

Introduction à l'Electronique Numérique

Jean-Christophe Le Lann

21 août 2016

Table des matières

Chapitre 1

Le Monde Numérique

1.1 Une histoire simplifiée

Les mathématiques qui gouvernent un ordinateur ou tout système numérique tiennent dans un nombre minuscule de lois remarquablement simples : l' *Algèbre de Boole*. On doit son invention à George Boole (1815–1864), qui se définissait autant comme mathématicien que comme philosophe. Son traité, publié en 1854, s'intitule “An Investigation of the Laws of Thought”. A l'époque, il n'était nullement question de machines, mais d'une quête sur le raisonnement lui-même. Boole fréquentait également des salons, et son algèbre eût un certain succès dans les jeux mondains. Ce n'est qu'au XX^e siècle que l'on prit conscience de capacités phénoménales de ces principes en terme technologiques : Alan Turing conceptualisa et fit construire la première machine numérique programmable, qui participa de manière très significative à la victoire des Alliés lors de la seconde guerre mondiale. Sa machine permit en effet de décoder les messages échangés par le troisième Reich, cryptés par la machine Enigma. On peut également remarquer, un peu à l'instar de Boole, qu'une grande partie de son activité scientifique fût par la suite dédiée à l'intelligence artificielle, entérinant cet intérêt entre l'inerte et la pensée elle-même. Une troisième étape fondamentale dans l'essor de cette Algèbre (et du monde numérique qu'elle sous-tend) est l'invention du transistor par Brattain, Shockley et Bardeen. Ces physiciens, spécialistes des semi-conducteurs¹, avaient pour mission, au sein des laboratoires Bell, de trouver une alternative aux tubes-à-vide. Bien que ces transistors aient également d'excellentes propriétés en matière d'amplification analogique, c'est leur capacité en commutation qui en fit leur essor que l'on connaît. Mais c'est Intel, à partir de 1971 et de son processeur 4004, qui démocratisa l'utilisation massive des systèmes à base de microprocesseurs.

Cette histoire est simplifiée à l'extrême : on a omis Charles Babbage, Conrad Suze, Von Neumann et tant d'autres ! On peut d'ailleurs noter que la paternité de l'ordinateur est une question sur laquelle les historiens de Sciences ne s'accordent pas, tant l'Histoire est réellement sinueuse. On pourrait aussi parler des influences venant d'autres disciplines : notamment Claude Shannon, qui utilisa la logique booléenne dans la théorie naissante de l'Information.

1.2 Un ordinateur vu d'en haut

1. ...entre autre : Bardeen est le seul physicien à avoir décroché deux prix Nobel de Physique, le second ayant trait aux supra-conducteurs

Chapitre 2

Représentations numériques

2.1 Notion de bases ou radix

Les nombres que nous manipulons tous les jours s'expriment en base (ou *radix* 10. On parle de base 10 car on utilise 10 symboles (chiffres) allant de 0 à 9 inclus. Par la suite, on s'autorisera à parler de *digits*. Dans un nombre, ces digits sont en fait des *coefficients* du système décimal : pour former un nombre, on multiplie ces coefficients par les puissances successives de 10. Par exemple le nombre 142 s'exprime ainsi :

$$142 = 1 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

Il en est de même des nombres fractionnaires comme :

$$142.34 = 1 \times 10^2 + 4 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2}$$

La formule générale pour la base 10 est bien entendu :

$$x = \sum_{i=-\infty}^{i=\infty} c_i \cdot 10^i, \text{ avec } c_i \in \{0 \dots 9\}$$

et se généralise pour toute base r :

$$x = \sum_{i=-\infty}^{i=\infty} c_i \cdot r^i, \text{ avec } c_i \in \{0 \dots (r-1)\}$$

Les ordinateurs n'utilisent pas la base 10, mais la base 2, qui ne possède que deux valeurs de coefficients possibles : 0 et 1.

Afin d'éviter toute confusion par la suite, nous plaçons en indice du nombre l'indication de la base utilisée, ce qui évitera de nous perdre lors des changements de base. Par exemple, on peut noter :

$$10110_{10} = 10011101111110_2$$

En pratique, il existe deux autres bases indispensables aux informaticiens et électroniciens¹ : la base octale (base 8) et la base hexadécimale (base 16). La première n'utilise que les digits allant de 0 à 7. En hexadécimal, on doit introduire de nouveaux symboles au delà du 9 : on utilise –dans l'ordre– les lettres (majuscules ou minuscules) A, B, C, D, E et F, correspondant respectivement à 10, 11, 12, 13, 14 et 15. La table suivante montre les 16 premiers nombres dans le système décimal, binaire, octal et hexadécimal, ainsi que deux autres représentations sur lesquelles nous reviendrons un peu plus tard).

1. Personnellement, je n'ai jamais eu à utiliser la première...

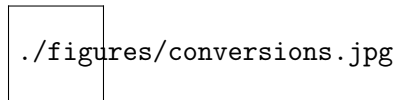


FIGURE 2.1 – Table des correspondances entre bases 10, 2, 8, 16

2.2 Conversion entre bases

Grâce aux formules précédentes, nous savons transformer un nombre exprimé dans une base r en un nombre décimal. Il est à présent important de savoir faire l'inverse, à savoir : convertir un nombre décimal x_{10} vers tout autre base r . Ceci peut se faire de manière mécanique : il suffit de diviser le nombre x (dividende) par r (diviseur), puis de répéter l'opération sur le quotient jusqu'à ce que ce quotient soit nul. Le premier reste obtenu sera le digit le plus à droite², et le dernier reste le digit le plus à gauche³.

Dans le cas de nombres fractionnaires, il faut procéder de la manière suivante : multiplier le nombre par r jusqu'à obtenir un nombre sans partie fractionnaire. Ceci n'est évidemment pas toujours possible.

2.2.1 Exemples

- **Convertir le nombre 12_{10} en binaire**

Solution :

$$12/2 = 6 + 0$$

$$6/2 = 3 + 0$$

$$3/2 = 1 + 1$$

$$1/2 = 0 + 1$$

Par conséquent : $(12)_{10} = (1100)_2$

- **Convertir $(0.75)_{10}$ en binaire**

Solution :

$$0.75 \times 2 = 1.5 = 1 + 0.50$$

$$0.50 \times 2 = 1.0 = 1 + 0$$

Ainsi :

$$(0.75)_{10} = (0.11)_2$$

- **Convertir $(0.39654)_{10}$ en binaire**

Solution :

2. En base binaire, on parle de LSB : least significant bit

3. MSB : most significant bit

$$\begin{aligned}
0.39654 \times 2 &= 0.79308 = 0 + 0.79308 \\
0.79308 \times 2 &= 1.58616 = 1 + 0.58616 \\
0.58616 \times 2 &= 1.17232 = 1 + 0.17232 \\
0.17232 \times 2 &= 0.34464 = 0 + 0.34464
\end{aligned}$$

etc...Cet conversion n'a pas de fin. La partie fractionnaire ne peut s'exprimer sous la forme d'une somme exacte de puissance de 2. On notera la solution :

$$(0.39654)_{10} = (0.0110...)_{2}$$

Il est important que vous sachiez rapidement effectuer ces conversions avec votre ordinateur ou calculatrice préférée⁴.

2.2.2 Conversions binaire, hexadécimal, octal

Comme $2^4 = 16$, il s'en suit que la représentation binaire de chaque digit de la base 16 nécessite 4 bits. Pour la base octale, il en suffit de 3. De ce fait, les conversions de nombres binaires vers l'hexadécimal ou l'octal se réalisent simplement, par groupement de bits. En partant de la droite, il suffit par exemple de regrouper 4 bits pour obtenir un digit hexa. Par exemple :

$$\text{le nombre } 101101_2 = 10_1101_2 = 0010_1101_2 = 2D_{16} = \mathbf{0x2D}$$

Dans le cas de nombre fractionnaires, le sens de lecture s'inverse pour la partie fractionnaire. Ainsi :

$$\text{le nombre } 1011.01_2 = 1011.0100_2 = B.4_{16}$$

On procède de même pour la traduction octale.

$$(10110001101011.1111)_2 = (101\ 110\ 001\ 101\ 011\ .\ 111\ 100)_2 = (26153.74)_8$$

2.3 Bits, bytes, nibbles, words

La plus petite unité informatique est le bit, qui est l'abréviation de *binary digit*. Toutefois rares sont les ordinateurs qui ne traitent qu'un seul bit à la fois : la plus petite la grandeur manipulable par la quasi-totalité des ordinateurs reste l'**octet** : c'est l'aggrégation de 8 bits. Sa traduction anglaise est le *byte*, qui se prononce *bait* et qui ne doit pas être confondu avec le bit. Le regroupement de 4 bits est moins connu, mais indispensable : il s'agit du **quartet** ou *nibble* en anglais. Nous verrons plus loin en quoi il est important. Enfin, on parle de **mot** ou *word* pour le regroupement de 16 bits, sans toutefois que cette définition soit parfaitement admise. On parlera d'ailleurs de processeurs qui opèrent sur des mots de 16 bits, 32 bits, 64 bits etc...

Les bits des quartets, octets et mots sont généralement manipulés simultanément, lorsque la machine possède les circuits qui le permettent. Nous y reviendrons par la suite.

4. Pour votre information, il est possible d'utiliser Google pour effectuer cela : tapez "0x123 in octal" dans l'invite de Google...Vous obtenez 0o443

2.4 L'addition binaire

Considérons le cas de l'addition binaire. Seuls 4 combinaisons sont à étudier :

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ et une retenue de } 1$$

Nous verrons, au chapitre suivant, comment forcer le silicium à réaliser électriquement ce calcul !

2.5 Données composées et données symboliques

2.5.1 Données composées

Les représentations simples (bits, octets etc) ne sont généralement pas suffisantes. En effet, un problème informatique particulier nécessite une représentation particulière des données manipulées. Par exemple, si on souhaite manipuler une coordonnées (x,y), on devra agglomérer deux nombres, qui peuvent posséder des plages de valeurs éventuellement différentes. Cette notion se retrouve dans les langages de programmation classiques : la notion de struct en langage C, de record en VHDL, et d'attributs en Java, etc. Toutefois, il est toujours possible de créer un nombre particulier, qui rassemble plusieurs informations. Dans le cas de coordonnées x et y représentées respectivement par un mot de 16 bits et un octet, on peut créer un nombre *pos* par le calcul suivant :

$$pos = (x \ll 8) + y$$

Ce calcul consiste à décaler x de huit positions à gauche, afin de laisser de la place à y : lors de l'addition, y aura exactement 8 emplacements pour que ses 8 bits soient positionnés.

2.5.2 Données symboliques ou énumérées.

Il existe d'autres cas où la représentation sous forme de nombre n'est pas imposée par la nature de l'algorithme. Par exemple, c'est le cas si l'on parle d'un ensemble de 5 couleurs (bleu, blanc, rouge, vert, noir) que peut prendre une variable *c*. Dans ce cas, il faudra passer par un *encodage* des valeurs possibles de la variable : on choisira par exemple 0=bleu, 1=blanc, 2=rouge, 3=vert, 4=noir. Le calculateur numérique devra bien entendu connaître cette règle d'encodage. Généralement, les langages informatiques réalisent cet encodage pour vous. Dans le cas des 5 couleurs, on devra utiliser 3 bits afin de compter jusqu'à 5 :

$$\text{bleu} \rightarrow 0 \rightarrow 000$$

$$\text{blanc} \rightarrow 1 \rightarrow 001$$

$$\text{rouge} \rightarrow 2 \rightarrow 010$$

$$\text{vert} \rightarrow 3 \rightarrow 011$$

$$\text{noir} \rightarrow 4 \rightarrow 100$$

La formule générale stipule que pour n valeurs énumérées, il faut $\lceil \log_2(n) \rceil$ bits de représentation, où le symbole \lceil est une fonction signifiant "entier supérieur ou égal le plus proche de".

Certaines combinaisons binaires peuvent ne pas avoir de sens (101 par exemple, dans notre exemple). L'efficacité de la représentation se mesure par :

$$1 - \frac{2^{\lceil \log_2 \text{Card}(E) \rceil} - \text{Card}(E)}{\text{Card}(E)}$$

Cette efficacité dans la représentation peut avoir une influence non-négligeable sur la constitution physique du processeur : il sera plus ou moins gros, plus ou moins rapide etc. Nous y reviendrons lorsque nous parlerons de machines d'états finis.

2.6 Différents encodages des entiers

Nous avons parlé des représentations des entiers dans différentes bases. Lorsque le nombre total d'entiers à encoder est connu, on peut également chercher à représenter ces entiers par d'autres encodages ou *système de numération*. Cela est très fréquent dans les ordinateurs, car les architectures sont par nature finie en dimension. Hors de question de compter à l'infini ! Il existe de nombreux encodages, du plus simple ou plus compliqué. Par exemple, certains codages prédisposent à la simplification de la détection et même la correction d'erreur lors d'une communication numérique (portable, TV numérique,...) ! Nous ne pouvons nous attarder sur tous ces codages, mais sachez que c'est en fait une discipline à part entière, avec ses problématiques et enjeux propres.

2.6.1 BCD

Le codage BCD signifie "binary coded decimal". Chaque digit décimal 0 à 9 est alors traduit directement par 4 bits. Ceci est évidemment assez naturel.

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

Pour coder un nombre tel 147 il suffit de coder chacun des chiffres 1, 4 et 7 ce qui donne 0001, 0100, 0111. Cette représentation a un intérêt pratique certain : lorsqu'on réalise un appareil qui affiche des valeurs entières de plusieurs digits, le code BCD permet d'isoler l'électronique de chaque digit, de manière triviale.

2.6.2 Code de Gray

Souvenons nous par exemple que le passage de 7 à 8 en binaire classique fait commuter 4 bits : tous les bits changent de valeurs ! Le code de Gray (aussi appelé *codage binaire réfléchi*) vise à éviter de telles commutations simultanées : il est constitué d'une manière telle que *seul 1 digit binaire change entre un entier n et son successeur $n + 1$* . En pratique, cela évite de commuter un grand nombre de dispositifs en même temps, lors d'un comptage régulier. Le codage de gray fait partie d'un ensemble de codes appelés à "distance minimale".

décimal	binaire classique	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Il existe un algorithme pour passer d'un nombre x à l'autre $x + 1$:

- on calcule le nombre de 1 dans x . On inverse le dernier bit de x quand ce nombre de 1 est pair.
- si le nombre de 1 est impair, on inverse le bit à gauche du 1 qui est le plus à droite.

Le terme “binaire réfléchi” provient d’une seconde méthode, graphique. A partir des deux codes

$$0 \rightarrow 0000$$

$$1 \rightarrow 0001$$

on réalise une réflexion dans un miroir, et un ajout de 1 en tête, afin d’obtenir valeurs suivantes :

$$0 \rightarrow 0000$$

$$1 \rightarrow 0001$$

— — — — —

$$2 \rightarrow 0011$$

$$3 \rightarrow 0010$$

En langage informatique comme le C, il est remarquablement facile de passer d’un nombre à sa représentation Gray :

$$g = b(b \gg 1)$$

Nous reviendrons très bientôt sur ces drôles de calculs...

L’inverse est plus compliqué⁵.

2.6.3 Code alphanumérique ASCII

Le code ASCII est à part : il s’agit d’un *encodage des caractères*, normalisé au niveau international dans les années 60. Il était important de s’assurer que les futurs échanges numériques recevraient un sens directement interprétable en terme de caractères : lettres, chiffres, caractères spéciaux, etc... Ils sont numérotés de 0 à 127 : seuls 7 bits suffisent à les représenter. Plusieurs caractères ne sont pas affichables. Le code 10₁₀ représente l’échappement (retour à la ligne). Le chiffre 0 correspond au code ASCII 48₁₀ et le *a* minuscule au code 97₁₀. Au niveau binaire, on représente ainsi la lettre j :

5. Je suis preneur d’une solution élégante...

b7	b6	b5	b4	b3	b2	b1
1	1	0	1	0	1	0

2.6.4 Bit de parité

Lors de la transmission d'une donnée binaire codée sur plusieurs bits, il est fréquent que l'on lui adjoigne un bit supplémentaire, appelé *bit de parité*. La valeur 0 ou 1 dépend des autres bits, de manière à ce que la somme totale des 1 du code binaire soit paire. Ceci permet, à moindre frais, de s'assurer qu'une erreur (simple) ne s'est pas produite lors de la transmission. En effet, le receptrice peut alors vérifier la parité de la donnée reçue. L'hypothèse sous-jacente stipule qu'une seule erreur binaire est tolérée. Au delà, il faudra rajouter des bits plus sophistiqués...

2.7 Représentation des entiers négatifs. Compléments.

Jusqu'ici nous n'avons pas parlé de nombres négatifs. Comment représenter des nombres négatifs? Le premier réflexe est de considérer un nombre négatif comme la composition de deux informations représentées isolément : le signe et la valeur absolue ("magnitude"). Le signe peut être encodé sur 1 seul bit : 0 ou 1 représentant par exemple le + et le - respectivement. Cette représentation est tout à fait envisageable, mais elle se révèle inefficace matériellement.

Certaines opérations sont grandement simplifiées par le recours au *complément* d'un nombre. Les ordinateurs stockent les valeurs négative en binaire, en complément à 2. Observons ce que cela signifie...

Pour toute base r , il existe en fait deux types de compléments : le complément à r et le complément à $r - 1$. Pour la base 10, le complément à 10 d'un nombre s'obtient de la manière suivante : on détecte le digit LSB non-nul et tous les 0 à sa droite. On soustrait alors ce nombre obtenu à 10. On retire à 9 la valeur des autres digits :

exemple : complément à 10 de 23567 Solution :

$$10 - 7 = 3$$

$$9 - 6 = 3$$

$$9 - 5 = 4$$

$$9 - 3 = 6$$

$$9 - 2 = 7$$

Le complément à 10 de 23567 est donc 76433.

Le complément à 1 d'un nombre binaire consiste à simplement inverser tous ses bits.

Le complément à 2 d'un nombre binaire consiste à :

- laisser inchangé le 1 le plus à droite, ainsi que tous les 0 les plus à droite.
- modifier tous les autres digits.

Il existe toute fois une autre méthode, qui se retient plus facilement (peut-être). Elle consiste en une formule simple :

$$-A = /A + 1$$

exemple : pour trouver le complément à deux de 1101100₂. On inverse tous les bits et l'on additionne 1.

$$0010011 + 1 = 0010100$$

2.8 Calcul de soustraction

La soustraction de deux nombres binaires⁶ se fait en utilisant le complément à 2⁷ : soient deux nombres x et y . La soustraction $x - y$ se réalise ainsi :

1. on additionne x et le complément à 2 de y
2. si le résultat présente une retenue finale, on l'oublie !
3. s'il n'y a pas de retenue finale, on prend le complément à 2 du résultat et on place un signe “-” devant le nombre.

exemple : $x = 1101100_2$ et $y = 1011011_2$

$$1101100 - 0100101 = 1\ 0010001$$

Le résultat est donc $x - y = 0010001_2$

2.9 Les nombres en virgule fixe

Nous avons déjà vu que les nombres qui possèdent une partie fractionnaire (sous entendu : de l'unité) c'est-à-dire qui présentent une virgule après la partie entière peuvent également se représenter en binaire ou dans tout autre base. Pour cela il suffisait d'étendre la formule aux puissances négatives de la base. En binaire, ces extensions s'appellent les “décimales binaires” : la première décimale binaire est $\frac{1}{2}$, la seconde est $\frac{1}{4}$, la troisième est $\frac{1}{8}$ et ainsi de suite.

Dans certains processeurs, lorsqu'on utilise une représentation des nombres dite *en virgule fixe*, on restreint cette extension à une puissance donnée, afin que les opérations se fassent avec une position établie de la virgule fixée, une bonne fois pour toute, ce qui simplifie l'architecture physique de l'unité de calcul. La position de la virgule est alors connue et admise, au point où on peut l'oublier : cela revient à manipuler un simple entier. Par exemple, on peut associer le nombre 125 à une mesure de 1.250 m. Pour une virgule placée au rang k , la différence entre deux nombres consécutifs (appelée résolution) est égale 2^{-k} et sa dynamique (différence entre le nombre le plus grand et le plus petit) est réduite à 2^{n-k} , n étant le nombre de bits du nombre.

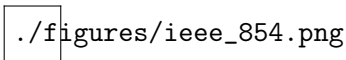
Le calcul en virgule fixe est idéal dans le cas de l'addition : la place de la virgule n'est pas appelée à changer dans le résultat. Ce n'est pas le cas de la multiplication : pour deux nombres ayant 1 chiffre après la virgule, on obtient un résultat qui possède 2 chiffres après la virgule. Il en est de même de 2 nombres de 2 chiffres chacun : le résultat est sur 4 chiffres, ce qui peut provoquer des débordements. Enfin, deux nombres de 2 digits présentant une virgule à l'extrême gauche donneront un résultat de multiplication à 4 chiffres après la virgule : il faudra tronquer le résultat... Bref : il est peut-être souhaitable de conserver une liberté dans la gestion des virgules...

2.10 Les nombres en virgule flottante

Le calcul scientifique nous amène à des formules qui mélangent différents ordres de grandeur, peu compatibles avec la représentation en virgule fixe. Les ordinateurs possèdent pour la plupart une unité en virgule flottante ; c'est désormais le cas de 100% des ordinateurs de bureau. Seuls certains processeurs embarqués n'en possèdent pas, car l'unité de calcul flottante est consommatrice de surface. La représentation en virgule flottante est normalisée par l'organisme IEEE, responsable de la standardisation de bon nombre de techniques, langages etc. Il s'agit ici de s'assurer que les manipulations flottantes d'un programme ne dépendront pas de l'architecture de la machine : Intel, AMD et. doivent fournir la même représentation. Il existe en réalité deux représentations normalisées IEEE 854 : simple et double précision. On ne s'attarde ici que sur la première représentation.

6. ceci se généralise dans tout autre base.

7. resp. à la base en question



La représentation des flottants selon la norme IEEE simple précision se fait à l’aide de mots de 32 bits, constitués :

- un bit de signe, dénoté S .
- 8 bits d’exposants (7 à 0), dénoté E
- 23 bits pour la partie fractionnaire, dénoté de 22 à 0, comme représenté sur la figure. E et F sont toujours représentés en base 2, non complémentés à 2.

2.11 Manipulations numériques

2.11.1 Récupération d’un champ

Revenons à la représentation de la position (x,y) . Bien entendu, lorsque l’on utilise le nombre particulier `Pos` précédent, il est nécessaire de connaître le mécanisme qui l’a créé, afin de récupérer une coordonnée x ou y :

$$y = (0x0000FF) \& \& Coord$$

Ce calcul permet d’extraire y du nombre `Coord`. C’est un ET binaire, bit à bit, entre `Coord` et une constante exprimée en headecimal. `0x0000FF` a été créé de manière à annuler la contribution des bits de x dans `Coord`. Pour cela les 4 digits hexadécimaux sont effectivement à 0 ; ils représentent 16 bits (4 fois 4 quartets) à 0. Quel que soit la valeur x , le ET bit à bit retournera des bits à 0. A l’inverse, les 2 derniers quartets de la constante sont à F et représentent donc 8 bits à 1. En conséquence, sur les 2 derniers quartets résultats, on retrouvera bien forcément la valeur de y , et elle seule, dans le résultat.

2.11.2 Autres manipulations au niveau bit

La section précédente nous a permis de récupérer un champ particulier d’une donnée composée, grâce à une manipulation de bits. Il existe en fait de nombreuses manipulations qu’il est utile de connaître, notamment dans le domaine de la programmation pour des systèmes embarqués : les micro-contrôleurs actionnent et agissent sur leur environnement en écrivant et lisant des données précises stockées dans des circuits spéciaux appelés *registres*. Ces registres sont à l’image de l’exemple de la position (x,y) , agglomérant plusieurs informations : par exemple, une alarme peut se manifester dans un bit de ce registre, à une position donnée (ex : le bit 7 du registre “alarmes” peut signifier “porte non-verrouillée”). Il est donc important d’accéder à chacun des bits ou champs entiers de ces registres.

```

1 a = a | 0x4; /* mise a 1 du bit 2 de la variable a*/
2 a |= 0x4; /* idem*/
3 b &= ~(0x4) /* mise a 0 du bit 2 */
4 b &= ~(1 << 2) /* idem, mais plus explicite*/
5 c ^= ~(1 << 5) /* inversion du bit 5*/
6 e >>= 2 /* division de e par 4 */

```

Listing 2.1 – Quelques manipulations de bits en C

2.12 Conclusion

Ce chapitre vous a permis de rentrer de plain-pied dans les représentations numériques, notamment avec la prise de connaissance de différents codes. Il existe d’autres codes que nous n’avons pas abordé, comme les codes en excès de 3 où le code de Aiken. Nous nous appuyerons sur ces premières connaissances afin de construire –bientôt– nos premières applications numériques.

Chapitre 3

Logique combinatoire

3.1 Définition

un système numérique est dit combinatoire si toutes ses sorties, à tout instant $t + \delta$, ne dépendent que de la combinaison de ses valeurs d'entrées à un instant t . Selon le niveau d'abstraction auquel on se situe, on pourra assimiler $\delta = 0$ ou non. L'électronique numérique est particulièrement intéressante car on peut se ramener à $\delta = 0$, car les raisonnements sur la fonction remplie par le système devient plus simple à analyser. On se situe alors à un niveau d'abstraction plus élevé : certains détails ont pu être volontairement omis lors de la conception, moyennant certaines hypothèses. Nous étudierons au chapitre suivant –dédié au second types de circuits, dits séquentiels– comment on peut toujours se ramener à $\delta = 0$...

Sous cette définition se cache en fait le pendant des systèmes combinatoires : les systèmes séquentiels, que nous étudierons au prochain chapitre. La définition proposée ici pour les systèmes combinatoires cherche en fait à dire : les *systèmes combinatoires n'ont pas de mémoire d'un état interne passé* (alors que c'est le cas pour les systèmes séquentiels, caractérisé par cet état). Malheureusement, les choses sont un peu plus compliquées, dans le détail : on peut en effet noter que, même dans un circuit combinatoire, il existe un effet mémoire lié aux différentes mini-capacités (condensateurs) qui sont susceptibles d'emmagasiner de l'énergie (sur les connexions etc). Toutefois, on sait que si l'on attend suffisamment longtemps, cet mémoire s'évanouit... Il faut donc retenir l'intuition des systèmes combinatoires : après avoir positionné des entrées, et lorsqu'on attend un temps suffisant (le δ de la définition), on verra un résultat en sortie, qui ne dépend que des entrées...

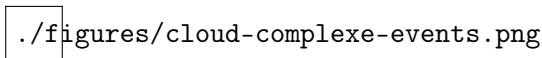



FIGURE 3.1 – Exemple de fonction combinatoire : les sorties ne dépendent que des entrées. Elles sont obtenues après un temps de propagation du calcul...

3.2 Portes logiques élémentaires

Il est temps de faire la connexion entre les éléments de l'algèbre de Boole et les circuits numériques qui vont les véhiculer. La table suivante nous les présente sous plusieurs formes : nom traditionnel, représentation standardisée IEC, représentation graphique et table de vérité. Il est à noter que la représentation standardisée – très inélégante – tombe en désuétude ! En effet, lorsqu'elle a été proposée, les ordinateurs ne disposaient que de peu de ressources graphiques pour supporter ce que les ingénieurs dessinaient traditionnellement (formes arrondies de la représentation graphique). Par la suite nous préférons donc la représentation graphique.


Si l'on observe cette table, le premier élément (porte YES¹) est surprenante : au niveau logique, elle ne fait rien (porte identité). Elle se retrouve toutefois sur des schémas électroniques, car elle possède une fonction importante de réhaussement des niveaux électroniques. En effet, lors de la propagation d'un signal électronique, un signal initialement à 5 Volts peut se dégrader et tomber dans un niveau dangereux où il s'approche de la limite de distinction entre le 1 logique et le 0.



./figures/table_verite.png

Composition de circuits numériques Le composition de circuits numériques est totalement simplifiée en électronique numérique, par rapport à l'analogique. C'est cette simplicité qui en fait son succès. Cet assemblage consiste pour l'essentiel à connecter les sorties d'une porte à l'entrée d'une autre porte.


Cet assemblage permet réaliser une fonction qui pourra être "encapsulée" dans un nouveau composant, placé dans une bibliothèque, et qui pourra être utilisé plus tard...On procède ainsi de manière hiérarchique. Ceci est illustré sur la figure suivante, qui fait référence aux fameuses poupées gigognes (ou poupées russes).



./figures/hierarchie.jpg

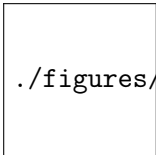
Le composant de plus haut niveau dans la hiérarchie est susceptible² de représenter un véritable produit commercialisable : on appelle ce composant un block IP (ou IP). IP signifie alors "intellectual property" : il représente un savoir faire numérique particulier. Pour votre information, il existe des startups ou des sociétés bien établies dont le métier est de délivrer ce genre de circuits : à titre d'exemple dans la région Bretonne, vous pouvez consulter le site de la société Turbo-Concept, qui fournit des IP correcteurs-d'erreurs pour les télécommunications.

Exemple de circuit "complexe" : l'additionneur 1 bit



./figures/half_adder.jpg

FIGURE 3.2 – Demi-additionneur 1 bit



./figures/full_adder.jpg

FIGURE 3.3 – Additionneur 1 bit composé de 2 demi-additionneurs et d'une porte OU

1. On appelle fréquemment cette porte un "buffer", mais cette appellation est malheureuse, car possède d'autres signification en électronique !

2. Ceci ne sera réellement réaliste qu'après le chapitre suivant.

3.3 Buffer trois-états

3.4 Notion de *netlist*

3.4.1 Combinaison de portes logiques

3.4.2 Chemins combinatoires et chemin critique

3.5 Opérateurs combinatoires usuels

3.5.1 Additionneur

3.5.2 Soustracteur

3.5.3 Additionneur-soustracteur

3.5.4 Multiplieur

3.5.5 Multiplexeur

3.5.6 Comparateur

3.5.7 Encodeur et décodeur

3.6 Opérateurs combinatoires avancés

3.6.1 Unité Arithmétique et logique

3.6.2 Vote majoritaire

3.6.3 Détection et corrections d'erreurs

3.7 Conclusion

Chapitre 4

Logique séquentielle

4.1 Introduction

4.2 Bascule D

4.3 RTL : notion de transfert de registres

4.3.1 Registre avec *enable*

4.3.2 Registre à décalage

4.3.3 Compteurs

4.4 Machine d'états finis

4.4.1 Définition

4.4.2 Exemple minimaliste

4.4.3 Diagrammes états-transitions

4.4.4 Table de vérité des FSM

4.5 Conclusion

Chapitre 5

Langages de description matériels

Exemple de VHDL

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris pulvinar consectetur congue. Vivamus facilisis lectus ac elementum vestibulum. Aenean vel nulla imperdiet, finibus nulla id, dignissim ipsum. Phasellus nunc tellus, scelerisque ac ante consequat, tincidunt elementum quam. Integer semper massa id ligula volutpat, quis efficitur urna iaculis. Mauris lacinia elementum eros eget mollis. Praesent vulputate enim hendrerit magna cursus semper viverra ut eros. Vivamus ac nibh ut dolor pharetra imperdiet. Praesent hendrerit felis eget lectus imperdiet aliquet. Sed mollis leo sit amet lectus dignissim sollicitudin. Interdum et malesuada fames ac ante ipsum primis in faucibus. Integer eu aliquet massa. Suspendisse potenti. Donec eu nulla dapibus, elementum justo et, suscipit augue. Sed in porta odio.

Sed semper, ante vitae iaculis commodo, est tortor gravida dui, pellentesque ornare mauris sapien pretium turpis. Quisque eu eros tempus ante varius tempus ut a turpis. Vestibulum sed elit dolor. Interdum et malesuada fames ac ante ipsum primis in faucibus. Mauris in finibus orci, eu consequat odio. Suspendisse non lacus non lorem fermentum consequat. In hac habitasse platea dictumst. Donec a dignissim erat. Nam semper venenatis mauris a congue.

Vestibulum vitae ipsum nisi. Maecenas tempus efficitur tortor, et feugiat quam fermentum a. Nam fringilla justo at metus placerat ultrices. Quisque consequat, diam quis volutpat facilisis, dui mi mattis orci, fermentum eleifend metus augue nec elit. Nunc non mauris eu sapien bibendum efficitur et ac turpis. Suspendisse a tincidunt dui. In ac mattis dui. Nunc ac varius nibh. Curabitur non augue orci. Maecenas metus enim, feugiat ut quam ut, consectetur imperdiet mi. Maecenas venenatis mi sollicitudin placerat tincidunt. Ut posuere justo mi, at scelerisque arcu hendrerit ac.

Pellentesque iaculis vehicula pharetra. Quisque imperdiet augue est, sed venenatis arcu cursus a. Vivamus dapibus hendrerit ipsum, vitae mattis nisi vulputate sed. Sed ut lectus sit amet nunc dignissim ultricies non nec lorem. Donec sit amet ante tincidunt, pharetra arcu non, viverra magna. Nunc at turpis felis. Integer non nisl vehicula, lacinia odio id, viverra mi. Nunc finibus gravida justo eu sodales. Sed faucibus pretium tellus eget interdum. Aenean accumsan quam nibh, ac vehicula nulla facilisis vel. Fusce auctor maximus dui, et consequat ligula bibendum vitae. Nulla imperdiet lacus quis eleifend luctus. Ut quis erat eu velit facilisis viverra. Cras posuere faucibus rhoncus.

5.1 Conception d'un système embarqué générique

5.2 Langages de description matériel

5.2.1 Exemple jouet : Tuzz

5.2.2 Simulateur à événement discret

5.3 VHDL

5.3.1 première vue d'ensemble

5.3.2 Librairies

5.3.3 Entité et architecture

5.3.4 Signaux et processus

5.4 Exemples de descriptions synthétisables en VHDL

5.4.1 Bascules D

5.4.2 Machines d'états finis

5.4.3 Datapath

5.5 Notion de banc de test virtuels *testbench*

5.5.1 Modèle de référence

5.5.2 Exemple de banc de test

5.5.3 Générateurs d'horloge et reset

5.6 Utilisation de GHDL simulateur VHDL open-source

5.7 Utilisation d'une carte FPGA

Chapitre 6

Microarchitectures FSMD

6.1 Séparation contrôle-données

6.2 Gabarits VHDL

6.3 Conclusion

Chapitre 7

Synthèse comportementale

Du logiciel au matériel

7.1 Langage séquentiel source

7.2 CDFG : control-data flow graphs

7.2.1 Notion de CFG

7.2.2 Principe de la traduction

7.3 Partage de ressource

7.4 Gabarits de traduction VHDL