HAL
archives-ouvertes.fr

RN-codes : algorithmes d'addition, de multiplication et d'élévation au carré.

Jean-Luc Beuchat, Jean-Michel Muller

To cite this version:

Jean-Luc Beuchat, Jean-Michel Muller. RN-codes : algorithmes d'addition, de multiplication et d'élévation au carré.. [Research Report] Laboratoire de l'informatique du parallélisme. 2004, 2+15p. hal-02101864

HAL Id: hal-02101864

https://hal-lara.archives-ouvertes.fr/hal-02101864

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

RN-codes : algorithmes d'addition, de multiplication et d'élévation au carré

Jean-Luc Beuchat et Jean-Michel Muller

Décembre 2004

Research Report N° 2004–60

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

RN-codes : algorithmes d'addition, de multiplication et d'élévation au carré

Jean-Luc Beuchat et Jean-Michel Muller

Décembre 2004

Abstract

A property of the original Booth recoding is that the first non-zero digit following a 1 is necessarily -1 and vice versa. This allows to prove that truncating the Booth recoding of a number x is equivalent to rounding x to the nearest. P. Kornerup and J.-M. Muller investigated the positional, radix β, number systems sharing this rounding property and called them RN-codings. This research report is devoted to the study of addition, multiplication, and squaring algorithms for radix 2 RN-codings (i.e. Booth recodings). We show that integer arithmetic and logic units operations allow to add or multiply Booth recodings. We also describe algorithms taking advantage of the properties of the original Booth recoding to generate optimized hardware operators.

Keywords: Computer arithmetic, RN-codings, Booth recoding, rounding to the nearest

Résumé

Une propriété du recodage de Booth dans sa forme originale est que le premier chiffre non nul qui suit un 1 est forcément -1 et réciproquement. Il est facile de montrer que, dans un tel recodage, tronquer est équivalent à arrondir au plus près. P. Kornerup et J.-M. Muller ont récemment étudié divers systèmes de numération à chiffres signés partageant cette propriété et les ont appelés RN-codes. Dans cet article, nous nous intéressons à des algorithmes d'addition et de multiplication pour des RN-codes en base 2 (c'est-à-dire des recodages de Booth). Nous montrons qu'il est possible d'utiliser les unités arithmétiques des processeurs pour additionner ou multiplier les RN-codes considérés. Nous proposons également des algorithmes tirant parti des spécificités des RN-codes pour générer des opérateurs matériels.

Mots-clés: Arithmétique des ordinateurs, RN-codes, recodage de Booth, arrondi au plus près

1 Introduction

Nous nous intéressons à des cas particuliers de représentations à chiffres signés des nombres. Formalisées par Avizienis en 1961 [1], ces représentations apparaissaient déjà implicitement dans le recodage de Booth [3] ou dans les résultats fournis par les algorithmes de division SRT [8, 9]. Notons encore que les premières idées dans cette direction remontent à Cauchy [4].

Le recodage de Booth dans sa forme originale (à ne pas confondre avec des idées différentes, souvent regroupées sous le nom de *recodage de Booth modifié*, qui sont les seules utilisées de nos jours pour réaliser des multiplieurs) consiste à remplacer dans l'écriture binaire d'un nombre les sous-chaînes de la forme $011111\ldots11$ par $100000\ldots0\bar{1}$, où $\bar{1}$ désigne le chiffre -1. L'idée de Booth était d'essayer de faire apparaître le plus possible de zéros dans l'écriture d'un multiplicateur afin d'effectuer des multiplications plus rapidement et/ou avec moins de matériel.

Une propriété des recodages de Booth est que le premier chiffre non nul qui suit un 1 est forcément $\bar{1}$ et réciproquement. Il est relativement facile d'en déduire que le nombre obtenu en tronquant à partir de n'importe quelle position, disons celle de poids 2^p, les bits de poids faibles d'un tel recodage est à une distance du nombre initial inférieure à 2^{p-1} : *dans un tel recodage, tronquer est équivalent à arrondir au plus près.* Cette propriété est très intéressante : elle évite tous les problèmes dits de *double arrondi.* P. Kornerup et J.-M. Muller [6] ont récemment étudié les divers systèmes à chiffres signés qui partagent cette propriété, et les ont appelés RN-codes (où *RN* vient de *Round to Nearest*). Il ont proposé des algorithmes de conversion de nombres écrits dans ces systèmes vers et depuis les systèmes usuels. Ils ont entre autres obtenu le résultat suivant :

Théorème (Kornerup & Muller, 2004). *Soit β la base de numération utilisée.*

- *Si β est impair, alors $\mathcal{D} = d_n d_{n-1} d_{n-2} d_{n-3} \ldots d_0.d_{-1}d_{-2}\ldots$ est un RN-code si et seulement si*
$$\forall i, \frac{-\beta+1}{2} \leq d_i \leq \frac{\beta-1}{2}.$$
- *Si $\beta = 2$, alors $\mathcal{D} = d_n d_{n-1} d_{n-2} d_{n-3} \ldots d_0.d_{-1}d_{-2}\ldots$ (avec $d_i = -1, 0, 1$) est un RN-code si et seulement si les chiffres non nuls sont de signes alternés.*
- *Si β est pair et supérieur ou égal à 4, alors $\mathcal{D} = d_n d_{n-1} d_{n-2} d_{n-3} \ldots d_0.d_{-1}d_{-2}\ldots$ est un RN-code si et seulement si*
  1. *tous les chiffres sont de valeur absolue inférieure ou égale à $\beta/2$;*
  2. *si $|d_i| = \beta/2$, alors le premier chiffre non nul qui suit à droite de d_i est de signe opposé à celui de d_i.*

Par exemple, en base 2, $100\bar{1}1\bar{1}$ est un RN-code, tandis que $1000\bar{1}\bar{1}$, qui représente le même nombre ne l'est pas. En base 3, toutes les écritures ne comportant que les chiffres $\bar{1}$, 0 et 1 sont des RN-codes (et on peut représenter ainsi tous les nombres). En base 10, $3.142\bar{4}\bar{1}3\bar{3}\bar{5}4\bar{4}\bar{1}0\bar{2}\bar{1}3$ constitue le début de la représentation de π par un RN-code.

Dans la suite, nous ne considérons que des RN-codes en base 2 (c'est-à-dire des recodages de Booth). Notre but des de montrer qu'il est aisé de manipuler des nombres représentés dans ce système. Remarquons tout d'abord que la propriété des signes alternés des chiffres d'un RN-code binaire permet de les mémoriser de manière compacte (il suffit de mémoriser le signe du premier chiffre non-nul, puis ensuite de représenter les ± 1 par le bit 1, et les zéros par le bit 0). Cette représentation compacte est intéressante à utiliser pour un stockage en mémoire secondaire. Pour manipuler ces codes lorsque nous effectuons des opérations arithmétiques, nous représentons un nombre X de n chiffres à l'aide de deux vecteurs de n bits X^+ et X^- tels que $X = X^+ - X^-$. Contrairement à la représentation *borrow-save* [2], nous imposons une représentation unique du chiffre 0, à savoir $x_i^+ = x_i^- = 0$. Cette contrainte garantit que X^+ et X^- ne contiennent aucune chaîne de 1 de longueur supérieure à un.

Exemple 1. *Soit $n = 8$. Les nombres $100\bar{1}100\bar{1}$ et $1\bar{1}1\bar{1}1\bar{1}1\bar{1}$ s'écrivent respectivement*

$$100\bar{1}100\bar{1} = \underbrace{10001000}_{X^+} - \underbrace{00010001}_{X^-} \quad \textit{et} \quad 1\bar{1}1\bar{1}1\bar{1}1\bar{1} = \underbrace{10101010}_{X^+} - \underbrace{01010101}_{X^-}$$

Dans la suite de cet article, nous présentons des algorithmes d'addition (sections 2 et 3) et de multiplication (section 4). Nous proposons finalement quelques optimisations pour implanter l'élévation au carré (section 5).

2 Addition avec conversion en *borrow-save*

Le calcul d'un nombre T de $(n+1)$ chiffres en *borrow-save* résultant de l'addition de deux RN-codes de n chiffres constitue un cas particulier de l'algorithme d'addition en temps constant en *borrow-save* et permet une simplification de l'opérateur proposé dans [2]. Afin de déterminer les bits positifs de T, nous additionnons X^+ et Y^+. Le codage adopté nous garantit que si $x_i^+ = 1$ (respectivement $y_i^+ = 1$), alors $x_{i+1}^+ = 0$ (respectivement $y_{i+1}^+ = 0$). Par conséquent, si l'addition de x_i^+ et de y_i^+ génère une retenue (c.-à.-d. si $x_i^+ y_i^+ = 1$), cette dernière ne sera propagée que d'une position. Nous obtenons ainsi

$$\begin{aligned} t_0^+ &= x_0^+ y_0^+ \\ t_n^+ &= x_{n-1}^+ \oplus y_{n-1}^+ \\ t_{i+1}^+ &= (x_{i+1}^+ \oplus y_{i+1}^+) \vee x_i^+ y_i^+ \quad \forall i \in \{0, \ldots, n-1\} \end{aligned}$$

Le même principe s'applique évidemment au calcul de T^-. L'addition avec conversion en *borrow-save* s'effectue à l'aide de $2n$ cellules *half-adder* (ou HA) et de $2(n-1)$ portes logiques OR[1] (algorithme 1). Le chemin critique de l'opérateur contient ainsi une porte logique XOR et une porte logique OR (figure 1) et le calcul s'effectue, d'un point de vue théorique, en temps constant.

Algorithm 1 Addition avec conversion en *borrow-save*.

Entrées: Deux RN-codes X et Y de n chiffres
Sortie: Somme T de X et Y codée en *borrow-save* avec $(n+1)$ chiffres

1: Calculer en parallèle $U^+ \leftarrow X^+ \oplus Y^+$, $U^- \leftarrow X^- \oplus Y^-$, $V^+ \leftarrow X^+ \wedge Y^+$ et $V^- \leftarrow X^- \wedge Y^-$ à l'aide de $2n$ cellules *half-adder*;
2: Calculer en parallèle $T^+ \leftarrow v_{n-1}^+ 2^n + 2(V_{n-2:0}^+ \vee U_{n-1:1}^+) + u_0^+$ et $T^- \leftarrow v_{n-1}^- 2^n + 2(V_{n-2:0}^- \vee U_{n-1:1}^-) + u_0^-$ à l'aide de $2(n-1)$ portes logiques OU ;

3 Addition de deux RN-codes

L'algorithme étudié au paragraphe précédent constitue la première étape de l'addition de deux RN-codes de n chiffres X et Y. Nous convertissons ensuite la représentation *borrow-save* T de $X+Y$ en complément à deux à l'aide d'une soustraction, puis construisons le RN-code associé à ce résultat. T^- étant un mot de $(n+1)$ bits, $(n+2)$ bits sont nécessaires à l'écriture de $(-T^-)$ en complément à deux. Par conséquent, $D = T^+ - T^-$ est également un nombre de $(n+2)$ bits. Si nous appliquons le recodage de Booth à ce résultat, nous obtenons un RN-code comportant, dans certains cas, $(n+2)$ chiffres.

Exemple 2. *Soient* $X = \bar{1}000$ *et* $Y = \bar{1}000$ *deux RN-codes de* 4 *chiffres. L'algorithme 1 retourne* $T^+ = 00000$ *et* $T^- = 10000$. *Une fois* T^+ *et* $(-T^-)$ *codés en complément à deux, nous obtenons* $U = T^+ + (-T^-) = 000000 + 110000$. *Le recodage de Booth de* U *est un nombre* $S = \bar{1}0000$ *de* 5 *chiffres. Toutefois, lorsque* $X = Y = 1000$, *nous obtenons* $U = 010000 + 000000$ *et* $S = 1\bar{1}0000$. *Remarquons que ce résultat admet également une écriture sur* 5 *chiffres* $S = 10000$.

Montrons qu'il est toujours possible de coder $S = X + Y$ à l'aide de $(n+1)$ chiffres. Nous utilisons une propriété découlant immédiatement de la définition du RN-code considéré dans cet article.

[1] Nous considérerons toujours des portes logiques à deux entrées.

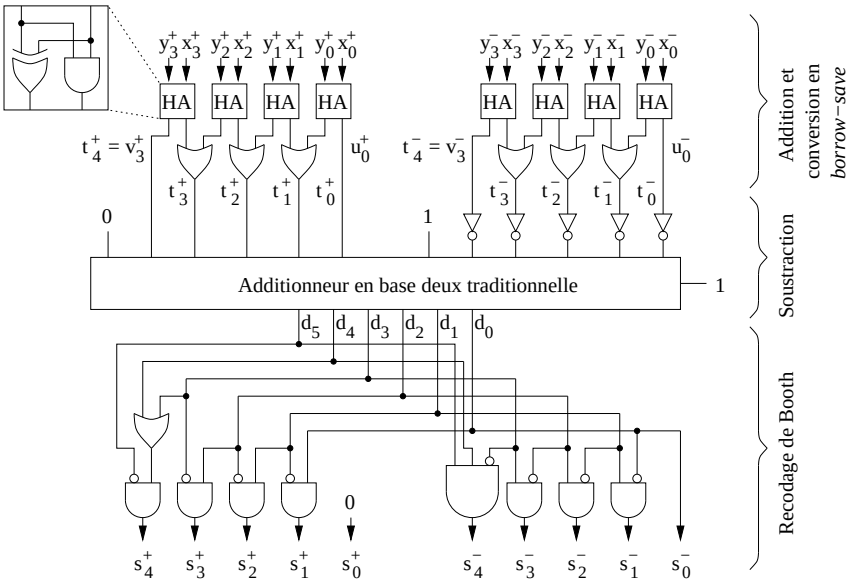


FIG. 1 – Addition de deux RN-codes de quatre chiffres.

Propriété 1. *Si X est un RN-code de n chiffres, alors $|X| \leq X_{max} = 2^{n-1}$.*

Le terme de $(n+2)$ bits $D = T^+ - T^-$ appartient ainsi à $\{-2^n, \ldots, 2^n\}$. Lorsque D est strictement négatif, nous déduisons de la définition du complément à deux que $d_{n+1} = d_n = 1$. Par conséquent, une fois le recodage de Booth effectué, $s_{n+1} = 0$ et tous les nombres de l'ensemble $\{-2^n, \ldots, -1\}$ se représentent par un RN-code de $(n+1)$ chiffres. Supposons maintenant que $D \in \{0, \ldots, 2^n - 1\}$. Comme $d_{n+1} = d_n = 0$, le recodage de Booth de D produit à nouveau un RN-code de $(n+1)$ chiffres. Le seul cas problématique survient par conséquent lorsque $D = 2^n$: appliqué à $01\underbrace{0\ldots0}_{n\times}$ le recodage de Booth génère le nombre de $(n+2)$ chiffres $1\bar{1}\underbrace{0\ldots0}_{n\times}$. Une modification des cellules de recodage de s_n^+ et s_n^- permet toutefois de représenter 2^n à l'aide du RN-code de $(n+1)$ chiffres $1\underbrace{0\ldots0}_{n\times}$:

$$
\begin{aligned}
s_n^+ &= \begin{cases} 1 & \text{si } d_{n+1} = 0 \text{ et } d_n = 1 \text{ (c.-à.-d. } D = 2^n) \\ d_{n-1} & \text{si } d_{n+1} = d_n = 0 \text{ (c.-à.-d. } D \in \{0, \ldots, 2^n - 1\}) \\ 0 & \text{sinon} \end{cases} \\
&= \bar{d}_{n+1}(d_n \vee d_{n-1}) \\
s_n^- &= \begin{cases} d_n \bar{d}_{n-1} & \text{si } d_{n+1} = 1 \text{ (c.-à.-d. } D \in \{-2^n, \ldots, -1\}) \\ 0 & \text{sinon} \end{cases} \\
&= d_{n+1} d_n \bar{d}_{n-1}
\end{aligned}
$$

Le recodage de Booth nécessite ainsi deux étages de portes logiques et s'effectue également, d'un point de vue théorique, en temps constant. Cet étage de calcul est constitué de $(2n+1)$ portes logiques AND, d'une porte logique OR et de $2n$ inverseurs. L'algorithme 2 résume les différentes étapes de l'addition de deux RN-codes et la figure 1 illustre l'architecture de l'opérateur correspondant. Il est constitué d'un additionneur traditionnel (additionneur à retenue propagée, additionneur préfixe,...) et de logique supplémentaire pour effectuer la conversion en *borrow-save*, déterminer le complément à deux de $-T^-$ et recoder le résultat. Dans le pire des cas, notre algorithme ajoute une porte XOR, une porte OR, deux portes AND et deux inverseurs au chemin critique de l'additionneur.

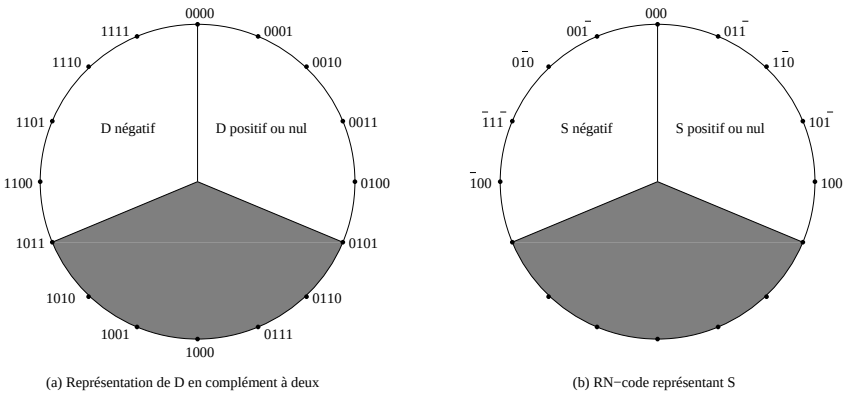
D'un point de vue logiciel, cet algorithme s'avère plus coûteux. La première étape de l'algorithme 1 nécessite à elle seule quatre instructions. Il est probablement plus judicieux de calculer $(X^+ - X^-) + (Y^+ - Y^-)$ à l'aide de deux soustractions et d'une addition, puis de recoder le résultat. Une autre solution serait d'étudier l'intégration des deux étages de logique additionnels dans un processeur commercial et de compléter son jeu d'instructions.

Algorithm 2 Addition de deux RN-codes de quatre chiffres.

Entrées: Deux RN-codes X et Y de n chiffres
Sortie: Un RN-code $S = X + Y$ de $(n+1)$ chiffres

1: Calcul de $X + Y$ en *borrow-save* (algorithme 1)
$T \leftarrow X + Y$;
2: Calcul de D en complément à deux $(n+2)$ bits)
$D \leftarrow T^+ - T^-$;
3: $s_0^+ \leftarrow 0$; $s_0^- \leftarrow d_0$;
4: **for** $i = 1$ à $n-1$ **do**
5: $\quad s_i^+ \leftarrow \bar{d}_i d_{i-1}$; $s_i^- \leftarrow d_i \bar{d}_{i-1}$;
6: **end for**
7: $s_n^+ \leftarrow \bar{d}_{n+1}(d_n \vee d_{n-1})$; $s_n^- \leftarrow d_{n+1} d_n \bar{d}_{n-1}$;

Exemple 3. *Considérons des RN-codes de deux chiffres afin d'illustrer le fonctionnement de l'algorithme d'addition proposé. La propriété 1 indique que D appartient à $\{-4, \ldots, 4\}$ (figure 2a). Tous les nombres de cet ensemble se représentent à l'aide d'un RN-code de 3 chiffres.*



(a) Représentation de D en complément à deux

(b) RN−code représentant S

Fig. 2 – Exemple d'addition de deux RN-codes de deux chiffres.

4 Multiplication de deux RN-codes

Afin de multiplier deux RN-codes X et Y, nous pouvons par exemple calculer

$$XY = (X^+ - X^-)(Y^+ - Y^-) = X^+Y^+ - X^+Y^- - X^-Y^+ + X^-Y^- \tag{1}$$

en complément à deux et convertir le résultat en un RN-code. La propriété 1 garantit que le produit appartient à $\{-2^{2n-2}, \ldots, 2^{2n-2}\}$ et nous utilisons la même astuce que ci-dessus afin de le coder sur $(2n-1)$ chiffres. Les produits $X^{\pm}Y^{\pm}$ et $X^{\pm}Y^{\mp}$ peuvent évidemment se calculer à l'aide de multiplieurs non signés. Plusieurs propriétés des RN-codes permettent toutefois des simplifications notables.

4.1 Calcul de $X^{\pm}Y^{\pm}$ et de $X^{\pm}Y^{\mp}$

Nous décrivons uniquement le calcul de $X^{\pm}Y^{\pm}$. Les propriétés et algorithmes proposés s'appliquent également à $X^{\pm}Y^{\mp}$.

Propriété 2. *Soit X un RN-code. Si $x_i^{\pm} = 1$, alors $x_{i+1}^{\pm} = x_{i-1}^{\pm} = 0$ et $x_i^{\mp} = 0$.*

Cette propriété se déduit immédiatement de la définition d'un RN-code. Appliquée au calcul de $X^{\pm}Y^{\pm}$, elle indique que si $x_i^{\pm}y_j^{\pm} = 1$, tous les termes de la forme $x_{i-1}^{\pm}y_k$, $x_{i+1}^{\pm}y_k$, $x_k^{\pm}y_{j-1}^{\pm}$ et $x_k^{\pm}y_{j+1}^{\pm}$ sont nuls, $\forall k \in \{0, \ldots, n-1\}$ (figure 3a). Elle permet également de réduire de moitié le nombre de produits partiels à additionner (figure 3c). En effet,

$$2^{i+j}(x_i^{\pm}y_j^{\pm} + x_{i+1}^{\pm}y_{j-1}^{\pm}) = 2^{i+j}(x_i^{\pm}y_j^{\pm} \vee x_{i+1}^{\pm}y_{j-1}^{\pm})$$

Nous souhaitons additionner les produits partiels en *carry-save* à l'aide de compresseur $(m, 2)$. Rappelons que ces opérateurs génèrent un chiffre de somme (c_j, s_j) et $(m-3)$ bits de retenue sortante $c_i^{(j)}$ à partir de m bits de même poids $\lambda_i^{(j)}$ et de $(m-3)$ bits de retenue entrante $c_i^{(j-1)}$ [10]

$$2\left(c_j + \sum_{i=0}^{m-4} c_i^{(j)}\right) + s_j = \sum_{i=0}^{m-1} \lambda_i^{(j)} + \sum_{i=0}^{m-4} c_i^{(j-1)}$$

Il est donc intéressant de regrouper tous les termes de même poids des produits partiels dans des vecteurs de bits $\Lambda^{(j)} = \lambda_{k-1}^{(j)} \ldots \lambda_0^{(j)}$ qui constitueront les entrées des compresseurs (algorithme 3 et figure 3c). Nous obtenons ainsi $(2n-1)$ mots de comportant de 1 à $\lceil n/2 \rceil$ bits.

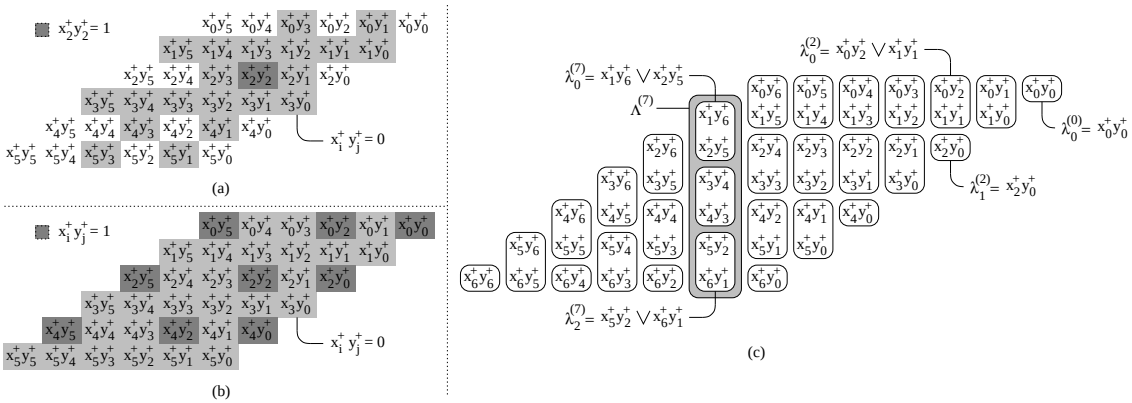


FIG. 3 – Application de la propriété 2. (a) Si $x_2^+y_2^+ = 1$, tous les termes de la forme $x_1^+y_k$, $x_3^+y_k$, $x_k^+y_1^+$ et $x_k^+y_3^+$ sont nuls. (b) Multiplication de $Y^+ = (100101)_2$ par $X^+ = (010101)_2$. (c) Regroupement des termes $x_i^+y_j^+$ et $x_{i+1}^+y_{j-1}^+$.

Dans la suite, nous dénotons le chiffre de poids 2^i de la somme des produits partiels par (c_i, s_i). Les propriétés des RN-codes permettent plusieurs simplifications de l'additionneur *carry-save* intervenant dans le calcul de $X^{\pm}Y^{\pm}$ (figure 4) :

- Le chiffre de poids 2^1 du produit est égal à $\Lambda^{(1)} = \lambda_0^{(1)}$.
- La propriété 2 garantit que si $\Lambda^{(2)} = (11)_2$, alors $\Lambda^{(3)} = (00)_2$. Une retenue générée dans la colonne de poids 2^2 est ainsi propagée d'une seule position et le bit de somme s_3 est défini par

$$s_3 = (\lambda_0^{(3)} \oplus \lambda_1^{(3)}) \vee \lambda_0^{(2)}\lambda_1^{(2)}$$

- Nous montrons de même que

$$s_4 = (\lambda_0^{(4)} \oplus \lambda_1^{(4)} \oplus \lambda_2^{(4)}) \vee \lambda_0^{(3)}\lambda_1^{(3)}$$

Algorithm 3 Génération des produits partiels pour le calcul de $X^{\pm}Y^{\pm}$.

Entrées: Deux RN-codes X et Y de n chiffres
Sortie: $(2n-1)$ termes $\Lambda^{(j)} = \lambda_{k-1}^{(j)} \dots \lambda_0^{(j)}$

1: $\lambda_0^{(0)} \leftarrow x_0^{\pm} y_0^{\pm}$; $\lambda_0^{(2n-2)} \leftarrow x_{n-1}^{\pm} y_{n-1}^{\pm}$;
2: **for** $i = 0$ à $\lfloor \frac{n}{2} - 1 \rfloor$ **do**
3: **for** $j = 0$ à i **do**
4: $\lambda_j^{(2i+1)} \leftarrow x_{2j}^{\pm} y_{2i+1-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j}^{\pm}$;
5: **end for**
6: **end for**
7: **for** $i = 1$ à $\lceil \frac{n}{2} - 1 \rceil$ **do**
8: **for** $j = 0$ à $i - 1$ **do**
9: $\lambda_j^{(2i)} \leftarrow x_{2j}^{\pm} y_{2i-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j-1}^{\pm}$;
10: **end for**
11: $\lambda_i^{(2i)} \leftarrow x_{2i}^{\pm} y_0^{\pm}$;
12: **end for**
13: **for** $i = \lfloor \frac{n+1}{2} \rfloor$ à $n - 2$ **do**
14: **for** $j = 0$ à $n - i - 2$ **do**
15: $\lambda_j^{(2i)} \leftarrow x_{2i+2j-n+1}^{\pm} y_{n-2j-1}^{\pm} \vee x_{2i+2j-n+2}^{\pm} y_{n-2j-2}^{\pm}$;
16: **end for**
17: $\lambda_{n-i-1}^{(2i)} \leftarrow x_{n-1}^{\pm} y_{2i-n+1}^{\pm}$;
18: **end for**
19: **for** $i = \lceil \frac{n-1}{2} \rceil$ à $n - 2$ **do**
20: **for** $j = 0$ à $n - i - 2$ **do**
21: $\lambda_j^{(2i+1)} \leftarrow x_{2i+2j-n+2}^{\pm} y_{n-2j-1}^{\pm} \vee x_{2i+2j-n+3}^{\pm} y_{n-2j-2}^{\pm}$;
22: **end for**
23: **end for**

Par conséquent, $c_i = 0, \forall i \in \{0, \dots, 4\}$. Le calcul du chiffre de poids fort de $X^{\pm}Y^{\pm}$ est plus subtil et exploite la propriété suivante :

Propriété 3. *Si X est un RN-code de n chiffres, alors*

$$|X^{\pm}| \leq X_{max}^{\pm} = \begin{cases} \dfrac{2(2^n - 1)}{3} & \textit{si } n \textit{ est pair} \\ \dfrac{2(2^n - 1) + 1}{3} & \textit{sinon} \end{cases}$$

Cette propriété nous permet de montrer qu'aucune propagation de retenue ne provient de la colonne de poids 2^{2n-3}. De plus, $s_{2n-2} = \lambda_0^{(2n-2)} = x_{n-1}^{\pm} y_{n-1}^{\pm}$ et $c_{2n-2} = 0$.

- Si $x_{n-1}^{\pm} = y_{n-1}^{\pm} = 0$, nous considérons $X^{\pm}$ et $Y^{\pm}$ comme des nombres de $(n-1)$ bits et obtenons

$$X^{\pm}Y^{\pm} \leq \begin{cases} \left(\dfrac{2(2^{n-1} - 1)}{3}\right)^2 = \dfrac{2^{2n} - 2^{n+2} + 4}{9} < 2^{2n-3} & \text{si } n \text{ est pair} \\ \left(\dfrac{2(2^{n-1} - 1) + 1}{3}\right)^2 = \dfrac{2^{2n} - 2^{n+1} + 1}{9} < 2^{2n-3} & \text{sinon} \end{cases}$$

  Le produit $X^{\pm}Y^{\pm}$ est ainsi un nombre de $(2n-3)$ bits et aucune retenue ne peut être générée ou propagée dans la colonne de poids 2^{2n-3}.
- Si $x_{n-1}^{\pm} \neq y_{n-1}^{\pm} = 1$, nous considérons un opérande comme un nombre de n bits et l'autre comme un nombre de $(n-1)$ bits. Ainsi,

$$X^{\pm}Y^{\pm} \leq \begin{cases} \dfrac{2(2^n - 1)}{3} \cdot \dfrac{2(2^{n-1} - 1)}{3} = \dfrac{2^{2n+1} - 2^{n+2} - 2^{n+1} + 4}{9} < 2^{2n-2} & \text{si } n \text{ est pair} \\ \dfrac{2(2^n - 1) + 1}{3} \cdot \dfrac{2(2^{n-1} - 1) + 1}{3} = \dfrac{2^{2n+1} - 2^{n+1} - 2^n + 1}{9} < 2^{2n-2} & \text{sinon} \end{cases}$$

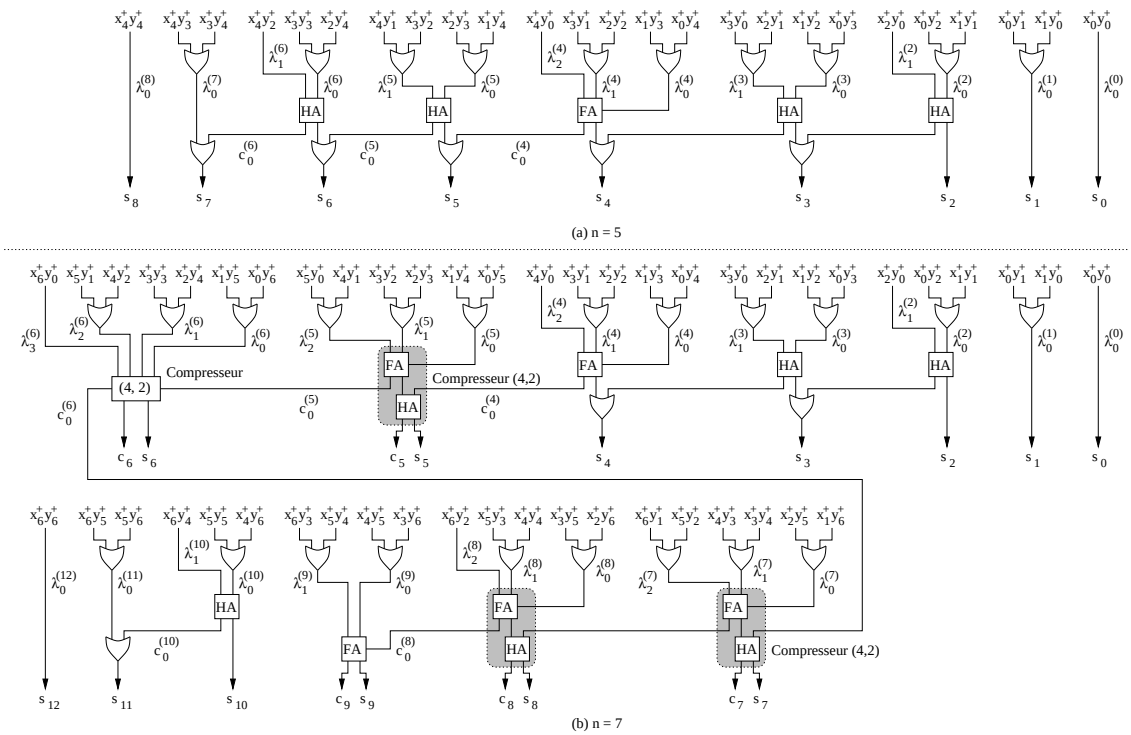


FIG. 4 – Calcul de X^+Y^+ lorsque (a) $n = 5$ et (b) $n = 7$.

Le produit $X^{\pm}Y^{\pm}$ étant un nombre de $(2n-2)$ bits, aucune retenue ne peut être générée ou propagée dans la colonne de poids 2^{2n-3}.

– Finalement, si $x^{\pm}_{n-1} = y^{\pm}_{n-1} = 1$, nous déduisons de la propriété 3 que $X^{\pm}Y^{\pm} < 2^{2n-1}$. Le produit est ainsi un nombre de $(2n-1)$ bits. Comme $\lambda_0^{(2n-2)} = x^{\pm}_{n-1}y^{\pm}_{n-1} = 1$, aucune propagation de retenue ne peut provenir de la colonne de poids 2^{2n-3}.

Lorsque $n = 5$ (figure 4a), la propriété 2 nous garantit qu'aucune retenue n'est propagée de plus d'une position[2]. Nous obtenons ainsi

$$s_5 = (\lambda_0^{(5)} \oplus \lambda_1^{(5)}) \vee (\lambda_0^{(4)}\lambda_1^{(4)} \vee \lambda_0^{(4)}\lambda_2^{(4)} \vee \lambda_1^{(4)}\lambda_2^{(4)}) \quad s_7 = \lambda_0^{(7)} \vee \lambda_0^{(6)}\lambda_1^{(6)}$$
$$s_6 = (\lambda_0^{(6)} \oplus \lambda_1^{(6)}) \vee \lambda_0^{(5)}\lambda_1^{(5)} \quad s_8 = \lambda_0^{(8)}$$

et $c_i = 0$, $\forall i \in \{0, \ldots, 8\}$. Dès que $n \geq 6$, des retenues peuvent par contre se propager de plusieurs positions et nous effectuons les calculs en *carry-save*. La difficulté consiste à dimensionner correctement les compresseurs $(m, 2)$.

Exemple 4. *Soient $n = 6$, $X^+ = (010101)_2$ et $Y^+ = (100101)_2$ (figure 3b). L'addition de $x_2^+y_2^+$ et de $x_4^+y_0^+$ provoque une retenue qui est propagée jusqu'à la colonne de poids 2^8 car $x_0^+y_5^+ = x_4^+y_2^+ = x_2^+y_5^+ = 1$.*

Lorsque $n \geq 6$, le calcul de s_5 et de c_5 requiert les trois termes $\lambda_0^{(5)}$, $\lambda_1^{(5)}$ et $\lambda_2^{(5)}$, ainsi que la retenue $c_4^{(0)}$ générée dans la colonne de poids 2^4. Nous utilisons ainsi un compresseur $(4, 2)$ dont l'une des entrées est toujours égale à zéro (figure 4b). Chacun des mots $\Lambda^{(i)}$, $6 \leq i \leq n-1$ comportant $m_i = \lceil \frac{i+1}{2} \rceil$ bits, nous calculons les couples (s_i, c_i) à l'aide de compresseurs $(m_i, 2)$. La seule difficulté consiste à déterminer le nombre de bits de retenue propagée de la colonne de poids 2^{i-1} :

[2]Ce résultat reste évidemment vrai pour $n < 5$.

- Si $i = 6$, nous recevons un unique bit de retenue $c_0^{(5)}$.
- Dans tous les autres cas, nous savons que la colonne de poids 2^{i-1} comporte un compresseur $(\lceil i/2 \rceil, 2)$ qui génère $(\lceil i/2 \rceil - 3)$ bits $c_j^{(i-1)}$. Ainsi, lorsque i est pair, un bit de retenue entrante de la colonne de poids 2^i est égal à zéro. Supposons par exemple $i = 8$. La colonne de poids 2^7 nécessite un compresseur $(4, 2)$ générant un seul bit de retenue $c_0^{(7)}$. La colonne de poids 2^8 comporte un compresseur $(5, 2)$ additionnant cinq bits de poids 2^8 et deux bits de retenue entrante. L'un deux est nécessairement égal à zéro, permettant ainsi une petite simplification du compresseur $(5, 2)$ (remplacement d'une cellule FA par une cellule HA).

Si n est pair, $\Lambda^{(n)}$ est un mot de $n/2$ bits. Comme la colonne de poids 2^{n-1} contient un compresseur $(n/2, 2)$ et génère $(n/2-3)$ bits de retenue, nous plaçons également un compresseur $(n/2, 2)$ dans la colonne de poids 2^n. Les vecteurs $\Lambda^{(n+1)}$ et $\Lambda^{(n+2)}$ comportant chacun $(n/2-1)$ bits, il faut encore placer des compresseurs $(n/2, 2)$ (dont l'une des entrées est fixée à zéro) dans les colonnes de poids 2^{n+1} et 2^{n+2}. Remarquons maintenant que $\Lambda^{(n+2i+1)}$ et $\Lambda^{(n+2i+2)}$ sont des mots de $(n/2-i-1)$ bits. Il est facile de montrer que la somme des $\lambda_j^{(n+2i+1)}$ s'effectue à l'aide d'un compresseur $(n/2-i, 2)$. Considérons la colonne de poids $(n+2i+1)$. Celle-ci reçoit $(n/2-i-2)$ bits de retenue[3] et contient $(n/2-i-1)$ bits $\lambda_j^{(n+2i+1)}$. Rappelons qu'un compresseur $(n/2-i, 2)$ comporte $(n-2i-3)$ entrées. Il permet donc le traitement des $(n/2-i-2)+(n/2-i-1) = (n/2-2i-3)$ bits de poids $(n+2i+1)$. Par conséquent, la somme des $\lambda_j^{(n+2i+2)}$ se calcule également avec un compresseur $(n/2 - i, 2)$. Nous générons ainsi des groupes de deux compresseurs $(m_i, 2)$, où $m_i = n/2 - i$, et arrêtons le processus lorsque $m_i = 4$, c'est-à-dire $i = n/2 - 4$. La colonne de poids 2^{2n-6} contient donc un compresseurs $(4, 2)$ générant un bit de retenue $c_0^{(2n-6)}$ que nous additionnons à $\lambda_0^{(2n-5)}$ et $\lambda_1^{(2n-5)}$ à l'aide d'une cellule FA

$$2c_5 + s_5 = \lambda_0^{(2n-5)} + \lambda_1^{(2n-5)} + c_0^{(2n-6)}$$

Ainsi, aucune retenue n'est propagée à la colonne de poids 2^{2n-4}. Un raisonnement analogue permet l'allocation des compresseurs lorsque n est impair. L'algorithme 4 résume ces considérations sur l'addition des produits partiels. Au terme de la première boucle générant des compresseurs, nous stockons dans la variable k le nombre de bits de retenue $c_i^{(n-1)}$ générés dans la colonne de poids 2^{n-1}. Dans le cas général, ce nombre est simplement égal à $\lceil n/2 \rceil - 3$. Toutefois, cette formule est incorrecte lorsque $n = 6$. Nous avons en effet calculé un bit de retenue sortante $c_0^{(5)}$ dans la colonne de poids 2^5. C'est pourquoi nous définissons $k = \max(\lceil \frac{n}{2} \rceil - 3, 1)$. A chaque itération de la seconde boucle, il est nécessaire de dimensionner le compresseur $(m, 2)$. Notons que le compteur i est initialisé à n, puis décrémenté. Etant donné un indice i, il alors est facile de montrer que

$$m = \min\left(\left\lceil \frac{n}{2} \right\rceil, \left\lceil \frac{i+1}{2} \right\rceil\right) = \begin{cases} n/2 & \text{si } n \text{ est pair et } i = n \\ \lceil \frac{i+1}{2} \rceil & \text{sinon} \end{cases} \tag{2}$$

L'expression ci-dessus est toutefois erronée si $n = 6$. Lors de l'unique passage dans la boucle, il faut en effet instancier un compresseur $(4, 2)$. En intégrant cette information dans l'équation (2), nous obtenons finalement

$$m = \max\left(4, \min\left(\left\lceil \frac{n}{2} \right\rceil, \left\lceil \frac{i+1}{2} \right\rceil\right)\right)$$

4.2 Premier algorithme de multiplication

Une fois les quatre produits X^+Y^+, X^-Y^-, X^+Y^- et X^-Y^+ calculés, le résultat s'obtient conformément à l'équation (1). Nous additionnons en *carry-save* X^+Y^+ et X^-Y^- ainsi que et X^+Y^- et X^-Y^+ (figure 5a) en tenant compte du fait que

$$\begin{aligned} x_0^+y_0^+ + x_0^-y_0^- &= x_0^+y_0^+ \vee x_0^-y_0^- \qquad & x_{n-1}^+y_{n-1}^+ + x_{n-1}^-y_{n-1}^- &= x_{n-1}^+y_{n-1}^+ \vee x_{n-1}^-y_{n-1}^- \\ x_0^+y_0^- + x_0^-y_0^+ &= x_0^+y_0^- \vee x_0^-y_0^+ \qquad & x_{n-1}^+y_{n-1}^- + x_{n-1}^-y_{n-1}^+ &= x_{n-1}^+y_{n-1}^- \vee x_{n-1}^-y_{n-1}^+ \end{aligned}$$

[3] Par hypothèse, la colonne de poids $(n + 2i)$ comporte un compresseur $(n/2 - i + 1, 2)$.

Algorithm 4 Calcul de $X^{\pm}Y^{\pm}$ lorsque $n \geq 6$.

Entrées: Deux RN-codes X et Y de n chiffres avec $n \geq 7$

Sortie: Produit $X^{\pm}Y^{\pm}$ de $(2n-1)$ chiffres représenté en *carry-save*

1: Calculer les termes $\Lambda^{(i)}$ à l'aide l'algorithme 3 ;

2: Calculer en parallèle

$$c_i \leftarrow 0, \forall i \in \{0,1,2,3,4,2n-4,2n-3,2n-2\};$$

$$c_0^{(4)} \leftarrow \lambda_0^{(4)}\lambda_1^{(4)} \vee \lambda_0^{(4)}\lambda_2^{(4)} \vee \lambda_1^{(4)}\lambda_2^{(4)}; \qquad c_0^{(5)} \leftarrow \lambda_0^{(5)}\lambda_1^{(5)} \vee \lambda_0^{(5)}\lambda_2^{(5)} \vee \lambda_1^{(5)}\lambda_2^{(5)};$$

3: Calculer en parallèle

$$\begin{aligned}
s_0 &\leftarrow \lambda_0^{(0)}; & s_5 &\leftarrow \lambda_0^{(5)} \oplus \lambda_1^{(5)} \oplus \lambda_2^{(5)} \oplus c_0^{(4)};\\
s_1 &\leftarrow \lambda_0^{(1)} \vee \lambda_1^{(1)}; & c_5 &\leftarrow (\lambda_0^{(5)} \oplus \lambda_1^{(5)} \oplus \lambda_2^{(5)})c_0^{(4)};\\
s_2 &\leftarrow (\lambda_0^{(2)} \vee \lambda_1^{(2)}) \oplus \lambda_2^{(2)}; & s_{2n-4} &\leftarrow \lambda_0^{(2n-4)} \oplus \lambda_1^{(2n-4)};\\
s_3 &\leftarrow (\lambda_0^{(3)} \oplus \lambda_1^{(3)}) \vee \lambda_0^{(2)}\lambda_1^{(2)}; & s_{2n-3} &\leftarrow \lambda_0^{(2n-3)} \vee \lambda_0^{(2n-4)}\lambda_1^{(2n-4)};\\
s_4 &\leftarrow (\lambda_0^{(4)} \oplus \lambda_1^{(4)} \oplus \lambda_2^{(4)}) \vee \lambda_0^{(3)}\lambda_1^{(3)}; & s_{2n-2} &\leftarrow \lambda_0^{(2n-2)};
\end{aligned}$$

4: **for** $i = 6$ à $n-1$ **do**

5: Calculer à l'aide d'un compresseur $(\lceil \frac{i+1}{2} \rceil, 2)$

$$2\left(c_i + \sum_{j=0}^{\lceil \frac{i+1}{2} \rceil - 4} c_j^{(i)}\right) + s_i = \sum_{j=0}^{\lceil \frac{i+1}{2} \rceil - 1} \lambda_j^{(i)} + \sum_{j=0}^{k} c_j^{(i-1)}, \quad \text{où } k = \max(0, \lceil i/2 \rceil - 4)$$

6: **end for**

7: $k \leftarrow \max(\lceil \frac{n}{2} \rceil - 3, 1)$; $r \leftarrow n$;

8: **for** $i = n$ à 6 **do**

9: $m \leftarrow \max(4, \min(\lceil \frac{n}{2} \rceil, \lceil \frac{i+1}{2} \rceil))$;

10: Calculer à l'aide d'un compresseur $(m, 2)$

$$2\left(c_r + \sum_{j=0}^{m-4} c_j^{(r)}\right) + s_r = \sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} \lambda_j^{(r)} + \sum_{j=0}^{k-1} c_j^{(r-1)}$$

11: $k \leftarrow m-3$; $r \leftarrow r+1$;

12: **end for**

13: $s_{2n-5} \leftarrow \lambda_0^{(2n-5)} \oplus \lambda_1^{(2n-5)} \oplus c_0^{(2n-6)}$; $c_{2n-5} \leftarrow \lambda_0^{(2n-5)}\lambda_1^{(2n-5)} \vee \lambda_0^{(2n-5)}c_0^{(2n-6)} \vee \lambda_1^{(2n-5)}c_0^{(2n-6)}$;

Nous obtenons ainsi quatre vecteurs de $(2n-1)$ bits $U^{(s)}$, $U^{(c)}$, $V^{(s)}$ et $V^{(c)}$ tels que

$$2U^{(c)} + U^{(s)} = X^+Y^+ + X^-Y^- \quad \text{et} \quad 2V^{(c)} + V^{(s)} = X^+Y^- + X^-Y^+$$

où $u_0^{(c)} = v_0^{(c)} = u_{2n-2}^{(c)} = v_{2n-2}^{(c)} = 0$. Nous effectuons ensuite une soustraction afin de calculer $2P^{(c)} + P^{(s)} = XY$ en *carry-save*. Les représentations complément à deux de $-V^{(s)}$ et de $-2V^{(c)}$ sont respectivement définies par

$$2^{2n-1} - V^{(s)} = 1 + \sum_{i=0}^{2n-2} \left(1 - v_i^{(s)}\right) 2^i = 1 + \sum_{i=0}^{2n-2} \bar{v}_i^{(s)}$$

et

$$2^{2n-1} - 2V^{(c)} = 1 + \sum_{i=0}^{1} 2^i + \sum_{i=2}^{2n-2} \left(1 - v_{i-1}^{(c)}\right) 2^i = 4 + \sum_{i=2}^{2n-2} \bar{v}_{i-1}^{(c)} 2^i$$

Par conséquent,

$$\begin{aligned} 2P^{(c)} + P^{(s)} &= 2U^{(c)} + U^{(s)} - 2V^{(c)} - V^{(s)} \\ &= 4 + \sum_{i=2}^{2n-2} \left(u_i^{(s)} + u_{i-1}^{(c)} + \bar{v}_i^{(s)} + \bar{v}_{i-1}^{(c)}\right) 2^i + 1 + \sum_{i=0}^{1} \left(u_i^{(s)} + \bar{v}_i^{(s)}\right) 2^i \end{aligned} \tag{3}$$

La première somme de l'équation (3) se calcule avec $(2n-3)$ compresseurs $(4,2)$ (figure 5b). En fixant la retenue entrante du premier compresseur à 1, nous additionnons automatiquement la constante 4. Une cellule HA et une cellule FA calculent respectivement $2p_1^{(c)} + p_1^{(s)} = u_1^{(s)} + \bar{v}_1^{(s)}$ et $2p_0^{(c)} + p_0^{(s)} = u_0^{(s)} + \bar{v}_0^{(s)} + 1$. Finalement, nous convertissons ce résultat en complément à deux à l'aide d'un additionneur traditionnel et effectuons le recodage de Booth selon la méthode décrite dans la section 3 (figure 5c).

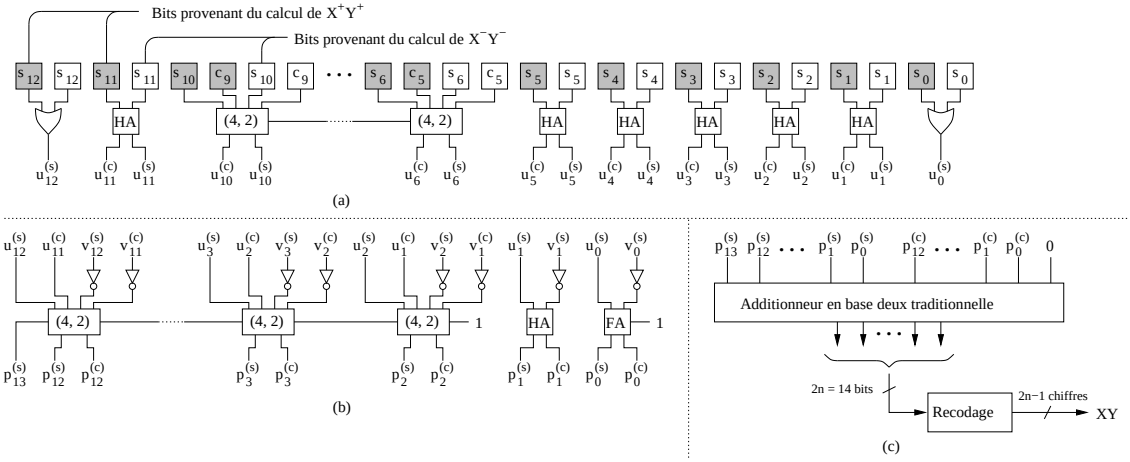


FIG. 5 – Produit de deux RN-codes de sept chiffres. (a) Calcul de $X^+Y^+ + X^-Y^-$. (b) Calcul de $P = (X^+Y^+ + X^-Y^-) - (X^+Y^- + X^-Y^+)$ en *carry-save*. (c) Conversion de P en complément à deux et recodage du résultat.

4.3 Second algorithme de multiplication

Une propriété évidente des RN-codes permet de remplacer les deux multiplieurs particuliers calculant X^+Y^+ et X^-Y^- par un multiplieur non signé traditionnel :

Propriété 4. *Si X un RN-code, alors*

$$X^{\pm} + X^{\mp} = X^{\pm} \vee X^{\mp}$$

Nous calculons ainsi XY en complément à deux à l'aide de trois multiplications et de deux soustractions :

$$\begin{aligned} XY &= (X^+ - X^-)(Y^+ - Y^-) = (X^+ + X^-)(Y^+ + Y^-) - 2X^+Y^- - 2X^-Y^+ \\ &= (X^+ \vee X^-)(Y^+ \vee Y^-) - 2X^+Y^- - 2X^-Y^+ \end{aligned}$$

Notons que cet algorithme est analogue à la méthode proposée par Karatsuba pour multiplier de grands entiers (voir par exemple [5]).

4.4 Troisième algorithme de multiplication

Les algorithmes étudiés ci-dessus minimisent les ressources nécessaires à la réalisation matérielle d'un multiplieur pour RN-codes. Toutefois, dans le cas d'une implantation logicielle ou sur un circuit FPGA (*Field Programmable Gate Array*) disposant de multiplieurs câblés, il semble plus intéressant de calculer $X = X^+ - X^-$ et $Y = Y^+ - Y^-$ en complément à deux à l'aide de deux soustracteurs, d'effectuer une multiplication signée et de recoder le résultat.

5 Quelques considérations relatives à l'élévation au carré

Bien que l'élévation au carré s'effectue aisément à l'aide des multiplieurs étudiés ci-dessus, la conception d'un algorithme spécifique à cette opération permet la réalisation d'un circuit nécessitant moins de ressources matérielles. Nous nous proposons de calculer

$$X^2 = (X^+ - X^-)^2 = (X^+)^2 - 2X^+X^- + (X^-)^2$$

ou

$$X^2 = (X^+ \vee X^-)^2 - 4X^+X^-$$

en tirant parti de la propriété suivante :

Propriété 5. *Soit X un RN-code de n chiffres.*
- *les produits $X^{\pm}X^{\pm}$ et X^+X^- sont respectivement des nombres de $(2n-1)$ et $(2n-2)$ bits ;*
- *$x_i^{\pm}x_{i+1}^{\pm} = 0$, $x_i^{\pm}x_i^{\mp} = 0$ et $x_i^{\pm}x_j^{\mp} + x_j^{\pm}x_i^{\mp} = x_i^{\pm}x_j^{\mp} \vee x_j^{\pm}x_i^{\mp}$.*

Les astuces intervenant dans l'élévation au carré classique (voir par exemple [7]) s'appliquent également au calcul de $(X^{\pm})^2$: $x_i^{\pm}x_i^{\pm} = x_i^{\pm}$ et $x_i^{\pm}x_j^{\pm} + x_j^{\pm}x_i^{\pm} = 2x_i^{\pm}x_j^{\pm}$. La figure 6 illustre le mécanisme de calcul de produits partiels pour X^+X^+ et X^+X^- lorsque $n = 6$. Nous n'avons toutefois pas encore formalisé les algorithmes de génération des vecteurs de bits $\Lambda^{(i)}$ pour l'élévation au carré.

Exemple 5. *Afin d'illustrer les simplifications intervenant lors d'une élévation au carré, considérons le cas où $n = 5$. Nous pouvons calculer $(X^{\pm})^2$ à l'aide du circuit de la figure 4a. La propriété 5 indique qu'aucune génération de retenue ne survient lorsque $n = 5$ et nous supprimons les cellules HA et FA (figure 7a). Il est également possible de simplifier, dans une moindre mesure, le calcul de X^+X^- (figure 7b).*

6 Conclusion

Les RN-codes sont tels que tronquer un tel code est équivalent à l'arrondir au plus près. Nous avons montré qu'il n'est pas trop difficile d'implanter des opérations arithmétiques dont les opérandes sont représentés par ces codes. Nous avons proposé des algorithmes permettant de

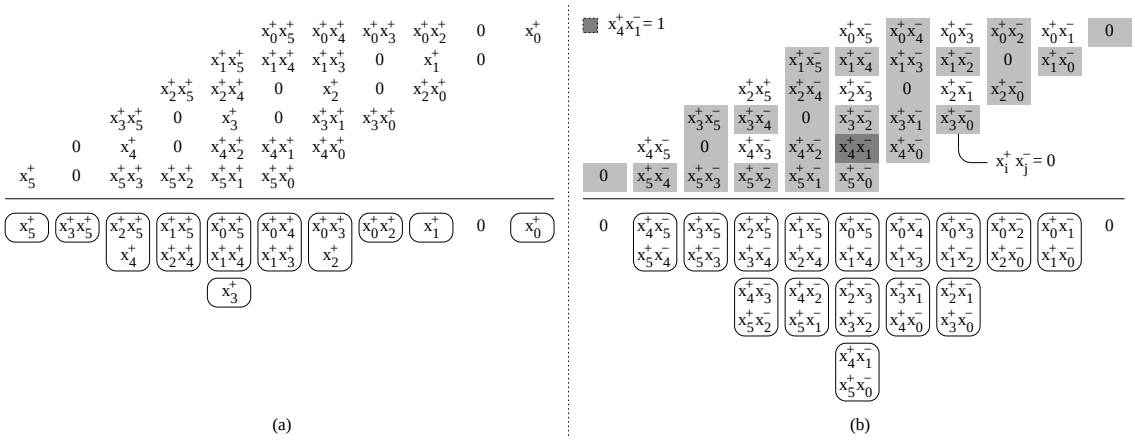


FIG. 6 – Application de la propriété 5 pour $n = 6$. (a) Calcul de $(X^+)^2$. (b) Calcul de X^+X^-. Si $x_4^+x_1^- = 1$ alors tous les termes de la forme $x_3^+x_k^-$, $x_5^+x_k^-$, $x_k^+x_4^-$, $x_k^+x_0^-$, $x_k^+x_2^-$ et $x_1^+x_k^-$ sont nuls.

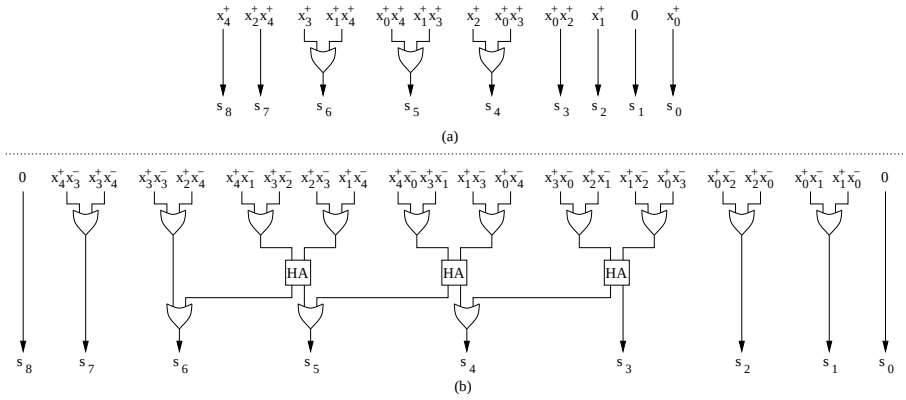


FIG. 7 – Calcul de (a) $(X^+)^2$ et de (b) X^+X^- lorsque $n = 5$.

construire des additionneurs et des multiplieurs, puis illustré des simplifications intervenant lors de l'élévation au carré. Nous devons toutefois encore formaliser cette dernière opération. Nous envisageons également de développer un autre algorithme de multiplication basé sur la propriété suivante : au plus un des termes $x_i^+y_j^+$, $x_i^-y_j^-$, $x_i^+y_j^-$ et $x_i^-y_j^+$ est égal à 1. Nous implanterons ensuite nos algorithmes en VHDL afin de les comparer.

References

[1] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961.

[2] J.-C. Bajard, J. Duprat, S. Kla, and J.-M. Muller. Some operators for on-line radix-2 computations. *Journal of Parallel and Distributed Computing*, 22:336–345, 1994.

[3] A. D. Booth. A signed binary multiplication technique. *Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[4] A. Cauchy. Sur les moyens d'éviter les erreurs dans les calculs numériques. *Comptes Rendus de l'Académie des Sciences, Paris*, 11:789–798, 1840. Republished in Augustin Cauchy, oeuvres complètes, 1ère série, Tome V, pp 431-442, available at http://gallica.bnf.fr/scripts/ConsultationTout.exe?O=N090185.

[5] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, 2nd edition, 1981.

[6] P. Kornerup and J.-M. Muller. RN-coding of numbers: definition and some properties. Technical Report 2004-43, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, October 2004.

[7] B. Parhami. *Computer Arithmetic*. Oxford University Press, 2000.

[8] J. E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, EC-7:218–222, 1958. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[9] K. D. Tocher. Techniques of multiplication and division for automatic binary divider. *Quarterly journal of mechanics and applied mathematics*, 11(3):364–384, 1958.

[10] R. Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. PhD thesis, Swiss Federal Institute of Technology Zurich, Hartung–Gorre Verlag, 1997.

A Preuves de quelques algorithmes et propriétés

Preuve de l'algorithme 3. Remarquons tout d'abord que

$$X^{\pm}Y^{\pm} = \sum_{i=0}^{n-1} 2^i \sum_{j=0}^{i} x_j^{\pm} y_{i-j}^{\pm} + \sum_{i=n}^{2n-2} 2^i \sum_{j=i-n+1}^{n-1} x_j^{\pm} y_{i-j}^{\pm} \tag{4}$$

La première double somme de l'équation (4) peut se réécrire

$$\begin{aligned}\sum_{i=0}^{n-1} 2^i \sum_{j=0}^{i} x_j^{\pm} y_{i-j}^{\pm} &= x_0^{\pm} y_0^{\pm} + \sum_{i=1}^{\lceil \frac{n}{2}-1 \rceil} 2^{2i} \sum_{j=0}^{2i} x_j^{\pm} y_{2i-j}^{\pm} + \sum_{i=0}^{\lfloor \frac{n}{2}-1 \rfloor} 2^{2i+1} \sum_{j=0}^{2i+1} x_j^{\pm} y_{2i+1-j}^{\pm} \\ &= x_0^{\pm} y_0^{\pm} + \sum_{i=1}^{\lceil \frac{n}{2}-1 \rceil} 2^{2i} \left(x_{2i}^{\pm} y_0^{\pm} + \sum_{j=0}^{2i-1} x_j^{\pm} y_{2i-j}^{\pm} \right) + \sum_{i=0}^{\lfloor \frac{n}{2}-1 \rfloor} 2^{2i+1} \sum_{j=0}^{2i+1} x_j^{\pm} y_{2i+1-j}^{\pm} \end{aligned} \tag{5}$$

Les deux sommes indicées par j comportent un nombre pair de termes que nous pouvons apparier grâce à la propriété 2

$$\sum_{j=0}^{2i-1} x_j^{\pm} y_{2i-j}^{\pm} = \sum_{j=0}^{i-1} (x_{2j}^{\pm} y_{2i-2j}^{\pm} + x_{2j+1}^{\pm} y_{2i-2j-1}^{\pm}) = \sum_{j=0}^{i-1} (x_{2j}^{\pm} y_{2i-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j-1}^{\pm})$$

et

$$\sum_{j=0}^{2i+1} x_j^{\pm} y_{2i+1-j}^{\pm} = \sum_{j=0}^{i} (x_{2j}^{\pm} y_{2i+1-2j}^{\pm} + x_{2j+1}^{\pm} y_{2i-2j}^{\pm}) = \sum_{j=0}^{i} (x_{2j}^{\pm} y_{2i+1-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j}^{\pm})$$

L'équation (5) s'écrit ainsi

$$\sum_{i=0}^{n-1} 2^i \sum_{j=0}^{i} x_j^{\pm} y_{i-j}^{\pm} = \lambda_0^{(0)} + \sum_{i=1}^{\lceil \frac{n}{2}-1 \rceil} 2^{2i} \left(\lambda_i^{(2i)} + \sum_{j=0}^{i-1} \lambda_j^{(2i)} \right) + \sum_{i=0}^{\lfloor \frac{n}{2}-1 \rfloor} 2^{2i+1} \sum_{j=0}^{i} \lambda_j^{(2i+1)}$$

où

$$\begin{aligned}
\lambda_0^{(0)} &= x_0^{\pm} y_0^{\pm} \\
\lambda_j^{(2i+1)} &= x_{2j}^{\pm} y_{2i+1-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j}^{\pm} \\
\lambda_j^{(2i)} &= \begin{cases} x_{2j}^{\pm} y_{2i-2j}^{\pm} \vee x_{2j+1}^{\pm} y_{2i-2j-1}^{\pm} & \text{si } i \neq j \\ x_{2i}^{\pm} y_0^{\pm} & \text{sinon} \end{cases}
\end{aligned}$$

Ces équations justifient les 12 premières lignes de l'algorithme 3. En appliquant le même principe à la seconde double somme de l'équation (4), nous obtenons

$$\begin{aligned}
\sum_{i=n}^{2n-2} 2^i \sum_{j=i-n+1}^{n-1} x_j^{\pm} y_{i-j}^{\pm} &= 2^{2n-2} x_{n-1}^{\pm} y_{n-1}^{\pm} + \sum_{i=\lfloor \frac{n+1}{2} \rfloor}^{n-2} 2^{2i} \sum_{j=2i-n+1}^{n-1} x_j^{\pm} y_{2i-j}^{\pm} + \\
&\quad \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-2} 2^{2i+1} \sum_{j=2i-n+2}^{n-1} x_j^{\pm} y_{2i-j+1}^{\pm} \\
&= 2^{2n-2} x_{n-1}^{\pm} y_{n-1}^{\pm} + \sum_{i=\lfloor \frac{n+1}{2} \rfloor}^{n-2} 2^{2i} \left(x_{n-1}^{\pm} y_{2i-n+1}^{\pm} + \sum_{j=2i-n+1}^{n-2} x_j^{\pm} y_{2i-j}^{\pm} \right) + \\
&\quad \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-2} 2^{2i+1} \sum_{j=2i-n+2}^{n-1} x_j^{\pm} y_{2i-j+1}^{\pm}
\end{aligned}$$

Les deux sommes indicées par j comportent à nouveau un nombre pair de termes. Il est toutefois utile de modifier l'indiçage avant d'appliquer la propriété 2

$$\begin{aligned}
\sum_{j=2i-n+1}^{n-2} x_j^{\pm} y_{2i-j}^{\pm} &= \sum_{j=0}^{2n-2i-3} x_{2i+j-n+1}^{\pm} y_{n-j-1}^{\pm} \\
&= \sum_{j=0}^{n-i-2} (x_{2i+2j-n+1}^{\pm} y_{n-2j-1}^{\pm} \vee x_{2i+2j-n+2}^{\pm} y_{n-2j-2}^{\pm})
\end{aligned}$$

et

$$\begin{aligned}
\sum_{j=2i-n+2}^{n-1} x_j^{\pm} y_{2i-j+1}^{\pm} &= \sum_{j=0}^{2n-2i-3} x_{2i+j-n+2}^{\pm} y_{n-j-1}^{\pm} \\
&= \sum_{j=0}^{n-i-2} (x_{2i+2j-n+2}^{\pm} y_{n-2j-1}^{\pm} \vee x_{2i+2j-n+3}^{\pm} y_{n-2j-2}^{\pm})
\end{aligned}$$

Il est facile de montrer que ces équations correspondent aux lignes 13 à 23 de l'algorithme 3. □

Preuve de la propriété 3. Si n est pair, nous déduisons de la définition du RN-code que

$$X_{\max}^{\pm} = \sum_{i=0}^{(n/2)-1} 2^{2i+1} = \underbrace{1010\ldots10}_{n \text{ chiffres}} \tag{6}$$

Si $n = 2$, il est facile de montrer que la propriété 3 est correcte : $X_{\max}^{\pm} = (10)_2 = 2 = 2(2^2 - 1)/3$. Supposons maintenant que la propriété soit vraie pour n et montrons qu'elle l'est également pour

$n+2$. Nous déduisons de l'équation (6) que

$$\begin{aligned} X^{\pm}_{\max} &= 2^{n+1} + \sum_{i=0}^{(n/2)-1} 2^{2i+1} \\ &= 2^{n+1} + \frac{2(2^n-1)}{3} = \frac{2^{n+2}+2^{n+1}+2^{n+1}-2}{3} \\ &= \frac{2(2^{n+2}-1)}{3} \end{aligned}$$

La preuve est analogue dans le cas où n est impair. Par définition,

$$X^{\pm}_{\max} = \sum_{i=0}^{(n-1)/2} 2^{2i} = \underbrace{1010\ldots101}_{n \text{ chiffres}}$$

Lorsque $n=1$, nous vérifions aisément que $(2(2^1-1)+1)/3 = 1$. En supposant la propriété vraie pour n, nous montrons à nouveau qu'elle l'est pour $n+2$

$$\begin{aligned} X^{\pm}_{\max} &= 2^{n+1} + \sum_{i=0}^{(n-1)/2} 2^{2i} \\ &= 2^{n+1} + \frac{2(2^n-1)+1}{3} = \frac{2^{n+2}+2^{n+1}+2^{n+1}-2+1}{3} \\ &= \frac{2(2^{n+2}-1)+1}{3} \end{aligned}$$

□

Preuve de la propriété 5. Nous déduisons de la propriété 3 que

$$X^{\pm}\cdot X^{\pm} \leq \begin{cases} \left(\dfrac{2(2^n-1)}{3}\right)^2 = \dfrac{2^{2n+2}-2^{n+3}+4}{9} < 2^{2n-1} & \text{si } n \text{ est pair} \\ \left(\dfrac{2(2^n-1)+1}{3}\right)^2 = \dfrac{2^{2n+2}-2^{n+2}+1}{9} < 2^{2n-1} & \text{sinon} \end{cases}$$

Supposons maintenant que $X^{\pm} = X_{\max}$. Par définition, $X^{\mp} \leq 2^n - 1 - X_{\max}$ et

$$\begin{aligned} X^{\pm}X^{\mp} &\leq 2^n X_{\max} - X_{\max} - X^2_{\max} \\ &= \begin{cases} \dfrac{2^{2n+1}-2^{n+2}+2}{3} < 2^{2n-2} & \text{si } n \text{ est pair} \\ \dfrac{2^{2n+1}-2^{n+2}-2^n+2}{9} < 2^{2n-2} & \text{sinon} \end{cases} \end{aligned}$$

La deuxième partie de la propriété découle immédiatement de la définition du RN-code étudié dans cet article. □