Université Libre de Bruxelles

Synthèse

# Circuits Logiques et Numériques
# ELEC-H-305

*Auteur :*
Cédric Hannotier

Année 2015 - 2016

# Appel à contribution

### Synthèse OpenSource

Ce document est grandement inspiré de l'excellent cours donné par Dragomir Milojevic à l'EPB (École Polytechnique de Bruxelles), faculté de l'ULB (Université Libre de Bruxelles). Il est écrit par les auteurs susnommés avec l'aide de tous les autres étudiants et votre aide est la bienvenue ! En effet, il y a toujours moyen de l'améliorer surtout que si le cours change, la synthèse doit être changée en conséquence. On peut retrouver le code source à l'adresse suivante

`https://github.com/nenglebert/Syntheses`

Pour contribuer à cette synthèse, il vous suffira de créer un compte sur *Github.com*. De légères modifications (petites coquilles, orthographe, ...) peuvent directement être faites sur le site ! Vous avez vu une petite faute ? Si oui, la corriger de cette façon ne prendra que quelques secondes, une bonne raison de le faire !

Pour de plus longues modifications, il est intéressant de disposer des fichiers : il vous faudra pour cela installer LaTeX, mais aussi *git*. Si cela pose problème, nous sommes évidemment ouverts à des contributeurs envoyant leur changement par mail ou n'importe quel autre moyen.

Le lien donné ci-dessus contient aussi le `README` contient de plus amples informations, vous êtes invités à le lire si vous voulez faire avancer ce projet !

### Licence Creative Commons

Le contenu de ce document est sous la licence Creative Commons : *Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)*. Celle-ci vous autorise à l'exploiter pleinement, compte- tenu de trois choses :

1. *Attribution* ; si vous utilisez/modifiez ce document vous devez signaler le(s) nom(s) de(s) auteur(s).
2. *Non Commercial* ; interdiction de tirer un profit commercial de l'œuvre sans autorisation de l'auteur
3. *Share alike* ; partage de l'œuvre, avec obligation de rediffuser selon la même licence ou une licence similaire

Si vous voulez en savoir plus sur cette licence :

`http://creativecommons.org/licenses/by-nc-sa/4.0/`

**Merci !**

# Table des matières

# Chapitre 1

# Systèmes de numérotation

## 1.1 Représentation des nombres

Soit un nombre $N$ en base $r$, sa forme généralisé s'écrit :

$$N = \underbrace{\sum_{i=0}^{i=n} a_i r^i}_{\text{Partie entière}} + \underbrace{\sum_{j=1}^{i=m} b_j r^{-j}}_{\text{Partie fractionnaire}} \tag{1.1}$$

Les indexes $i, j$ s'appellent les poids. On distingue 2 parties :
- Partie entière : $n+1$ chiffres $(a_0, \dots, a_n)$
  - $i = n$ : bit de poids plus fort (Most Significant Bit (MSB))
  - $i = 0$ : bit de poids le plus faible (Least Significant Bit (LSB), donné par $\frac{N-a_0}{r}$)
- Partie fractionnaire : $m$ chiffres $(b_1, \dots, b_m)$

Parmi l'infinité de bases $r$ possibles, on en distingue 4 :

| Type | Base |
|---|---|
| Décimale | {0,1,2,3,4,5,6,7,8,9} |
| Binaire | {0,1} |
| Octal | {0,1,2,3,4,5,6,7} |
| Hexadécimal | {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F} |

Tableau 1.1 – Bases utiles

## 1.2 Conversions

De manière générale, pour passer d'une base $p$ à une base $q$, on fera $(N)_p \to (N)_{10} \to (N)_q$.

Pour passer de la base $p$ à la base 10, il suffit de réécrire le nombre sous sa forme générale et de calculer le résultat.
Pour passer de la base 10 à la base $q$, il faut séparer la partie entière de la fractionnaire :
- Partie entière : diviser par la base $q$, noter le reste de la division et répéter l'opération jusqu'à ce que le nombre soit plus petit que $q$. Une fois le résultat obtenu, le chiffre résultant se lit dans le sens **contraire** que celui du calcul. Exemple :

Sens de calcul

| | | |
|---|---|---|
| 245 | :8 | 5 |
| 30 | :8 | 6 |
| 3 | :8 | 3 |
| 0 | | |

Sens de lecture

— Partie fractionnaire : écrire la partie fractionnaire sous la forme $0.xxx...$, la multiplier par $q$, noter le chiffre avant le . du résultat, réinitialiser ce chiffre à 0 et répéter l'opération. Le nombre résultant se lit dans le **même** sens que celui du calcul. Exemple :

Sens de calcul

| | | | |
|---|---|---|---|
| .4375 | x5 | 2.1875 | 2 |
| .1875 | x5 | 0.9375 | 0 |
| .9375 | x5 | 4.6875 | 4 |
| .6875 | x5 | 3.4375 | 3 |
| .4375 | x5 | 2.1875 | 2 |
| .1875 | x5 | 0.9375 | 0 |

Sens de lecture

⚠Faire gaffe avec l'hexadécimal, ne pas oublier de remplacer les chiffres $> 9$ par les lettres correspondantes

Néanmoins, dans le cas des bases utiles (Tableau 1.1), on préfèrera passer par la base 2. En effet, il suffit de regrouper les chiffres par groupe de $x$ où $x$ vaut l'exposant entre 2 bases (exemple : $x = 4$ pour les bases $2 \leftrightarrow 16$). On regroupera les chiffres en partant de part et d'autre de la virgule.

<table>
<tr><th>Base</th><th colspan="3">Nombre</th></tr>
<tr><td>10</td><td colspan="3">245</td></tr>
<tr><td>2</td><td colspan="3">11110101</td></tr>
<tr><td>Par 3</td><td>11</td><td>110</td><td>101</td></tr>
<tr><td>8</td><td>3</td><td>6</td><td>5</td></tr>
<tr><td>Par 4</td><td></td><td>1111</td><td>0101</td></tr>
<tr><td>16</td><td></td><td>F</td><td>5</td></tr>
</table>

<table>
<tr><th>Base</th><th colspan="3">Nombre</th></tr>
<tr><td>16</td><td colspan="3">1F2</td></tr>
<tr><td>2</td><td colspan="3">0001|1111|0010</td></tr>
<tr><td>Par 3</td><td>111</td><td>110</td><td>010</td></tr>
<tr><td>8</td><td>7</td><td>6</td><td>2</td></tr>
</table>

FIGURE 1.1 – Conversion entre bases utiles

⚠Surtout ne pas oublier les chiffres manquants pour la partie fractionnaire sinon un $(110)_2$ se transformera en $((0)11)_2$

**Astuce** En binaire, lorsqu'un nombre est composé majoritairement de 1 et qu'on veut sa valeur, comme le nombre 111110010, comptez le nombre de chiffre $\Rightarrow x$ et faire $2^x - 1$ et soustraire $2^i$ pour chaque 0. Dans notre exemple :

$$(111110010)_2 = (?)_{10} \tag{1.2}$$
$$x = 9 \tag{1.3}$$
$$2^9 - 1 = (511)_{10} \tag{1.4}$$
$$511 - 2^0 - 2^2 - 2^3 = (498)_{10} \tag{1.5}$$
$$(111110010)_2 = (498)_{10} \tag{1.6}$$

## 1.3 Opération arithmétiques

Les additions, soustractions, multiplications et divisions se déroulent comme en base 10, si ce n'est qu'au lieu de reporter quand on arrive au-dessus de 9, on reporte quand on arrive au dessus de la base$-1$ (grosso modo). Voir TP 1

**Remarque** dans le cas de codage binaire sur 8 bits (chiffre maximum 255), si l'on fait $236 + 170$, nous obtenons un chiffre au-dessus de la limite pour 8 bits, nous aurons donc un $9^{\text{ème}}$ bit. Le résultat sera donc tronqué car codé sur 8 bits.Ce problème de débordement s'appelle l'*overflow*

## 1.4 Nombres négatifs

En binaire, il existe 3 modes de représentation pour les nombres négatifs :

1. Signe et Valeur Absolue (SVA)
2. Complément à la base ($C_1$)
3. Complément à 2 ($C_2$)

### 1.4.1 Signe et Valeur Absolue (SVA)

Par convention :
- 1 bit réservé pour le signe tel que :
  - 0 = positif
  - 1 = négatif
- le reste réservé pour la valeur absolue

Ainsi, sur un mode à 8 bits, on peut représenter des chiffres $\in [-127, 127]$
Pour faire des opérations arithmétiques avec cette notation, il faut :

1. Comparer les signes pour déterminer le signe des résultats
2. Comparer la magnitude des nombres pour déterminer le sens ($A < B \rightarrow B - A$, $A > B \rightarrow A - B$)

Niveau matériel, c'est galère…

### 1.4.2 Complément à la base

**Complément à 1 ($C_1$)**

Cette notation est valeur $\forall$ base. Le principe est de faire la soustraction comme une addition. Soit 2 nombre $A$ et $B$, en base $r$, codés sur $m$ chiffres

$$A - B = A + (-B) \tag{1.7}$$
$$= A + (r^m - B) = A + B' \tag{1.8}$$

$$\text{où} \quad \begin{array}{l} B' = compl\acute{e}ment\grave{a}labaser \\ B + B' = r^m \end{array}$$

Mais pour arriver à ça, il faut quand même faire la soustraction $r^m - B$. Réorganisons

$$B' = (r^m - B) = ((r^m - 1) - B) + 1 \tag{1.9}$$

$(r^m - 1) - B$ est un complément de chaque chiffre de $B$

$$(r^m - 1) - B = (\underbrace{(r-1)\ (r-1)}_{\triangle\text{Concaténation}} \dots\ (r-1) - (b_{m-1}\ b_{m-2} \dots\ b_0) \tag{1.10}$$

$$= ((r-1) - b_{m-1}((r-1)b_{m-2}) \dots ((r-1) - b_0) \tag{1.11}$$

$$= b'_{m-1}\ b'_{m-2} \dots\ b'_0 \tag{1.12}$$

Cette méthode est très pratique en **binaire**, chaque chiffre est simplement inversé ! Seul problème, nous avons 2 manière de représenter le 0 (sur 8 bits : 0000000 et 11111111). Les opérations arithmétique en $C_1$ sont faisable mais comporte parfois quelques cas particuliers.

#### Complément à 2 ($C_2$)

Au lieu d'utiliser le complément à 1 $B' = (r^m - 1) - B$, on utilise le complément à 2 $B' = (r^m - 1) - B + 1$. Grosso modo, on ajoute 1 au complément à 1. Contrairement à $C_1$ et SAV, on peut représenter les nombre $\in [-128, 127]$ et nous n'avons qu'une notation pour 0. La conséquence de tout ça est qu'il sera beaucoup plus facile de faire des opération arithmétique (soustraction $\Rightarrow$ sommation). Seul problème, l'overflow...

#### Méthode de conversion pour complément

Nous utiliserons la plupart du temps le complément à 2. Pour convertir un nombre en complément à 1 ou 2 rien de plus simple :

1. Prendre la valeur absolue du nombre en **base 2** et compléter avec des 0 pour avoir les $m$ bits demandés
2. Si le nombre est négatif :
   (a) Inverser chaque bit $\Rightarrow C_1$
   (b) $C_1 + 1 \Rightarrow C_2$

### 1.4.3 Overflow en $C_2$

une règle très simple permet de savoir si nous somme en overflow ou si le bit débordant peut être oublié sans risque

Si les 2 derniers bits du résultats sont **différent** (01 ou 10)$\Rightarrow$ **OK**
Si Si les 2 derniers bits du résultats sont **les mêmes** (11 ou 00) $\Rightarrow$ **Overflow**

## 1.5 Virgule flottante : Forme généralisée

Les nombres en virgule fixe à 32 ou 64 bits limitent fortement les calculs et augmenter le nombre de bit n'est pas une solution $\rightarrow$ Nombre en virgule flottante :

$$N = mantisse \times (base)^{exposant} \tag{1.13}$$

### 1.5.1 Standard IEEE 754

On distingue 2 type de précision :
— Simple précision : 32 bits

| Nombre de bits | 1 | 8 | 23 |
|---|---|---|---|
| Type | Signe de la mantisse | Exposant (0 à 255) | Fraction normalisée |
| Biais | | 127 | |

Tableau 1.2 – IEEE 754 - Simple Précision

— Double précision : 64 bits

| Nombre de bits | 1 | 11 | 52 |
|---|---|---|---|
| Type | Signe de la mantisse | Exposant (0 à 2047) | Fraction normalisée |
| Biais | | 1023 | |

Tableau 1.3 – IEEE 754 - Double Précision

Pour convertir un chiffre en virgule flottante il faut :

1. Convertir le nombre en binaire
2. Mettre sous forme 1.*abcd*... $(\times x)$
3. Déterminer le signe (0 ou 1, voir sous-section 1.4.1)
4. Calculer l'exposant : $E = x + biais$
5. Écrire la mantisse (*abcd*...)

⚠Le 1 de 1.*abcd*... n'est pas a écrire dans la mantisse

# Chapitre 2

# Codes et Algèbre de Boole

## Chapitre 3

# Expressions logiques

## Chapitre 4

# Simplification des fonctions logiques

# Chapitre 5

# Synthèse des systèmes combinatoires

# Chapitre 6

# Synthèse des systèmes séquentiels asynchrones

## Chapitre 7

# Synthèse des systèmes séquentiels synchrones