 – Symbole de la porte OU (Norme anglo-saxonne)

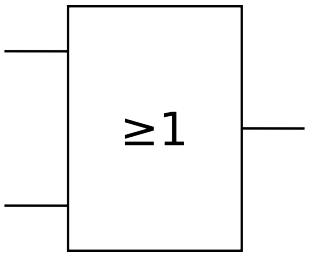

FIGURE 3.2 – Symbole de la porte OU (Norme européenne)

Propriétés de la fonction OU

La fonction OU est associative.

$$a + b + c = a + (b + c) = (a + b) + c$$

La fonction OU est commutative.

$$a + b = b + a$$

La fonction OU est idempotente.

$$a + a = a$$

L'élément neutre de la fonction OU est 0.

$$a + 0 = a$$

L'élément absorbant de la fonction OU est 1.

$$a + 1 = 1$$

3.2.2 La porte ET

La fonction ET, également appelée produit logique, possède aux moins deux entrées. Sa sortie vaut 1 si toutes ses entrées valent 1. Si au moins une des ses entrées vaut zéro, sa sortie vaut 0.

L'opérateur ET est noté . (point). Il peut éventuellement être omis dans la notation.

$$y = a.b = ab$$

a	b	$y = a.b$
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 3.2 – Table de vérité de la porte ET (deux entrées)

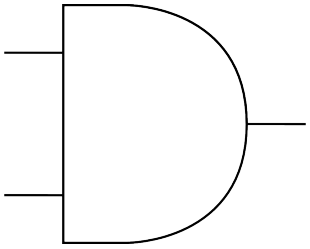

FIGURE 3.3 – Symbole de la porte ET (Norme anglo-saxonne)

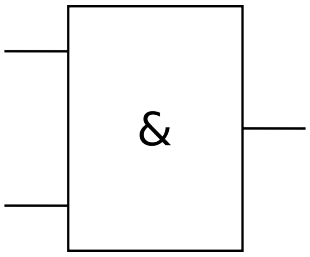

FIGURE 3.4 – Symbole de la porte ET (Norme européenne)

Propriétés de la fonction ET

La fonction ET est associative.

$$abc = a(bc) = (ab)c$$

La fonction ET est commutative.

$$ab = ba$$

La fonction ET est idempotente.

$$a.a = a$$

L'élément neutre de la fonction Et est 1.

$$a.1 = a$$

L'élément absorbant de la fonction ET est 0.

$$a.0 = 0$$

subsubsectionPropriétés

Priorité L'opérateur ET est prioritaire sur l'opérateur OU.

$$\begin{aligned} a.b + c &= (a.b) + c \\ &\neq a.(b + c) \end{aligned}$$

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

$$a.(b + c) = ab + ac$$

$$a + bc = (a + b).(a + c)$$

3.2.3 La porte NON

La fonction NON (inversion) ne possède qu'une seule entrée. Sa sortie vaut 1 quand son entrée vaut 0. Réciproquement, sa sortie vaut 0 si son entrée vaut 1.

L'opérateur NON est noté $\overline{a}$.

a	$\overline{a}$
0	1
1	0

TABLE 3.3 – Table de vérité de la porte NON

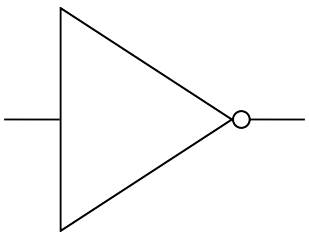

FIGURE 3.5 – Symbole de la porte NON (Norme anglo-saxonne)

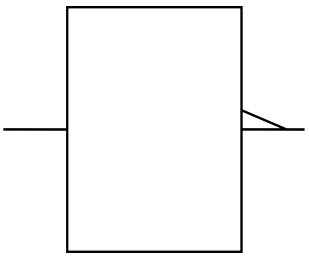

FIGURE 3.6 – Symbole de la porte NON (Norme européenne)

Propriétés de la fonction NON

$$\overline{\overline{a}} = a$$

$$a + \overline{a} = 1$$

$$a.\overline{a} = 0$$

$$a + \overline{a}b = a + b$$

Théorèmes de De Morgan

$$\overline{a.b.c\ldots} = \overline{a} + \overline{b} + \overline{c} + \ldots$$

$$\overline{a + b + c + \ldots} = \overline{a}.\overline{b}.\overline{c}\ldots$$

En appliquant les théorèmes de De Morgan, on peut obtenir une porte ET à partir d'une porte OU et de portes NON.

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

$$\begin{aligned} ab &= \overline{\overline{a}}.\overline{\overline{a}} \\ &= \overline{\overline{a} + \overline{b}} \end{aligned}$$

Mais on peut également obtenir une fonction OU avec une porte ET et des portes NON.

$$\begin{aligned} a + b &= \overline{\overline{a + b}} \\ &= \overline{\overline{a}.\overline{b}} \end{aligned}$$

$$\begin{aligned} a + b &= \overline{\overline{a}} + \overline{\overline{b}} \\ &= \overline{\overline{a}.\overline{b}} \end{aligned}$$

3.3 Autres portes logiques

Les portes ET, OU et NON sont les opérateurs de bases de l'algèbre de Boole. Elles permettent de construire toutes les fonctions logiques.

Cependant, quelques portes sont communément utilisées en raison de leur implémentation matérielle aisée et du fait que toutes les fonctions logiques peuvent également être représentées à l'aide de ses portes.

3.3.1 Porte NON ET (NAND)

La porte NON ET, plus communément appelée NAND est équivalent à la fonction ET dont la sortie est inversée.

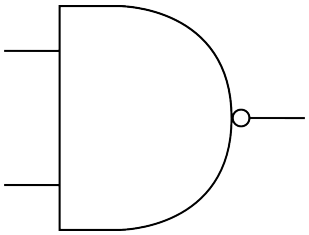

FIGURE 3.7 – Symbole de la porte NON ET (Norme anglo-saxonne)

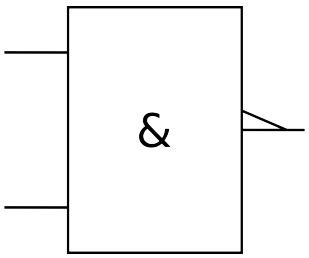

FIGURE 3.8 – Symbole de la porte NON ET (Norme européenne)

a	b	$y = \overline{a.b}$
0	0	1
0	1	1
1	0	1
1	1	0

TABLE 3.4 – Table de vérité de la porte NAND (2 entrées)

La fonction NAND permet de coder les fonctions ET, OU et NON.

$$\begin{aligned} ab &= \overline{\overline{ab}} \\ &= \overline{\overline{ab}.\overline{ab}} \end{aligned}$$

$$\begin{aligned} a+b &= \overline{\overline{a}+\overline{b}} \\ &= \overline{\overline{a}.\overline{b}} \\ &= \overline{\overline{a.a}.\overline{b.b}} \end{aligned}$$

$$\overline{a} = \overline{a.a}$$

Les opérateurs ET, OU et NON suffisent pour représenter toutes les fonctions logiques. Par ailleurs, ET, OU et NON peuvent être représentées à partir de portes NAND. Donc toutes les fonctions logiques peuvent être représentées à partir du seul opérateur NAND.

3.3.2 La porte NON OU (NOR)

La porte NON OU, plus communément appelée porte NOR, est équivalente à une porte OU dont la sortie est inversée.

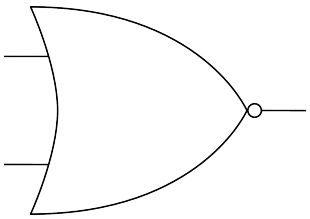

FIGURE 3.9 – Symbole de la porte NON OU (Norme anglo-saxonne)

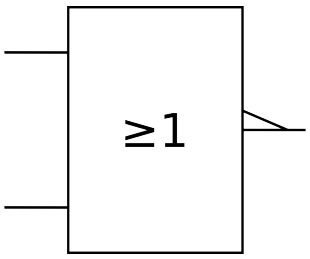

FIGURE 3.10 – Symbole de la porte NON OU (Norme européenne)

a	b	$y = \overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

TABLE 3.5 – Table de vérité de la portes NOR

La fonction NOR peut être utilisées pour représenter les fonctions ET, OU et NON.

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

$$\begin{aligned} a+b &= \overline{\overline{a+b}} \\ &= \overline{\overline{a+b} + \overline{a+b}} \end{aligned}$$

$$\overline{a} = \overline{a+a}$$

De la même façon que la porte NAND, la porte NOR peut être utilisée seule pour représenter toutes les fonctions logiques.

3.3.3 La porte OU EXCLUSIF (XOR)

La porte OU EXCLUSIF, aussi appelée XOR, possède exactement deux entrées. Sa sortie vaut 1 si une et une seule de ses deux entrées vaut 1. En d'autres termes la sortie vaux 1 si les deux entrées sont différentes. Par opposition, sa sortie vaut 0 si ses deux entrées sont égales.

La porte OU EXCLUSIF a pour symbole $\oplus$.

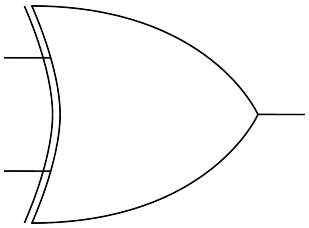

FIGURE 3.11 – Symbole de la porte OU EXCLUSIF (Norme anglo-saxonne)

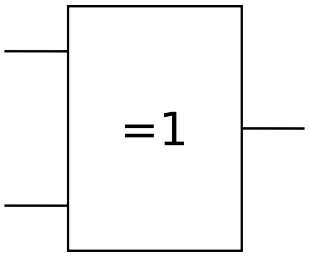

FIGURE 3.12 – Symbole de la porte OU EXCLUSIF (Norme européenne)

a	b	$y = a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.6 – Table de vérité de la porte OU EXCLUSIF

Écritures équivalentes

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

$$a \oplus b = a\bar{b} + \bar{a}b$$

$$a \oplus b = \overline{ab + \bar{a}\bar{b}}$$

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

Par extension, on définit le OU EXCLUSIF à plusieurs entrées qui vaut 1 lorsque le nombre d'entrées à 1 est impair et qui vaut zéro quand le nombre d'entrée à 1 est pair.

OU	$(a + b) + c = a + (b + c) = a + b + c$ $a + b = b + a$ $a + a = a$ $a + 0 = a$ $a + 1 = 1$	associativité commutativité idempotence élément neutre élément absorbant
ET	$(a \cdot b) \cdot c = a \cdot (b \cdot c) = a \cdot b \cdot c$ $a \cdot b = b \cdot a$ $a \cdot a = a$ $a \cdot 1 = a$ $a \cdot 0 = 0$	associativité commutativité idempotence élément neutre élément absorbant
Distributivité	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ $a + (b \cdot c) = (a + b) \cdot (a + c)$	
NON	$\bar{\bar{a}} = a$ $a + \bar{a} = 1$ $a \cdot \bar{a} = 0$	
Théorèmes de De Morgan	$\overline{a + b + c + \ldots} = \bar{a} \cdot \bar{b} \cdot \bar{c} \ldots$ $\overline{a \cdot b \cdot c \ldots} = \bar{a} + \bar{b} + \bar{c} + \ldots$	
OU EXCLUSIF	$a \oplus b = (a + b) \cdot \overline{a \cdot b}$ $a \oplus b = a\bar{b} + \bar{a}b$ $a \oplus b = \overline{ab + \bar{a}\bar{b}}$ $a \oplus b = (a + b) \cdot (\bar{a} + \bar{b})$	

TABLE 3.7 – Résumé des fonctions logiques élémentaires et de leurs propriétés

3.4 Écritures des fonctions logiques

Les fonctions logiques peuvent être décrite par leur *table de vérité*, des écritures algébriques ou des *logigrammes*. Toutes ces écritures sont équivalentes, si bien qu'à partir de l'une de ces écritures, il est possible d'obtenir toutes les autres.

3.4.1 Table de vérité

La table de vérité d'une fonction logique consiste à écrire de façon explicite la valeur prise par la fonction pour chaque combinaison possible de ses variables.

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

TABLE 3.8 – Exemple de table de vérité d'une fonction logique

3.4.2 Somme canonique de produits

L'écriture algébrique en somme canonique de produits peut être obtenue depuis la table de vérité de la fonction en repérant les combinaisons d'entrées pour lesquelles la fonction vaut 1. À chacune de ces combinaisons correspond un terme de la somme. Ce terme est le produit des variables, complementées lorqu'elles valent 0.

Exemple Soit la fonction logique f décrite par la table de vérité 3.9.

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

TABLE 3.9 – Exemple de table de vérité

On cherche les combinaisons des variables a, b et c pour lesquelles f vaut 1. À chacune de ces combinaisons correspond un terme de la somme.
À la combinaison 000 correspond le terme $\overline{a}\overline{b}\overline{c}$.
À la combinaison 011 correspond le terme $\overline{a}bc$.
À la combinaison 101 correspond le terme $a\overline{b}c$.

À la combinaison 110 correspond le terme $ab\overline{c}$.

Finalement, l'écriture algébrique de la fonction f en somme canonique de produits est :

$$f = \overline{a}\overline{b}\overline{c} + \overline{a}bc + a\overline{b}c + ab\overline{c}$$

3.4.3 Produit canonique de sommes

L'écriture canonique en produit de sommes peut être obtenue à partir de la table de vérité. À chaque combinaison des variables pour lesquelles la fonction vaut zéro correspond un facteur du produit. Ce facteur est composé de la somme des variables, complémentées si elles valent 1.

Exemple Reprenons la fonction f définie par la table de vérité 3.9.

On cherche les combinaisons des variables a, b et c pour lesquelles f vaut 0. À chacune de ces combinaisons correspond un facteur du produit.
À la combinaison 001 correspond le facteur $a + b + \overline{c}$.
À la combinaison 010 correspond le facteur $a + \overline{b} + c$.
À la combinaison 100 correspond le facteur $\overline{a} + b + c$.
À la combinaison 111 correspond le facteur $\overline{a} + \overline{b} + \overline{c}$.

Finalement, l'écriture algébrique canonique en produit de sommes de la fonction f est :

$$f = (a + b + \overline{c}).(a + \overline{b} + c).(\overline{a} + b + c).(\overline{a} + \overline{b} + \overline{c})$$

3.4.4 Simplification des écritures des fonctions logiques

Étant donné l'écriture d'une fonction logique (expression algébrique ou table de vérité), l'objectif de la simplification des écritures des fonctions logiques est de donner une écriture algébrique la plus simple possible, c'est-à-dire faisant intervenir un nombre de variable et d'opérateur logique le plus faible possible.

Simplification algébrique

L'expression algébrique d'une fonction logique peut être simplifiée en appliquant les formules données par le table 3.7.

Exemple Soit la fonction F définie par

$$F = \overline{a}bc + a\overline{b}c + ab\overline{c} + abc$$

On peut obtenir une écriture algébrique simplifiée peut être obtenue de la façon suivante :

$$\begin{aligned} F &= \overline{a}bc + a\overline{b}c + ab\overline{c} + abc \\ &= \overline{a}bc + a\overline{b}c + ab\overline{c} + abc + abc + abc \text{ par idempotence} \\ &= \overline{a}bc + abc + a\overline{b}c + abc + ab\overline{c} + abc \text{ par commutativité} \end{aligned}$$

$$
\begin{aligned}
&= (\overline{a}bc + abc) + (a\overline{b}c + abc) + (ab\overline{c} + abc) \text{ par associativité} \\
&= (\overline{a} + a)bc + (\overline{b} + b)ac + (\overline{c} + c)ab \text{ par factorisation} \\
&= bc + ac + ab
\end{aligned}
$$

Dans l'écriture initiale, il y avait 4 termes contenant chacun un produit entre 3 variables. Dans l'écriture finale, il y a 3 termes contenant produit de 2 variables.

Simplification par tableau de Karnaugh

Ce type de simplification s'utilise lorsque le nombre de variables est restreint (jusqu'à 4 voire 5 maximum). Cette simplification exploite la relation $a + \overline{a} = 1$.

Pour effectuer, cette simplification il faut disposer dans un tableau les valeurs prises par la fonction pour toutes les combinaisons des variables. Ces cases sont disposées de telle sorte que deux cases adjacentes du tableau ne diffèrent que par la valeur d'une seule variable (code de Gray).

Il s'agit ensuite d'effectuer des regroupements "rectangulaires" des cases de ce tableaux. Ces regroupements ne devront contenir que des 1 et l'union de tous ces regrouepement devra inclure tous les 1. Chacun de ces regroupements doit avoir une taille puissance de 2.

À chacun de ces regroupements correspondra un terme de la somme, expression algébrique simplifiée de la fonction logique.

Les regroupements doivent inclure tous les 1. Donc pour avoir le moins de termes possible, il faut inclure tous les 1 avec le moins de regroupements possible. Par ailleurs, plus le regroupement est gros, plus simple est le terme. En effet, le terme est constitué du produit des variables constantes pour chaque combinaison des variables du regroupement. Les variables du produit sont complémentées si elles sont constantes dans le regroupement.

Exemple Soit la fonction F définie par la table de vérité 3.10.

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

TABLE 3.10 – Exemple : Table de vérité de la fonction F

Le tableau correspondant est donné sur le tableau 3.11.

Il s'agit maintenant d'inclure tous les 1 grâce à un nombre minimum de regroupements "rectangulaires" dont la taille est une puissance de 2. Ceci est fait dans le tableau 3.12.

Dans le tableau 3.12, le regroupement bleu correspond à un ensemble de combinaison où b et c sont constantes et valent 1. Le terme correspondant sera bc.

De la même façon, le terme correspondant au regroupement rouge sera ab et celui correspondant au regroupement vert sera ac.

F :

y \ x,z	00	01	11	10
0	0	0	1	0
1	0	1	1	1

TABLE 3.11 – Tableau de Karnaugh de la fonction F

f :

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

TABLE 3.12 – Tableau de Karnaugh de la fonction F

Au final, l'écriture algébrique simplifiée par tableau de Karnaugh de la fonction F est

$$F = bc + ab + ac$$

Chapitre 4

Logique combinatoire

Dans ce chapitre nous présentons un certain nombre des fonctions logiques usuelles.

4.1 Addition binaire

Un nombre exprimé en binaire naturel sur n bits est compris entre 0 et $2^n - 1$. L'addition de 2 nombres représentables en binaire sur n bits est compris entre 0 et $2^{n+1} - 2$ et donc représentable sur $n + 1$ bits.

4.1.1 Demi-additionneur

Dans le cas particulier où n vaut 1, les entiers représentables sont 0 et 1. Le résultat de l'addition 1 bit est compris entre 0 et 2. Ce résultat est représentable sur 2 bits.

On peut en déduire le schéma du demi-additionneur prenant en entrée 2 variables a et b représentant les opérandes de l'addition et donnant en sortie un mot cs, représentation en binaire naturel du résultat, d'où la table de vérité 4.1 du demi-additionneur.

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

TABLE 4.1 – Table de vérité du demi-additionneur

De la table de vérité 4.1, on peut déduire les expressions algébriques de s et c.

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

4.1.2 Addition n bits

Lorsqu'on souhaite effectuer une addition binaire de 2 entiers exprimés sur n bits, chaque étage de l'addition produit un mot cs. Le bit c est alors une retenue qu'il convient d'intégrer dans l'addition de l'étage suivant.

La figure 4.1 donne le schéma de l'addition de deux mots de 4 bits $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$. Sur cette figure, à l'étage 0, l'addition des bits a_0 et b_0 produit un mot c_0s_0. Le bit c_0 est une retenue (carry) à prendre en compte à l'étage 1.

	c_2	c_1	c_0		
	a_3	a_2	a_1	a_0	Nombre A
+	b_3	b_2	b_1	b_0	Nombre B
c_3	s_3	s_2	s_1	s_0	Somme S = A + B
	c_3	c_2	c_1	c_0	Retenues

FIGURE 4.1 – Schéma de l'addition de deux mots de 4 bits

	0	1	0		
	0	0	1	0	2
+	0	0	1	1	3
0	0	1	0	1	5 = 2+3
	0	0	1	0	Retenues

FIGURE 4.2 – Exemple d'addition 4 bits

4.1.3 Additionneur complet

Le schéma de la figure 4.1 montre l'insuffisance du demi-additionneur puisqu'il faut en réalité pouvoir additionner 3 bits (les deux des opérandes et celui de la retenue entrante).

On en déduit le schéma de l'additionneur complet. L'additionneur complet est construit en effectuant l'addition des bits des opérandes par un demi-additionneur. Le résultat de cette addition est à son tour additionné à la retenue entrante grâce à un second additionneur.

Si une de ces deux additions produit une retenue, on aura une retenue sortante de l'additionneur complet. Ceci peut être obtenu par un opérateur OU prenant en entrée les retenues des deux demi-additionneurs.

L'additionneur complet peut également être obtenu directement. La table de vérité 4.2 donne la valeur de la somme et de la retenue produite en fonction des opérandes de l'addition et de la retenue entrante.

a	b	c_{in}	c_{out}	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

TABLE 4.2 – Table de vérité de l'additionneur complet

De cette table de vérité, on déduit les expressions algébriques de s et c_{out}.

$$\begin{aligned} s &= a \oplus b \oplus c_{in} \\ c_{out} &= ab + ac_{in} + bc_{in} \\ &= ab + c_{in}(a + b) \\ &= ab + c_{in}(a \oplus b) \end{aligned}$$

4.1.4 Addition parallèle

Pour effectuer l'addition n bits, on doit disposer de n additionneurs complets. Chacun est dévolu à un étage de l'addition. La retenue entrante du premier étage est fixée à 0.

L'étage i doit disposer en entrée de la retenue produite par l'étage $i-1$. De ce fait, le résultat est disponible après n fois le temps de transition d'un additionneur.

Il existe des techniques permettant d'anticiper les retenues et donc d'aténuer le temps de transmission des retenues.

4.1.5 Addition séquentielle

Dans une addition séquentielle, on ne dispose que d'un seul additionneur. Les entrées sont présentées bit après bit (train d'impulsions synchrones).

La retenue produite à un instant t doit être mémorisée de façon à être présentée à l'instant $t+1$.

L'addition parallèle est plus rapide que l'addition séquentielle mais nécessite en contrepartie plus de composants.

4.2 Soustraction

4.2.1 Demi-soustracteur

On souhaite effectuer la soutraction de deux bits a et b, pouvant chacun valoir 0 ou 1.

Le résultat de cette soustraction peut valoir 1, 0 ou -1. Ces trois combinaisons peuvent être codées sur 2 bits en utilisant la convention du complément à 2. Il s'ensuit la table de vérité 4.3.

a	b		s_1	s_0
0	0	0	0	0
0	1	-1	1	1
1	0	1	0	1
1	1	0	0	0

TABLE 4.3 – Table de vérité du demi-soustracteur

$$\begin{aligned} s_0 &= \overline{a}b + a\overline{b} = a \oplus b \\ s_1 &= \overline{a}b \end{aligned}$$

4.2.2 Additionneur-soustracteur

Avec un mot de n bits, il est possible de coder les entiers de 0 à $2^n - 1$.

Soit $A = (a_{n-1} \dots a_1 a_0)$ un mot de n bit. On note $\overline{A}$ le complément de A. $\overline{A}$ est obtenu en complémentant tous les bits de A.

$$\overline{A} = (\overline{a_{n-1}} \dots \overline{a_1}\,\overline{a_0})$$

L'addition $a_i + \overline{a_i}$ vaut 1 et ne produit pas de retenue, si bien que l'addition $A + \overline{A}$ donne un résultat sur n bits où tous les bits valent 1. Un mot de n bits où tous les bits valent 1 code $2^n - 1$.

$$\begin{aligned} & A + \overline{A} = 2^n - 1 \\ \Leftrightarrow \quad & -A = \overline{A} + 1 - 2^n \\ \Leftrightarrow \quad & -A = \overline{A} + 1 \text{ car } 2^n \text{ équivaut à } 0 \text{ sur } n \text{ bits} \end{aligned}$$

$\overline{A} + 1$ est le complément à 2 de A. On en déduit que la soustraction $B - A$ peut être effectuée en sommant B au complément à 2 de A.

$$B - A = B + \overline{A} + 1$$

Matériellement, la soustraction est effectuée en faisant l'addition $B + \overline{A}$ avec une retenue entrante valant 1 sur le premier étage de l'additionneur.

4.3 Autres fonctions usuelles de la logique combinatoire

4.3.1 Comparaison

Les fonctions de comparaison de bits sont données par la table de vérité 4.4.

a	b	$f_1 : (a < b)$	$f_2 : (a > b)$	$f_3 : (a = b)$
0	0	0	0	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

TABLE 4.4 – Table de vérité des fonctions de comparaison de bits

On en déduit les expressions algébriques des fonctions f_1, f_2 et f_3.

$$\begin{aligned} f_1 &= a\overline{b} \\ f_2 &= \overline{a}b \\ f_3 &= \overline{a}\,\overline{b} + ab \\ &= \overline{a \oplus b} \\ &= \overline{\overline{a}b + a\overline{b}} \\ &= \overline{f_1 + f_2} \end{aligned}$$

4.3.2 Contrôle de parité

La parité d'un mot binaire correspond à la parité du nombre de bits à 1 dans le mot, ou encore la parité de la somme des bits du mot.

parité paire : Le mot comporte un nombre pair de bits à 1

parité impaire : Le mot comporte un nombre impair de bits à 1

Par convention la parité paire est codée par 0 et la parité impaire est codée par 1. En utilisant cette convention, la parité d'un mot est obtenue par la fonction OU EXCLISIF appliquée à tous les bits du mot.

Le contrôle de parité est utilisé pour contrôler la bonne transmission d'un mot. En effet, la parité d'un mot auquel on adjoint sa parité vaut toujours zéro.

4.3.3 Codage – Décodage

Le codage consiste à faire correspondre un code à une information. Dans le cas d'un code binaire, le code associée à l'information est un mot binaire.

Un mot de n bit permet de coder 2^n informations différentes. Réciproquement, si on souhaite coder k informations différentes par un codage binaire, le mot doit comporter un nombre de bits valant l'entier immédiatement supérieur ou égal à $\log_2(k)$.

Encodeur

Un encodeur est un dispositif disposant de n sorties et de 2^n entrées telles qu'une et une seule de ces entrées doit être active à un instant donné.

En réponse à l'entrée active, l'encodeur présente en sortie un mot de n bits. Ce mot désigne de façon biunivoque l'entrée active.

Décodeur

Un décodeur est le dispositif opposé qui dispose de n entrées et de 2^n sorties dont une seule est active à la fois.

En réponse à la combinaison présentée en entrée, le décodeur active la sortie correspondante.

Exemple : codage décimal codé binaire

Le tableau donne le codage décimal codé binaire.

Le encodage DCB consiste à obtenir le code DCB lorsqu'une des entrées correspondant aux chiffres de 0 à 9 est active.

Le décodage consiste à obtenir une sortie active sur présentation du code DCB.

Ces deux opérations utilisent la table 4.6

On en déduit les équations des sorties de l'encodeur.

$$\begin{aligned} s_0 &= e_1 + e_3 + e_5 + e_7 + e_9 \\ s_1 &= e_2 + e_3 + e_6 + e_7 \\ s_2 &= e_4 + e_5 + e_6 + e_7 \\ s_3 &= e_8 + e_9 \end{aligned}$$

Décimal	DCB			
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
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

TABLE 4.5 – Table du codage décimal codé binaire

e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	s_3	s_2	s_1	s_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

TABLE 4.6 – Codage – décodage DCB

4.3.4 Multiplexage

Le multiplexage est l'action de faire passer sur une seule ligne l'information provenant de plusieurs sources et à destination de plusieurs récepteurs.

Un démultiplexeur est un dispositif disposant d'une entrée, de n fils d'adresse et de 2^n sorties. Sa fonction est de positionner la valeur présente en entrée sur la sortie désignée par la combinaison présentée sur les fils d'adresse.

Un multiplexeur réalise l'opération inverse. Ce dispositif dispose de n fils d'adresse, de 2^n entrées et d'une sortie. Sa fonction est de sélectionner l'entrée identifiée par la combinaison présentée sur les fils d'adresse et de recopier sa valeur sur la sortie.

Chapitre 5

Unité arithmétique et logique

5.1 Présentation

L'unité arithmétique et logique (UAL ou ALU) est un des composants principaux du microprocesseur, lui-même étant le composant opératif d'un ordinateur.

Le rôle de l'UAL est de réaliser des opérations logiques et arithmétiques de base.

Parmi les opérations qu'elle effectue on trouve en particulier :

— Des opérations logiques de base réalisées bit à bit sur un ou deux mots (ET, OU, NON, XOR...) ;

— Des décalages [1], des rotations sur des mots ;

— Des additions, soustractions, complémentations à 1 et à 2 ;

— Des comparaisons.

L'unité arithmétique et logique dispose de deux entrées $A = a_{n-1} \dots a_0$ et $B = b_{n-1} \dots b_0$ exprimées sur n bits (les opérandes) et d'une sortie $F = f_{n-1} \dots f_0$ également exprimée sur n bits.

Un mot de commande permet de sélectionner parmi les opérations permises par l'UAL laquelle doit être réalisée.

Enfin, un mot donne l'état de l'UAL après exécution de l'opération. Son rôle sera détaillé plus loin.

La figure 5.1 représente le symbole communément utilisé pour désigner une unité arithmétique et logique.

5.2 Constitution d'une UAL

Parmi les bits de commande, les bits M sélectionne le mode arithmétique ($M = 1$) ou le mode logique ($M = 0$).

Le bit C_0 du mot de commande joue le rôle de retenue entrante. Il est en particulier à 1 pour la soustraction (+1 de la complémentation à 2).

Les autres bits de commande $S_{n-1} \dots S_0$ permettent de choisir l'opération à réaliser.

Une UAL n bits est constituée de n tranches mises en cascade. La tranche d'indice i dispose de :

6 entrées : a_i et b_i les opérandes, c_i la retenue entrante, M le sélecteur de mode et S_1 et S_0 les sélecteurs d'opération.

1. Les décalages sont en particulier utilisés pour des multiplications ou des divisions par 2

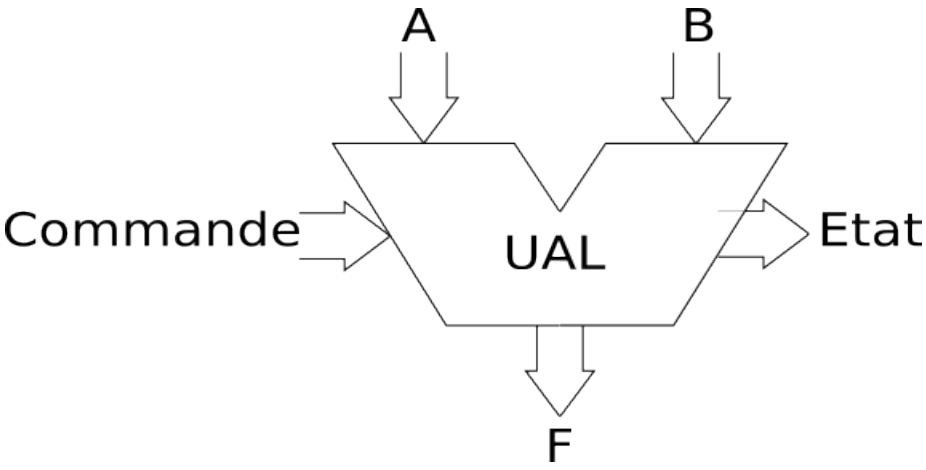

FIGURE 5.1 – Unité arithmétique et logique

2 sorties : f_i et c_{i+1} la retenue sortante.

La figure 5.2 représente l'exemple d'une tranche d'UAL.

Sur cette figure, un premier étage permet d'effectuer des opérations logiques (ET, OU, NON) ou arithmétiques (Addition). Par l'intermédiaire du décodeur prenant en entrée les signaux S_1 et S_0, il est possible de choisir laquelle de ces opérations aura son résultat répercuté sur la sortie.

Le tableau 5.1 donne en fonction de S_1 et S_0 les fonctions réalisée par UAL simple 1 bit.

S_1	S_0	F	C_{out}
0	0	$A.B$	0
0	1	$A+B$	0
1	0	$\overline{A}$	0
1	1	$A \oplus B \oplus C_{in}$	$A.B + C_{in}(A \oplus B)$

TABLE 5.1 – Fonctions de l'UAL simple 1 bit

En mettant en cascade n circuits de ce type, on obtient une unité arithmétque et logique permettant de réaliser pour A et B exprimés sur n bits :

$S_1S_0 = 00$ **:** A AND B bit à bit

$S_1S_0 = 01$ **:** A OR B bit à bit

$S_1S_0 = 00$ **:** NON A bit à bit (complément à 1 de A)

$S_1S_0 = 00$ **:** $A + B$ (addition)

Avec 2 bits de commande, il est possible de choisir une opération à effectuer par 4 possibles. En augmentant le nombre de bits de commande, on augmente le nombre d'opérations qu'il est possible de commander.

Le tableau donne les fonctions réalisée par l'UAL TTL 74LS181 qui est une UAL 4 bits disposant de 6 bits de commandes (C_0, M, S_3, S_2, S_1 et S_0).

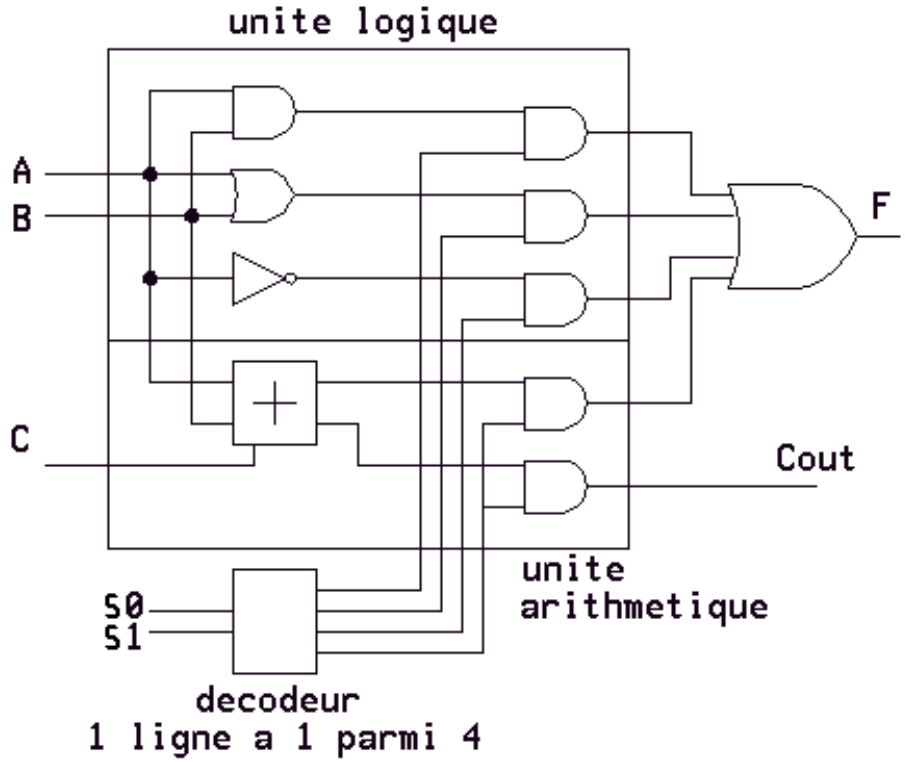

FIGURE 5.2 – Schéma d'une UAL 1 bit simple

S_3	S_2	S_1	S_0	$M = 1, C_0 = 1$ fonctions logiques	$M = 0, C_0 = 1$ fonctions arithmètiques
0	0	0	0	$\overline{A}$	A
0	0	0	1	$\overline{A + B}$	$A + B$
0	0	1	0	$\overline{A}B$	$A + \overline{B}$
0	0	1	1	0 logique	moins 1
0	1	0	0	$\overline{AB}$	A plus $A\overline{B}$
0	1	0	1	$\overline{B}$	$(A + B)$ plus $A\overline{B}$
0	1	1	0	$A \oplus B$	A moins B moins 1
0	1	1	1	$A\overline{B}$	$A\overline{B}$ moins 1
1	0	0	0	$\overline{A} + B$	A plus AB
1	0	0	1	$\overline{A \oplus B}$	A plus B
1	0	1	0	B	$(A + \overline{B})$ plus AB
1	0	1	1	AB	AB moins 1
1	1	0	0	1 logique	A plus A
1	1	0	1	$A + \overline{B}$	$(A + B)$ plus A
1	1	1	0	$A + B$	$(A + \overline{B})$ plus A
1	1	1	1	A	A moins 1
S_3	S_2	S_1	S_0	$M = 1, C_0 = 0$ fonctions logiques	$M = 0, C_0 = 0$ fonctions arithmétiques
0	0	0	0	$\overline{A}$	A moins 1
0	0	0	1	$\overline{AB}$	AB moins 1
0	0	1	0	$\overline{A} + B$	$A\overline{B}$ moins 1
0	0	1	1	1 logique	moins 1
0	1	0	0	$\overline{A + B}$	A plus $(A + \overline{B})$
0	1	0	1	$\overline{B}$	(AB) plus $(A + \overline{B})$
0	1	1	0	$\overline{A \oplus B}$	A moins B moins 1
0	1	1	1	$A + \overline{B}$	$A + \overline{B}$
1	0	0	0	$\overline{A}B$	A plus $(A + B)$
1	0	0	1	$A \oplus B$	A plus B
1	0	1	0	B	$(A\overline{B})$ plus $(A + B)$
1	0	1	1	$A + B$	$A + B$
1	1	0	0	0 logique	A plus A
1	1	0	1	$A\overline{B}$	(AB) plus A
1	1	1	0	AB	$(A\overline{B})$ plus A
1	1	1	1	A	A

TABLE 5.2 – Fonctions réalisées par l'UAL TTL 74LS181 en fonction des bits de commande

5.3 Mot d'état

Outre les bits $f_{n-1} \dots f_0$, l'UAL dispose de sorties complémentaires qui caractérise le résultat F de l'opération exécutée. Ces sorties sont rassemblées au sein d'un mot d'état.

Parmi ces bits on trouve en particulier les bits :

Z **(Zéro) :** Ce bit est à 1 quand tous les bits f_i sont à 0. Ce bit est en particulier utile pour tester l'égalité de deux nombres. Il suffit de demander à l'UAL de les soustraire et de tester le bit Z.

N **(Negative) :** Ce bit est à 1 quand l'interprétation signée de F est négative. Le bit N vaut le bit f_{n-1}. Ce bit est en particulier utile pour trouver le sens de l'inégalité entre deux nombres. Il suffit de les soustraire et de tester le bit N.

C **(Carry) :** Ce bit indique, lorsqu'il vaut 1, que l'interprétation non signée du résultat est erronée du fait d'un dépacement de capacité. $C = C_n \oplus C_0$ avec C_0 retenue entrant du premier étage de l'UAL et C_n retenue sortante du dernier étage.

V **(oVerflow) :** Ce bit indique, lorsqu'il vaut 1, que l'interprétation signée du résultat est erronée du fait d'un dépassement de capacité. $V = C_n \oplus C_{n-1}$ avec C_n et C_{n-1} retenue sortante des deux derniers étages.

Selon les circuits, le mot d'état de l'UAL est complétée par d'autres bits, par exemple, un bit indiquant la parité du résultat.

A	0111	1110	0010	1001
non signé	7	14	2	9
signé	+7	-2	+2	-7
B	0011	0101	1000	0110
non signé	3	5	8	6
signé	+3	+5	-8	+6
$A + B$	1010	0011	1010	1111
non signé	10	3	10	15
signé	-6	+3	-6	-1
C	0	1	0	0
V	1	0	0	0
$\overline{B}$	1100	1010	0111	1001
$A + \overline{B} + 1$	0100	1001	1010	0100
non signé	4	9	10	4
signé	+4	-7	-6	+4
C	0	0	1	0
V	0	0	1	1

TABLE 5.3 – Illustration des bits C et V

Le tableau 5.3 donne une illustration de l'utilisation des bits C et V pour l'addition et la soustraction de mots de 4 bits. Dans ce tableau, les deux premières lignes donnent les interprétations signées et non signées de deux opérandes exprimées sur 4 bits. La troisième ligne donne le résultat sur 4 bits de l'addition, ses interprétations signées et non signées et la valeur des bits C et V. La quatrième ligne donne le résultat sur 4 bits de la soustraction $(A + \overline{B} + 1)$, ses interprétations signées et non signées et la valeur

des bits C et V [2].

2. l'addition $+1$ est implantée en positionnant C_0 à 1

Chapitre 6

Logique séquentielle

6.1 Définitions

6.1.1 Logique séquentielle *vs.* logique combinatoire

En logique combinatoire, les états de sortie ne dépendent que de l'état des entrées. En logique séquentielle, les sorties dépendent des entrées mais également de l'état du système.

Soit un système séquentielle dont l'entrée est notée X, la sortie Y et l'état Q.

$$\begin{cases} Q = f(X, Q) \\ Y = g(X, Q) \end{cases}$$

La logique séquentielle permet de réaliser des circuits dont le comportement est variable avec le temps.

L'état du système constitue une mémoire du passé.

6.1.2 Synchrone *vs.* asynchrone

Dans les système *asynchrones*, le moment des changements d'état des divers composants dépendent des temps de réponse des autres composants ainsi que du temps de propagation des signaux.

Ces changements d'état peuvent donc subir certains *aléas*.

Pour éviter ces aléas, on peut décider de synchroniser les changements d'états en les contraignants pour survenir à des moments précis. Le plus souvent, pour ce faire, on utilise une horloge qui délivre un signal oscillant de façon périodique entre 0 et 1. Le systèmes est alors dit *synchrone*.

6.2 Les bascules

Une bascule est un composant qui mémorise une information élémentaire. Elle possède en général deux sorties complémentaires Q et $\overline{Q}$.

Le fonctionnement d'une bascule est basé sur le principe du verrou (latch).

La figure 6.1 illustre le principe de fonctionnement du verrou.

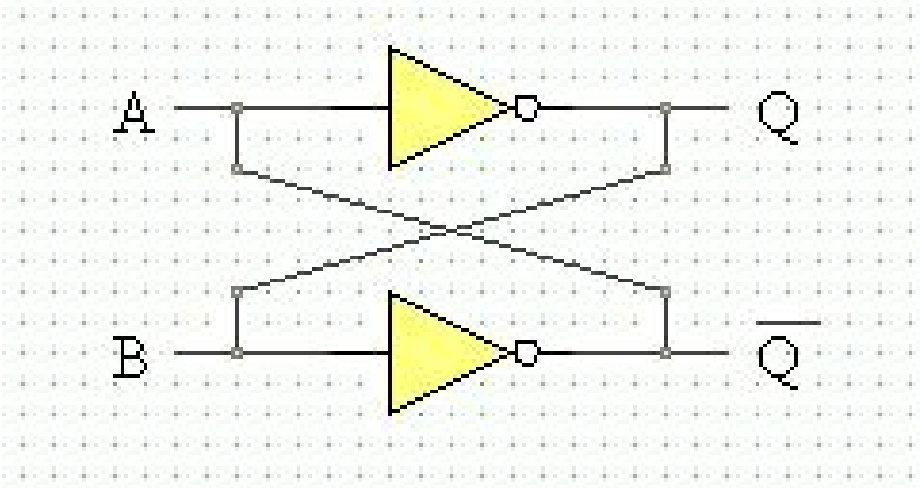

FIGURE 6.1 – Principe du verrou

6.2.1 Bascule RS

La figure 6.2 donne le principe de fonctionnement de la RS.

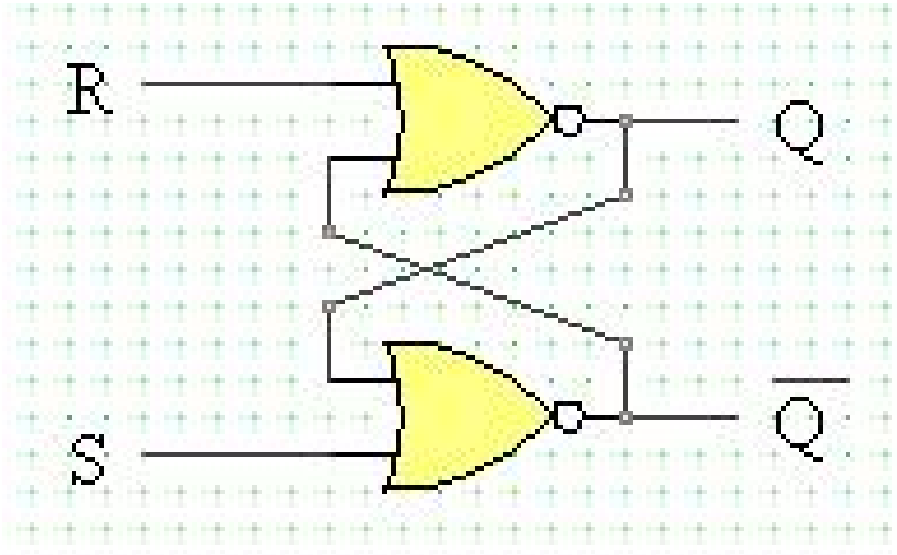

FIGURE 6.2 – Logigramme de la bascule RS (portes NOR)

On en déduit la table de vérité 6.1

Cette table de vérité se synthétise en la table 6.2.

La combinaison 11 est à proscrire car il n'est pas correct d'avoir à la fois Q et $\overline{Q}$ à 0.

R est une abbréviation de "Reset" (mise à zéro) et S est une abbréviation de "Set" (mise à un).

Un autre logigramme possible de la bascule RS est donné par la figure 6.3.

R	S	Q (avant)	Q (après)	$\overline{Q}$ (après)
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

TABLE 6.1 – Table de vérité de la bascule RS

R	S	Q	$\overline{Q}$	fonction
0	0	Q	$\overline{Q}$	mémoire
0	1	1	0	Mise à un
1	0	0	1	Mise à zéro
1	1	0	0	Prohibé

TABLE 6.2 – Table de vérité condensée de la bascule RS

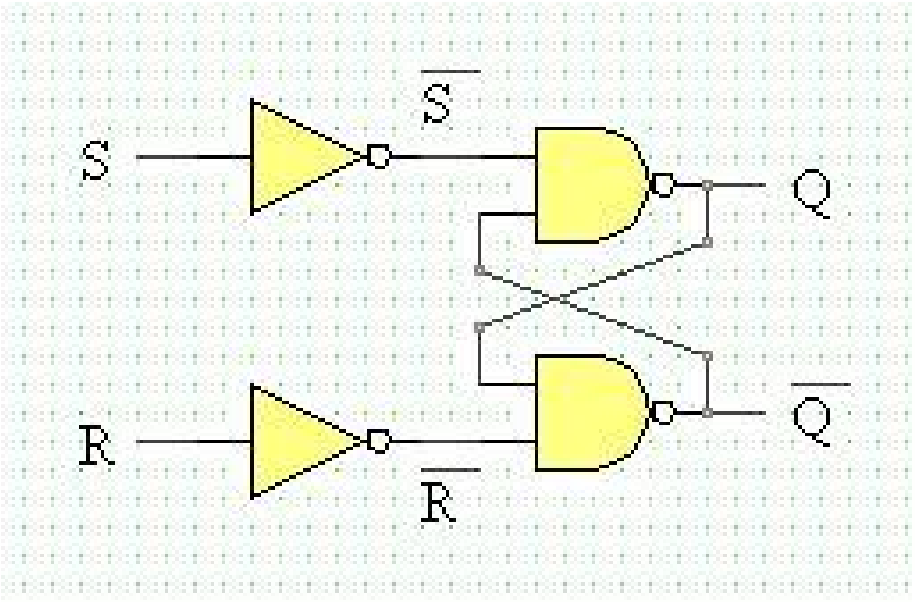

FIGURE 6.3 – Logigramme de la bascule RS (portes NAND)

6.2.2 Bascule RST - RS Clock

La bascule RST ou RS Clock est identique à la bascule RS si ce n'est que les entrées R et S ne sont prises en compte que lorsque le signal d'horloge H est à 1. Il s'agit donc d'une bascule synchrone.

Le logigramme de la bascule RST est donné par la figure 6.4.

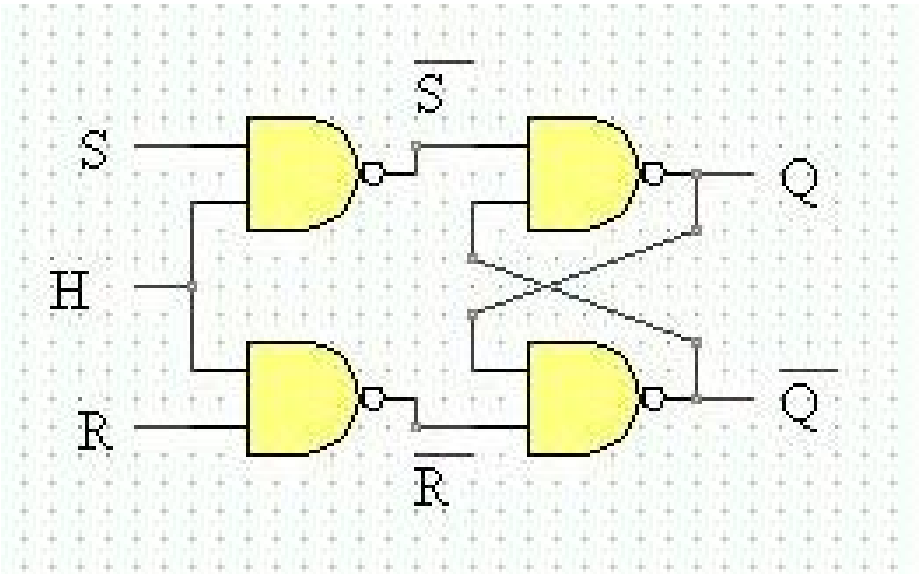

FIGURE 6.4 – Logigramme de la bascule RST

Lorsque le signal d'horloge vaut 1, le fonctionnement est identique à la bascule RS. En revanche, lorsque le signal d'horloge vaut 0, la bascule est bloquée car les changements de R et de S n'ont pas d'influence sur les valeurs de Q et $\overline{Q}$. Les valeurs mémorisées sont celles qui avaient cours lorsque que le signal d'horloge est passé de 1 à 0.

Comme indiqué sur la figure 6.5, le signal d'horloge doit être une succession d'impulsions périodiques de largeur t_p très petite de façon à éviter les évolutions de R et de S lorsque le signal d'horloge est à 1.

Ainsi les modifications de Q et $\overline{Q}$ ne pourront intervenir que lors de ces impulsions. Le reste du temps Q et $\overline{Q}$ seront blockées. On note Q_n la valeur de Q pendant le n^{e}intervalle précédent la n^{e}impulsion et Q_{n+1} la valeur de Q après cette impulsion.

On obtient alors la table de vérité 6.3 synthétisée par la table de vérité condensée 6.4.

6.2.3 Bascule JK

Les bascules RS et RST présente une lacune. En effet, selon l'implantation de la bascule, la combinaison 11 est interdite ou sujette à indétermination des sorties.

La bascule JK comble cette lacune. Basée sur une bascule RST, la bascule JK interdit la combinaison 11. Le logigramme de la bascule RS est donné sur la figure 6.6.

Les entrées de la bascule RST inclue dans la bascule JK sont définies par

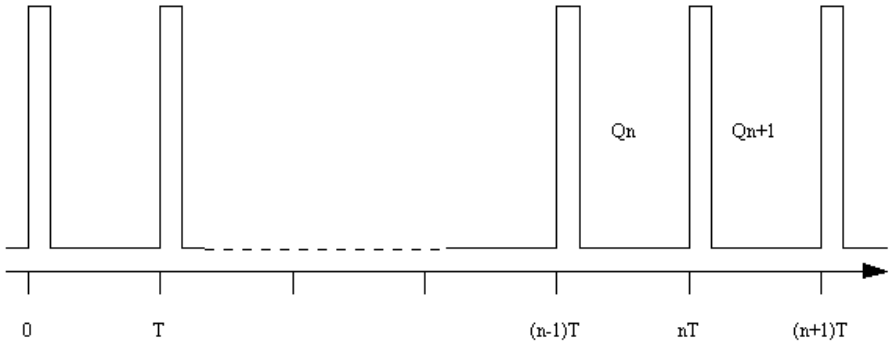

FIGURE 6.5 – Signal d'horloge

R	S	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	?
1	1	1	?

TABLE 6.3 – Table de vérité de la bascule RST

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	?

TABLE 6.4 – Table de vérité condensée de la bascule RST

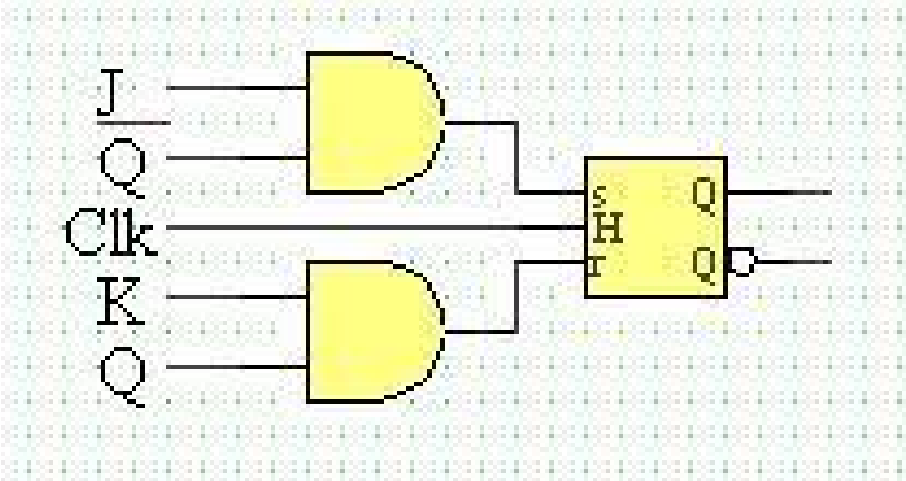

FIGURE 6.6 – Logigramme de la bascule JK

$$\begin{cases} S = J\overline{Q} \\ R = KQ \end{cases}$$

De ce fait, on peut déduire la table de vérité 6.5 de la bascule JK.

J	K	Q_n	$\overline{Q_n}$	S	R	Q_{n+1}
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	1	0	1
1	1	1	0	0	1	0

TABLE 6.5 – Table de vérité de la bascule JK

Cette table de vérité peut être condensée (cf. Tab. 6.6).

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

TABLE 6.6 – Table de vérité condensée de la bascule JK

De ces tables de vérité, on peut déduire la table de transition 6.7 qui en fonction de l'état Q_n constaté et de l'état Q_{n+1} désiré, donne la combinaison des entrées J et K à positionner.

On peut également déduire l'expression algébrique de Q_{n+1}.

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

TABLE 6.7 – Table de transition de la bascule JK

$$Q_{n+1} = J\overline{Q_n} + \overline{K}Q_n$$

Entrées asynchrones

La bascules JK possède des entrées dites asynchrones car elles agissent indépendament du signal d'horloge.

Ces entrées sont nommées *Pr* (Preset) et *Cr* (Clear). Elles sont, en général, actives à l'état bas, c'est-à-dire qu'elle agissent sur le comportement de la bascule lorsqu'elles valent 0.

La figure 6.7 donne le logigramme interne d'une bascule JK avec des entrées asynchrones *Pr* et *Cr* actives à l'état bas.

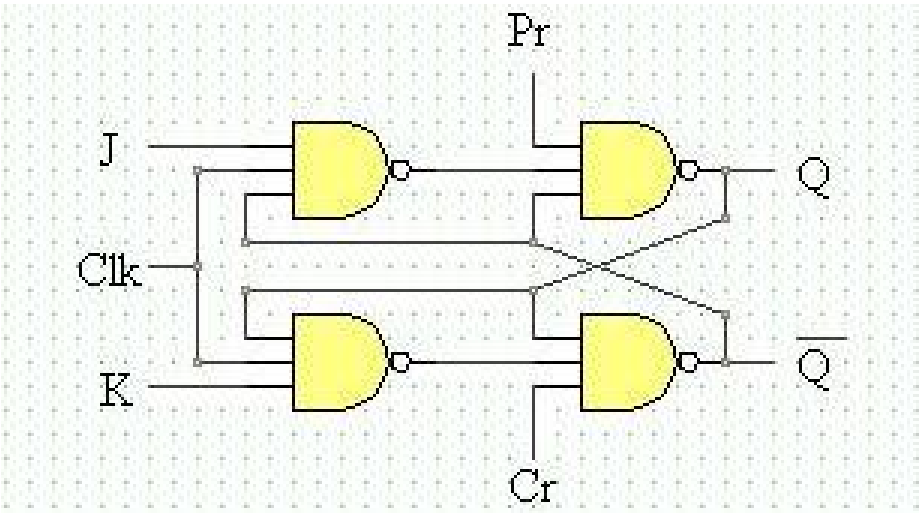

FIGURE 6.7 – Logigramme de la bascule JK avec entrées asynchrones *Pr* et *Cr*

Les entrées *Pr* et *Cr* permettent de positionner l'état initial de la bascule pour éviter tout aléa.

En fonctionnement normal, *Pr* et *Cr* valent 1. Lorsque *Pr* vaut 0, la sortie Q est positionnée à 1. Lorsque *Cr* vaut 0, la sortie Q est positionnée à 0 (cf. Tab. 6.8).

Bascule JK Maître-Esclave

La combinaison JK=11 a pour vocation à faire commuter (inverser) la sortie de la bascule. Or avec le schéma de la figure 6.7, durant la largeur de l'impulsion du signal d'horloge, il est possible que la sortie commute plusieurs fois, et même un nombre

Pr	Cr	Q
1	1	Q
0	1	1
1	0	0
0	0	interdit

TABLE 6.8 – Action des entrées asynchrones *Pr* et *Cr* sur la bascule JK

indéterminé à l'avance. Il n'est alors pas possible de prévoir la valeur de Q. Le résultat est ambigu.

Pour éviter ce problème, on donne le schéma 6.8 de la bascule JK maître-esclave.

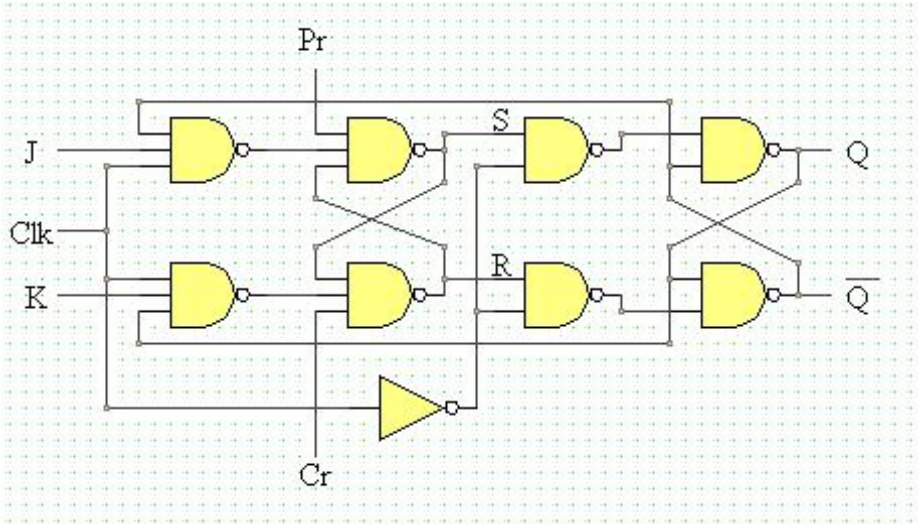

FIGURE 6.8 – Logigramme de la bascule JK maître-esclave

Une bascule JK Maître-Esclave est constituée de deux bascules RS montées en cascade. Les entrées de la première (maître) sont asservies aux sorties de la seconde (esclave). Le signal de la bascule esclave est inversé par rapport à celui de la bascule maître, si bien que les deux bascules ne peuvent pas commuter en même temps.

Pendant la n^{e}impulsion, le signal d'horloge est haut pour la bascule maître et bas pour la bascule esclave. L'état de Q est donc inchangé pendant toute la durée de l'impulsion.

L'état de sortie de la bascule maître est donné par la table de vérité de la bascule JK.

Lorsque le signal d'horloge passe à 0, la bascule maître est bloquée et la bascule esclave est libérée.

$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} Q_M = 1 \\ \overline{Q_M} = 0 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} S = 1 \\ R = 0 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Q_{n+1} = 1 \\ \overline{Q_{n+1}} = 0 \end{array} \right\} \\ \left\{ \begin{array}{l} Q_M = 0 \\ \overline{Q_M} = 1 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} S = 0 \\ R = 1 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} Q_{n+1} = 0 \\ \overline{Q_{n+1}} = 1 \end{array} \right\} \end{array} \right.$$

L'état de la bascule maître est transmis à l'esclave lors de la transition $1 \rightarrow 0$ du

signal d'horloge (front descendant [1]).

6.2.4 Bascule D

Une bascule D (Delay) est construite à partir d'une bascule JK en l'entrée D sur J et son complémentaire sur K.

Les bascules D fonctionnent sur front du signal d'horloge.

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

TABLE 6.9 – Table de vérité de la bascule D

Sur la table de vérité 6.9, on constate que l'état de la sortie Q_{n+1} ne dépend pas de Q_n. Cette table de vérité peut être condensée par la table 6.10.

D	Q_{n+1}
0	0
1	1

TABLE 6.10 – Table de vérité condensée de la bascule D

Cette table montre que quel que soit l'état initial de la bascule, la valeur de l'entrée est répercutée sur la sortie au moment du front de l'horloge.

En revanche, tant qu'il n'y a pas de front actif sur la bascule les évolutions de l'entrée n'ont aucune influence sur la sortie. La bascule D a mémorisé l'état de l'entrée lors du dernier front actif.

6.2.5 Bascule T

La bascule T (trigger) est construite à partir d'une bascule JK sur laquelle l'entrée T est présentée sur les entrées J et K.

La bascule T fonctionne sur front du signal d'horloge.

On obtient alors la table de vérité

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

TABLE 6.11 – Table de vérité de la bascule T

Si l'entrée T vaut 0, la sortie reste inchangée à chaque front actif du signal d'horloge.

Si l'entrée T vaut 1, la sortie commute à chaque front actif du signal d'horloge.

1. Par opposition, une transition $0 \rightarrow 1$ est appelé front montant

6.3 Les registres

6.3.1 Registres de mémorisation

Un registre de mémorisation permet la mémorisation de n bits. Il est constitué de n bascules mémorisant chacune 1 bit.

La figure 6.9 représente un registre de mémorisation 4 bits contruit à partir de bascules D.

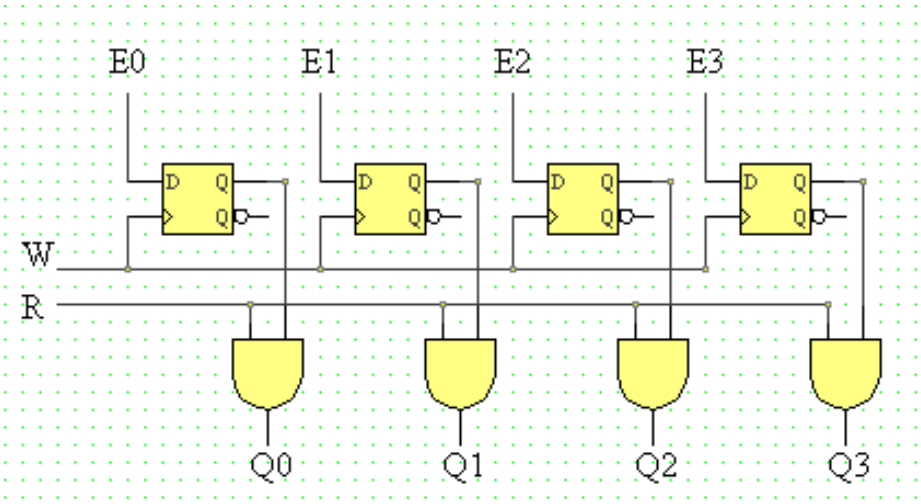

FIGURE 6.9 – Registre de mémorisation 4 bits construit à partir de bascules D

Sur cette figure, le signal W est relié aux entrées d'horloge de chaque des bascules. Un front montant sur W permet de mémoriser sur chaque bascule les entrées E_0, E_1, E_2 et E_3.

Un niveau haut sur R permet de répercuter les valeurs mémorisées sur les sorties Q_0, Q_1, Q_2 et Q_3.

6.3.2 Registres à décalage

Dans un registre à décalage, les bascules sont connectées de telle sorte que :

— la sortie de la i^ebascule soit l'entrée de la $(i+1)^e$. On obtient alors un registre à décalage à gauche [2].

— la sortie de la i^ebascule soit l'entrée de la $(i-1)^e$. On obtient alors un registre à décalage à droite [3].

La figure 6.10 représente un registre à décalage à gauche 4 bits. À chaque front montant de l'horloge H, l'entrée série ES est répercuté sur S_0, chaque S_0 sur S_1, S_1 sur S_2 et S_2 sur S_3, cette dernière étant équivalente avec la sortie série du registre.

Ce registre peut être utilisé pour transformer une entrée série (succession de bit dans le temps) en une sortie parallèle (codage spatial).

La figure 6.11 représente un registre deux bits à sortie série. Selon la valeur du signal de commande X, l'entrée est effectuée en série ($X = 1$) ou en parallèle ($X = 0$).

2. Un décalage à gauche correspond à une multiplication par 2
3. Un décalage à droite correspond à une division entière par 2

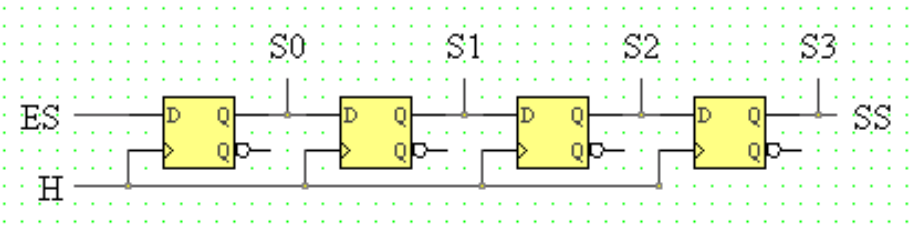

FIGURE 6.10 – Registre à décalage à gauche

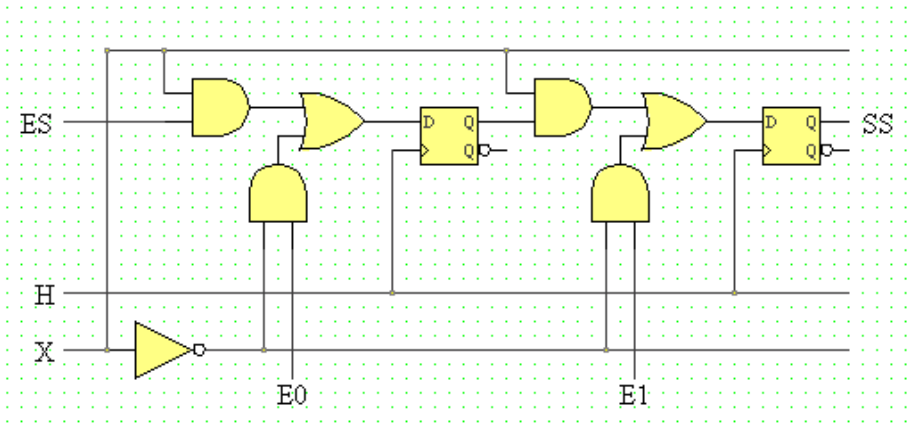

FIGURE 6.11 – Registre à décalage sortie série

La figure 6.12 représente un registre deux bits à entrée parallèle et sortie parallèle. Le signal X permet d'activer ou d'inhiber le signal d'horloge. Si X vaut 0, alors $Pr = Cr = 1$, le registre fonctionne alors normalement en entrée série. Si $X = 1$, la mémorisation est effectuée par activation (au niveau bas) des entrées asynchrones *Pr* et *Cr*.

$$\left\{ \begin{array}{l} E_i = 1 \Rightarrow \left\{ \begin{array}{l} Pr_i = 0 \\ Cr_i = 1 \end{array} \right\} Q_i = 1 \\ E_i = 0 \Rightarrow \left\{ \begin{array}{l} Pr_i = 1 \\ Cr_i = 0 \end{array} \right\} Q_i = 0 \end{array} \right\} \Rightarrow Q_i = E_i$$

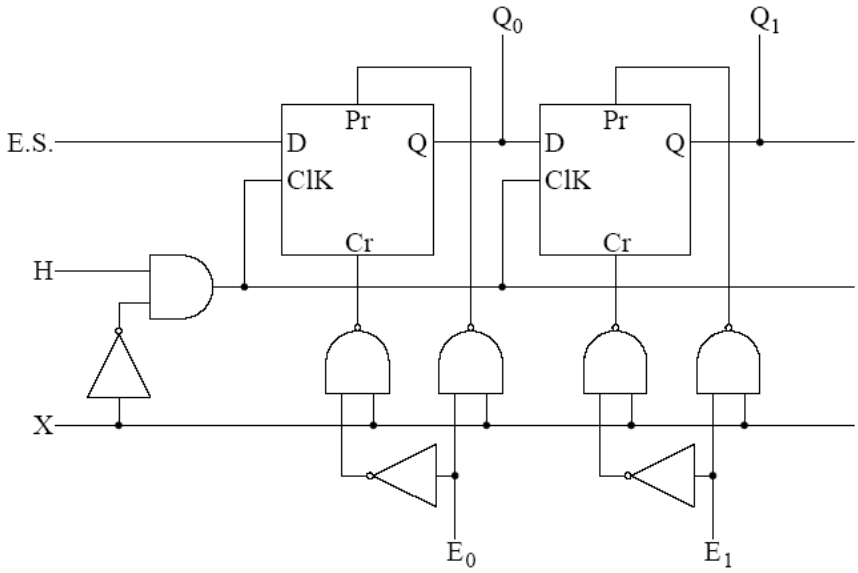

FIGURE 6.12 – Registre à décalage deux bits entrée parallèle sortie parallèle

La figure 6.13 représente un registre à décalage universel 4 bits. Les entrées A, B, C et D sont des entrées séries. Les entrées E_0 et E_1 sont des entrées séries selon le sens du décalage. Les sorties Q_A, Q_B, Q_C et Q_D sont des sorties parallèles, mais les sorties Q_A et Q_D peuvent être utilisées en sortie série.

Le fonctionnement de ce registre est commandé par les lignes S_0 et S_1.

$$Clk = \overline{\overline{H} + \overline{S_0}.\overline{S_1}} = H.(S_0 + S_1)$$

Le signal d'horloge est donc inhibé pour la combinaison $S_0 = S_1 = 0$.

Pour sélectionner le chargement paralèlle de A, B, C et D il faut $\overline{\overline{S_0} + \overline{S_1}} = S_0 S_1 = 1$.

Le décalage à gauche (entrée E_1) est activé pour la combinaison $S_0 = 1$ et $S_1 = 0$.

Le décalage à droite (entrée E_0) est activé pour la combinaison $S_0 = 0$ et $S_1 = 1$.

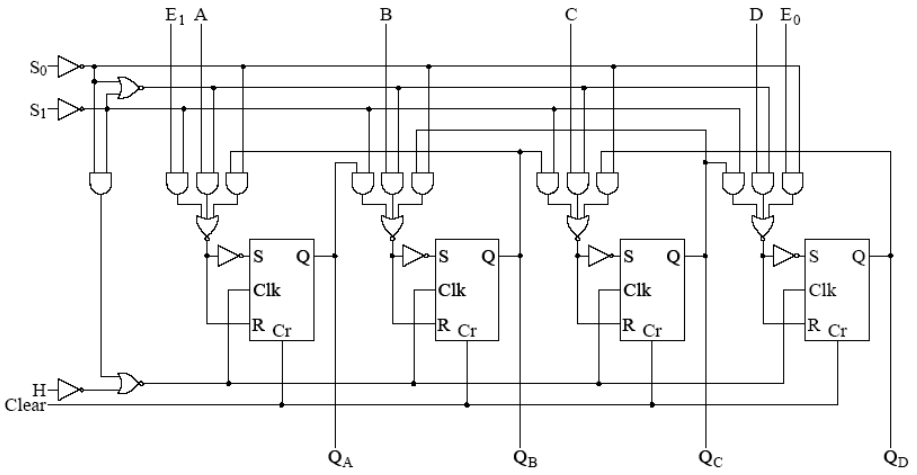

FIGURE 6.13 – Registre à décalage universel 4 bits

6.4 Compteurs

Un compteur (ou séquenceur) est un ensemble de n bascules interconnectées par des portes logiques permettant ainsi de mémoriser des mots de n bits.

Au rythme d'un signal d'horloge, le compteur décrit une séquence de combinaisons (d'états binaires).

La séquence peut comporter au maximum 2^n états différents. Ces états restent stables et accessibles entre 2 impulsions de l'horloge.

Le nombre N d'états du compteur est appelé *modulo* du compteur.

$$N \leq 2^n$$

Si $N \neq 2^n$, un certain nombre d'états ne sont jamais utilisés.

On distingue les compteurs asynchrones pour lesquels l'entrée d'horloge d'une bascule est liée à la sortie de la bascule précédente, des compteurs compteurs synchrones qui recoivent le même signal d'horloge au même instant.

6.4.1 Compteurs asynchrones

La figure donne le schéma d'un compteur asynchrone modulo 8 construit à partir de bascules JK.

Le signal d'horloge est reçu par la première bascule, celle dont la sortie donne le bit de poids faible (LSB).

Le signal d'horloge des autres bascules est fourni par la sortie des bascules de rang précédent.

Pour chacune de ces bascules les entrées J et K valent 1. La sortie d'une bascule particulière commutera sur un front actif de son signal d'horloge.

Sur l'exemple suivant, supposons que les bascules fonctionnent sur front descendant et que la combinaison $Q_2Q_1Q_0$ vaille 000.

La sortie Q_0 commute sur front descendant de H, la sortie Q_1 commute sur front descendant de Q_0 et la sortie Q_2 commute sur front descendant de Q_1.

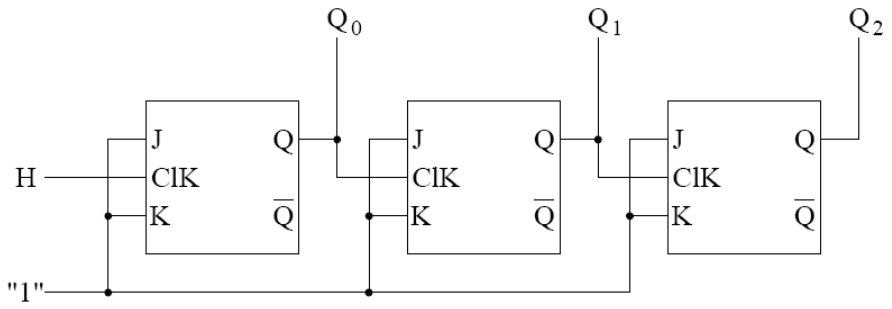

FIGURE 6.14 – Compteur asynchrone modulo 8 construit avec des bascules JK

Ainsi, la figure 6.15 représente le chronogramme (évolution des valeurs des signaux en fonction du temps) des sorties Q_2, Q_1 et Q_0 en fonction de H.

À partir de ce chronogramme, on peut établir la liste des états successifs des sorties Q_2, Q_1 et Q_0 (cf. Tab. 6.12).

Impulsion	Q_2	Q_1	Q_0
état initial	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
7	1	1	1
8	0	0	0

TABLE 6.12 – États succesifs des sorties Q_2, Q_1 et Q_0 du compteur asynchrone modulo 8

Il s'agit donc d'un compteur modulo 8 comptant de 0 à 7. Il est possible d'obtenir un décompteur comptant de 7 à 0 en lisant les sorties $\overline{Q_i}$.

Pour remettre à 0 ou charger une valeur dans le compteur, on utilise des fonctions logiques dont les sorties sont raccordées aux entrées asynchrones Pr et Cr comme indiqué sur la figure 6.16.

Pour une mise à zéro, on positionne R à 0 ce qui implique $J = K = 0$ interdisant ainsi le basculement et $Pr = 1$ et $Cr = 0$ pour positionner Q à 0.

Pour charger une autre valeur, on positionne $R = 1$, $DS = 0$ et la valeur à charger sur D. Ainsi, $J = K = 0$ interdit le basculement interdit. Si $D = 0$, on a $Pr = 1$ et $Cr = 0$ et donc $Q = 0$. Si $D = 1$, on a $Pr = 0$ et $Cr = 1$ et donc $Q = 1$.

On peut également souhaiter avoir un compteur avec un modulo N qui ne soit pas une puissance de 2 (on parle de cycle incomplet). On utilise alors n bascules de telles sorte que $2^{n-1} < N < 2^n$. On asservit alors les entrées Cr de telle sorte qu'elle passe à 0 à la première combinaison interdite. Par exemple, la figure 6.17 représente un compteur modulo 10 de 0 à 9.

Du fait des temps de transition des portes, la combinaison interdite apparaît un court instant.

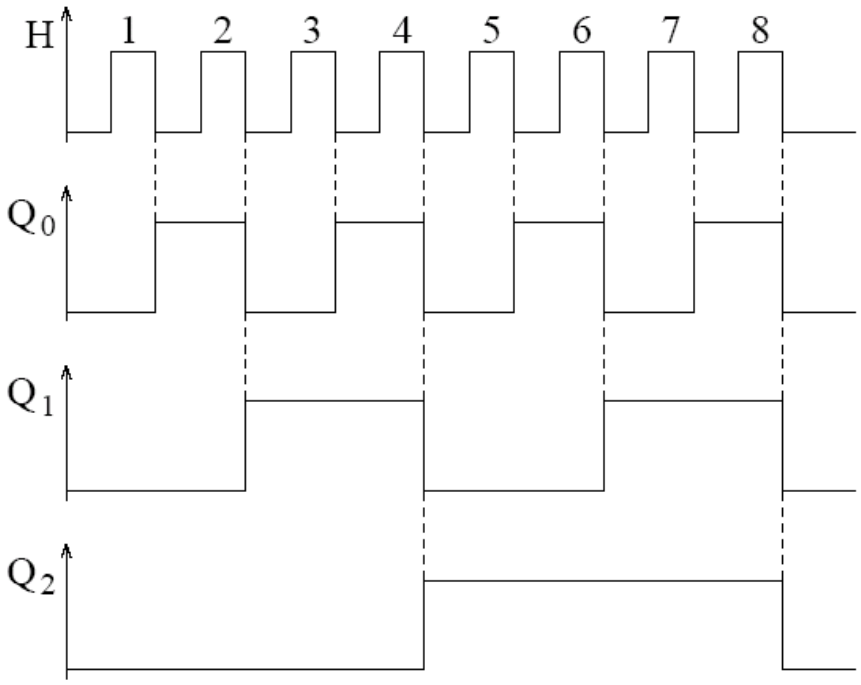

FIGURE 6.15 – Chronogramme du compteur asynchrone modulo 8

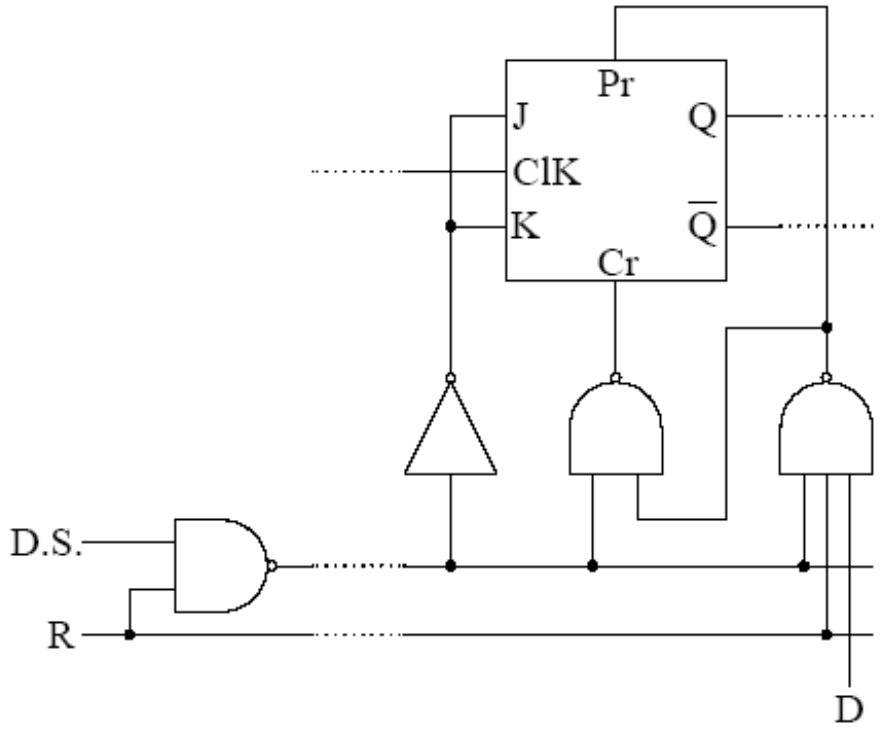

FIGURE 6.16 – Utilisation des entrées asynchrones Pr et Cr pour charger une valeur dans le compteur

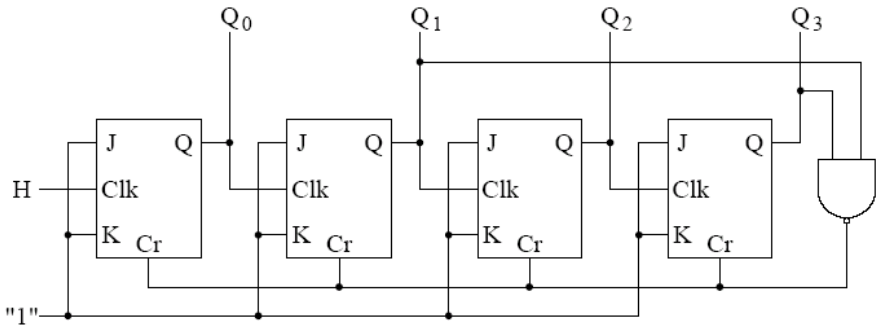

FIGURE 6.17 – Logigramme d'un compteur asynchrone modulo 10 de 0 à 9

Pour obtenir un compteur modulo N avec un cycle différent du comptage 0 à $N-1$, on réalise un compteur de 0 à $N-1$ dont les sorties sont connectées à un transcodeur vers le cycle désiré.

Incovénient des compteurs asychrones Chaque bascule du compteur asynchrone a un temps de transition. Les fronts actifs ne parviennent pas au même instant pour toutes les bascules. Les temps de transition des bascules se cumulent par rapport au front actif du signal d'horloge.

Il y a ainsi des états transitoires entre l'instant où le front actif parvient à la première bascule et celui où il parvient à la dernière.

Par exemple, la figure 6.18 met en évidence les états transitoires intervenant sur un compteur modulo 16 lorsque celui-ci passe de l'état 0111 à l'état 1000.

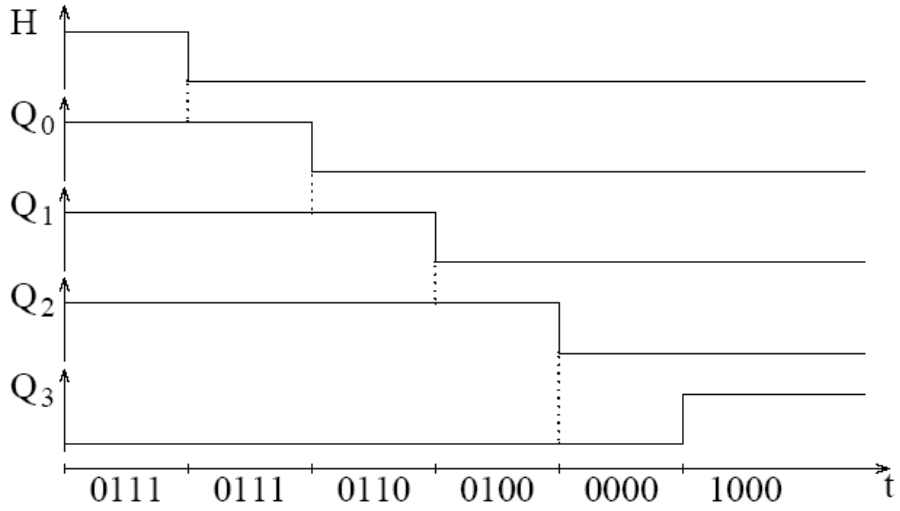

FIGURE 6.18 – Illustration des états transitoires lors du passage de l'état 0111 à l'état 1000

6.4.2 Compteurs synchrones

Dans un compteur synchrone, toutes les bascules recoivent le même signal d'horloge au même instant.

Pour que le compteur décrive la séquence souhaitée, il faut définir les fonctions logiques des entrées de chaque bascule en fonction de l'état précédent du compteur.

On utilise la table de transition des bascules utilisées pour définir à quelles valeurs doivent être fixées les entrées pour passer d'un état à un autre.

Par exemple, si on souhaite réaliser un compteur modulo 8 comptant de 0 à 7 à l'aide de bascules T. Ce compteur nécessite 3 bascules dont les entrées sont nommées T_2, T_1 et T_0 et les sorties Q_2, Q_1 et Q_0.

De ce tableau, on déduit les expressions logiques des entrées des bascules en fonction des sorties à l'intant n. Sur cet exemple, on obtient

états	Sorties à l'instant n			Sorties à l'instant $n+1$			Entrées des bascules		
	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	1
2	0	1	0	0	1	1	0	0	1
3	0	1	1	1	0	0	1	1	1
4	1	0	0	1	0	1	0	0	1
5	1	0	1	1	1	0	0	1	1
6	1	1	0	1	1	1	0	0	1
7	1	1	1	0	0	0	1	1	1

TABLE 6.13 – Tableau définissant les entrées pour obtenir un compteur synchrone comptant de 0 à 7 à partir de bascules T

$$\begin{cases} T_0 = 1 \\ T_1 = Q_0 \\ T_2 = Q_1.Q_0 \end{cases}$$

Le même raisonnement peut être tenu pour réaliser des compteurs synchrones avec d'autres types de bascules, des décompteurs, des compteurs à cycles incomplets ou des compteurs à cycle non naturel.

Index