Arithmétique redondante

Nous présentons ici l'arithmétique redondante et l'arithmétique mixte. Cette présentation est brève, car ne faisant pas parti de cette thèse. Elle a pour but d'expliquer les principes fondamentaux de ces arithmétiques de manière à mieux comprendre l'intérêt de leur utilisation. Le lecteur intéressé par plus de détails pourra se reporter à la thèse de Yannick Dumonteix [Dum01]. Nous montrons ensuite les performances des différents opérateurs arithmétiques, avant de nous intéresser à différentes architectures qui ont été mises en œuvre en utilisant l'arithmétique redondante.

2.1 Généralités

2.1.1 Représentations

L'arithmétique redondante doit son nom au fait qu'il est possible d'affecter plusieurs codages à un même nombre [Avi61]. Elle est constituée de deux représentations.

La représentation Carry-Save

En base 2, les chiffres sont des éléments de 0, 1, 2 et leur codage, sur deux bits de même poids, est tel que $cs_i = cs_i^0 + cs_i^1$. Un nombre Carry-Save est donc codé sur deux termes CS^0 et CS^1 tel que le terme CS en notation classique est : $CS = CS^0 + CS^1$. On parle d'addition non effectuée.

Cette notation permet de représenter des nombres non signés (CS_{NS}) avec des termes sont codés en notation simple de position, et des nombres signés (CS_S) avec des termes en complément à deux :

$$CS_{NS} = \sum_{i=0}^{i=N-1} cs_i^0 \cdot 2^i + \sum_{i=0}^{i=N-1} cs_i^1 \cdot 2^i$$

$$CS_S = -(cs_i^0 + cs_i^1) \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} cs_i^0 \cdot 2^i + \sum_{i=0}^{i=N-2} cs_i^1 \cdot 2^i$$

La représentation Borrow-Save

En base 2, les chiffres sont des éléments de -1, 0, 1 et leur codage, sur deux bits de même poids, est tel que $bs_i = bs_i^+ + bs_i^-$. Un nombre Borrow-Save est donc codé sur deux termes BS^+ et BS^- tel que le terme BS en notation classique est : $BS = BS^+ - BS^-$. On parle de soustraction non effectuée.

Le principe est le même que pour la notation Carry-Save : de la même façon, un nombre Borrow-Save permet de représenter des nombres non signés (BS_{NS}) et des nombres signés (BS_S) :

$$BS_{NS} = \sum_{i=0}^{i=N-1} bs_i^+ \cdot 2^i - \sum_{i=0}^{i=N-1} bs_i^- \cdot 2^i$$

$$BS_S = (-bs_{N-1}^+ \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} bs_i^+ \cdot 2^i) - (-bs_{N-1}^- \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} bs_i^- \cdot 2^i)$$

2.1.2 Arithmétique mixte

Un chemin de données arithmétiques ne se réduit pas à des opérateurs arithmétiques, il contient également des opérateurs non arithmétiques, comme les multiplexeurs ou les opérateurs booléens par exemple. Ces opérateurs doivent conserver les signaux à leur interface en arithmétique classique, c'est pourquoi l'utilisation de l'arithmétique redondante seule dans un circuit s'avère illusoire. De plus l'interface du chemin de données ne doit pas non plus être modifiée.

L'idée qui a été émise est donc d'utiliser conjointement l'arithmétique redondante et l'arithmétique classique. Ainsi, les opérateurs arithmétiques peuvent avoir des entrées/sorties aussi bien en représentation redondante qu'en représentation classique. Cela permet de traiter les signaux pour lesquels la représentation classique est nécessaire.

C'est ce que nous appelons **l'arithmétique mixte**.

Utiliser conjointement ces deux arithmétiques nécessite donc :

1. d'avoir des opérateurs qui acceptent toutes les combinaisons de représentations possibles en interface : $NR\ op\ NR \rightarrow NR$, $NR\ op\ NR \rightarrow R$, $R\ op\ R \rightarrow NR$, $R\ op\ R \rightarrow R$, $NR\ op\ R \rightarrow NR$, $NR\ op\ R \rightarrow R$,
2. d'avoir les convertisseurs nécessaires au passage d'une notation à une autre : en pratique, cela s'avère assez simple étant donné que passer de la notation Carry-Save à une notation classique se fait en additionnant les deux termes du nombre Carry-Save avec un additionneur classique, et passer d'une notation Borrow-Save à une notation classique se fait de même avec un opérateur de soustraction.

2.2 Architectures

2.2.1 Addition

Arithmétique classique

L'addition est l'opération la plus usitée dans la conception de chemins de données arithmétiques. De nombreuses architectures ont donc été étudiées en arithmétique classique de façon à la rendre la plus rapide possible [Mul89], nous n'en présentons ici que deux, ayant des performances surface/temps opposées :

Additionneur à propagation de retenue séquentielle L'algorithme employé est l'algorithme intuitif et purement séquentiel d'une addition *à la main*, comme montré dans la Figure 2.1. On parlera de *Carry Ripple Adder*. La complexité de la surface et du délai de cet additionneur sont en $O(N)$ (N étant la dynamique des nombres à additionner).

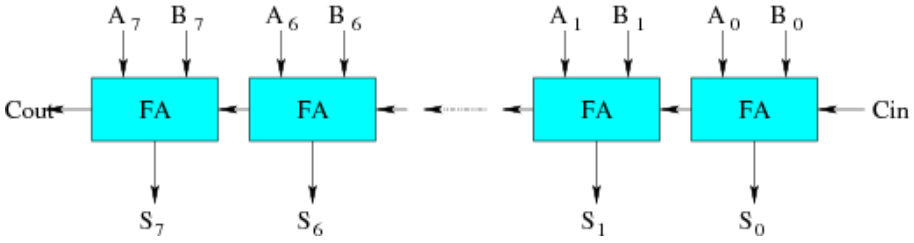


FIG. 2.1 – Additionneur séquentiel

Le délai important de cette architecture est dû à la propagation de la retenue, qui est réalisée de façon purement séquentielle. Les autres architectures d'additionneurs ont donc pour but de réduire ce temps de propagation.

Additionneur de Sklansky Cet additionneur, aussi appelé CLA pour *Carry Lookahead Adder*, fait partie des additionneurs à retenue anticipée. Le principe est d'anticiper la propagation de la retenue de façon à s'affranchir de l'attente du calcul de l'addition du bit précédent. L'architecture correspondante est montrée dans la Figure 2.2. C'est une structure ayant une complexité de $O(Nlog_2(N))$ en surface et $O(log_2(N))$ en délai.

La chaine critique a donc pu être améliorée par rapport au *Ripple*, mais au détriment de la surface.

Soustraction Il n'y a pas d'architecture spécifique pour la soustraction en arithmétique classique. Effectuer une soustraction se fait en additionnant à un nombre le complément à deux du nombre à soustraire : $A - B = A + not(B) + 1$. En pratique, un additionneur est utilisé, avec une série d'inverseurs pour obtenir le complément à un du nombre à soustraire et la retenue entrante de l'additionneur positionnée à 1.

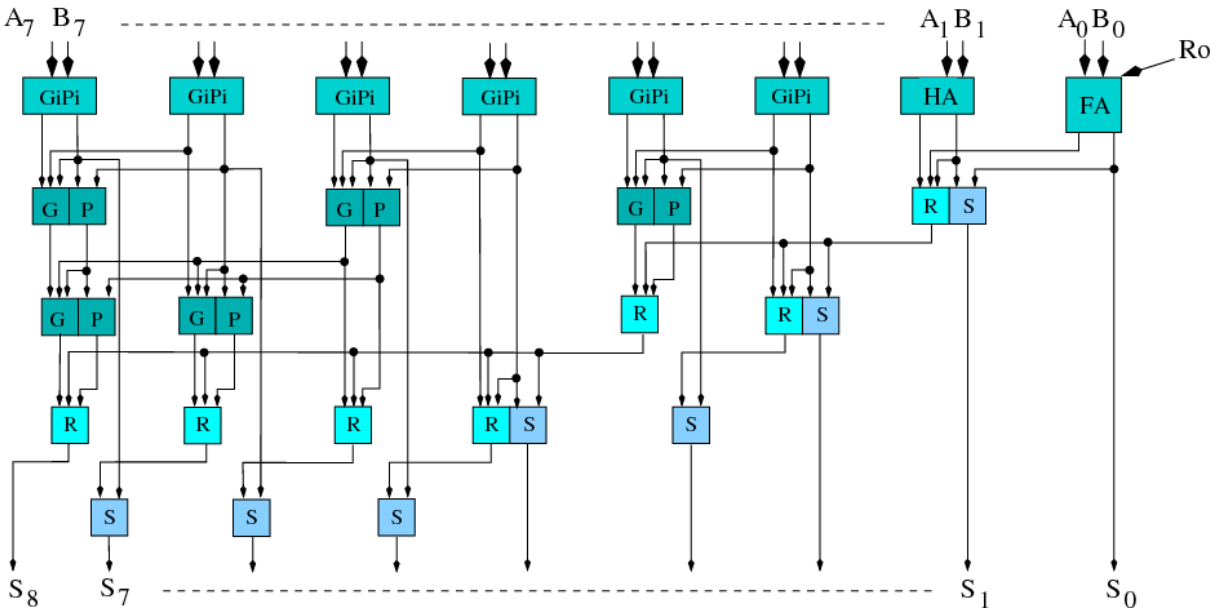


FIG. 2.2 – Additionneur de Sklansky

Arithmétique redondante

L'additionneur redondant effectue l'addition de deux nombres redondants et fournit un résultat en redondant.

Addition Carry-Save La Figure 2.3 représente l'architecture de l'additionneur $CS + CS \rightarrow CS$. Cet additionneur a une complexité de $O(N)$ en surface et $O(1)$ en délai.

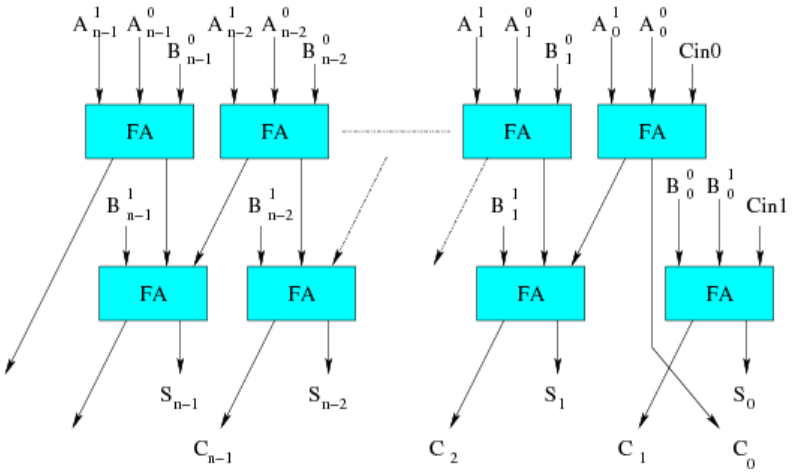


FIG. 2.3 – Additionneur redondant

Plus précisément, sa surface est de l'ordre de grandeur de deux fois celle du *Ripple*, l'additionneur "le plus petit" en arithmétique classique, et il est en temps constant, quelle que soit la dynamique de ses entrées (temps correspondant à la traversée de deux étages d'additionneurs complets - noté *FA* pour *full-adder*).

En effet, le fait qu'un nombre Carry-Save soit la somme de deux nombres entraîne que l'on propage en parallèle la somme et la retenue. On s'affranchit donc de la propagation de retenue dans le calcul.

Addition Borrow-Save L'utilisation de la notation Borrow-Save permet de réaliser les soustractions avec des architectures redondantes. Un additionneur redondant effectue soit l'addition de deux nombres Borrow-Save : $BS_0 + BS_1 = BS_0^+ - BS_0^- + BS_1^+ - BS_1^-$, soit l'addition d'un nombre Borrow-Save et d'un nombre Carry-Save : $BS + CS = BS^+ - BS^- + CS^0 + CS^1$.

Deux types d'architectures existent pour ces différentes opérations : soit avec une sortie en représentation Carry-Save, soit avec une sortie en représentation Borrow-Save :

- pour obtenir une sortie Carry-Save, le principe est le même qu'en arithmétique classique : une série d'inverseurs pour chaque nombre Borrow-Save en entrée, et une ou deux des retenues d'entrée positionnée à '1' (cf. exemple de l'addition de deux nombres Borrow-Save dans la Figure 2.4),

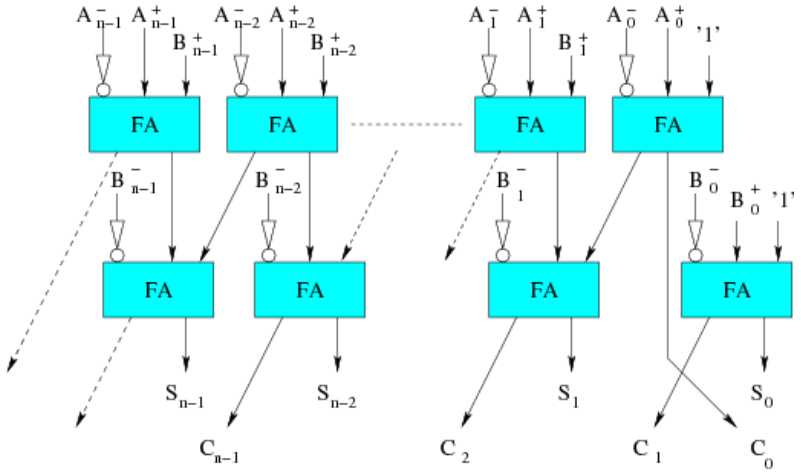
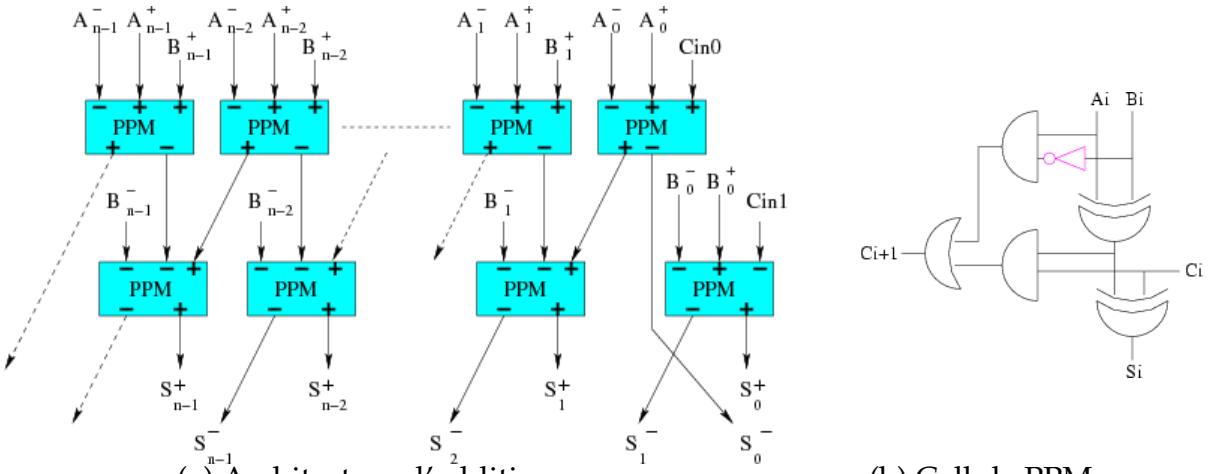


FIG. 2.4 – Additionneur redondant Borrow-Save avec sortie Carry-Save

- pour obtenir une sortie Borrow-Save, l'architecture est une nouvelle fois identique à celle pour le Carry-Save, à ceci près qu'il faut modifier la cellule de base qu'est le *FA* en une cellule dite *plus-plus-minus* (notée *PPM*, cf. Figure 2.5.b) [JDK91] (cf. exemple de l'addition de deux nombres Borrow-Save dans la Figure 2.5.a).



(a) Architecture d'additionneur (b) Cellule PPM

FIG. 2.5 – Additionneur redondant Borrow-Save avec sortie Borrow-Save

Arithmétique mixte

L'additionneur mixte effectue l'addition d'un nombre redondant et d'un nombre non redondant et fournit un résultat en redondant.

Addition Carry-Save La Figure 2.6 représente l'architecture de l'additionneur mixte $CS + NR \rightarrow CS$. Cet additionneur a une complexité de $O(N)$ en surface et $O(1)$ en délai.

Il est naturellement lui aussi en temps constant (temps correspondant à la traversée d'un étage de *FA*) et sa surface est identique à celle d'un *Ripple*.

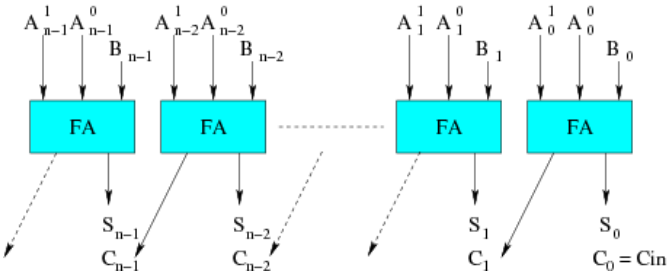


FIG. 2.6 – Additionneur mixte

Remarque :

Là où un additionneur redondant réduit le nombre d'opérandes de 4 à 2, un additionneur mixte le réduit de 3 à 2. Un additionneur mixte représente ainsi la moitié du calcul d'un additionneur redondant. En d'autres termes, un additionneur redondant peut être considéré comme la concaténation de deux additionneurs mixtes.

Addition Borrow-Save En utilisant la représentation Borrow-Save, un additionneur mixte effectue l'addition entre un nombre Borrow-Save et un nombre en complément à 2 : $BS + NR = BS^+ - BS^- + NR$.

Là aussi, deux architectures différentes existent en fonction de la représentation de la sortie de l'additionneur. Le principe est identique à celui de l'additionneur redondant, comme montré dans les Figures 2.7 et 2.8.

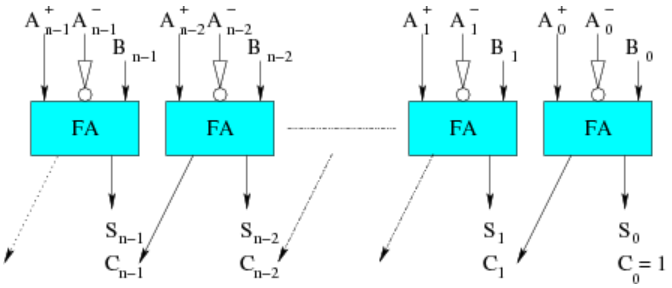


FIG. 2.7 – Additionneur mixte Borrow-Save avec sortie Carry-Save

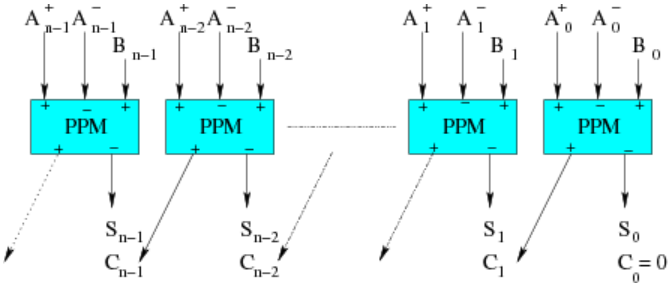


FIG. 2.8 – Additionneur mixte Borrow-Save avec sortie Borrow-Save

Synthèse

Les architectures d'additionneurs redondants et mixtes allient une petite taille (comparable à celle du plus petit additionneur classique) et un temps de propagation constant. On en déduit l'atout majeur de l'arithmétique redondante : **l'addition de deux nombres se fait en temps constant, quelle que soit la taille des opérandes à sommer**. L'addition étant un opérateur fondamental, le gain potentiel est important.

2.2.2 Multiplication

Dans ce qui suit, nous présentons les différentes étapes nécessaires pour effectuer une multiplication. Ces étapes nous servent à montrer l'impact de l'utilisation de l'arithmétique redondante sur la multiplication. Nous ne présentons pas en détail l'architecture des différentes étapes, les lecteurs intéressés peuvent se référer à [DM00, Dum01, AFT02].

Arithmétique classique

Tout comme pour l'addition, un algorithme intuitif de multiplication par additions / décalages conduit à une architecture ayant un mauvais délai (complexité en $O(N)$).

L'idée émise de façon à minimiser ce délai consiste à augmenter le nombre de 0 du nombre multiplicateur en le recodant dans une base plus grande, c'est le recodage de *Booth modifié*. Ce recodage permet de diviser globalement par deux le nombre de produits partiels.

Dans la Figure 2.9, nous pouvons voir les différentes étapes nécessaires à la réalisation d'une telle multiplication : le recodage du multiplicateur (délai en $O(1)$), le calcul des produits partiels (délai en $O(1)$), puis la somme des produits partiels, elle même décomposée en un arbre de Wallace (délai en $O(log_{3/2}(N/2))$) et un additionneur final de *Sklansky* (délai en $O(log_2(N))$). Cet algorithme permet donc d'obtenir des multiplieurs arborescents ayant une complexité en délai en $O(log_{3/2}(N/2))$.

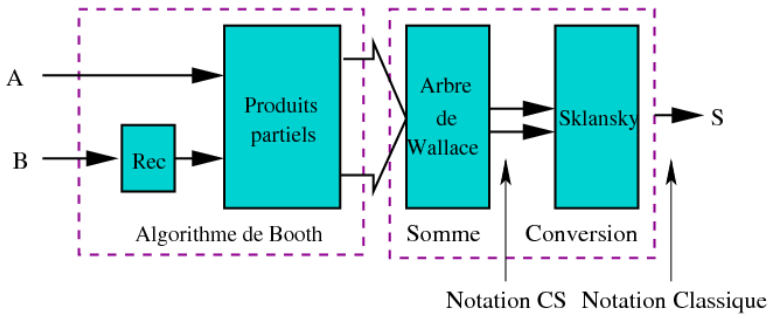


FIG. 2.9 – Struture d'un multiplieur classique

Remarque :

Nous avons déjà évoqué le fait que l'arithmétique redondante a tout d'abord été utilisée dans l'architecture interne des multiplieurs. Comme on peut le voir, un arbre de Wallace est utilisé pour effectuer la somme des produits partiels. Or, son rôle est de diminuer le nombre d'opérandes à sommer de n à 2 : il fournit un résultat en représentation Carry-Save. C'est pourquoi dans l'architecture classique d'un multiplieur, il est suivi d'un additionneur, qui a pour rôle de convertir le nombre Carry-Save obtenu en un nombre en représentation classique. Cette particularité de l'architecture du multiplieur montre d'ores et déjà l'intérêt de permettre l'utilisation de la notation Carry-Save : elle permet la suppression du convertisseur final.

Arithmétique redondante

Multiplication Carry-Save La multiplication de deux nombres *CS* consiste en l'opération suivante : $CS_0 * CS_1 = (CS_0^0 + CS_0^1) * (CS_1^0 + CS_1^1)$. Le résultat est fourni en représentation Carry-Save.

La Figure 2.10 représente les modifications à apporter à l'architecture classique de façon à utiliser les notations redondantes. L'architecture présentée est de type *direct*, dans laquelle il n'est pas effectué un recodage de Booth. Une étape de recodage est cependant nécessaire pour prendre en compte deux entrées en format Carry-Save. En effet, le calcul des produits partiels ne peut gérer les cas où un chiffre égal à 2 est présent à la fois dans les deux opérandes. L'étape de recodage sert donc à modifier les propriétés des entrées de façon à éviter ce cas.

L'étape de produits partiels est également différente de la version classique étant donné qu'elle prend en compte deux nombres redondants en entrée i.e. l'équivalent de quatre nombres classiques (au lieu de deux pour l'architecture classique).

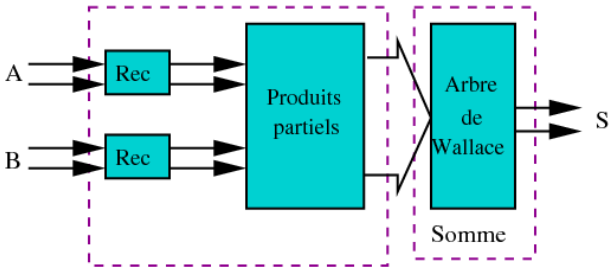


FIG. 2.10 – Struture d'un multiplieur redondant

Multiplication Borrow-Save Un multiplieur redondant effectue soit la multiplication de deux nombres *BS* : $BS_0 * BS_1 = (BS_0^+ - BS_0^-) * (BS_1^+ - BS_1^-)$, soit la multiplication d'un nombre *BS* et d'un nombre *CS* : $BS * CS = (BS^+ - BS^-) * (CS^0 + CS^1)$.

D'un point de vue architecture, seule l'étape de recodage est modifiée, en fonction de la représentation de chaque entrée. En effet, on profite du fait qu'un recodage est de toute façon nécessaire pour effectuer les produits partiels pour recoder la ou les entrées Borrow-Save en Carry-Save. De ce fait, la construction des produits partiels (puis la sommation) reste la même. La sortie est donc en représentation Carry-Save.

Arithmétique mixte

Multiplication Carry-Save La multiplication entre un nombre *CS* et un nombre *NR* consiste en l'opération suivante : $CS * NR = (CS^0 + CS^1) * NR$.

Les modifications à apporter à l'architecture classique se situent au niveau du calcul des produits partiels, de façon à prendre en compte un nombre Carry-Save et un nombre classique i.e. l'équivalent de trois nombres, comme montré dans la Figure 2.11. Comme on peut le voir, cette architecture ne nécessite pas d'étape de recodage. En effet, l'étape de recodage servait à s'assurer, dans la multiplication redondante, qu'il n'y aurait jamais deux nombres égaux à 2 en même temps lors de la phase de produits partiels, cas qui n'est de toute façon pas possible dans le cas d'un nombre redondant et d'un nombre classique.

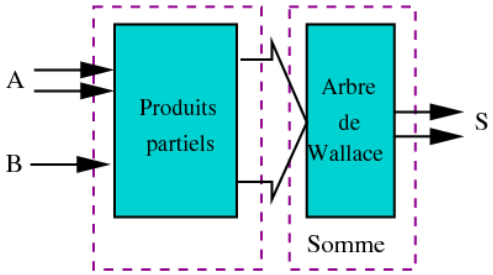


FIG. 2.11 – Struture d'un multiplieur mixte

Multiplication Borrow-Save La multiplication entre un nombre *BS* et un nombre *NR* consiste en l'opération suivante : $BS * NR = (BS^+ - BS^-) * NR$.

Les modifications à apporter à l'architecture classique se situent au même niveau que pour la multiplication Carry-Save.

Synthèse

Comme nous l'avons déjà souligné, le fait qu'un multiplieur fournisse un résultat en représentation Carry-Save se traduit par la suppression de l'additionneur final de conversion dans son architecture. Cependant, plus un multiplieur a d'entrées redondantes, plus la phase de calcul des produits partiels est coûteuse, principalement quand il y a deux entrées redondantes, car un recodage des entrées redondantes est alors nécessaire. L'idéal est donc un multiplieur dont les entrées sont en représentation classique et la sortie en représentation redondante.

A un niveau unitaire, l'intérêt des multiplieurs redondant et mixte est moins évident que pour les additionneurs. Il se fera ressentir principalement avec l'enchaînement des opérateurs.

2.3 Performances

Dans cette partie, nous présentons un récapitulatif des résultats que nous avons obtenus pour les additionneurs et multiplieurs mixtes et redondants, comparées à ceux de ces mêmes opérateurs en arithmétique classique, en terme de surface et de délai. Cette évaluation a pour objectif de montrer l'intérêt de telles architectures.

Les opérateurs évalués ont été décrits avec le langage **Stratus** et font partie de la bibliothèque **ArithLib**, tous deux développés durant cette thèse et qui seront étudiés au Chapitre 5.

L'évaluation se fait avec emploi de la bibliothèque de cellules précaractérisées de la chaîne **Alliance** [GP92] (en 0.35μm) et les outils de placement/routage de la chaîne de *CAO* Cadence : **Encounter**.

2.3.1 Addition

Les Figures 2.12 et 2.13 présentent les performances des additionneurs classiques, redondants et mixtes. Les additionneurs utilisés sont les additionneurs mixtes et redondants (avec les différentes représentations possibles *CS* et *BS* en entrées / sorties), ainsi que deux additionneurs classiques : le *Sklansky*, qui est la référence en matière de rapidité pour les additionneurs classiques, et le *Ripple*, qui est le plus petit additionneur classique. Pour chaque additionneur, plusieurs évaluations sont faites en faisant varier la dynamique des entrées (de 4 à 128 bits).

Les résultats présentés montrent que les performances obtenues pour les opérateurs d'addition sont en accord avec ce que les architectures présageaient, les additionneurs mixtes et redondants sont :

- **petits** :
  - l'ordre de grandeur est celui du *Ripple* pour les additionneurs mixtes avec sortie Carry-Save, et de deux fois cette surface pour les additionneurs redondants avec sortie Carry-Save,
  - les additionneurs mixtes et redondants avec sorties Borrow-Save sont un peu plus grands que leur équivalent avec sortie Carry-Save, mais restent plus petits que le *Sklansky*,

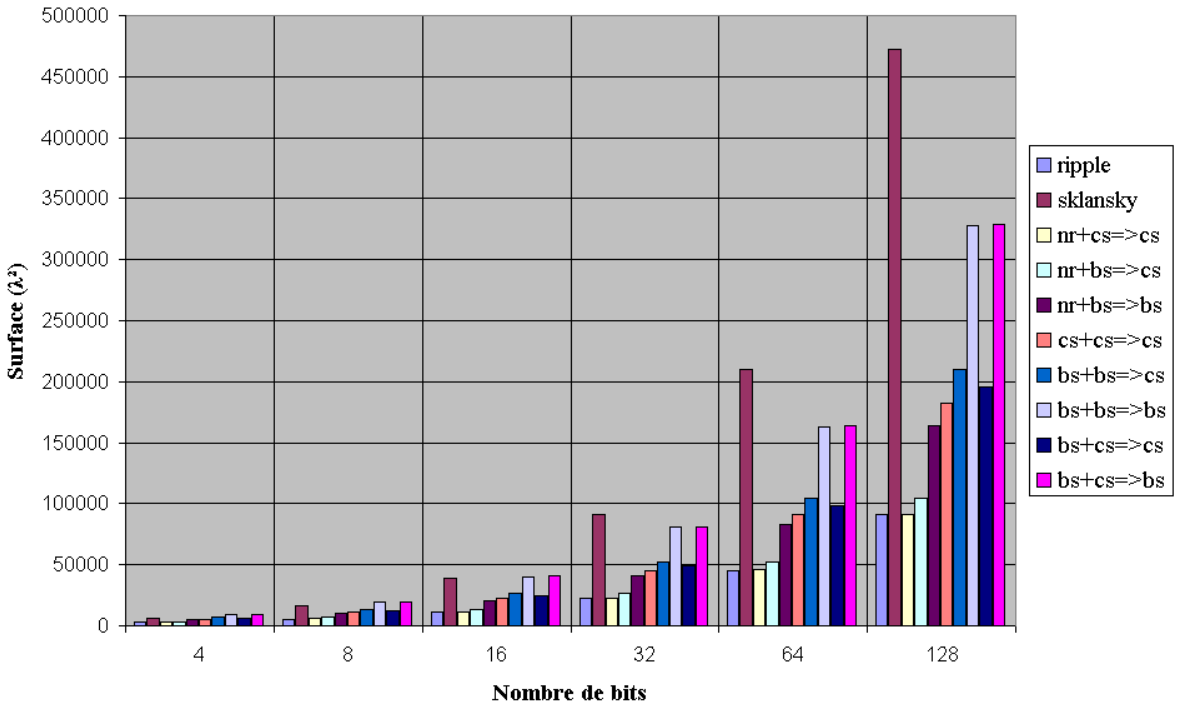


FIG. 2.12 – Comparaison en surface des additionneurs

- **rapides et en temps constant** :
  - les additionneurs mixtes sont systématiquement plus rapides que le *Sklansky*,
  - les additionneurs redondants le sont également, quand la dynamique des entrées est supérieure ou égale à 8 bits,
  - le temps de propagation des additionneurs redondants est environ deux fois celui des additionneurs mixtes.

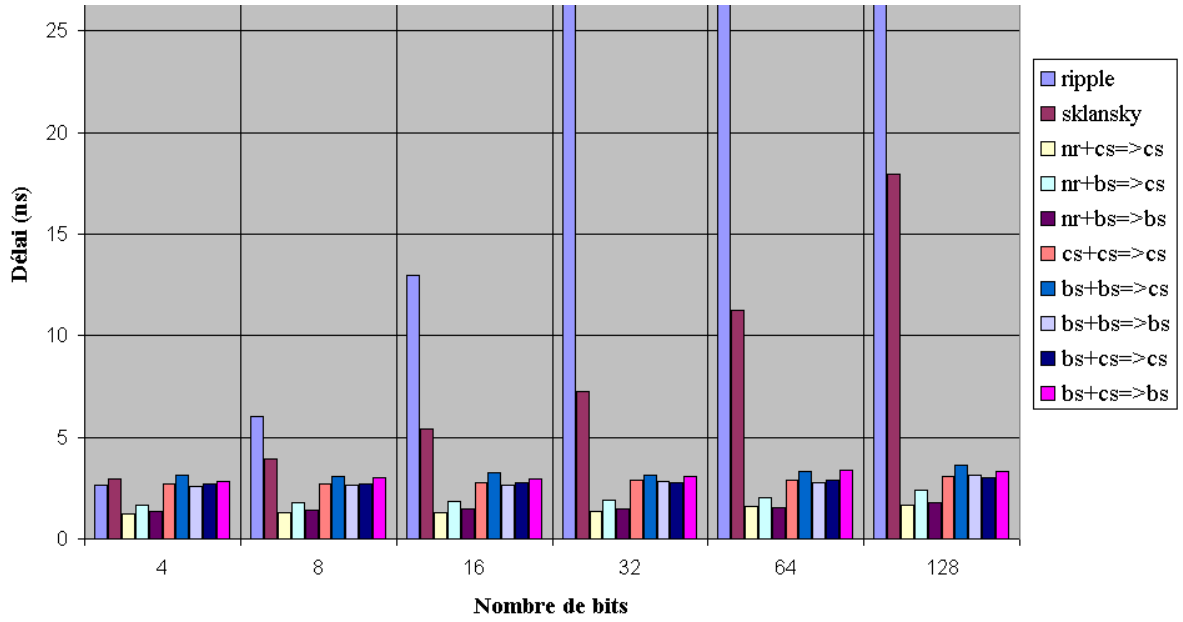


FIG. 2.13 – Comparaison en délai des additionneurs

Remarques :

1. Les données en délai du *Ripple* sont tronquées à partir de 32 bits dans le graphique, étant donné les mauvaises performances de cet additionneur, sur 128 bits, son délai est par exemple de 304 ns, valeur qui aurait rendue le graphique peu lisible.
2. Les performances de l'additionneur redondant sont d'autant plus importantes que, là où le *Sklansky* diminue le nombre d'opérandes de 2 à 1, l'additionneur redondant le diminue de 4 à 2, soit potentiellement deux fois le même travail.
3. La différence de taille entre les additionneurs avec sortie Carry-Save et ceux avec sortie Borrow-Save est principalement dûe au fait que la cellule dite *PPM* (cf. Figure 2.5.b) n'existe pas dans la bibliothèque de cellules utilisée, contrairement au *FA*, les différentes portes la composant ont donc été utilisées, amenant à une cellule de base moins optimisée. Il est envisageable de créer cette cellule de base de façon à obtenir de meilleurs résultats.

2.3.2 Multiplication

Les Figures 2.14 et 2.15 montrent les mêmes expérimentations que précédemment, mais avec la multiplication. Les multiplieurs comparés sont : le classique, le mixte et le redondant (différentes représentations en entrées / sorties pour les deux derniers), le classique étant avec utilisation de l'algorithme de Booth, et les deux autres avec utilisation de l'algorithme dit direct [Dum01]. Comme précédemment, plusieurs évaluations sont faites en faisant varier la dynamique des entrées (de 4 à 24 bits).

Les résultats présentés sont là aussi en accord avec ce que nous avions déduit de l'architecture des multiplieurs. Les multiplieurs redondants et mixtes sont :

- **gros** :
  - les multiplieurs mixtes sont légèrement plus petits que le multiplieur classique,
  - les multiplieurs redondants sont eux un peu plus gros, à cause des phases de recodage des entrées redondantes et de calcul des produits partiels,

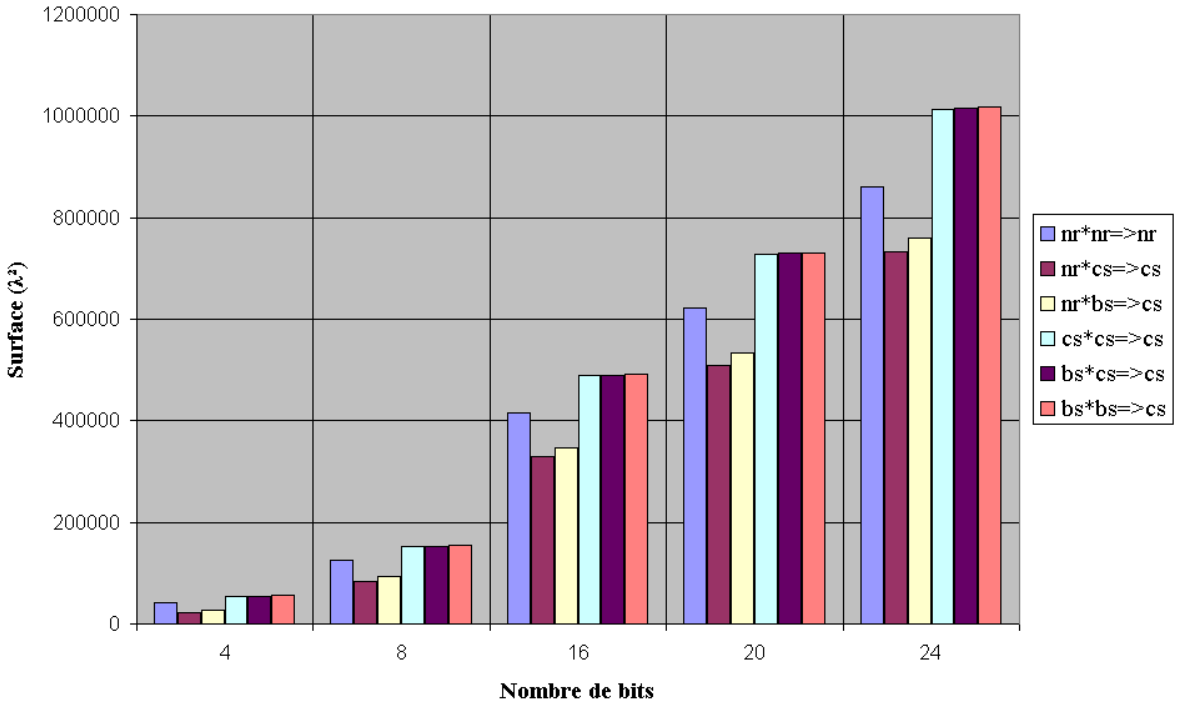


FIG. 2.14 – Comparaison en surface des multiplieurs

- **rapides** :
  - ils sont tous plus rapides que le multiplieur classique.

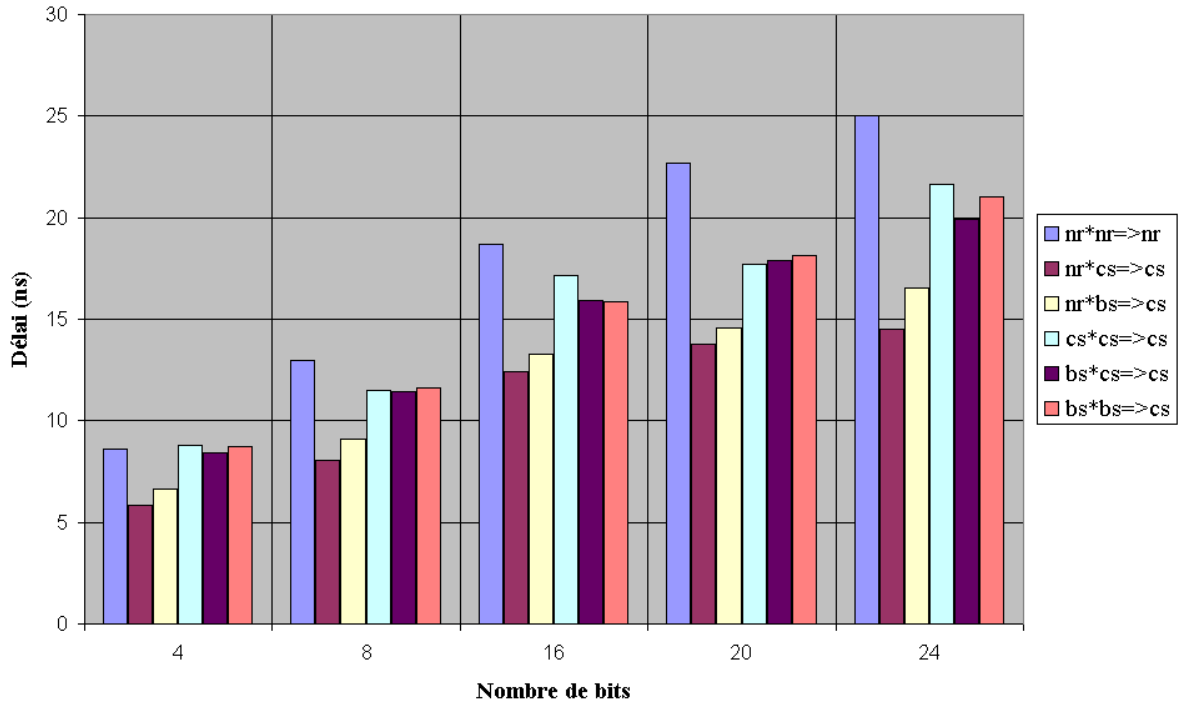


FIG. 2.15 – Comparaison en délai des multiplieurs

2.4 Applications

Au-delà des bonnes performances intrinsèques des opérateurs redondants et mixtes, c'est le chaînage d'opérations arithmétiques qui permet d'exploiter au mieux ces architectures de par la suppression des conversions. La meilleure façon de montrer l'intérêt de l'utilisation de l'arithmétique redondante dans la conception de chemins de données arithmétiques est donc d'utiliser *à la main* cette arithmétique dans la réalisation de différentes architectures [VBG+94, LCSJJ03, CMRS04]. Nous détaillons ici plusieurs résultats extraits de la littérature et obtenus de cette façon.

2.4.1 PGCD

Plusieurs algorithmes existent de façon à trouver le *PGCD* et le *PGCD* étendu[1]. Deux grandes approches existent : algorithmes utilisant le bit de poids faible des nombres et ceux utilisant le bit de poids fort. Dans la première approche, des architectures classiques sont utilisées, tandis que des architectures redondantes sont utilisées dans la seconde. Ces deux approches sont comparées dans [Guy91], elles-mêmes divisées en deux sous approches : additions / soustractions effectuées en série (comparaison pour un algorithme) ou en parallèle (comparaison pour quatre algorithmes).

[1]Calcul des deux entiers vérifiant l'égalité de Bézout i.e. s et t tels que $a * s + b * t = pgcd(a, b)$

	bits / cycle			transistors / bit		
	Classique	Redondant	%	Classique	Redondant	%
Série (1 algo.)	0.6	1	+67%	75	240	+220%
Parallèle (4 algo.)	0.6 à 0.95	0.95 à 1.05	jusqu'à +75%	140 à 220	140 à 150	jusqu'à -36%

TAB. 2.1 – PGCD : Résultats

	bits / cycle			transistors / bit		
	Classique	Redondant	%	Classique	Redondant	%
Série (1 algo.)	-	0.48	-	-	320	-
Parallèle (4 algo.)	0.22 à 0.32	0.46 à 0.50	jusqu'à +127%	265 à 305	155 à 165	jusqu'à -49%

TAB. 2.2 – PGCD étendu : Résultats

Les résultats présentés dans les Tableaux 2.1 et 2.2 proviennent de simulations et font une estimation du délai en considérant le ratio bits par cycle, et de la surface avec le ratio nombre de transistors par bit.

Ils montrent que les architectures utilisant l'arithmétique redondante ont de meilleurs ratios bits / cycle (jusqu'à +127%) couplés à de meilleurs ratios transistors / bit (jusqu'à -49%) sauf dans le cas de l'algorithme série du *PGCD* (+220%).

2.4.2 DCT

Un macro-générateur effectuant la *DCT* 2 dimensions a été réalisé dans [DCM00] (architecture décrite dans le Chapitre 6), avec comme paramètres les tailles des données et des coefficients ainsi que le nombre d'étages de pipeline. Deux implantations de ce macro-générateur ont été réalisées : avec ou sans pipeline, et avec les entrées sur 8 bits, les coefficients sur 12 bits et les sorties sur 16 bits.

Les performances de ces implantations sont présentées après placement / routage et comparées à celles d'une implantation classique.

	Surface (mm^2)			Fréquence (MHz)		
	Classique	Redondant	%	Classique	Redondant	%
sans pipeline	0.37	0.36	-2.7%	36	45	+25%
3 étages de pipeline	0.38	0.41	+7.9%	75	116	+54.7%

TAB. 2.3 – DCT : Résultats

Les résultats présentés dans le Tableau 2.3 montrent qu'il n'y a pas d'amélioration significative d'un point de vue de la surface, mais une amélioration importante du

point de vue du délai, +25% pour la fréquence pour la version sans pipeline et +54.7% pour la version avec pipeline.

2.4.3 DCU

L'implantation d'une *DCU* a été réalisée avec un opérateur de carré redondant (amélioration en temps de 10 à 20% par rapport à l'opérateur classique) dans [DBM01] (architecture décrite dans le Chapitre 6). Plusieurs implantations ont été effectuées, en fonction de la dynamique des entrées : de 8 à 32 bits.

Les performances sont ici aussi présentées après placement / routage et comparées à celles d'une implantation classique.

Nombre de bits	Surface (μm^2)			Délai (ns)		
	Classique	Redondant	%	Classique	Redondant	%
8	6259	7166	+14.5%	10.7	9.9	-7.5%
16	19293	22491	+16.6%	14.3	13.0	-9.1%
24	44222	51480	+16.4%	16.9	14.8	-12.4%
32	86350	99849	+15.6%	20.4	17.5	-14.2%

TAB. 2.4 – DCU : Résultats

Les résultats présentés dans le Tableau 2.4 se résument à une amélioration du délai et un surcoût de la surface : par exemple sur 32 bits, -14.2% en délai et +15.6% en surface. On peut en déduire que, pour ce circuit, les produits surface.délai ne sont pas améliorés en utilisant l'arithmétique redondante. Cela ne diminue cependant pas l'intérêt de l'usage de l'arithmétique redondante si l'on considère que le critère primordial à optimiser est le délai.

2.5 Conclusion

Dans ce chapitre nous avons présenté les caractéristiques de l'arithmétique redondante. Nous avons montré que l'usage de cette arithmétique seule dans la conception de circuits s'avère illusoire, nous avons donc présenté une arithmétique étant la combinaison des arithmétiques redondante et classique : l'arithmétique mixte.

Nous avons présenté les architectures des différents opérateurs d'addition et de multiplications utilisant cette arithmétique, ainsi que leurs performances. Le point crucial est que l'addition en arithmétiques redondante/mixte est exempte de toute propagation de retenue, elle se fait donc en temps constant quelque soit la dynamique des opérandes à sommer.

Suite aux bonnes performances intrinsèques des opérateurs présentés, nous avons fait un état de l'art des différents blocs arithmétiques réalisés à la main avec cette arithmétique. Les résultats présentés sont très encourageants et montrent l'intérêt d'outils d'optimisation automatiques utilisant l'arithmétique mixte.