I) Introduction

I-1) Sujet

La rapidité et la réduction en taille des composants ont toujours été les objectifs de l'électronique numérique moderne pour améliorer les puissances de calculs. En effet les domaines de l'électronique et de l'informatique d'aujourd'hui imposent l'utilisation de composants très performants. Par exemple, dans le cadre d'une étude sur des algorithmes de compression d'images vidéo, il est nécessaire, pour le calcul d'une Transformée de Cosinus Discrète, de faire appel à des opérateurs rapides de multiplication. Pour développer un circuit à haute capacité de traitement qui utilise la multiplication, il est nécessaire de connaître les performances des opérateurs. Les critères les plus importants sont les suivants:

- la vitesse de multiplication est un paramètre très important, car elle est déterminante au niveau des performances atteintes,
- la surface de la puce est un paramètre primordial car le prix du circuit intégré en dépend fortement,
- la consommation est également un critère de premier ordre.

Ces trois critères déterminent l'emploi ou non d'un multiplieur précis.

Notre travail va consister à faire une étude des différents algorithmes de multiplication et d'addition à travers une implémentation hardware en VHDL des différents opérateurs étudiés et de faire des tests en vue de leur synthèse.

Les techniques de multiplication rapide sont nombreuses et complexes et le but de notre travail n'est pas de faire un exposé de ces méthodes. D'une manière générale, pour améliorer la rapidité d'un multiplieur, on cherche à diminuer le nombre de produits partiels en effectuant un recodage du multiplieur et on utilise des additionneurs rapides. Il existe plusieurs méthodes de codage du multiplieur. Il y a le codage par paires adjacentes (Wallace) , le codages différentiateur pur (Booth), le codage différentiateur modifié pour les 0 et les 1 isolés (Reitwiesner). On ne s'intéressera qu'aux algorithmes de Booth car ce sont les techniques les plus utilisées dans le domaine de la multiplication rapide. En ce qui concerne les additionneurs rapides, on utilise des architectures plus ou moins complexes. On ne comparera que des additionneurs fondamentaux (additionneur classique, additionneur à report anticipé) et on montrera l'influence de l'architecture des additionneurs sur les performances du multiplieur.

On se s'intéressera pas à la multiplication signée car ce n'est pas l'objet de notre étude. Cependant, on rappellera les codages des nombres binaires signés et on présentera simplement quelques techniques utilisées pour le traitement de la multiplication signée.

I-2) Cahier des charges

Ayant choisi une technologie bien précise, on fera une étude comparative des différents opérateurs. Il n'y a pas, a priori, de contraintes en ce qui concerne le temps de propagation et la surface (nombres de portes logiques utilisées). On s'intéressera à des multiplieurs 16*16 bits. Le but de cette étude est de permettre de choisir un opérateur précis en fonction d'un cahier des charges pour un projet précis.

I-3) Types de codage en binaire - Multiplication signée

Le tableau suivant décrit, pour des nombres binaires signés sur 4 bits, les codages en "Valeur absolue + Signe ", en " Complément Vrai ", en "Binaire Décalé".

DECIMAL	VA+S	CV	BD
7	0111	0111	1111
6	0110	0110	1110
5	0101	0101	1101
4	0100	0100	1100
3	0011	0011	1011
2	0010	0010	1010
1	0001	0001	1001
0	0000	0000	1000
-0	1000		
-1	1001	1111	0111
-2	1010	1110	0110
-3	1011	1101	0101
-4	1100	1100	0100
-5	1101	1011	0011
-6	1110	1010	0010
-7	1111	1001	0001
-8		1000	0000

Tableau 1: Types de codage en nombre binaire

- Le codage en Valeur Absolue + Signe (VA + S) donne le signe sur le bit de poids fort, les autres bits donnant la valeur absolue. Dans cette représentation le

chiffre 0 possède 2 codes 0000 (0) et 1000 (-0). Ce type de codage n'est pas très utilisé car il ne facilite pas les opérations algébriques.

- Le codage en Complément Vrai (CV) impose un poids de -2^{n-1} sur le $n^{ième}$ bit. Ce type de codage est intéressant dans la mesure où un opérateur arithmétique logique d'addition est utilisable sans modification pour les entiers relatifs codés en Complément Vrai (valable que dans certains cas suivant qu'on exprime le résultat sur n ou n+1 bits).

- Le codage en Binaire Décalé (BD) s'obtient à partir du Complément Vrai en inversant le bit de poids fort. Celui-ci permet d'utiliser les opérateurs arithmétiques sans limitation, le résultat s'obtenant sur un format de n+1 bits.

Un problème important à résoudre est celui de la multiplication signée. D'une manière générale, on retient le codage en Complément Vrai. Lorsqu'on effectue une multiplication, on obtient des produits partiels et on effectue ensuite des additions. Or un additionneur donne toujours un résultat correct (sur n+1 bits) en Binaire décalé. On est obligé, pour avoir un résultat correct en Complément Vrai, de convertir les données qui entrent dans l'additionneur en Binaire Décalé (on inverse le bit de poids fort) et d'inverser le report sortant.

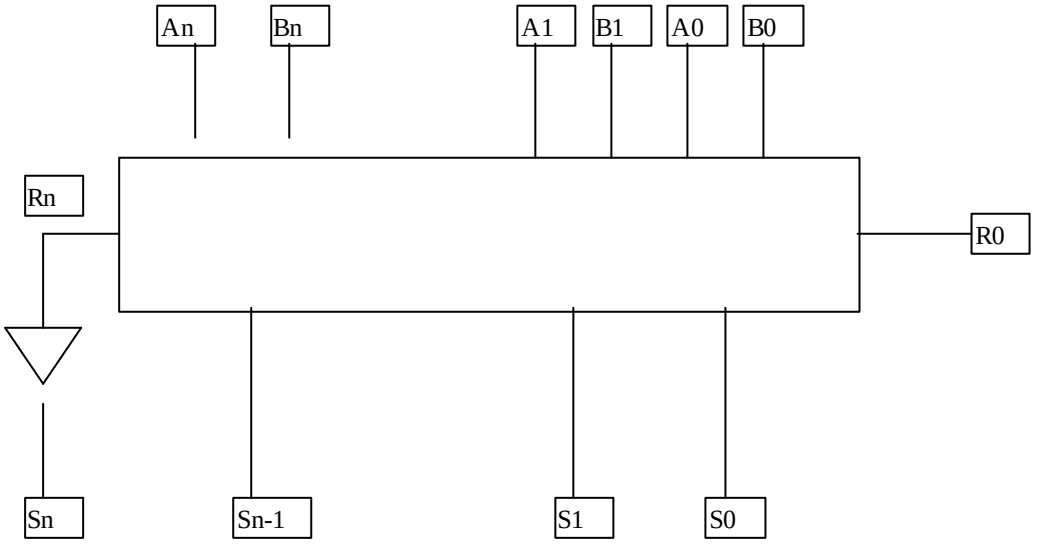


Figure 1

Il existe également des méthodes utilisant les valeurs absolues des deux nombres à multiplier. Lorsqu'on regarde les signes des opérandes (bits de poids fort), on connaît, sans faire le moindre calcul, le signe du produit (l'opérateur booléen est un OU EXCLUSIF). Par conséquent, on effectue une multiplication non signée

des deux valeurs absolues des opérandes et, si le produit est négatif, on transforme à nouveau le résultat (codage en Complément Vrai).

I-4) Accélération de la multiplication

La multiplication fondée sur l'examen chiffre par chiffre du multiplicateur nécessite n+1 étapes, où n+1 est la longueur de mot (signe compris). Le temps de multiplication dépend donc d'abord du temps d'addition et du nombre d'additions effectives (on considère uniquement les chiffres non nuls). Par conséquent, pour réduire le temps de multiplication, il faut:
- diminuer le temps d'addition par l'emploi d'additionneurs rapides (ce sera l'objet de la partie III),
- diminuer le nombre d'additions effectives grâce à un recodage approprié du multiplicateur (c'est ce que l'on va aborder dans la partie suivante).

II) Produits partiels - Codage du multiplicateur

II-1) Multiplication classique

Dans la multiplication classique, le multiplicande (que l'on notera A) est multiplié successivement par chaque bit du multiplicateur (que l'on notera B). Puis les produits partiels sont additionnés entre eux avec un décalage afin d'obtenir le produit final (noté P).

Le diagramme suivant (figure 2) montre une multiplication simple 16*16 bits.

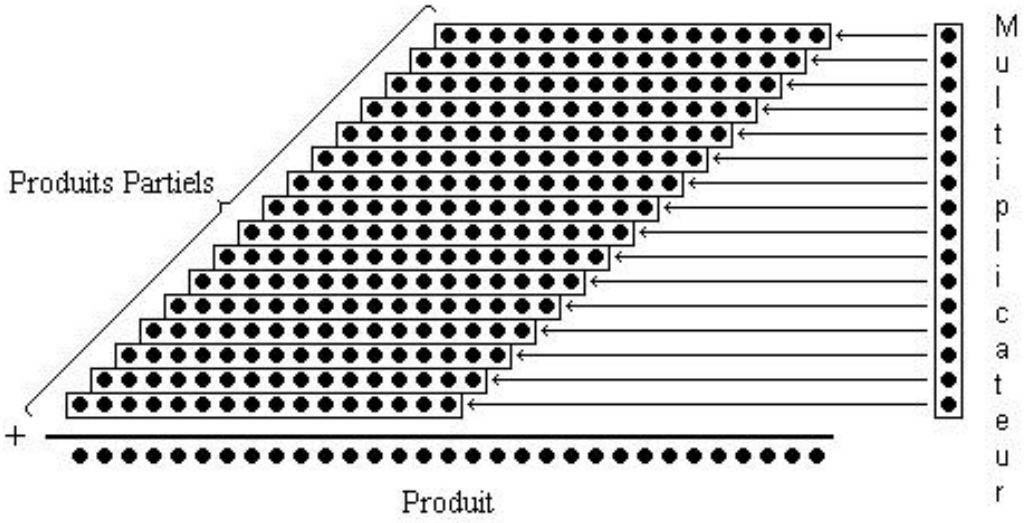


Table de Sélection des Produits Partiels

Bit du Multiplicateur	Sélection
0	0
1	Multiplicande

Figure 2

Avec des opérandes de n bits chacun, on obtient un résultat sur 2n bits. Chaque point du diagramme correspond à un bit qui peut valoir 0 ou 1. Les produits partiels sont représentés par des rangées horizontales de points et la méthode utilisée pour la production de chaque produit partiel est expliquée par la table à coté du diagramme. On remarque que les produits partiels s'obtiennent facilement à l'aide d'un opérateur ET.

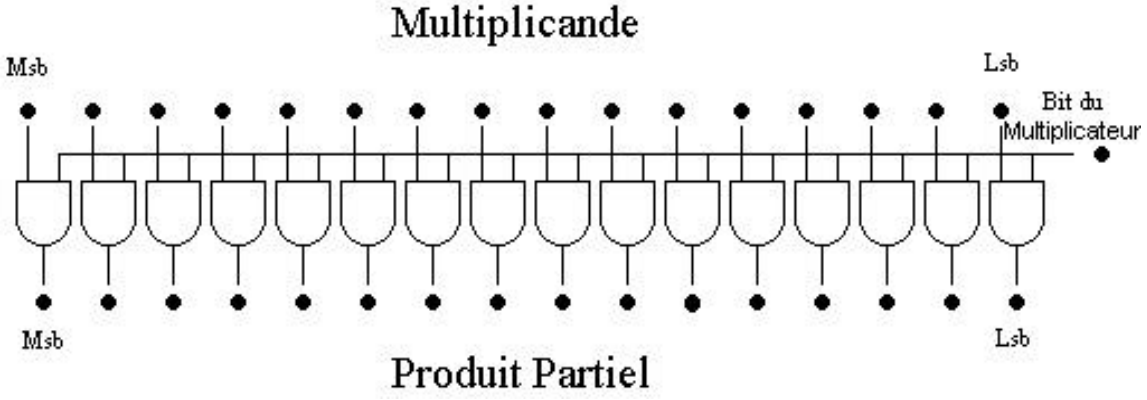


Figure 3

Si on a une multiplication signée, où les nombres sont codés en Complément Vrai, les additions se font en binaire décalé. Il faut par conséquent complémenter les produits partiels issus du bit de signe du multiplicande (A) et du bit de signe du multiplicateur (B). Le produit partiel issu des bits de signe du multiplicande (A) et du multiplicateur (B) restera inchangé puisqu'il faut prendre le complément deux fois. Puis on n'inverse pas les retenues sortantes, puisqu'il faut les inverser une première fois à la sortie de l'additionneur et une seconde fois à l'entrée de l'additionneur suivant, exceptée la dernière retenue qui donne le MSB du résultat final (P). Dans la figure suivante les inversions sont représentées par des cercles blancs.

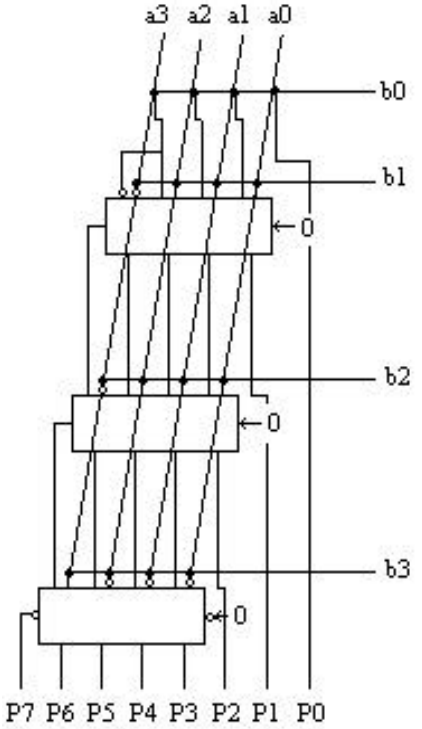


Figure 4

Les performances d'un opérateur utilisant la multiplication classique sont souvent insuffisantes en fonction d'un cahier des charges précis (surtout au niveau des temps de retard). Pour améliorer le temps de calcul, on peut jouer sur les différentes implémentations des portes logiques (synthèse optimisée) en fonction d'un algorithme particulier (ici c'est l'algorithme de la multiplication simple), mais pour avoir des résultats significatifs, il faut jouer sur le nombre de produits partiels à additionner. Ainsi, pour réduire le nombre d'additions élémentaires, on effectue un codage du multiplicateur.

II-2) Les codages de Booth

Le principe de l'accélération de la multiplication par recodage du multiplicateur consiste à effectuer le produit sur la base d'un examen du multiplicateur dans le sens des poids croissants en recodant ce dernier pour diminuer le nombre d'additions à réaliser.

Un procédé connu utilisant ce principe est le codage de Booth.
Pour réduire le nombre de produits partiels, on groupe les bits du multiplicateur (B) en plusieurs paires, et on sélectionne les produits partiels parmi les nombres 0,A,2A,3A,4A où A est le multiplicande. On réduit le nombre de produits partiels de moitié mais cela nécessite des additionneurs supplémentaires pour générer, par exemple 3A (2A et 4A s'obtiennent à partir de A par des décalages à gauche (multiplication par 2)).

II-2-1) Booth2

Considérons le multiplicateur B, d'une longueur de 16 bits, codé en complément vrai et A le multiplicande:

On notera les produits partiels PPi.

$$B = -b_{15}2^{15} + \sum_{i=0}^{14} b_i 2^i$$

$$B = -b_{15}2^{15} + b_{14}2^{14} + b_{13}2^{13} + \ldots + b_2 2^2 + b_1 2 + b_0$$

$$B = -b_{15}2^{15} + b_{14}2^{14} + b_{13}2^{13}(2-1) + \ldots + b_2 2^2 + b_1 2(2-1) + b_0$$

$$B = -b_{15}2^{14} + b_{14}2^{14} + b_{13}2^{14} - 2b_{15}2^{12} + \ldots + b_2 2^2 + b_1 2^2 - 2b_1 + b0$$

$$B = 2^{14}\left[-2b_{15} + b_{14} + b_{13}\right] + 2^{12}\left[-2b_{13} + b_{12} + b_{11}\right] + \ldots + 2^2\left[-2b_3 + b_2 + b_1\right] - 2b_1 + b_0$$

$$B = PP7 \cdot 4^7 + PP6 \cdot 4^6 + \ldots + PP1 \cdot 4^1 + PP0 = \sum_{i=1}^{7} PPi \cdot 4^i + PP0$$

Le produit AB peut s' écrire:

$$AB = \sum_{i=1}^{7} PPi \cdot A \cdot 4^i + PP0 \cdot A$$

On réduit le nombre de produits partiels qui peuvent prendre les valeurs -2A,-A,0,A,2A (excepté le produit PP0 qui peut prendre les valeurs -2A,-A,0,A).
La figure 5 illustre ce principe et montre le diagramme d'un multiplieur 16*16 bits qui utilise cet algorithme noté Booth2 (algorithme de Booth d'ordre 2).

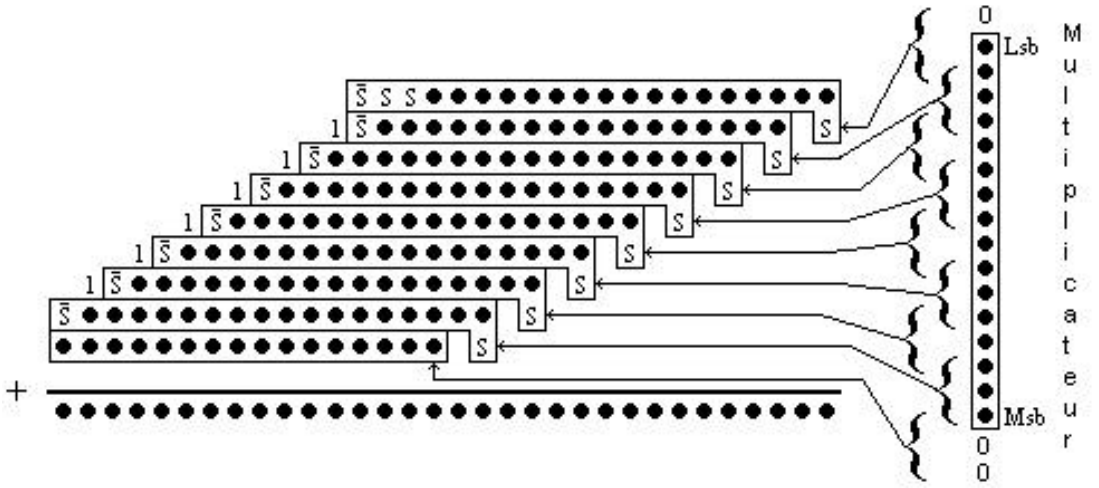


Table de Sélection des Produits Partiels

Bits du Multiplicateur	Sélection
000	+ 0
001	+ Multiplicande
010	+ Multiplicande
011	+2 x Multiplicande
100	-2 x Multiplicande
101	- Multiplicande
110	- Multiplicande
111	- 0

S = 0 si le produit partiel est positif (partie haute du tableau)
S = 1 si le produit partiel est négatif (partie basse du tableau)

Figure 5

Le multiplicateur est décomposé en plusieurs groupes de 3 bits, et chaque groupe est décodé pour sélectionner le produit partiel PPi à l'aide de la table de sélection (0,-0,A,-A,2A,-2A). Chaque produit partiel est décalé de 2 bits par rapport au produit partiel précédent (multiplication par 4). Les termes négatifs (codés en Complément Vrai) s'obtiennent en complémentant bit à bit le multiple positif correspondant , avec le rajout de 1 au LSB (ce sont les bits S du diagramme qui se trouvent sur le coté droit des produits partiels):

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

$$-2A = \overline{2A} + 1$$

$$-0 = "111...1" + 1$$

C'est une légère modification du codage et le rajout de 1 se fera sur le report entrant des additionneurs. De plus, si on considère une multiplication signée, on

inverse les bits de poids fort issus des produits partiels pour effectuer le passage Complément Vrai - Binaire décalé et on n'inverse pas la retenue de l'additionneur précédent (contrairement à la multiplication classique).

Le nombre de produits partiels est passé de 16 (multiplication classique) à 9. Si n est le nombre de bits des opérandes, le nombre de produits partiels pour l'algorithme Booth2 vaut (n+2)/2. Si le nombre d'additions est moins élevé et par conséquent on obtient une multiplication plus rapide (le gain de temps par rapport à la multiplication classique est estimé environ à 26%), la logique de sélection des produits partiels est plus complexe et la taille de l'opérateur devient plus grande. Il apparaît déjà le compromis temps de retard/surface.

II-2-2) Booth3

On peut réduire encore le nombre de produits partiels. L'algorithme de Booth3 (algorithme de Booth d'ordre 3) est donné par la figure 6.

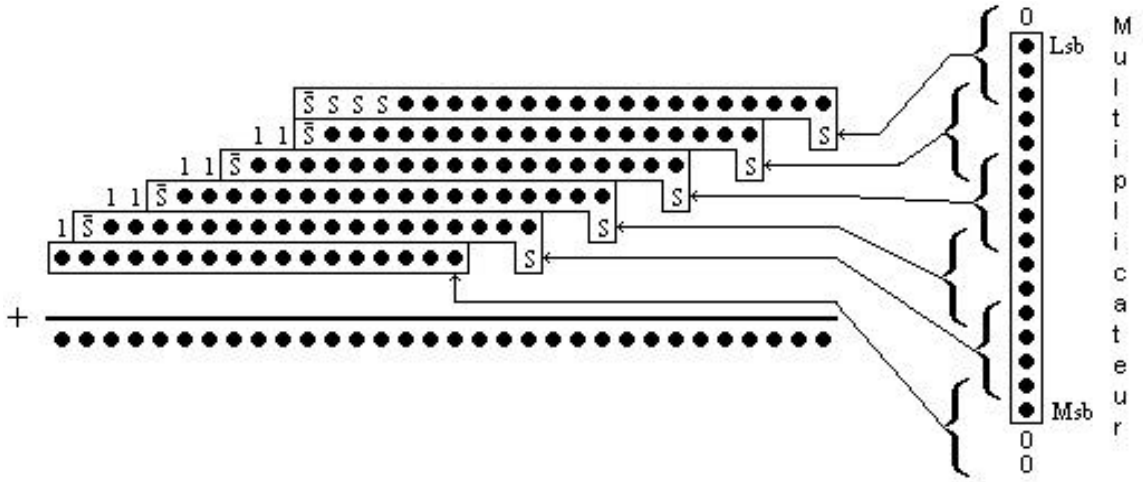


Table de Sélection des Produits Partiels

Bits du Multiplicateur	Sélection	Bits du Multiplicateur	Sélection
0000	+ 0	1000	- 4 x Multiplicande
0001	+ Multiplicande	1001	- 3 x Multiplicande
0010	+ Multiplicande	1010	- 3 x Multiplicande
0011	+ 2 x Multiplicande	1011	- 2 x Multiplicande
0100	+ 2 x Multiplicande	1100	- 2 x Multiplicande
0101	+ 3 x Multiplicande	1101	- Multiplicande
0110	+ 3 x Multiplicande	1110	- Multiplicande
0111	+ 4 x Multiplicande	1111	- 0

S = 0 si le produit partiel est positif (partie gauche du tableau)
S = 1 si le produit partiel est négatif (partie droite du tableau)

Figure 6

Le multiplicateur est décomposé en plusieurs groupes de 4 bits, et chaque groupe est décodé pour sélectionner le produit partiel PPi à l'aide de la table de sélection (0,-0,A,-A,2A,-2A,3A,-3A,4A,-4A). Chaque produit partiel est décalé de 4 bits par rapport au produit partiel précédent (multiplication par 8). Les termes négatifs (codés en Complément Vrai) s'obtiennent de la même manière que pour l'algorithme Booth2.

On réduit le nombre de produits partiels (ici 6 au lieu de 9 pour l'algorithme Booth2) mais la logique de sélection des produits partiels devient de plus en plus complexe. Si les générations des multiples 2A et 4A s'obtiennent facilement (décalages à gauche), il n'en est pas de même pour le multiple 3A qui nécessite une addition supplémentaire (ou soustraction) et qui, par conséquent, augmente le temps de calcul. Pour remédier à cela, il existe des algorithmes de Booth modifiés pour améliorer la rapidité de la production des multiples 3A (par exemple, au lieu de générer le produit partiel 3A, on génère deux produits partiels A et 2A et on optimise les additions). On peut continuer à diminuer le nombre de produits partiels (on arrive à l'algorithme Booth4) mais il faut alors générer les multiples 3A,5A,7A. Il y a un compromis à trouver entre ces générations de multiples et le nombre de produits partiels pour avoir un temps de calcul le plus petit possible.

III) Les additionneurs pour la multiplication

Le fait que l'on s'intéresse à la rapidité des additionneurs pour obtenir le résultat final se comprend aisément. Pour améliorer les performances du multiplieur, on cherchera à minimiser le temps de propagation des additionneurs.
Ici, on se place dans le contexte d'additionneurs utilisés pour la multiplication. Il est possible d'utiliser un additionneur dans un circuit séquentiel, mais on cherche à caractériser le multiplieur dans son ensemble (ASIC). On utilisera une description combinatoire de l'opérateur: les produits partiels seront additionnés progressivement par plusieurs additionneurs. Des registres (bascules D) pourront être utilisés pour rendre le multiplieur synchrone lorsque ce dernier sera utilisé dans une architecture complexe synchrone.

III-1) Additionneur classique

On additionne les nombres **A** (n bits) et **B** (n bits) et on obtient le résultat **S** (n bits). On note les retenues $\mathbf{r_k}$ et la retenue sortante est $\mathbf{r_n}$.

La représentation binaire de **A** s'écrit:

$A = \sum_{k=0}^{n-1} a_k 2^k$ (les expressions sont similaires pour **B** et **S**).

Les équations de l'addition binaire de **A** et **B** donnant le résultat **S** et les retenues $\mathbf{r_k}$ sont:

Pour k = 0,1,...,n - 1

$s_k = a_k \oplus b_k \oplus r_k$ (1)

$r_{k+1} = a_k . b_k + a_k . r_k + b_k . r_k$ (2)

+ correspond à l'opérateur booléen OU.
. correspond à l'opérateur booléen ET.
⊕ correspond à l'opérateur booléen OU EXCLUSIF.

Dans le cadre de l'additionneur classique, la retenue r_{k+1} s'obtient à partir des termes a_k, b_k et de la retenue précédente r_k. C'est la méthode que l'on utilise manuellement pour faire une addition (on écrit la retenue au dessus des chiffres que l'on additionne).

On parle d'additionneur en cascade (figure 7).

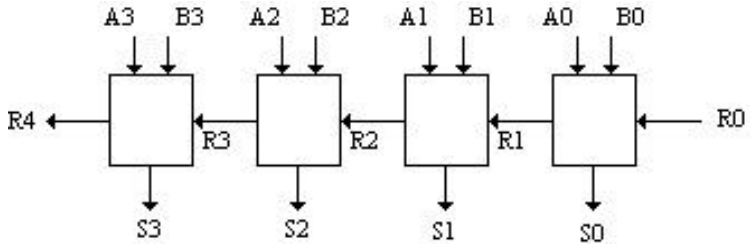


Figure 7

On réalise une synthèse série (les réseaux cellulaires sont en série) et on constate que le report issu du premier étage peut se propager à travers toutes les cellules (connexions cascades). Le temps de propagation peut alors devenir important si la longueur des nombres à additionner est grande (elle est supérieure à 16 bits dans notre étude). Pour diminuer le temps de propagation, on réalise une synthèse "plus parallèle". On va étudier un exemple d'additionneur accéléré.

III-2) Additionneur à report anticipé

Pour améliorer le rapidité de l'additionneur, on utilise les fonctions auxiliaires **p** (terme de propagation) et **g** (terme de génération) qui sont définies par les équations suivantes:

$$g_k = a_k . b_k \quad (3)$$
$$p_k = a_k \oplus b_k \quad (4)$$

D'après les équations (1) et (2), on obtient:

$$r_{k+1} = g_k + r_k . p_k \quad (5)$$
$$s_k = p_k \oplus r_k \quad (6)$$

Par conséquent, il est possible d'obtenir directement les retenues $\mathbf{r_k}$ à partir de $\mathbf{r_0}$ et par la suite les termes $\mathbf{s_k}$. En effet, en développant l'équation (5), on obtient:

$$r_1 = g_0 + r_0 . p_0$$
$$r_2 = g_1 + r_1 . p_1 = g_1 + p_1 . g_0 + p_1 . p_0 . r_0$$
$$r_3 = g_2 + r_2 . p_2 = g_2 + p_2 . g_1 + p_2 . p_1 . g_0 + p_2 . p_1 . p_0 . r_0$$
$$\vdots$$
$$r_{k+1} = g_k + r_k . p_k$$
$$= g_k + p_k . g_{k-1} + p_k . p_{k-1} . g_{k-2} + \ldots + p_k . p_{k-1} . p_{k-2} \cdots p_2 . p_1 . g_0$$
$$+ p_k . p_{k-1} . p_{k-2} + \ldots + p_2 . p_1 . p_0 . r_0$$

On obtient un gain de temps pour l'obtention des retenues $\mathbf{r_k}$, mais les équations logiques deviennent complexes et l'additionneur risque d'être trop volumineux. Le gain en vitesse est au détriment du nombre de portes. De plus, cette synthèse oblige les portes logiques à avoir des sortances importantes, ce qui est, par exemple, néfaste pour des portes CMOS qui sont sensibles aux charges capacitives.

Cet additionneur rapide porte le nom d'additionneur à report anticipé (ou à retenue anticipée), car on calcule d'abord les retenues $\mathbf{r_k}$ qui se propagent avant les termes $\mathbf{s_k}$ de la somme. La figure 8 montre l'architecture de cet additionneur.

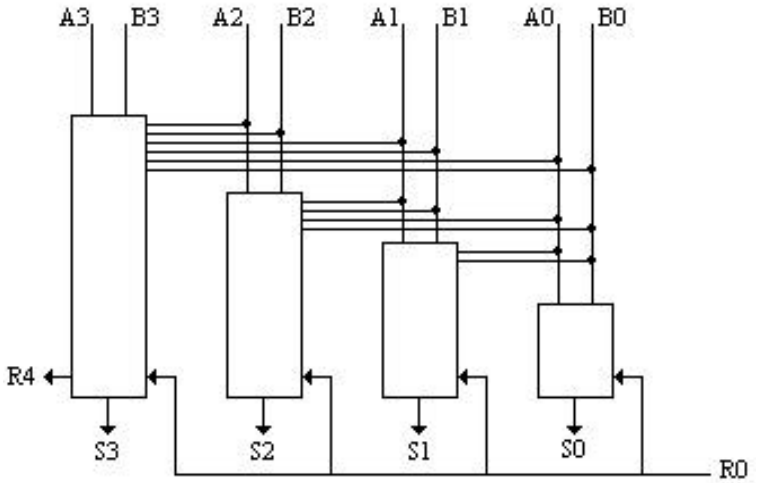


Figure 8

Il apparaît, là encore, le compromis temps de propagation/surface et une étude de structures intermédiaires entre les solutions série et parallèle reste à faire. On ne fera pas cette étude très complexe, mais à travers les simulations des deux additionneurs décrits ci-dessus, on regardera l'évolution de la surface des additionneurs en fonction de leur vitesse de calcul. Cela permettra de mieux cerner les variations des différents paramètres et de mieux cibler les recherches futures sur la synthèse d'additionneurs rapides pour la multiplication.

III-3) Additionneurs 4-2

Jusqu'à maintenant, on a supposé que l'on utilise des opérateurs qui additionnent au fur et à mesure les produits partiels (on fait PP0+PP1, puis PP2+somme(PP0,PP1) , etc...). Dans le but d'améliorer les vitesses de calcul des multiplieurs, des recherches ont été effectuées sur des architectures parallèles au niveau des additionneurs (arbres). Il semblerait que les performances en vitesse soient meilleures que celles des multiplieurs utilisant l'algorithme de Booth d'ordre 2. L'idée des additionneurs 4-2 est de diminuer le nombres de produits partiels à additionner de moitié. La figure 9 montre la réduction de 8 produits partiels avec des additionneurs 4-2.

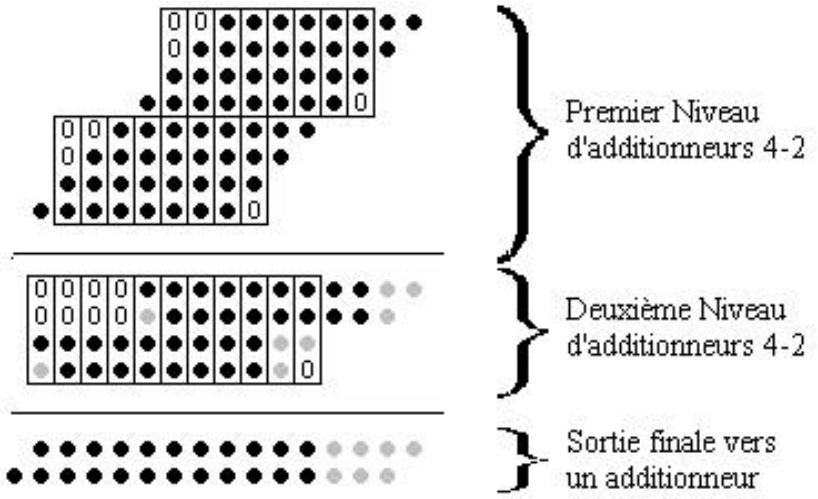
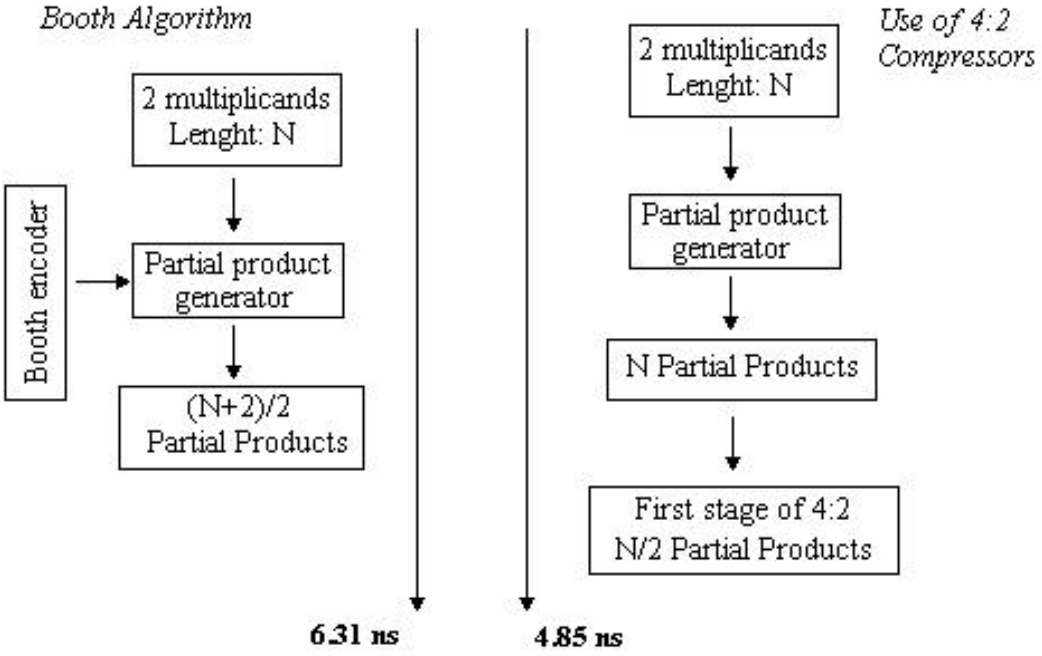


Figure 9

Des études comparatives avec l'algorithme de Booth2 montrent que le temps de calcul est plus faible pour le multiplieur utilisant les additionneurs 4-2.



Technologies CMOS 1u3m1p
Epoch library
T≈25°C
Power 5V
External capacity load of 50 ff added at each output.

Référence: Article intitulé 'Evaluation of Booth's Algorithm for Implementation in Parallel Multipliers' par Pascal Bonnato et Vojin G. Oklobdzija (Department of Electrical and Computer Engineering, University of California, Davis, CA 95616) qui s'appuie sur d'autres références citées en annexes.

Il semblerait, par contre, que cette architecture n'offre aucun avantage en ce qui concerne la surface et la consommation (le nombre d'additionneurs devient très

important). Pour avoir une idée plus précise, on effectuera une synthèse VHDL de cette architecture pour une multiplieur 16*16 bits et on fera une comparaison avec les multiplieurs utilisant les algorithmes de Booth avec une architecture classique (additions progressives).

IV) Implémentations et Tests de multiplieurs

IV-1) Les multiplieurs étudiés

On effectue une sorte de combinaison entres les multiplieurs et additionneurs décrits dans les parties II et III. On va comparer (temps de calcul, surface) les multiplieurs 16*16 bits suivants:

- Multiplieur Booth2 avec des additionneurs classiques (*).
- Multiplieur Booth3 avec des additionneurs classiques (*).
- Multiplieur Booth2 avec des additionneurs à report anticipé (*).
- Multiplieur Booth3 avec des additionneurs à report anticipé (*).
- Multiplieur classique avec des additionneurs 4-2 classiques (*).

(*): deux versions sont proposées:
- multiplieur utilisant des registres en sortie des additionneurs,
- multiplieur purement combinatoire (des registres pourront être ajoutés aux deux entrées et en sortie du multiplieur pour rendre ce dernier synchrone).

Par souci de simplicité, on ne s'intéresse qu'à des multiplieurs non signés.

IV-2) Description des fichiers VHDL

Chaque fichier VHDL décrit tous les composants utilisés par les multiplieurs (un fichier VHDL correspond à un multiplieur).
On trouve les composants suivants:

*
- *code0-booth2(ou 3)*
- *codeI-booth2(ou 3)*
- *codeN1-booth2(ou 3)*
- *codeN-booth2(ou 3)*
- *codesimple*

Les premiers composants (8 codeurs) génèrent les différents produits partiels issus des tables de sélection des algorithmes de Booth2 et Booth3.
Codesimple génère les produits partiels pour la multiplication classique.

*
- *addi xx*
(xx repésente les longueurs des nombres à additionner)
- *addi xx-simple*

Le premier nom correspond à un additionneur qui utilise les reports anticipés, le second correspond à un additionneur classique.
*
- *demi-addi-0-booth2(ou 3)*

Ce sont des demi-additionneurs qui utilisent uniquement un opérande A et une retenue cin (c'est r_0). Ils additionnent le premier produit partiel des algorithmes de Booth avec une retenue (premier bit S à additionner sur les diagrammes de Booth).
*
- *reg xx*

Ce sont des bascules D qui sont utilisées à la sortie de chaque additionneur pour rendre le multiplieur synchrone (temps de calcul en nombre de périodes d'horloge) et améliorer sa rapidité à l'intérieur d'un système synchrone.
*
- *multipl16-booth2(ou 3)*
- *multipl16s-booth2(ou 3)*
- *multipl16-booth2(ou 3)-sans-reg*
- *multipl16s-booth2(ou 3)-sans-reg*
- *multipl16-4-2*
- *multipl16-4-2-avec-reg*

multipt16 utilise les additionneurs à report anticipé.
multipl16s utilise les additionneurs classiques.
multipl16-4-2 utilise l'architecture des additionneurs 4-2.

C'est à ce niveau que l'on décrit l'implémentation des différents composants.

La figure 10 montre le schéma du multiplieur Booth2 avec des additionneurs (classiques ou à report anticipé) qui utilisent des registres (il suffit de quelques adaptations simples pour obtenir les schémas des autres multiplieurs).
Le structure du multiplieur Booth3 est équivalente à celle du multiplieur Booth2 avec seulement 6 codeurs (et donc 5 additionneurs) au lieu de 9.

La figure 11 montre l'arbre des additionneurs 4-2 pour le multiplieur 16*16 bits.

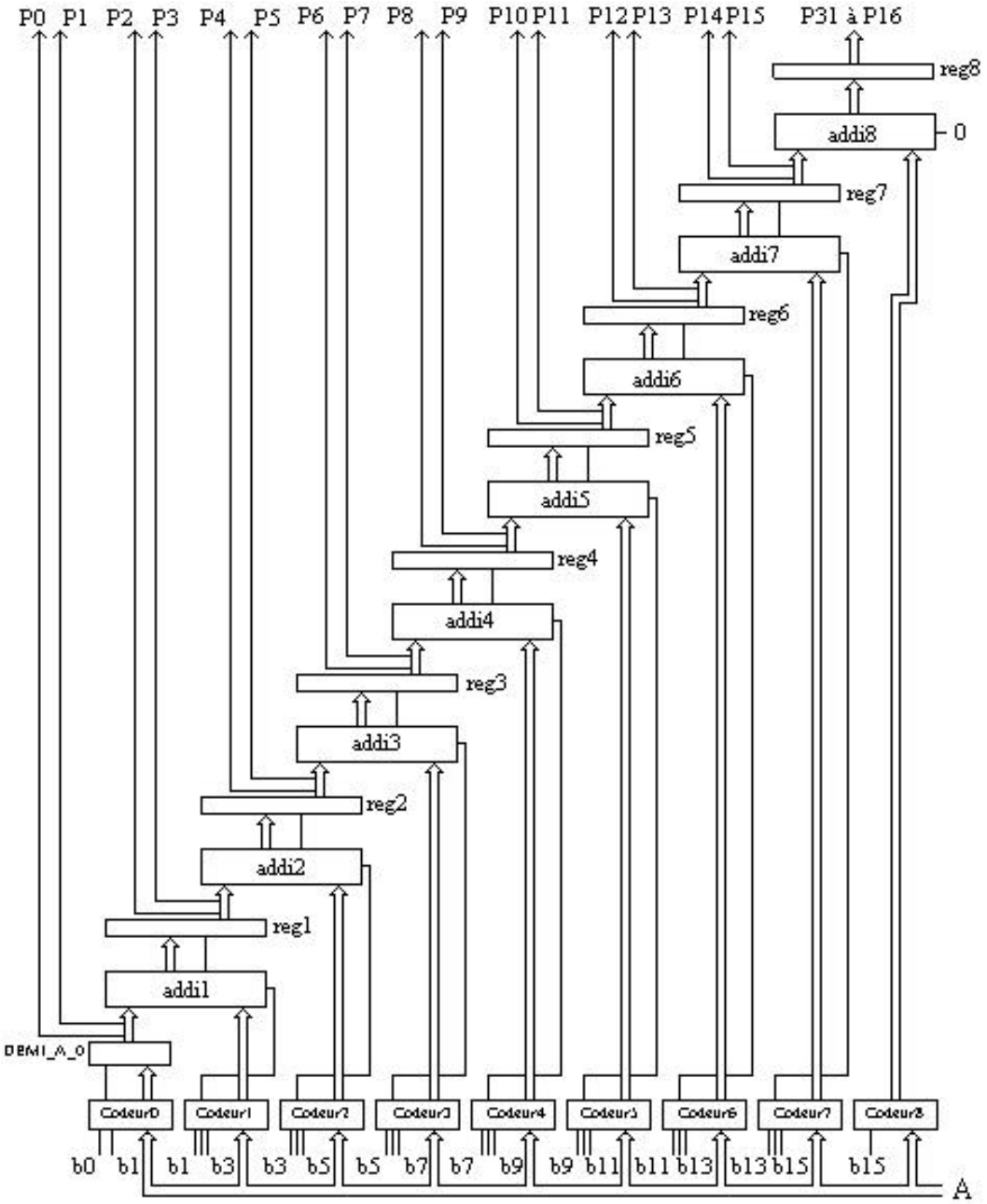


Figure 10

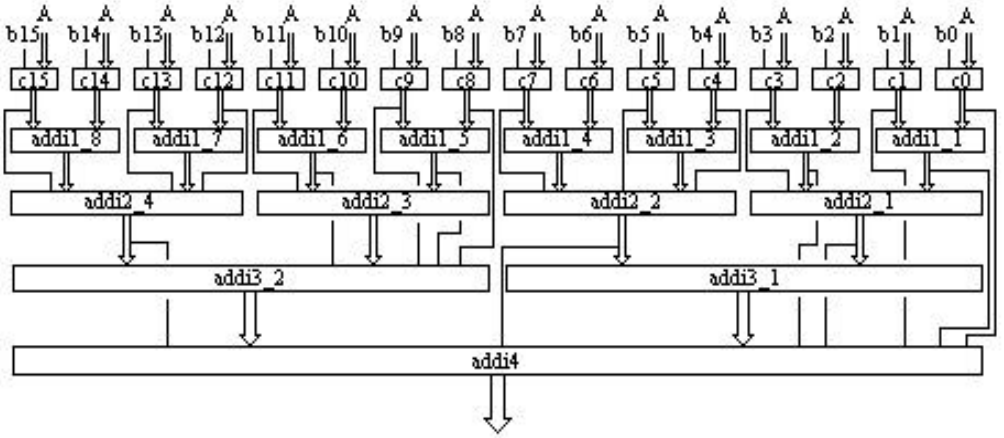


Figure 11

Le tableau 2 montre pour chaque multiplieur étudié les divers composants utilisés.

Multiplieurs 16*16 bits	Génération des Produits Partiels	Demi-additionneurs	Additionneurs	Version avec Registres	temps de calcul maximal (version avec registres)
Booth2	1 codeur: 20 bits + bit S 6 codeurs: 19 bits + bit S 1 codeur: 18 bits + bit S 1 codeur: 16 bits	1 demi-addi: 20 bits	6 additionneurs: 19 bits 1 additionneur: 18 bits 1 additionneur: 16 bits	6 reg: 20 bits 1 reg: 18 bits 1 reg: 16 bits	8 périodes d'horloge ou multiples
Booth3	1 codeur: 22 bits + bit S 3 codeurs: 21 bits + bit S 1 codeur: 20 bits + bit S 1 codeur: 17 bits	1 demi-addi: 22 bits	3 additionneurs: 21 bits 1 additionneur: 20 bits 1 additionneur: 17 bits 5 additionneurs: 17 bits génération de 3A	3 reg: 22 bits 1 reg: 20 bits 1 reg: 17 bits	5 périodes d'horloge ou multiples
Addi 4-2	16 codeurs: 16 bits	-	8 additionneurs: 16 bits 4 additionneurs: 19 bits 2 additionneurs: 23 bits 1 additionneur: 32 bits	8 reg: 17 bits 4 reg: 20 bits 2 reg: 24 bits 1 reg: 32 bits	4 périodes d'horloge ou multiples

Tableau 2: Composants utilisés par chaque multiplieur

D'après ce tableau, on constate que lorsqu'on diminue le nombre de produits partiels , on augmente la taille des codeurs et le nombre d'additionneurs. C'est le compromis entre le nombre de composants (et donc la surface de la puce) et la vitesse de calcul. Dans ce tableau, on donne des vitesses de calcul en périodes d'horloge, mais ces résultats ne nous intéressent que partiellement, car on cherche à caractériser les temps de propagation. En effet, la fréquence de l'horloge dépendra de ces temps de propagation (on considère par définition que le temps de propagation correspond au temps de traversé des portes par le chemin le plus long).

Les trois chronogrammes suivants montrent les périodes d'horloge nécessaires pour obtenir le résultat FFFF*FFFF pour les trois types de multiplieurs (versions avec registres).

- premier chronogramme: algorithme de Booth2,
- le second chronogramme: algorithme de Booth3
- troisième chronogramme: additionneurs 4-2.

On va donc comparer les multiplieurs étudiés en regardant les temps de propagation, le nombre de portes utilisées par chaque composant. Ainsi, pour améliorer les performances des multiplieurs, on maîtrisera davantage certains paramètres comme la taille des additionneurs et des codeurs, le gain de temps de l'additionneur à report anticipé par rapport à l'additionneur classique, les avantages et les inconvénients de la structure 4-2 par rapport aux structures classiques pour l'addition des produits partiels, et on pourra cibler les recherches futures.

IV-3) Synthèses et Simulations

Après avoir décrit les différents multiplieurs en langage VHDL, nous avons utilisé le logiciel AUTOLOGIC2 pour la synthèse sous l'environnement Mentor. Parmi les technologies disponibles, on a choisi la technologie AMS 0.8 micron avec la librairie BYB (BiCMOS).

La principale difficulté a été le fait que le logiciel impose un critère d'optimisation pour la synthèse vers la technologie AMS (ce qui est normal pour un logiciel de synthèse qui cherche à optimiser les composants en surface et en temps de retard). Par conséquent, les synthèses ont été effectuées sans, puis

avec un critère d'optimisation en surface. Le logiciel nous a alors fourni des informations sur le nombre de portes logiques et sur les temps de retard maximum (temps de traversée des portes logiques par le chemin le plus long).

Les tableaux 3 et 4 montrent les nombres de portes utilisées pour les différents multiplieurs. La synthèse a été faite à l'aide de deux critères d'optimisation: le premier demande au logiciel de ne pas optimiser en surface (cela signifie qu'il utilise d'autres critères), contrairement au second.

	Additionneurs	Codeurs	Autres primitives	Registres	Total
Booth2 Additionneurs Classiques	2101 2101	1214 1214	2 2	- 316	3317 3633
Booth2 Additionneurs à Report Anticipé	9205 9205	1338 1338	2 2	- 316	10545 10861
Booth3 Additionneurs Classiques	1595 1595	2367 2367	2 2	- 211	3964 4175
Booth3 Additionneurs à Report Anticipé	6782 6782	5411 5411	2 2	- 211	12195 12406
Structure 4-2 Additionneurs Classiques	2779 2779	512 512	1 1	- 626	3292 3918

Tableau3: Nombres de portes logiques de la technologie AMS ByB
sans critère d'optimisation en surface.

	Additionneurs	Codeurs	Autres primitives	Registres	Total
Booth2 Additionneurs Classiques	773 773	759 759	2 2	- 162	1534 1696
Booth2 Additionneurs à Report Anticipé	969 969	759 759	2 2	- 162	1730 1892
Booth3 Additionneurs Classiques	500 500	1389 1389	2 2	- 108	1891 1999
Booth3 Additionneurs à Report Anticipé	682 682	1264 1264	2 2	- 108	1948 2056
Structure 4-2 Additionneurs Classiques	1234 1234	512 512	1 1	- 330	1747 2077

Tableau4: Nombres de portes logiques de la technologie AMS ByB
avec un critère d'optimisation en surface.

Les tableaux 5,6,7 et 8 montrent les temps de propagation maximum calculés par le logiciel pour les multiplieurs dans leur ensemble et pour chaque composant. Il n'y a pas de critère d'optimisation en surface (c'est ce qu'on lui demande) mais le logiciel utilise d'autres critères , comme par exemple une optimisation en temps de retard. Par conséquent, on ne peut pas évoquer les types d'additionneurs utilisés dans le tableau suivant.

Booth2	117.20 ns
Booth3	110.83 ns
Additionneurs 4-2	78.48 ns

Tableau 5: Temps de retard maximum globaux.

Remarque: Ces temps de propagation ont été calculés pour les trois types de multiplieurs dans les mêmes conditions sans critère d'optimisation en surface.

Dans les tableaux 6, 7 et 8, les composants ont été testés séparément avec un critère d'optimisation qui ne touche pas (contraintes d'optimisation) aux temps de retard. Par conséquent il n'y a, a priori, aucune corrélation entre les tableaux

6,7,8 et le tableau 5. Cependant, on peut légitimement faire des comparaisons entre les valeurs issues du même tableau.

Booth2	Additionneurs Classiques	Additionneurs à Report Anticipé
Additionneurs 19 bits	49.23 ns	19.70 ns
Additionneur 18 bits	48.55 ns	18.78 ns
Additionneur 16 bits	41.40 ns	16.92 ns
Codeur 20 bits	8.46 ns	8.46 ns
Codeurs 19 bits	6.29 ns	6.29 ns
Codeur 18 bits	6.29 ns	6.29 ns
Codeur 16 bits	0.91 ns	0.91 ns

Tableau 6: Temps de retard maximum pour les composants du multiplieur Booth2

Booth3	Additionneurs Classiques	Additionneurs à Report Anticipé
Additionneurs 21 bits	52.10 ns	20.44 ns
Additionneur 20 bits	51.95 ns	20.25 ns
Additionneur 17 bits	44.30 ns	16.92 ns
Codeur 22 bits	44.30 ns	20.21 ns
Codeurs 21 bits	44.05 ns	19.95 ns
Codeur 20 bits	44.05 ns	19.95 ns
Codeur 17 bits	5.53 ns	5.45 ns

Tableau 7: Temps de retard maximum pour les composants du multiplieur Booth3

Remarque: La différence entre les deux colonnes pour les temps des codeurs provient des additionneurs de types différents qui génèrent les multiples 3A.

Architecture 4-2	Additionneurs Classiques
Additionneurs 16 bits	38.82 ns
Additionneurs 19 bits	47.02 ns
Additionneurs 23 bits	52.76 ns
Additionneur 32 bits	71.25 ns
Codeurs simples 16 bits	0.91 ns

Tableau 8: Temps de retard maximum pour les composants de la structure 4-2.

Remarque: On ne retrouve pas forcément les mêmes temps de propagation pour des additionneurs de tailles égales à cause des critères d'optimisation pendant les synthèses.

Avant de faire un commentaire sur ces temps de propagation, il faut prendre certaines précautions: Les valeurs proposées n'ont aucune signification physique réelle. En effet, les multiplieurs (dans leur ensemble) et les composants ont été testés avant les opérations de placement-routage. Les calculs du logiciel ne tiennent pas compte des capacités parasites et des charges éventuelles qui viendront s'ajouter dans le circuit intégré final. Par conséquent, il est nécessaire de réaliser le circuit pour avoir une véritable idée de ces temps de calcul.

Des simulations avec le logiciel QUICKSIM (chronogrammes) donnent les résultats suivants (les remarques précédentes restant valables pour les temps de retard proposés ici):

Booth2	44.7 ns
Booth3	36.0 ns
Additionneurs 4-2	31.9 ns

Tableau 9: Temps de retard des multiplieurs issus des chronogrammes.

Remarque: On ne cherche pas à trouver le temps maximum (chemin critique) mais à comparer avec des entrées quelconques (ici FFFF*FFFF) les variations des temps de retard des différents multiplieurs.

Ces temps (nettement inférieurs aux temps maximum calculés du tableau 5, la comparaison étant possible ici) ont été obtenus à l'aide des chronogrammes de la page suivante.

Les valeurs qui ont été proposées dans les tableaux précédents sont discutables à cause des remarques précédentes (temps de retard proposé différent du temps réel, nombres de portes dépendant des critères d'optimisation). Par contre ces résultats vont nous permettre, en considérant ces valeurs de manière relative, de faire les études comparatives souhaitées.

IV-4) Analyse des Résultats

IV-4-1) Comparaison des temps de propagation

D'après les tableaux 5 et 9, on remarque que le multiplieur ayant la structure des additionneurs 4-2 est plus rapide (temps maximum: 78.48 ns,

simulation: 31.9 ns) que les multiplieurs utilisant les algorithmes de Booth3 (temps maximum: 110.83 ns, simulation: 36.0 ns) et de Booth2 (temps maximum: 117.20 ns, simulation: 44.7 ns).

Les proportionnalités entre les temps de simulation et les temps de retard maximum calculés par le logiciel de synthèse sont respectées, excepté pour le multiplieur Booth3 où l'on trouve un temps de retard maximum un peu plus grand. Cela est dû au fait que les additionneurs qui génèrent les multiples 3A augmentent, dans les pires cas, les temps de propagation.

IV-4-2) Comparaison des surfaces

Les tableaux 3 et 4 permettent de faire les remarques suivantes:

- Sans critère d'optimisation en surface, le multiplieur Booth3 (3964 portes) est plus volumineux que le multiplieur Booth2 (3317 portes) et le multiplieur qui utilise les additionneurs 4-2 (3292 portes). Avec le critère d'optimisation, c'est le multiplieur (Booth2) qui devient le moins volumineux (1534 portes) par rapport aux multiplieur 4-2 (1747 portes), le multiplieur Booth3 ayant toujours une surface plus grande (1891 portes). Le classement reste inchangé avec les versions qui utilisent les additionneurs à report anticipé. L'ajout des registres en sortie des additionneurs augmente très légèrement les nombres de portes.

- On constate que lorsqu'on utilise le critère d'optimisation en surface, le nombre de portes reste du même ordre de grandeur pour les versions utilisant les additionneurs classiques et les versions utilisant les additionneurs à report anticipé. Ceci montre que le logiciel a "cassé" la structure des deux types additionneurs en essayant de simplifier les équations logiques et d'optimiser le nombre de portes. Par contre, sans critère d'optimisation, les surfaces des multiplieurs utilisant les additionneurs à report anticipé sont beaucoup plus importantes.

IV-4-3) Comparaison des deux types d'additionneurs

D'après le tableau 3 (sans critère d'optimisation en surface), on constate que les multiplieurs utilisant les additionneurs à report anticipé sont **3** fois plus volumineux que ceux qui utilisent les additionneurs classiques (Booth2: 10545/3317 = 3.18, Booth3: 12195/3964 = 3.08), le rapport de nombres de portes pouvant atteindre 5 si on considère les additionneurs seuls.

D’après les tableaux 6 et 7, il est possible de faire une estimation du gain de temps du multiplieur qui utilise les additionneurs à report anticipé par rapport à celui qui utilise les additionneurs classiques. D’après l’architecture des multiplieurs, on effectue la somme des temps de retard maximum pour les deux types d’additionneurs (ce calcul n’est pas rigoureux car le temps maximum global est inférieur à la somme des temps maximum des composants en cascade mais il permet d’avoir une estimation convenable, compte tenu du fait que le temps de retard global dépend surtout de la vitesse des additionneurs) et on effectue le rapport des deux résultats obtenus. On obtient pour les multiplieurs Booth2 et Booth3 le gain de temps suivant:

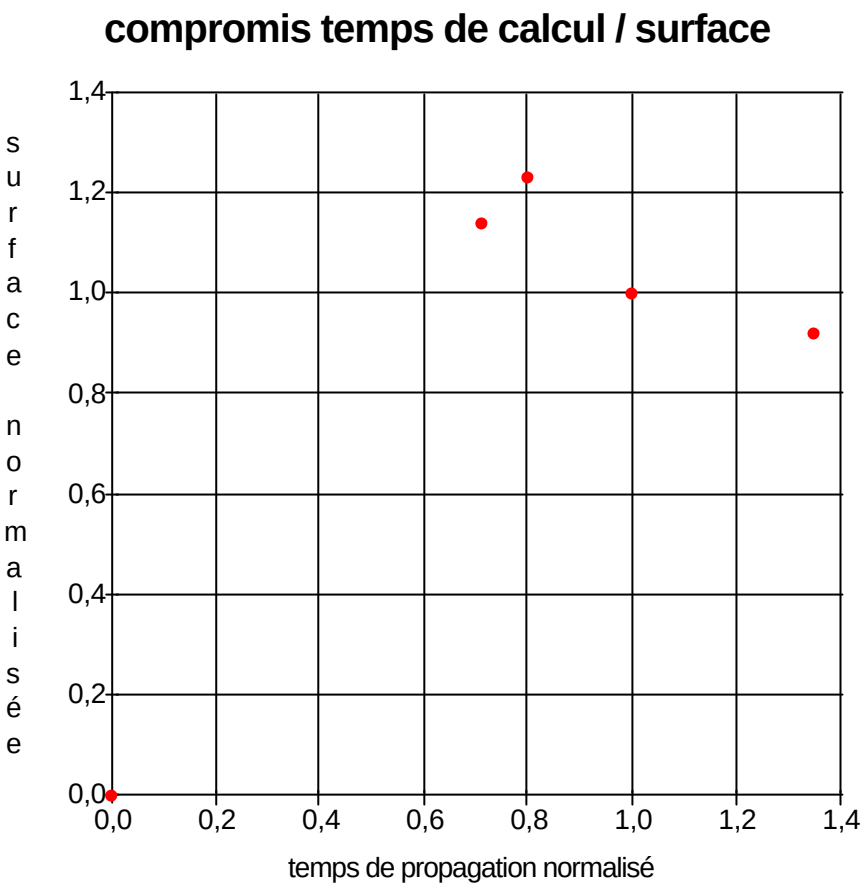
$$\frac{\text{temps de calcul du multiplieur (additionneurs à report anticipé)}}{\text{temps de calcul du multiplieur (additionneurs classique)}} \approx 43\%$$

Des études théoriques rigoureuses montrent que ce rapport est en réalité très légèrement supérieur à **50** %. On peut affirmer avec une approximation correcte que le multiplieur utilisant les additionneurs à report anticipé calcule **environ 2 fois plus vite** que celui qui utilise les additionneurs classiques.

Si on regarde le compromis vitesse de calcul / surface, l’additionneur classique donne un meilleur résultat car l’additionneur à report anticipé est trop volumineux. Par conséquent, il est nécessaire de réaliser des synthèses semi-parallèles pour obtenir un additionneur avec un meilleur compromis. C’est un des objectifs des programmes de synthèse qui réduisent le nombre de portes à partir des équations logiques proposées.

IV-4-4) Conclusion sur les multiplieurs étudiés

Le graphique suivant a été obtenu à partir des résultats précédents.



(1): Multiplieur classique
(2): Multiplieur Booth2 (référence)
(3): Multiplieur Booth3
(4): Multiplieur utilisant les additionneurs 4-2

Remarque: Le multiplieur classique n'a pas été simulé, mais les divers résultats obtenus permettent de le situer (multiplication classique = 16 codeurs simples 16 bits et 15 additionneurs classiques 16 bits).

Le meilleur compromis temps de calcul / surface est obtenu par le multiplieur utilisant les additionneurs 4-2. Le multiplieur Booth3 arrive en deuxième place devant le multiplieur Booth2 mais les deux multiplieurs possèdent des résultats très proches. Bien sûr, le multiplieur classique est moins performant.

L'utilisation des registres en sortie des additionneurs n'est pas pénalisante en surface et le temps de calcul (périodes d'horloge du tableau 2) reste du même ordre que le temps de propagation du circuit combinatoire (il faut prendre une période T telle que n*T ≈ temps de retard où n est un nombre entier). Cette structure permet également un meilleur découpage des signaux.

V) Conclusion

L'amélioration des vitesses de calcul des multiplieurs peut s'effectuer à deux niveaux. Il faut, d'une part, diminuer le nombre de produits partiels (c'est le rôle des codeurs de Booth), et, d'autre part, améliorer les temps de propagation des additionneurs. Dans ce deuxième cas, on joue sur les architectures parallèles à des niveaux hiérarchiques différents, le haut niveau correspond par exemple à la structure 4-2 et le bas niveau aux additionneurs à report anticipé. Notre étude montre que le compromis temps de retard / surface est bon pour une architecture parallèle de haut niveau (additionneurs 4-2) et moins bon pour une architecture de bas niveau (additionneurs à report anticipé trop volumineux). Elle montre aussi que le développement des algorithmes de réduction du nombre de produits partiels n'amène pas forcément un meilleur compromis (multiplieur Booth2 presque équivalent au multiplieur Booth3). Pour améliorer les performances des multiplieurs, on peut, par exemple, compte tenu des remarques précédentes, s'intéresser à un multiplieur Booth2 avec une architecture 4-2 qui utilise des additionneurs ayant une structure semi-parallèle. Un autre critère important pour les performances des multiplieurs est la consommation. Ce critère n'a pas été évoqué dans notre étude, car il aurait fallu concevoir un circuit intégré par multiplieur. De plus, même si l'architecture du multiplieur joue un rôle important pour la détermination de ce critère, la technologie choisie peut être également un paramètre déterminant. Cette remarque reste valable pour les temps de propagation et les nombres de portes logiques.

Notre étude a montré également que la conception des circuits intégrés fait appel à des logiciels de synthèse qui optimisent les paramètres de temps de retard et de surface. La personne qui conçoit le multiplieur (où n'importe quel ASIC) n'a plus réellement besoin de se préoccuper des architectures. Il suffit de décrire, à l'aide plusieurs équations logiques et d'une structure globale arbitraire le fonctionnement du multiplieur. Et le logiciel optimise la synthèse, en lui demandant, par exemple, de respecter des architectures de haut ou de bas niveau avant les phases d'optimisation. La connaissance des processus d'optimisation du logiciel est donc primordiale si on souhaite appliquer une stratégie de conception et maîtriser les résultats obtenus après le processus de synthèse.