THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

Spécialité Informatique

Présentée par Yannick DUMONTEIX

Pour obtenir le grade de

Docteur de L'Université Paris VI

OPTIMISATIONS DES CHEMINS DE DONNÉES ARITHMÉTIQUES PAR L'UTILISATION DE PLUSIEURS SYSTÈMES DE NUMÉRATION

Soutenue le 10 octobre 2001, devant le jury composé de :

Alain Guyot	Rapporteur
Marc Daumas	Rapporteur
Michel Robert	Examinateur
Jean-Marie Chesneaux	Examinateur
Alain Greiner	Examinateur
Habib Mehrez	Directeur de thèse

À Fabienne,
à Cassandre et Andreas,
à eux et à tous ceux pour qui j'ai de l'affection.

« Entre
ce que je pense,
ce que je veux dire,
ce que je crois dire,
ce que je dis,
ce que vous avez envie d'entendre,
ce que vous croyez entendre,
ce que vous entendez,
ce que vous avez envie de comprendre,
ce que vous comprenez,
il y a dix possibilités qu'on ait des difficultés à communiquer.
Mais essayons quand même ... »

Le père de nos pères
et *L'encyclopédie du savoir relatif et absolu*
Bernard Werber

Avant-Propos

Je souhaite exprimer toute ma reconnaissance au professeur Alain GREINER, pour m'avoir accueilli au sein du thème ASIM du laboratoire LIP6 de l'université Pierre et Marie Curie, et pour l'intérêt qu'il a porté au déroulement de mes travaux.

Je remercie le professeur Habib MEHREZ de l'université Pierre et Marie Curie, d'avoir dirigé mes recherches, et de son aide durant la rédaction de ce mémoire. Je tiens également à lui exprimer ma gratitude pour la confiance et l'amitié qu'il m'a témoignées ainsi que pour sa gentillesse légendaire.

Je tiens à remercier monsieur Alain GUYOT, maître de conférence à l'institut National Polytechnique de Grenoble et monsieur Marc DAUMAS, chargé de recherches au CNRS à l'École Normale Supérieure de Lyon, pour l'honneur qu'ils m'ont fait en acceptant de participer au jury de cette thèse, et pour le soin qu'ils ont apporté à l'examen de ce document.

De même je tiens à remercier monsieur Michel ROBERT, professeur au Laboratoire d'Informatique, de Robotique et de Microélectronique de l'université de Montpellier II, le professeur Jean-Marie Chesneaux Directeur des études à l'Institut de Sciences et Technologie de l'Université Pierre et Marie Curie, et de nouveau le professeur Alain GREINER, d'avoir accepté d'être examinateurs de cette thèse.

Je profite de cette occasion pour témoigner ma reconnaissance à l'ensemble des personnes qui ont contribué de près ou de loin à mes travaux de thèse, tout spécialement à Nicolas VAUCHER, à Frédéric PÉTROT et Ludovic JACOMME pour leurs compétences multiples, et à Roselyne CHOTIN et Arnaud MONTREUIL, anciens étudiants du DEA ASIME, dont les stages m'ont été précieux.
Je remercie aussi tous les membres du pôle ASIM du Laboratoire LIP6 pour leur sympathie et particulièrement Mademoiselle Marie-Catherine HURGUES et Messieurs Yann BAJOT, Joachim PISTORIUS, Mourad ABERBOUR, Cyril SPASEVSKI, Hassan ABOUSHADY, Olivier GLUCK, Walid MAROUFI, Arnaud CARRON, Mounir BEN ABDENBI, Grégoire AVOT et Fabrice ILPONSE pour leurs conseils, aides et amitié pendant toutes ces dernières années.

Résumé

Cette thèse présente l'intégration de nouveaux systèmes de représentations des nombres, plus précisément les systèmes de notations redondantes, dans le flot de conception de cœurs de calculs. Les travaux effectués se découpent en trois phases.

La première est consacrée à l'introduction des systèmes de notations redondantes aux côtés des systèmes de notations classiques. À cet effet nous avons défini une nouvelle arithmétique qualifiée de *mixte*. Celle-ci répond aux problèmes liés à l'usage simultané des notations classiques et redondantes. Elle a donné lieu au développement de nouveaux opérateurs très performants capables de tenir compte de toutes les combinaisons de notations classiques/redondantes sur leurs entrées/sorties. Les trois opérations élémentaires que sont l'addition, la somme et la multiplication, ont été étudiées. Nous distinguons le cas particulier de l'ajout de deux opérandes (addition) du cas général de l'ajout de trois opérandes et plus (somme). Ces diverses opérations ont été réalisées sous formes de générateurs où la taille, le signe et la notation de chacune des opérandes ainsi que l'algorithme de calcul utilisé sont paramétrables.

La deuxième phase a eut pour objectif de déterminer l'impact de l'arithmétique mixte dans la conception de chemins de données. L'étude porte sur la redéfinition des enchaînements combinatoires et séquentiels entre opérateurs et sur l'utilisation d'arbres d'additions (somme). Cette seconde phase a permis d'identifier des règles d'optimisations génériques liées à l'usage d'opérateurs arithmétiques dans une architecture.

La troisième phase est consacrée à la prise en compte de nouveaux systèmes de représentations dans la la synthèse d'architecture. Nous nous intéressons essentiellement à la phase de traduction *comportements* → *structures physiques*. L'objectif est de proposer une méthode de projection équivalente à celle utilisée dans la synthèse bas niveau, incorporant en plus les opérateurs arithmétiques et le savoir-faire lié à leur usage. Pour répondre à ces particularités, la projection ne se fait pas directement vers une bibliothèque de cellules pré-caractérisées, mais vers des générateurs d'architectures. Cette dernière phase a donné lieu à la définition d'une méthodologie de conception de chemins de données basée sur l'utilisation de générateurs de fonctions élémentaires et à la spécification d'un outil d'aide à la conception de chemins de données. Ce dernier permet de définir un chemin de données par une description simplifiée.

Mots clefs :

Arithmétique des ordinateurs, systèmes classiques de numération, systèmes redondants de numération, algorithmes et opérateurs arithmétiques, IP core, optimisations arithmétiques, synthèse de chemins de données arithmétiques.

Abstract

This thesis presents the integration of redundant number representations in arithmetic circuit design flow. The work is divided in three parts.

The first part is dedicated to the introduction of the redundant number systems in addition to the classical number systems. For that purpose we have defined a new arithmetic, named *mixed arithmetic*, which takes into account the problems of the simultaneous use of classic and redundant notations. We have also developed a set of new operators which ensure all the possible input / output combinations in redundant and classical notations. The three elementary operations : addition (two operands), sum (more than two operands) and multiplication, have been studied. For each one we have created a generator. The generation parameter list is composed of the size, the sign and the notation of all inputs/outputs, and of the computation algorithm to use.

The aim of the second part is to study the impact of the mixed arithmetic in data path design. The analysis concerns the choice of the notations used between the different operators of an architecture. Combinatorial and sequential chains have been studied. The analysis also applies to the use of the addition tree properties (sum properties). This part allowed us to identify several generic rules of arithmetic optimizations in arithmetic circuit design.

The third part is dedicated to the introduction of the mixed arithmetic in the architecture synthesis. We are essentially interested in the behavioral-to-structural conversion. The goal is to propose a mapping phase taking into account the arithmetic operators and our knowledge in their usage. Therefore, the mapping is not directly based on standard cells but on a generator set. These generators build the operator architectures (standard cells based netlist) depending on the context. For this last part, first we present a arithmetic data path synthesis methodology using basic function generators then we propose the specification of a data path synthesis aid tool. The defined tool input is a simplified data path description.

Keywords :

Computer arithmetic, classical number systems, redundant number systems, arithmetic operators and algorithms, IP core, arithmetic optimization, arithmetic datapath synthesis.

Table des matières

Table des figures

Liste des tableaux

Glossaire

ALU	— Arithmetic and Logic Unit — Unité Arithmétique et Logique
ASIC	— Application-Specific Integrated Circuit — Circuit intégré spécifique conçu pour réaliser de façon optimale une fonction précise dans un système.
ASIP	— Application-Specific Instruction set Processor — Coeur de processeur spécialisé et optimisé, offrant un compromis entre efficacité et flexibilité
BS	Notation Borrow-Save
C2	Notation en Complément à 2
CAO	Conception Assitée par Ordinateur
CLB	— Configuable Logic Block — Bloc logique programmable dans les circuits $FPGA$
co-design	Conception Matérielle/Logicielle : Méthodologie de conception visant la réduction de l'écart entre l'implantation d'une fonction sur un $ASIC$ ou en logiciel. Elle permet la conception simultanée de systèmes hétérogène impliquant des $ASICs$ et des processeurs de type général.
CS	Notation Carry-Save
DCT	— Discret cosine Transform — Transformée cosinus discrète
DSP	— Digital Signal Processor — Processeur spécialisé pour les applications du traitement du signal et de l'image.
FA	— Full-Adder — Cellule élémentaire additionneur
FFT	— Fast Fourrier Transform — Transformée discrète de Fourrier rapide
FIR-IIR	— FIR/IIR — Finite Impulse Response / Infinite Impluse Response — Filtres numériques à réponse impulsionnelle finie ou infinie.
FPGA	— Field Programmable Gate Array — Circuit spécifique dont la fonctionnalité est programmée par l'utilisateur.
Full-Custom	Technique de réalisation matérielle dédiée d'un bloc ou d'un circuit ; en $VLSI$ la description est faite au niveau transistor.
GSM	— Global System for Mobile Communication — Standard de téléphonie mobile utilisé en Europe et dans une grande partie du monde.
H26X	Ensemble de normes d'encodage et de décodage de vidéos bas débits, mis en place par le groupe *UIT-T* : http ://www.itu.int/

HA — Half-Adder — Cellule élémentaire demi-additionneur

HDTV — High Definition TV — Télévision Haute définition, norme de diffusion de la télévision par cable et réseau hertzien, basée sur la norme $MPEG-2$

IP — Intellectual Property — Terme désignant une fonction réutilisable, pré-construite, et pouvant exister sous différentes formes (modèle HDL, modèle physique, etc.).

LSB — Lower Significant Bit — Bit/chiffre d'un nombre de poids le moins significatif

MAC — Multiplication-ACcumulation — instruction principale des processeurs de traitememt du signal et de l'image *(DSP)*

MPEG — Motion Picture Expert Group — Ensemble de normes d'encodage et de décodage de vidéos haut débits, mis en place par le groupe *ISO-MPEG* :
http ://www.mpeg.org/

MSB — Most Significant Bit — Bit/chiffre d'un nombre de poids le plus significatif

NR Notations Non-Redondantes *(C2, NSP)*

NSP Notation Numération Simple de Position

RAM — Random Access Memory — Mémoire dynamique à accès aléatoire.

RNS — Residu Number System — Notation par résidus.

RTL — Register Transfert Level —

R Notations Redondantes

UMTS — Universal Mobile Telecommunications System — Standard de systèmes de communication mobile pour les téléphones portables de troisième génération, mis en place par *the International Telecommunications Union's RIMT-2000G* :
http ://www.umts-forum.org/.

VERILOG Langage de description de circuits intégrés. Note : VHDL International et Open Verilog International sont actuellement en train de s'assembler dans une nouvelle organisation (ACCELLERA) : http ://www.accellera.org .

VHDL — VHSIC (Very High Speed Integrate Circuit) Design Langage — Langage de description de circuits intégrés. Note : VHDL International et Open Verilog International sont actuellement en train de s'assembler dans une nouvelle organisation (ACCELLERA) : http ://www.accellera.org.

VLSI — Very Large Scale Integration —

Introduction

Contexte de recherche

L'essor important des nouvelles technologies associées à la communication, aux médias, à l'imagerie médicale et à bien d'autres domaines, sont à l'origine de l'émergence de nombreuses applications et de standards liés aux traitements du signal et de l'image : téléphonie (GSM, $UMTS$), vidéo ($H26X$, $MPEG$, $HDTV$), pour ne citer qu'eux. Un des dénominateurs communs à ces diverses applications est la nécessité de disposer des circuits pouvant effectuer simultanément une énorme quantité de calculs.
Plusieurs approches et types d'architectures cibles ont été développées pour réaliser de tels circuits : principalement les microprocesseurs spécialisés (DSP) où l'application est essentiellement obtenue par programmation logicielle, les coprocesseurs ou circuits spécialisés ($ASIC$, $ASIP$) où la réalisation est essentiellement matérielle, les $FPGA$ où l'approche est matérielle mais où la réalisation est reprogrammable par logiciel, et plus récemment des circuits mixtes où l'application est répartie entre logiciel et matériel ($co-design$). Dans ce dernier cas, les architectures s'articulent autour de bus auxquels sont rattachés plusieurs ressources comme des coprocesseurs, de la mémoire et un ou plusieurs processeurs.
Quels que soient le type d'architecture cible et l'approche choisie, le développement de noyaux de calculs est nécessaire. L'évolution de leur complexité suit celle des applications visées. Aussi, le nombre et la diversité des opérateurs arithmétiques implantés ont augmenté. Les enchaînements d'opérateurs ont fait leur apparition. Cette dernière évolution a ouvert la voie à l'utilisation de systèmes de représentation des nombres comme les systèmes de notations redondantes ou les systèmes de notations par résidus (RNS).

Parallèlement à l'évolution de l'arithmétique dans la micro-électronique, la complexité croissante des applications visées et la durée de vie très courte des produits développés (téléphonie, par exemple) impliquent le besoin de méthodes et d'outils d'aide à la conception. La combinaison de ces deux contraintes s'est traduite par un désengagement des techniques de conception *bas niveau* en faveur de l'automatisation du flot de conception d'architecture.
Schématiquement cette automatisation prend la forme d'outils d'aide à la conception décomposables en trois étapes interdépendantes. Les deux premières opèrent respectivement au niveau de la spécification du système et de l'architecture et résolvent des problèmes comme la répartition logicielle/matérielle ou l'allocation et l'utilisation de ressources. Les solutions mises en

œuvre sont entre autres basées sur des modèles de transactions de données et des modèles comportementaux de blocs élémentaires comme les opérateurs arithmétiques. La troisième étape effectue la traduction des modèles de comportement en modèles physiques. Plusieurs techniques sont utilisables comme la *synthèse logique* ou l'exploitation de générateurs encapsulant un *savoir-faire* (*Intellectual-Property/IP*).

Outre les problèmes de rapidité de prototypage, la raison d'être des évolutions de l'arithmétique et des méthodes de conception, est l'amélioration des performances des circuits réalisés. Historiquement la qualité des circuits s'est mesurée en premier aux résultats en surface, car la taille des circuits était la contrainte majeure. Comme la capacité d'intégration a fortement augmenté cette contrainte est passée au second plan derrière le temps de propagation. Plus récemment la densité d'intégration des circuits et l'ouverture des semi-conducteurs aux applications embarquées ont introduit un nouveau critère essentiel qui est la consommation d'énergie.

Motivations et objectifs

La principale motivation de ce mémoire est d'introduire les nouveaux systèmes de représentations des nombres, plus précisément les systèmes de notations redondantes, dans le flot de conception de cœurs de calculs. Cette prise en compte nécessite plusieurs étapes.

La première est consacrée à l'introduction des systèmes de notations redondantes aux côtés des systèmes de notations classiques. À cet effet nous définirons une nouvelle arithmétique que nous qualifierons de *mixte* : redondante/classique. Cette arithmétique doit répondre aux problèmes liés à l'usage simultané des notations classiques et redondantes. Cela implique l'étude et le développement d'opérateurs tenant compte de toutes les notations et de toutes les combinaisons de notations sur leurs entrées/sorties.

La deuxième étape a pour objectif de déterminer quel est l'impact de l'arithmétique mixte sur la conception de chemins de données. Elle a aussi pour vocation la recherche de règles d'optimisations générales, automatisables, liées à l'usage d'opérateurs arithmétiques dans une architecture.

La troisième étape est consacrée à la prise en compte de nouveaux systèmes de représentations dans les outils d'aide à la conception. Nous nous intéressons essentiellement à la phase de traduction *comportements vers structures physiques*. Plus précisément, nous visons la spécification d'un outil d'aide à la conception de chemin de données prenant en compte les contraintes introduites par l'usage de plusieurs systèmes de notations des nombres et les contraintes liées à l'emploi d'opérateurs arithmétiques.

L'objectif est double :

1. Permettre de définir un chemin de données par une description simplifiée. C'est à dire principalement en s'affranchissant des problèmes de dynamique et de représentation des données, et en considérant les opérateurs élémentaires arithmétiques et autres : fonctions booléennes, fonctions de transfert de données, etc., comme de simples mnémoniques.
2. Proposer une méthode de projection (*mapping*) équivalente à celle utilisée dans la synthèse logique, mais incorporant en plus les opérateurs arithmétiques et le savoir-faire lié à leur usage. La particularité de ces opérateurs vient de la dimension vectorielle de leurs entrées/sorties, de la représentation arithmétique des données, et de la diversité des performances de leurs possibles architectures. Pour répondre à ces particularités, la projection ne se fait pas directement vers une bibliothèque de cellules pré-caractérisées, mais vers des générateurs d'architectures.

Le savoir-faire ou plutôt les optimisations sous-entendues dans les deux dernières étapes, peuvent s'envisager de plusieurs façons. Dans notre cas, il s'agit de mettre en place des mécanismes permettant l'amélioration de la traduction $VLSI$ d'une architecture liée à un algorithme. Il n'est pas question de retravailler les équations arithmétiques ou de transformer l'architecture développée. En clair, nous allons chercher à tirer parti des qualités de l'arithmétique mixte et des opérateurs qui lui sont rattachés et s'interdire de modifier l'architecture initiale dans son fonctionnement. Ce dernier point implique que l'ajout de matériel et les techniques comme le $retiming$ [Leis83] ou l'insertion de $pipeline$ sont exclus des optimisations. Le comportement de l'architecture optimisée doit être fidèle à l'architecture initiale.

Contenu du mémoire

L'ensemble des travaux présentés dans ce mémoire traite de la réalisation matérielle d'applications fortement calculatoires. Ils sont découpés en quatre parties.
Classiquement la première partie est consacrée à la définition des problèmes considérés. Le chapitre 1 *Arithmétique et synthèse d'architectures*, s'y référant, se décompose en deux étapes. La première correspond à un exposé général sur l'arithmétique utilisée en micro-électronique et sur la synthèse d'architecture. La seconde est dédiée plus précisément à la définition de la problématique.
La seconde partie est destinée à la composition d'une arithmétique mixte regroupant les systèmes de notations classiques et redondants. Cette partie est sous divisée en trois chapitres. Le premier (chapitre 2 *Arithmétique Mixte*) est consacré à l'introduction de l'arithmétique redon-

dante auprès de l'arithmétique entière classique et à une réflexion sur leur usage conjoint. Le second (chapitre 3 *Opérateurs élémentaires en arithmétique mixte : Architectures*) présente les diverses architectures associées aux opérations fondamentales que sont l'addition, la somme et la multiplication. Le troisième (chapitre 4 *Opérateurs élémentaires en arithmétique mixte : Performances*) est dédié à l'étude des performances intrinsèques des nouveaux opérateurs développés selon les trois critères que sont la puissance consommée, le temps de propagation et la surface. L'objectif est double : à la fois positionner les opérateurs mixtes et redondants par rapport aux opérateurs classiques mais aussi démontrer l'intérêt de l'usage des notations redondantes.
La troisième partie a pour objectif l'étude de l'impact de l'arithmétique mixte, dans divers contextes d'utilisations. Elle a aussi pour vocation la recherche de règles d'optimisations automatisables liées à l'usage d'opérateurs arithmétiques dans une architecture. Elle est articulée autour de deux chapitres. Le premier (chapitre 5 *Étude des enchaînements d'opérateurs*) traite de l'impact des opérateurs mixtes et redondants précédemment développés (addition, somme et multiplication), dans des architectures données. Nous parlerons de problèmes d'enchaînements d'opérateurs. Le second (chapitre 6 *Regroupement, fusion et glissement d'opérateurs*) est plus centré sur l'utilisation de la somme (addition de plus de deux opérandes) et des registres. Dans ce second cas les optimisations sont plus liées à l'arithmétique en générale et aux chemins de données en particulier.
La dernière partie de ces travaux est consacrée à la définition d'un outil d'aide à la conception de chemin de données incluant des traitements arithmétiques. L'idée directrice est d'encapsuler dans un outil, notre *savoir-faire* lié à l'utilisation conjointe de plusieurs systèmes de représentations des nombres et à la réalisation d'opérateurs arithmétiques. L'objectif est de permettre de réutiliser à volonté notre expertise. Plus précisément, les propos développés portent sur la phase de traduction *description de haut niveau* $\rightarrow$ *description bas niveau* d'un chemin de données. L'objectif est de tendre vers une méthode de projection équivalente à celle utilisée dans la synthèse logique, automatisant le passage d'une description structurelle simplifiée d'un bloc à une description structurelle précise du même bloc. Cette partie est traitée dans le chapitre 7 *Vers un outil d'aide à la conception de chemin de données arithmétique*.

Chapitre 1 Arithmétique et synthèse d'architectures

Cette thèse ayant pour but de généraliser l'usage simultané de plusieurs systèmes de représentations des nombres dans la conception automatisée de circuits numériques, nous commencerons par présenter l'arithmétique utilisée et la conception automatisée d'architectures. Ces présentations dépassant le cadre de cette thèse, elles resteront générales. Les détails techniques nous intéressant plus particulièrement seront vus dans les chapitres suivants. Dans un deuxième temps, nous présenterons notre problématique.

1.1 Arithmétique des ordinateurs

Nous présentons ici un panorama sur l'arithmétique utilisée dans la conception de circuits. Nous commencerons par un survol des diverses évolutions de *l'arithmétique des ordinateurs* dans les dernières décennies, puis nous ferons une présentation sommaire des systèmes de représentations employés. Nous nous attacherons plus à leurs propriétés et à l'impact de celles-ci dans la conception de circuits numériques, qu'à définir les notations associées à chaque système.

1.1.1 Évolution de l'arithmétique dans les circuits numériques

La loi de Moore [Moor65], énoncée à la fin des années 60 [Moor65], tablait sur le quadruplement de la densité d'intégration des circuits tous les 3 à 4 ans. Jusqu'à présent cette loi s'est vérifiée. Cette augmentation de la densité d'intégration s'est accompagnée de l'augmentation de la complexité des applications intégrées et des cœurs de calculs utilisés. Ces évolutions ont donné lieu, entre autres, à un enrichissement graduel de l'arithmétique employée dans les semi-conducteurs.

Dans un premier temps, l'élargissement a porté sur les opérateurs disponibles. Initialement composé d'additionneur/soustracteur, les cœurs de calculs ont peu à peu pu intégrer la multiplication, la division, les fonctions logarithme, exponentielle ou trigonométriques comme le sinus, le cosinus ou la tangente. La progression des unités et coprocesseurs arithmétiques de la famille de processeurs *80X86* et *Pentium* de la société Intel illustre parfaitement ce propos.

La seconde évolution introduite par la diminution de la contrainte matérielle a porté sur l'évolution des techniques et algorithmes mis en œuvre pour câbler les opérations dans le silicium. Ceux-ci sont passés d'un traitement des données effectué en série (des chiffres les moins significatifs/LSB vers les chiffres les plus significatifs/MSB) et inversement (arithmétique en ligne), à un traitement effectué en parallèle. Conjointement, la dynamique des données a cru, passant de mots de 4, 8 bits dans les années 80 à des mots de 64 à 128 bits dans les années 1990. Si les arithmétiques séries ont été supplantées par l'arithmétique parallèle dans la conception d'$ASIC$, l'arithmétique en ligne (calcul du MSB vers le LSB), est encore exploitée dans les $FPGA$ où les contraintes *matériel disponible* et *routage*, sont encore importantes. Il est intéressant de noter, qu'aussi bien pour l'arithmétique en ligne que pour l'arithmétique série (calcul du LSB vers le MSB) les traitements unitaires sont effectués par blocs de chiffres/bits. Les calculs peuvent alors être semi-parallèles.

Enfin un troisième type d'évolution a été rendue possible par l'augmentation de la densité d'intégration des circuits. Celui-ci correspond à l'emploi de nouveaux systèmes de représentations des nombres. En premier lieu l'introduction des notations flottantes a permis d'élargir l'espace des nombres utilisés des nombres entiers ou à virgule fixe aux nombres réels. En deuxième lieu, l'usage des notations redondantes et des notations par résidus a permis d'améliorer les performances en fréquences des opérateurs et des circuits réalisés. Nous reviendrons sur ces systèmes de représentations dans la sous-section 1.1.2.

Il est intéressant de noter que si la mise en œuvre de l'arithmétique dans les semi-conducteurs a été et est encore graduelle, de très nombreux concepts, problèmes et solutions ont été proposés et formalisés dans les années 1950/1960. C'est le cas des systèmes de représentations flottants [Camp92], des représentations par résidus [Garn59, Szab67] où des représentations redondantes [Metz59, Aviz62]. C'est aussi le cas pour les algorithmes utilisés pour câbler les opérations arithmétiques comme la multiplication [Dadd65, Wall64] ou les fonctions logarithme, exponentielle et trigonométriques $CORDIC$ [Brig28, Vold59].

1.1.2 Systèmes de représentations

Du point de vue de la conception de circuits numériques, les systèmes de représentations peuvent être séparés en deux types : ceux définissant une arithmétique autonome et les autres. Nous entendons par arithmétique autonome une arithmétique permettant de couvrir l'ensemble des besoins arithmétiques élémentaires tout en offrant des performances matérielles cohérentes et acceptables.

Le minimum nécessaire comprend les opérateurs basiques : un additionneur / soustracteur, un comparateur, un multiplieur et un diviseur, et les primitives élémentaires nécessaires au contrôle des calculs comme le signe, la valeur nulle et la détection de dépassement de capacité. La notion de contrôle évoquée ici n'est pas à confondre avec la notion de commande des calculs (soustraction/addition par exemple) ; nous parlons ici uniquement du contrôle des résultats des calculs.
Un coût à ne pas négliger dans l'évaluation d'un système de représentation, est celui de la taille des codages des nombres sur les opérations non arithmétiques comme la mémorisation ou les aiguillages (multiplexeurs/démultiplexeurs ou bus).

Systèmes de représentations définissant une arithmétique autonome

Dans un premier temps les nombres ont été représentés et manipulés sous forme d'une valeur entière signée ou non signée permettant de coder un nombre entier ou un nombre à virgule fixe ; la position de la virgule étant définie arbitrairement en fonction de la dynamique des valeurs manipulées. Nous détaillerons les principales représentations correspondantes dans le chapitre 2. Pour fixer le lecteur, il s'agit de la *numération simple de position*, de la notation *signe et grandeur* et de la notation *en complément à la base*. Nous nous référerons à elles dans le reste de ce mémoire en parlant de notations ou de représentations *classiques* ou *conventionnelles*.
Très rapidement, et afin de représenter les nombres réels, les systèmes de représentations classiques ont été complétés par le développement des représentations des nombres à virgule flottante [Camp92, Yohe73, Cody73, Coon84, Gold91, Kaha96]. Ces travaux ont débouché sur un norme établi en 1985 [IEEE85].
Les *flottants* constituent une notation de plus haut que les systèmes déjà exploités où les nombres sont codés tel que $Nombre = Mantisse * 2^{Exposant}$; $Mantisse$ et $Exposant$ sont des nombres *classiques* de taille fixe. Ces notations permettent d'étendre la représentation des nombres entiers ou à virgule fixe, aux nombres réels ou plutôt à virgule flottante. Cette extension s'est concrétisée par la dilatation de l'intervalle des nombres pouvant être codés sur une quantité N de bits donnés. Le nombre de codes possible reste toutefois de 2^N comme pour un codage classique sur N bits. L'élargissement a été rendu possible par le passage de l'écart entre deux codes contigus, d'une valeur fixe pour les notations classiques à une valeur coefficientée par l'exposant pour les flottants. Aussi, sur un intervalle donné et pour N fixé, les flottants sont moins précis que les notations classiques. Pour obtenir une précision équivalente, la mantisse des flottants doit être égale à la taille de la notation classique correspondante.
Nous l'avons dit, les notations flottantes sont un sur-ensemble des notations classiques. Il en va de même pour les opérateurs arithmétiques associés [Mull89, Parh99]. Ceux-ci se composent

d'opérateurs arithmétiques classiques qui effectuent le traitement de l'opération proprement dite, et d'autres opérateurs arithmétiques classiques pour exécuter en parallèle le calcul du nouvel exposant. Certaines opérations comme l'addition nécessitent au préalable l'alignement des virgules. Cette dé-normalisation des nombres avant traitement implique une re-normalisation après calcul. Le coût des opérateurs flottants et donc plus élevé que celui des opérateurs classiques tant au plan de la vitesse de calcul que du matériel nécessaire.

Globalement les systèmes classiques et flottants définissent deux arithmétiques indépendantes et minimales. Celles-ci partagent les mêmes propriétés :

1. Toutes les opérations arithmétiques élémentaires comme l'addition, la soustraction, la multiplication la division ou les comparaisons sont réalisables sans recourir à d'autre systèmes de notations.
2. La détection des indicateurs d'états comme le signe, la valeur zéro ou le dépassement de capacité, est aisée, facilitant ainsi le contrôle des calculs.
3. Le codage d'un nombre est unique. Il est aussi compact. Les nombres négatifs peuvent être représentés.

Toutefois le coût plus important tant au niveau du codage des nombres que des opérateurs arithmétiques élémentaires, sont à l'origine de l'écartement des systèmes flottants des nombreuses applications où les contraintes de temps et de place (matériel) sont essentielles. Actuellement les flottants sont cantonnés à certains processeurs généraux ou DSP et à une utilisation logicielle plutôt que matérielle. Le fait que ces systèmes soient normalisés, peut aussi se révéler une contrainte dans leur réalisation matérielle.
Nos objectifs étant orientés vers la mise en place d'une arithmétique rapide, nous ne considérerons que les notations classiques dans la suite de ce mémoire.

Autres systèmes de représentations

Si les systèmes de représentations flottants sont exploités parce qu'ils permettent un élargissement de l'espace des nombres utilisables, d'autres systèmes sont eux employés afin d'améliorer les performances des blocs arithmétiques. Le premier objectif visé est de réduire le temps de calcul de l'addition qui est l'opération la plus élémentaire et la plus usitée. En circuiterie cela se traduit par la volonté de réduire la chaîne de propagation des retenues.
Les systèmes de représentations redondantes : *notations à retenues conservées* [Metz59] et *systèmes de nombres signés* [Aviz62], sont utilisés dans ce sens. Ces notations ont en point commun la particularité d'associer plusieurs codages à un même nombre. Cette particularité a permis

de restreindre la propagation des retenues de telle sorte que le calcul de l'addition est en temps constant [Aviz62, Mull89, Parh99]. Par comparaison le meilleur temps de calcul obtenu en utilisant les notations classiques, est en $O(log_2(N))$ [Mull89, Parh99]. Nous reviendrons en détail sur ces systèmes dans le chapitre 2.

Une seconde famille de systèmes de représentations est employée dans le but d'améliorer les performances des circuits. Il s'agit des systèmes de représentations par résidus [Garn59, Szab67]. L'idée directrice est de découper les nombres en sous-ensembles indépendants. Pour se faire, on définit un ensemble $M = \{m_1, m_2, ..., m_N\}$ de N entiers, appelé la base. En assurant que tous les m_i sont premiers entre eux, chaque entier I tel que $0 \leq I < \prod_{i=1}^{N} m_i$, est représentable par un unique N-tuple $I_{RNS} = \{< I >_{m_1}, < I >_{m_2}, ..., < I >_{m_N}\}$; $< I >_{m_i}$ correspond à I modulo m_i. Cette propriété acquise, les calculs s'effectuent parallèlement et indépendamment sur les divers sous-ensembles. Les temps de calculs sont alors réduits ainsi que le matériel requis pour la réalisation des opérateurs. Dans le cas de l'addition, le temps de propagation est en $O(log(log(N)))$ [Parh99]. Le premier log correspond à la décomposition en sous-ensembles, le deuxième au meilleur résultat obtenu dans la conception d'un additionneur classique. Ces systèmes sont très intéressants car ils offrent aussi une réduction conséquente du temps de propagation et du matériel nécessaire au calcul de la multiplication [Parh99, Pali00]. Cette seconde opération est aussi essentielle que l'addition à la réalisation de circuits numériques.

L'introduction de ces nouveaux systèmes s'est d'abord faite dans la conception même des opérateurs. C'est le cas des notations redondantes qui sont employées dans la réalisation d'opérateurs comme les multiplieurs ou les diviseurs [Cava84, Mull97, Parh99]. Cet usage porte à la fois sur la réduction des chaînes de propagation des retenues dans les arbres d'additions [Wall64, Dadd65], et sur l'utilisation des propriétés des systèmes définis par Avizienis dans [Aviz62]. Dans ce second cas on peut citer comme exemples les multiplieurs parallèles de *Booth* [Boot51, Souf00] ou les multiplieurs par une constante [Aber95] et annexe A. Il est intéressant de noter que dans le cas des opérateurs incorporant une somme à la fin du calcul, le résultat est d'abord exprimé en notation redondante puis converti en notation classique. Le convertisseur n'est autre qu'un additionneur classique.

Plus récemment, l'augmentation de la complexité des applications visées a nécessité la conception de cœurs de calculs câblant le chaînage de plusieurs opérateurs arithmétiques. Ces enchaînements ont ouvert la voix à l'emploi explicite des systèmes de représentations redondantes et RNS. Nous entendons par *explicite* le fait que ces notations sont utilisées comme lien entre opérateurs.

Dans le cas des redondants, il s'agit d'éviter la phase de conversion de notation *redondante*

→ *classique* présente à la fin de nombreux opérateurs comme les additionneurs/soustracteurs ou les multiplieurs. Il existe quelques exemples qui suivent cette direction. Briggs et Matula [Brig93] ont conçu un bloc effectuant une multiplication/accumulation (MAC) utilisé par le coprocesseurs Cyrix 83D87, dans laquelle le résultat de la multiplication n'est pas converti avant d'être accumulé. Un autre exemple très intéressant d'utilisation explicite des redondants, est proposé par Olivier Peyran qui, dans sa thèse [Peyr97], traite de l'introduction des notations redondantes dans la synthèse de haut niveau.
Les exemples d'utilisations des notations RNS portent essentiellement sur des cœurs de calculs très fortement arithmétique, incluant de multiples multiplications et additions, comme le calcul de la transformée cosinus discrète [Rami00] ou les filtres numériques [Conw00]. Cette particularité s'explique par le coût (en délai) assez élevé des conversions entre notations RNS et classiques [Parh99]. De nombreux travaux actuels traitent de ce problème [Soud00, Pali00] et cette contrainte tend à se réduire. Toutefois elle reste encore une limite à l'utilisation des RNS.

Le premier défaut majeur de ces deux systèmes de représentations, est de ne pas permettre une détection aisée (à moindre coût) des indicateurs de signe, de nullité et de dépassement de capacité, comme avec les représentations classiques ou flottantes. Cette difficulté est à l'origine d'une succession de problèmes et de limitations :

1. Sans indicateurs de contrôle les systèmes redondants et RNS ne définissent pas une arithmétique autonome. Ils doivent être utilisés conjointement avec une arithmétique autonome, généralement l'arithmétique classique. Des convertisseurs de format sont alors nécessaires. Leur coût est non nul et l'amélioration des performances apportée par l'usage des notations redondantes et RNS en est réduite.
2. La complexité des opérateurs dont l'algorithme est basé sur ces indicateurs, croît fortement ce qui est à l'origine de la dégradation ou de l'inversion du gain potentiellement introduit par les autres opérateurs. C'est le cas pour les opérateurs de division et de comparaison qui exploitent le signe de résultats partiels ou du résultat final. Le recours à des opérateurs définissant ces opérations dans d'autres systèmes de représentations est nécessaire.
3. Ces systèmes sont écartés des processeurs, et ne sont utilisables qu'à l'intérieur de blocs arithmétiques enchaînant des calculs sans contrôle des résultats. Dans le cas des notations redondantes ces blocs peuvent même être des opérateurs.

Le second défaut majeur de ces notations, est qu'elles ne sont pas compactes. L'augmentation de la taille des codages sous-entendue a un coût important sur les opérations non arithmétiques (mémorisation, multiplexeur, etc.). Pour une même dynamique et par rapport aux systèmes classiques, la taille des codages varie autour du double que ce soit pour les systèmes

redondants [Mull89, Parh99] (*notations à retenues conservées* section 2.2) ou pour les RNS. Dans le cas de ces derniers l'augmentation dépend à la fois des modulos choisis et de la dynamique maximum des données que l'on souhaite traiter.

1.2 Conception automatisée d'architectures / synthèse d'architectures

Parallèlement à l'évolution de l'arithmétique dans la conception de circuits, la complexité croissante des applications visées et la durée de vie très courte des produits développés, impliquent le besoin de méthodes et d'outils d'aide à la conception. La combinaison de ces deux contraintes s'est traduite par un désengagement des techniques de conception *bas niveau* en faveur de l'automatisation du flot de conception d'architecture.
Schématiquement cette automatisation prend la forme d'outils d'aide à la conception décomposable en trois étapes successives. Les deux premiers opèrent respectivement au niveau de la spécification du système et de l'architecture. Les solutions mises en œuvre sont entre autres basées sur des modèles de transactions de données et des modèles comportementaux de blocs élémentaires comme les opérateurs arithmétiques. On parlera de *synthèse de systèmes* et de *synthèse de haut niveau*. Le troisième étage effectue lui, la traduction des modèles comportementaux en modèles structurels *bas niveau*.
Dans ce contexte, la conception d'un circuit est descendante et passe par plusieurs phases de raffinement. L'architecture obtenue est hiérarchique. Sa réalisation sera ascendante : des blocs de plus bas niveau vers les blocs de plus haut niveau.

1.2.1 Synthèse de système

L'objectif de la synthèse de systèmes, est de définir la répartition d'une application entre réalisation logicielle et matérielle. Cette réparation comprend trois parties. La première correspond à la synthèse logicielle et permet, par exemple, de définir le code exécutable pour un processeur. La deuxième correspond à la synthèse matérielle. Elle effectue indifféremment des appels à des ressources préexistantes (processeur, coprocesseur, RAM, etc.) ou en définit de nouvelle par un comportement et une technologie cible ($FPGA$, bibliothèque de portes logiques précaractérisées, blocs dur déjà câblés $Full - Custom$). La troisième partie effectue la synthèse des communications (bus, connexions, protocoles de transactions, etc.) permettant ainsi d'assurer le dialogue entre les diverses ressources.
À ce niveau, un système est constitué d'éléments décrits par des comportements (algorithmes) généralement de très haut niveau. Chacun de ces éléments correspond à un circuit numérique synchrone, la synchronisation étant assurée par une ou plusieurs horloges. Par la suite, chaque

description comportementale va être traduite en une description structurelle implantant les parties de contrôle et les parties opératives des algorithmes.

1.2.2 Synthèse *de haut niveau*

Cette seconde étape a pour rôle de traduire un circuit, décrit par un comportement haut niveau (algorithme) en une description sous forme de *transferts de registres* (RTL). Un des objectifs est de dissocier la partie opérative de la partie de contrôle.
Cette étape est décomposable en 4 phases : l'extraction des graphes de dépendances de contrôles et de données à partir de l'algorithme, la sélection du type des ressources (opérateurs), l'ordonnancement des opérations et l'allocation de ressources.
Classiquement deux types d'ordonnancements sont effectués : l'ordonnancement sous contraintes de ressources où les ressources sont fixées et où l'objectif est de minimiser le temps en cycles d'horloges, et l'ordonnancement sous contraintes de temps où le nombre de cycles d'horloge est fixé et où l'objectif est de minimiser les ressources. L'ordre d'enchaînement des trois dernières étapes n'est pas figé. La solution généralement adoptée consiste à sélectionner le type des opérateurs, puis à effectuer l'ordonnancement et l'allocation de ressource.
La traduction sous forme de *transferts de registres* s'appuie sur un ensemble de ressources dont on connaît le comportement et les performances, comme les opérateurs arithmétiques, les multiplexeurs, les bus ou les registres. Ces derniers sont synchronisés par une horloge. À travers ces ressources, il est tenu compte de la technologie cible choisie : portes logiques précaractérisées $FPGA$ ou $Full - Custom$.
À ce point, on sait sur quel opérateur et à quel cycle d'horloge, chaque opération de l'algorithme est exécutée. De même, on sait quel registre va être affecté par une variable. La structure de la partie opérative est connue. On peut alors facilement extraire de cette description, un organigramme de contrôle dont les états sont les opérations RTL. La partie contrôle est aussi connue.
Globalement la structure du circuit est définie. Chaque élément des parties *opérative* et *contrôle* ainsi que leurs interconnexions sont décrites, par exemple sous forme d'équations booléennes. Pour la partie opérative, ces descriptions s'arrête au niveau des ressources. En effet, bien que les propriétés (comportements) de celles-ci aient été utilisées lors des phases de définition et d'allocation des opérateurs, leurs structures ne sont pas forcément connues. Ce niveau de raffinement permet de réduire la complexité de la synthèse de haut niveau.
Deux types d'outils ont été développés pour réaliser la synthèse de haut niveau. Dans les premiers tous les mécanismes de décisions (choix opérateurs,...) sont intégrés. Nous parlerons de synthèse globale. Dans les seconds le choix et l'allocation des ressources sont définis par l'utili-

sateur via la description de la partie opérative sous forme de *chemin de données*. La construction de la partie contrôle est obtenue, elle, par synthèse de haut niveau. Elle est basée sur la description haut niveau du circuit, sur la description de la partie opérative et sur les performances des ressources utilisées. Ces outils définissent un sous-ensemble des premiers.
Cette approche *synthèse de partie contrôle* a été retenue dans divers outils $HUGH$ actuellement développé au sein de notre laboratoire [Augé97, Augé99].

Sur le plan arithmétique et jusqu'à récemment, cette étape de la synthèse d'architecture ne prenait en compte que les notations et les opérateurs classiques. Dans sa thèse [Peyr97], Olivier Peyran propose d'intégrer en plus, les systèmes de notations redondants. Ces travaux novateurs ont montré la faisabilité et l'intérêt d'utiliser plusieurs systèmes de représentations simultanément.

1.2.3 Synthèse au niveau transfert de registres (RTL)

Ce dernier étage effectue la traduction de la description RTL en description structurelle bas niveau ; c'est à dire la transformation des équations logiques et des ressources en un réseau de cellules élémentaires interconnectées. Suivant la technologie cible, les cellules élémentaires sont des portes logiques précaractérisées (bibliothèques de portes), des blocs *durs optimisés* ($Full - Custom$) ou des CLB ($FPGA$).
Deux approches sont possibles pour réaliser la synthèse RTL : l'approche dominée par le contrôle et l'approche généralisée. Dans la première, le circuit est découpé en une partie opérative (chemin de données) et une partie de contrôle (contrôleur) synthétisées séparément. Dans la seconde, le circuit est considéré dans son ensemble, le contrôle est alors intégré dans chaque bloc du graphe de dépendance de contrôle de départ. Les outils de synthèse adoptent généralement l'approche dominée par le contrôle.
Dans les deux cas, la synthèse RTL se décompose en deux phases. Une phase de traduction en équations logiques des ressources, de la connectique et du contrôle et une phase de projection (*mapping*). Ces phases font appel à la synthèse de contrôleurs, à la *synthèse logique* et à l'exploitation de générateurs encapsulant un savoir-faire (*Intellectual-Property* / IP).

Synthèse logique ou synthèse *bas niveau*

L'objectif de la synthèse logique est de convertir les équations logiques définies lors de la synthèse RTL en une description minimale en termes de cellules élémentaires. La notion de *minimale* peut-être raccrochée à plusieurs critères comme la consommation, la quantité et la taille des cellules élémentaires (surface) et le temps de propagation du bloc obtenu.

Cette synthèse est décomposable en deux étapes distinctes. La première correspond à l'optimisation booléenne des équations logiques. Le but est entre autres d'éliminer la redondance des équations (partage de partie d'équation commune), de les simplifier via leur substitution par des équations *logiquement* identiques mais moins coûteuses en matériel. La seconde a pour rôle la projection (*mapping*) des nouvelles équations optimisées sur la cible technologique souhaitée. Il s'agit ici de découper les équations en fonctions des possibilités (disponibilité, complexité des cellules) et des qualités (surface, délai, consommation) offertes par les cellules élémentaires.
Les deux principaux avantages de la synthèse logique sont la facilité d'utilisation et la portabilité qu'elle offre. La facilité car c'est une approche *presse-bouton* et les langages de description sont normalisés : $VHDL$ et $Verilog$ par exemple. La portabilité car avec la même description *haut niveau* il est possible de changer de cible technologique.
Le revers de la médaille vient de la difficulté de synthétiser les ressources arithmétiques. En effet, la synthèse ne peut pas/ne sait pas tirer parti des structures régulières, de la variabilité (taille entrée/sortie, algorithmes), et de la forte corrélation des équations logiques internes à ces blocs.
Ce problème est encore d'actualité et il se retrouve dans les outils commerciaux comme Synopsys [SYN00]. Une des solutions adoptées consiste à remplacer la synthèse d'un opérateur arithmétique par l'appel, de façon transparente, à un générateur de l'opération souhaitée, celui-ci fournissant directement les équations à projeter.

Générateurs et chemin de données

La principale motivation du développement de générateurs est d'offrir aux concepteurs des blocs performants et *prêt à l'emplois*. L'idée directrice est d'encapsuler un savoir-faire, une expertise, lié à un domaine, comme l'arithmétique, afin de le réutiliser à volonté. L'expertise intégrée peut porter aussi bien sur les aspects algorithmiques que sur les aspects électriques.
Dans un premier temps le but recherché a été de proposer des blocs optimisés en surface et en délai. De tels générateurs fournissent une description très bas niveau (silicium, transistors) et prennent en compte les problèmes de routage et de placement. C'est le cas, par exemple, des générateurs de mémoires [Grei94, Boua94, Turi00]. Cette approche est très intéressante dans le cas de blocs ayant de très fortes contraintes électriques ou intégrant des fonctionnalités autonomes (mémoires, conversions analogique $\leftrightarrow$ numérique).
Par contre, dans le cas de fonctionnalités noyées dans un ensemble (opérateurs arithmétiques par exemple), cette approche pose des problèmes de compatibilité technologique entre blocs générés. Ces problèmes sont amplifiés par la rapidité d'évolution des technologies submicroniques. Dans ce contexte, une description très bas niveau est une véritable barrière à la réutilisa-

tion car le changement de pas de la technologie submicronique implique la réécriture du générateur. Un autre point gênant de l'approche *génération de très bas niveau* est la rigidité induite par les placements/routages, qui sont difficilement exploitables en dehors des blocs autonomes.
Une seconde approche *générateur* a été proposée. Cette fois le but recherché est la réutilisation ou plutôt la portabilité. Cette volonté implique de pouvoir suivre à la fois la diminution du pas des technologiques submicroniques et les évolutions des cibles technologiques : modification des CLB dans les $FPGA$, changements de ou dans les bibliothèques de cellules précaractérisées (ajout/suppression de cellules par exemple). La solution adoptée consiste à fournir des descriptions de plus haut niveau : structurelles voir comportementales.
Dans ce cas les contraintes électriques sont minimisées voir non prises en compte et le placement routage est réduit à un pré-placement ou n'est pas fourni. La portabilité des structures générées se fait aux dépens de leurs performances. Toutefois de tels générateurs offrent un bon compromis portabilité / performance. Cette approche a été mise en place pour toutes les cibles technologiques. Elle a donné lieu au développement de plusieurs méthodologies. Dans [Amar91] la conception sous formes de dessins symboliques des motifs permet de glisser d'une technologie λ_1 μm à une technologie λ_2 μm. Les descriptions obtenues sont dépendantes des motifs mais permettent de fournir un pré-placement.
La solution proposée dans [Houe97, Vauc97] consiste à écrire un générateur en s'appuyant sur une bibliothèque de cellules virtuelles. Le lien entre cellules virtuelles et réelles est assuré par des générateurs bas niveau construisant les cellules manquantes et instanciant les cellules existantes. À chaque bibliothèque correspond alors une liste des cellules existantes (ainsi que leurs caractéristiques électriques) et une liste de *constructeurs*.
Cette solution rend les générateurs indépendants des bibliothèques réelles et de la technologie submicronique, permet de prendre en compte certaines contraintes électriques et peut fournir un pré-placement. La combinaison des mécanismes proposés : cellules virtuelles + générations, remplace efficacement la synthèse RTL de la partie opérative et est particulièrement intéressante dans le cas de la description de chemin de données.

Plus généralement la méthodologie utilisée consiste à fournir une description structurelle détaillée synthétisable. La description obtenue est alors portable sur toutes les technologies quels que soient le pas en micron ou la cible. Toutefois les contraintes électriques et les algorithmes utilisés pour générer un opérateur donné diffèrent suivant la cible. Si la portabilité permet de suivre l'accroissement de la densité d'intégration, elle ne permet pas d'offrir une solution valable pour l'ensemble des cibles technologiques. En pratique, il est défini un générateur par cible.

1.3 Problématique

À travers la littérature [Peyr97, Parh99, Rami00, Conw00], il apparaît que l'usage des notations redondantes et des notations par résidus permet, grâce aux opérateurs associés, d'améliorer les performances des circuits réalisés. Il serait intéressant de prendre en compte et d'automatiser le recours à ces notations, dans les diverses phases de la synthèse d'architecture. Un tel objectif implique de tenir compte des deux défauts majeurs de ces systèmes que sont :

1. l'obligation de composer avec un système définissant une arithmétique autonome (systèmes classiques en arithmétique entière) afin d'élaborer une arithmétique répondant aux besoins minimums de la conception de circuits.
2. le surcoût par rapport aux systèmes de représentations classiques des codages associés (quantité de bits), et l'impact de celui-ci sur les opérateurs non arithmétiques.

Les problèmes posés portent sur des domaines aussi variés que *l'arithmétique des ordinateurs*, la synthèse de haut niveau ou les langages de description. Proposer une solution pour chacun d'eux, dépasse largement le cadre d'une thèse. Toutefois et afin de mieux introduire notre travail, nous avons choisi de présenter une vue de la problématique plus large que celle que nous allons traiter. Nous introduirons la problématique plus spécifique à ce document, dans la sous-section 1.3.3.

1.3.1 Arithmétique minimum

Le premier problème soulevé est la mise au point d'une arithmétique autonome basée sur l'utilisation commune de plusieurs systèmes de représentations. Dans le cas de l'arithmétique entière l'objectif est de regrouper les systèmes redondants, RNS et classiques. Globalement, il est impératif de disposer, au minimum, des opérateurs d'addition et de multiplication, des signaux de contrôle des calculs mais aussi de passerelles entre représentations. Deux solutions sont possibles pour développer une telle arithmétique :

1. L'approche minimale, c'est à dire se contenter d'utiliser les opérateurs associés à chaque système pour couvrir tous les besoins, et basculer d'un système à un autre pour assurer de bonnes performances. Cela revient à définir une arithmétique par système et développer uniquement les passerelles entre représentations. Le recours à un système donné est alors conditionné par le coût des conversions.
2. Exploiter, si possible, la compatibilité aux niveaux les plus élémentaires (chiffre/bit par exemple). En sus des opérateurs correspondant aux divers systèmes employés, cette approche implique de disposer d'opérateurs prenant en compte différents systèmes de notations. Il va de soi que ces opérateurs *mixtes* doivent avoir des propriétés meilleures que la

combinaison *conversion + opérateur* qu'ils remplacent. Cette approche plus bas niveau doit permettre d'obtenir de meilleures performances et offrir plus de souplesse.

La deuxième solution est la plus intéressante. Nous parlerons d'arithmétique *mixte*. Celle-ci demande de développer à la fois l'ensemble des opérateurs liés à un système, les passerelles entre représentations mais aussi les opérateurs mixtes. Il est nécessaire de disposer au minimum des opérateurs d'addition et de multiplication pour que le recours à un système soit intéressant.

Une fois l'arithmétique *mixte* mise au point, il faut en déterminer les besoins, les limites et les performances. Au-delà des opérateurs eux-mêmes et de leurs performances intrinsèques, il est important de connaître l'impact de cette arithmétique dans différents contextes d'utilisations. Cette seconde phase va donner lieu à la recherche de règles d'optimisations liées à l'arithmétique mixte.

1.3.2 Conception automatisée

Le second problème posé par le recours explicite à plusieurs systèmes de représentations des nombres, est de déterminer qu'elles sont les modifications à apporter aux outils d'aide à la conception de circuits, qui vont permettre de prendre en compte la multiplication des notations mais aussi la prolifération des opérateurs associés à une même opération et les optimisations/limitations arithmétiques liées à ces opérateurs et à leur utilisation.
La principale modification à apporter est le changement du modèle de connexion entre ressources. Jusqu'ici, ce modèle était unique, que ce soit au niveau le plus élémentaire (manipulation exclusivement de bits) ou à un niveau plus élevé (manipulation de vecteur de bits). Les problèmes de gestion de signe et de choix entre notations d'un même système étaient écartés par l'emploi systématique d'une seule notation ou par l'élimination de ces notions abstraites. En fait, seule la taille pouvait différer.
La multiplication des représentations détruit cette unicité. En premier car, pour coder matériellement un même entier, les connexions élémentaires (bit/chiffre) et leur quantité diffèrent suivant la notation, impliquant un coût (nombre de bits) variable. En deuxième, car l'indisponibilité de certains opérateurs associés aux systèmes non autonomes, oblige de connaître la nature (la notation) des connexions. Les connexions entre ressources sont alors hétérogènes et nécessitent une gestion explicite de leur notation et de leur dynamique.

Synthèse de haut niveau

Concernant la *synthèse globale* (sous-section 1.2.2) l'évolution de la nature des connexions doit être intégrée directement dans les mécanismes de choix des opérateurs, dans l'allocation des

ressources et dans l'ordonnancement. Cela suppose que la *synthèse globale* prenne en compte les limitations des systèmes non autonomes (notamment par la gestion des conversions entre notations), la multiplication des architectures associées à une opération, et la modification du statut des ressources banalisées comme les registres ou les multiplexeurs (leur coût est maintenant fonction des notations à leur(s) entrée(s)). Les changements à introduire sont conséquents. Par contre les adaptations à apporter restent internes à la synthèse. Les descriptions en entrée et en sortie de celle-ci sont inchangées.

Une telle approche est proposée par O. Peyran dans sa thèse [Peyr97]. Toutefois elle ne concerne que les systèmes classiques et partiellement les systèmes redondants. Bien que les solutions adoptées soient en partie basées sur les similitudes entre notations redondantes et classiques, celles-ci constituent une base intéressante notamment pour la gestion des conversions.

Dans le cas de la *synthèse orientée contrôle* (sous-section 1.2.2), la partie opérative est définie par le concepteur. Dans ce cas, la synthèse ne s'occupe ni du choix des ressources ni de leur allocation. Elle porte essentiellement sur la partie contrôle. Jusqu'ici, réaliser cette partie demande seulement de connaître les propriétés (associativité, commutativité, ..., performances) des diverses ressources, et une description simplifiée de la partie opérative.

La difficulté de générer les signaux de contrôle des calculs, à partir des représentations RNS et redondantes, fait apparaître comme un impératif le besoin de connaître en plus la nature (notation) de chacune des ressources pouvant être associée au contrôle des calculs. Nous verrons que cet impératif est aussi lié aux limitations des contextes où les systèmes non autonomes peuvent être utilisés. Il est toutefois possible de traiter ce problème sans modifier la synthèse. Pour cela, il suffit de s'assurer, lors de la description de la partie opérative, qu'aucune connexion utilisable pour le contrôle des calculs n'est définie par une notation redondante ou RNS. Par exemple, on peut jouer sur les propriétés allouées à une ressource (additionneur sans détection de dépassement de capacité ou de signe, etc.).

Toute la problématique, liée à la gestion des représentations des nombres et aux optimisations/limitations arithmétiques, est reportée dans la spécification de la partie opérative. Les choix incombent alors au concepteur du chemin de données. Il serait souhaitable de l'assister dans la création de la partie opérative. Pour trois raisons au moins : premièrement car les opérateurs, les optimisations et les limitations arithmétiques sont sujets à évolutions (changement de cible technologique ou de pas technologique, nouvel algorithme, etc.), deuxièmement afin de réduire le temps de conception (un des objectifs de la synthèse de circuits) et troisièmement car le niveau de compétence arithmétique d'un concepteur est variable.

L'objectif est à partir de la description de la partie opérative fournie par le concepteur, de définir

ou de redéfinir (améliorer) la nature de toutes les ressources en fonction *d'un savoir-faire* arithmétique. Cela suppose que si au départ la quantité, la fonctionnalité ainsi que les connexions des entrées/sorties des ressources sont impérativement définies, ce n'est pas forcément le cas en ce qui concerne les structures (algorithmes) et la nature (notation) des entrées/sorties des ressources.
La description du chemin de données en entrée peut-être qualifiée de *molle*. Cette mollesse n'est pas forcément compatible avec les langages de description actuels et suppose une adaptation de l'existant ou la définition d'un nouveau langage. Par contre, il est préférable de ne pas modifier les langages de description en sortie de la synthèse, ceux-ci sont *normalisés* ($VHDL$, $Verilog$). Le problème de l'expression des notations peut-être écarté en s'appuyant sur un groupe de ressources différenciées à la fois par leur traitement et par la combinaison des notations de leurs entrées/sorties.
Un autre point important à considérer et qu'associer une structure et des notations à une ressource suppose de posséder le comportement et les propriétés des possibles structures.

Le travail d'une telle *aide à la spécification de chemin de données* s'apparente à la phase de définition des ressources dans la synthèse globale.

Synthèse RTL / phase de projection

Nous venons de voir que l'inclusion explicite de plusieurs systèmes de représentations des nombres dans la synthèse de haut niveau, n'implique pas la modification de la description fournie à sa sortie. Par suite, la synthèse RTL ne nécessite pas d'adaptation si elle tient compte de la multiplication des structures associées à une ressource, dans la phase de production des équations élémentaires nécessaires à la projection vers une cible technologique. Cela sous-entend que la notion de représentation n'existe plus.
L'ensemble des évolutions introduites par l'arithmétique mixte, porte sur le chemin de données. Dans ce contexte, la méthode communément adoptée consiste à utiliser la génération plutôt que la synthèse bas niveau. Finaliser l'intégration de l'arithmétique mixte dans le flot de conception demande donc de développer les générateurs des divers nouveaux opérateurs. La phase de projection est, elle, inchangée : synthèse logique ou assumée par génération. Ces remarques sont valables que la synthèse de haut niveau soit globale ou orientée contrôle.

Concernant la partie opérative, la phase de définition des équations élémentaires peut se restreindre à l'appel de générateurs. Intégrer cet appel dans *l'aide à la spécification de chemin de données* le ferait évoluer vers une *aide à la conception de chemin de données*. Créer ce lien permet-

trait de disposer si besoin de modèles plus précis des structures associées à une ressource, afin d'affiner les choix matériels.

1.3.3 Notre problématique

Comme nous l'avons déjà dit, les problèmes soulevés par l'usage simultané de plusieurs systèmes de représentations des nombres dans le flot de conception d'un circuit, sont nombreux et dépassent le cadre d'une simple thèse. Aussi, dans ce mémoire, nous nous intéresserons essentiellement à la définition d'une arithmétique entière mixte et à la spécification d'un outil d'aide à la conception de chemins de données arithmétiques.
En premier lieu, il nous faut composer une arithmétique comprenant plusieurs systèmes de représentations, et étudier ses performances et ses limites, en terme d'opérateurs élémentaires minimums et en terme d'impact dans un contexte architectural donné. Un des objectifs de cette étude est d'extraire des méthodes d'optimisation automatisables. La problématique de projection soulevée par les systèmes redondants et RNS étant identique, nous nous intéressons surtout à la composition d'une arithmétique incluant les systèmes redondants et les systèmes classiques.
En deuxième lieu il nous faut définir les objectifs d'une aide à la conception de chemins de données arithmétiques. Un tel outil doit :

1. Permettre de définir un chemin de données par une description simplifiée. C'est à dire principalement en s'affranchissant des problèmes de dynamique et de représentation des données, et en considérant les opérateurs élémentaires arithmétiques et autres : fonctions booléennes, fonctions de transfert de données, etc., comme de simples mnémoniques.
2. Inclure une phase de définition/redéfinition des structures attachées aux ressources, tenant compte de notre savoir faire arithmétique : performances et disponibilité des opérateurs, limitation de contexte, par exemple. Une des contraintes majeures des optimisations, que nous nous sommes fixées, est le respect du comportement extérieur du chemin initial au bit près et au cycle près.
3. Intégrer une phase de projection tenant compte de la cible technologique. Comme couvrir l'ensemble des cibles possibles demanderait un travail trop important, nous validerons notre approche sur les bibliothèques de cellules précaractérisées. Cette phase serait basée sur la génération.

Chapitre 2 Arithmétique Mixte

Ce chapitre s'attache à définir l'arithmétique mixte entière qui sera utilisée par la suite. Nous nous intéressons à ses propriétés et aux systèmes de représentations des nombres qui lui sont associés. Mais au préalable, il nous faut définir les arithmétiques classique et redondante qui la composent.

2.1 Arithmétique Classique (NR)

L'arithmétique entière classique est l'arithmétique employée dans la vie courante (base 10) ou en micro-électronique (base 2). Une des caractéristiques essentielles des représentations qui lui sont associées, est qu'à chaque nombre ne peut correspondre qu'un seul codage. Cette propriété induit que ces notations sont compactes.

2.1.1 Représentations

La Numération simple de position (NSP) :

La numération simple de position est la représentation la plus simple et la plus compacte d'un nombre entier positif. Dans ce système de représentation un nombre A en base B sur N chiffres est tel que :

$$A = \sum_{i=0}^{N-1} x_i \cdot B^i \ \ avec \ \ A \ \in \ \{0, ..., B^N - 1\} \ \ et \ \ x_i \in \{0, ..., B-1\} \tag{2.1}$$

La notation en signe et grandeur :

Ce système de notation décompose un nombre en $\pm$ sa valeur absolue. Le signe est généralement représenté par le chiffre de poids fort et la valeur absolue codée en NSP sur les autres chiffres. Dans ce système, un nombre A en base B sur N chiffres, est tel que :

$$A = (-1)^s \cdot \sum_{i=0}^{N-2} x_i \cdot B^i \ avec \ A \in \{-B^{N-1}-1, ..., B^{N-1}-1\}, \ S \in \{0,1\} \ et \ x_i \in \{0, ..., B-1\} \tag{2.2}$$

Cette notation se prête mal au besoin de l'électronique numérique entière. En premier lieu car le zéro peut se coder de deux façons différentes ($-1*0$ ou $1*0$). En second lieu car l'addition de

deux nombres nécessite deux algorithmes différents : un dans le cas ou les signes des opérandes sont identiques et un second quand les signes des opérandes sont différents.

Complément à la base ($C2$) :

Comme pour la notation en signe et grandeur, le complément à la base permet le codage de nombres négatifs. Un nombre A en base B sur N chiffres est tel que :

$$A = -x_{N-1} \cdot B^{N-1} + \sum_{i=0}^{N-2} x_i \cdot B^i \ \ avec \ \ A \ \in \ \{-B^{N-1}, ..., B^{N-1} - 1\} \ et \ x_i \in \{0, ..., B-1\} \quad (2.3)$$

Cette notation ne présente pas les défauts de la précédente. Ici, le zéro ne se code que d'une façon et l'addition de deux nombres de signe opposé ou de même signe, s'effectue avec le même algorithme.
Toutefois, pour certaines opérations, comme l'addition, les entrées doivent impérativement avoir la même taille pour que le calcul soit possible. À cet effet, on fera l'extension du signe de tous les termes trop petits.

2.1.2 Remarques générales :

La numération simple de position et le complément à la base, sont les représentations les plus employées en électronique numérique. Ce sont les notations que nous utiliserons quand nous ferons appel à l'arithmétique entière classique. Nous leurs associerons l'acronyme NR.
Les opérateurs associés à cette arithmétique ont été largement étudiés, et couvrent l'ensemble des besoins en arithmétique ; entre autres l'addition, la multiplication, la division, la racine carrée ou les fonctions trigonométriques.

2.2 Arithmétique Redondante (R)

L'arithmétique redondante doit son nom à la possibilité d'affecter plusieurs codages à un même nombre. Comme nous allons le voir, cette propriété est la source à la fois des principales qualités et des principaux défauts des notations associées à cette arithmétique.

2.2.1 Représentations

La représentation d'Avizienis

Avizienis suggéra, en 1961, l'utilisation d'un nouveau système de notation appelé *système de nombres signés* où les chiffres en base B appartiennent à $\{-a, ..., a\}$ avec $a \leq B-1$ et $2a \geq B-1$.

Il montra que les systèmes tels que $2a > B-1$ autorisent plusieurs représentations d'un nombre et qu'ils sont redondants. Il proposa alors des systèmes où $2a > B$, permettant la réalisation d'additions sans propagation de retenue et donc à temps constants. Malheureusement la base 2 ne satisfait pas à toutes ces conditions. Il est toutefois possible d'imaginer des notations en binaire permettant d'effectuer des additions à temps constant.

On trouve un exemple de système d'Avizienis avec le codage de Booth. Comme nous le verrons dans le chapitre sur les opérateurs, ce codage est employé dans les algorithmes de multiplications où il permet de réduire par deux le nombre de produits partiels.

La représentation Carry-Save (CS)

Dans ce système en base deux, les chiffres sont éléments de $\{ 0, 1, 2 \}$ et leur codage sur deux bits de même poids, est tel que $cs_i = cs_i^0 + cs_i^1$. Respectivement un nombre CS sur N chiffres, est codé sur deux termes tel que $CS = CS^0 + CS^1$. Cette notation permet de représenter aussi bien des nombres non signés (CS_{NS}) que signés (CS_S). Dans les deux cas l'optique d'une arithmétique mixte exploitant à la fois les arithmétiques classique et redondante, impose la compatibilité des diverses représentations. Un CS pouvant être la somme de 2 NRs, les 2 termes d'un CS non signé seront en Numération Simple de Position (NSP) et ceux d'un CS signé seront en Complément à la base 2 ($C2$) :

$$\begin{aligned} CS_{NS} &= \sum_{i=0}^{i=N-1} cs_i^0 \cdot 2^i + \sum_{i=0}^{i=N-1} cs_i^1 \cdot 2^i \\ CS_S &= -(cs_i^0 + cs_i^1) \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} cs_i^0 \cdot 2^i + \sum_{i=0}^{i=N-2} cs_i^1 \cdot 2^i \end{aligned} \tag{2.4}$$

Le codage Carry-Save sur N chiffres permet de représenter des nombres appartenant à l'ensemble $\{ 0, ..., 2^{N+1} - 2 \}$ en non signé et des nombres appartenant à l'ensemble $\{-2^N, ..., 2^N - 2\}$ en signé.
Cette représentation est très employée à l'intérieur des opérateurs utilisant une somme, car elle permet de réaliser des additions totalement parallèles et en temps constant (chapitre 3). Ce système est dit *à retenues conservées*.

La représentation Borrow-Save (BS)

Comme pour la notation Carry-Save, la notation Borrow-Save définit un système en base deux. Les chiffres sont éléments de l'ensemble $\{-1, 0, 1\}$. Leur codage, sur deux bits, est tel que $bs_i = bs_i^+ - bs_i^-$. Respectivement, un nombre BS sur N chiffres est codé sur deux termes tel que

$BS = BS^+ - BS^-$. Pour les mêmes raisons de compatibilité que pour le Carry-Save, chaque terme correspond à un nombre en notation classique :

$$\begin{aligned} BS_{NS} &= \sum_{i=0}^{i=N-1} bs_i^+ \cdot 2^i - \sum_{i=0}^{i=N-1} bs_i^- \cdot 2^i \\ BS_S &= (-bs_{N-1}^+ \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} bs_i^+ \cdot 2^i) - (-bs_{N-1}^- \cdot 2^{N-1} + \sum_{i=0}^{i=N-2} bs_i^- \cdot 2^i) \end{aligned} \quad (2.5)$$

Que les deux termes soient signés ou non signés, un BS sur N chiffres permet de représenter des nombres appartenant à $\{-2^N + 1, ..., 2^N - 1\}$. En fait, une simple permutation des bits de poids forts des deux termes, permet de passer d'une représentation à l'autre. Comme le BS est implicitement une représentation signée, nous utiliserons par la suite uniquement la forme en complément à 2.
Ce système est dit *à retenues conservées négatives*. Il permet d'effectuer des additions parallèles et en temps constant.

2.2.2 Remarques générales

L'atout majeur de l'arithmétique redondante, est de permettre l'addition de deux nombres en temps constant, quelle que soit leur dynamique. Cette propriété a été démontrée par Avizienis pour les *systèmes de nombres signés*. Nous verrons dans le chapitre suivant qu'elle est aussi vraie pour les *notations à retenues conservées*.
Comme l'addition est un opérateur fondamental en traitement du signal, le gain potentiel est important.

Le second point positif à associer à cette arithmétique, découle de l'utilisation des notations redondantes CS et BS à l'intérieur de nombreux opérateurs. La conjugaison de cette pratique à l'emploi des notations redondantes comme représentation de sortie des opérateurs, permet de supprimer les conversions $R \rightarrow NR$ impératives en arithmétique classique. Ces conversions correspondent à une addition en CS, et à une soustraction en BS. Le gain potentiel est là aussi important.

Les notations CS et BS correspondent, en arithmétique classique, à des opérations non encore effectuées. Cette particularité implique qu'il est impossible de connaître la valeur ou le signe d'un nombre redondant sans effectuer au préalable une conversion en notation NR ou une manipulation apparentée à cette conversion. Trois conséquences au moins :

1. Tout opérateur de test général nécessitera au préalable une conversion de redondant vers non redondant.

2. L'addition modulo n'est pas réalisable car la valeur d'un redondant étant inconnue, les dépassements de capacité ne sont pas détectables.
3. Comme pour les tests, les opérateurs booléens ne peuvent s'appliquer qu'après une conversion vers une notation NR.

En fait, toutes les opérations impliquant une transformation ou une manipulation au niveau bit, nécessitent un retour partiel ou complet en NR. Une autre solution consiste à développer un opérateur dédié.
Ces limitations, plus le fait qu'un redondant, codé sur deux NRs prend plus de place, impliquent qu'il est impossible ou aberrant d'utiliser les notations redondantes dans certains contextes. Par exemple, l'adressage de la mémoire, les comparaisons, l'affichage, etc., nécessiteront une conversion en Non Redondant. Autre exemple, la mémorisation en masse ou le décalage de redondants est possible mais serait très coûteux en terme de complexité.
Le champ d'application des redondants est nettement réduit. Il se limite aux blocs arithmétiques et aux connexions locales. Malgré tout, il reste suffisamment étendu pour que les représentations redondantes aient un intérêt réel.
Dernier point : peu d'opérateurs redondants ont été développés. Il nous faudra donc, étudier et développer au minimum les deux opérateurs élémentaires que sont l'addition et la multiplication.

Nous exploiterons indifféremment les notations CS et BS comme représentations redondantes. On associera l'acronyme R pour l'ensemble de ces notations.

2.3 Arithmétique Mixte

Comme nous l'avons indiqué au début de ce chapitre, l'arithmétique mixte entière regroupe les arithmétiques redondante et classique. Le choix d'utiliser l'arithmétique redondante s'explique facilement par les gains potentiels introduits par l'addition à temps constant et par la suppression des conversions $CS \rightarrow NR$ présentes dans de nombreux opérateurs classiques.
Le besoin d'utiliser parallèlement l'arithmétique classique découle en grande partie, des limitations de l'arithmétique redondante : impossibilité d'obtenir certains opérateurs, coût prohibitif des notations redondantes dans certaines conditions, etc., comme nous venons de le voir.
La représentation des entrées/sorties, généralement en notation classique, est une seconde raison. Cette particularité impose l'usage d'opérateurs conventionnels et donc de l'arithmétique classique. Composer avec les non redondants est une obligation pour pouvoir utiliser les redondants.

Dynamique	Non signé	signé
N bits	$NR \in \{0, .., 2^N - 1\}$	$NR \in \{-2^{N-1}, .., 2^{N-1} - 1\}$
$N-1$ bits	$CS \in \{0, .., 2^N - 2\}$	$CS \in \{-2^{N-1}, .., 2^{N-1} - 2\}$
$N-1$ bits		$BS \in \{-2^{N-1} + 1, .., 2^{N-1} - 1\}$

TAB. 2.1 – Intervalle de définition des notations mixtes

Les notations associées à l'arithmétique mixte sont l'union des notations NR et R : principalement la Numération Simple de Position, le Complément à 2, le Carry-Save et le Borrow-Save. Leur usage conjoint impose l'étude de la compatibilité des formats de données et les passages entre représentations.

2.3.1 Compatibilité des données en arithmétique mixte

L'usage commun des notations redondantes et classiques implique la cohérence des formats au niveau le plus élémentaire. Comme nous avons vu dans l'arithmétique redondante, cela se traduit par l'usage des deux formats non redondants NSP et $C2$ pour exprimer les nombres redondants. Ainsi, tous les redondants sont, soit une somme, soit une différence (notation CS ou BS) de non redondants.
Ces opérations implicites impliquent l'extension de 1 bit pour le passage des notations redondantes aux notations classiques. Attention toutefois, en notation non signée, un CS sur $N-1$ chiffres et un NR sur N bits n'ont pas le même intervalle de définition. Ce problème est le même en notation signée table 2.1.
Doit-on alors utiliser un CS ou un BS sur N chiffres pour obtenir l'équivalent d'un NR signé sur N bits ?
En fait c'est un faux problème car, les redondants correspondant à une opération non encore effectuée ($CS = NR + NR$ et $BS = NR - NR$), aucune sortie d'opérateur n'utilise l'intégralité de l'intervalle de définition. Ce qui fait que nous pouvons considérer qu'à un NR sur N bits correspond un redondant sur $N-1$ chiffres.
Cette remarque est très importante car elle implique que la dynamique des redondants sera systématiquement inférieure de 1 chiffre par rapport à celle des NR, pour coder un même **résultat**.

2.3.2 Passerelle entre notations mixtes

Le deuxième point indispensable à l'usage simultané des notations R et NR est de permettre le passage d'une notation à une autre. Nous allons voir que ce passage ne présente pas un coût nul.

Passage des notations Redondantes aux notations Classiques : $R \rightarrow NR$

Le passage des représentations redondantes aux représentations classiques se fait aisément. Suivant le codage, CS ou BS, il suffit d'effectuer la somme ou la différence des deux termes redondants en faisant attention aux opérations signées. Le coût est d'un additionneur conventionnel ce qui n'est pas négligeable.

Passage des notations Classiques aux notations Redondantes : $NR \rightarrow R$

Le passage des représentations classiques aux représentations redondantes demande juste l'adjonction au 1^{er} NR d'un 2^{e} NR. Ce second terme peut être nul ou non nul suivant les propriétés que l'on souhaite donner au nouveau redondant.

Le problème peut s'envisager sous un autre angle. Plutôt que de transformer le NR en redondant, nous allons le conserver tel quel et modifier le ou les opérateurs redondants qui l'emploient. Ces opérateurs, initialement redondants, vont être dégradés en opérateurs mixtes. Le passage d'une ou de plusieurs des entrées, en format NR, va entraîner la suppression de matériel dans l'architecture redondante initiale.

Nous voyons ici l'introduction de toute une gamme d'opérateurs intermédiaires répondant aux besoins de l'arithmétique mixte. L'intérêt est que les architectures correspondantes vont être plus petites et plus rapides que leur équivalent *en tout redondant*. L'exemple le plus frappant est l'addition de deux NR, avec un résultat en Redondant, l'opérateur disparaît purement et simplement ! De manière générale, plus un opérateur redondant a d'entrées non redondantes, plus il est petit et performant.

Passerelles entre les notations CS et BS : $R \rightarrow R$

Les passerelles entre ces deux notations posent deux problèmes. Premièrement, le Borrow-Save étant une notation signée, les passerelles ne sont possibles qu'entre nombres signés. Deuxièmement, pour un codage sur la même quantité de chiffres (prenons N), les ensembles définis par ces deux notations ne sont pas recouvrants : $CS \in \{-2^N, .., 2^N - 2\}$, $BS \in \{-2^N + 1, .., 2^N - 1\}$. Le nombre de chiffres nécessaire à l'expression d'un BS en CS et inversement, augmente de un.

Ce second point est le plus ennuyeux. Pour éviter cette extension de dynamique, il est préférable de tenir compte des deux types de représentations dans la conception des opérateurs mixtes ou totalement redondants.

Pour obtenir la conversion $CS \rightarrow BS$, l'idée consiste à partir de l'équation de fonctionnement

d'un demi-additionneur (*Half-Adder - HA*) :

$$a_i + b_i = 2 \cdot c_{i+1} + s_i \tag{2.6}$$

et de faire le complément de certaines de ces E/S pour modifier ses propriétés. On parle ici de *recodage par retenue* [Daum97, Daum00b]. L'objectif est d'exploiter, au niveau bit, l'égalité

$$\overline{x} = 1 - x \ , \tag{2.7}$$

pour introduire le changement de signe. Considérons le Carry-Save $CS = CS^1 + CS^2$ et le Borrow-Save $BS = BS^+ - BS^-$. Appliquons le chiffre $CS_i = cs_i^1 + cs_i^2$ de CS, sur les entrées d'un HA et utilisons (2.7) sur cs_i^1, cs_i^2 et c :

$$\begin{aligned} cs_i^1 + cs_i^2 &= 2 \cdot c_{i+1} + s_i \\ (1 \cdot 2^i - cs_i^1) + (1 \cdot 2^i - cs_i^2) &= 2 \cdot (1 \cdot 2^i - c_{i+1}) + s_i \\ -cs_i^1 - cs_i^2 + 2 \cdot 2^i &= 2 \cdot 2^i - 2 \cdot c_{i+1} + s_i \\ cs_i^1 + cs_i^2 &= 2 \cdot c_{i+1} - s_i \end{aligned} \tag{2.8}$$

La notation BS apparaît en posant $bs_{i+1}^+ = c_{i+1} - s_i$ et $s_i = bs_i^-$, soit $cs_i^1 + cs_i^2 = 2 \cdot bs_{i+1}^+ - bs_i^-$. Les divers compléments permettent de passer d'une addition à une soustraction, ou d'un Carry-Save à un Borrow-Save. La figure 2.1 présente l'architecture au niveau nombre. Les divers compléments peuvent être fusionnés avec l'architecture du Half-Adder. On obtient ainsi une nouvelle cellule au niveau bit visible dans la bulle figure 2.1.

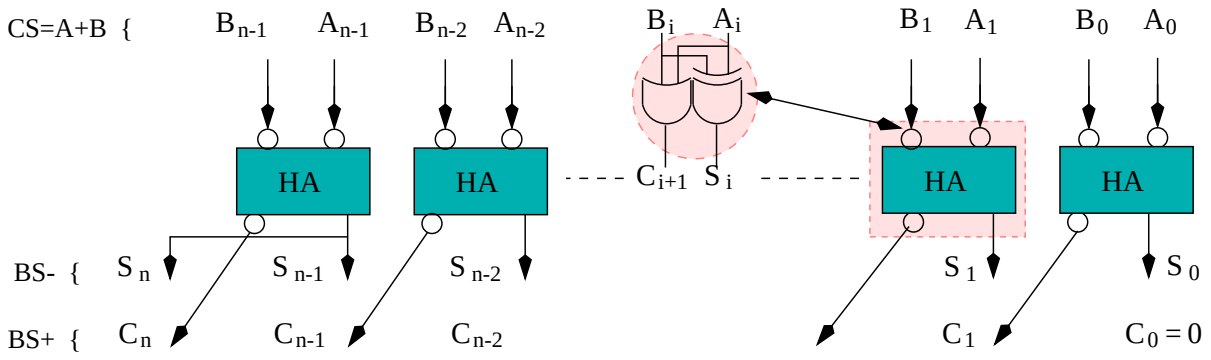


Figure 2.1 – Conversion $CS \rightarrow BS$

La conversion $BS \rightarrow CS$, est aussi obtenue grâce au *recodage par retenue*. Toutefois ici, la démonstration est plus triviale : $BS^+ - BS^-$=$BS^+ + \overline{BS^-} + 1$. L'architecture correspondante est donnée figure 2.2. La structure est la traduction directe de l'addition du terme positif du BS avec le complément à deux du terme négatif du BS.

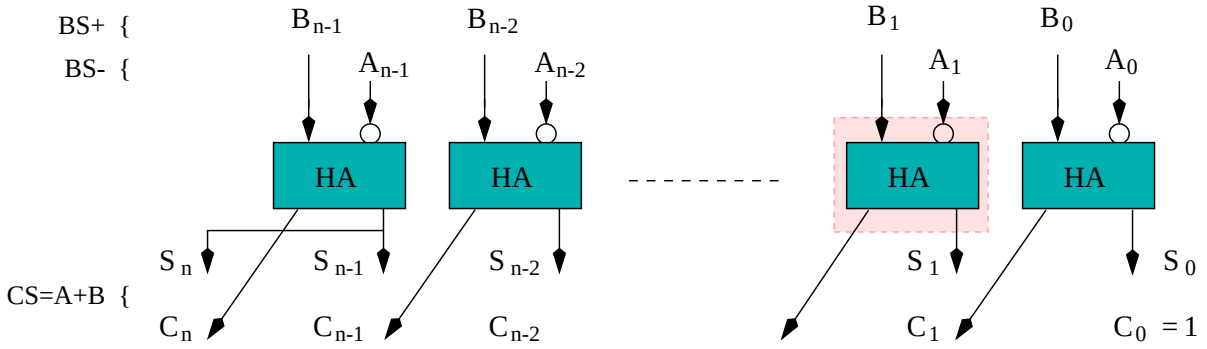


Figure 2.2 – Conversion $BS \rightarrow CS$

Ces deux architectures ont un délai constant quelle que soit la dynamique (N) du redondant en entrée. Ce délai est équivalent à celui d'une porte XOR à deux entrées. La surface à une complexité en $O(N)$. En dehors du problème de dynamique, les conversions $R \rightarrow R$ ont un coût faible.

2.3.3 Remarques générales :

L'adaptation des arithmétiques classique et redondante ne pose pas de problème particulier. La compatibilité au niveau bit est assurée par la composition des nombres redondants en deux nombres classiques et par le respect des extensions des dynamiques entre représentations. Le passage entre notation de même nature (signée / non-signée) est toujours possible. Cependant son coût est rarement nul : additionneurs ($R \rightarrow NR$) ou recodage par retenue ($R \rightarrow R$). Les passages $R \rightarrow R$ impliquent même une extension de dynamique de un chiffre. Seule la conversion $NR \rightarrow R$ ne nécessite aucun matériel.
Pour éviter ces divers coûts, il semble avantageux de conserver au maximum, les notations redondantes à la sortie des divers opérateurs. Cela implique que ces opérateurs doivent accepter l'ensemble des notations mixtes sur leurs entrées/sorties.
Toutefois, si l'ensemble de toutes les combinaisons de représentations possibles est accepté en entrée des opérateurs, alors couvrir ce même ensemble n'est pas nécessaire en sortie de ces opérateurs ; une seule notation par système est suffisante. Cette remarque est importante, car elle permet d'éviter le problème d'extension de dynamique des conversions $R \rightarrow R$, et réduit le nombre de variante d'architecture à développer par opérateur.
On pressent déjà que les gains potentiels, liés à l'ajout de l'arithmétique redondante aux côtés de l'arithmétique classique, viendront en partie des enchaînements d'opérateurs, grâce à l'économie des conversions de format.

2.4 Conclusion

Les performances connues et largement exploitées des additionneurs redondants, alliées à la disparition des conversions de format $R_{(CS,BS)} \rightarrow NR_{(NSP,C2)}$ font apparaître l'arithmétique redondante comme très attrayante. De fait elle l'est.
Toutefois, les opérations inachevées que représentent les notations Carry-Save (addition) et Borrow-Save (soustraction), le coût 2 fois plus élevé de la mémorisation des nombres correspondants, et la représentation des entrées/sorties généralement en notations conventionnelles, diminuent fortement le champ d'application de l'arithmétique redondante. Ainsi les opérations logiques, les manipulations au niveau bit, les tests et la mémorisation de masse, ne sont pas directement réalisables en arithmétique redondante.
Composer avec l'arithmétique classique est une obligation. L'arithmétique mixte ainsi définie, permet de répondre à l'ensemble des besoins arithmétiques. L'adaptation des deux systèmes de notations, ne pose pas de problèmes fondamentaux. Cependant, l'usage combiné des diverses notations classiques et redondantes, nécessite le développement d'opérateurs arithmétiques couvrant toutes les combinaisons : notations redondantes/notations conventionnelles, sur leurs entrées/sorties.
Il est important de noter que les diverses limitations de l'arithmétique redondante font que les notations associées sont destinées aux portions d'architectures définissant au minimum un enchaînement d'opérateurs (chemins de données par exemple). De fait, l'arithmétique redondante est difficilement exploitable dans les parties de contrôles où les opérateurs arithmétiques sont généralement isolés.

Chapitre

3 Opérateurs élémentaires en arithmétique mixte : Architectures

3.1 Quelques remarques préliminaires

Ce chapitre est consacré à la réalisation des architectures des opérateurs arithmétiques élémentaires que sont l'addition, par extension la somme, et la multiplication en arithmétique mixte. L'objectif est de répondre à l'ensemble des combinaisons classique/redondant en entrée/sortie tout en limitant le nombre d'architectures développées.

3.1.1 Choix des opérateurs arithmétiques

Le choix de ces trois opérateurs arithmétiques n'est pas fortuit :

- L'addition est la brique arithmétique la plus utilisée dans la conception de circuits $VLSI$. Elle est présente à la fois dans les parties calculatoires : chemins de données, et dans les parties de contrôles. Du point de vue $VLSI$, elle ne concerne que les sommes de deux nombres.
- La sommation est l'extension à N nombres de l'addition ($N > 2$). Sémantiquement elle reste une addition. Toutefois sa réalisation matérielle diffère fortement de celle de l'addition. Cette opération est présente à la fois dans de nombreux algorithmes et dans de nombreux opérateurs élémentaires comme la multiplication et la division. La particularité de cette opération est la variabilité de son nombre d'entrées.
- La multiplication est la clé de voûte de nombreuses applications de traitements du signal et de l'image. Elle est souvent associée à la sommation : filtrages numériques ou réseaux de neurones par exemple. Ce duo est à l'origine d'une des principales instructions des processeurs dédiés aux traitements du signal et de l'image (DSP) : la multiplication accumulation (MAC).

À eux trois, ces opérateurs arithmétiques constituent l'essentiel des calculs effectués en traitement du signal et de l'image.

3.1.2 Problématique des entrées/sorties

Dans le chapitre précédent, l'arithmétique mixte a été définie comme la fusion des arithmétiques redondante et classique. Cela implique que les entrées/sorties (E/S) des opérateurs sont définissables comme redondantes ou non redondantes. Il en découle que, pour un usage total de l'arithmétique mixte, les opérateurs doivent impérativement répondre à tous les cas de figure E/S : redondant, non redondant et mixte. Pour être complètes les architectures développées doivent en plus, tenir compte des diverses associations signées / non signées (S/NS) des opérandes. La gestion de ce problème n'implique que les entrées.

Remarque sur les sorties

Avant tout, il est important de noter que l'arithmétique mixte ayant pour but de généraliser l'emploi des notations redondantes, les sorties des opérateurs partiellement ou totalement redondants sont en représentations redondantes.
Une rapide analyse des principaux opérateurs existants en arithmétique conventionnelle : additionneur, multiplieur, diviseur, somme, etc., fait apparaître que l'implémentation d'une opération se divise en deux parties distinctes : la réalisation de l'algorithme proprement dit, le résultat étant généralement exprimé en notation redondante, et un additionneur conventionnel permettant de convertir le résultat en notation classique. Ce constat permet de dire que l'obtention d'une sortie d'opérateur dans une arithmétique donnée, ne pose aucun problème. Ce postulat est vérifié par les opérateurs développés dans ce chapitre.
Dans le cas des opérateurs partiellement ou totalement redondants, le choix de la notation de sortie allié à la bi-composition des architectures, se traduit par la disparition de la conversion $R \rightarrow NR$ énoncée dans le chapitre 2 section 2.4.

Représentations multiples des entrées

La représentation des sorties n'étant plus un problème, il faut maintenant gérer la variété de notations possibles en entrée. Afin de simplifier les développements, chaque opération est décomposée en trois *sous* opérateurs : les opérateurs purement non redondants, c'est à dire les opérateurs actuels de l'arithmétique classique, les opérateurs purement redondants où toutes les entrées sont redondantes, et les opérateurs partiellement redondants, soient tous les autres cas de figures. Pour ce troisième groupe, nous parlerons d'opérateurs mixtes, la mixité venant de l'usage simultané de différentes arithmétiques (redondant et classique) pour décrire les entrées. Plus tard, nous rajouterons à ce groupe les opérateurs dont les notations d'entrées et de sortie(s) diffèrent : sortie(s) redondante(s) et entrées classiques ou le contraire.

3.1.3 Remarques sur les architectures

L'ensemble des combinaisons R/NR et S/NS est considérable. Le nombre d'architectures potentielles l'est aussi. Pour limiter ce nombre, nous aurons comme souci permanent de réutiliser au maximum les mêmes structures. Deux des solutions adoptées consistent à ajouter un prétraitement à une architecture existante (complément à deux pour la soustraction ou du terme négatif d'un BS par exemple) et à composer un nouvel opérateur en utilisant ceux précédemment étudiés (usage de l'addition ($R \rightarrow NR$) et de la sommation dans les multiplieurs par exemple).

Toutes les architectures proposées et étudiées dans ce chapitre, ont leurs entrées/sorties parallèles et sont purement combinatoires. Les chiffres indicés 0 correspondent aux chiffres les moins signifiants (LSB).

Remarque sur les architectures redondantes et mixtes

L'ajout de l'arithmétique redondante à l'arithmétique classique pour composer l'arithmétique mixte, ne se justifie que si les performances des opérateurs associés sont meilleures ou équivalentes à celles des opérateurs classiques. Meilleures car il en va de l'intérêt même de l'arithmétique mixte. Équivalentes car si un opérateur n'apporte pas de gain par ses performances propres, il le permettra par son enchaînement avec d'autres opérateurs partiellement ou totalement redondants, qui eux sont plus performants. Au pire, le gain serait nul.

Remarque succincte sur les architectures conventionnelles

Nous n'avons pas ici l'intention de faire une nouvelle fois l'étude des architectures associées à l'addition et à la multiplication en arithmétique classique. Il existe différentes thèses, articles et livres qui traitent parfaitement du sujet. Nous nous contenterons de présenter les additionneurs et les multiplieurs les plus généralement utilisés. Notre intérêt portera plutôt sur leurs performances intrinsèques. Elles serviront de référence à l'évaluation des opérateurs mixtes et totalement redondants.

3.2 Addition entière

L'addition est l'opération la plus rencontrée dans la conception de circuits dédiés aux traitements du signal et de l'image. Elle est présente à la fois dans les chemins de données, pour les traitements proprement dits, et dans les parties de contrôles.

Dans cette section, nous ne considérerons que l'addition de 2 nombres et cela quelles que soient les notations utilisées. Nous appellerons *additionneurs* les architectures qui lui sont associées.

Contraintes d'utilisations

Plusieurs autres opérations ou fonctionnalités sont associées à l'addition. En vrac, on peut citer sa jumelle la soustraction, les arrondis, les comparaisons et les opérations dégradées comme le comptage et les incrémentations / décrémentations. Chacune a des besoins spécifiques nécessitant l'adjonction de mécanismes particuliers.
Un bon nombre impliquent une addition modulo 2^N. D'ailleurs, dans de nombreux circuits comme les processeurs, l'addition de deux termes est systématiquement effectuée au modulo de la dynamique de ses entrées. Par déformation, l'adjonction de cette opération est souvent considérée comme partie intégrante de l'addition *(sic)*.
Il apparaît une autre contrainte dans cet exemple : le calcul itératif. Celui-ci permet l'addition de nombres de dynamique M sur un même additionneur modulo 2^N ($N < M$). Plusieurs passes sont nécessaires au calcul. Pour assurer un résultat juste, à chaque passe l'additionneur doit fournir une indication du dépassement de capacité (retenue sortante), qui sera recyclée sous forme d'une retenue entrante dans la passe suivante. Les mécanismes associés sont connus.
L'indication de dépassement est aussi employée dans les mécanismes de calcul cyclique et de saturation. La retenue entrante est, elle, utilisée dans la gestion d'arrondis ou dans la soustraction : communément le +1 du complément à deux d'un nombre à soustraire ($-A = \overline{A} + 1$).
La multitude d'usages possibles fait que les contraintes imposées aux architectures peuvent fortement différer. La question qui se pose est : Dans le contexte qui nous intéresse, doit-on prendre en compte toutes ces contraintes ? Ou plutôt, sachant que les additionneurs classiques répondent déjà à toutes ces particularités, les additionneurs mixtes et redondants doivent-ils répondre à tous ces besoins ?

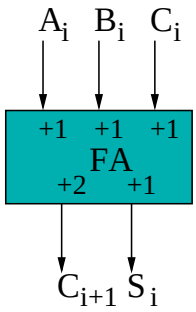
Choix architecturaux

La principale limitation de l'arithmétique redondante (pour mémoire l'impossibilité de connaître la valeur d'un nombre sans retour en notation NR) exclut d'emblée les opérations de modulo, tout ce qui est opérations de comptage / décomptage et cyclique, de saturation et de comparaison.
Par contre, la gestion d'une retenue entrante et la génération d'une retenue sortante ne pose pas de problème particulier. Toutefois, en arithmétique redondante, la retenue sortante ne correspond pas au dépassement de capacité. Son utilité en dehors du calcul itératif semble compromise.

La seule vraie contrainte à laquelle doivent répondre les architectures mixtes et redondantes, est la gestion de retenue(s) entrante(s). Dans tous les autres cas les choix architecturaux s'orienteront vers les additionneurs classiques.
Nous ne reviendrons pas sur ces problèmes de retenue et de modulo par la suite. Nous nous contenterons de présenter les éventuelles modifications architecturales indispensables à la prise en compte de ces contraintes quand elles existent.

3.2.1 Addition élémentaire

Nombre d'architectures présentées en aval, sont construites à partir de la même cellule élémentaire : l'additionneur complet (Full-Adder - FA) figure 3.1. Cette cellule effectue l'addition de trois bits de même rang et code le résultat sur deux bits tel que la somme pondérée des sorties est égale à la somme pondérée des entrées : $A_i + B_i + C_i = 2 * C_{i+1} + S_i$ avec i le poids du bit considéré.



A_i	B_i	C_i	C_{i+1}	S_i	*logique positive*	*logique negative*
0	0	0	0	0	0	3
0	0	1	0	1	1	2
0	1	0	0	1	1	2
0	1	1	1	0	2	1
1	0	0	0	1	1	2
1	0	1	1	0	2	1
1	1	0	1	0	2	1
1	1	1	1	1	3	0

$S_i = A_i \oplus B_i \oplus C_i$ *somme modulo 2*
$C_{i+1} = (A_i \& B_i) + (A_i \& C_i) + (B_i \& C_i)$ = majorité(A_i,B_i,C_i)

Figure 3.1 – Additionneur élémentaire : synoptique et table de vérité

Il est intéressant de noter que cette cellule a un comportement identique que l'on soit en logique positive ou en logique négative.

De nombreuses architectures réalisent cette fonction. Certaines favorisent la propagation de la retenue entrante vers la retenue sortante. On parlera de structure asymétrique. D'autres sont orientées vers la limitation des commutations transitoires (glitches), par exemple en équilibrant les chemins temporels des deux sorties.
Les premières ont été développées essentiellement pour améliorer les performances en temps des additionneurs dont la limitation temporelle vient de la chaîne de propagation des retenues.

Les secondes sont, elles, plus intéressantes pour les structures où la propagation se fait à travers plusieurs étages de Full-Adders : par exemple les sommations section 3.3 en aval. Dans ce cas, l'objectif est la réduction de la consommation.
Toutefois, peu de bibliothèques de cellules pré-caractérisées offrent plusieurs choix de structures de Full-Adder, ce qui est bien dommage.

Nous l'avons dit, de nombreux additionneurs sont basés sur le Full-Adder. En conséquence, nous allons considérer comme unitaire son temps de propagation et sa taille. Cette considération facilitera la comparaison de la complexité des diverses architectures proposées.

3.2.2 Addition Entière non redondante, signée et non signée

L'objectif est de proposer une gamme d'additionneurs aux propriétés diverses, permettant de couvrir tous les besoins de l'arithmétique mixte, et offrant différents points de comparaisons pour évaluer les opérateurs mixtes et redondants. Trois types d'architectures ont été retenus : les additionneurs séquentiels (*Carry Ripple Adder*), les additionneurs à saut de retenue (*Carry Skip Adder*) et les additionneurs à retenues anticipées (*Carry Lookahead Adder - CLA*).

Dans un premier temps, la présentation ne concernera que les architectures en notation non signé. Ensuite nous présenterons une solution simple et commune à tous les additionneurs, permettant de gérer les signes. En troisième lieu, nous aborderons le problème de la soustraction.

Additionneurs à propagation de retenue séquentielle

Cet algorithme est la transposition de la technique que nous employons couramment pour sommer deux nombres. Il consiste à faire, tour à tour, l'addition élémentaire de deux chiffres de même rang avec la retenue du rang précédent. Le calcul est séquentiel et s'effectue des chiffres les moins signifiants vers les chiffres les plus signifiants figure 3.2. Le calcul suit la propagation des retenues. On parlera de *Carry Ripple Adder*.
On peut noter que dans le cas où la retenue entrante C_{in} est non utilisée, le premier Full-Adder peut-être transformé en demi-additionneur (Half-Adder). L'expression du dépassement de capacité n'est autre que le bit de poids fort S_n si l'addition est effectuée modulo 2^N. Aussi, en non signé, l'architecture correspondant à l'addition modulo 2^N est identique.
Cet additionneur présente une complexité en surface et en délai en $O(N)$, N étant la plus grande dynamique des deux nombres à sommer. Le délai est essentiellement l'expression de la

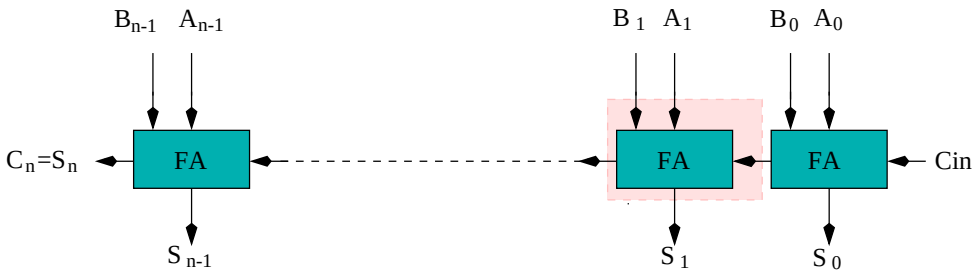


Figure 3.2 – Additionneur séquentiel non signé

propagation de la retenue la moins signifiante à travers toute l'architecture. Les opérateurs qui suivent ont pour objectif de réduire ce chemin de propagation.

Anticipation de la retenue

Une des solutions pour améliorer les performances en temps de ce premier additionneur, est d'anticiper la progression des retenues. La technique consiste à calculer les retenues intermédiaires en s'appuyant sur deux signaux : Un signal de propagation (P) signifiant qu'une retenue entrante va se propager dans l'additionneur sans être modifiée, et un signal de génération (G) traduisant la création d'une retenue dans cet additionneur : $R_i = G_{i-1} + P_i$ & $R_{entrante}$. On calcule ensuite les sommes : $S_i = P_i \oplus R_i$.
Le tableau suivant reprend le calcul des signaux de propagations et de générations pour la somme de deux opérandes.

A_i	B_i	P_i	G_i	Retenue
0	0	0	0	Absorption
0	1	1	0	Propagation
1	0	1	0	Propagation
1	1	0	1	Génération

TAB. 3.1 – Table de vérité des signaux de propagation et de génération

Les équations de P_i et G_i correspondent respectivement aux portes logiques *ou exclusif* et *et* à deux entrées. L'ensemble lui, correspond à un demi-additionneur : Half-Adder.

Additionneurs à saut de retenue

L'additionneur à saut de retenue s'appuie seulement sur le mécanisme de propagation. Un exemple d'architecture est donné figure 3.3, pour des entrées sur 10 bits. Il fonctionne en deux temps. En premier et dans chaque segment, les signaux de propagations (P) sont calculés pour chaque paire de bits de même poids, puis combinés pour sélectionner la propagation de la

retenue entrante ou de la retenue sortante. En second, chaque segment effectue son morceau de somme, comme un additionneur à propagation de retenue séquentielle, en prenant comme retenue entrante la retenue sélectionnée par le segment précédent.

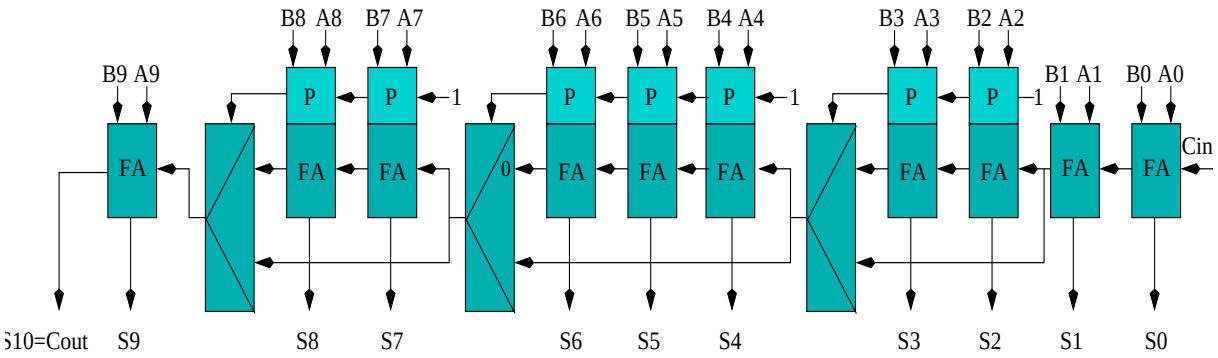


Figure 3.3 – Additionneur 10 bits à saut de retenue en non signé

Comme pour l'additionneur à propagation de retenue séquentielle, l'expression du dépassement de capacité n'est autre que le bit de poids fort S_n, et l'architecture permettant l'addition modulo 2^N reste la même.

Il existe différentes techniques pour ajuster la taille de l'additionneur séquentiel interne à chaque segment ainsi que pour réduire le matériel utilisé par chaque paire $P - FA$ [Parh99, Kant00]. Toutefois la complexité reste la même en délai et en surface. Ce type d'additionneur permet de passer d'une propagation en $O(N)$ à $O(\sqrt{N})$. La surface reste, elle, en $O(N)$.

Additionneurs à retenue anticipée

Cette famille d'additionneurs exploite complètement les mécanismes d'anticipation de retenue. Le fonctionnement de ces additionneurs s'appuie sur le calcul récursif des signaux g_i, p_i, $G_{i,j}$ et $P_{i,j}$ tel que :

$G_{i,i}$	=	$G_i = A_i \,\&\, B_i$	*génération de retenue au rang i*
$P_{i,i}$	=	$P_i = A_i \oplus B_i$	*propagation de retenue au rang i*
$G_{i,k}$	=	$G_{i,j} + P_{i,j} \,\&\, G_{j-1,k}$	*génération entre les rangs i et k* ($N \geq i \geq j \geq k \geq 0$)
$P_{i,k}$	=	$P_{i,j} \,\&\, P_{j-1,k}$	*propagation de la retenue du rang k au rang i*
R_i	=	$G_{i,k} + P_{i,k} \,\&\, R_k$	*retenue au rang i+1* ($R_k = G_{k-1,0}$)
S_i	=	$p_i \oplus R_{i-1}$	*bit de la sortie au rang i*

Nous parlerons ici de propagation et de génération de groupe. L'ensemble des architectures correspondantes se décompose en trois parties. La première calcule les p_i et les g_i à partir des

opérandes en entrée. La seconde permet de générer les signaux $P_{i,k}$ et $G_{i,k}$ utiles au calcul des retenues intermédiaires (R_i), et de la retenue au rang n (R_{n+1}). L'élaboration de ces signaux est faite à l'aide d'arbre binaire. Différentes structures ont été proposées [Mull97, Zimm99, Parh99]. La dernière partie calcule les retenues R_i et les bits S_i du résultat de l'additionneur. À chaque partie sont associées les différentes cellules élémentaires données figure 3.4.

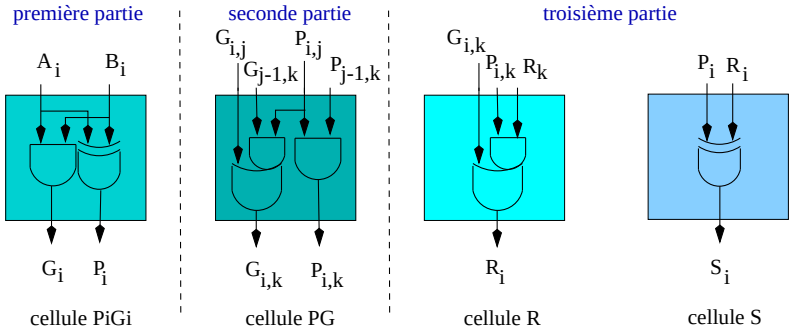


Figure 3.4 – Addition à retenue anticipée : Cellules élémentaires

Nous nous sommes intéressés plus particulièrement à la structure d'additionneur CLA *Carry Lookahead Adder* ou additionneur de *Sklansky* [Skla60]. Ce choix s'explique par les performances et la régularité de la structure proposée. La figure 3.5 présente un additionneur 8 bits comme exemple.

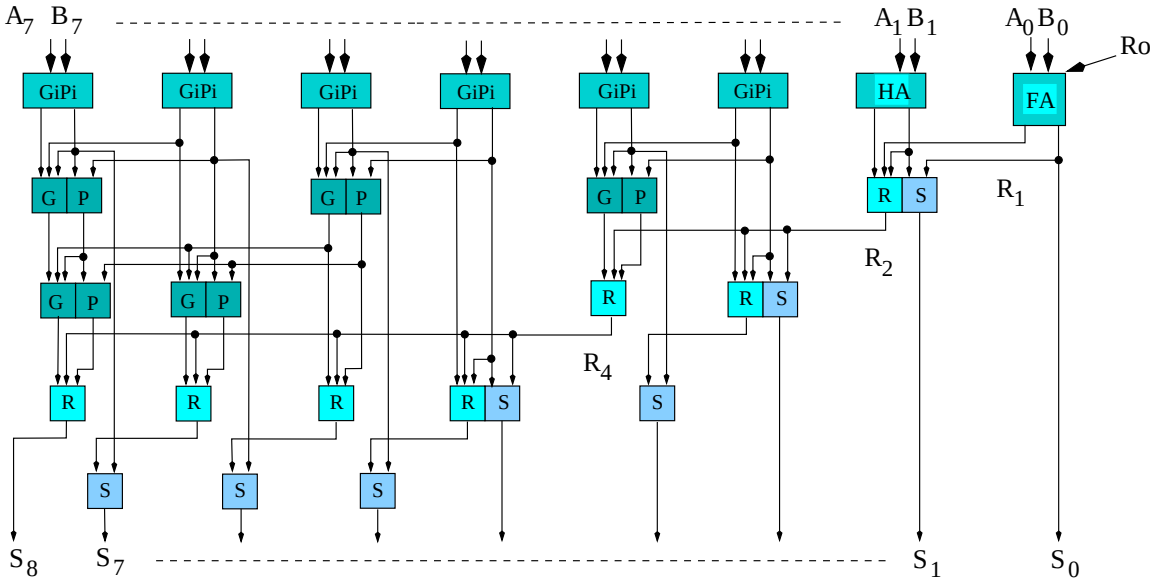


Figure 3.5 – Additionneur 8 bits à retenue anticipée

On retrouve dans cette architecture les trois phases de calculs. La prise en compte de la retenue entrante ne demande qu'une très légère modification du premier P_iG_i. Ce changement

équivaut à la substitution du *Half-Adder* par un *Full-Adder*.

Le principal reproche fait à cette structure, est la forte sortance des signaux de retenues intermédiaires. Ce problème est résolu facilement en amplifiant les signaux en questions.

Ici aussi, l'expression du dépassement de capacité est le bit de poids fort S_n. L'architecture modulo 2^N est la même. Cet additionneur est le plus rapide des trois. Sa structure en arbre binaire permet une propagation en $O(log_2(N))$. Sa surface est en $O(N.log_2(N))$ ce qui correspond à une croissance plus importante que pour les deux additionneurs précédents.

Gestion de signe

La gestion de signe ou plutôt des bits de signes pose plusieurs problèmes. Il faut à la fois prendre en compte l'ensemble des combinaisons signées/non signées en entrée, gérer l'addition modulo et générer éventuellement la détection de dépassement de capacité.

Deux orientations sont possibles. Soit développer des variantes des architectures initiales, soit déterminer une méthode permettant de couvrir l'ensemble des cas avec les mêmes architectures. Notre optique étant de réduire le nombre d'architectures développées, notre choix s'est porté sur la deuxième solution.

Direct	Extension de signe		
000 (S)	0000 (S)	1100 (S)	0100 (NS)
+100 (S)	+1100 (S)	+1100 (S)	+1110 (S)
$\overline{0100}$ (S)	$\overline{1100}$ (S)	$\overline{1000}$ (S)	$\overline{0010}$ (S)
Faux dans le cas de l'extension de dynamique	Résultat juste avec ou sans extension de dynamique (modulo)	MSBs différents, dépassement de capacité en modulo	Prise en compte d'une entrée non signée

Figure 3.6 – Additionneur : gestion de signe

Pour résoudre directement l'ensemble de ces problèmes, l'idée est simple. Il suffit d'effectuer l'extension des signes de un bit, faire le calcul avec un additionneur non signé et ne garder que les bits correspondants à la bonne dynamique de sortie.

La figure 3.6 illustre ce principe avec plusieurs exemples. Les (S) indiquent des nombres signés, les (NS) les nombres non signés. Dans le premier cas, le calcul est fait sans extension de dynamique. Le résultat est faux en pleine dynamique, et juste en modulo. Le second cas reprend le même exemple mais avec une extension de dynamique. Le résultat est juste à la fois, en modulo et en pleine dynamique. Le troisième exemple illustre un dépassement de capacité. Si les deux

bits de poids fort diffèrent, il y a dépassement de capacité en modulo. Le quatrième cas donne la solution pour prendre en compte une entrée non-signée (création d'un bit de signe).
Cette solution n'est certes pas la plus efficace en terme de délai ou de surface. Cependant, les écarts sont faibles voire inexistants, et surtout chaque architecture permet de répondre à l'ensemble des combinaisons signées/non signées en entrée.
Le dépassement de capacité est obtenu à l'aide d'une simple porte *ou exclusive* : $OVR = S_{MSB} \oplus S_{MSB+1}$. En effet, en l'absence de dépassement de capacité, le chiffre supplémentaire S_{MSB+1} équivaut à une extension du signe, et dans le cas contraire (le résultat dépasse la dynamique de sortie) le MSB diffère du bit supplémentaire.

Soustraction

Grâce à la notation en complément à deux, la soustraction en arithmétique classique, est facile à obtenir. Il suffit juste d'effectuer le complément à deux du terme à soustraire avant de l'ajouter au deuxième terme. Soit A et B deux nombres en notation classique alors $A - B = A + \overline{B} + 1$. L'adaptation des architectures présentées est facile : une ligne d'inverseurs pour obtenir $\overline{B}$ et la retenue entrante câblée à 1.

3.2.3 Addition Entière mixte, signée et non signée

Cette addition réalise la somme d'un nombre redondant avec un nombre non redondant. Rapporté en arithmétique classique, le nombre de termes à sommer est de trois. Le résultat est en notation redondante. Comme l'additionneur élémentaire (FA) effectue la somme de trois bits et donne un résultat sur 2 bits, il sera le cœur des architectures des additionneurs mixtes.
Dans un premier temps, les redondants sont considérés qu'en notation CS. La notation BS est traitée avec la soustraction car elle contient une soustraction implicite.

Addition mixte non signée

Obtenir l'architecture mixte en notation non signée ne pose aucune difficulté (figure 3.7). La structure est composée exclusivement de FA. Chacun d'eux effectue la somme des bits de même poids parallèlement. La sortie est en CS. Pour assurer sa cohérence : deux NR de même taille, le bit S_n est mis à zéro. De même, le bit de poids faible C_0 est mis à zéro, s'il n'est pas utilisé pour gérer une retenue entrante. Malgré la différence de notation des deux opérandes, les entrées de l'architecture proposée restent commutatives.
Cette architecture a un temps de propagation constant, quelle que soit la dynamique N des entrées. Cette propriété est due à l'absence de propagation de retenue entre FA. Cette structure

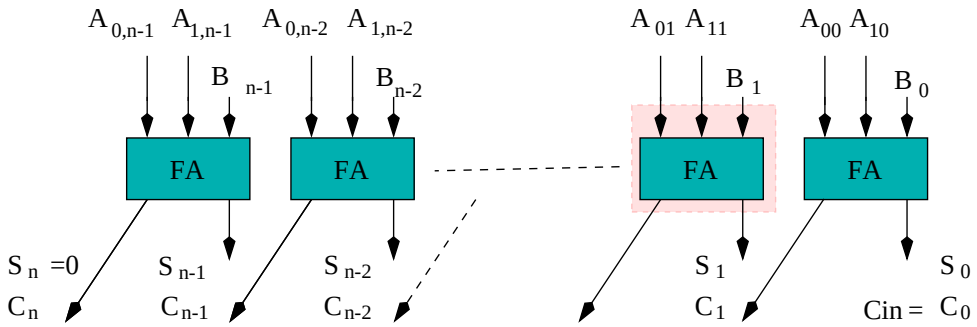


Figure 3.7 – Addition mixte non signée

illustre bien l'appellation de *notation à retenue conservée* employée pour les notations CS et BS. La surface et la consommation de cet opérateur sont en $O(N)$. Le traitement est uniforme.

Addition mixte signée

La gestion d'opérandes signés (R_{CS} et NR_{C2} pour l'instant) implique juste une modification mineure de l'architecture non signée figure 3.8. Le mise à zéro systématique du bit de poids fort S_n, est remplacé par la copie du dernier bit de somme calculé S_{n-1}.
Cette modification correspond à une extension de signe. En effet, le dernier FA additionnant que des bits de signes, ses résultats sont aussi des bits de signes de poids $n-1$ et n. Cette extension est nécessaire pour assurer la cohérence de format du CS de sortie : deux termes de même taille utilisant la même notation classique.

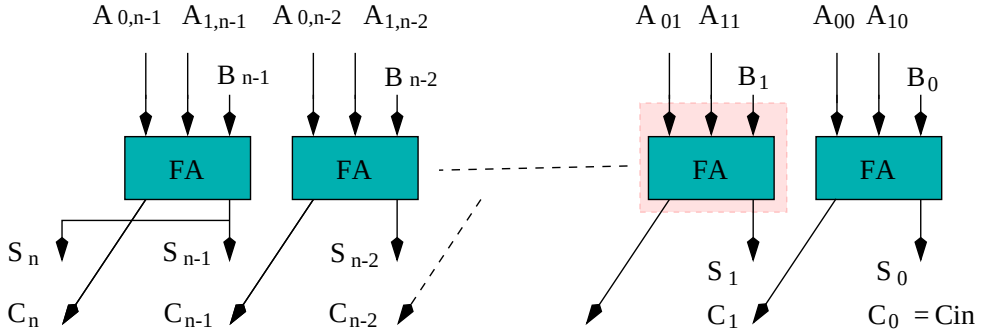


Figure 3.8 – Addition mixte signée

Cette architecture a les mêmes propriétés que la précédente : propagation en temps constant, surface et consommation en $O(N)$, une retenue entrante possible, un traitement uniforme et la commutativité des entrées.
Remarque : Matériellement parlant, les architectures non signées et signées étant identiques, la technique de gestion de signe par extension des bits de signes utilisée en arithmétique classique, n'est pas nécessaire.

Soustraction

En mixte, la soustraction, qu'elle soit implicite ($NR + BS$) ou explicite, n'est pas aussi triviale qu'en arithmétique classique. Le problème vient de la quantité de termes à soustraire. On distingue deux cas : les opérations ayant seulement un terme à soustraire : $CS - NR$, $NR - BS$ et $BS + NR$, et les opérations ayant deux termes à soustraire : $NR - CS$ et $BS - NR$. Dans le premier cas, la solution est identique à celle employée en arithmétique classique : complément à deux du terme à soustraire et utilisation de la retenue entrante. Dans le deuxième cas, la technique du complément à deux n'est pas exploitable car il faudrait deux retenues entrantes. La solution consiste à utiliser le recodage par retenue, déjà employé pour les conversions $R \rightarrow R$. Cette fois, le cœur du calcul ne sera pas un HA mais un FA. Soit l'équation d'un FA

$$x_i + y_i + z_i = 2.c_{i+1} + s_i \tag{3.1}$$

où i correspond au poids des divers bits. Appliquons (2.7) à (3.1) sur x et s nous obtenons

$$\begin{aligned} \overline{x_i} + y_i + z_i &= 2.c_{i+1} + \overline{s_i} \\ (1.2^i - x_i) + y_i + z_i &= 2.c_{i+1} + (1.2^i - s_i) \\ -x_i + y_i + z_i &= 2.c_{i+1} - s_i \\ +x_i - y_i - z_i &= -2.c_{i+1} + s_i \end{aligned}$$

Étendons (3.2) à un vecteur, nous avons

$$X - Y - Z = -2.C + S\,. \tag{3.2}$$

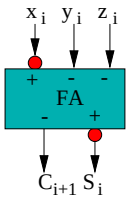


Figure 3.9 – Cellule élémentaire de soustraction

La cellule de base obtenue est dite PPM [Dupr91]. Ce résultat peut-être obtenu par une voie différente. Considérons un additionneur mixte dont toutes entrées/sorties sont négatives

$$-X - Y - Z = -2.C - S \tag{3.3}$$

l'architecture et la cellule élémentaire (ici un FA) sont alors inchangées. Appliquons le complément à deux à X et à S, on obtient

$$\begin{aligned} -\overline{X} - Y - Z &= -2.C - \overline{S} \\ -(-X + 1) - Y - Z &= -2.C - (-S + 1) \\ +X - Y - Z &= -2.C + S \end{aligned} \tag{3.4}$$

La figure 3.9 accompagnant la démonstration, présente la nouvelle cellule élémentaire obtenue. Cette cellule est toujours la même, y compris pour les bits de signes. Le résultat est en notation Borrow-Save. X, Y et Z seront affectés suivant l'opération souhaitée : $NR - CS$ ou $BS - NR$. Les propriétés des deux soustracteurs mixtes sont équivalentes à celles des additionneurs mixtes.

3.2.4 Addition Entière redondante, signée et non signée

Cette addition fait la somme de deux nombres redondants. Le nombre équivalant d'opérandes en arithmétique classique, est de quatre. La sortie est en notation redondante. Dans un premier temps et comme pour les additionneurs mixtes, nous ne considérons les nombres redondants qu'en notation CS. L'impact des BS est étudié ultérieurement avec la soustraction.

Addition redondante non signée

Plusieurs structures sont possibles pour réaliser un additionneur totalement redondant. La première consiste à réutiliser l'additionneur mixte de la section précédente. L'architecture obtenue figure 3.10, est basée sur l'enchaînement de deux additionneurs mixtes. Le premier réduit le nombre de termes à sommer de quatre à trois, le second de trois à deux. La sortie est en notation Carry-Save. Nous dirons de cette structure qu'elle est de type 3/2.
Quelle que soit la dynamique (N) des opérandes, cette architecture présente un temps de propagation en temps constant. La surface et la consommation sont en $O(N)$. Les valeurs de ces critères sont sensiblement égales à deux fois celles de l'additionneur mixte correspondant. Cette architecture autorise deux retenues entrantes.

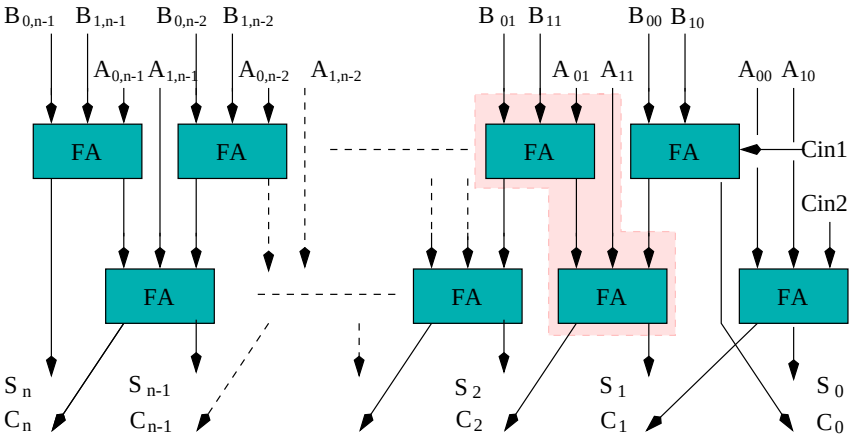


Figure 3.10 – Addition redondante non signée de type 3/2 (Full-Adder)

La deuxième structure part d'un constat simple : dans la structure 3/2, le traitement n'est pas totalement parallèle. En effet la quatrième entrée n'est prise en compte que dans le second étage. L'idée consiste donc à substituer les éléments traitant les bits de même poids (partie colorée / grisée sur la figure 3.10) par une autre structure plus parallèle ; appelons la *structure de type 4/2* figure : 3.11. Dans cette architecture, les quatre entrées sont prises en compte en même temps. De façon similaire à la structure type 3/2, une cinquième entrée *tardive* est gérée et il existe une sortie *précoce*. Le fonctionnement est basé sur la décomposition du résultat en

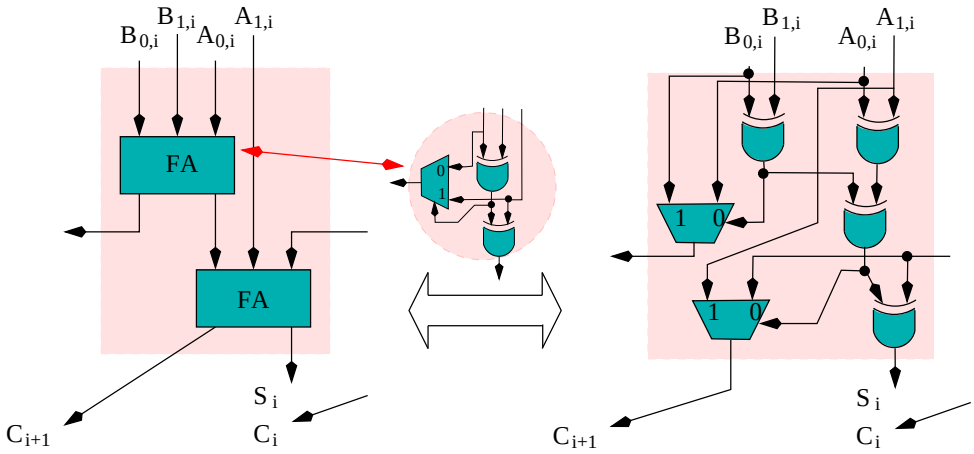


Figure 3.11 – Addition redondante non signée de type 4/2 : équivalence de traitement

deux retenues et un bit de somme, et sur l'exclusivité des signaux intermédiaires qui permettent d'intégrer l'entrée *tardive* dans le calcul.
La figure 3.11 présente l'architecture en porte correspondante. En décomposant aussi en porte la cellule de type 3/2, on constate que la surface est identique et que la mise en parallèle du traitement permet de réduire de 25% la chaîne longue (4 $\text{XOR}_{HA} \rightarrow 3\ \text{XOR}_{HA}$). Le temps de propagation est constant et la surface est en $O(N)$. Autre point intéressant, tous les chemins sont équilibrés ce qui est important pour la consommation.

Addition redondante signée

La prise en compte des notations signées ne demande qu'une légère modification des architectures non signées. Cette altération est située au niveau de la cellule élémentaire traitant les bits de poids forts (bits de signes), comme l'illustre la vue complète d'un additionneur signé en structure 3/2 : figure 3.12. La modification correspond à la substitution du Full-Adder du second étage par la cellule présentée figure 3.9. Elle est rendue nécessaire par la différence de signe de ses diverses entrées : les quatre entrées primaires se composent des bits de signes des deux redondants à additionner, et l'entrée retardée prend un bit positif.
Il est intéressant de noter que les modifications introduites portent uniquement sur l'interface de la cellule de poids fort initiale (inversion de l'entrée retardée et de la sortie S_{n-1} sur la figure 3.12). Cette solution peut donc être transposée à l'additionneur redondant signé de type 4/2. La figure 3.13 présente l'adaptation à effectuer pour cette seconde structure. On retrouve l'inversion de la retenue intermédiaire et de la sortie (S_{n-1} sur la figure 3.13).
Les changements apportés aux deux architectures non signées, n'induisent pas d'altération sensible des performances (le délai augmente du temps de traversé d'un inverseur et le matériel

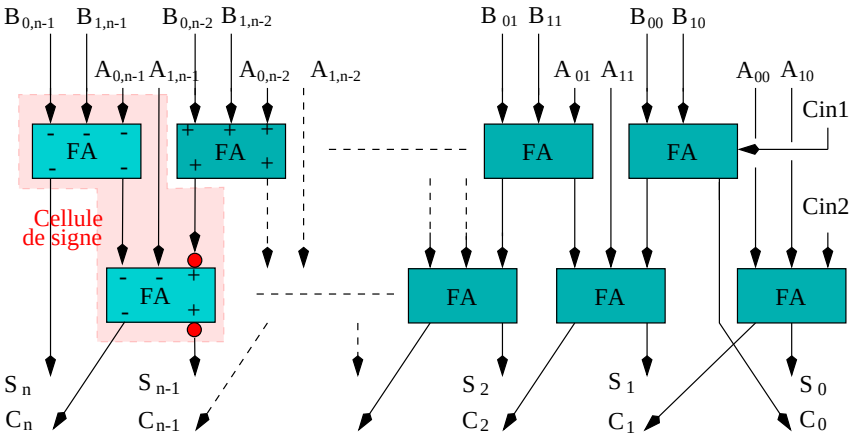


Figure 3.12 – Addition redondante signée de type 3/2 (Full-Adder)

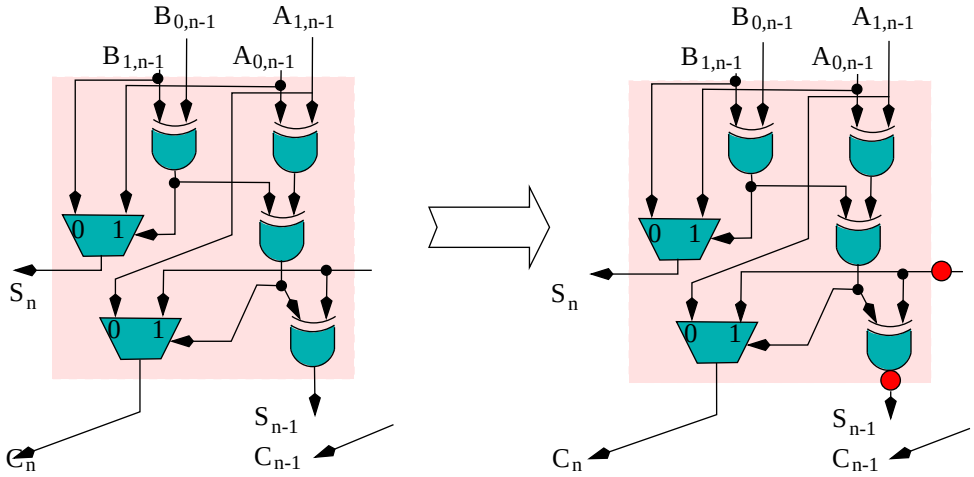


Figure 3.13 – Addition redondante signée 4/2 : modification pour la gestion de signe

croît de la surface de deux inverseurs). Les temps de propagation restent en temps constant et les surfaces en $O(N)$. Les rapports de performances entre architectures mixtes et redondantes sont conservés.

Remarque : La solution adoptée pour gérer les signes en notation classique (extension de signe de 1 chiffre) n'est pas exploitable ici. La gestion de différence de signe serait repoussée sur les nouveaux bits de poids forts.

Soustraction

Comme pour les additionneurs mixtes, la soustraction n'est pas triviale. Le problème vient du nombre de termes à soustraire. On distingue deux cas : les opérations ayant moins de trois termes à soustraire : $CS + BS$, $BS + BS$, $CS - CS$, $CS - BS$ et $BS - BS$, et l'opération

$BS - CS$ qui a trois termes à soustraire. Dans le premier cas, la solution est identique à celle employée en arithmétique classique : complément à deux des termes à soustraire. Le résultat est en Carry-Save. Dans le second cas, l'architecture $CS + CS \rightarrow CS$ ne disposant que de deux retenues entrantes, la technique du complément à deux n'est pas exploitable.
La solution permettant de prendre en compte les trois termes à soustraire, consiste à utiliser de nouveau l'équation (2.7), mais cette fois afin de modifier le comportement d'un additionneur $CS+CS \rightarrow CS$ signé. Du point de vue extérieure, les deux structures élémentaires non-signées (cellule 3/2 et 4/2 figure 3.11) composant cet opérateurs, sont décrites par l'équation :

$$w_i + x_i + y_i + z_i + a_i = 2.c_{i+1} + s_i + 2.a_{i+1} \tag{3.5}$$

où w_i, x_i, y_i et z_i sont les bits composant les deux chiffres de poids i, a_i est l'entrée retardée, a_{i+1} la retenue précoce, et c_{i+1} et s_i sont les deux sorties. Effectuons le complément de w_i et s_i, nous obtenons

$$\begin{aligned} (1.2^i - w_i) + x_i + y_i + z_i + a_i &= 2.c_{i+1} + (1.2^i - s_i) + 2.a_{i+1} \\ -w_i + x_i + y_i + z_i + a_i &= 2.c_{i+1} - s_i + 2.a_{i+1} \end{aligned} \tag{3.6}$$

puis inversons les deux termes de l'égalité

$$+w_i - x_i - y_i - z_i + a_i = -2.c_{i+1} + s_i - 2.a_{i+1} \tag{3.7}$$

La sortie est en notation BS.

L'équation des structures élémentaires signées 3/2 et 4/2 (figure 3.13) est

$$-w_{n-1} - x_{n-1} - y_{n-1} - z_{n-1} + a_{n-1} = -2.c_n + s_{n-1} - 2.a_n \tag{3.8}$$

où n est le poids des chiffres en entrée. La différence de signe des chiffres additionnés par les deux dernières cellules élémentaires $n-1$ et $n-2$ est déja pris en compte dans l'additionneur initial. En appliquant (2.7) à (3.8) sur w_i, a_i et a_{i+1} nous avons :

$$w_{n-1} - x_{n-1} - y_{n-1} - z_{n-1} + a_{n-1} = -2.c_n - s_{n-1} + 2.a_n \tag{3.9}$$

La sortie est en notation BS avec a_n égale au bit de signe BS^+ et c_n égale au bit de signe BS^-. La figure 3.14 présente les nouvelles cellules obtenues dans leur contexte d'utilisation. Les deux cellules de type 3/2 et de type 4/2 peuvent être utilisées pour construire le bloc $CS + CS$. Le résultat est en notation Borrow-Save. Les deux retenues entrantes possibles sont négatives. Cette architecture est utilisable quelle que soit la structure : 3/2 ou 4/2. En fait, les

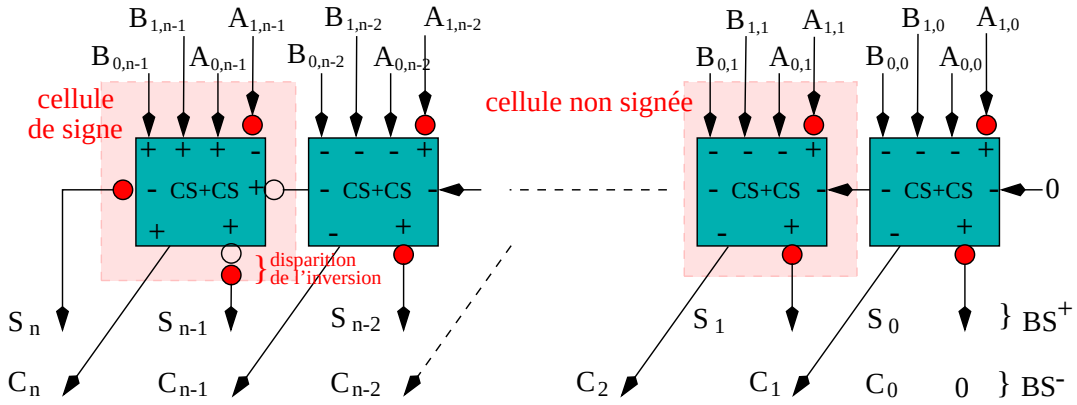


Figure 3.14 – Soustraction redondante

modifications apportées à l'additionneur redondant correspondent au complément à 1 d'une de la sortie S et d'une des entrées.
Comme pour les additionneurs mixte, l'architecture proposée peut s'obtenir directement au niveau vecteur. Pour cela il suffit de considérer toutes les E/S de l'additionneur de départ ($CS + CS \rightarrow CS$) comme négatives et de complémenter la sortie S et une entrée (ici W) :

$$\begin{aligned} -\overline{W} - X - Y - Z &= -2.C - \overline{S} \\ +(W-1) - X - Y - Z &= -2.C + (S-1) \\ +W - X - Y - Z &= -2.C + S \end{aligned} \tag{3.10}$$

Les propriétés des soustracteurs redondants sont équivalentes à celles des additionneurs redondants : calcul en temps constant, surface et consommation en $O(N)$.

3.3 Somme

La somme est une opération très présente en traitement du signal et de l'image. Bien que sémantiquement ce soit une addition, sa réalisation matérielle pour un nombre de termes à sommer Nb_s supérieur à 2, diffère fortement de l'addition classique où $Nb_s = 2$.

3.3.1 Algorithme

La somme a été très largement étudiée dans la littérature. Les livres [Mull97, Parh99] en font un bon résumé. Le principe consiste à réduire le nombre de termes à sommer à l'aide d'arbres d'additions. La technique employée repose sur l'exploitation des propriétés d'associativité et de commutativité de l'addition :

– L'associativité permet de diviser la somme en sous-sommes. Cette opération peut-être répétée plusieurs fois. La limite de division est atteinte quand une sous-somme définie un ad-

ditionneur élémentaire. Cet additionneur peut-être classique mixte ou redondant. La structure ainsi obtenue, décrit un arbre d'addition. Cet arbre peut-être totalement parallèle, semi-parallèle ou série, suivant la décomposition en sous-sommes effectuée. La quantité totale d'additions est conservée.

– La commutativité permet de redistribuer les termes. Mieux, cette redistribution peut-être effectuée directement au niveau chiffre ou bit. Le nouvel arbre obtenu ne tient compte que du poids des divers bits. Grâce à cette propriété, on s'affranchit complètement des différences de dynamique et de poids des LSB des termes à sommer. Une somme peut alors s'envisager comme une succession de colonne C_i de bits de même poids i.

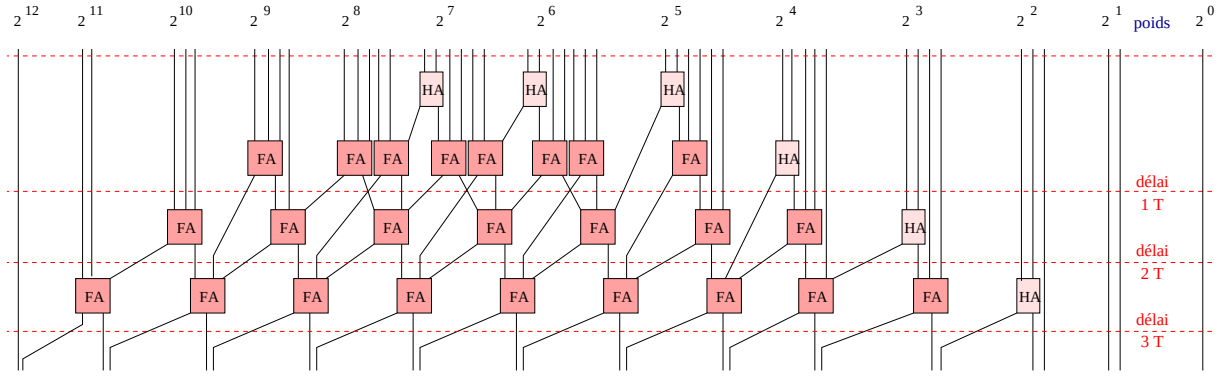
Il reste à trouver quel arbre utiliser. Les algorithmes les plus performants employés sont basés sur les travaux de C.S. Wallace [Wall64] et de L. Dadda [Dadd65, Dadd76]. L'additionneur élémentaire est le Full-Adder. L'arbre employé est totalement parallèle. La somme est effectuée en plusieurs passes définissant ainsi plusieurs paliers de réduction. Le problème qui se pose est de déterminer quels sont les paliers minimisant le temps de propagation et le nombre de Full-Adders utilisés.

Selon les travaux déjà cités, les paliers sont donnés par la suite géométrique : $U_n = U_{n-1} * 3/2$ avec $U_0 = 2$. Pour déterminer la succession des paliers d'une somme donnée, il suffit d'itérer cette suite jusqu'à obtenir la première valeur majorant le nombre de bits (M) de la colonne la plus haute. Le nombre de couche de FA est lui de : $Nb_c = log_{3/2}(M)$ soit $1.7095 * log_2(M)$.

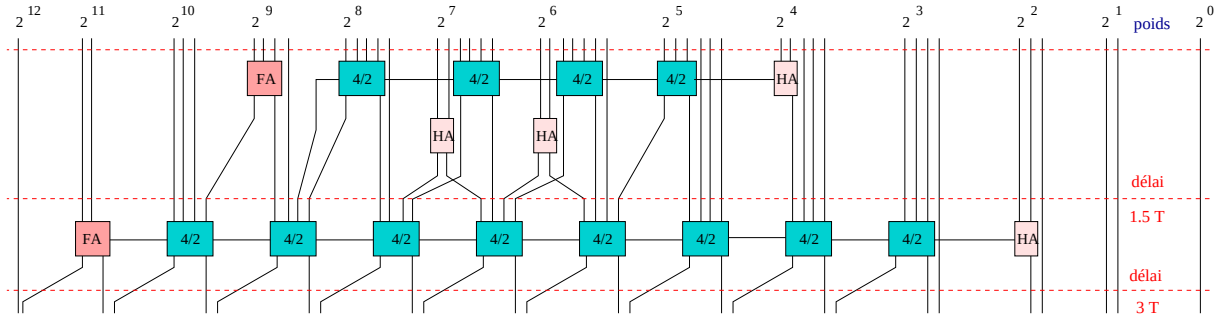
Une somme est donc une succession de couches de Full-Adders. Or, dans la sous-section 3.2.4, nous avons vu que deux Full-Adders successifs peuvent être remplacés par une cellule plus performante (*structure de type 4/2* figure 3.11). En considérant cette nouvelle cellule comme additionneur élémentaire, la nouvelle règle déterminant les paliers devient alors : $U_n = 2 * U_{n-1}$ avec $U_0 = 2$. Le nombre de couches est lui de : $Nb_c = log_2(M)$, avec M taille de la plus haute colonne de bits.

La figure 3.15 reprend l'exemple de [GuyoX1] d'une même somme réalisée avec les deux types d'additionneurs élémentaires, qui sont respectivement dans le cas (a) des Full-Adders et dans le cas (b) des cellules en *structure 4/2*. Sachant qu'une cellule de type 4/2 est composée du même matériel que deux Full-Adders et que son temps de propagation est de 1,5 fois celui d'un Full-Adder, alors, les deux architectures de la figure 3.15 ont la même surface et le même délai. Dans le cas 3.15.a, les temps de propagations des HA et des FA, dans le premier étage, ne s'additionnent pas car une des trois entrées des FA accepte un retard équivalent au temps de propagation d'un HA.

Les algorithmes utilisés donnent un résultat en notation Carry-Save. Pour obtenir le résultat en notation classique, il suffit juste d'utiliser une conversion $R \rightarrow NR$, soit, un additionneur à



(a) Full-Adder



(b) Structure 4/2

Figure 3.15 – Somme à partir de la matrice de produits partiels d'un multiplieur 7x7

deux termes.

Quelle que soit la somme réalisée, le nombre de couches est l'expression du temps de propagation. Le Full-Adder servant de référence, nous utiliserons $O(log_{3/2}(M))$ pour exprimer le délai. Donner une équation générale pour la surface est quasi impossible ; le nombre de termes à sommer, et surtout leur dynamique et le poids de leur LSB apporte trop de variabilité. Une évaluation simple consiste à considérer la quantité totale de bits à sommer. Cette valeur peut aussi servir pour estimer la consommation.

Remarque : Travailler au niveau des bits rend l'algorithme indépendant des notations des termes à sommer. C'est d'autant plus vrai qu'une fois mis en ligne, les additionneurs élémentaires correspondent aux additionneurs non signés mixtes et redondants des sous-sections 3.2.3 et 3.2.4.

3.3.2 Sommes signées

Dans de nombreux cas les sommes comportent des termes signés. Généralement ces termes n'ont pas tous la même taille. Il se pose alors le problème de la gestion des bits de signes.

L'extension directe de ces bits étant trop coûteuse en matériel, l'idée consiste à généraliser la méthode d'anticipation de l'extension de signe proposée par Fadavi-Ardekani [Fada93]. Considérons un bit de signe s_i de poids i. Ce bit est négatif et représente la valeur $v_i = -1 * 2^i$. En reprenant l'équation 2.7, on peut écrire :

$$-s_i = \overline{s_i} - 1 \ . \tag{3.11}$$

Dans ce cas l'expression de v_i peut être modifiée, tel que

$$v_i = (\overline{s_i} - 1) * 2^i = \overline{s_i} * 2^i - 1 * 2^i \ , \tag{3.12}$$

ce qui correspond à la décomposition de $-v_i$ en deux termes à additionner : une constante et le complément de s_i. En appliquant ce traitement à l'ensemble des bits de signe contenus dans une somme de termes donnés, on obtient une nouvelle somme composée des termes initiaux (dont le bit de signe a été complémenté), à laquelle s'ajoute les constantes introduites par la transformation. En profitant de la nature invariante des constantes, leur somme peut-être précalculée et ramenée à un seul terme que nous qualifierons de constante de signe. La prise en compte de termes signés ne modifie que très peu les algorithmes utilisés.
Du point de vue architectural, la méthode d'anticipation de l'extension de signe présentée, se traduit par l'ajout de la constante de signe à la somme initiale, plus l'inversion de tous les bits de signes, avant d'effectuer l'addition totale.
Cette technique n'implique pas que tous les termes à additionner soient signés. De plus, elle est exploitable quelle que soit la notation des termes à ajouter. Un simple complément à deux suffit pour prendre en compte les termes à soustraire (soustraction ou composante négative de la notation BS).

3.3.3 Sommes classique, mixte et redondante

Nous avons vu dans la section 2.2 que les notations CS et BS équivalent, en arithmétique classique, respectivement à une addition et une soustraction non encore effectuées. Dans ce cas, sommer des redondants n'implique que l'augmentation du nombre de termes à additionner ; les algorithmes à utiliser sont les mêmes. Dans ce contexte, le passage de toutes les entrées en notation redondante d'une somme classique (même nombres d'entrées sur les même dynamiques) correspond à additionner deux fois plus de termes.
Concernant le délai, obtenir le résultat demande une couche de réduction $4 \rightarrow 2$ supplémentaire. La surface est, elle, multipliée par plus de deux : deux arbres identiques en parallèles (deux fois plus de termes) plus l'étage de réduction supplémentaire. La progression de la surface à la même complexité que celle des additionneurs *sklansky* : $O(N * log(N))$ (sous-section

3.2.2), mais ici N correspond à la quantité de bit de la plus haute colonne. La consommation suit la surface. Ces résultats ne sont valables que pour la partie *réducteur de Wallace*. Concernant les sommes classiques et redondantes dans leur ensemble, il faut en plus tenir compte de la présence ou de l'absence de conversion $R \rightarrow NR$.
Déterminer les propriétés des sommes mixtes est plus compliqué, car le nombre de termes en entrée est variable et il existe donc, une multitude de répartitions des entrées entre notations NR et R (sans parler en plus des différences de dynamiques entre entrées). Concernant la partie *réducteur de Wallace*, les performances des sommes mixtes sont bornées par les sommes classiques (borne inférieure) et par les sommes redondantes (borne supérieure).

3.3.4 Remarques et bilan

Sommer plusieurs termes avec les meilleures performances possibles, se résume à construire un arbre d'additions le plus parallèle possible. Pour être le plus optimal, cet arbre doit effectuer les additions au niveau le plus élémentaire : bit, sans tenir compte de l'aspect vecteur des termes ou des chiffres à sommer. Travailler au niveau le plus bas implique, qu'avec le même algorithme, la mixité de notation, de signe et d'opération (addition/soustraction) ne pose aucun problème. Il suffit juste de bien prendre en compte les bits de signes et d'utiliser des compléments à deux si nécessaire (BS et soustraction).

Remarques :
- Pour une somme donnée et quel que soit le degré de parallélisme de l'arbre, la surface est très sensiblement la même. Le délai, lui, est proportionnel au degré de parallélisme de l'arbre ; plus c edernier est faible et plus le temps de propagation est important. La consommation est *a priori* elle aussi plus importante, car pour sensiblement le même matériel la diminution du degré de parallélisme se traduit par l'augmentation du nombre de couches combinatoires impliquant plus de commutations parasites (*glitches*).
- La sortie du *réducteur de Wallace* est en Carry-Save. Exprimer la même somme en notation classique demande juste de rajouter une conversion $CS \rightarrow NR$.
- Les additionneurs mixtes et redondants sont des sommes. L'algorithme utilisé diffère uniquement dans la gestion de signe : prise en compte directe pour les additionneurs et gestion par constante pour les sommes.
- Le choix entre les algorithmes de réduction 3/2 et 4/2, dépend de la technologie visée. Comme nous le verrons dans la section 4.2, ce choix est aussi fonction de la disponibilité de blocs optimisés correspondant aux cellules élémentaires FA et réducteur $4 \rightarrow 2$ (existence des cellules précaractérisés,...).

3.4 Multiplication Entière

La multiplication est la seconde opération essentielle aux traitements du signal et de l'image. Elle est couramment employée dans des applications où les produits et les sommes de produits sont très fréquents : le filtrage numérique, la transformée de Fourrier ou la transformée cosinus discrète par exemple. Cette opération ajoutée à un mécanisme d'accumulation, est le cœur des DSP (Digital Signal Processor).
Deux techniques de réalisation ont retenu notre attention : l'algorithme que nous appelons *Direct* où à chaque produit partiel ($chiffre \cdot chiffre$) correspond **1 bit**, et l'algorithme de *Booth*. Nous appellerons multiplieur les architectures associées à cette opération.

Modèle générique de multiplieur

La multiplication est décomposable en deux étapes fondamentales : le calcul des produits partiels et la sommation de ceux-ci. En $VLSI$, nous avons choisi de réaliser la multiplication en quatre blocs distincts (figure 3.16) : le recodage des entrées qui permet d'en modifier les propriétés, la réalisation des produits partiels, la sommation des produits partiels et la conversion de notation.
Les deux premiers blocs effectuent le calcul des produits partiels (première étape), et les deux suivants leur somme (deuxième étape). Le premier et le quatrième blocs n'ont pas forcément d'existence. Ils dépendent de l'algorithme employé et des notations utilisées en entrée / sortie. Par exemple, la prise en compte d'une entrée redondante entraîne des modifications dans le calcul des produits partiels et, seule une sortie en notation NR nécessite une conversion.
Les deux étapes peuvent être considérées comme indépendantes. C'est vrai au sens où les algorithmes employés dans chaque partie sont indépendants. Le seul lien les raccordant est la quantité de produits partiels à sommer. Aussi, dans la présentation qui suit, l'étude des multiplieurs

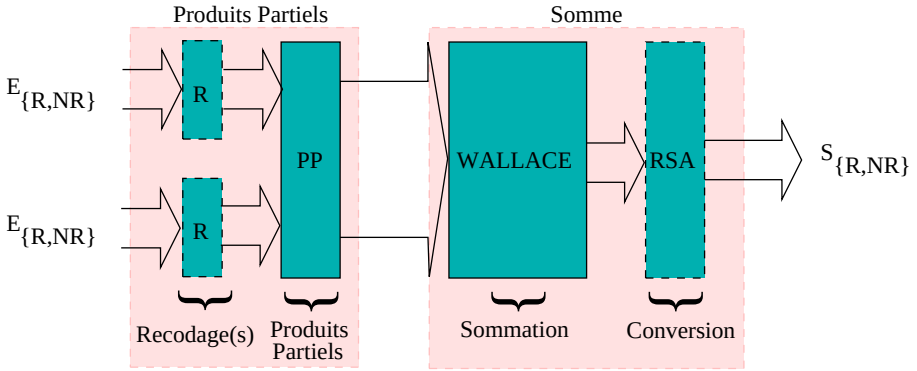


Figure 3.16 – Multiplieur générique

est volontairement limitée à la réalisation des produits partiels et des éventuels recodages, les deux blocs *somme* et *conversion* venant d'être étudiés dans les sections 3.2 et 3.3.
Sur le plan des performances, la différenciation de deux sommes vient uniquement du nombre de produits partiels à sommer, l'algorithme employé étant forcement le même. En d'autres termes, la qualité d'un multiplieur est étroitement liée au nombre de produits partiels à additionner et par conséquent à la technique de réalisation employée pour les obtenir.

Remarque sur les architectures des multiplieurs redondants et mixtes

Pour que l'arithmétique mixte soit viable, il est nécessaire que les architectures développées soient au minimum aussi performantes qu'en notation classique. L'importance des multiplieurs en traitement du signal fait qu'ils ne peuvent pas échapper à cette règle.
Une rapide analyse des structures existantes : Wallace, Booth, Braun, etc. [Mull97, Parh99], montre que le temps de propagation et la surface sont directement liés au nombre de produits partiels générés, l'élément le plus coûteux selon ces deux critères, étant la somme. Ce constat est d'autant plus vrai en redondant et en mixte, qu'il n'y a pas de conversion en sortie. Le second bloc le plus coûteux est la matrice des produits partiels. Aussi, btenir des architectures mixtes et redondantes performantes en sachant que les sommes des divers multiplieurs sont réalisées avec le même algorithme, impose de mettre l'accent à la fois sur la limitation du nombre de produits partiels générés et sur la limitation de la taille des éléments de la matrice.
Les diverses architectures proposées sont en notation signée. La différence avec les architectures non-signées (quand elles sont possibles) tient exclusivement au complément des divers bits de signes de chaque architecture et à l'ajout de la constante de signe directement dans la somme : sous-section 3.3.2.

3.4.1 Multiplication entière non redondante

Algorithme *direct*

En arithmétique classique, cet algorithme est la transposition de la multiplication que nous effectuons à la main. Plusieurs architectures ont été développées : multiplieurs de Braun et de Wallace pour les plus connus. La grande différence entre ces diverses réalisations réside essentiellement dans le parallélisme de l'arbre d'addition choisi pour le bloc de somme. La structure de la matrice des produits partiels, elle, est toujours la même. Le produit élémentaire est calculé au niveau bit. Il correspond à une simple porte ET. La figure 3.17 présente l'architecture réalisant un produit ligne $A_{NR} \cdot B_{NR,i \neq N-1}$ de la multiplication $A_{NR} \cdot B_{NR}$; A et B sont respectivement sur M et N bits. Pour $i = N - 1$ il faut inverser tous les produits partiels. Le nombre

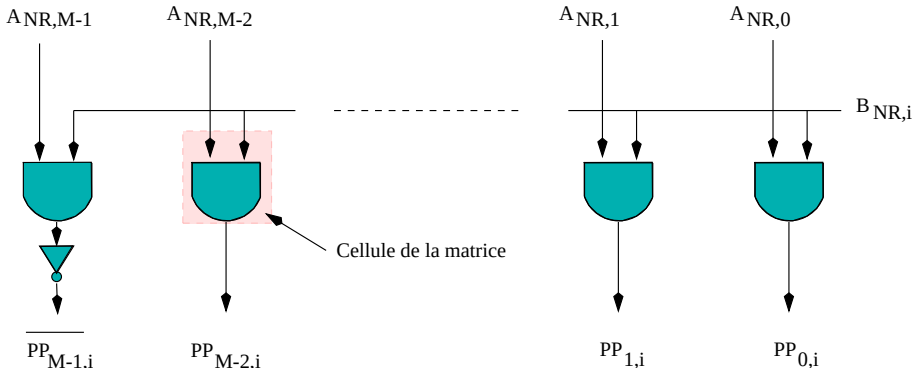


Figure 3.17 – Multiplieur *Direct* : produits partiels $A_{NR} \cdot B_{NR,i \neq N-1}$ signés

de produits partiels à additionner est de $M \cdot N$, auquel il faudra ajouter les bits nécessaires au codage de la constante de signe (sous-section 3.3.2). La quantité de ces derniers est fonction des valeurs de M et de N. Elle sera toujours inférieure à $max(M, N)$; pour $M = N$, un seul bit à 1 est nécessaire. Cette architecture permet d'obtenir toutes les combinaisons d'entrées non-signée/signée.

Algorithme de *Booth*

Cet algorithme est basé sur la décomposition de la multiplication $A_{NR} \cdot B_{NR}$ en produit chiffre par ligne $A_{NR} \cdot B_{NR,i}$ sous la forme $A_{NR} \cdot B_{NR,i} = \sum_{j=0}^{N-1} a_j \cdot (b_{i-1} - b_i) \cdot 2^j$. L'opération à effectuer sur chaque ligne de produits partiels, est l'addition ou la soustraction, en fonction de la valeur de deux bits consécutifs de B tableau 3.2.(a). L'algorithme réellement utilisé : *Booth modifié*, est une extension de ce premier en base quatre. Les bits sont regroupés par trois : tableau 3.2.(b). Avec cette variante de Booth, le nombre de produits partiels est globalement divisé par deux. Il est à noter qu'il existe plusieurs possibilités de correspondances entre le *code* et les trois

B_i	B_{i-1}	Opérations
0	0	addition de 0
0	1	addition de A
1	0	soustraction de A (CA2)
1	1	addition de 0

(a) Algorithme de Booth

B_{i+1}	B_i	B_{i-1}	code	$\overline{X}$	$X1$	$X2$	$+1$
0	0	0	0	0	0	0	0
0	0	1	+1	0	1	0	0
0	1	0	+1	0	1	0	0
0	1	1	+2	0	0	1	0
1	0	0	-2	1	0	1	1
1	0	1	-1	1	1	0	1
1	1	0	-1	1	1	0	1
1	1	1	0	1	0	0	0

(b) Algorithme de Booth Modifié

TAB. 3.2 – Algorithmes de Booth

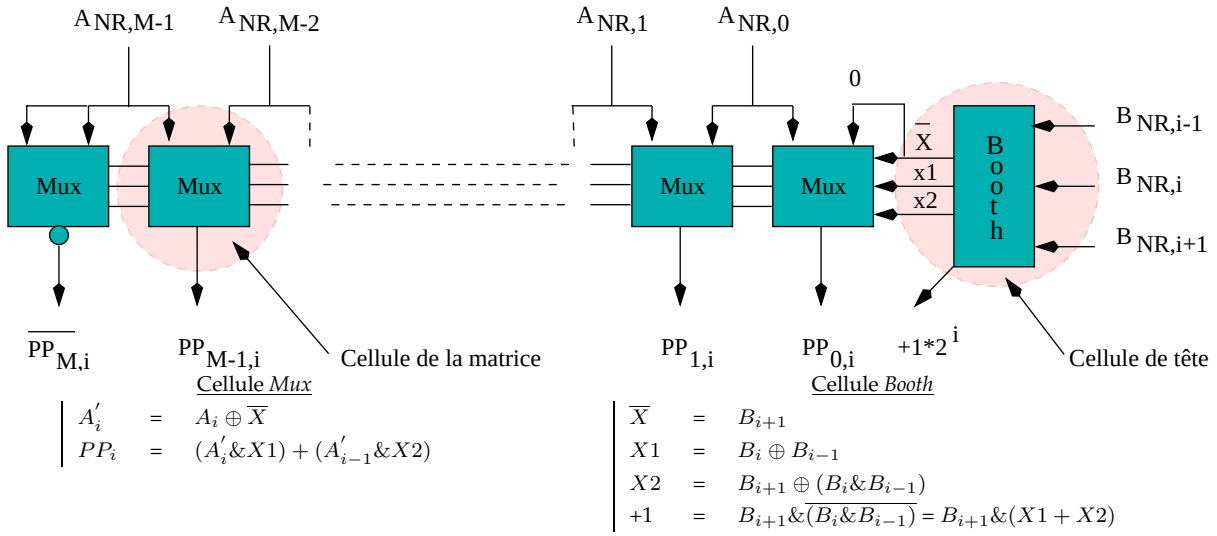


Figure 3.18 – Multiplieur de *Booth* : produits partiels $A_{NR} \cdot B_{NR,i}$ signés

signaux $\overline{X}$, $X1$ et $X2$. La solution proposée ici, est légèrement différente de celle communément présentée. Elle tient compte des priorités des signaux pour simplifier les équations booléennes : l'effet du complément est nul si $X1$ et $X2$ sont nuls par exemple.

L'architecture correspondante est présentée figure 3.18. Le calcul des produits partiels est divisé en deux parties : des cellules de têtes *Booth*, qui recodent l'entrée $B_{NR,i}$ et des cellules *Mux*, qui effectuent réellement le calcul des produits partiels de la ligne $A_{NR} \cdot B_{NR,i}$. Les équations booléennes détaillant chaque cellule sont données avec la figure. La solution choisie vise à réduire le plus possible la matrice. Il est à noter que l'ensemble *deux ET suivis d'un OU* (PP_i) existe sous forme d'une porte précaractérisée dans les bibliothèques de cellules.

Pour une multiplication $A_{NR} \cdot B_{NR}$ où A et B sont respectivement sur M et N bits, le nombre de produits partiels à additionner est de $(M+2) \cdot ((N+1)/2)$, auquel va s'ajouter $1+(N+1)/2$ bits à 1, nécessaires au codage de la constante de signe. Le 2 ajouté à M correspond à l'extension de 1 bit introduite par le *fois* 2 ($X2$) et au $+1$ nécessaire au *fois* -1 ($\overline{X} + 1$). Le 1 ajouter à N permet de prendre en compte à la fois les valeurs paires et impaires de N. Les divisions par 2 sont entières.

L'utilisation d'un codage différentiel pour recoder une des opérandes, implique que ce multiplieur est forcement signé.

3.4.2 Multiplication entière redondante

L'architecture du multiplieur redondant développée est de type *direct*. Elle reprend le modèle initial en quatre blocs. Les entrées/sorties sont en notations redondantes. Dans un premier temps, nous nous intéresserons uniquement à la notation Carry-Save.

Matrice des Produits Partiels

Soit la multiplication redondante $A_{CS} \cdot B_{CS}$, pour prendre en compte les $A_{i,CS} = 2$ et les $B_{j,CS} = 2$ dans le calcul des produits partiels, l'usage de la technique de décalage (multiplication par deux) a été généralisé aux lignes et aux colonnes. Ainsi chaque intersection i,j génère trois sous produits. Le premier correspond à la multiplication sans décalage $(A_{i,CS} = 1)ET(B_{j,CS} = 1)$, le second à un décalage selon i $(A_{i,CS} = 2)ET(B_{j,CS} \neq 0)$ et le troisième à un décalage selon j $(A_{i,CS} \neq 0)ET(B_{j,CS} = 2)$. Leur somme est égale au produit de l'intersection : $PP_{i,j} \in \{0, 1, 2, 4\}$.

Pour réduire le nombre de termes à sommer, l'ensemble des sous produits est combiné par trois en fonction de leur *décalage*. Le nombre de sous produits est alors de 1 par intersection i, j. Cette technique pose un problème à l'intersection des lignes et des colonnes. Comme les produits partiels sont codés sur un bit et que les deux sous-produits décalés ne sont pas exclusifs, le résultat est faux lorsqu'un chiffre égal à 2 est présent à la fois dans les deux opérandes. Pour éviter ce cas, il est nécessaire de modifier les propriétés des entrées. Si les deux entrées sont recodées avec la propriété : si $E_{i,CS} = 2$ alors $E_{i-1,CS} = 0$, alors les trois sous-produits sont nuls à l'intersection de deux décalages. En réalité, un seul sous-produit décalé doit être nul et un seul recodage est impératif. Le détail de la technique de recodage est donné ci-après.

L'architecture d'un produit partiel général $PP_{i,j}$ est donnée figure 3.19. Pour limiter la taille des éléments de la matrice, la structure est décomposée en deux blocs. Les deux premiers regroupent les parties communes à chaque ligne ou à chaque colonne. Chacune génère un signal $E_{i,CS} \geq 1$ et deux signaux exclusifs : $E_{i,CS} = 1$ et $E_{i,CS} = 2$. Le second bloc effectue le calcul des sous produits (portes ET) et leur recombinaison en fonction des décalages (porte OU à 3 entrées). L'utilisation de têtes de lignes et de têtes de colonnes, permet de réduire sensiblement le matériel nécessaire à l'élaboration des éléments de la matrice. Lors de la réalisation, on groupera le calcul des sous produits et la recombinaison, en une seule porte : (a ET b) OU (c ET d) OU (e ET f). Ainsi, on réduit encore plus la surface et le délai de la matrice.

Avec A_{CS} sur $M - 1$ chiffres et B_{CS} sur $N - 1$ chiffres (-1 car notation redondante), le total des bits à sommer est de $M \cdot N + M + N - 1$, auquel il faut ajouter ceux nécessaires au codage de la constante de signe. La quantité de ces derniers est fonction des valeurs de M et de N. Elle sera toujours inférieure à $max(M, N)$. Le total de bits à sommer est légèrement supérieur à celui d'un multiplieur de forme *Directe* en arithmétique conventionnelle ($M \cdot N$+constante), pour une dynamique d'entrée équivalente. Cette différence est due à l'extension d'un chiffre de la dynamique introduite par les recodages. La contribution $M + N$ peut-être diminuée en jouant sur les valeurs limitées à l'intervalle {-1,0,1}, prises par les chiffres MSBs et LSBs du nombre recodé. Cette architecture permet d'obtenir toutes les combinaisons d'entrées non-signée/signée.

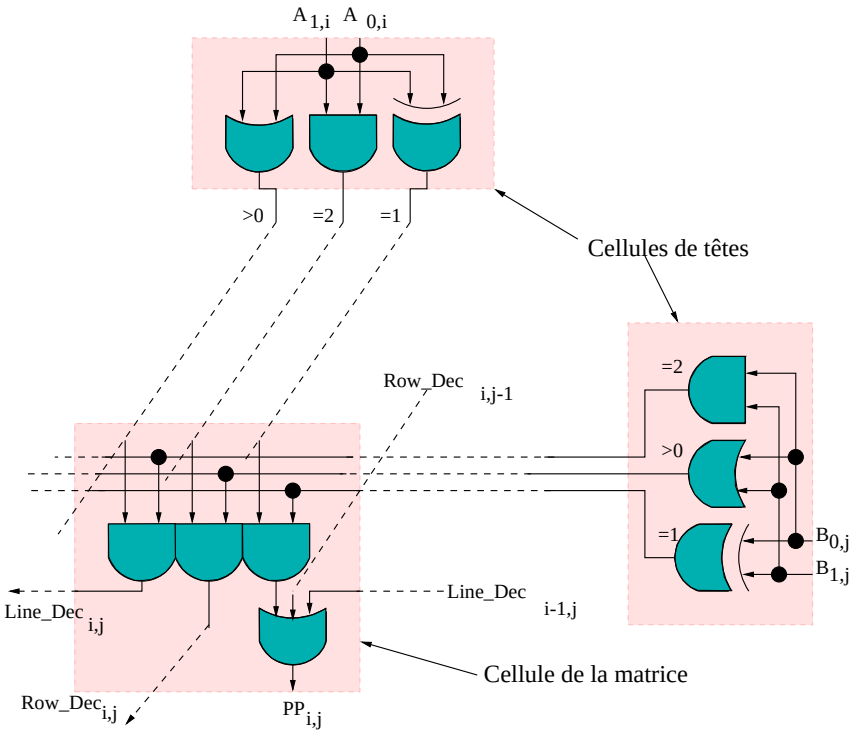


Figure 3.19 – Multiplication redondante : Architecture associée aux produits partiels

Recodage

Dans leurs rapports de recherche [Daum97, Daum00b], M. Daumas et D. Matula proposent une technique de recodage par retenue des nombres redondants. Nous l'avons déjà utilisée pour les conversions $R \rightarrow R$: sous-section 2.3.2, et étendue aux soustractions mixtes et redondantes : sous-sections 3.2.3 et 3.2.4.
Pour mémoire, le recodage est basé sur l'addition indépendante de plusieurs bits (deux ou trois) composant les chiffres de plusieurs termes à sommer. On obtient ainsi un bit de somme et un bit de retenue par chiffre. Suivant les besoins, les entrées sont complémentées ou non.
Cette fois l'objectif qui nous intéresse est de modifier les propriétés du nombre de départ sans altérer sa valeur. La notation de sortie doit être en CS pour permettre d'exploiter la technique de décalage. La cellule d'addition élémentaire utilisée est le Half-Adder.

Prenons un redondant E_{CS} en notation Carry-Save. Chacun de ses chiffres est tel que $E_{i,CS} \in \{0, 1, 2\}$. L'ensemble définissant la somme de deux chiffres consécutifs est $\{0, 1, 2, 3, 4, 5, 6\}$. Le recodage d'un chiffre de l'entrée, par addition des deux bits le composant, implique au minimum qu'un des deux bits résultant (la somme ou la retenue) est nul. Ainsi dans le CS de sortie, au moins la moitié des bits valent '0'. La retenue et la somme liées à 1 même chiffre, n'ayant pas

le même poids, l'intervalle de définition des chiffres du CS de sortie reste $\{0, 1, 2\}$. Par contre, l'ensemble définissant la somme de deux chiffres consécutifs est maintenant $\{0, 1, 2, 3, 4, 5\}$. La modification du voisinage se traduit par l'impossibilité de voir deux chiffres consécutifs prendre la valeur 2.

Point de vue pratique :

Considérons un redondant E_{CS} en notation Carry-Save ou $E_{i,CS} \in \{0, 1, 2\}$, et considérons les 2 étages de recodage que nous utiliserons pour le multiplieur. Le premier ($R1$) assure que deux chiffres consécutifs, du nombre recodé, ne peuvent pas prendre la valeur 2 : si $R1_{i,CS} = 2$ alors $R1_{i-1,CS} \neq 2$. Le second ($R2$) implique en plus que si $R2_{i,CS} = 2$ alors $R2_{i-1,CS} = 0$. L'ensemble défini par la somme de deux chiffres consécutifs se réduit à $\{0, 1, 2, 3, 4\}$.
La présence systématique de 0 à côté des 2, va permettre de recombiner tous les produits issus d'une multiplication par un 2 avec le produit nul qui le jouxte. Comme nous l'avons décrit plus haut, cette seconde propriété permet d'atteindre l'objectif de limitation du nombre de produits partiels à 1 par produit élémentaire $A_{CS,i} \cdot B_{CS,j}$.

La figure 3.20 présente l'architecture du codage utilisé dans le multiplieur.

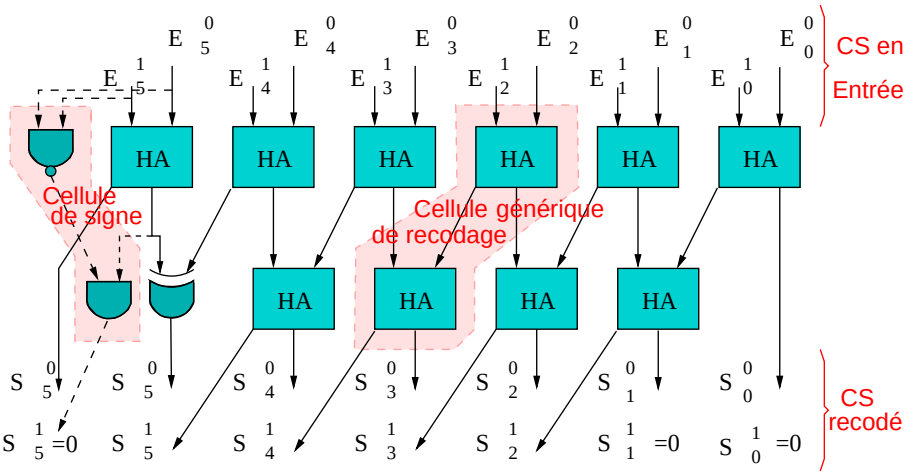


Figure 3.20 – Multiplication redondante : recodage signé en deux étages

Chaque étage correspond à une ligne de *Half-Adder* et réalise une addition de deux termes sans propagation des retenues. Afin de limiter l'extension de la dynamique et de gérer les signes, la régularité n'est pas respectée pour les bits de poids forts. Pour un signal E_{CS} sur N chiffres, la sortie recodée S_{CS}, est sur $N + 1$ chiffres, d'où la légère croissance de la matrice de produits partiels.

Réduction du coût de la matrice :

Dans l'objectif de réduire le matériel nécessaire à la réalisation de la matrice de produits partiels, nous proposons ici une modification de l'architecture précédente figure 3.20. La nouvelle structure, figure 3.21, est aussi composée de deux étages. Le premier assure que 2 chiffres consécutifs ne peuvent prendre la valeur 2, mais en plus assure qu'un chiffre sur deux n'est composé que d'un seul bit. Comme précédemment, le second étage assure que quand le chiffre i est égal à 2, le chiffre $i-1$ est égal à 0.

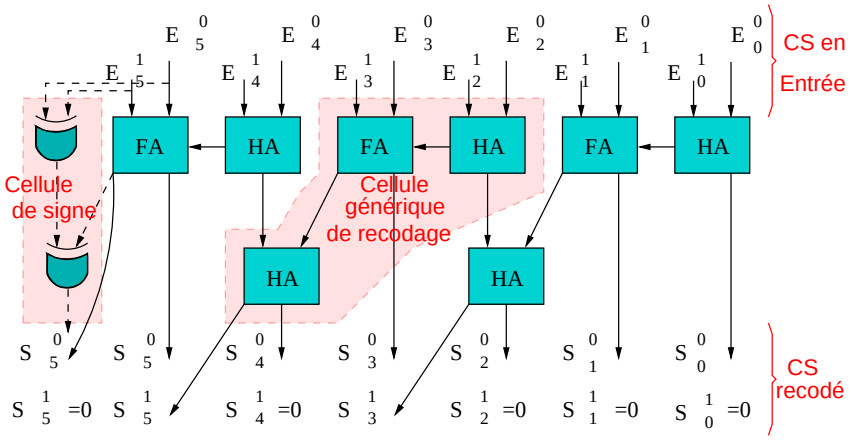


Figure 3.21 – Modification de l'étage de recodage : exemple avec un CS sur 6 chiffres

La nouvelle propriété introduite : un chiffre sur deux défini par un bit unique, implique que de nombreux sous-produits vont être nuls. En conséquence, dans de nombreux cas la quantité de sous-produits combinés par les cellules élémentaires de la matrice (figure 3.19), vont passer de 3 à 2, voir 1, introduisant une forte diminution du matériel nécessaire à la matrice.
La meilleure réduction est obtenue quand les deux entrées sont recodées. Dans ce contexte, une cellule de la matrice sur deux voit son nombre de sous-produits réduit à 2 et une cellule sur quatre voit sa quantité de sous-produits réduite à 1. De même, une cellule d'en-tête sur deux est réduite à la distribution du bit non nul du chiffre à son entrée sur $X1$ et ≥ 0 (figure 3.19).
Les bénéfices apportés par ce nouveau recodage ont pour contre partie une très légère augmentation du temps de propagation du recodage.

Ajustement de la dynamique

L'augmentation de la dynamique des entrées (due au recodage) accroît la taille de la matrice des produits partiels. Aussi la sommation donne aussi un résultat sur une dynamique trop grande. Supprimer les bits de poids fort en trop ne suffit pas à corriger la taille de la sortie. Tel quel,

on perdrait de temps en temps une retenue. L'idée est plutôt de réajuster les nouveaux bits de poids forts en fonction des anciens. La figure 3.22 présente la modification à apporter. Celle-ci

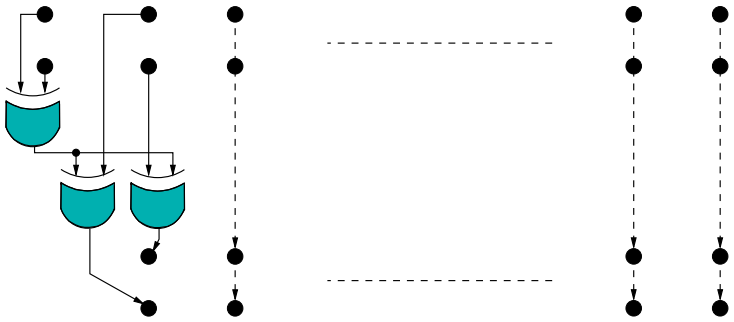


Figure 3.22 – Multiplication redondante : ajustement de la dynamique

dégrade légèrement le temps de propagation de ce multiplieur. Toutefois, plus la dynamique des entrées est importante, plus l'impact est réduit.

Multiplieur redondant avec une ou deux entrées BS

L'objectif souhaité est de proposer des multiplieurs autorisant toutes les combinaisons CS/BS sur leurs entrées. Les modifications à apporter au multiplieur $CS \cdot CS$, sont minimes et ne concernent que le recodage.
L'idée est simple : On se sert du premier étage du recodage pour effectuer un complément à 2 du terme négatif du BS. Le complément à 1 est obtenu en inversant tous les E_i^-. Le *plus 1* se fait en positionnant à 1 le bit $S_{1,0}$ (toujours nul) du Carry-Save de sortie : figure 3.23.

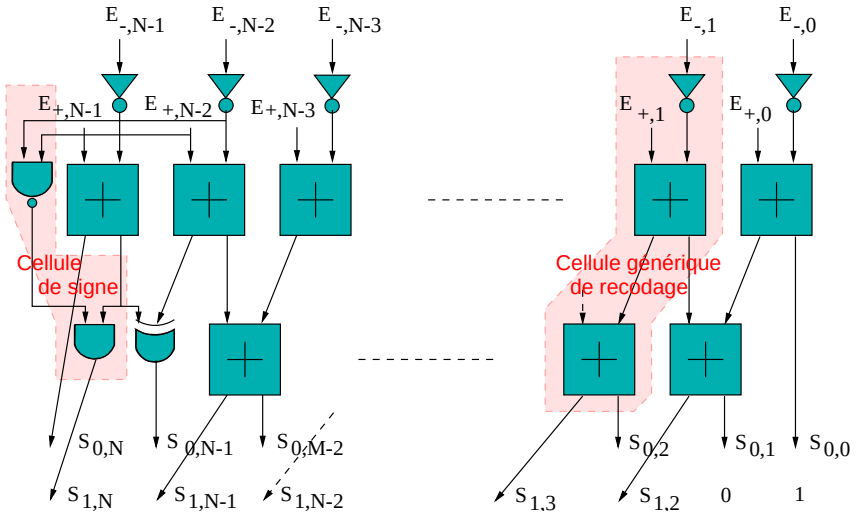


Figure 3.23 – Multiplication redondante : recodage d'une entrée BS

Dans le cas d'un multiplieur $BS \cdot BS$, la conversion du terme négatif du second BS est nécessaire. Deux solutions sont possibles : utiliser soit une conversion $BS \rightarrow CS$ soit un second recodage. Employer un recodage complet est plus intéressant. Certes la surface augmente, mais le délai reste identique et les divers chemins combinatoires sont équilibrés, ce qui permet de réduire la quantité de commutations parasites (*glich*) et donc la consommation.
D'ailleurs, pour l'ensemble des multiplieurs redondants, équilibrer les chemins avec un second recodage, même s'il n'est pas nécessaire, va permettre de limiter la consommation. Cette remarque est détaillée plus longuement dans le chapitre suivant.

Remarque

Les architectures proposées pour la multiplication $R \cdot R$ permettent de réaliser l'opération X^2 en notation CS et BS. Dans ce contexte, un seul recodage est nécessaire. Ce développement particulier est détaillé dans l'annexe B.

3.4.3 Multiplication entière mixte

Cette multiplication réalise le produit d'un nombre redondant avec un nombre non redondant. Deux possibilités : $CS \cdot NR$ et $BS \cdot NR$. Les deux algorithmes, *Direct* et *Booth*, sont exploitables pour réaliser ces multiplieurs. Le résultat est en notation CS.

Algorithme *Direct*

Les deux multiplications, $CS \cdot NR$ et $BS \cdot NR$, constituent des cas particuliers de la multiplication $R \cdot R$. Pour obtenir ces deux multiplieurs, la première idée consiste donc à partir de l'architecture du multiplieur $R \cdot R$ et à la dégrader.

Multiplieurs mixtes $CS \cdot NR$:

L'ensemble des réductions de matériel vient du fait qu'une seule entrée est redondante. Le nombre de décalage tombe à 1 et les intersections entre décalages disparaissent. Les recodages ne sont plus nécessaires. Parallèlement, le matériel utilisé pour chaque cellule et à chaque entête diminue. Le nombre de sous produits par intersection i,j passe de 3 à 2.
On peut voir sur la figure 3.24 que l'architecture du produit ligne $CS_j \cdot E_{NR}$, est fortement dégradée. Chaque élément de la matrice est réalisé avec une porte (a ET b) OU (c ET d) (zone colorée sur la figure) 3.24. Les entêtes sont des Half-Adders. Cette architecture permet d'obtenir toutes les combinaisons d'entrées non-signé/signée.

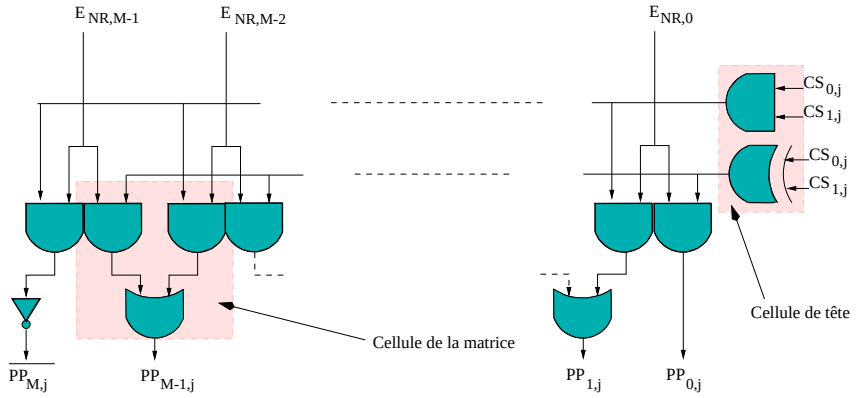


Figure 3.24 – Multiplication mixte *Directe* : produits partiels $CS_j \cdot E_{NR}$ signés

Multiplieurs mixtes $BS \cdot NR$:

Comme précédemment nous pourrions effectuer le complément à deux de la composante négative du Borrow-Save pour se ramener à un Carry-Save. Il existe toutefois une structure plus simple et plus performante. Cette architecture est aussi basée sur la décomposition des produits partiels en entêtes et cellules. Elle est présentée figure 3.25. Le principe est simple. Dans

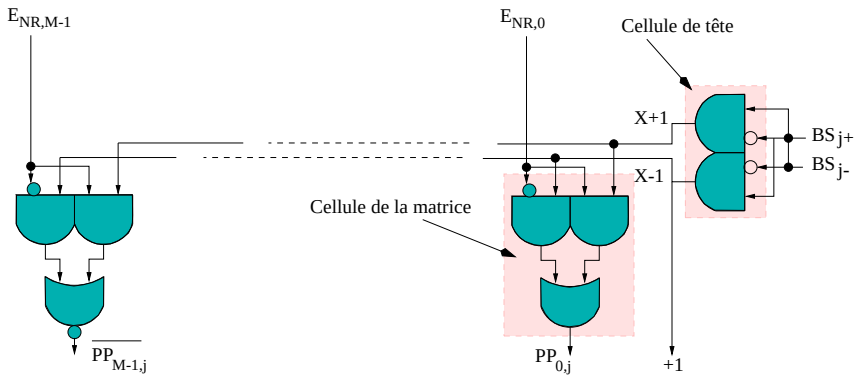


Figure 3.25 – Multiplication mixte *Directe* : produits partiels $BS_j \cdot E_{NR}$ signés

un premier temps, chaque BS_j est décomposé en deux signaux exclusifs : multiplication par 1 (BS_j^+ ET $\overline{BS_j^-}$) et multiplication par -1 ($\overline{BS_j^+}$ ET BS_j^-). Puis les produits partiels sont calculés (portes AND et OR). Pour la multiplication par -1, le complément à 2 est obtenu en inversant chaque NR_i et en rajoutant un signal +1 dans la somme générale. Il est intéressant de noter que chaque cellule correspond à un multiplexeur 2 vers 1 avec décodage.

Pour ces deux architectures et avec A_{NR} sur M chiffres et $B_{CS,BS}$ sur $N-1$ chiffres, le total des bits à sommer est de $M \cdot N$ auxquels s'ajoutent ceux nécessaires au codage de la constante

de signe. Dans le cas des $CS \cdot NR$, cette dernière est identique à celle des multiplieurs *Directs* classiques. Les quantités de bits à sommer sont les mêmes en classique et en mixte.
Remarque : Dégrader un peu plus l'architecture du $R \cdot R$ nous amènerait au multiplieur *Direct* classique : une matrice de portes ET, pas d'entête, et pas de recodage.

Algorithme de *Booth*

La première question qui se pose est de savoir à quelle entrée appliquer l'algorithme de *Booth*. Recoder l'opérande NR offre deux possibilités d'architectures : soit les valeurs recodées sont appliquées directement au redondant, soit on cherche à utiliser en plus la technique de décalage déjà éprouvée. Dans le premier cas, on aboutit à deux bits par produit partiel, ce qui réduit à néant l'avantage de l'algorithme de *Booth*. Dans le second cas, les décalages selon les colonnes et les lignes, introduisent des différences de signe entre les bits composant un produit ligne. Faire la différence entre +1 et -1 nécessite aussi deux bits. La solution de recoder l'entrée non redondante n'apporte rien. Le codage est donc appliqué à l'opérande redondante.
Recoder un nombre classique avec l'algorithme de *Booth*, correspond à découper ce nombre en plusieurs paquets d'une même quantité de bits consécutifs, et à effectuer un codage différentiel entre ces divers paquets. Dans le cas de la multiplication classique de deux variables, un paquet comprend 3 bits ce qui équivaut à la base 4. L'ensemble des valeurs possibles défini pour un paquet, est $\{-2, -1, 0, +1, +2\}$. Les bases supérieures ne sont pas utilisées car le coût matériel serait trop élevé.
Le découpage par paquet de deux chiffres d'un CS et d'un BS, donne respectivement les ensembles $\{0, +1, +2, +3, +4, +5, +6\}$ et $\{-3, -2, -1, 0, +1, +2, +3\}$. Dans les deux cas, le nombre de valeurs possibles est supérieur à celui de l'ensemble $\{-2, -1, 0, +1, +2\}$. De plus, les valeurs prises pour le CS ne sont pas centrées sur zéro. La modification des propriétés de l'opérande redondante est donc nécessaire. Une fois de plus nous allons utiliser le *recodage par retenue*.

Cette approche a déjà été utilisée par M. Daumas et D. Matula, dans [Daum00a]. Les développements architecturaux présentés ci-dessous diffèrent sur plusieurs points, mais l'esprit reste le même. L'objectif est de réutiliser le multiplieur de Booth classique proposé dans la sous-section 3.4.1. Dans un premier temps, nous nous intéresserons uniquement à la notation Carry-Save.

Multiplieurs mixtes $CS * NR$:

Le recodage $CS \rightarrow CS$ utilisé dans la sous-section 3.4.2, réduit l'ensemble $\{0, 1, 2, 3, 4, 5, 6\}$, défini par la somme de deux chiffres consécutifs, à $\{0, 1, 2, 3, 4\}$. Pour centrer cet ensemble sur

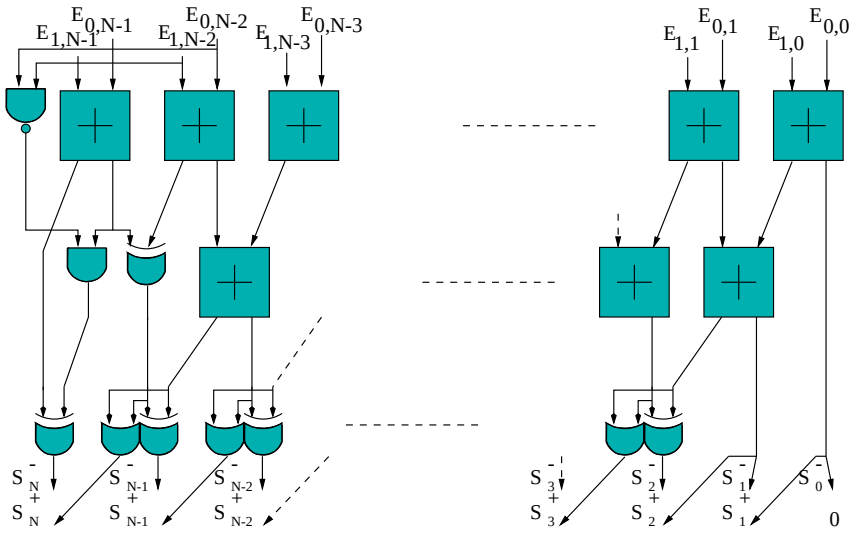


Figure 3.26 – Multiplication de *Booth* mixte : étage de recodage supplémentaire

zéro, une simple conversion $CS \rightarrow BS$ suffit. La figure 3.26 présente la modification de l'architecture. Nous n'avons pas utilisé complètement la conversion $CS \rightarrow BS$, présentée section 2.3.2, afin de mieux maîtriser l'extension de dynamique introduite par le troisième étage du recodage.

Les propriétés en entrée étant assurées, il reste à effectuer la multiplication. L'idée est de réutiliser au maximum les développements effectués pour le multiplieur de *Booth* classique (figure 3.18). L'objectif est de garder la même cellule de matrice car elle est minimale. Par conséquent, les trois signaux : $\overline{X}$, $X1$ et $X2$, doivent avoir les mêmes propriétés. Deux choix sont possibles : soit adapter le BS de sortie à la cellule *Mux* existante, soit développer une nouvelle cellule *Mux*. Le nombre de couches combinatoires introduites par le recodage étant déjà important, nous avons opté pour la seconde possibilité. Le tableau 3.3 présente la table de vérité et les nouvelles équations des signaux : $\overline{X}$, $X1$ et $X2$.
Cette table comporte des trous (des cellules vides) car le recodage rend plusieurs combinaisons impossibles. Les X coïncident aux *cas impossibles* utilisés pour réduire les équations. Le coût supplémentaire par rapport au multiplieur de *Booth* classique est de deux portes logiques ET à trois entrées et de deux inverseurs par cellule *Booth*. Le délai est augmenté d'une couche logique.
Cette nouvelle cellule est exploitable à la fois en mixte et en classique. En faisant correspondre B_{i+1} à B^-_{i+1}, B_i à B^-_i et B_{i-1} à B^+_i, seule l'équation de $X2$ diffère : figure 3.27. En classique, le signal B^+_{i+1} est décomposé en deux afin d'annuler le deuxième terme du $X2$.

B_{i+1}^-	B_i^-	B_{i+1}^+	B_i^+	code	$\overline{X}$	$X1$	$X2$	$+1$
0	0	0	0	0	0	0	0	-
0	0	0	1	+1	0	1	0	-
0	0	1	0	+2	0	0	1	-
0	0	1	1	-	-	X	-	-
0	1	0	0	-	-	X	-	-
0	1	0	1	-	-	-	-	-
0	1	1	0	+1	0	1	0	-
0	1	1	1	+2	0	0	1	-
1	0	0	0	-2	1	0	1	1
1	0	0	1	-1	1	1	0	1
1	0	1	0	-	-	-	X	-
1	0	1	1	-	-	X	-	X
1	1	0	0	-	-	X	-	X
1	1	0	1	-	-	-	-	-
1	1	1	0	-1	1	1	0	1
1	1	1	1	0	1	0	0	-

$$
\begin{array}{lcl}
\overline{X} & = & B_{i+1}^- \\
X1 & = & B_i^- \oplus B_i^+ \\
X2 & = & B_{i+1}^- \oplus (B_i^- \& B_{i+1}^+ \& B_i^+) + \overline{B_i^-} \& B_{i+1}^+ \& \overline{B_i^+} \\
+1 & = & B_{i+1}^- \& \overline{(B_i^- \& B_{i+1}^+ \& B_i^+)} = B_{i+1}^- \& (X1 + X2)
\end{array}
$$

TAB. 3.3 – Multiplication de *Booth* mixte : cellule *Booth*

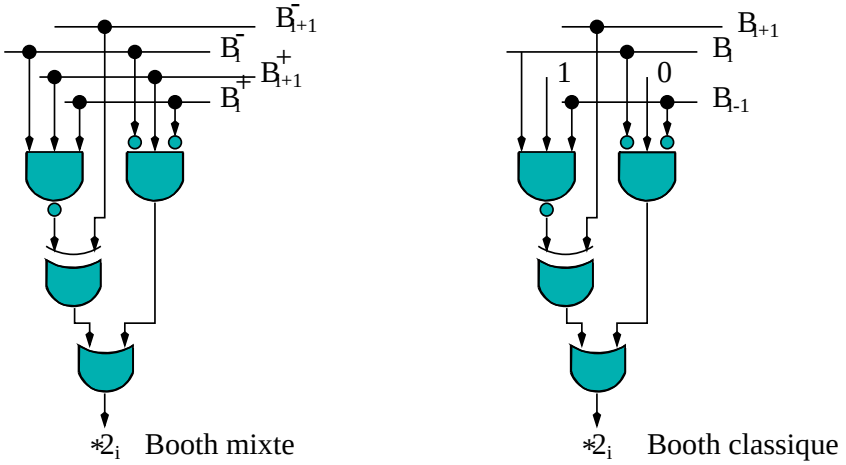


Figure 3.27 – Multiplieur de *Booth* mixte : correspondance entre cellule *Booth*

Multiplieurs mixtes $BS * NR$:

Comme pour la multiplication redondante, la prise en compte d'une opérande en notation BS demande juste d'effectuer le complément à deux du terme négatif du Borrow-Save. La modification de l'architecture est faite sur le recodage : une ligne d'inverseur et le positionnement

à 1 d'une des *retenues entrantes*. Le reste de l'architecture est identique.

Pour ces deux architectures et avec A_{NR} sur M chiffres et $B_{CS,BS}$ sur $N-1$ chiffres, le nombre de produits partiels à additionner est de $(M+2)\cdot((N+1)/2)$. Ce nombre est identique à celui obtenu dans le cas de l'utilisation de l'algorithme de *Booth* en notation classique, car l'extension de dynamique introduite par le recodage ramène le redondant de $N-1$ à N chiffres. Comme pour les multiplieurs directs, les sommes des multiplieurs de *Booth* classique et mixtes sont intégralement identiques. Ces deux multiplieurs de *Booth* mixte sont signés.

Autre point important à noter, le recodage en trois étages introduit une différence de chemin entre les deux entrées et le calcul effectif des produits partiels. Comme nous le verrons dans la sous-section 4.4.4 du chapitre 4, cet écart a un fort impact sur la consommation.

3.4.4 Multiplication signée

Les multiplications signées posent les mêmes problèmes que les sommes contenant des termes signés : sous-section 3.3.2. Les modifications à apporter au calcul des produits partiels, consistent juste à inverser les bits de signe des lignes de produits partiels, et à ajouter la constante de signe dans la somme des produits partiels. Toutes les architectures présentées ci-dessus, intègrent ces modifications.
Il est intéressant de noter que pour l'ensemble des multiplieurs *Directs*, toutes les combinaisons signées/non signées sont possibles en entrées. Avec l'algorithme de *Booth*, ces mêmes combinaisons sont aussi possibles. Toutefois, l'utilisation d'un codage différentiel implique des calculs signés. L'adjonction d'un bit de signe aux opérandes non signées est alors nécessaire et la dynamique de la sortie augmente de 1 à 2 bits suivant si une ou les deux entrées sont non signées.

3.5 Autres usages des opérateurs mixtes et redondants

Du point de vue arithmétique classique, les notations redondantes sont assimilables à des additions ou à des soustractions non effectuées. Cela sous-entend que les divers opérateurs mixtes et redondants développés effectuent préalablement ces additions / soustractions en plus de la tâche qui leur est initialement attribuée.
Dans ces conditions, rien n'empêche de détourner la fonctionnalité de ces opérateurs pour obtenir des fonctionnalités différentes en arithmétique classique. L'ajout d'une conversion $R \rightarrow NR$ est toutefois nécessaire si la sortie est souhaitée en notation classique.

Deux exemples pour montrer l'importance de ce *détournement* :

1. De nombreuses applications utilisent des filtres numériques FIR. Ceux-ci sont souvent symétriques. Pour réduire par deux le nombre de multiplications, on effectue préalablement à chaque multiplication, l'addition des termes multipliés par un même coefficient. L'opération effectuée correspond à un multiplieur mixte. Dans un DSP, le multiplieur de la MAC étant en arithmétique classique, il n'est pas possible de tirer parti de la symétrie. En introduisant un multiplieur mixte le nombre de cycles peut-être divisé par deux, pour un coût modique. Accessoirement cet opérateur peut servir d'additionneur ou de soustracteur. Si l'entrée non redondante est égale à 1, l'opération réalisée devient : $1 \cdot (A \pm B)$.
2. Une autre opération possible est le calcul de distance ($(A-B)^2$). Cette opération est présente dans les réseaux de neurones par exemple. Pour cela, le multiplieur redondant ou plutôt sa version dégradée R^2 est détournée : $(A - B) * (A - B)$. De même l'identité remarquable $A^2 - B^2$ peut-être obtenue avec le multiplieur redondant complet cette fois-ci : (A-B)*(A+B). Ici encore cet opérateur peut servir d'additionneur ou de soustracteur. Si une des entrées est égale à 1, l'opération effectuée devient : $(1 + 0) \cdot (A \pm B)$. L'opérateur de calcul de distance a été développé [Dumo01]. Ce travail est repris dans l'annexe B.

On voit ici, un autre usage des opérateurs mixtes et redondants, basé sur l'enchaînement addition / soustraction préalable et opérations proprement dîtes. C'est une bonne illustration de la souplesse des opérateurs mixtes.

3.6 Conclusion / Bilan

Dans ce chapitre, nous avons présenté un ensemble d'architectures permettant de couvrir toutes les combinaisons redondantes/classiques en entrée/sortie, pour les trois opérations élémentaires que sont l'addition, la somme et la multiplication. Les diverses structures proposées autorisent toutes les combinaisons signées/non signées en entrée. De plus elles respectent les dynamiques minimums de sortie.
Accessoirement, les opérateurs mixtes et totalement redondants peuvent être utilisés en arithmétique classique. Sous cet angle, leurs fonctionnalités diffèrent quelque peu et font apparaître de nouvelles fonctionnalités. Ce qu'il est important de retenir c'est que la compatibilité des représentations permet l'emploi de l'ensemble des opérateurs mixtes en arithmétique classique, et élargit le champ des opérateurs disponibles.
On peut noter que les complexités de l'ensemble des architectures mixtes et redondantes proposées pour la multiplication et la somme, sont identiques. Dans le cas de l'addition, ces indicateurs sont même meilleurs. Il reste à savoir, dans le détail, si les architectures mixtes et redondantes soutiennent la comparaison après réalisation physique.

Chapitre 4 Opérateurs élémentaires en arithmétique mixte : Performances

Ce chapitre est essentiellement consacré à l'évaluation et à la comparaison des architectures, des opérations élémentaires que sont l'addition, la sommation et la multiplication, en arithmétique mixte. Nous abordons successivement, la méthodologie d'évaluation, la méthode de conception $VLSI$ et la comparaison des principales architectures associées à ces trois opérations. Les critères de comparaisons sont la puissance consommée, le temps de propagation et la surface.
L'objectif de cette évaluation est double : à la fois positionner les opérateurs mixtes et redondants par rapport aux opérateurs classiques mais aussi démontrer l'intérêt de l'usage des notations redondantes.

4.1 Évaluation et comparaison

4.1.1 Méthodologie d'évaluation

L'évaluation des diverses architectures proposées peut se faire à plusieurs niveaux : théorique, structurel ou physique :

- Au niveau théorique, il s'agit d'étudier la complexité des diverses architectures proposées, afin de connaître leur croissance en surface, en délai, et si possible en consommation. La complexité permet de déterminer la validité des algorithmes employés, sur une plage de dynamique donnée. Malheureusement, si cet indicateur est incontournable, il n'est pas suffisant pour départager des architectures ayant des propriétés proches. C'est le cas des additionneurs et des multiplieurs.
- Au niveau structurel, l'algorithme est traduit sous forme de réseau de portes ou *Netlist*. La pluralité des critères possibles : surface, temps de propagation ou consommation, et la connaissance des structures, permettent une différenciation plus fine. Toutefois, les valeurs obtenues sont indépendantes de toute technologie et ne constituent que des indicateurs en-

core trop imprécis.

– Au niveau physique, les réseaux de portes associés aux algorithmes étudiés sont placés et routés. Les valeurs des critères tiennent compte des contraintes technologiques. Ce sont les estimations les plus précises, pouvant être obtenues pour une évaluation hors contexte d'utilisation.

Les niveaux structurel et physique sont redondants. Leur choix respectif dépend de la précision souhaitée, pour les critères. Dans notre cas, les architectures étant très proches, la précision de l'évaluation physique est nécessaire.

4.1.2 Critères d'évaluation

Dans le contexte actuel de la micro-électronique, l'évaluation physique porte par ordre décroissant sur :

1. La puissance consommée. Premièrement car dans de nombreuses applications : téléphonie, satellite, etc., les ressources en énergie sont limitées. Deuxièmement car avec l'augmentation de la densité d'intégration, la dissipation d'énergie pose de plus en plus de problèmes [Bork99].
2. Le temps de propagation. Ce critère ne vient qu'en second car le passage aux technologies submicroniques permet de tenir plus aisément les contraintes de délais. Le temps de propagation reste toutefois un critère important.
3. La surface. Son importance est moins prédominante qu'avant, car le passage aux technologies submicroniques a accru considérablement les possibilités d'intégrations. De plus, dans nombres de circuits actuels, la surface est plus l'expression de la mémoire intégrée que celle de la partie opérative.

Il est à noter que le délai et la surface sont indépendants du contexte d'utilisation du bloc placé-routé évalué. *A contrario*, la consommation est fortement corrélée au contexte d'utilisation.

4.1.3 Outils d'évaluations

Toutes les chaînes de CAO actuelles fournissent les outils nécessaires à l'évaluation du délai et de la surface. Dans le cas de la consommation, les outils sont émergeant voire indisponibles.
En ce qui nous concerne, la technique d'estimation de la consommation qui a été employée, est basée sur les travaux de thèse menés par Julien Dunoyer [Duno99], au sein de notre laboratoire. Deux techniques d'évaluations sont possibles. La première est basée sur la simulation avec un jeu de vecteurs de tests (*patterns*), la seconde sur une approche probabiliste. Nous avons choisi la première.

Utiliser ces travaux implique l'emploi de la bibliothèque de cellules précaractérisées de la chaîne Alliance [Grei92] http ://www-asim.lip6.fr/alliance/. La bibliothèque utilisée est en $0.35\mu m$. Pour que les estimations de surface, de délai et de consommation obtenues, soient cohérentes, toutes les architectures ont été réalisées avec les outils de la chaîne Alliance.
Le positionnement du placement/routage en entrée du flot de conception, et l'actuelle mise au point d'un *design-kit* Alliance pour Cadence [CAD] nous a permis d'utiliser le dernier placeur/routeur de Cadence : *Silicon-Ensemble*.

4.1.4 Comparaison

Le premier objectif de la comparaison est de valider notre postulat de départ énonçant que *les architectures mixtes et redondantes associées aux opérations d'addition et de multiplication, sont globalement plus performantes que les architectures classiques*. Le second est de mesurer, pour une opération donnée, les écarts de performances entre chaque type d'architecture.

Du point de vue structurel, les architectures non signées et signées associées à un opérateur, sont très proches. De même, les soustracteurs sont sensiblement identiques aux additionneurs. Aussi, l'étude d'une seule structure par version d'architecture, est nécessaire à la comparaison. Comme toujours, il existe une exception. Dans notre cas, c'est la multiplication redondante. Les architectures associées peuvent être réalisées avec un ou deux recodages sous-section 3.4.2. Pour la comparaison, trois structures sont proposées : le multiplieur $BS \cdot BS$ pour connaître l'impact des entrées en notation Borrow-Save, et deux versions du multiplieur $CS \cdot CS$ pour connaître les différences de résultats entre une architecture où une seule entrée est recodée et une architecture où les deux entrées sont recodées.

La comparaison porte simultanément sur les trois critères sélectionnés. Les opérateurs classiques servent de référence. Toutes les évaluations sont faites avec des circuits purement combinatoires.

4.1.5 Un mot sur la réalisation matérielle des architectures

L'approche utilisée pour réaliser l'ensemble des opérateurs étudiés dans ce chapitre, est basée sur la génération. Ce choix s'inscrit dans la volonté plus large d'encapsuler notre savoir-faire afin de le réutiliser à volonté.
Les architectures fournies sont sous forme de listes de portes précaractérisées accompagnées de leurs interconnexions (*netlist*). Pour chaque opérateur, il est possible de spécifier l'algorithme utilisé ainsi que la taille, la notation et le signe de chacune des entrées.

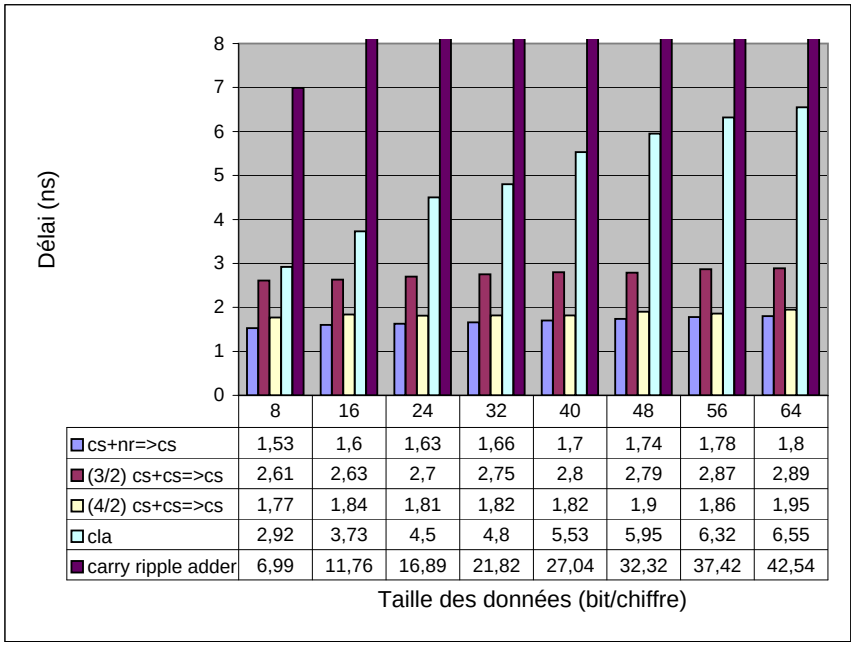
Il est à noter que pour l'ensemble des circuits, un placement de forme carré est utilisé par défaut. On pourrait penser que fournir un pré-placement serait un plus permettant de profiter de la régularité des opérateurs arithmétiques. Toutefois, cela n'a pas de sens car lors de leur utilisation ces opérateurs sont immergés dans un plus grand ensemble rendant tout pré-placement inutilisable. Nous verrons par la suite que ce placement par défaut a lui aussi des conséquences notamment sur les délais hors contexte d'opérateurs comme les additionneurs mixtes et redondants.

4.2 Évaluation des additionneurs

Dans le chapitre précédent, trois additionneurs classiques ont été présentés : le *Carry Ripple Adder*, le *Carry Skip Adder* et le *CLA*. La connaissance de ces trois structures et de leur complexité respective, permet de dire que les performances du *Carry Skip Adder* sont comprises entre celle du *Carry Ripple Adder* et du *CLA*. C'est vrai quel que soit le critère. Cette remarque est d'ailleurs valable pour l'ensemble des autres additionneurs classiques [Mull97, Parh99] comme le *Carry Select Adder*.
Détailler les valeurs des trois critères pour le *Carry Skip Adder* n'est donc pas nécessaire à la comparaison. Son architecture a été essentiellement présentée pour montrer qu'il existe des solutions intermédiaires entre les additionneurs *à retenue séquentielle* et *à retenue anticipée*.
Les additionneurs *Carry Ripple Adder* et *CLA* vont servir de points de références à l'évaluation des additionneurs mixtes et redondants.

4.2.1 Délai

La figure 4.1 présente les temps de propagations des additionneurs classiques, mixtes et redondants, en fonction de la dynamique des entrées. Les valeurs de temps indiquées sont en ns.
Afin de bien visualiser l'évolution des écarts de performances entre les divers additionneurs, les temps de propagations de l'additionneur *Carry Ripple Adder* portés sur le graphe, sont tronquées dès 16 bits. Ce choix s'explique par les mauvaises performances de cet additionneur. Dès 8 bits, son temps de propagation est supérieur à ceux de l'ensemble des autres additionneurs et ce quelle que soit la dynamique des entrées. Ce constat nous amène à ne prendre que les valeurs du *CLA* comme référence.
Comme attendu, les additionneurs mixtes et redondants sont en temps constant. L'évolution des délais entre les architectures 8 bits et 64 bits (+10% pour les additionneurs redondants et 17% pour les additionneurs mixtes) est artificielle. Les écarts de délai sont liés à la forme du placement utilisé. Ce dernier (par défaut carré) ne suit pas la distribution en tranche (poids/bit-



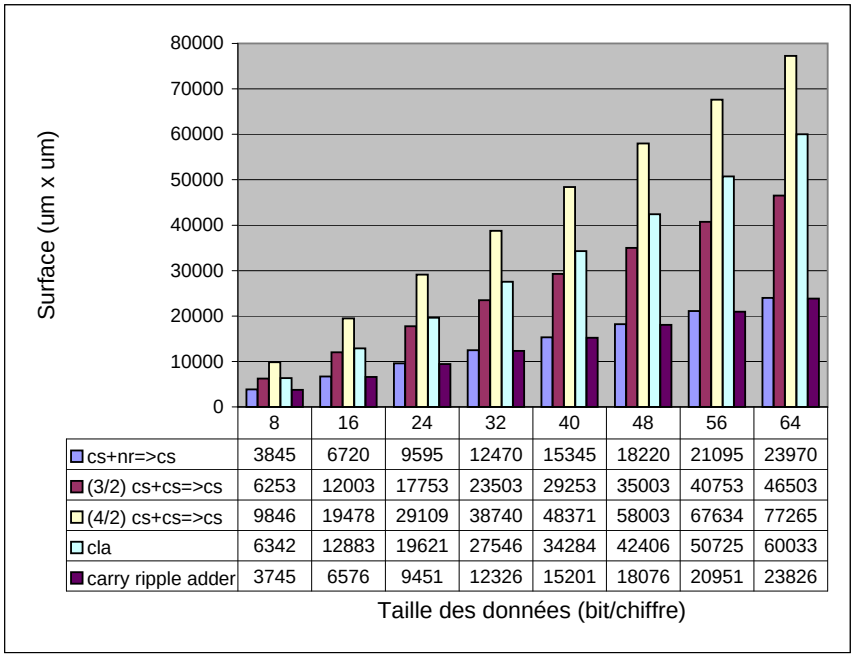
	8	16	24	32	40	48	56	64
cs+nr=>cs	1,53	1,6	1,63	1,66	1,7	1,74	1,78	1,8
(3/2) cs+cs=>cs	2,61	2,63	2,7	2,75	2,8	2,79	2,87	2,89
(4/2) cs+cs=>cs	1,77	1,84	1,81	1,82	1,82	1,9	1,86	1,95
cla	2,92	3,73	4,5	4,8	5,53	5,95	6,32	6,55
carry ripple adder	6,99	11,76	16,89	21,82	27,04	32,32	37,42	42,54

Figure 4.1 – Additionneurs : comparaison en délai

slice) des FA (4/2) d'un placement idéal pour les additionneurs mixtes et redondants. Aussi le routage des connecteurs vers l'instance souhaitée est plus importants ; cela se traduit par une légère augmentation des délais. Les écarts constates semblent importants car les temps de propagation des additionneurs mixtes et redondants sont faibles.
Le gain en temps, appelons le G_t, introduit par les additionneurs mixtes et redondants, est important. Dans le cas de l'additionneur mixte G_t est compris entre 47% et 73% pour une dynamique d'entrée respectivement comprise entre 8 et 64 bits. Pour les additionneurs redondants, l'évolution de G_t est la même à une constante près. Avec la structure *3/2* on a : 10% <G_t< 56%, et avec la structure *4/2* on a : 40% <G_t< 70%.

Deux remarques :

1. Le rapport de performance entre les deux structures redondantes de *type 3/2* et de *type 4/2* est de 0.66. Cette valeur est inférieure au rapport théorique de 0.75 (3/4) attendu : sous-section 3.2.4. De même, le rapport entre l'additionneur redondant de *type 4/2*, et l'additionneur mixte est de 1.16 au lieu de 1.5 (3/2). Ces écarts sont liés aux performances moyennes de la cellule pré-caractérisée Full-Adder de la librairie Alliance.
2. Le rapport de temps minimum entre les additionneurs *Carry Ripple Adder* et mixtes est de 4.56. Entre ce même *Carry Ripple Adder* et les deux additionneurs redondants le rapport est de 3.95 et de 2.7 respectivement pour les structures de type *4/2* et *3/2*.



	8	16	24	32	40	48	56	64
cs+nr=>cs	3845	6720	9595	12470	15345	18220	21095	23970
(3/2) cs+cs=>cs	6253	12003	17753	23503	29253	35003	40753	46503
(4/2) cs+cs=>cs	9846	19478	29109	38740	48371	58003	67634	77265
cla	6342	12883	19621	27546	34284	42406	50725	60033
carry ripple adder	3745	6576	9451	12326	15201	18076	20951	23826

Figure 4.2 – Additionneurs : comparaison en surface

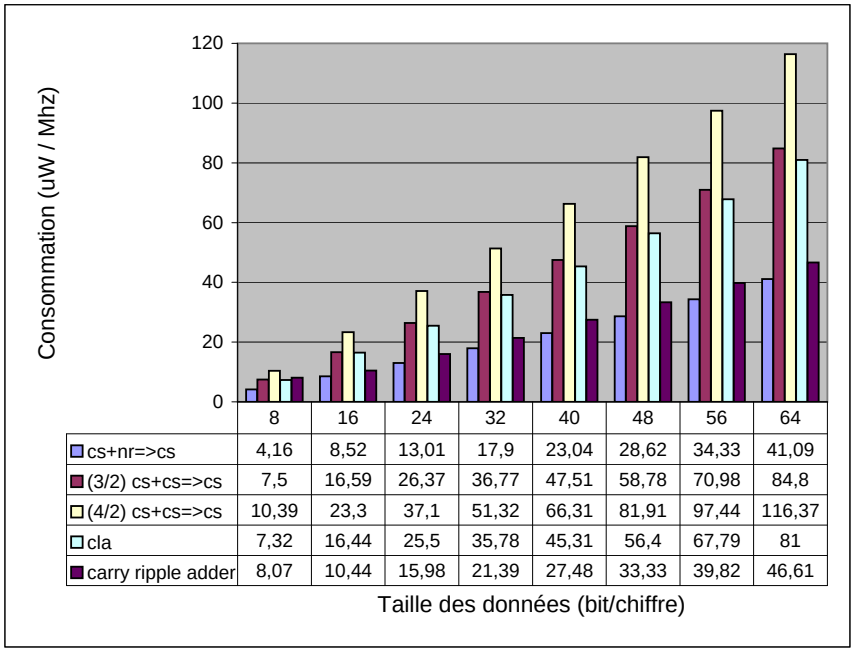
4.2.2 Surface

Avant d'effectuer la comparaison des divers additionneurs, il est important de noter que la cellule précaractérisée *4/2* n'existant pas, l'additionneur en *structure 4/2* est réalisé en portes élémentaires. Cette particularité implique que les valeurs portées sur le graphe qui suit, sont donc supérieures à ce qu'elles devraient être. Théoriquement, ces surfaces devraient être très sensiblement identiques à celles des additionneurs redondants en *structure 3/2*.

Le graphe figure 4.2, reprend l'ensemble des surfaces obtenues pour les divers additionneurs classiques, mixtes et redondants, en fonction de la dynamique des entrées. Les valeurs de surfaces indiquées sont en μm^2.

Globalement les additionneurs mixtes et redondants, offrent des résultats meilleurs que les additionneurs classiques. Quelle que soit la dynamique des opérandes, les plus petites surfaces sont celles des additionneurs *Carry Ripple Adder* et mixte. L'additionneur redondant de *type 4/2* étant écarté, les plus grandes surfaces sont celles du *CLA*.

Dans ce contexte, le rapport de surface entre les additionneurs redondants et le *Carry Ripple Adder* est légèrement inférieur à 2. Avec l'additionneur *CLA* comme référence, ce rapport s'inverse. Le gain (G_s) est en faveur des opérateurs redondants et augmente avec la dynamique : 2.5% <G_s< 22.5%. La variation est liée à la composante $log_2(N)$ de la complexité du *CLA*. Dans le cas de l'additionneur mixte et avec le *CLA* comme référence, on a : 40% <G_s< 69%.



	8	16	24	32	40	48	56	64
cs+nr=>cs	4,16	8,52	13,01	17,9	23,04	28,62	34,33	41,09
(3/2) cs+cs=>cs	7,5	16,59	26,37	36,77	47,51	58,78	70,98	84,8
(4/2) cs+cs=>cs	10,39	23,3	37,1	51,32	66,31	81,91	97,44	116,37
cla	7,32	16,44	25,5	35,78	45,31	56,4	67,79	81
carry ripple adder	8,07	10,44	15,98	21,39	27,48	33,33	39,82	46,61

Figure 4.3 – Additionneurs : comparaison en consommation

Plusieurs remarques :

1. Le *Carry Ripple Adder* partage certes la plus petite surface avec l'additionneur mixte, mais ces mauvaises performances en délai font qu'il est difficilement exploitable. Comme pour le délai, le *CLA* s'impose comme la référence.
2. Dans l'ensemble, la croissance de la surface, des divers additionneurs, est linéaire. Ce résultat suit les complexités données dans la section 3.2. Ce n'est pas tout à fait vrai pour le *CLA* car sa complexité est en $N.log_2(N)$. Dans son cas, la croissante est quasi linéaire dans le domaine de valeur de N qui nous intéresse.
3. Le rapport de surface entre les additionneurs mixtes et redondants est proche de deux. Cette valeur correspond parfaitement à la conception de l'additionneur redondant de *type 3/2*, basée sur l'enchaînement de deux additionneurs mixtes (figure 3.10).

4.2.3 Consommation

La figure 4.3 présente la consommation obtenue par les divers additionneurs classiques, mixtes et redondants, en fonction de la dynamique des entrées. Les valeurs de consommation indiquées sont en $\mu W/MHz$.
Dans l'ensemble, les performances des additionneurs mixtes et redondants sont comparables à celles des additionneurs classiques. La plus faible consommation est obtenue par l'addition-

neur mixte, suivi de très près par le *Carry Ripple Adder*. Les écarts sont dûs aux commutations transitoires (*glitches*) générées par la chaîne de propagation de la retenue entrante. Le matériel dans les deux architectures est pratiquement identique.
Les plus fortes consommations sont obtenues à la fois par l'additionneur redondant en *structure 3/2* et par le *CLA*, ce dernier ayant des résultats légèrement meilleurs. L'écart de gain G_c, est tel que 2.5% <G_c< 4.5%. L'architecture redondante de *type 4/2* a été écartée pour les mêmes raisons que lors de l'étude de la surface.
Dans le cas de l'additionneur mixte et avec le *CLA* comme référence, le gain G_c est tel que 43% <G_c< 49%. Ces gains sont identiques entre le *Carry Ripple Adder* et le *CLA* et entre les additionneurs mixtes et redondants. L'inverse de ces valeurs est proche de deux. Dans le cas des opérateurs mixtes et redondants ce rapport correspond au nombre de couches de FA.

Remarques :

1. À la lecture du graphe on constate que les rapports de consommations sont quasi constants. Cette remarque est valable pour tous les additionneurs. Globalement, l'évolution de la consommation suit celle de la surface, et ce, quelle que soit l'architecture étudiée.
2. Le matériel des additionneurs redondants de type $4/2$ et $3/2$, étant le même pour un nombre de couches logiques plus faible, la consommation de l'additionneur en *structure 4/2* devrait être moins importante que celle de l'additionneur en *structure 3/2*.

4.2.4 Synthèse

L'additionneur *Carry Ripple Adder* ayant de très mauvaises performances en délai, il est rarement utilisé. Toutefois ces performances sont intéressantes pour connaître les bornes inférieures de consommation et de surface des additionneurs classiques.
Les augmentations moyennes de consommation et de surface entre cet additionneur et les additionneurs redondants sont respectivement de 70% et de 80%. Ces valeurs peuvent sembler importantes, mais les rapports de surface entre un *Carry Ripple Adder* et un *Carry Skip Adder* ou un *Carry Select Adder* sont aussi proches de ces valeurs, et la consommation suit la surface. L'additionneur mixte présente des surfaces et des consommations identiques à celles de l'additionneur *Carry Ripple Adder*.
Avec le *CLA* comme référence, les gains minimums apportés par l'additionneur mixte sont supérieurs à 47% en délai, 40% en surface et 43% en consommation. Pour les additionneurs redondants les gains en délai commencent à 10% et peuvent dépasser les 56% dans le cas de la *structure 3/2*. Ils sont au minimum de 40% pour la *structure 4/2*. Le gain en surface part de 2.5% et va jusqu'à 22%. La consommation est, elle, la même.

Globalement, les performances obtenues par les additionneurs mixtes et redondants sont sensiblement meilleures que celles obtenues par les additionneurs classiques. Au pire elles sont identiques.
Dans le cas des additionneurs mixtes, les résultats sont systématiquement meilleurs, quels que soient le critère et l'additionneur classique pris comme référence. Les écarts de performances sont très importants. Dans le cas des additionneurs redondants, les surfaces et les consommations sont sensiblement identiques, pour un gain en délai significatif.

4.3 Évaluation de la somme

Si les architectures classiques, mixtes et redondantes sont bien tranchées pour l'addition et la multiplication, il n'en va de même pour la somme. Certes, l'algorithme employé est le même, mais le nombre d'entrées ainsi que leur dynamique respective sont variables. La comparaison devient alors des plus difficiles. Aussi dans cette section, nous nous contenterons de donner des indications quant aux performances respectives des sommes en notations classiques et redondantes. Nous nous intéresserons plus spécifiquement à étalonner les sommes redondantes par rapport aux sommes classiques.

4.3.1 Délai

Nous avons vu (section 3.3) que pour une même somme (même nombre d'entrées sur les mêmes dynamiques), le réducteur de Wallace redondant nécessite une couche de réduction $4 \rightarrow 2$ supplémentaire. Le temps de propagation de l'arbre de Wallace est augmenté d'autant. Concernant la somme en notation classique, une conversion $CS \rightarrow NR$ est nécessaire en plus de l'arbre de Wallace.
Dans ces conditions, la différence de performance en délai entre les deux sommes, classique et redondante, est égale à la différence de temps de propagation entre la conversion de notation et la couche de réduction supplémentaire.
Comme le temps de propagation de la réduction $4 \rightarrow 2$ est toujours inférieur à celui du meilleur additionneur classique correspondant (figure 4.1), la somme redondante est toujours plus rapide que la somme classique. Le gain est variable et croît avec la dynamique globale des données ; il est aussi fonction de l'existence de cellule de type $4/2$ précaractérisées.

4.3.2 Surface

Effectuer la même somme en classique et en redondant (même nombre d'entrées sur les mêmes dynamiques) sous-entend la multiplication par deux du nombre de termes à sommer (section

3.3). Ce doublement implique le doublement du matériel nécessaire à la réduction, plus une couche $4 \rightarrow 2$ supplémentaire de réduction.
Comme on peut le voir sur la figure 4.2, la réduction $4 \rightarrow 2$ a la même surface qu'un additionneur classique (CLA). Leur coût respectif s'annule, et l'augmentation de la surface est égale au doublement de l'arbre de réduction initial (entrées en notation classique). Plus le nombre d'entrées à sommer augmente plus le différentiel de surface augmente.

4.3.3 Consommation

Proposer une évaluation de l'évolution de la consommation entre les réalisations classique et redondante de la même somme est très délicat. Deux critères sont à prendre en compte : la différence de quantité de couches combinatoires traversées et le doublement du matériel nécessaire au réducteur de Wallace initial.
Dans le cas de petites sommes (nombre faible d'entrées) le facteur *nombre de couches combinatoires* est prépondérant. Les additionneurs mixtes et redondants (section 3.2) sont l'illustration de ce propos. Avec l'augmentation du nombre d'entrées c'est le deuxième facteur qui devient prépondérant. Dans ces conditions, déterminer le point de basculement des performances est délicat. Cette incertitude est amplifiée par le fait que le basculement est dépendant de la technologie. Nous nous contenterons de dire que plus une somme a d'entrées, plus sa réalisation totalement redondante consommera que sa version classique.
Concernant la technologie que nous avons utilisée, les consommations des additionneurs $4/2$ et classiques (CLA) sont identiques. Dans ce cas le basculement s'effectue après 2 entrées, et la réalisation redondante d'une somme donnée, consommera systématiquement plus que sa version classique. Ce basculement précoce est entre autre lié à l'inexistence de cellule de type $4/2$ et aux *mauvaises* propriétés des Full-Adders utilisés.

4.3.4 Synthèse

Extraire des règles précises et générales concernant les performances respectives des réalisations classiques et redondantes d'une somme quelconque (nombre d'entrées et dynamiques respectives données) est plutôt difficile. Toutefois, il est possible de fournir des indications générales.
Concernant les délais, les sommes redondantes seront toujours les plus performantes. Le gain est égal à la différence de propagation entre un additionneur de type $4/2$ et un additionneur classique dont la dynamique des entrées est égale à la dynamique de sortie du réducteur de Wallace de la somme.

Concernant la surface, les sommes classiques seront toujours les plus petites. La différence est égale à la surface du réducteur de la somme classique. Le rapport de surface est forcément inférieur à deux.
Concernant la consommation, dégager une indication générale est difficile. Avec l'augmentation du nombre d'entrées, les consommations des deux réalisations se croisent et leur différence change de signe. Globalement les versions redondantes sont les moins consommatrices quand le nombre d'entrées est faible, et *a contrario* les réalisations classiques sont les moins consommatrices quand le nombre d'entrées est important. Dans notre cas, nous considérerons que les sommes redondantes sont systématiquement les plus consommatrices. Comme pour la surface, le rapport est limité à deux.
Donner des résultats concernant les réalisations mixtes des sommes est encore plus délicat que pour les sommes redondantes et ce, pour les mêmes raisons que celles évoquées dans la section 3.3. Toutefois nous pouvons dire que quels que soient les critères, les résultats des versions mixtes sont meilleurs que ceux des sommes redondantes. Les surfaces et les consommations peuvent même être inférieures à celles des versions classiques (additionneurs mixtes figures 4.2 et 4.3).

4.4 Évaluation des multiplieurs

Afin de faciliter la comparaison des divers multiplieurs, les résultats sont décodés en deux temps. Pour chaque critère, la comparaison porte d'abord sur les multiplieurs mixtes, puis sur les multiplieurs redondants.

Deux remarques :

1. Afin d'obtenir les meilleures performances possibles en délai, les conversions $R \rightarrow NR$ employées, sont des additionneurs *CLA*.
2. L'abréviation *Rec.* est utilisée sur les divers graphes ci-dessous, pour indiquer un recodage.

4.4.1 Choix de la référence de comparaison

Comme avec les additionneurs, il se pose le problème du choix d'une référence pour la comparaison des divers multiplieurs. Dans le chapitre précédent sous-section 3.4.1, deux architectures ont été présentées, pour effectuer la multiplication en arithmétique classique : le multiplieur *Direct* et le multiplieur de *Booth*.
Quel que soit le critère, ces deux architectures ont des complexités comparables. Toutefois l'algorithme de *Booth* permet de réduire par deux le nombre de produits partiels générés. Par

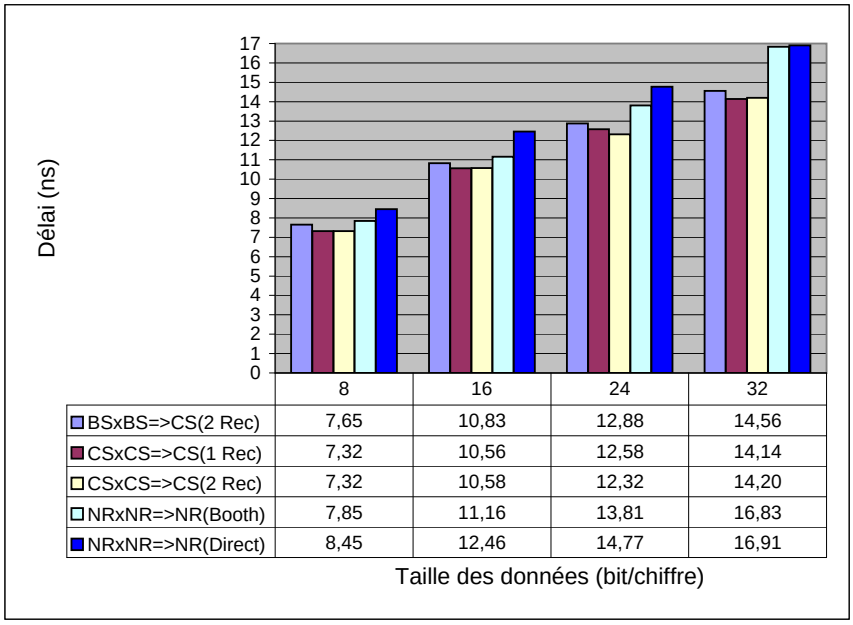
suite, la somme est, elle aussi deux fois plus petite. Théoriquement, ce multiplieur est le plus performant à la fois sur le plan du délai, de la surface et *a priori* de la consommation.
Seulement la cellule de la matrice de produits partiels est plus considérable (une cellule ET en *Direct* contre une cellule [(a ET b) OU (c ET d)] avec l'algorithme de *Booth*). Cela réduit fortement l'impact du rapport de 2 en surface et augmente le délai et la consommation.
De plus, en notation non signée, le choix de l'algorithme de *Booth* introduit plusieurs problèmes. Le rajout d'un bit de signe à chaque entrée, implique l'augmentation du nombre de produits partiels à générer et à sommer, et nécessite des mécanismes de réduction de la dynamique de la sortie. Ces problèmes entraînent forcément une diminution de la qualité de ses performances.
Mais le vrai point négatif de cette architecture vient de la décomposition du calcul des produits partiels en deux types de cellules : une cellule de matrice et une cellule de tête de ligne (figure 3.18). Plus précisément, le problème vient du décalage temporel, introduit par la cellule de tête, entre les deux entrées. Ce décalage est propice à la génération de commutations transitoires : *glitches*. Ces commutations sont produites dès les entrées et se propagent à travers toute l'architecture. Comme nous allons le voir elles ont un fort impact sur la consommation. Ces remarques sont aussi applicables aux multiplieurs de *Booth* en arithmétique mixte.

Nous utiliserons donc le multiplieur classique *Direct* comme référence. Les performances du multiplieur classique de Booth sont aussi détaillées.

4.4.2 Délai

Multiplieurs mixtes : délai

La figure 4.4 présente les délais des divers multiplieurs mixtes et classiques, en fonction de la dynamique des entrées. Les valeurs des délais sont indiquées en ns.
Globalement, les multiplieurs mixtes ont des performances identiques. Toutefois quand la dynamique est faible, les multiplieurs *Directs* offrent des résultats meilleurs de 5% à 7%. Comparés aux multiplieurs *Directs* classiques, cet ensemble de multiplieurs introduit un gain G_t quasi-constant de 27%. L'amélioration du temps de propagation est considérable.
Que le gain reste constant sur toute la dynamique est normal car dans chacune des deux parties composant n'importe lequel des multiplieurs (figure 3.16), les blocs ont la même complexité en délai ; les temps de propagations des recodages et du calcul des produits partiels sont constants, et la propagation dans une somme et dans la conversion $R \rightarrow NR$ de sortie, est en $log_2(N)$ (section et sous-sections 3.3 et 3.2.2).



	8	16	24	32
BSxBS=>CS(2 Rec)	7,65	10,83	12,88	14,56
CSxCS=>CS(1 Rec)	7,32	10,56	12,58	14,14
CSxCS=>CS(2 Rec)	7,32	10,58	12,32	14,20
NRxNR=>NR(Booth)	7,85	11,16	13,81	16,83
NRxNR=>NR(Direct)	8,45	12,46	14,77	16,91

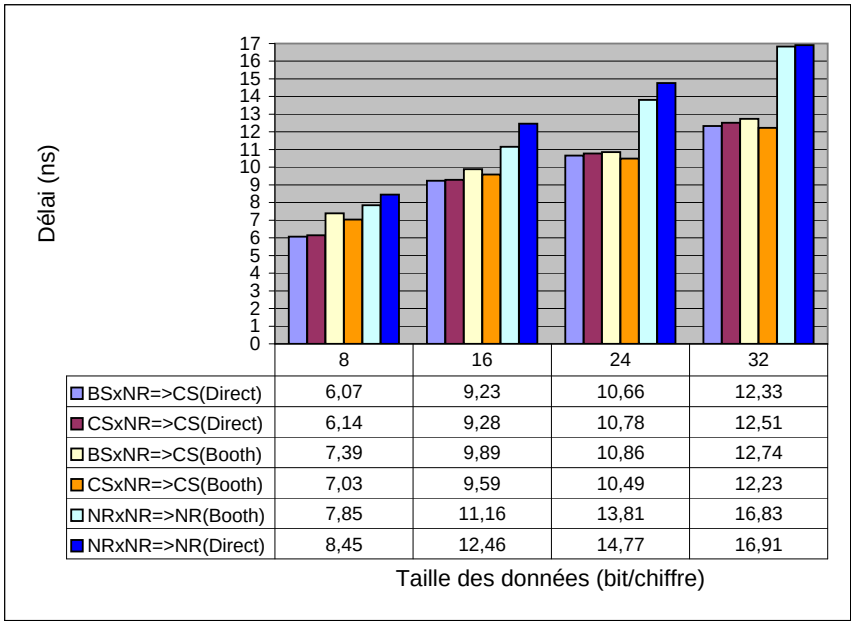
Figure 4.4 – Multiplieurs mixtes : comparaison en délai

Deux remarques :

1. Le multiplieur classique de *Booth* offre des délais légèrement meilleurs que ceux du multiplieur *Direct*, -7% <G_t< 0%. Ces performances sont toutefois inférieures à celle des multiplieurs mixtes de -19% à -27%.
2. Les fluctuations des gains introduits par les multiplieurs de *Booth* mixte et classique sont essentiellement dues à la réduction de la complexité du routage entre les produits partiels et la somme.

Multiplieurs redondants : délai

La figure 4.5 présente les temps de propagations des divers multiplieurs redondants et classiques, en fonction de la dynamique des entrées. Les valeurs de temps indiquées pour le délai sont en ns.
Comme pour les multiplieurs mixtes, les multiplieurs redondants introduisent un gain en temps (G_t) à peu près constant. La valeur de G_t est de 15%. Ce gain est valable quelles que soient l'architecture et la dynamique des entrées. Les écarts entre multiplieurs redondants sont dus à la prise en compte des termes à soustraire des Borrow-Save.
Le gain G_t est moins important que pour les multiplieurs mixtes. Les écarts de performances sont dus aux recodages des entrées. Leur coût temporel est compris entre 13% et 5% du délai



	8	16	24	32
BSxNR=>CS(Direct)	6,07	9,23	10,66	12,33
CSxNR=>CS(Direct)	6,14	9,28	10,78	12,51
BSxNR=>CS(Booth)	7,39	9,89	10,86	12,74
CSxNR=>CS(Booth)	7,03	9,59	10,49	12,23
NRxNR=>NR(Booth)	7,85	11,16	13,81	16,83
NRxNR=>NR(Direct)	8,45	12,46	14,77	16,91

Figure 4.5 – Multiplieurs redondants : comparaison en délai

total, la première valeur correspondant aux multiplieurs redondants 8 bits et la seconde aux multiplieurs 32 bits.

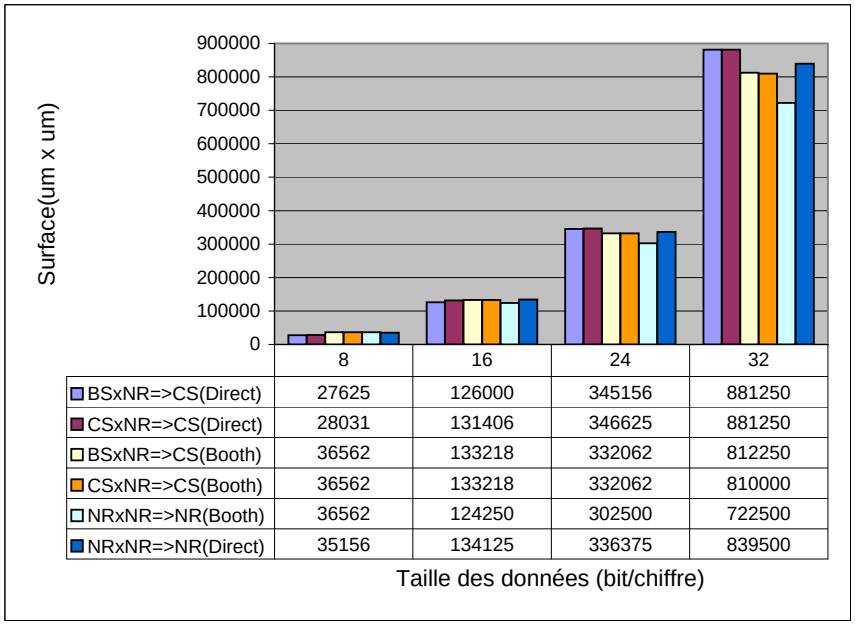
Remarques :

1. Les gains de performances, entre le multiplieur classique de *Booth* et les multiplieurs redondants, sont compris entre 7% et 16% (le premier multiplieur servant de référence).
2. Les remarques sur l'aspect *gain constant* et les complexités, faites pour les multiplieurs mixtes, sont aussi valables pour les multiplieurs redondants.
3. Le coût temporel de la conversion de notations internes aux multiplieurs classiques, est de 34%. Cette valeur est constante sur toute la dynamique.

4.4.3 Surface

Multiplieurs mixtes : surface

Les figures 4.6 présentent les surfaces des divers multiplieurs mixtes et classiques, en fonction de la dynamique des entrées. Les valeurs de surfaces indiquées sont en μm^2.
Dans l'ensemble, les surfaces des multiplieurs mixtes sont plus petites ou égales, à celles du multiplieur classique *Direct*. Le gain en surface (G_s) est compris entre 20% et -5% pour les architectures mixtes *Directes*. Ce même gain est inclus entre 4% et -4%, pour les architectures



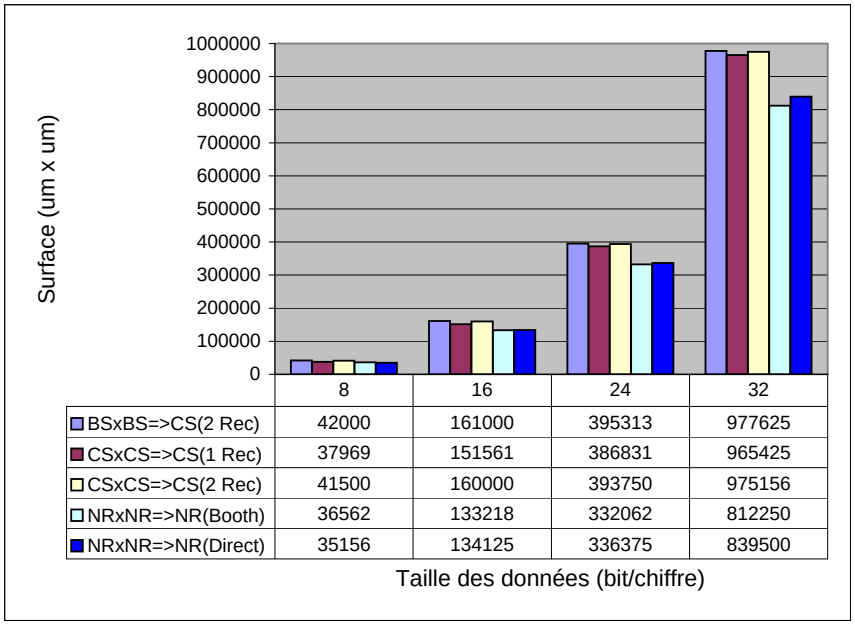
	8	16	24	32
BSxNR=>CS(Direct)	27625	126000	345156	881250
CSxNR=>CS(Direct)	28031	131406	346625	881250
BSxNR=>CS(Booth)	36562	133218	332062	812250
CSxNR=>CS(Booth)	36562	133218	332062	810000
NRxNR=>NR(Booth)	36562	124250	302500	722500
NRxNR=>NR(Direct)	35156	134125	336375	839500

Figure 4.6 – Multiplieurs mixtes : comparaison en surface

mixtes utilisant l'algorithme de *Booth*. L'évolution de G_s est liée à la combinaison de trois facteurs. D'abord la différence de croissance de la matrice et de la somme (conversion incluse), respectivement en $O(N^2)$ et en $O(N.log_2 N)$, ensuite à la différence de cellule élémentaire de calcul des produits partiels, et à l'utilisation ou de la non utilisation de conversion $R \rightarrow NR$. Passée une valeur de N, il s'opère un changement de la partie du multiplieur prédominante ; ici le passage s'effectue de la somme vers la matrice.
L'inversion du gain s'opère à une dynamique légèrement supérieure à 16 bits pour les multiplieurs mixtes *Directs*, et à une dynamique supérieure à 24 bits pour les multiplieurs mixtes de *Booth*.

Remarques :

1. Les multiplieurs de *Booth* classiques offre des surfaces légèrement meilleures que celles des multiplieurs *Directs* classiques tel que -4% <G_s< 4%. Il en est de même entre les multiplieurs de *Booth* et *Directs* mixtes.
2. Avec le multiplieur classique de *Booth* comme référence, G_s est compris entre 24% et -22%, pour l'algorithme *Direct* en mixte, et entre 0% et 12%, pour l'algorithme de *Booth* en mixte.
3. La croissance des architectures de *Booth* est plus faible que celle des architectures *Directes*. Comme nous l'avons déjà indiqué, la réduction de la complexité du routage entre les produits partiels et la somme, prend une part importante dans ce phénomène.



	8	16	24	32
BSxBS=>CS(2 Rec)	42000	161000	395313	977625
CSxCS=>CS(1 Rec)	37969	151561	386831	965425
CSxCS=>CS(2 Rec)	41500	160000	393750	975156
NRxNR=>NR(Booth)	36562	133218	332062	812250
NRxNR=>NR(Direct)	35156	134125	336375	839500

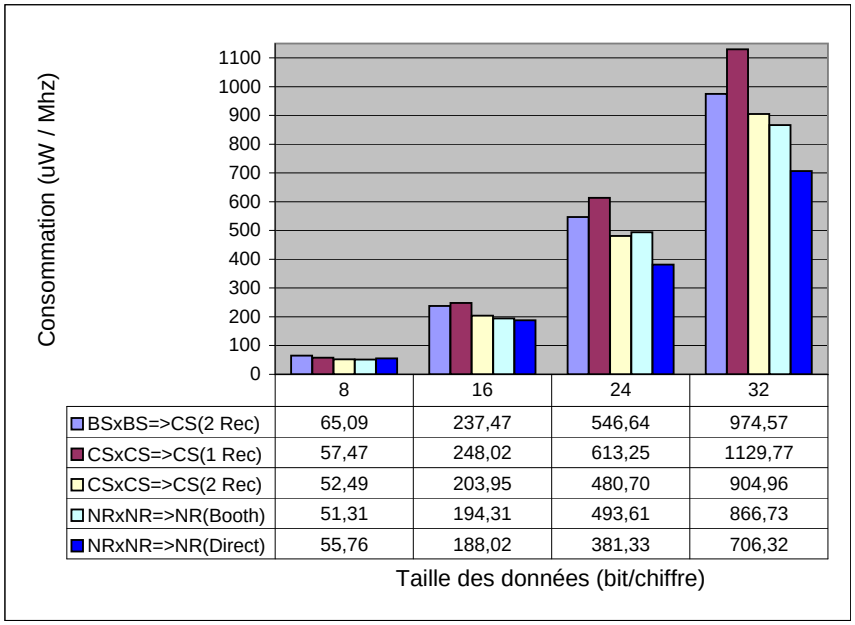
Figure 4.7 – Multiplieurs redondants : comparaison en surface

Multiplieurs redondants : surface

Les surfaces des divers multiplieurs redondants et classiques sont présentées sur la figure 4.7. Elles sont fonction de la dynamique des entrées, et indiquées en μm^2.
Tous les multiplieurs redondants présentent une surface supérieure à celle de la référence. Dans le cas des multiplieurs ayant deux recodages (2 Rec. sur le graphe), l'augmentation est de 18%, et est quasi constante. Le léger écart entre les deux structures $CS \cdot CS$ et $BS \cdot CS$, est dû aux inversions des termes à soustraire des entrées en notation Borrow-Save. Cet écart s'estompe avec l'augmentation de la dynamique. Cela est du à la différence de croissance entre les inverseurs et celles de la matrice des produits partiels et de la somme, respectivement $O(N)$ contre $O(N^2)$ et $N.log_2(N)$. Que le différentiel en surface entre les divers multiplieurs soit constant sur la plage de dynamique testée semble indiquer un équilibrage des croissances des recodages (multiplieurs redondants) et de l'additionneur de sortie (multiplieurs classiques).
Dans le cas du multiplieur $CS \cdot CS$ avec un seul recodage (1 Rec. sur le graphe), l'augmentation est plus faible et est comprise entre 8% et 15%. Elle n'est pas constante. La variation concerne essentiellement les petites dynamiques. L'augmentation est déjà de 13% à 16 bits.

Remarques :

1. Pour des raisons de consommation, les architectures redondantes développées intègrent deux recodages alors qu'un seul est nécessaire (sous-section 3.4.2). Si la surface est le cri-



	8	16	24	32
BSxBS=>CS(2 Rec)	65,09	237,47	546,64	974,57
CSxCS=>CS(1 Rec)	57,47	248,02	613,25	1129,77
CSxCS=>CS(2 Rec)	52,49	203,95	480,70	904,96
NRxNR=>NR(Booth)	51,31	194,31	493,61	866,73
NRxNR=>NR(Direct)	55,76	188,02	381,33	706,32

Figure 4.8 – Multiplieurs mixtes : comparaison en consommation

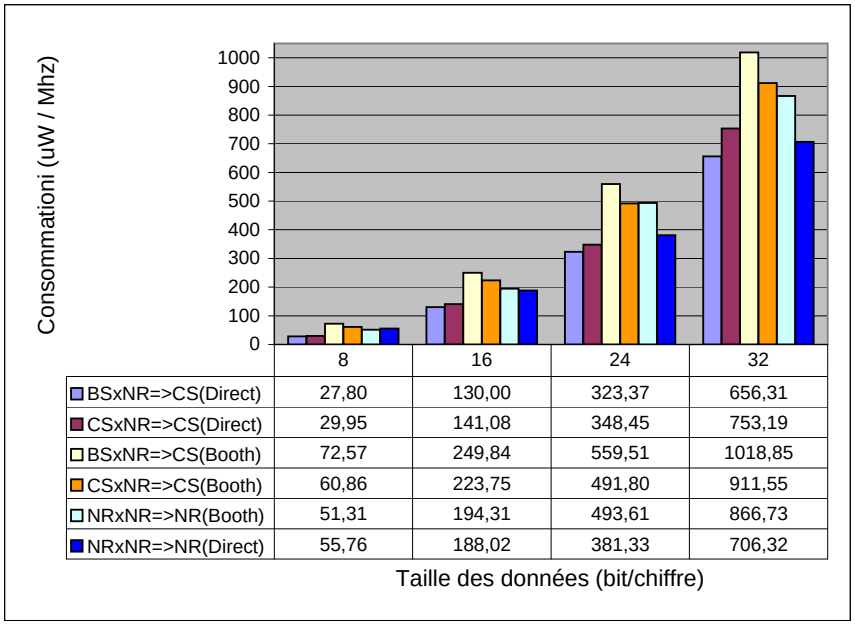
tère principal, l'écart peut-être réduit en utilisant la structure 1 recodage.

2. En prenant le multiplieur de *Booth* classique comme référence, les résultats sont sensiblement les mêmes. Les écarts ne sont pas significatifs.

4.4.4 Consommation

Multiplieurs mixtes : consommation

La figure 4.8 présente la consommation des divers multiplieurs mixtes et classiques, en fonction de la dynamique des entrées. Les valeurs indiquées sont en $\mu W/MHz$.
Les performances des multiplieurs mixtes, se décomposent en deux groupes. Le premier correspond aux multiplieurs *Directs*, le second aux multiplieurs de *Booth*.
Dans le premier cas, les architectures mixtes offrent des performances très intéressantes. Le gain (G_c) évolue avec la dynamique, et est compris entre +50% et -6%. L'inversion se produit pour une valeur comprise entre 24 et 32 bits.
Dans le second cas, les multiplieurs de *Booth* mixtes ont une consommation supérieure à celle de la référence tel que 18% <G_c< 42%. Comme supposé lors du choix de la référence de comparaison, les mauvaises performances de ce second groupe sont liées au décalage temporel entre les deux entrées lors du calcul des produits partiels.
La différence de résultat entre les deux multiplieurs de *Booth* mixtes, illustre parfaitement ce propos. Pour seulement une simple couche d'inverseur, le multiplieur $BS \cdot NR$ consomme



	8	16	24	32
BSxNR=>CS(Direct)	27,80	130,00	323,37	656,31
CSxNR=>CS(Direct)	29,95	141,08	348,45	753,19
BSxNR=>CS(Booth)	72,57	249,84	559,51	1018,85
CSxNR=>CS(Booth)	60,86	223,75	491,80	911,55
NRxNR=>NR(Booth)	51,31	194,31	493,61	866,73
NRxNR=>NR(Direct)	55,76	188,02	381,33	706,32

Figure 4.9 – Multiplieurs redondants : comparaison en consommation

entre +30% et +44% de plus que la référence, alors que l'augmentation de la consommation est seulement de +9 à +29% pour le $CS \cdot NR$ mixte. Ces valeurs sont données respectivement pour 8 bits et 32 bits.

Une remarque : le multiplieur classique de *Booth* a une consommation supérieure à celle du multiplieur classique *Direct* : -8% <G_c< +36%. Cet écart croît avec la dynamique des entrées et, est déjà positif pour une valeur comprise entre 8 et 16 bits. Si la référence est le multiplieur de *Booth* classique, les résultats des multiplieurs mixtes sont alors bien meilleurs.

Multiplieurs redondants : consommation

La figure 4.9 présente la consommation des divers multiplieurs redondants et classiques, en fonction de la dynamique des entrées. Les consommations indiquées sont en $\mu W/MHz$.
Globalement, tous les multiplieurs redondants ont une consommation supérieure à celle de la référence. Dans le cas du multiplieur $CS \cdot CS$, l'augmentation va de -6% à +28%, et croît avec la dynamique. Dans le cas des multiplieurs ayant des entrées notations en Borrow-Save, l'augmentation est plus forte : de +3% à +38%.
Les plus mauvais résultats sont obtenus par le multiplieur $CS \cdot CS$ ayant un seul recodage. L'augmentation de la consommation est comprise entre +16% à +59%. Ces fortes différences sont dues à la même raison que pour les multiplieurs de *Booth* mixtes et classiques : différence

temporelle entre les deux entrées lors de leur prise en compte pour le calcul des produits partiels.

Remarque : Avec le multiplieur de *Booth* classique comme référence, l'augmentation est comprise entre +1% et +4%, pour le multiplieurs $CS \cdot CS$ ayant deux recodages, et entre 12% et 24%, pour le multiplieur ayant un seul recodage, les valeurs étant données respectivement pour 8 bits et 32 bits.

4.4.5 Synthèse

Le premier constat qui ressort de cette étude, est que le multiplieur de *Booth* classique n'est pas aussi avantageux qu'on aurait pu le penser. Comparé au multiplieur classique *Direct*, il n'offre que de légers gains en temps, g_t < 7%, et en surface : G_s < 4%, pour une consommation très rapidement supérieure à celle de la référence : -8% à 8 bits et +3.3% à 16 bits, et croissant rapidement avec la dynamique : +29% à 24 bits et +36% à 32 bits.
Ce constat est aussi valable dans le cadre d'une comparaison entre multiplieurs mixtes *Directs* et multiplieurs de *Booth* mixtes. En fait, c'est pire, car le gain en délai est négatif.
De manière assez inattendue, les multiplieurs classiques *Directs* s'imposent comme étant les architectures les plus intéressantes. Cela est d'autant plus vrai, que la comparaison ne tient pas compte des problèmes suscités par l'algorithme de *Booth*, en notation non signée.

Parallèlement, il apparaît que les multiplieurs mixtes ont des performances supérieures ou au pire équivalentes à celles des multiplieurs classiques *Directs*. Les gains sont significatifs à la fois en délai : $g_t \sim$ 27%, en surface entre -4% et 22%, et en consommation de -6% à 50%. Les deux gains négatifs ne sont pas très importants et sont obtenus pour une dynamique de 32 bits.
L'apport des multiplieurs redondants est moins significatif. Le gain en délai est intéressant : g_t $\sim$ 16%, mais il se fait au détriment de la surface : $\sim$ +18% (2 recodages) ou $\sim$ +14% (1 recodage), et de la consommation : entre -6% et +28% (2 recodages). Gagner en surface semble impossible. Par contre, ces architectures sont exploitables en délai sur toute la dynamique, et en consommation jusqu'à 16 bits. Ces performances moyennes ne sont pas une limite à l'utilisation de l'arithmétique redondante, car ces multiplieurs ont peu d'usages possibles. En effet, dans la quasi-intégralité des applications de traitement du signal, les multiplications sont effectuées avec un terme connu (coefficients des filtres numériques par exemple). Cela implique qu'au moins une des entrées est en notation classique.
Ces opérateurs ne sont pas pour autant inutiles. Comme nous l'avons déjà indiqué dans la sous-section 3.4.2, ces architectures sont exploitables pour de nouvelles fonctionnalités en no-

tations classiques : $(A \pm B)^2$ par exemple (Annexe B).

Dans ce contexte, les performances des multiplieurs mixtes et redondants sont intéressantes. Suivant le critère, les gains sont significatifs ou au pire équivalents à ceux des multiplieurs classiques.

4.5 Conclusion / Bilan

Dans ce chapitre, nous avons présenté les performances des trois opérateurs élémentaires que sont l'addition, la somme et la multiplication, en arithmétique mixte. Les divers résultats obtenus nous ont permis de positionner les opérateurs mixtes et redondants par rapport aux opérateurs classiques.
Comme supposé en début de cette étude sur l'arithmétique mixte, les architectures mixtes et redondantes des additionneurs et des multiplieurs sont sources de gains. L'amélioration du délai est systématique et peut-être très importante. Suivant l'architecture, ce premier résultat est accompagné des diminutions de la surface et/ou de la consommation. Pour ces deux critères aussi, les réductions peuvent être importantes.
En fait, le peu d'usage possible de la multiplication redondante nous amène à dire que les nouveaux additionneurs et multiplieurs proposés, sont systématiquement plus performants que les opérateurs classiques correspondants, et cela quel que soit le critère.
Concernant la somme, les résultats sont plus nuancés. Les délais des sommes redondantes et mixtes seront toujours meilleurs que ceux des sommes classiques. Par contre, les surfaces des sommes redondantes seront toujours les plus importantes. Celles des réalisations mixtes fluctueront entre inférieures et supérieures aux surfaces des sommes classiques. Les consommations suivront les surfaces.
Il est important de noter que, quelle que soit la réalisation (mixte, redondante ou classique), les algorithmes utilisés sont les mêmes. Dans ce contexte la différenciation entre les différentes versions d'une somme est fortement liée à la technologie employée.

L'ensemble de ces résultats a été obtenu pour des opérateurs hors d'un contexte d'utilisation. Pour valider ces résultats et valider l'intérêt des notations redondantes, il est impératif d'étudier l'impact de ces notations et des opérateurs mixtes et redondants associés, sur la conception et sur les performances d'une architecture donnée.

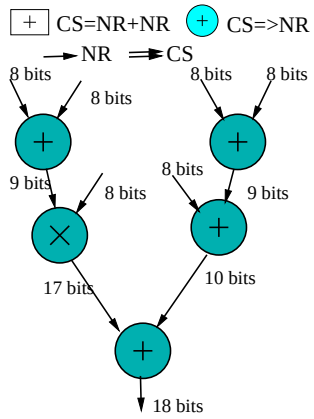
Chapitre 5 Étude des enchaînements d'opérateurs

Ce chapitre a pour objectif de déterminer l'impact de l'utilisation simultanée de plusieurs systèmes de représentations des nombres à l'occasion de la conception de circuits numériques comportant des enchaînements d'opérateurs de diverses natures.
Il s'agit de bien situer le contexte. Une architecture ne se réduisant pas uniquement à une succession d'opérateurs arithmétiques, cas idéal pour la redéfinition des enchaînements, des facteurs autres que le changement de notation vont intervenir. Il va falloir tenir compte des *perturbations* introduites par l'aspect séquentiel des circuits, par les blocs non arithmétiques, par les boucles de rétroactions ou par d'autres phénomènes que nous envisageons tout au long de ce chapitre.
Pour mieux appréhender l'impact et les modifications induites par l'ensemble de ces cas de figures, la méthode adoptée a été de partir du cas idéal puis d'ajouter graduellement les contextes perturbateurs. Un premier découpage est fait autour de deux points essentiels : circuits combinatoires et circuits séquentiels.

Parallèlement à cette étude, nous chercherons à extraire des règles d'optimisations, éventuellement automatisables, liées à l'usage de l'arithmétique mixte dans la conception de chemins de données.

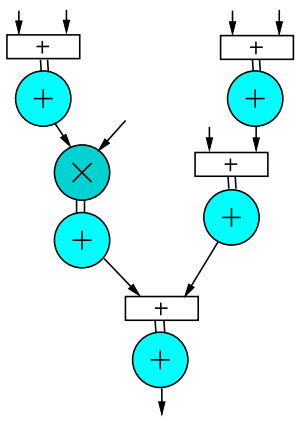
5.1 Enchaînement dans un circuit combinatoire

Dans un premier temps, nous allons considérer que la traversée des registres par les représentations non conventionnelles est impossible. Cette réduction de l'utilisation des systèmes non autonomes revient à découper une architecture en plusieurs arbres combinatoires indépendants dont les entrées/sorties sont en notations classiques. Dans ce contexte, l'étude porte sur les enchaînements d'opérateurs internes aux divers arbres définis.
L'analyse commence par le cas idéal d'une application purement combinatoire composée seulement d'opérations dont les opérateurs associés couvrent tout l'espace mixte. L'analyse s'appuie sur un ensemble d'exemples d'architectures.

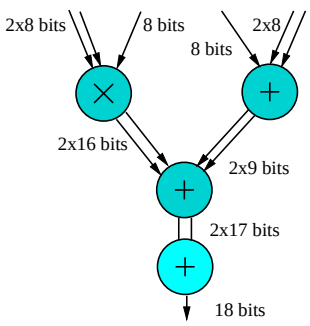


(a) arbre combinatoire initial

nS	μm^2	$\mu W/MHz$
14.00	135056	284
réf.	réf.	réf.



(b) décomposition des opérateurs en opération + conversion



(c) passage en redondant

nS	μm^2	$\mu W/MHz$
11.36	122500	181
-18.9%	-9.3%	-36.3%

Figure 5.1 – Passage en *tout redondant*

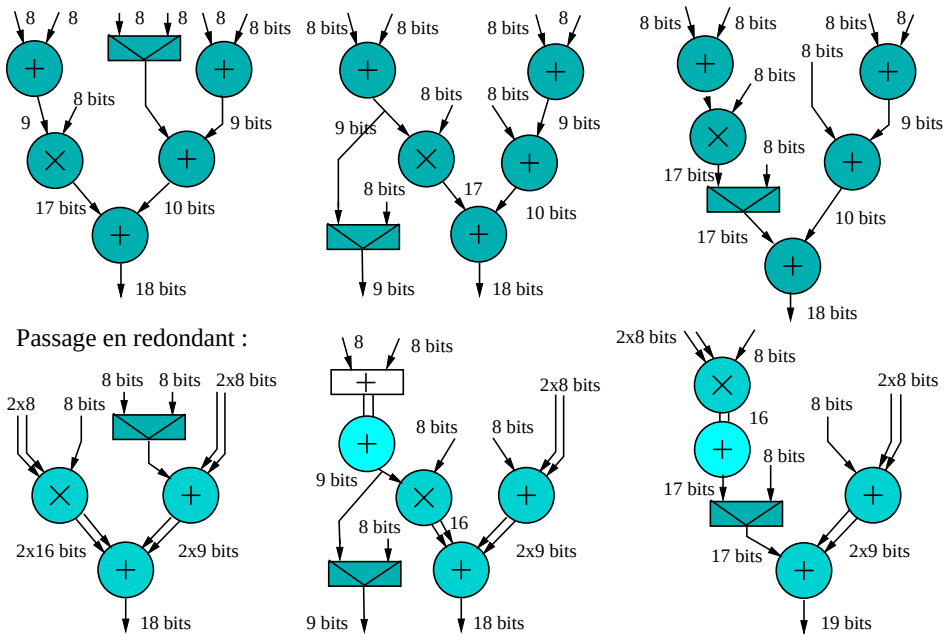
5.1.1 Passage en tout redondant

Connaissant les performances des opérateurs redondants (au pire équivalentes à celles des opérateurs NR), la première idée qui vient à l'esprit est de translater les représentations non redondantes en redondantes. Ces modifications seront accompagnés par la sélection des opérateurs correspondants. Cette méthode est la plus simple que l'on puisse imaginer pour exploiter les propriétés redondantes de l'arithmétique mixte.

Comme le montre la figure 5.1, les changements de notation impliquent le retrait des conversions $CS \rightarrow NR$ présentes en sortie de tous les opérateurs ayant un résultat NR (hors sortie de l'arbre combinatoire). Ils imposent aussi la substitution des opérateurs NR par des opérateurs redondants ou mixtes répondant à la nouvelle définition des entrées.

On note aussi, la disparition de plusieurs opérateurs : toujours des additionneurs. En fait, ce n'est pas l'addition qui disparaît mais la conversion de la sortie. Dans ce cas, le changement de notation étant, nous le rappelons, effectué par un additionneur $NR + NR \rightarrow NR$, l'additionneur $CS = NR + NR$ est un opérateur virtuel et n'a pas d'existence physique. Sur les figures 5.1, il est symbolisé par un plus dans un rectangle.

Ce phénomène peut s'envisager sous un autre angle : L'addition n'a pas disparu mais a été absorbée par les opérateurs utilisant son résultat. Si on reprend l'exemple des figures et tableaux 5.1, la multiplication mixte ne fait plus $CS \cdot NR$ mais $(NR + NR) \cdot NR$ et l'addition mixte $CS + NR$ mais $(NR + NR) + NR$. C'est très intéressant car, cela implique que les opérateurs ayant une ou plus entrées redondantes sont décomposables en opérateurs plus élémentaires ce



(a) Bloc NR sans influence

ns	μm^2	$\mu W/MHz$
11.27	123725	189
0.0%	+1.0 %	+5.0%

(b) Limitation de type donnée

ns	μm^2	$\mu W/MHz$
12.80	126175	222
+12.6%	+3.0%	+23.3%

(c) Limitation de type opérateur

ns	μm^2	$\mu W/MHz$
13.50	137638	209
+18.8%	12.35%	+16.1%

Figure 5.2 – Impact des blocs non redondants

qui pourra servir ultérieurement dans l'optimisation. Le découpage pourra se faire en entrée (absorption) et en sortie (conversion).

Dernier point : la limitation du *tout redondant* aux chaînes combinatoires impose la conversion des résultats en non redondant.

Comme l'illustre les tableaux associés à la figure 5.1, pour l'exemple donné, les améliorations des performances tant en surface qu'en délai et en consommation sont notables. Plus généralement, ces gains seront très variables. Ils dépendront de la taille des données, des opérations utilisées dans l'application, des opérateurs substitués, ainsi que de la nature et la quantité des enchaînements.

Dans tous les cas, les gains apportés par le passage en *tout redondant* seront toujours positifs. Cela est lié aux meilleures performances intrinsèques des opérateurs redondants : chapitre 4.

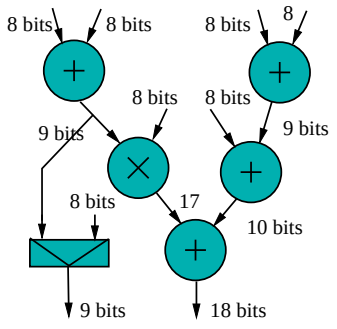
5.1.2 Impact des blocs n'acceptant pas les notations redondantes

L'insertion, dans une chaîne combinatoire, de blocs ne supportant pas les notations redondantes n'est pas un vrai problème. Pour prendre en compte ces blocs, il suffit d'adapter la règle précédente *passage en tout redondant* en *passage en redondant partout où c'est possible*. C'est

d'ailleurs une des raisons à l'introduction de la notion d'arithmétique mixte.
Comme le montre la figure 5.2, où un multiplexeur est inséré à trois endroits différents de l'exemple précédent, l'effet de ces blocs sur les performances du *tout redondant* peut-être très important. Deux types de limitations apparaissent : la limitation de type donnée (cas 5.2.b) et la limitation de type opérateur (cas 5.2.c). Afin de juger de l'influence de ce petit bloc NR sur les performances des architectures mixtes, les pourcentages donnés sont exprimés en fonction des performances de la figure 5.1.c. Nous considérerons que dans le cas 5.2.a le bloc rajouté n'a pas d'influence sur les performances. Par rapport aux autres opérateurs, son impact sur le délai et la surface est négligeable.
Dans les cas 5.2.b et 5.2.c, les performances sont sensiblement altérées. La dégradation de la surface et du délai se traduit par la réapparition d'une conversion $R \rightarrow NR$. Il faut donc chercher à limiter l'impact de ces blocs NR. Éliminer la fonction (multiplexeur) n'étant pas possible nous allons chercher à supprimer la conversion resurgis.

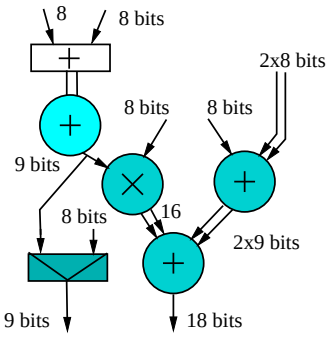
Double représentation

Pour limiter l'effet des blocs NR, l'idée la plus simple consiste à distribuer le signal à la fois en redondant et à la fois en non redondant. Ainsi, il est possible de répondre aux impératifs d'optimisations tout en ne modifiant que très légèrement l'architecture.



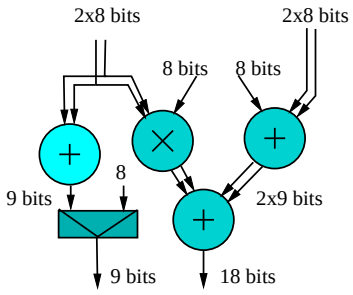
(a) Bloc NR sans influence

ns	μm^2	$\mu W/MHz$
14.22	136342	298
indicatif	indicatif	indicatif



(b) Limitation de type donnée

ns	μm^2	$\mu W/MHz$
12.80	126175	222
+12.6%	+3.0%	+23.3%



(c) Dédoublement de la donnée

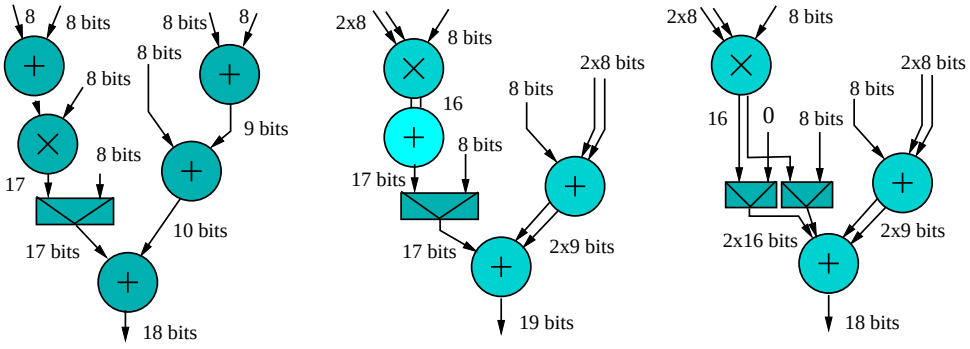
ns	μm^2	$\mu W/MHz$
11.42	134700	198
+0.5%	+10%	+10%

Figure 5.3 – Impact des blocs non redondants : double représentation

Si cette proposition résout efficacement le cas 5.2.b repris figure 5.3, elle n'a aucun impact sur le cas 5.2.c (Les pourcentages sont toujours donnés par rapport au cas figure 5.1.c). Cette solution n'est pas globale. C'est très gênant car le dernier cas 5.2.c correspond à la situation la plus fréquemment rencontrée dans les applications visées.

Doublement de l'opérateur

La seconde idée est, non plus de proposer une double notation de la donnée, mais de dédoubler le bloc pour répondre au passage en redondant de la donnée : figures 5.4. Cette modification a deux implications : D'abord sémantiquement le redondant substitué au NR initial est séparable en deux NR ayant les mêmes caractéristiques (hypothèse retenue lors de la définition de l'arithmétique mixte) et deuxièmement que le traitement effectué par le bloc non mixte puisse être appliqué séparément aux deux composantes du redondant.



(a) Bloc NR sans influence

ns	μm^2	$\mu W/MHz$
14.15	136342	287
indicatif	indicatif	indicatif

(b) Limitation de type opérateur

ns	μm^2	$\mu W/MHz$
13.50	137638	209
+18.8%	12.35%	+16.1%

(c) Doublement du bloc

ns	μm^2	$\mu W/MHz$
12.14	126175	204
+6.8%	+3.00%	+13.3%

Figure 5.4 – Impact des blocs non redondants : doublement du bloc NR

Dans notre exemple, le doublement du multiplexeur est profitable à la fois en surface, en consommation et en délai. Plus généralement, si le gain en délai est assuré, concernant la surface, cette seconde méthode a un prix non nul qu'il faudra ramener à une fonction de coût entre le doublement du bloc et la disparition de la conversion $R \rightarrow NR$.

Quelques remarques :

1. Le doublement de l'opérateur ne sert à rien si la sortie du bloc n'est pas exploitée en redondant. Dans ce cas il y a juste déplacement de la conversion.
2. Le doublement d'un bloc permet l'élargissement de l'ensemble des opérateurs définis dans tout l'espace mixte, aux blocs ne transformant pas et ne recombinant pas les bits de ses entrées. En clair, le doublement peut s'appliquer à toutes les structures effectuant des sélections ou des transferts de données au niveau le plus élémentaire (multiplexeurs, cellules 3 états,...) mais aussi aux fonctions spécialisées qui respectent l'intégrité des entrées ou une uniformité de traitement (décaleurs (*shifters*), complément à un, etc.).

3. Les opérateurs logiques élémentaires, bit à bit, les opérateurs arithmétiques indéfinis en mixte, les tests, etc., sont exclus du doublement. Tous ces opérateurs effectuent des traitements hétérogènes au niveau élémentaire.
4. Le point le plus délicat ici, est la déclinaison du bloc en mixte car dans ce contexte, les entrées du bloc n'ont pas la même notation. Toutefois le doublement impliquant des sorties en redondants, il suffira juste de modifier les entrées NR en R en ajoutant un terme NR (cas 5.4.c des figures précédentes). Ce terme sera fonction de la représentation redondante de sortie CS ou BS. Dans les deux cas, il peut s'envisager comme nul.

Élaboration d'un bloc mixte équivalent

Proposer des variantes mixtes et redondantes à un bloc NR ne fait évidemment pas partie de l'optimisation arithmétique telle qu'elle est envisagée ici (automatisation). Cette solution est la plus lourde à mettre en œuvre. Toutefois effectuer un tel travail constitue une alternative qu'il est nécessaire d'évoquer. Celui-ci consiste à développer les architectures mixtes et redondantes équivalentes au bloc non redondant de départ.
Pour que ces nouveaux blocs soient viables, il faut que leurs performances soient meilleures ou similaires à celles du bloc NR initial. L'accent sera mis en priorité sur le temps de propagation et la consommation, puis dans un deuxième temps sur la surface. Dans le cas où les propriétés sont proches, les gains ne viendront pas des nouvelles structures proposées mais de l'élargissement des possibilités et des contextes d'utilisation de l'arithmétique mixte.
Ce travail a été fait pour les opérateurs arithmétiques élémentaires : addition, somme et multiplication, permettant ainsi l'usage de l'arithmétique mixte.

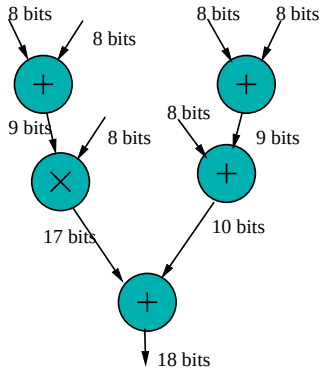
Alors ?

Malgré tous ces mécanismes qui se traduisent par l'élargissement de l'espace mixte, il reste toujours des cas de figures où le passage en redondant reste impossible (opérateurs logiques, etc.) ou inutile (doublement d'un bloc ayant une sortie NR par exemple). Le *passage en tout redondant* n'est pas réalisable. La règle reste donc *passage en redondant partout où c'est possible*.

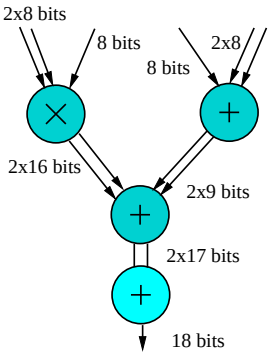
5.1.3 Intégration de l'aspect temporel des enchaînements d'opérateurs

Malgré le gain substantiel apporté par le passage en *redondant dès que possible* il apparaît que l'optimisation n'est pas maximale en terme délai : figure 5.5. Il n'y a là rien d'étonnant vue que jusqu'ici, nous n'avons pas pris en compte la dimension temporelle du chaînage des opérateurs et des opérateurs eux-mêmes.

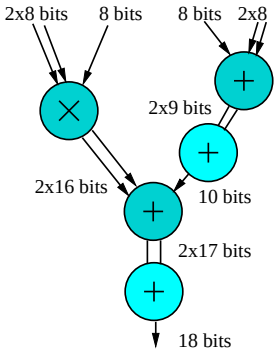
Le troisième ensemble tableau figure 5.5.c montre que l'insertion *intelligente* d'un convertisseur $CS \rightarrow NR$ est à l'origine d'un meilleur temps de propagation.
Le principe utilisé, consiste à profiter du déséquilibre des divers chemins de propagations pour effectuer des retours en NR (conversion $R \rightarrow NR$). L'objectif est de réduire le nombre d'entrées redondantes des opérateurs de la chaîne longue. Nous rappelons que plus un opérateur redondant a d'entrées non redondantes plus sa surface est petite et plus son temps de propagation est faible, ses sorties étant toujours exprimées avec les mêmes notations.



(a) arbre combinatoire initial



(b) Passage en redondant dès que possible



(c) retour en NR conditionné par l'analyse

ns	μm^2	$\mu W/MHz$
14.00	135056	284
réf.	réf.	réf.

ns	μm^2	$\mu W/MHz$
11.36	122500	181
-18.9%	-9.3%	-36.3%

ns	μm^2	$\mu W/MHz$
10.73	126175	183
-23.4%	-6.6%	-35.5%

Figure 5.5 – Aspect temporel des enchaînements

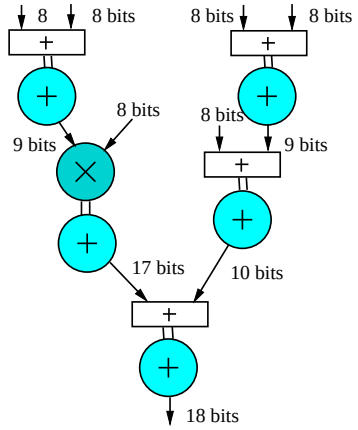
Sur la figure 5.5.c l'exploitation du déséquilibre des chemins se traduit par le remplacement de l'additionneur de sortie $NR = CS + CS$ par un additionneur $NR = CS + NR$. Ce second est plus rapide et plus petit que le premier. Nous parlerons ici de *retour en notation non redondante*. Le gain en délai est égal à la traversée d'un FA. Par contre la surface augmente à cause du rajout de la conversion.
Le principal défaut de cette technique est qu'elle repose sur l'analyse des temps de propagation des divers chemins combinatoires du circuit. De plus cette analyse devra être réitérée après chaque modification de l'architecture afin de converger vers une solution optimale. Le coût potentiel en temps d'un tel mécanisme est très élevé, d'autant qu'*a priori*, la rétroaction passe par un placement/routage.

Quelques mots sur l'analyse de temps

L'analyse de temps peut se faire de deux façons. Partant de l'architecture initiale, on effectue un passage en redondant en fonction de l'analyse ou on passe en tout redondant puis on effectue

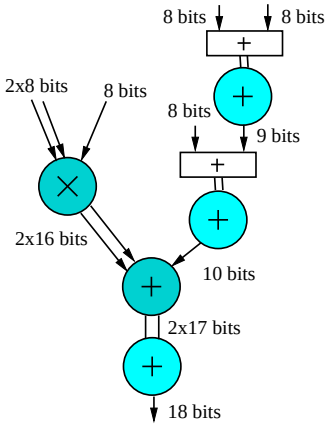
un retour en NR via l'analyse. Comme l'illustre les figures 5.6, les architectures obtenues ne sont pas identiques. Les temps de propagation sont égaux mais les surfaces et consommations respectives diffèrent. Quelle solution adopter alors ?

Sachant que la surface des opérateurs purement redondants est inférieure ou sensiblement égale à celle des opérateurs NR et que la redéfinition locale d'enchaînement apporte toujours un gain en surface, le passage en tout redondant suivie d'un ajustement par l'analyse temporelle, apparaît comme l'idée la plus payante puisque à délai égal la surface est inférieure.



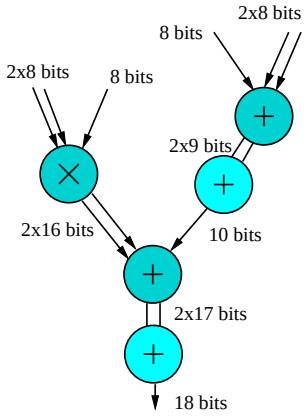
(a) arbre combinatoire initial

ns	μm^2	$\mu W/MHz$
14.00	135056	284
réf.	réf.	réf.



(b) Passage conditionné en CS par l'analyse

ns	μm^2	$\mu W/MHz$
10.74	130464	206
-23.4%	-3.5%	-27.5%



(c) Redondant puis retour en NR conditionné par l'analyse

ns	μm^2	$\mu W/MHz$
10.73	126175	183
-23.4%	-6.6%	-35.5%

Figure 5.6 – Analyse de temps

Pour permettre le retour en notation classique, l'analyse doit passer un à un tous les opérateurs de l'arbre combinatoire considéré, et détecter ceux dont les entrées ont un écart de disponibilité de l'ordre du temps de propagation d'une conversion $R \rightarrow NR$. Cette valeur n'est pas fixe et dépend de la dynamique des entrées. Une fois ces opérateurs spécifiés, on va regarder s'ils sont connectés au chemin critique de l'arbre mais sans en faire partie. Si c'est le cas, un retour en NR est effectué. En procédant ainsi, le nombre de retour en NR est limité.

Remarques :

1. Malgré la diminution de la surface de l'additionneur de sortie, la surface globale augmente car la conversion n'est pas gratuite (additionneur $NR = NR + NR$). Ce sera le cas général.
2. Appliquer le *redondant partout où c'est possible* avant l'analyse, apparaît comme la solution donnant la plus faible surface dans un arbre combinatoire, pour le meilleur délai. Dans ces conditions, l'analyse de temps portant sur l'intégralité des arbres combinatoires, le nombre

de retour en NR est encore plus limité. De même le nombre d'itérations nécessaires à l'étude des délais est réduit.

3. Si l'on souhaite privilégier la surface, l'analyse de temps ne s'impose pas.

5.1.4 Enchaînements combinatoires : Synthèse

Il ressort de cette étude que la redéfinition des enchaînements combinatoires est potentiellement source de gain à la fois en surface et en délai. L'amélioration des performances apportée sera positive ou au pire nulle.
L'optimisation est décomposable en deux phases. La première effectue un changement de notation systématique partout où c'est possible. Cette première étape offre la plus faible surface et une diminution du temps de propagation. La seconde s'appuie sur l'analyse des temps de propagation pour introduire des retours ciblés en notation non redondante. Cette deuxième phase, itérative, permet d'abaisser un peu plus le temps de propagation mais fait croître la surface par l'insertion de conversions $R \rightarrow NR$.

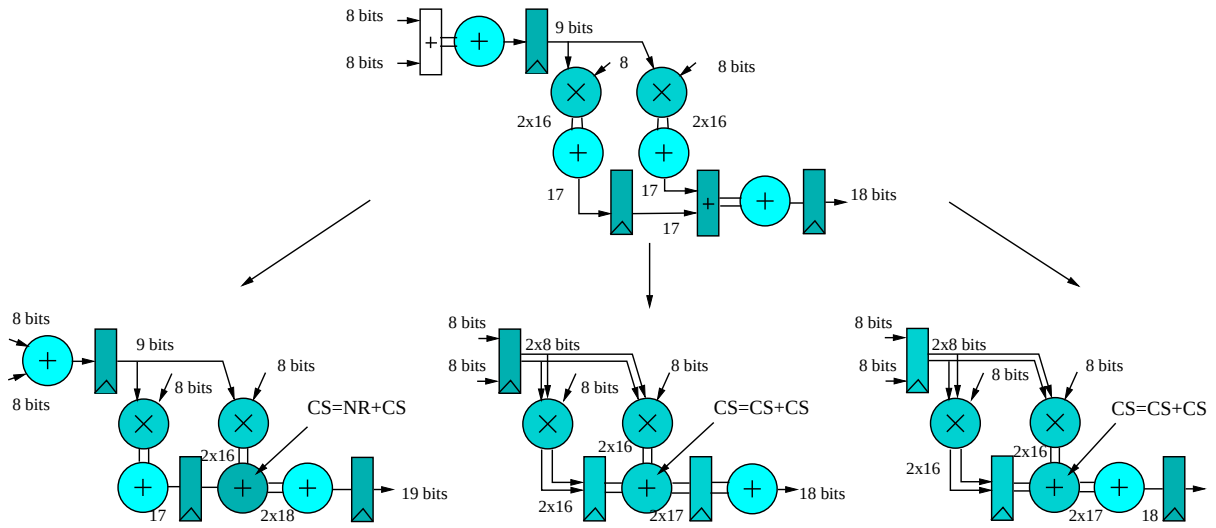
5.2 Enchaînement dans un circuit séquentiel

Jusqu'ici, nous nous sommes volontairement limités aux enchaînements combinatoires. Il nous faut maintenant étudier le passage des notations redondantes à travers les points mémorisants. Nous utiliserons indifféremment registre ou point mémorisant pour exprimer la même fonctionnalité. L'objectif est toujours l'amélioration des performances.

Comme pour les enchaînements combinatoires, l'étude proposée s'appuie sur des exemples d'architectures ; les notations initiales des entrées/sorties du circuit seront respectées.

5.2.1 Passage d'un point mémorisant

Bien que les registres répondent à la définition des blocs pouvant être doublés, les problèmes soulevés ne sont pas tout à fait les mêmes. La différence tient essentiellement à la propriété de barrière temporelle des points mémorisants, qui fait d'eux à la fois le début et la fin des chaînes combinatoires.
La traversée d'un registre par un redondant implique la diminution du délai de l'arbre combinatoire rattaché à son entrée (disparition d'une conversion $R \rightarrow NR$). Elle implique aussi l'augmentation potentielle du délai de l'arbre câblé sur la sortie du registre car les opérateurs chaînés sur ce point voient leur nombre d'entrées redondantes augmenter. Afin d'illustrer ce propos, un exemple est donné figure 5.7.



(a) Redéfinition des enchaînements combinatoires uniquement

(b) Sorties combinatoires

(c) Sorties sur registres

ns	μm^2	$\mu W/MHz$
11.3	208617	199
réf.	réf.	réf.

ns	μm^2	$\mu W/MHz$
9.65	211802	159
-14.6%	+5.7%	-20.1%

ns	μm^2	$\mu W/MHz$
12.7	207025	214
+12.4%	+0.08%	+7.5%

Figure 5.7 – Redéfinition des enchaînements des registres isolés

Dans le premier cas correspondant au *tout redondant* (figure 5.7.b), le temps de propagation est plus faible car toutes les entrées/sorties de tous les registres, ont été redéfinies. Ce meilleur résultat est lié à l'uniformité du traitement. Celle-ci permet, dans toutes les chaînes combinatoires, de compenser l'augmentation du nombre d'entrées redondantes des blocs rattachés à la sortie d'un registre, par la disparition des conversions en sortie des arbres combinatoires. Seule la sortie du circuit voit son délai augmenter. Comme au préalable elle se résumait à une simple connexion, cela n'a pas d'effet sur le temps de propagation. Toutefois la sortie est maintenant sous contrainte de temps ce qui peut-être défavorable ou irréaliste suivant l'utilisation de l'architecture.

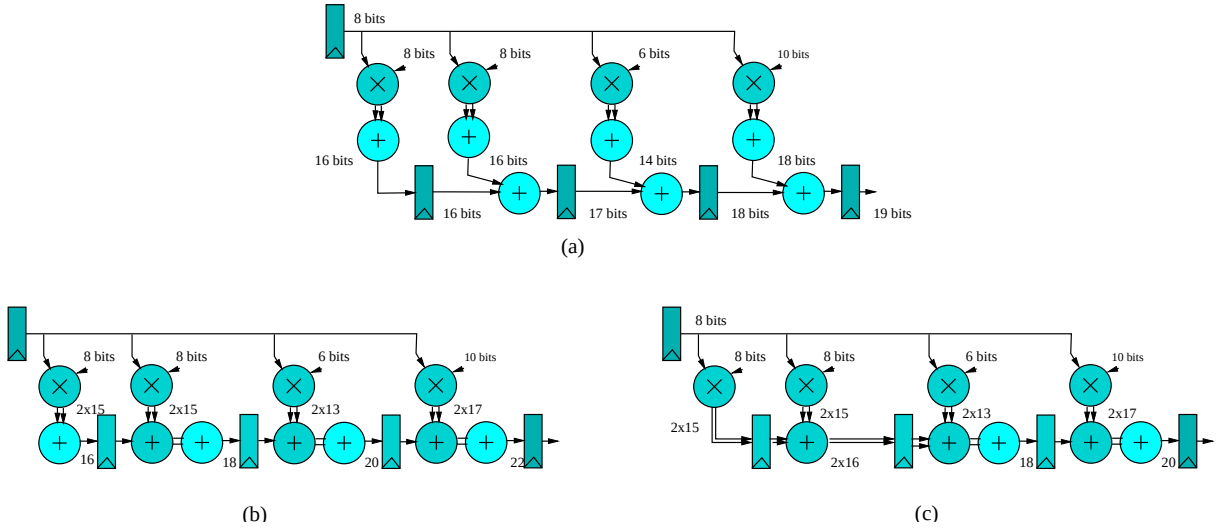
Dans le second cas (figure 5.7.c), la sortie sur registre est respectée. Le passage en redondant ne s'effectue pas partout. Cela se traduit par l'augmentation du temps de propagation (et de la consommation) de l'arbre combinatoire connecté sur l'entrée du registre de sortie. Le passage en redondant devient alors improductif sur le plan du délai et de la consommation.

Ce second cas illustre parfaitement le défaut majeur du passage des redondants à travers les registres où, contrairement aux enchaînements combinatoires, la redéfinition des enchaînements ne garantit pas une amélioration toujours positive du temps de propagation. Pour cet exemple, le plus faible délai sera obtenu par un passage en R uniquement combinatoire car tous les

chemins sont liés à la chaîne longue.
Plus généralement l'obligation de retour en NR dans une chaîne combinatoire, peut-être dû à l'interdiction des sorties combinatoires, à un bloc non mixte ou à un rebouclage comme nous le verrons plus loin. Globalement cette obligation empêche la disparition de certaines conversions et implique des perturbations dans les gains attendus. Le passage des notations redondantes à travers les registres est rendu inefficace car il n'y a pas de compensation des modifications en entrée de la chaîne combinatoire par des modifications en sortie de la chaîne.
Le passage de registre est-il alors à proscrire ? Non, car l'augmentation du délai d'une partie d'une chaîne combinatoire n'implique pas forcément la dégradation du temps de propagation de cette chaîne et encore moins celui de l'architecture. Par contre, il faut conditionner le passage des points mémorisants isolés par une analyse de temps.
Dernier point : Le cas 5.7.b conduit à la plus petite surface car à chaque passage de registre correspond la disparition d'une conversion. C'est vrai dans la mesure où le nombre d'opérateurs en sortie des divers registres est faible, cas le plus rencontré. Par contre le délai n'est pas minimum. La restauration de la conversion en entrée du premier registre, permet d'abaisser un peu plus le temps de propagation. Toutefois, le retour en NR nécessite une analyse de temps.

5.2.2 Analyse temporelle / Aspect temporel

Contrairement aux circuits combinatoires, le passage des notations redondantes à travers les registres, ne garantit pas une modification des performances au pire nulle. Pour assurer ce minimum, une analyse de temps s'impose. Comme précédemment, le principe consiste à exploiter les déséquilibres entre les divers chemins de propagations pour obtenir de meilleures performances.
Deux questions se posent : quand et comment l'analyse sera appliquée ? La figure 5.8 présente deux possibilités. Le premier cas 5.8.b correspond à un passage en redondant conditionné par l'analyse temporelle, soit, la redéfinition uniquement des chemins critiques. Le second cas 5.8.c correspond lui au passage en redondant suivi d'un retour en notation classique conditionné par l'analyse de temps. Tous les chemins sont étudiés.
L'architecture proposée est un filtre numérique à réponse impulsionnelle finie. Sa sortie s'effectue sur registre. Les deux solutions proposées apportent un gain en délai, en consommation et en surface. Pour le même temps de propagation, les surface et consommation en 5.8.c sont inférieures à celles en 5.8.b. Les différences sont dues à la disparition de plusieurs conversions.
L'exemple proposé est extrême au sens où le passage de registre conditionné par l'analyse n'introduit aucune modification de l'architecture par rapport au passage en redondant combinatoire. Toutefois, il est très intéressant car il correspond à un circuit réel.



(a) Architecture initiale

ns	μm^2	$\mu W/MHz$
12.86	396900	426
réf.	réf.	réf.

(b) Passage en CS conditionné

ns	μm^2	$\mu W/MHz$
11.7	381587	373
-9.0%	-4.0%	-12.5%

(c) Passage en CS puis Analyse

ns	μm^2	$\mu W/MHz$
11.6	360272	328
-9.8%	-9.2%	-23.0%

Figure 5.8 – Enchaînement séquentiel : analyse temporelle

Ce résultat est généralisable. Le choix du *tout redondant puis analyse* s'impose car il offre le plus grand nombre de disparitions de conversions, soit, la surface la plus faible pour le même temps de propagation. On rejoint ici, l'étude effectuée sur les enchaînements combinatoires où l'idée du passage systématique en redondant suivi d'un retour conditionné en NR est la plus payante.

Le *quand* étant défini, il faut maintenant déterminer le *comment*. Le passage en redondant étant systématique, l'action de l'analyse consiste à déterminer les points où un retour en notation NR est utile à l'amélioration des performances. Cette fois ce ne sont pas les additionneurs qui sont considérés mais les points mémorisants. L'idée est de repérer les registres inclus dans la chaîne critique et dont les entrée/sortie sont en notation redondante, afin d'effectuer un retour en NR. Ce dernier se fera via la réintroduction d'une conversion en entrée des registres sélectionnés.

Remarques :

– Dans le pire cas le retour en NR va nous ramener à l'architecture obtenue après le passage en redondant combinatoire. Les performances seront les mêmes.
– Comme pour les enchaînements combinatoires, le passage systématique en redondant sans analyse offre la plus petite surface.

– La transformation des enchaînements séquentiels sera toujours positive grâce à la correction apportée par l'analyse des temps de propagations.
– La plupart des circuits ayant plusieurs chemins, la chaîne critique peut sauter d'un premier à un autre. Le mécanisme décrit plus haut, est donc à itérer jusqu'à convergence.
– Dans le cas ou il y a conservation de conversion, comme nous allons le voir avec les boucles et les chaînes de registres, le retour de l'entrée d'un point mémorisant en notation NR refait glisser la conversion mais cette fois, de sa sortie vers son entrée.

5.2.3 Cas particulier du passage d'une chaîne de registres

Comme précédemment, le passage en redondant n'est pas un problème en soit. Par contre, si le coût surface, du doublement des registres isolés, était inférieur au coût surface des conversions économisées, pour les chaînes de registres ce ne sera pas toujours vrai car le nombre de points mémorisants est variable et la conversion unique.
Pire, la modification des notations de sorties des registres, implique l'augmentation du nombre d'entrées redondantes sur les opérateurs rattachés à ces points mémorisants, ce qui équivaut à un accroissement encore plus sensible de la surface et une détérioration potentielle du délai.
Heureusement il existe une solution simple qui tire parti de l'enchaînement direct des points mémorisants. Il suffit de décomposer la chaîne en deux blocs : premier registre et reste de la chaîne, et de faire glisser la conversion de l'entrée vers la sortie du premier point mémorisant en séparant les deux blocs comme l'illustre la figure 5.9.

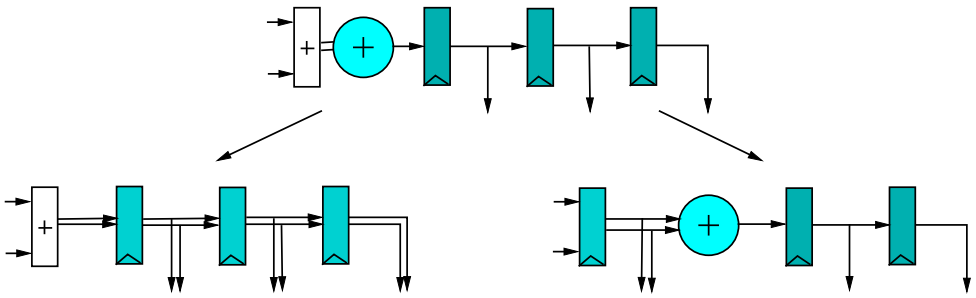


Figure 5.9 – traitement d'une chaîne de registres

Le prix en surface du passage en redondant va se limiter au doublement du premier registre et à l'augmentation du nombre d'entrées redondantes des opérateurs utilisant la sortie du premier registre. Il y a conservation de la conversion.
Attention, séparer le premier registre revient à l'isoler ! On se ramène donc aux problèmes soulevés précédemment dans l'étude du *Passage d'un point mémorisant isolé*.

Quelques remarques :

- Pour un petit nombre de registres chaînés, on est en droit de se demander si le passage en redondant de tous les registres n'est pas plus performant. En terme de délai le problème est identique voir pire car toutes les chaînes combinatoires utilisant les sorties des points mémorisants voient augmenter le nombre de leur entrées redondantes et potentiellement leur temps de propagation (Les sorties ne sont pas forcement compensées).
- En terme de surface, il n'est pas possible de dégager une règle globale car le doublement des registres et le coût du passage en redondant des entrées des opérateurs rattachés aux sorties des registres comparés à un convertisseur, n'est pas constant. La frontière reste donc très floue.
- La solution du glissement de la conversion séparant la chaîne en deux apparaît comme la plus intéressante car le temps de propagation est le meilleur que l'on puisse obtenir et l'augmentation de la surface est faible et est contrôlable. Toutefois, le glissement ne règle pas la question du premier registre.

5.2.4 Problèmes inhérents aux rebouclages

Limitations

En première analyse on retrouve le problème classique des données qui ne peuvent pas être définies sur l'infini, la réalisation physique étant impossible. Cette complication connue, nécessite l'ajustement de la taille de quelques données de la boucle.
Si l'on souhaite définir le moins de point possible il faut chercher ce que nous appelons les *sorties primaires*. Celles-ci correspondent au nombre minimum de points permettant d'établir la dynamique de toute une boucle. *Sorties primaires* car ces points sont toujours des sorties et sont systématiquement à l'origine d'une boucle. Il n'y a là rien de nouveau. La figure 5.10 donne un exemple de sorties primaires dans une architecture aux boucles enchevêtrées.

La vraie question qui se pose est peut-on redéfinir les notations des sorties primaires ?

On distingue deux types de fonctionnement des boucles. Soit les sorties primaires ont une borne supérieure et dans ce cas les opérateurs de sorties ont un comportement classique. Soit les sorties primaires sont cycliques et dans ce cas les résultats des opérations de sorties sont modulo 2^N, et génèrent régulièrement un dépassement de capacité (*overflow*).
Dans le premier cas, le passage en redondant ne pose aucun problème car, les opérateurs définissant les sorties primaires ont le même fonctionnement que tous les autres opérateurs. La

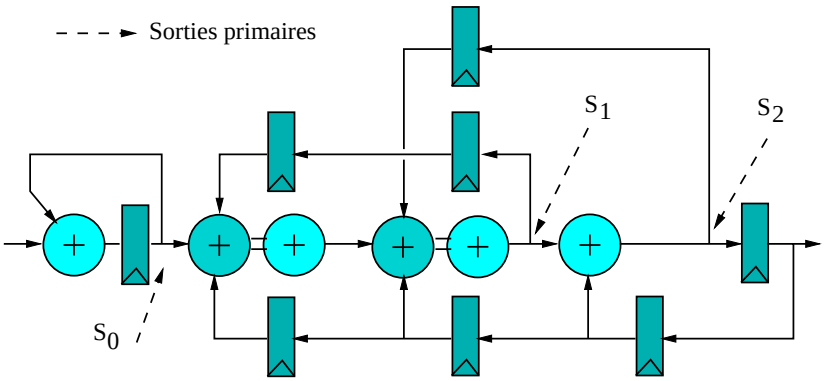


Figure 5.10 – Boucles : illustration des sorties primaires

redéfinition des enchaînements répondra aux même critères que pour le passage d'un registre isolé.

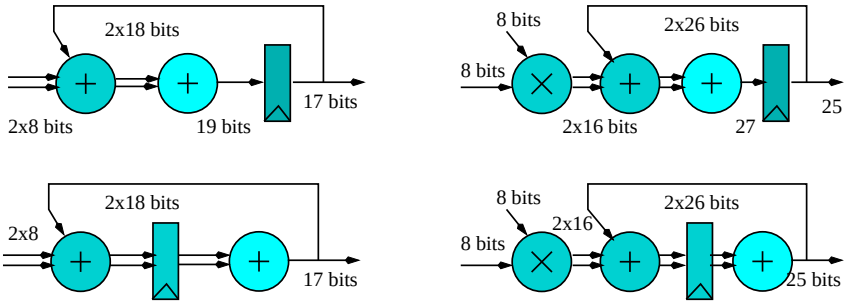
Dans le second cas, la difficulté ne vient pas de la limitation de la dynamique en elle-même mais des qualités requises pour les sorties primaires. Les notations redondantes CS et BS étant à retenue conservée, elles ne permettent pas de détecter les dépassements de capacité sans effectuer une conversion en NR. L'exploitation directe en redondant, devient alors impossible. Il faudra convertir chaque sortie primaire R en NR pour assurer le bon fonctionnement de l'architecture.

Passage des registres

Malgré le retour obligatoire en notation NR imposé sur les entrées primaires, il est intéressant de voir comment vont évoluer les performances si un passage de registre est effectué. La conversion étant *irréductible*, l'opération consiste à la faire glisser de l'entrée sur la sortie du point mémorisant.

La figure 5.11, présente deux exemples de sorties primaires sur registres. Dans le premier cas 5.11.a, l'application du passage n'apporte aucune amélioration des performances. Au contraire, la surface augmente avec le doublement du point mémorisant. Le glissement de registre est inefficace. Dans le second cas 5.11.b, le glissement fait sortir la conversion de la chaîne longue et le délai est nettement amélioré. Parallèlement, la surface augmente.

Dans les deux cas, l'arbre attaché au registre de sortie, voit une de ses entrées fortement modifiée et potentiellement augmenter son temps de propagation. Les problèmes soulevés recoupent ceux rencontrés dans le cas du passage d'un point mémorisant étudié dans la sous-section 5.2.1 : la traversée de registre par une notation redondante est possible, mais implique une analyse des temps de propagations afin de garantir une optimisation positive.



(a) Passage de registre improductif

(b) Passage de registre productif

ns	μm^2	$\mu W/MHz$
5.37	45570	27.6
5.27 (-1.8%)	54941 (+20.5%)	27.7 (0.0%)

ns	μm^2	$\mu W/MHz$
11.46	114047	92
7.85 (-31.5%)	135056 (+18.4%)	72 (-21.7%)

Figure 5.11 – Boucles : Passage en redondant

Il est à noter que la conversion doit s'effectuer avant la distribution de la sortie primaire. Dans le cas contraire, il faudra autant de conversions que de branches utilisant la sortie primaire.

5.2.5 Enchaînements séquentiels : Synthèse

La propriété de barrière temporelle des registres fait que le passage des notations redondantes à travers les points mémorisants n'est pas systématiquement source d'amélioration des performances.
Ce problème est amplifié par l'impossibilité en notations redondantes de gérer la propriété cyclique d'un résultat (boucle), par l'aspect *sortie non combinatoire* fréquemment retrouvé dans les circuits et plus généralement par le respect des notations initiales (NR) des E/S du circuit.
Le cheminement conduisant vers l'abaissement du délai et de la surface est le même qu'avec les enchaînements combinatoires : *passage en redondant partout où c'est possible* suivi d'un retour en NR conditionné par une analyse temporelle. Ce processus est itératif.

Quelques remarques :

- Que les sorties soient combinatoires ou sur registres, le déroulement de la redéfinition des notations en entrée / sortie des points mémorisants restera le même.
- Le passage partout où c'est possible en notation redondante sans analyse (mais avec disparition de conversion), amène à la plus petite surface.
- Les chaînes de registres et les boucles sont réductibles aux problèmes de passage d'un registre isolé. Par contre, dans ces deux cas et pour des raisons différentes, la conversion $R \rightarrow NR$ en entrée du point mémorisant considéré, n'est pas supprimée.

5.3 Notations redondantes partielles

Un rapide examen des diverses architectures des opérateurs mixtes et redondants proposées dans le chapitre 3, fait apparaître que les premiers et derniers chiffres composant la sortie redondante de ces opérateurs, sont fréquemment constitués d'un unique bit. Jusqu'ici et afin d'assurer la cohérence entre notations redondantes et non redondantes, les bits indéfinis de ces chiffres, étaient fixés à 1, à 0 ou prenaient comme valeur la recopie d'un signe.
Plus généralement, il n'est pas impossible qu'un nombre représenté en CS ou en BS soit localement (au niveau chiffre) non redondant. Nous parlerons de notation redondante partielle. Cette section est consacrée à l'étude de cette particularité.

5.3.1 Usage local des notations redondantes

Afin d'illustrer l'intérêt de considérer les chiffres définis localement comme NR des nombres redondants, la figure 5.12 reprend les architectures de recodages $R \rightarrow R$ figures 3.20 et 3.21.

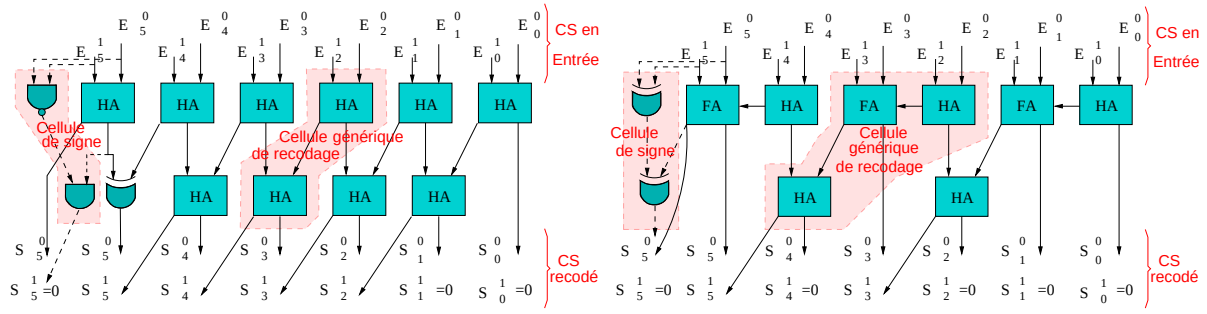


Figure 5.12 – Multiplication redondante : architectures des recodages $R \rightarrow R$

Les propriétés de ces deux architectures (surface, délai et fonctionnalité) sont identiques. Leurs sorties sont toutes deux en Carry-Save. D'un point de vue extérieur, la différence entre ces deux architectures ne tient qu'à la quantité de chiffres partiellement définis (partiellement constants), soient, respectivement deux et plus de la moitié. Propagé dans le calcul des produits partiels du multiplieur redondant, cette différence a permis de réduire de plus de 12% la surface de l'opérateur complet, sans altérer le temps de propagation et la consommation. Un second exemple simple est celui des barrières de registres où chaque bit fixé se traduit par l'économie d'un registre.
Ces résultats sont généralisables. La propagation des bits indéfinis forcés à une valeur constante, permet de dégrader l'architecture initiale. Cette opération va se traduire par une réduction du matériel utilisé et potentiellement par une réduction du temps de propagation et de la consommation.

La règle *plus un opérateur a d'entrées non redondantes pour une même notation de sortie, plus il est performant*, énoncée précédemment, devient *plus un opérateur a d'entrées non redondantes ou partiellement redondantes pour une même notation de sortie, plus il est performant*.
Toutefois, en ce qui concerne les opérateurs proposés dans le chapitre 3, le nombre de chiffres partiellement définis est faible et leur quantité n'augmente pas avec la dynamique des données. *A priori* les gains potentiels restent marginaux.

5.3.2 Retour partiel en NR

Associée à la notion de notation redondante partielle, il est intéressant de définir la notion de retour partiel en notation classique. Effectuer cette opération peut sembler basique : application du signal redondant considéré en entrée d'un ou de plusieurs additionneurs classiques. Elle ne l'est pas, car le recours à des opérateurs classiques induit l'augmentation d'un chiffre de la dynamique du signal considéré. Aussi, un retour en NR ne va pas être effectuer directement sur les signaux redondants, mais être introduit dans les opérateurs existants. Dans ces conditions, la seconde contrainte à observer est le respect des performances générales des opérateurs modifiés, notamment le temps de propagation.

Principe

Typiquement le retour partiel en notation classique constitue une addition. Comme la somme (section3.3) représente le cas général de l'addition, nous allons nous intéresser de nouveau à cet opérateur et à ces constituants élémentaires HA, FA et $4/2$ (figure 5.13).

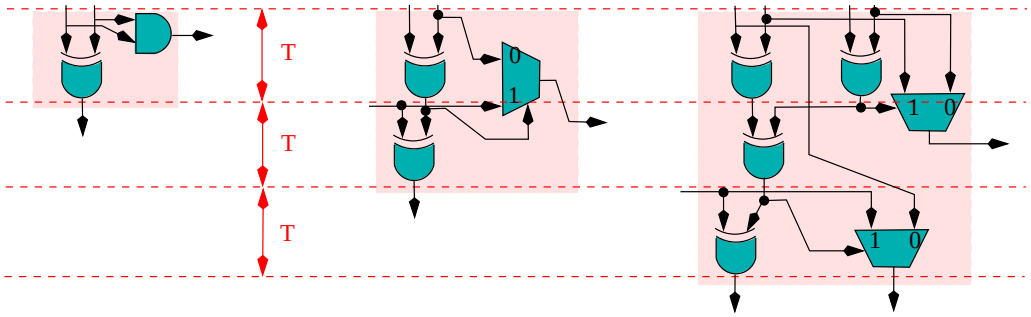


Figure 5.13 – Délai respectif des cellules HA, FA et $4/2$

Notre attention porte plus précisément sur les fourchettes de temps entre la prise en compte d'une entrée ou la génération d'une sortie, et le temps de propagation global de la cellule considérée.

Trois constats :

1. d'abord que pour les cellules FA et $4/2$, une des entrées peut arriver avec un retard allant respectivement jusqu'à la moitié et les des deux tiers du temps de propagation total de la cellule. Nous parlerons d'entrées tardives.
2. ensuite, que les sorties dîtes de *retenues* sont disponibles avant les sorties dîtes de *sommes*. Nous parlerons de sorties précoces.
3. enfin que si l'on considère les cellules deux à deux de la plus petite (HA) vers la plus grosse ($4/2$), toutes les sorties des plus petites cellules peuvent être absorbées par les cellules les plus grosses, $HA \rightarrow FA, 4/2$ et $FA \rightarrow 4/2$, sans modifier le temps de propagation de ces dernières. Cela est vrai, si toutes les entrées sont présentes en même temps et si les poids respectifs des diverses sorties sont respectés.

Parallèlement à ces trois constats, le facteur de forme des sommes (répartition des bits des termes à additionner dans les diverses colonnes à réduire) implique des fluctuations dans la traduction matérielle des algorithmes de sommation (section 3.3). Ces fluctuations donnent lieu à l'emploi, non pas d'une, mais de plusieurs cellules élémentaires : si la cellule principale est une cellule $4/2$ alors on utilise aussi des FA et des HA, si la cellule principale est un FA alors on utilise aussi des HA (figure 3.15). Cette multiplication des cellules usitées, peut-être considérée comme la déclinaison de la cellule principale ($4/2$ ou FA) en plusieurs versions dégradées, déclinaisons liées à la nullité (à l'inexploitation) de plusieurs entrées de la cellule de départ.
Dans ces conditions, en modifiant la construction des sommes, il est possible de profiter des entrées retardées pour réintégrer des sorties précoces dans un même étage de réduction. Mettre en place un tel mécanisme n'est certes pas utile à la sommation, mais va permettre d'effectuer des retours partiels en notation NR et cela, sans aucune dégradation temporelle. Le coût en surface sera toutefois non nul.
Deux remarques :

1. La longueur maximum de recombinaison des retenues précoces, sur un même étage de réduction, sans modification du temps de propagation, est donnée par l'enchaînement d'un HA, d'un FA et d'une cellule $4/2$.
2. Il est intéressant de noter, que les additionneurs classiques CLA présentent un temps de propagation pratiquement identique à celui d'une cellule $4/2$ pour des dynamiques inférieure ou égale à quatre bits.

Modification de l'algorithme de somme

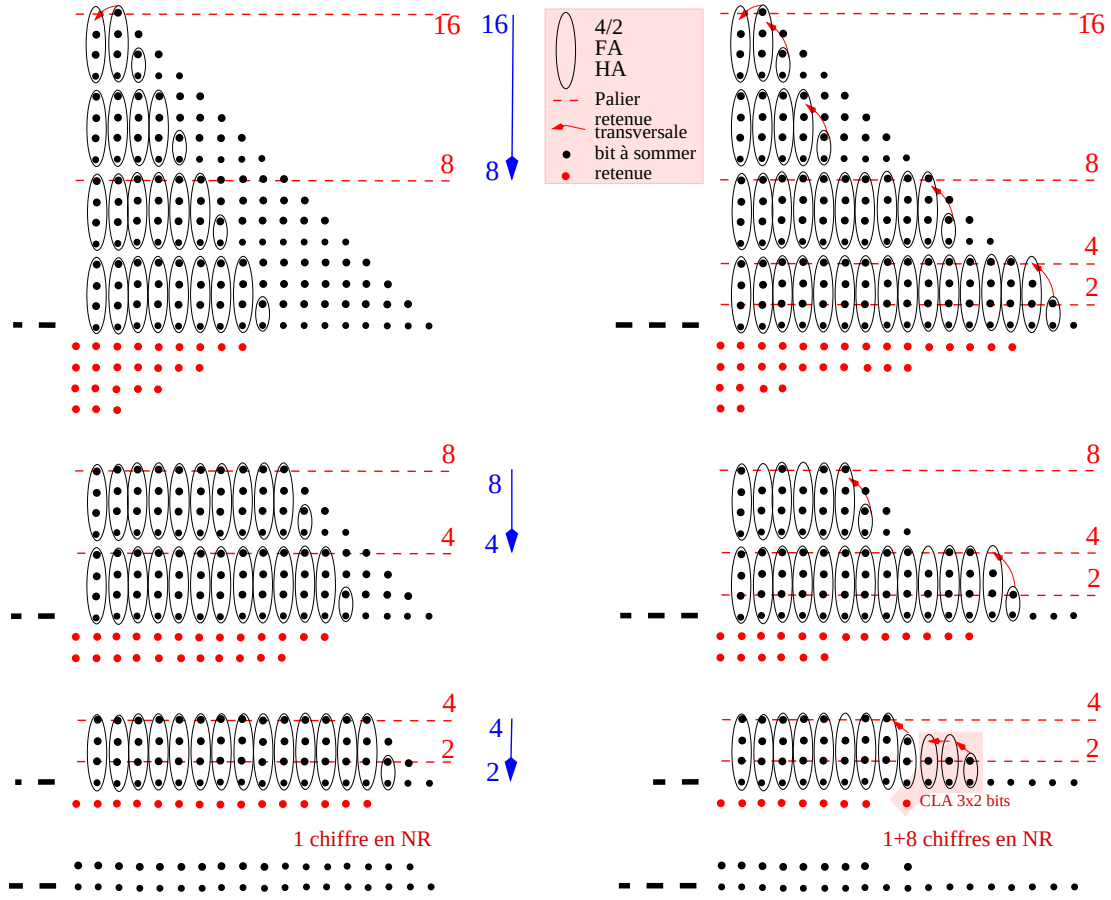
L'objectif est d'effectuer des retours en notation NR au niveau chiffre, dès que possible, tout en limitant au maximum le coût matériel et en respectant la dynamique de la sortie. Toutefois, il

ne s'agit pas pour nous de présenter, ici, un algorithme traitant tous les cas possibles. Notre but se limite juste à proposer des modifications mineures de l'algorithme de réduction existant, afin de répondre aux deux cas généraux de somme les plus rencontrés : facteur de formes triangulaire (multiplieur) et rectangulaire (somme de résultat de multiplication par exemple).
La première contrainte à respecter est celle de la conservation de la dynamique de la sortie. Pour ce faire, il suffit de limiter les modifications de l'algorithme initial aux cas où la colonne à réduire considérée est moins haute que sa suivante. Dans ces conditions aucun bit superflu ne peut-être généré. La dynamique est respectée.
Les modifications à apporter à l'algorithme initial, que nous proposons sont au nombre de trois :

1. Anticiper la réduction en effectuant dès que possible des abaissements vers le plus petit palier possible.
2. Recombiner sur le même étage de réduction, les retenues des HA et des FA dans le respect d'une profondeur de 3 cellules (remarque précédente). On trouve une proposition similaire dans [GuyoX2].
3. Remplacer toute succession de 4 colonnes de 2 bits par un CLA si elle ne modifie pas la dynamique, la retenue finale n'étant pas recombinée transversalement. Dans le cas où la quantité de colonnes est inférieure à 4, la retenue peut-être recombinée.

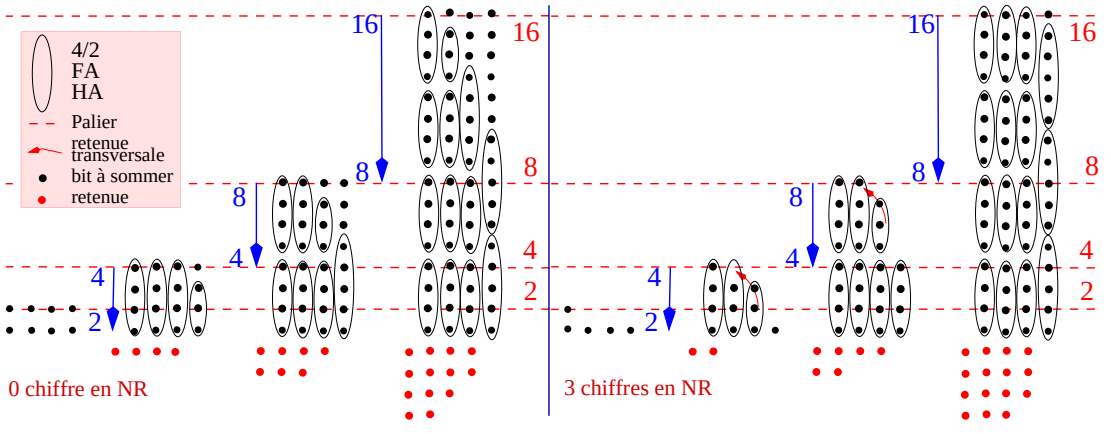
Exemples

La figure 5.14 présente l'application des modifications de l'algorithme de somme, sur la matrice de produits partiels d'un multiplieur 16x16. Comme ce nouvel algorithme n'a pas d'incidence sur la réduction des bits de poids forts ($16 \rightarrow 31$), pour plus de clarté, les diverses étapes de réduction présentées ne portent que sur la partie basse de la matrice ($1 \rightarrow 16$).
Dans cet exemple le nombre de chiffres définis sur un seul bit, passe de 1 à 9. Le temps de propagation n'a pas été modifié. Le coût matériel est de 2 cellules $4/2$ et de 3 cellules FA. L'augmentation de la surface sur tout le réducteur est inférieure à 2%. Pour un multiplieur 8x8, le coût serait d'un seul $4/2$ pour le passage de 1 à 5 du nombre de chiffres sur 1 bit, soit moins de 5% de la surface totale.
Un second exemple est présenté figure 5.15. Cette fois, le facteur de forme est rectangulaire. Comme précédemment, les modifications portent uniquement sur les bits de poids faibles d'où la limitation sur la figure à une dynamique de quatre bits. Dans ce second exemple le nombre de chiffres définis sur un seul bit augmente aussi. Il passe de 0 à 3 pour un coût matériel équivalent à 1.5 cellules $4/2$. Respectivement pour des dynamiques d'entrées de 4, 8 et 16 bits, l'augmentation du matériel représente 6%, 3% et 1.5% de la surface totale.



(a) Algorithme initial (b) Algorithme modifié

Figure 5.14 – Retour partiel en NR : réduction des produits partiels d'un multiplieur 16x16



(a) Algorithme initial (b) Algorithme modifié

Figure 5.15 – Retour partiel en NR : facteur de forme rectangulaire

Remarques et synthèse

Les retours partiels en notations classiques permettent d'élargir sensiblement l'impact des notations partiellement redondantes. Toutefois leur utilisation est contextuelle et donc difficile à mettre en œuvre dans une approche haut niveau. Nous reviendrons sur ce problème dans le chapitre 7.
Globalement le nombre de chiffres définis en NR obtenu en plus, et le coût matériel fluctueront en fonction du facteur de forme de la somme à réduire. Dans les exemples proposés, ce coût est inférieur à une cellule FA (un demi $4/2$) par chiffre redéfini. L'augmentation de la surface sera toujours faible.
Les retours en NR ne s'appliquent pas directement sur les signaux redondants ; ils s'effectuent à l'intérieur des opérateurs. Dans cette section nous avons exclusivement traité la modification de la somme. Toutefois, comme les additionneurs et les multiplieurs mixtes et redondants correspondent ou intègrent des sommes, ils sont aussi traités.

5.3.3 Remarques et conclusions sur les notations redondantes partielles

Lors de la conception d'un circuit, il n'est pas impossible d'avoir affaire à des signaux partiellement redondants : composition d'un signal redondant à partir de deux signaux classiques non uniformes, résultat redondant asymétrique d'un opérateur,... Assurer la cohérence de ces signaux revient à compléter chaque chiffre défini sur un seul bit par un second bit. Celui-ci prendra une valeur constante (zéro ou un) ou sera la recopie d'un autre (extension de signe par exemple).
Propager la composante constante de ces signaux dans les opérateurs, permettra de dégrader les architectures, et par-là même de réduire la surface et potentiellement le délai et la consommation. Un tel mécanisme est proche de la réduction d'équations logiques effectuée dans la synthèse logique.
Parmi les opérateurs présentés dans le chapitre 3 la quantité de chiffre défini par un seul bit est faible. Afin d'augmenter ce nombre et donc les gains potentiels introduits par les notations redondantes partielles, l'algorithme de somme a été légèrement modifié. Ces modifications se sont traduites par une augmentation sensible du nombre de chiffre NR. Le coût en surface est très faible, et le temps de propagation global n'est pas altéré.
Remarque : manipuler des représentations redondantes partielles revient à manipuler des redondants dont les deux termes n'ont pas la même taille, dont les LSB, MSB et bits de signe non pas les mêmes poids. Chaque nombre devient alors une représentation à part entière. Dans le cadre d'un conception hiérarchique basée sur la génération, cette particularité pose de nombreux problèmes. Nous reviendrons sur ce problème dans le chapitre 7.

5.4 Remarques et Conclusions sur la redéfinition des enchaînements

À travers les exemples exposés dans ce chapitre, que ce soit en combinatoire ou en séquentiel, la redéfinition des enchaînements apparaît comme constructive. Applicable aussi bien au niveau vecteur qu'au niveau chiffre, elle introduit des gains pouvant être importants à la fois en terme de surface, de délai et de puissance consommée.
Dans le cas des chaînes combinatoires, l'usage direct et sans contrainte des redondants, garantit au pire cas que les performances initiales ne seront pas détériorées. Dans le cas des circuits séquentiels, obtenir des résultats similaires implique l'utilisation d'une analyse temporelle. Celle-ci est rendue nécessaire par l'impossibilité de modifier l'ensemble des connexions d'un circuit et donc de compenser la redéfinition des entrées d'une chaîne combinatoire par la redéfinition des sorties de la même chaîne. Cette analyse a pour rôle d'autoriser ou non le franchissement des points mémorisants par un redondant. Elle est basée sur l'étude des déséquilibres des divers chemins des blocs combinatoires.
Remarque : Appliquer à ces chemins combinatoires, la même analyse temporelle permet d'affiner et d'améliorer les résultats en délai déjà obtenus. Dans ce contexte, la diminution du temps de propagation se fait au détriment de la surface car elle correspond à la réintroduction d'une conversion $R \rightarrow NR$ préalablement supprimée.

Sur le plan de la recherche de règles d'optimisations, il apparaît que le *redondant partout où c'est possible*, avec ou sans respect des sorties sur registres, conduit à la surface minimum mais pas au délai minimum. Pour ce second critère, le passage des registres va le plus souvent être source d'une contre performance.
Pour obtenir le temps de propagation le plus faible possible, il est impératif d'effectuer en plus une analyse temporelle. Elle se traduit par la diminution du délai et par une augmentation de la surface. Les deux sont liées à la réapparition de conversions $R \rightarrow NR$ préalablement supprimées.
Les notations redondantes permettent de mettre l'accent indifféremment sur la diminution de la surface ou du temps de propagation. Dans les deux cas le deuxième critère sera quand même aussi amélioré par rapport à une réalisation en notation classique. Ce constat et l'analyse de temps semblent aussi automatisables.

Seul l'aspect *opérateurs redondants* a été pris en compte dans ce chapitre. Les particularités de la somme et les propriétés de commutativité et d'associativité n'ont pas été étudiées. L'impact de l'arithmétique en générale est présenté dans le chapitre suivant.

Chapitre 6 Regroupement, fusion et glissement d'opérateurs

Au-delà des possibilités offertes par la redéfinition des notations en entrées/sorties des divers opérateurs arithmétiques, il est possible de profiter des propriétés intrinsèques à certains opérateurs, l'associativité ou la commutativité par exemple, pour optimiser un peu plus l'implémentation d'une architecture.

Dans ce chapitre nous porterons notre attention essentiellement sur les propriétés des opérations de somme et de mémorisation. La somme, car sa réalisation sous forme d'arbre d'additions élémentaires [Wall64, Dadd65] permet une mise en parallèle optimale ou modulable des additions composant la somme à effectuer. Les registres, pour leur qualité de borne temporelle et pour leur aspect *transparence* en terme de traitement effectué. Faire glisser un opérateur de l'entrée vers la sortie d'un point mémorisant ne modifie pas le fonctionnement de l'architecture, mais va permettre d'équilibrer les chaînes combinatoires et de tirer profit des propriétés d'autres blocs.

Dans un premier temps, nous nous intéresserons à la sommation et aux arbres d'additions utilisés pour sa réalisation. Nous distinguerons deux modes d'exploitation des propriétés des sommes : le regroupement, s'appliquant à un même type d'opérateurs, et la fusion qui consiste à l'absorption d'un premier opérateur par un second de *nature* différente. Dans les deux cas, les mécanismes mis en œuvre sont identiques : dilution d'additions dans une ou des sommations et mise en parallèle de ces opérations. Toutefois la réalisation et les implications ne sont pas les mêmes.

Dans un deuxième temps, nous nous concentrerons sur les registres et sur les circuits séquentiels. Nous parlerons du *glissement* d'opérateurs à travers les registres. Cette première forme de glissement sera accompagnée de l'étude de la traversée des opérateurs arithmétiques par des blocs effectuant des traitements uniformes comme les multiplexeurs.

Ce chapitre s'inscrit principalement dans la recherche de mécanismes d'optimisations liés aux chemins de données et à l'arithmétique en générale, susceptibles d'être automatisés. Une des contraintes majeure que nous nous sommes fixé est le respect du comportement du chemin initial, à son interface, au bit près et au cycle près.

Les optimisations présentées recoupent celles déjà exploitées par des outils comme BOA et BRT de Synopsys [BOA00]. Nous partons du principe que la fusion et le regroupement s'appliquent après *le passage en redondant partout où c'est possible*.

6.1 Quelques remarques préliminaires

La fusion et le regroupement s'appuyant sur les sommes, nous détaillons ici les propriétés des réducteurs de Wallace [Wall64], que nous allons exploiter. Certaines des remarques faites dans cette section recoupent celles effectuées dans la section 3.3.

Réducteur de Wallace

Le réducteur de Wallace effectue une sommation le plus parallèlement possible. Il utilise comme élément de base le Full-Adder et, si elles existent, les cellules 4/2 (sous-section 3.2.4). Le calcul s'effectue en ne tenant compte que du poids des divers bits composant les chiffres des nombres à sommer. Aussi, l'intégrité des chiffres et des nombres n'est pas respectée. Un réducteur de Wallace correspond à un arbre totalement parallèle composé d'additionneurs redondants $CS = CS + NR$ ou $CS = CS + CS$ travaillant au niveau bit/chiffre.
Comme le traitement est totalement parallèle, toutes les entrées doivent être présentes en même temps afin de ne pas retarder le calcul. En schématisant, si une donnée arrive avec un retard T, le temps de propagation global est équivalent à celui du réducteur, plus T. En réalité, le décalage peut être d'un étage sur quelques entrées, soit le coût de la traversée d'un Full-Adder ou d'une cellule 4/2, mais ce n'est pas systématique.
Autre point important : rajouter une ou plusieurs entrées dans les étages autres que le premier d'un arbre de Wallace n'introduit pas de gain en délai. L'augmentation du nombre de termes à sommer impose une nouvelle réduction vers la valeur de convergence de l'étage initial ce qui revient en terme de délai, à ajouter un étage de plus.

Additionneurs mixtes et redondants

Les additionneurs susceptibles d'être utilisés dans les arbres d'additions : redondant ($CS = CS + CS$) et mixte ($CS = CS + NR$), sont en fait des réducteurs de Wallace. L'élément de base est aussi le Full-Adder ou la cellule de type 4/2.
La généralisation de l'usage des redondants impose que tous les nombres redondants soient composés de deux nombres classiques ayant les mêmes propriétés. Dans ces conditions, la sortie d'un additionneur mixte voit l'extension de ces deux composantes à la taille du plus grand des deux. Il y a génération d'un bit de poids fort superflu à l'expression du résultat.

Plus généralement, la prise en compte de tailles d'opérandes différentes, de la gestion des signes et de la cohérence des redondants, implique une dynamique de sortie pouvant être trop importante par rapport à un résultat asymétrique et donc un surplus de matériel. Cette remarque est valable pour les deux additionneurs mixte et redondant.

Arbres d'additions

Dans un arbre, les nombres sont additionnés deux à deux. Le calcul de la somme globale respecte l'intégrité des nombres, c'est à dire que l'on ne sépare pas les deux NR composant un redondant comme dans un réducteur de Wallace. La sommation donne lieu à des résultats intermédiaires devant respecter les règles établies pour permettre la généralisation de l'usage des redondants.
Quel que soit l'arbre, série, parallèle ou série/parallèle, le nombre d'additions nécessaires reste le même. Il est égal à $N-1$ si N est le nombre de termes à sommer. Cette valeur est valable que les entrées soient redondantes ou non.
Pour les arbres totalement série le calcul est trivial. Pour un arbre totalement parallèle complet ($N = 2^n$ entrées), le nombre de couche d'additions est en $1 + log_2(N)$ et le nombre d'opérations par étage est égal à la moitié du nombre d'entrées de cet étage. En équation : $\sum_{n=1}^{n=1+log_2 N}(2^{n-1}) = N - 1$. Pour les structures semi-série/semi-parallèle, le remplacement d'une partie série par une partie parallèle correspond au remplacement d'un arbre totalement série par un arbre totalement parallèle : le nombre d'opérations est conservé. La quantité d'additions à effectuer reste identique.
L'utilisation des deux additionneurs, mixte et redondant, n'implique pas de différence de matériel. La constitution des $CS = CS + CS$ fait que le nombre de $CS = CS + NR$ est conservé sous-section 3.2.4. Conséquences : les fluctuations de surfaces entre deux arbres effectuant la même somme, ne sont liées qu'aux propagations de différences de dynamiques.
Dernier point, avec les notations mixtes et redondantes, la correspondance opération / opérateur n'existe que si on tient compte des opérateurs virtuels : $CS = NR + NR$.

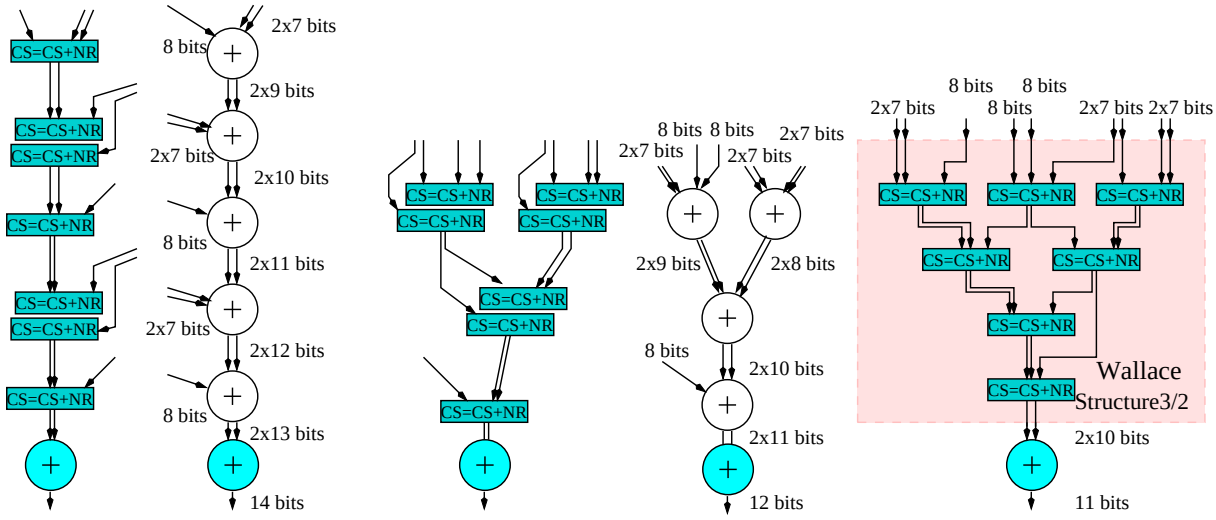
6.2 Regroupement d'Opérateurs

Le principe du regroupement est simple. Il consiste à profiter de l'associativité de certaines opérations pour grouper des opérateurs chaînés effectuant la même opération en un seul opérateur ou en un nouvel ensemble d'opérateurs dont les performances sont meilleures.
Concernant l'arithmétique, les regroupements sont destinés à réduire les arbres d'additions en les remplaçant soit par un réducteur de Wallace (Opérateur de somme section 3.3), soit par un

ensemble de réducteurs de Wallace. L'amélioration des performances viendra de la mise en parallèle totale ou partielle des calculs, et des qualités propres au réducteur.
Afin d'exprimer les gains introduits par un regroupement, il nous faut définir une référence. Nous prendrons comme base d'évaluation les architectures obtenues après le *passage en redondant partout où c'est possible*.

6.2.1 Remplacement d'un arbre d'additions par un regroupement unique

La première question qui se pose est : le remplacement d'un arbre d'additions par un regroupement sous forme de réducteur de Wallace est-il intéressant et si oui dans quelle mesure ?
Quelle que soit la réalisation choisie, réducteur ou arbre, les remarques préliminaires montrent que les mécanismes de sommation mis en œuvre sont similaires. La nuance tient exclusivement au respect de l'intégrité des entrées et des résultats intermédiaires. Cette nuance se traduit par une différence de grain de parallélisme entre les arbres et les réducteurs, respectivement additionneurs au niveau nombre et additionneurs au niveau bit.
Si l'on considère le remplacement d'un arbre par un réducteur, indépendamment du contexte d'utilisation, les performances du second seront toujours meilleures. Pour illustrer ce propos, la figure 6.1 présente trois réalisations différentes de la même somme. Les deux premières correspondent aux deux formes extrêmes d'arbres : additions effectuées en séries et additions effectuées parallèlement, la troisième correspond à la réalisation de la somme sous forme de regroupement. Toutes les entrées ont une dynamique de huit bits (soit un codage sur sept chiffres pour les entrées redondantes).
L'exemple proposé figure 6.1 illustre parfaitement la conservation du nombre d'additions et du nombre d'opérateurs, énoncée précédemment. Concernant les deux arbres, cette conservation se traduit par des surfaces proches. La surface du regroupement est, elle, bien moins importante. Ces écarts sont la conséquence du respect, dans le cas des arbres, de l'intégrité des E/S et des opérateurs (poids faibles recalculés, poids forts superflus). Les différences de surface sont importantes car la dynamique des entrées est faible. Avec une dynamique plus grande, les écarts seraient plus réduits. Une autre conséquence de la différence du niveau de parallélisme (série, parallèle, parallèle au niveau bits) est la fluctuation de la dynamique de la sortie. Une fois de plus, les meilleurs résultats sont obtenus pour le regroupement.
L'examen des tableaux associés à la figure 6.1 montre, comme attendu, une nette amélioration du temps de propagation par la mise en parallèle des additions de l'arbre série. La substitution de l'arbre parallèle par un regroupement présente elle aussi un écart de délai important. Sachant que les réducteurs de Wallace et les arbres de réductions réalisent tous les deux une sommation, le plus parallèlement possible et avec le même élément de base : le Full-Adder



(a) Arbre série respect des E/S des additionneurs

ns	μm^2	$\mu W/MHz$
11.77	70743	192
réf.	réf.	réf.
71 FA 0 HA		

(b) Arbre parallèle respect des E/S des additionneurs

ns	μm^2	$\mu W/MHz$
9.61	60025	138
-18.3%	-15.1%	-28.1%
60 FA 0 HA		

(c) Regroupement : traitement indifférencié au niveau bits

ns	μm^2	$\mu W/MHz$
8.06	47040	80
-31.5%	-33.5%	-58.3%
48 FA 4 HA		

Figure 6.1 – Exemple de regroupement

($CS = CS + NR$), on pouvait s'attendre à des résultats beaucoup plus proches. La différence vient de l'utilisation dans l'arbre parallèle de blocs $CS = CS + CS$, nécessaire au respect de la correspondance opération / opérateur.

Dans l'exemple donné, l'additionneur $CS = CS + CS$ a une structure interne équivalente à deux $CS = CS + NR$ mis en série. Le parallélisme de l'arbre n'est donc pas total. Ce problème disparaît si les cellules de type $4/2$ existent (sous-section 3.2.4). Les temps de propagation sont alors identiques.

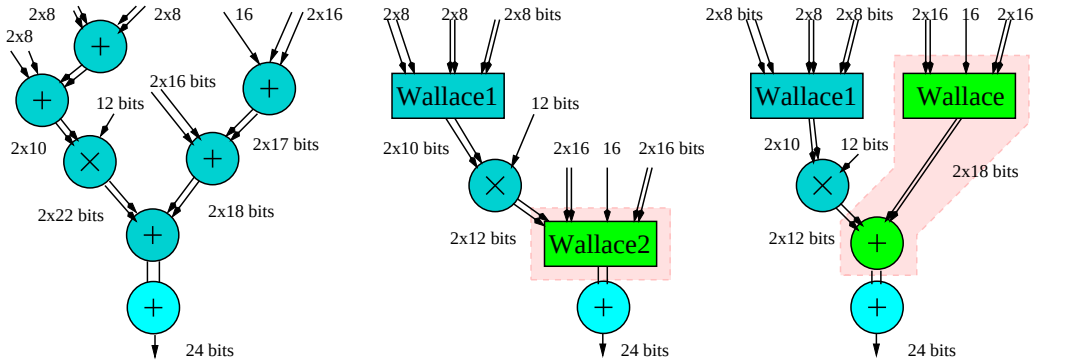
Les consommations des trois architectures présentent des écarts importants. Ces derniers s'expliquent par la diminution des commutations transitoires (*glitches*) introduite par la différence de parallélisme dans le calcul de la somme. Les meilleurs résultats sont une fois de plus obtenus par le réducteur. Toutefois, et comme pour la surface et le délai, la faible dynamique des entrées et l'indisponibilité des cellules de type $4/2$ ont *gonflé* les gains.

Ces résultats sont généralisables. Quel que soit le nombre de termes à sommer, un regroupe-

ment unique effectuera les opérations en utilisant moins de matériel que tout arbre d'additionneurs équivalent, tout en offrant le meilleur délai. Les gains fluctueront en fonction du taux de parallélisme de l'arbre de départ et de la disponibilité de la cellule de type 4/2. Un autre point intéressant apporté par les regroupements est un meilleur contrôle de l'extension de la dynamique des données.
Ces remarques sont les mêmes dans le cas d'opérations signées. La gestion des signes introduisant quelques différences de matériel : section 3.3, les écarts de performances ne seront pas exactement les mêmes. Toutefois, les résultats des réducteurs seront toujours meilleurs que ceux des arbres.

6.2.2 Prise en compte de l'aspect temporel dans les mécanismes de regroupements

Nous venons de voir que les performances intrinsèques des regroupements sont toujours meilleures que celles des arbres remplacés. Toutefois ces résultats ont été obtenus pour des substitutions hors contexte d'utilisation. Il nous faut maintenant considérer le remplacement d'un arbre d'additions par un réducteur de Wallace, dans le cadre d'une architecture. Soit, déterminer comment tenir compte de la variation de la disponibilité temporelle des divers termes à sommer. La figure 6.2 présente un arbre d'additions ainsi que deux regroupements équivalents.



(a) Arbre initial

ns	μm^2	$\mu W/MHz$
18.05	259332	943
réf.	réf.	réf.

(b) Regroupement direct

ns	μm^2	$\mu W/MHz$
18.20	245245	804
+1.0%	-5.4%	-14.07%

(c) Décomposition des regroupements

ns	μm^2	$\mu W/MHz$
16.35	246960	720
-9.42%	-4.77%	-23.65%

Figure 6.2 – Disponibilité des Entrées

Dans le cas du regroupement direct : figure 6.2.b, l'impératif de disponibilité mis en avant dans les remarques préliminaires n'est pas respecté. La somme n'est plus effectuée partiellement parallèlement à la multiplication comme dans l'arbre initial, mais intégralement après celle ci.

En conséquence si la surface diminue bien comme prévue, le temps de propagation subit un retard. On est en face d'une contre performance.
L'exemple proposé, est typique d'un bon nombre d'architectures où le remplacement direct d'un arbre d'additions par un opérateur somme unique, rallonge la chaîne longue. Il existe une solution permettant de contourner ce problème. Elle consiste à décomposer le réducteur de Wallace unique en un arbre de réducteurs, en tenant compte de la disponibilité des diverses entrées. Dans notre exemple le regroupement unique est décomposé en un réducteur et un additionneur $CS = CS + CS$: figure 6.2.c.
Comme pour les arbres d'additions, les performances propres d'un arbre de réducteurs sont moins bonnes que celle d'un réducteur unique et ce pour les mêmes raisons. Toutefois, ces pertes (en délai, surface, consommation et en dynamique) permettent de compenser les retards de disponibilités des divers termes à sommer. Finalement ils introduisent un gain en terme de délai pour une surface sensiblement identique. Parallèlement, la diminution du nombre de couches combinatoires permet de réduire le taux de *glitches* et donc la consommation. Dans le pire des cas, la substitution nous ramènera vers l'arbre initial : les regroupements auront toujours un gain positif. Pour être mise en œuvre, cette solution nécessite une analyse des temps de propagations.

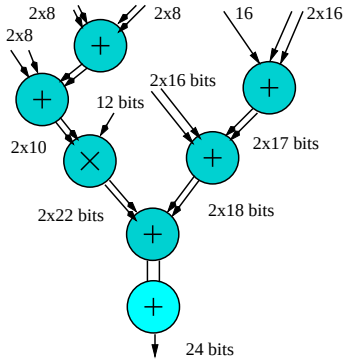
6.2.3 Analyse de temps

Comme le remplacement d'un arbre par un regroupement est toujours bénéfique en terme de surface (et vraisemblablement en consommation exemple figure 6.2.b), l'analyse a pour seul but de corriger les contres performances potentielles introduites par la substitution de tous les arbres d'additions inclus dans un circuit donné. Le problème posé est double. Il faut à la fois déterminer comment effectuer le découpage des arbres pour que les regroupements ne soient pas une source de contre performance en délai, mais aussi choisir entre appliquer ce traitement à tous les regroupements ou seulement à ceux impliqués dans le chemin critique.

Dans un premier temps nous allons considérer que la méthode de délimitation des regroupements est connue, pour s'intéresser plus particulièrement à l'application de l'analyse.

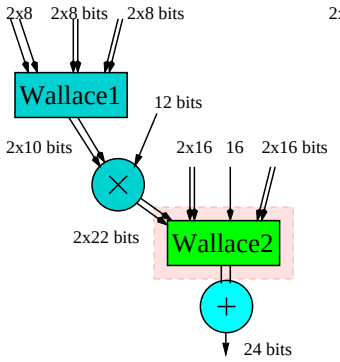
Application de l'analyse

Considérons l'arbre combinatoire de la figure 6.3 qui reprend l'exemple précédent (figure 6.2). Nous partirons du postulat que l'architecture proposée ne fait pas partie du chemin critique. Dans ce cas l'analyse peut être appliquée de deux façons différentes : *regroupements directs puis*



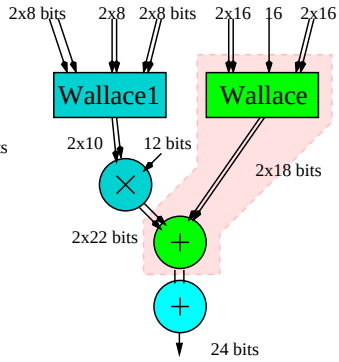
(a) Arbre initial

ns	μm^2	$\mu W/MHz$
18.05	259332	943
réf.	réf.	réf.



(b) Découpage uniquement des chaînes longues

ns	μm^2	$\mu W/MHz$
18.20	245245	804
+1.0%	-5.4%	-14.07%



(c) Partitionnements systématiques

ns	μm^2	$\mu W/MHz$
16.35	246960	720
-9.42%	-4.77%	-23.65%

Figure 6.3 – Regroupements : Application de l'analyse des temps

partitionnement uniquement des regroupements appartenant au chemin critique (donc pas de modification), et *regroupements directs avec découpage systématique*.

Dans cet exemple, les différences de disponibilité temporelle des entrées n'influent pas sur le temps de propagation puisque selon le postulat de départ, la chaîne longue est ailleurs. La différence de surface est non significative. Ces résultats reflètent le cas général. Comme ces deux indications ne sont pas déterminantes, nous allons essayer de fixer la méthode d'application de l'analyse en fonction d'autres critères :

– La consommation : positionner un réducteur de Wallace en sortie d'un multiplieur, comme le suppose le *découpage uniquement sur le chemin critique*, implique un nombre de couches combinatoires plus élevé entre les entrées et les sorties. Dans ce cas, le taux de *glitches* est probablement plus élevé et la consommation est plus forte.
– La dynamique : à la vue de l'exemple figure 6.1, on peut déduire que plus on partitionne un regroupement, plus la dynamique de sortie risque d'augmenter. Dans ce cas le *découpage selon les chemins critiques* redevient intéressant. Dans notre exemple, la somme des deux branches se faisant avec un additionneur $CS = CS + CS$, la dynamique ne diffère pas. Comme on cherche à généraliser les notations redondantes, ce cas sera fréquent.

Dans ces conditions le choix d'application de l'analyse se portera sur *le découpage systématique des regroupements via l'analyse*, d'autant que la consommation est un critère de plus en plus prépondérant.

Il faut maintenant développer la méthode de délimitation du découpage des regroupements.

Découpage des regroupements

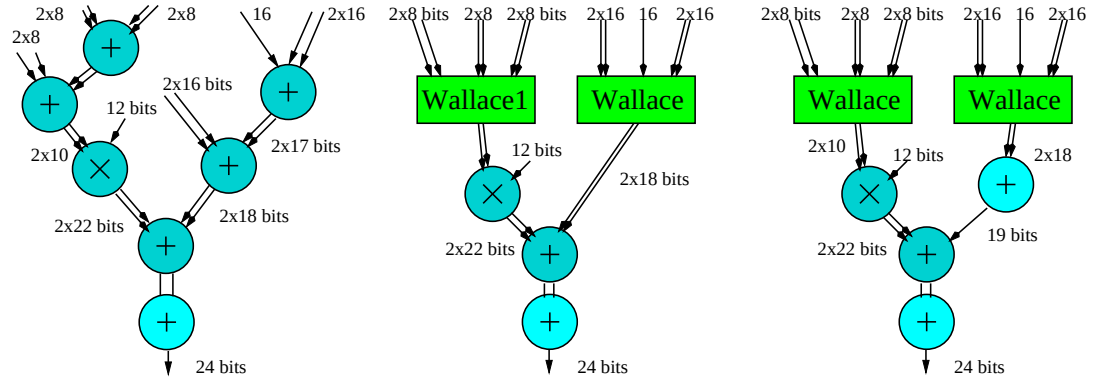
Le découpage des regroupements se fait en plusieurs étapes. La première consiste à délimiter les arbres d'additions qui seront ensuite substitués par un ou des réducteurs de Wallace. Cette étape est basée sur la localisation des divers arbres d'additions et tient compte de la distribution éventuelle des résultats partiels des sommes effectuées. Les délais ne sont pas encore pris en compte. Nous qualifierons de *directs* les regroupements ou sommes ainsi localisés.
La seconde étape consiste à découper les regroupements *directs* en fonction de la disponibilité de leurs entrées. C'est à dire, rassembler par paquets les entrées comprises dans une même fourchette de temps. Ce tri s'effectue au niveau le plus élémentaire (bit). Une fois les entrées triées, l'arbre sera parcouru pour composer les divers sous-regroupements et pour préciser l'enchaînement de ceux-ci. Les sous-sommes obtenues correspondent à la décomposition des sommes *directes* en arbres de réducteurs.
Les groupements doivent être considérés dans l'ordre chronologique de la plus proche des entrées vers la plus proche des sorties. Le but est d'être sur de la stabilité temporelle des entrées avant d'effectuer un découpage.

Spécifier une fourchette de temps est nécessaire pour prendre en compte au même niveau, les entrées dont la disponibilité temporelle est proche. La valeur de cet intervalle est fonction du contexte. Plus l'écart entre la sortie du regroupement et l'entrée la plus tardive est important, plus la fourchette sera grande. Par contre, plus le nombre de termes à sommer est grand plus la fourchette va se rétrécir.

Remarques :
- Comme les regroupements directs sont considérés du plus vieux au plus jeune, les fluctuations de la fenêtre temporelle ne seront pas source d'erreur.
- La souplesse de cette méthode permet de prendre en compte les *petits* retards liés aux opérateurs de transferts (multiplexeurs, ...).
- L'équilibrage des temps s'effectuant au niveau bit, les signaux vectorisés (nombres) ne seront pas forcément respectés par le découpage en plusieurs regroupements.

Retour en Non Redondant

Comme lors de la redéfinition des enchaînements combinatoires, le temps de propagation peut encore être amélioré par un retour en notation non redondante (figure 6.4). Le cas 6.4.a présente une architecture où le passage en redondant a déjà été effectué. Le cas 6.4.b correspond aux regroupements des divers arbres d'additions et le cas 6.4.c aux divers regroupements ac-



(a) Passage en redondant

ns	μm^2	$\mu W/MHz$
18.05	259332	943
réf.	réf.	réf.

(b) Regroupement

ns	μm^2	$\mu W/MHz$
16.35	246960	720
-9.42%	-4.77%	-23.65%

(c) Retour en NR

ns	μm^2	$\mu W/MHz$
16.10	261108	728
-10.80%	+0.7%	-22.8%

Figure 6.4 – Retour en notation non redondante

compagnés d'un retour en NR. Comme attendu, à chaque étape le temps de propagation est amélioré. Le gain attendu correspond au passage d'un additionneur CS+CS à un additionneur $CS + NR$, soit, la suppression d'une couche de FA. Dans notre exemple la fourchette de temps n'est que très légèrement supérieure au coût de la conversion et les temps de propagation sont très proche. Par contre, pour la surface il en va autrement ; l'ajout d'une conversion introduit une augmentation du matériel. La consommation quant à elle, reste stable. Ces résultats sont généralisables.

Globalement, pour que le retour en notation classique soit envisageable, la sortie d'un regroupement doit présenter un écart temporel de l'ordre d'une conversion $CS \rightarrow NR$ entre le moment où elle est générée et le moment où elle est prise en compte. De plus cette conversion doit être recombinée dans le chemin critique du circuit. Les retours éventuels en NR s'effectueront après le traitement de tous les regroupements directs. Les mécanismes mis en œuvre et les remarques sont les mêmes que celles formulées lors de la redéfinition des enchaînements combinatoires sous-section 5.1.

Regroupements systématiques

Dans le cas où toutes les entrées d'un arbre ou d'une partie d'arbre d'additions n'ont pas de contraintes de temps (sortie de registres), il est possible de s'affranchir de l'analyse de temps en définissant un regroupement que nous qualifierons de *systématique*. Pour cela on va regrouper tous les additionneurs utilisant uniquement ces entrées ou les résultats partiels qui leur sont associés.

6.2.4 Regroupement : synthèse

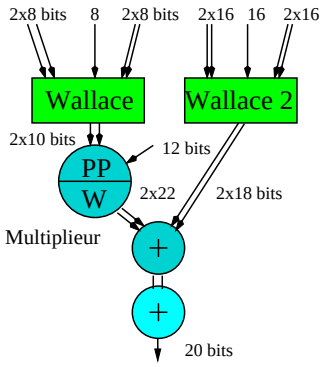
Potentiellement les regroupements peuvent être à l'origine de gains en délai, en surface et en consommation. Toutefois la solution idéale du regroupement *direct* ne constitue pas le cas général mais seulement une étape du processus. Elle sera même, le plus souvent, source d'une contre performance en délai.
Une analyse de temps s'impose pour tirer parti des qualités des réducteurs de Wallace. Le déséquilibre des chemins combinatoires permettra, en décomposant le réducteur *direct* en un ensemble de réducteurs et en réordonnant l'ordre d'addition des entrées (au niveau bit), de tirer le meilleur parti des opérateurs de somme. En plus de l'amélioration du temps de propagation, l'équilibrage des chemins va aussi permettre de diminuer la quantité de*glitches* et par-là même la consommation.
Ces manipulations correspondent à l'exploitation des propriétés de commutativité et d'associativité de l'addition. Ces mécanismes sont similaires à ceux employés dans la synthèse logique.

Trois remarques :

1. L'optimisation effectuée est purement combinatoire. Elle s'inscrit dans la continuité des améliorations introduites dans le chapitre 5 traitant de la redéfinition des enchaînements. Les regroupements s'insèrent entre le passage en notations redondantes et avant le retour en notations non redondantes.
2. Dans l'architecture initiale, les arbres décrits pouvant inclure des sommes, il serait profitable d'ajouter la notion de *dé*-groupement pour casser d'éventuels réducteurs. Globalement les sommes seront considérées comme des arbres parallèles d'additionneurs mais au niveau le plus élémentaire.
3. Concernant l'exemple présenté des figures 6.2 à 6.4, les regroupements ont permis de réduire de 20% le délai, de 5.4% la surface et de 29% la consommation.

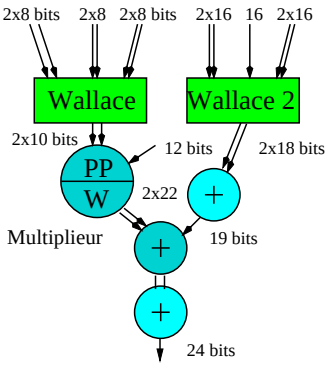
6.3 Fusion d'Opérateurs

L'optimisation peut-être poussée plus loin en autorisant la recombinaison des additionneurs avec les sommes (réducteurs de Wallace) internes à d'autres opérateurs. La fusion n'est autre que la généralisation des regroupements.
Dans l'absolu, rien n'interdit d'effectuer tous les regroupements possibles et donc de fusionner les sommes inclues dans les opérateurs dont les résultats sont additionnés. Un tel processus équivaudrait à remplacer un arbre de sommes et d'additions par un réducteur de Wallace unique. Dans le cadre de la synthèse d'architecture, mettre en place un tel mécanisme



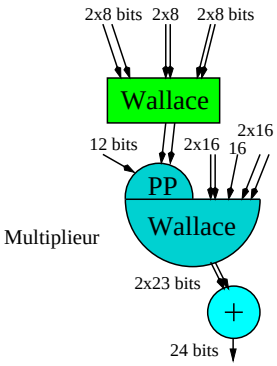
(a) Redéfinition des enchaînements

ns	μm^2	$\mu W/MHz$
16.35	246960	720
réf.	réf.	réf.



(b) Regroupement

ns	μm^2	$\mu W/MHz$
16.10	261108	728
-1.53%	+5.73%	+1.11%



(c) Fusion

ns	μm^2	$\mu W/MHz$
15.24	241815	700
-6.8%	-2.08%	-2.7%

Figure 6.5 – Exemple de fusion

serait problématique car l'automatisation du flot de conception s'appuie sur l'exploitation de macro-fonctions comme les opérateurs arithmétiques. Fondre ensemble les sommes internes à plusieurs opérateurs reviendrait à casser ces opérateurs complexes et donc à interdire les macro-fonctions arithmétiques.

La fusion peut quand même être exploitée en autorisant l'ajout de termes externes exclusivement dans les opérateurs complexes intégrant une somme. Dans ces conditions, la fusion est locale, l'intégrité de la notion de macro-bloc est respectée et la conception hiérarchique est possible. On pourra par exemple, ajouter plusieurs nombres dans un multiplieur mais en aucun cas fondre ensemble les sommes internes à deux multiplieurs dont les résultats sont additionnés.

6.3.1 Exemple de fusion

La figure 6.5 ci-dessus reprend l'architecture utilisée comme exemple dans la section 6.2 consacrée au regroupement. Sur les trois schémas proposés PP et W correspondent à la décomposition du multiplieur en matrice de Produits Partiels et Somme (réducteur de Wallace).

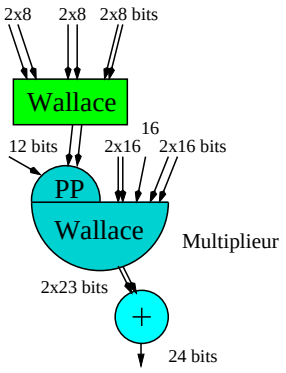
En premier lieu, la redéfinition des enchaînements, suivie du regroupement des divers sommes, a conduit à l'architecture figure 6.5.a. Sur la figure 6.5.b, ces optimisations ont été complétées par un retour en notation classique. Comme nous l'avons vu dans la section précédente, les améliorations des performances introduites portent à la fois sur la surface et sur le délai. Suivant l'architecture, le critère privilégié n'est pas le même. Dans le premier cas, l'optimisation porte plus sur la réduction de la surface alors que dans le deuxième cas l'effort porte plus sur la réduction du temps de propagation.

Les deux architectures décrites comprennent deux réducteurs de Wallace dont les résultats sont sommés, un interne au multiplieur et un externe (Wallace 2). L'optimisation arithmétique va alors être poussée plus loin en fusionnant ces deux sommes en une seule : ensemble 6.5.c.
Cette nouvelle optimisation se traduit par l'absorption de la branche d'additionneurs de droite par la somme interne au multiplieur, et par la disparition de l'additionneur mixte ou redondant de sortie (respectivement architectures figure 6.5.a et figure 6.5.b). Les nouvelles performances obtenues en délai et en surface correspondent aux meilleures performances des deux architectures précédentes. Ces résultats étaient prévisibles car la fusion est un regroupement.

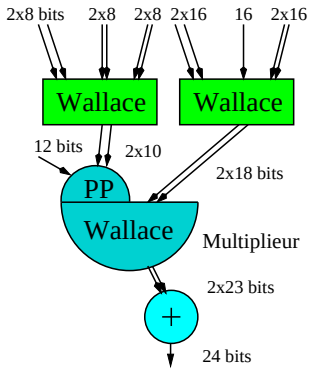
La question qui se pose maintenant est : combien de termes peut-on ajouter à une somme interne ?
Hors contexte ce nombre est infini, car la fusion correspondant au remplacement d'un arbre de réducteurs et d'additionneurs par un réducteur unique. Le gain en délai est toujours positif et l'augmentation du matériel est compensée par la disparition des opérateurs qui effectuaient précédemment la somme absorbée. Toutefois, la disponibilité temporelle des diverses entrées n'a pas été prise en compte ; l'exemple proposé correspond à un regroupement direct.
Il est quand même possible de déterminer rapidement cette quantité. On peut noter que la fusion fait disparaître l'additionneur effectuant l'addition des résultats des deux sommes à fusionner. Dans le meilleur des cas, cet opérateur est un additionneur mixte (figure 6.5.b). Si l'on veut éviter une contre-performance en délai il suffit de limiter la croissance du délai de la somme interne au coût d'un FA ($CS+NR \rightarrow$ traversée d'un FA). Aux vues de l'algorithme de somme utilisé (section 3.3), cette quantité est égale au $2/3$ du nombre de la plus haute colonne de bits de même poids de la somme interne.

6.3.2 Aspect temporel

La prise en compte des retards temporels des diverses entrées des sommes à fusionner, soulève les mêmes problèmes que ceux rencontrés pour les regroupements. La solution à adopter sera la même : analyse temporelle et équilibrage de tous les chemins. Toutefois, l'impossibilité de *dé*-grouper les sommes *internes* va limiter l'équilibrage.
Afin d'illustrer ces propos, l'exemple précédent (figure 6.5.c) est repris figure 6.6. L'architecture considérée a été construite de deux façons : fusion directe et fusion conditionnée par l'analyse. Chacune des ces deux réalisations a été déclinée en trois versions. Les différences tiennent aux dynamiques des entrées à fusionner (8, 12 et 16 bits) et à la dynamique du coefficient du multiplieur (8 bits pour les deux premières versions et 12 bits pour la dernière). Dans un premier temps nous ne considérerons que la première série de résultats.



(a) Fusion directe



(b) Fusion avec analyse de temps

1^{ere} série → *Prise en compte en bloc des termes à fusionner :*

bits	ns	μm^2	$\mu W/MHz$
16	15.24 (réf.)	241815 (réf.)	700 (réf.)
12	14.45 (réf.)	176400 (réf.)	482 (réf.)
8	14.25 (réf.)	152267 (réf.)	431 (réf.)

bits	ns	μm^2	$\mu W/MHz$
16	15.05 (-1.3%)	245245 (+1.4%)	668 (-4.6%)
12	14.74 (+2.0%)	179340 (+1.7%)	440 (-8.7%)
8	14.70 (+2.6%)	164707 (+8.2%)	424 (-1.6%)

2^{eme} série → *Répartition uniforme des termes à fusionner :*

bits	ns	μm^2	$\mu W/MHz$
16	16.25 (réf.)	241815 (réf.)	776 (réf.)
12	15.45 (réf.)	176400 (réf.)	517 (réf.)
8	14.25 (réf.)	152267 (réf.)	431 (réf.)

bits	ns	μm^2	$\mu W/MHz$
16	15.25 (-6.2%)	245245 (+1.4%)	672 (-13.4%)
12	14.90 (-3.6%)	179340 (+1.7%)	455 (-13.6%)
8	14.70 (+2.6%)	164707 (+8.2%)	424 (-1.60%)

Figure 6.6 – Fusion : analyse de temps

Concernant l'exemple utilisé (16 bits), les performances obtenues sont en accord avec nos conclusions précédentes : les surfaces augmentent légèrement avec le *re*-découpage en deux réducteurs de la somme et l'équilibrage des chemins introduit, au final, une amélioration du temps de propagation global et une réduction de la consommation.

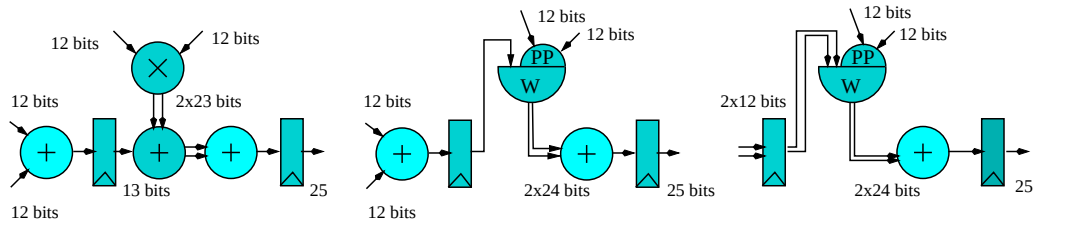
Toutefois les délais obtenus pour les architectures *8 bits* et *12 bits* ne sont pas en accord avec cette première analyse : la fusion directe (figure 6.6.a) offre un meilleur délai que la fusion avec analyse de temps (figure 6.6.b). Ce phénomène s'explique facilement par la différence de facteur de forme entre les deux sommes initiales. La répartition par poids, des bits à sommer, décrit un triangle dans le multiplieur et un rectangle dans la somme externe. Suivant le rapport de largeur entre le triangle et le rectangle (suivant la dynamique maximum de leur entrées respectives) la somme externe peut-être absorbée par la somme interne, avec une augmentation moindre du nombre de couches de réduction. Dans l'exemple 8 bits, la fusion directe des entrées (5x8 bits) n'induit pas d'augmentation du nombre d'étage de réduction alors que la fusion du résultat de l'addition des cinq entrées (2 fois 10 bits), nécessite un demi-étage de plus (coût de la traversée d'un HA). Ces résultats sont généralisables.

Le deuxième problème induit par l'impossibilité de *dé*-grouper les sommes internes, est comment les termes à fusionner doivent-ils être pris en compte dans la somme interne. À cet effet, les deux architectures précédentes (figure 6.6) ont données lieu à une nouvelle série de réalisations. Les délais et surface obtenus sont regroupés dans la deuxième série de résultats. Si dans la deuxième série, les surfaces n'ont subi aucun changement, les temps de propagations, eux, diffèrent fortement, au point d'être maintenant quasi identiques et d'annuler les contre-performances constatées précédemment.
Ces écarts sont liés uniquement à la différence de répartition des entrées fusionnées dans le calcul de la somme interne. La 1^{ere} *série* correspond à une distribution uniforme des nouveaux termes à sommer et la 2^{eme} à une prise en compte en bloc des nouveaux termes. Ce second cas revient pratiquement à un regroupement temporel des divers termes à sommer et, est très proche d'un découpage en deux arbres de réduction. Ceci explique les meilleurs résultats obtenus. Ces résultats peuvent aussi être généralisés.

Nous venons de le voir, l'impossibilité de *dé*-grouper les sommes internes, introduit des effets de bord dans l'application de l'analyse de temps au niveau opérateurs, *empêchant* d'obtenir les meilleures architectures. Remédier à ce problème suppose de connaître parfaitement la somme interne. Avec une approche opérateur, ce n'est pas possible. Les règles appliquées à la fusion seront donc les mêmes que celles appliquées aux regroupements.
Remarques :
- La solution fusion avec analyse de temps n'offre certes pas systématiquement de meilleures performances que la fusion directe, mais comparée aux architectures précédentes (figures 6.2 à 6.4), les gains sont toujours en sa faveur.
- comme avec les regroupements, la solution avec analyse de temps permet de limiter les commutations parasites (*glitches*), et *a priori* la consommation, ce qui est aussi un bon point pour cette solution.

6.3.3 Fusion systématique

Comme pour les regroupements, il est possible de définir des cas de fusion systématique. La condition nécessaire et suffisante est que tous les termes à additionner soient des données sans délai (sorties de registres par exemple). En fait, ce qui nous intéresse plus particulièrement c'est la fusion d'un terme unique dans le contexte d'une sortie directe sur registre comme présenté figure 6.7.
La première architecture (figure 6.7.a) correspond à la redéfinition des enchaînements combinatoires mais aussi à la meilleure solution obtenue à l'occasion de la redéfinition des enchaîne-



(a) Architecture initiale

ns	μm^2	$\mu W/MHz$
13.11	210210	424
réf.	réf.	réf.

(b) Fusion sans passage de registre

ns	μm^2	$\mu W/MHz$
11.87	208617	424
-9.4%	-0.75%	-0.0%

(c) Fusion après passage de registre

ns	μm^2	$\mu W/MHz$
12.32	208617	396
-6.03%	-0.75%	-6.60%

Figure 6.7 – Fusion Systématique

ments séquentiels. Les deux architectures suivantes, figure 6.7.b et figure 6.7.c, correspondent à l'application de la fusion avant et après le passage des redondants à travers les registres, soit, respectivement avec analyse de temps (retour en NR) et sans analyse.
On constate trois choses : d'abord que les résultats obtenus par l'utilisation de la fusion sont meilleurs dans les deux cas. C'était attendu. Ensuite que la deuxième et la troisième architectures offrent des performances très sensiblement identiques. Les écarts s'expliquent par la différence de matériel fusionné dans la somme interne au multiplieur. Le dédoublement du registre est compensé par la disparition de la conversion $CS \rightarrow NR$. Enfin et surtout, cette égalité de délai implique que la contre performance temporelle du passage des notations redondantes à travers les registres (sous-section 5.2.1), a été compensée.
Aux vues de ces résultats, on peut dire que la fusion permet de limiter les effets négatifs du passage de registre par les notations redondantes. Cette conclusion est généralisable à toute absorption à travers un registre, d'un redondant par une somme. La somme peut être interne à un opérateur (fusion) ou externe (regroupement).

6.3.4 Fusion : synthèse

Nous venons de le voir, la fusion correspond à la généralisation des regroupements. Les gains introduits sont du même ordre et sont aussi conditionnés par l'étude des temps de propagation. La fusion ou plutôt la généralisation de la substitution des arbres d'additions par des réducteurs de Wallace n'est toutefois pas complète. En effet, les optimisations arithmétiques proposées dans ce chapitre et dans le précédent, sont effectuées au niveau opérateur ce qui implique une description hiérarchique de l'architecture.
Dans ce contexte il est nécessaire de respecter les macro-blocs ce qui interdit le *dé*-groupement des sommes internes aux opérateurs complexes comme la multiplication. Par contre, augmenter la complexité de ces opérateurs n'est pas interdit. C'est ce que nous proposons avec la fu-

sion en autorisant l'absorption de termes externes dans les sommes internes aux opérateurs complexes. En terme d'algorithme rien n'est modifié. On se contente d'élargir le nombre et la diversité des opérateurs sur lesquels l'application sera projetée.
Dernier point, généraliser la fusion ou le regroupement est possible. Pour pouvoir effectuer cette optimisation, il est nécessaire que la description de l'architecture soit *à plat* (pas de hiérarchie) et que cette description permette de détecter les additionneurs au niveau le plus élémentaire.

Remarques :

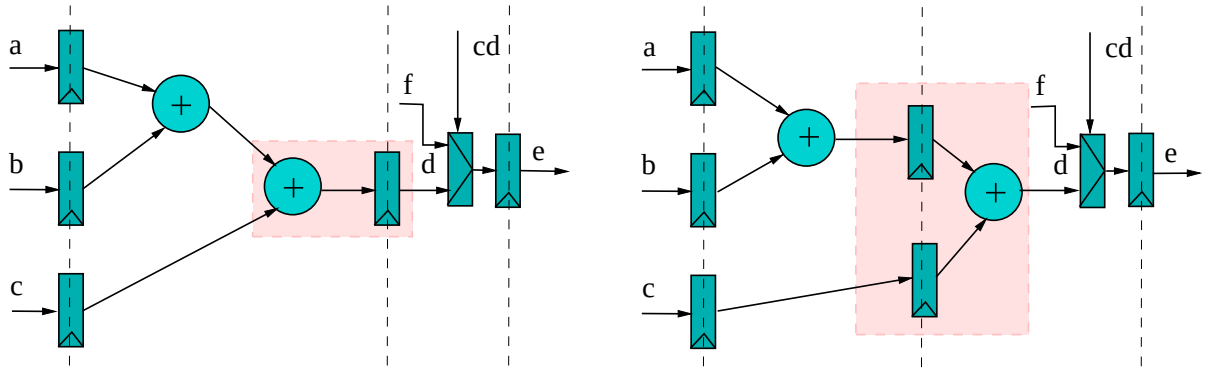
1. L'optimisation effectuée est purement combinatoire. Elle s'inscrit dans la continuité des améliorations introduites dans le chapitre 5 traitant de la redéfinition des enchaînements. Les regroupements et la fusion sont à effectuer après le passage en notations redondantes.
2. La fusion et plus généralement les regroupements permettent de corriger le problème du passage des registres par les notations redondantes. Les retours en NR doivent donc être effectués après la fusion et les regroupements.
3. Une fois de plus une analyse des temps est nécessaire.

6.4 Glissement d'opérateurs

Dans cette section, nous nous intéressons au passage d'un bloc de l'entrée vers la sortie d'un second. Nous parlerons de glissement. L'étude porte dans un premier temps, sur l'application de ce mécanisme aux points mémorisants, dont on va chercher à exploiter la propriété de borne temporelle. Il nous semble intéressant d'aborder ce problème car dans de nombreuses architectures, les opérateurs arithmétiques intègrent des barrières temporelles (*pipeline*). Toutefois, il ne s'agit pas ici, de déterminer où les barrières de registres doivent être positionnées dans un opérateur. Ce qui nous intéresse est de déterminer dans quelle mesure ces barrières potentielles peuvent être validées ou invalidées suite aux modifications apportées par la redéfinition de la nature des ressources arithmétiques (enchaînements, regroupements, fusions). Globalement nous distinguerons deux cas : les glissements *entiers* correspondant au transfert d'un bloc de l'entrée vers la sortie d'un point mémorisant et les glissements *partiels* qui reviennent à l'absorption de registre dans un bloc existant. Dans les deux cas, l'objectif est de repositionner les registres afin de mieux équilibrer les divers chemins combinatoires d'une architecture donnée ainsi que réduire le taux de *glitches* et par-là même la consommation.
Dans un deuxième temps, nous étudierons la traversée des opérateurs arithmétiques par des blocs effectuant des traitements uniformes comme les multiplexeurs.

6.4.1 Glissement *entier* de bloc

Transférer un bloc de l'entrée vers la sortie d'un registre ($E \rightarrow S$) et inversement ($S \rightarrow E$), implique un retard ou une avance d'un cycle dans la prise en compte des données par le bloc déplacé, et une latence ou une avance d'un cycle dans la disponibilité de ses sorties. Il apparaît nécessaire de déterminer quel est l'impact des glissements sur les données en entrée et en sortie des blocs déplacés ainsi que sur la gestion des signaux de commandes (sélection des multiplexeurs par exemple) et sur la gestion des signaux d'états des calculs (signe, etc.) associées à ces blocs. Nous qualifions de données tous les signaux restant internes au chemin de données. La figure 6.8 présente un double exemple de glissement ($a \rightarrow b$ et $b \rightarrow a$). Si on le considère le bloc de calcul complet, du point de vue de son interface, les modifications introduites par les deux glissements n'ont aucun impact. Les deux architectures présentées effectuent les mêmes calculs et délivrent les mêmes résultats en même temps. Les glissements n'affectent que les propriétés physiques du bloc de calcul complet (surface et temps de propagation).



(a) architecture initiale

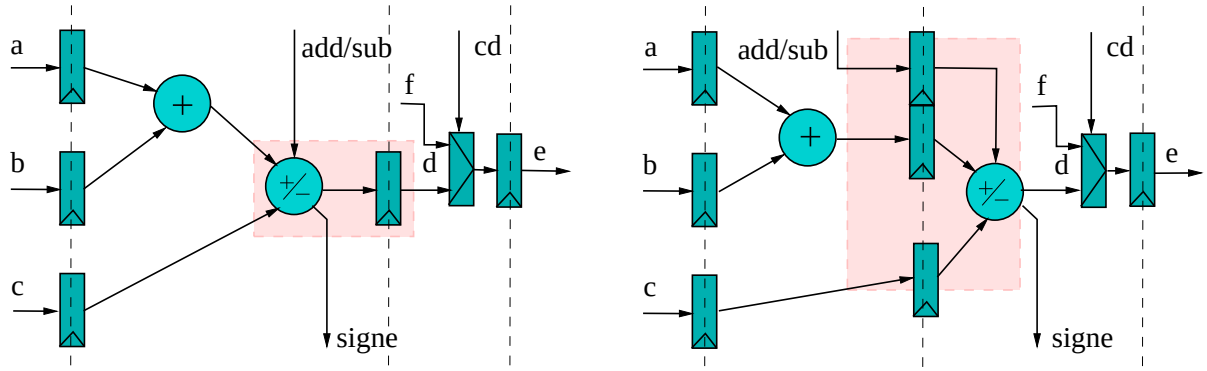
(b) architecture après glissement

Figure 6.8 – Glissements *entiers* $a \rightarrow b$ ($E \rightarrow S$) et $b \rightarrow a$ ($S \rightarrow E$) : impact sur les données

Ces résultats peuvent être généralisés. Les glissements n'ont pas d'impact sur les données en entrée ou en sortie des blocs transférés. Cette remarque est vraie dans la mesure où les données sortent où entrent respectivement dans un registre. Assurer le bon fonctionnement de l'architecture correspond juste au remplacement de la mémorisation des sorties du bloc déplacé, par la mémorisation de toutes ses entrées ($E \rightarrow S$) ou inversement ($S \rightarrow E$). Le coût matériel d'un tel transfert est fonction de l'évolution du nombre de registres élémentaires nécessaires à établir la barrière temporelle. Ce coût est proportionnel au rapport du nombre de bits/chiffres des diverses entrées sur le nombre de bits/chiffres des sorties.
Concernant les données en entrée/sortie des blocs transférés, il faut toutefois faire attention à respecter les entrées primaires et les sorties sur registres du chemin de données. Il est intéres-

sant de noter que contrairement au passage des notations à travers les registres (section 5.2), les glissements s'appliquent sans contrainte aux boucles séquentielles.

À la différence des données (E/S), l'insertion d'une latence ou d'une avance d'un cycle dans la disponibilité des signaux d'états et de commandes des opérateurs déplacés pose problème. Modifions l'architecture figure 6.8.a en substituant le second additionneur par un additionneur/soustracteur, et en ajoutant sur l'interface du chemin de données complet, la commande correspondante et l'indication du signe du résultat de l'additionneur/soustracteur (figure 6.9). Cet indicateur pourrait très bien être pris en compte dans l'élaboration de la commande cd du multiplexeur (figure 6.8). Dans ce cas, la réalisation de cd et l'application de cd sont effectuées dans la même chaîne combinatoire imposant une modification de la partie contrôle. Par répercussion le comportement du circuit et *a priori* de la partie opérative, sont aussi modifiés. Dans ce cas, le glissement ne peut s'effectuer automatiquement. Ce constat est aussi valable dans le cas de glissement de signaux de commandes de la sortie vers l'entrée d'un point mémorisant (dans notre exemple passage du multiplexeur de la sortie vers l'entrée du registre qui le précède).



(a) architecture initiale

(b) architecture après glissement

Figure 6.9 – Glissement *entier* : signal de commande

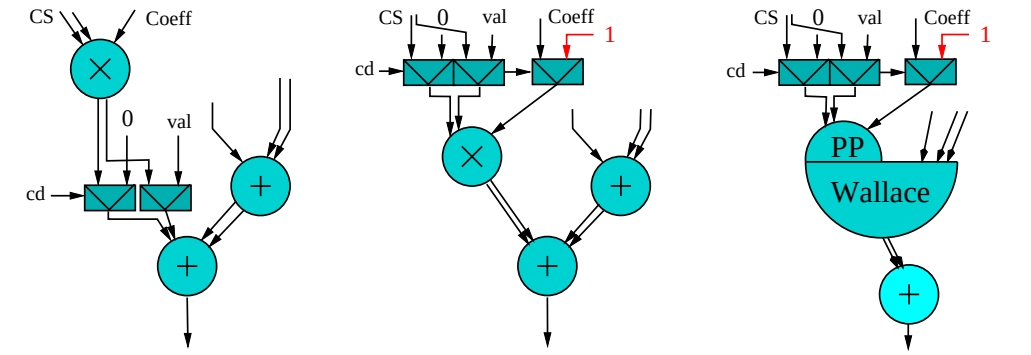
Plus généralement, l'absence de connaissance de l'utilisation des sorties du chemin de données implique de ne pas en modifier les propriétés (latence, synchronisme ou notation). Le recours aux glissements est limité aux blocs/opérateurs dont l'utilisation des résultats est connue. Ces blocs ayant plus d'entrées que de sorties, l'application des glissements se fera uniquement sur les chaînes critiques et après analyse temporelle.
Dernier point, une différentiation dans la nature des signaux (commandes, états et données) apparaît nécessaire en plus de la notation.

6.4.2 Glissement *partiel* de bloc ou absorption de registre

Les mécanismes et les limites des glissements *entiers* sont communs aux glissements *partiels*. Le raisonnement précédent est juste poussé plus loin, en permettant à un bloc d'absorber un ou des registres pour les positionner à de meilleurs endroits. Ici non plus le traitement et l'algorithme ne sont pas modifiés. L'idée est très intéressante dans le cas de blocs qui comme le multiplieur $CS \cdot CS$, intègre en interne des phases de pré-traitements et de post-traitements facilement détachables. C'est le cas des recodages, des entêtes de lignes ou de colonnes ou des conversions de sortie.
Par contre, l'absorption sous-entend une nouvelle adaptation des générateurs de blocs et la gestion explicite des blocs d'opérateurs séquentiels (*pouvant ou étant pipeliné*). De plus, utiliser ce type d'opérateur pose un problème au niveau de l'analyse de temps. Cela implique de rentrer dans les opérateurs séquentiels pour déterminer les nouveaux délais. Autre point délicat, la fluctuation du nombre de registres n'est pas connue. Ces problèmes peuvent être contournés en augmentant la complexité des modèles des opérateurs et en marquant à l'avance les opérateurs pouvant intégrer un *pipeline* (les générateurs gérant la quantité de barrières temporelles pouvant être absorbée ainsi que leurs positions). Ces solutions restent cohérentes avec notre ligne de conduite d'une optimisation au niveau opérateurs/blocs.

6.4.3 Glissement et blocs *neutres*

Une des principales qualités des registres est de ne pas modifier les signaux et par extension les codages des nombres qui les traversent. D'autres blocs possèdent cette caractéristique ou une autre tout aussi proche qui est d'appliquer à un signal un traitement complètement uniforme (multiplexeur, mise à zéro, fois -1 dans certains contextes,...). Ces deux propriétés ont déjà été exploitées dans la sous-section 5.1.2. Par rapport aux représentations des nombres, nous qualifierons ces opérateurs de *neutre*. La question qui se pose est quel va être l'impact du glissement de ces blocs sur un chemin de données. Dans cet objectif, la figure 6.10 reprend l'architecture traitant de la limitation de type opérateurs de la sous-section 5.1.2 (figure 5.2).
Le déplacement du multiplexeur de la sortie vers l'entrée du multiplieur se traduit par l'élargissement de la portion d'architecture purement arithmétique (figures 6.10.a $\rightarrow$ 6.10.b). Cet élargissement va permettre d'exploiter d'autres méthodes d'optimisations. Dans le cas de l'exemple donné les diverses additions vont pouvoir être regroupées et fusionnées avec la somme interne au multiplieur : figures 6.10.b $\rightarrow$ 6.10.c.
Matériellement parlant, le déplacement des multiplexeurs en entrée du multiplieur, se traduit par la diminution du nombre de multiplexeurs élémentaires. Soient les entrées Val, $Coeff$ et CS sur N bits ou chiffres. Dans ce cas la sortie du multiplieur en Carry-Save est sur $2.N$



(a) Architecture initiale (b) Glissement des blocs *neutres* (c) Nouvelle optimisation

Figure 6.10 – Glissement des blocs *neutres*

chiffres ($4.N$ bits). Dans ces conditions, le multiplexeur figure 6.10.a est composé de $4.N$ multiplexeurs élémentaires, tandis que les multiplexeurs figure 6.10.b nécessitent seulement $3.N$ multiplexeurs élémentaires (Les valeurs en multiple de N données ne tiennent pas compte de la dégradation possible, pour les deux architectures, d'un bon nombre de multiplexeurs élémentaires en simple porte ET ou OU). Puisque les multiplexeurs restent dans la chaîne critique, le temps de propagation global n'est pas modifié.

Plus généralement, l'amélioration en surface ne sera pas systématique. Elle dépendra du rapport de quantité de bits composant les entrées et les sorties des opérateurs traversés. De même le temps de propagation ne sera pas modifié tant que de nouvelles optimisations ne seront pas appliquées et que les contraintes temporelles sur les signaux de commandes seront respectées.

Le plus grand intérêt des glissements des opérateurs *neutres* en entrée ou en sortie des opérateurs arithmétiques est de dégager des portions d'architectures purement arithmétiques plus larges. Celles-ci permettront d'appliquer d'autres traitements assurant une meilleure optimisation du chemin de données étudié.

Un point important à respecter est que tout glissement demande de recomposer la fonctionnalité du bloc déplacé. C'est le cas dans l'exemple donné où la multiplication par 1 permet d'assurer Val en sortie du multiplieur.

Dernier point : en raison de la propriété de barrière temporelle des points mémorisants, l'élargissement par glissement des portions purement arithmétique d'un chemin de données ne concerne pas les registres. Le traitement de ces blocs leur est spécifique.

6.5 Conclusion

Dans ce chapitre, nous avons présenté un ensemble d'optimisations lié au chemin de données et à l'arithmétique en générale (regroupement, fusion et glissement d'opérateurs).

Les regroupements et les fusions exploitent l'associativité et la commutativité de l'addition. L'objectif est d'équilibrer les temps de propagations des diverses branches des arbres d'additions internes au chemin de données considéré, et par là même, réduire le temps de propagation global et la consommation des circuits. Parallèlement, ces optimisations effectuent le remplacement des additionneurs au niveaux vecteurs par des additionneurs au niveaux chiffres ou bits.
Ces deux mécanismes requièrent une analyse des temps de propagations pour être efficaces. Ils s'insèrent après *le passage en redondant partout où c'est possible*. Les gains concernent aussi bien le temps de propagation que la consommation et la surface, et apportent un meilleur contrôle de l'extension de la dynamique des données. Concernant l'utilisation des systèmes de représentations redondants, la fusion et les regroupements permettent de corriger la contre performance en délai introduite par le passage des notations redondantes à travers les registres. Ce point est très intéressant pour la généralisation des notations redondantes.
Les glissements exploitent la propriété de commutativité de certains blocs (transparence ou uniformité en terme de traitement) ainsi que la qualité de barrière temporelle dans le cas des registres. Le premier objectif est d'équilibrer les divers chemins combinatoires d'un circuit en permutant des registres et d'autres blocs afin de réduire le chemin critique du circuit considéré. Ce mécanisme est basé sur l'analyse des temps de propagations. Sa mise en œuvre impliquant une augmentation possible de la surface (rapport entre le nombre d'entrées et le nombre de sorties des opérateurs transférés). Le deuxième objectif est par le glissement de blocs *neutres* d'élargir les portions purement arithmétiques des architectures afin d'exploiter plus largement les optimisations proposées à travers ce chapitre et le précédent. Le glissement des blocs neutres s'applique en même temps que *le passage en redondant partout où c'est possible*, tandis que les glissements des registres s'insèrent après la redéfinition des enchaînements. L'exploitation de ces derniers suppose une application alternée et itérative avec les phases de regroupements et de fusions.

Remarque : dans le contexte d'un outil d'aide à la conception de chemin de données, la fusion et les glissements partiels sous-entendent l'absorption d'un opérateur par un autre. Mettre en place un tel mécanisme demande une prise en compte directe de ces mécanismes dans l'écriture des générateurs de macro-blocs. Dans le cas de l'absorption de registres, une gestion de la quantité et du positionnement des registres à l'intérieur du blocs absorbants, et une indication de la modification des temps de propagations sont impératives.

Chapitre 7

Vers un outil d'aide à la conception de chemin de données arithmétique

L'ensemble des chapitres précédents, traite essentiellement des aspects architecturaux - développements matériels et contraintes d'utilisations - liés à l'élargissement des systèmes de représentations des nombres aux notations redondantes. Ces travaux ont des impacts sur les domaines plus vastes que sont la synthèse d'architecture et la conception automatisée de circuits (chapitre 1). Il nous a alors semblé important de proposer un cadre de développement permettant la mise en œuvre automatisée de l'arithmétique mixte et des règles d'optimisations décrites précédemment.
Il ne s'agit pas pour nous de définir un outil ou une méthodologie complète - nous ne savons répondre jusqu'ici qu'aux seuls besoins de l'arithmétique - mais de présenter notre vision de ce que nous appellerons *une aide à la conception de chemins de données*, ces derniers étant le champ d'application privilégié de l'arithmétique mixte.
Les propos développés dans ce chapitre portent plus précisément sur la phase de traduction *description de haut niveau* → *description bas niveau* d'un chemin de données. L'objectif est de proposer une méthode de projection équivalente à celle utilisée dans la synthèse logique, incorporant en plus les opérateurs arithmétiques. Cette méthode doit encapsuler notre *savoir-faire* lié à l'utilisation conjointe de plusieurs systèmes de représentations des nombres. Elle doit aussi permettre d'automatiser le passage d'une description structurelle simplifiée d'un bloc à une description structurelle précise du même bloc. Pour être opérationnelle la traduction doit à la fois définir/redéfinir les ressources (essentiellement arithmétique) en fonction de notre savoir-faire, de critères de conception (surface, délai,...), d'un contexte architectural, mais aussi assurer la projection vers une cible technologique donnée.

Ce chapitre est composé de deux parties. À travers un ensemble de remarques préliminaires, la première a pour fonction de décrire plus précisément les contraintes et les objectifs de *l'aide à la conception de chemin de données* arithmétique souhaitée. Les propos correspondants sont regrou-

pés dans la section 7.1. La deuxième partie a, elle, pour vocation de détailler la méthodologie de conception retenue ainsi que les diverses composantes et mécanismes répondant aux besoins exprimés dans la section 7.1. Ces divers points sont développés de la section 7.2 à la section 7.4.

7.1 Remarques préliminaires et objectifs détaillés

Dans le contexte actuel de la micro-électronique, le point d'orgue de la conception d'un circuit est la rapidité de prototypage. Celle-ci est obtenue de plusieurs façons. En premier lieu par l'automatisation du flot de conception (traitements à effectuer/encapsulation de savoir-faire), en deuxième lieu par la mise en place de mécanismes permettant la réutilisation des développements antérieurs. L'automatisation prend la forme d'un cadre de développement propice à l'exploitation de méthodes correspondantes à divers savoir-faire. Le bon usage de ces méthodes étant défini par une méthodologie globale. Ce cadre nécessite au moins un point d'entrée et un point de sortie. La réutilisation des développements antérieurs, quant à elle, se décline au moins de deux manières différentes. La première consiste à autoriser l'association de plusieurs travaux, initialement distincts, dans un même développement. La deuxième est de pouvoir suivre les évolutions de la technologie cible employée (changement de pas technologique, modification des blocs élémentaires,...). Elle sert aussi à pouvoir passer d'une technologie cible ($FPGA$, $Full-Custom$,...) donnée, à une autre. Dans les deux cas l'objectif est de compléter, améliorer, corriger, *et caetera*, une architecture en cours de développement. L'ensemble des actions à effectuer, dépend de directives globales généralement liées aux critères physiques comme la surface ou le temps de propagation, ainsi qu'à la technologie cible employée.
Deux points essentiels apparaissent à travers ces premières remarques : premièrement, la conception d'une architecture, d'un circuit, est généralement incrémentale, et deuxièmement, sa réalisation est dépendante d'un contexte.

7.1.1 Entrées/sorties

Pour une automatisation efficace, l'entrée doit être la moins possible définie, par exemple une description comportementale de haut niveau. La sortie, elle, doit être la plus précise possible, par exemple une description structurelle bas niveau (liste des blocs élémentaires et de leurs interconnexions - *netlist* - dans une technologie cible donnée).
Dans le cadre qui nous intéresse : définition/redéfinition des ressources et projection sur une technologie cible, les phases d'ordonnancement et d'allocation des ressources de la synthèse d'architecture ont déjà été effectuées, soit par un concepteur lambda soit par un outil de synthèse de haut niveau. Dans ce cas, le point d'entrée le moins défini correspond à une description

structurelle où les ressources ne sont connues que par leurs interconnexions et par leur fonctionnalité. Toutefois, pour qu'un concepteur puisse garder le contrôle des blocs fabriqués, il faut qu'il puisse aussi donner des informations précises sur les ressources (par exemple l'algorithme à employer ou des contraintes physiques comme la taille ou un temps de propagation maximum autorisé). Il en va de même pour les interconnexions dont la nature (notation,...), la dimension, etc., doivent être contrôlable. Dans ces conditions, le point d'entrée est une description structurelle, dont le degré de définition est variable, et qui demande à être complétée. Nous parlerons de *description molle*. Cette description suppose un langage pour définir l'architecture en entrée. Dans la mesure du possible, il serait intéressant de réutiliser l'existant. Toutefois, à notre connaissance aucun des langages généraux employés ne répondent aux besoins de différenciation entre représentations arithmétiques.
Concernant la sortie, plusieurs niveaux de description sont souhaitables. Le premier correspond à une description de type RTL, c'est à dire après le complément des informations manquantes à la définition/redéfinition de l'ensemble des ressources. Le deuxième correspond à une description en blocs élémentaires, c'est à dire après la projection vers une technologie cible. Les ressources sont alors structurellement complètement décrites. Dans ce cas, la principale contrainte sur la sortie est qu'elle soit exprimée dans un langage connu ($VHDL$, $Verilog$, ...), pour être facilement interfaçable avec le reste du flot de conception (placement/routage par exemple). La sortie doit aussi pouvoir être exprimée dans le langage d'entrée car la conception d'une architecture procède d'un *recuit/simulé* (conception incrémentale). Celui-ci permet de fixer des choix architecturaux fournis/validés par les optimisations avant de continuer le développement du circuit souhaité. Comme nous le verrons en aval, deux autres raisons conduisent à la nécessaire expression de la sortie dans le même langage que l'entrée (la réutilisation de l'existant et la génération).

7.1.2 Encapsulation de notre savoir-faire arithmétique

Notre savoir-faire est décomposable en deux grandes catégories. La première correspond à notre connaissance des architectures associées aux opérateurs arithmétiques élémentaires et la seconde à l'utilisation de ces opérateurs et de l'arithmétique mixte correspondante. Ces deux expertises coïncident respectivement avec la phase de génération/projection du circuit (chaque ressource est complètement spécifiée) et avec les règles d'optimisations contextuelles qui, elles, ont pour objectifs de définir ou de redéfinir les ressources du circuit considéré (complément de la description *molle* d'entrée et/ou modification de l'architecture courante en fonction des optimisations). Automatiser ces compétences pose la double question de la forme de leur encapsulation et de leur utilisation.

Automatisation de la connaissance architecturale des opérateurs arithmétiques

Généralement, l'encapsulation de la connaissance architecturale des opérateurs arithmétiques est assurée par le recours à des générateurs paramétrables. Classiquement l'appel à un de ces générateurs se réduit au passage des informations définissant l'interface du bloc à produire (rôle, notation et dynamique, des Entrées/Sorties), à l'algorithme de calcul utilisé, à des indicateurs généraux comme les critères physiques (minimisation du délai, de la surface,...) et des paramètres plus spécifiques comme le nombre de couches séquentielles internes (*pipeline*). La sortie, elle, prend la forme d'une description structurelle bas niveau. Toutefois la nature fortement voir exclusivement combinatoire des opérateurs générés (chapitre 3), fait que leurs performances sont étroitement liées au milieu dans lequel les blocs produits seront immergés. Aussi dissocier une génération de son contexte d'utilisation réduirait sensiblement l'efficacité des opérateurs fournis. À cet effet il est nécessaire de donner aux générateurs des informations contextuelles complémentaires - essentiellement l'indication du retard de chacune des entrées. Nous parlerons ici de *génération contextuelle*.

Automatisation de l'expertise portant sur l'utilisation de l'arithmétique mixte

En premier lieu il est important de remarquer que les diverses optimisations proposées dans les chapitres 5 et 6, constituent chacune une méthode d'optimisation répondant à un contexte bien particulier. D'ailleurs, le ou les critères améliorés ne sont pas toujours les mêmes et suivant ces derniers, le gain peut-être négatif. Aucune de ces méthodes n'est générale. Pour être exploitables, ces diverses méthodes doivent être associées, encadrées, par une méthodologie globale. Cette dernière doit assurer la cohérence des optimisations en fixant l'ordre, la quantité et la répétition des traitements, en fonction de critères globaux comme la minimisation du temps de propagation, de la surface ou de la consommation. Chaque méthode va prendre la forme d'une fonction évaluant le coût de l'optimisation étudiée selon les divers critères généraux et le contexte architectural. Le choix de retenir ou non les modifications de l'architecture, introduites par l'optimisation, incombera à la méthodologie générale. Cette séparation entre les méthodes et leur(s) application(s) permettra d'élargir à tout moment la méthodologie à de nouvelles optimisations arithmétiques (gestions des notations RNS, nouveaux opérateurs,...). Elle assurera aussi l'ouverture vers d'autres problématiques associées aux chemins de données. Nous reviendrons sur la méthodologie générale d'optimisation dans la sous-section 7.1.4.
En deuxième lieu, il est important de noter que plusieurs des optimisations proposées (dédoublement des opérateurs, regroupement/fusion, glissement ou passage de registres par une ressource ou une notation,...) sont dépendantes des propriétés et des performances des ressources (et des notations) environnantes ainsi que de la ressource (notation) en cours de définition. Une

première solution pour accéder à ces informations consisterait à générer à la volée les diverses architectures possibles pour en connaître les propriétés et les performances, avant de choisir la bonne. Le coût de ces générations multiples n'est pas négligeable. Il sera d'autant plus élevé que certains traitements sont recouvrants (passage de registres et fusion par exemple) impliquant plusieurs appels à une même optimisation. À cela s'ajoute que la conception d'un circuit est généralement incrémentale entraînant aussi des modifications de contexte (déplacement de la chaîne longue, nouvelle contrainte sur les notations utilisables en un point donné,...). Cette première solution apparaît comme trop coûteuse. La réponse communément adoptée dans les outils de synthèse de haut niveau, consiste à associer à chaque ressource des modèles permettant une estimation rapide du ou des critères souhaités. Comme ceux-ci sont fortement liés à la fonctionnalité de chacune des ressources et qu'ils doivent suivre les évolutions de celles ci, leur estimation doit être la plus proche possible de la production des architectures. La notion de générateur doit donc évoluer et inclure des modèles comportementaux. Ceux-ci fourniront une première estimation des critères physiques (délai, surface,...) mais aussi une estimation des domaines de définition (dynamiques) des sorties. Cette liste est non exhaustive. Il est intéressant de noter que ces dynamiques seront identiques quelle que soit la représentation choisie. Malheureusement les modèles comportementaux sont généralement moins précis que les blocs générés et introduisent de fait une incertitude dans le choix des ressources et des notations. Ce point sera à prendre en compte lors de l'élaboration de la méthodologie globale d'optimisation.

7.1.3 Association et réutilisation de travaux antérieurs / Portabilité

La réutilisation de développements antérieurs a pour objectif d'éviter le re-développement de fonctionnalité déjà construite. Dans un contexte technologique inchangé (pas et cible technologique et outils identiques) l'utilisation d'un langage de description unique, hiérarchique pour faciliter l'intégration des blocs antérieurs, et de préférence normalisé ($VHDL$, $Verilog$,...), est une solution efficace aux problèmes soulevés.

Dans le cas où le contexte technologique a évolué, les conditions permettant de réutiliser un développement antérieur sont plus strictes. Au-delà de la compatibilité de langage, il faut pouvoir suivre les évolutions de la technologie cible employée (changement de pas technologique, modification des blocs élémentaires,...), mais aussi de pouvoir passer d'une technologie cible ($FPGA$, $Full-Custom$,...) donnée à une autre. Nous parlerons de portabilité. Une solution pour répondre à ces restrictions est d'assurer une indépendance relative entre la description et la cible technologique. Celle-ci est aisément réalisable grâce aux mêmes langages de descriptions que précédemment. Il suffit de conserver une description *comportementale* très bas niveau (équations booléennes) n'explicitant pas les blocs élémentaires utilisés.

Toutefois, le contexte technologie n'est pas le seul à évoluer. Il est plus que probable que les conditions à l'interface d'un bloc réutilisé, change d'une application à une autre (dimension des données, retard temporel des entrées) ou que les critères physiques généraux ne seront pas les mêmes. Dans ce cas une description bas niveau n'est plus adaptée. Il est donc souhaitable que la conservation d'un bloc se fasse avec une description de niveau intermédiaire, RTL par exemple, où certaines ressources comme les opérateurs arithmétiques élémentaires ne sont connus que par leur interface et leur fonctionnalité. Ces ressources seront alors synthétisées suivant le contexte architectural et les contraintes physiques générales. Une telle génération contextuelle est un des points forts d'un outil comme *Synopsys* [SYN00].
Reste que si une description au niveau RTL autorise une grande souplesse d'adaptation, elle ne permet pas d'exploiter les diverses représentations des nombres possibles et les optimisations liées à la redéfinition des enchaînements d'opérateurs arithmétiques. Les limites des langages classiques ($VHDL$, $Verilog$, ...) font que la *description molle* s'impose de nouveau. Cependant, rien n'interdit d'accepter les descriptions plus bas niveau (RTL et logique) tant que celles-ci sont structurelles. Nous recoupons ici le choix effectué lors de la définition du point de sortie du cadre de développement.
Sémantiquement, utiliser un générateur d'architecture correspond à l'appel à une ressource antérieurement définie. Aussi, les générateurs encapsulant notre savoir-faire architectural, doivent identiquement utiliser la *description molle*. À travers la réutilisation et la portabilité, nous voyons apparaître une des raisons nous ayant conduit à définir la notion de génération contextuelle.

L'ensemble des solutions proposées correspond à augmenter le niveau d'abstraction de la structure décrite réduisant, voir annihilant, son lien avec la cible technologique. Obtenir le circuit demande alors de mettre en œuvre un mécanisme de projection vers la cible visée. Cette étape de traitement nécessite de disposer des informations relatives aux cellules élémentaires (interface, performances). Sachant que les cibles possibles sont multiples, ces informations doivent être mémorisées sous une forme interchangeable et indépendante par exemple une bibliothèque. Comme les différences entre cibles peuvent être très importantes, c'est le cas des blocs élémentaires respectifs aux *cellules précaractérisées* et aux $FPGA$, la notion de bibliothèque devrait être étendue aux générateurs, et pae extension aux traitements et aux optimisations.

7.1.4 Traitements et méthodologie générale d'optimisation

Globalement l'utilisation d'une description partiellement définie, les optimisations souhaitées et la production d'une description structurelle bas niveau d'un circuit, supposent plusieurs phases de traitements. La première regroupe les actions assurant le complément, au niveau

ressource, des informations minimales manquantes à la description d'entrée (calcul et propagation des dynamiques des données, notation impérative en un point, etc.). Ces premiers traitements seront entre autres assurés par l'exploitation des modèles comportementaux associés à chaque générateur. Accessoirement cette phase va permettre de vérifier la cohérence de l'architecture décrite et indiquer si sa construction est possible (sortie primaire de boucle définie, notation demandée en point possible,...). Elle va aussi donner une indication sur les possibilités contextuelles. À la sortie de cette phase, tous les paramètres nécessaires à la génération des ressources ne sont pas encore connus notamment la nature de toutes les entrées/sorties et les contraintes de génération des ressources (délai, surface, consommation). Nous parlerons de phase préparatoire.

L'objectif de la méthodologie va être de compléter ces informations en fonction des critères généraux. Deux choix s'offrent à nous pour déterminer ces informations. Le premier consisterait à explorer l'ensemble des solutions possibles et de sélectionner celle qui cadre le plus avec les critères généraux choisis. Dans ce cas, il n'y a pas besoin des diverses optimisations proposées dans les chapitres précédents. Toutefois le nombre de solutions potentielles augmente très vite d'autant plus qu'à chaque opérateur il faudra tester toutes les combinaisons redondantes/non redondantes en entrées/sorties. Cet aspect sera à terme amplifié par l'introduction d'autres systèmes de représentations comme les RNS.

Le second choix consiste à encadrer la méthodologie en appliquant entre autres les méthodes d'optimisations présentées dans les chapitres 5 et 6. Dans ce cas, il est nécessaire de définir une base permettant l'expression des contextes avant de pouvoir appliquer les optimisations. Si l'on suit les indications des chapitres précédents, c'est l'objectif du passage en notation redondante partout où c'est possible. Viendront après les diverses méthodes proposées précédemment comme la redéfinition des enchaînements, la fusion, les regroupements ou les glissements. Il est intéressant de noter que nombre de ces méthodes sont appliquées brutalement et que leurs effets indésirables sont corrigés après. Cette constante devra se retrouver dans la méthodologie d'optimisation arithmétique.

Deux remarques :

1. Assurer la portabilité suppose de garder une représentation haut niveau du circuit. Dans ce cas les optimisations devront respecter la pérennité des notions de ressource et de nombre. Toutefois les ressources et les données décrites par la représentation de sortie ne seront vraisemblablement pas les mêmes que celles de la description en entrée. L'important c'est qu'elles soient identifiables de la même manière.
2. Il est important de noter que la méthodologie proposée ici est fortement volatile puisqu'elle

est centrée sur l'introduction de nouveaux systèmes de notation des nombres. Pour que l'aide à la conception proposé soit viable, il est impératif que la méthodologie soit séparée des méthodes d'optimisation qu'elle exploite. Elle doit aussi être ouverte au traitement des blocs non arithmétiques et à l'introduction d'autres problématiques.

7.1.5 Cadre de développement

Clairement le cadre de développement a pour objectif de fournir un support à l'application des méthodes d'optimisations et des traitements quels qu'ils soient. De même il doit assurer la pérennité et la portabilité des travaux courants et antérieurs. Enfin, il est nécessaire qu'il soit évolutif et compatible avec l'existant sous peine d'être très vite obsolète.
Globalement les traitements et les optimisations se traduisent par la modification de l'architecture courante ou par le complément de cette dernière. Dans les deux cas, le circuit proposé en entrée évolue. Les changements sont pour partie liés au contexte architectural local. Il en découle que les premières fonctionnalités devant être fournies par le cadre vont être :

1. Disposer entre les points d'entrée et de sortie, d'une représentation ajustable/modifiable de l'architecture en cours de conception. Celle-ci devra différencier clairement les ressources et les signaux. Cette représentation doit nécessairement être hiérarchique puisque les circuits décrits seront plus ou moins définis (langage mou) et qu'une partie des changements s'opèrent au niveau ressource. Elle doit aussi s'accompagner de mécanismes pouvant assurer sa construction et sa manipulation, afin d'appliquer notre savoir-faire et divers autres traitements. Puisque ces mécanismes vont être utilisés par plusieurs traitements et méthodes d'optimisations, il serait prudent de les rendre autonomes de leur utilisation.

2. Pouvoir manipuler les diverses notations arithmétiques utilisables pour représenter les nombres (classique, redondante, RNS, flottante, ...), puisque une bonne part de notre expertise est basée sur l'ajustement à un contexte architectural donné de ces notations. Comme ces dernières sont modifiables à tout moment, leur gestion doit être impérativement explicite. Plus généralement les données ne sont pas toutes des nombres. Différencier leur nature serait intéressant, à la fois pour expliciter les points où certaines notations ne sont pas exploitables (opérateur non mixte, sortie primaire,...), et pour répondre aux besoins des *glissements* (section 6.4). Dans un premier temps ce typage sera divisé en trois groupes : nombre, commande et état (au sens défini dans la section 6.4). Cette liste n'est bien sur pas exhaustive.

3. Des mécanismes d'extractions des informations concernant le contexte architectural local et l'évaluation des critères physiques, relatifs à un point du circuit étudié. De la même manière que les mécanismes précédents sont dissociés de l'instant de leur usage, l'extraction des

informations en un point de l'architecture doit être indépendante de son utilisation. Les informations seront entre autres, accessibles par interrogation des modèles comportementaux accompagnant les générateurs, après requête des traitements ou des optimisations.

4. Un mécanisme générique assurant le passage des paramètres de génération d'une ressource vers son générateur. Ce mécanisme sera accompagné d'un second permettant le rattachement de la description structurelle produite, à la représentation du circuit développé, afin de permettre sa modification ou son adaptation.

Globalement la réutilisation de l'existant et la portabilité sont assurées par l'augmentation du degré d'abstraction de la description du circuit étudié. Toutefois cette solution introduit une phase de traitement supplémentaire afin de faire le lien avec la cible technologique voulue. Ce lien est permanent puisque les optimisations et la génération de ressource sont contextuelles et donc dépendantes des propriétés de la cible technologique. Le cadre de développement doit assurer l'accès aux informations technologiques (nature, disponibilité et performances des blocs élémentaires, générateurs associés). Les cibles possibles étant multiples et mémorisées sous forme de bibliothèque, ce mécanisme sera générique.

Si l'on résume, le cadre de développement souhaité est ouvert et indépendant des traitements. Ses rôles sont multiples. Le premier est de proposer et de manipuler une structure représentative du circuit étudié. Le deuxième est d'assurer le dialogue et la propagation des informations entre les divers savoir-faire mais aussi avec la représentation du circuit étudié. Le troisième est d'assurer à travers les générateurs et la projection, la construction du circuit. La séparation des savoirs, des mécanismes de manipulation de la représentation du circuit étudié ainsi que de l'évaluation locale des architectures, de leur application va permettre à l'ensemble : générateurs, méthodes et méthodologie d'optimisation, mécanismes liés au cadre de développement et projection, de supporter les évolutions des savoir-faire (modification ou nouvelles notations possibles en E/S d'une ressource donnée, évaluation des critères,...) et permettre une optimisation contextuelle. Le recours à des bibliothèques, à une description structurelle de haut niveau (langage mou) et à la projection va lui, assurer la pérennité des développements et leur adaptabilité aux évolutions technologiques (changement de pas technologies, de blocs élémentaires, ou de cible technologique). La méthodologie générale d'optimisation et l'indication de critères généraux vont, elles, assurer l'automatisation des traitements.

Le cadre de développement, et les divers autres points abordés dans cette section : la conception des générateurs, la phase de projection, la méthodologie d'optimisation ainsi que le langage de description en entrée, vont être détaillés plus avant dans les sections suivantes.

7.2 Méthodologie de conception de générateurs et projection

Cette section s'attache à définir la méthodologie de génération et les diverses contraintes et mécanismes qui permettront de concevoir et utiliser des générateurs. Nous nous intéressons plus spécifiquement à l'inclusion de la portabilité dans la génération, à l'association de fonctions comportementales et autres, aux générateurs, et à l'environnement de la génération.

7.2.1 Génération et projection

La fonction d'un générateur est de faire coïncider à une ressource donnée, définie au minimum par sa fonctionnalité et son interface (entrées/sorties), une description structurelle bas niveau. Cette dernière doit tenir compte des critères généraux d'optimisations, du contexte physique à son interface (retards temporels des entrées, sorties sur registre, sortance, etc.), mais aussi de la nature et des performances des blocs élémentaires constituant la cible technologique visée.
Globalement les mécanismes de génération correspondent à l'application de savoir-faire architecturaux encapsulés, aboutissant à une description constituée d'une liste des blocs élémentaires employés et de leurs interconnexions (*netlist*). La description structurelle obtenue pour un contexte donné, est déterministe. La génération sera qualifiée de *synthèse d'architecture directive*. *A contrario*, la projection vers un ensemble de cible technologique, n'aboutira pas systématiquement à la même description structurelle bas niveau. Les solutions utilisées pour répondre à la variation des contraintes bas niveaux (disponibilités et performances des blocs élémentaires) sont génériques. Nous dirons de la projection qu'elle effectue une synthèse d'architecture *générique*.

Méthodologie de génération et projection

Assurer la portabilité en plus de la génération, suppose d'insérer dans le déroulement des algorithmes de construction de la *netlist*, suffisamment de latitude, pour assurer une projection optimale vers chacune des cibles technologiques voulues. L'idée directrice est de définir un niveau de description intermédiaire, indépendant de la technologie, définissant le point de passage entre la génération et la projection. Nous parlerons aussi de description *virtuelle*. Tout le problème est de déterminer le niveau d'abstraction de la description intermédiaire satisfaisant à la fois les directives de la génération et le degré de latitude nécessaire à la projection. Trop haut, les directives architecturales risquent d'être mal suivies, trop bas, la projection deviendrait déterministe et donc sous optimale pour un certain nombre de cibles.
Une solution pour maîtriser le degré d'abstraction de la représentation intermédiaire est de traiter les cibles technologiques aux propriétés trop éloignées comme les *cellules précaractérisées*

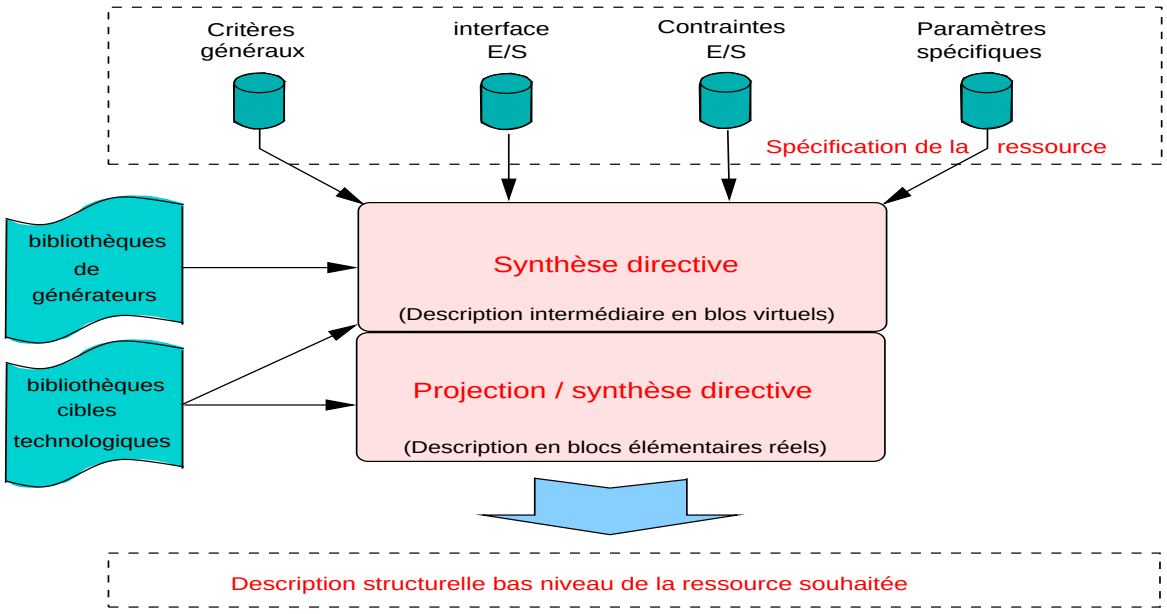


Figure 7.1 – Principe de génération et de projection

et les $FPGA$, séparément. Il est à noter que pour une même ressource et un même contexte, cet éloignement se traduit déjà par l'application pour chaque cible, d'une solution architecturale différente. Le passage par un niveau intermédiaire va être précédé par la séparation de la génération d'une ressource en plusieurs générateurs associés chacun à une famille de cible technologique.

Ainsi la forte disparité de blocs existant entre cibles, en terme de fonctionnalité et de performances, est résolue. La projection n'aura à traiter que de l'indisponibilité de certains blocs élémentaires et de la différence relative de performance entre les blocs élémentaires remplissant la même fonctionnalité. La description intermédiaire peut alors être très proche des fonctionnalités couvertes par les blocs élémentaires, ce qui assure la synthèse directive. Ce point est repris sur la figure 7.1 présentant une vision globale de la méthodologie de génération.

Mécanismes de génération et de projection retenus

Au-delà de la méthodologie de génération, il nous reste à détailler la synthèse directive et la portabilité. La première effectue le découpage de la fonctionnalité souhaitée en un ensemble de sous fonctionnalités interconnectées. Cette transcription est obtenue par le déroulement d'un ensemble d'algorithmes correspondants à un savoir-faire en terme d'architecture. Elle tient compte du contexte et des critères généraux. Les solutions adoptées sont spécifiques à la ressource souhaitée ce qui revient à dire que la synthèse directive est dédiée. La descrip-

tion structurelle obtenue définit une quantité variable de sous-blocs comme des réducteurs $4/2$ ou des fonctions logiques vectorielles (multiplexeur à k entrées, etc.). Un exemple parfait de découpage en blocs intermédiaires est celui de l'additionneur de *Sklansky*, figure 3.5 chapitre 3. C'est aussi le cas pour la quasi-totalité des autres architectures présentées dans le chapitre 3. À ce stade, la description structurelle est indépendante de la technologie cible. Elle est *virtuelle*.Toutefois les blocs intermédiaires sont *taillés* pour répondre essentiellement à une famille de cibles technologiques.

Le respect de la *synthèse directive* sera assuré en imposant aux traitements ultérieurs, tout spécialement à la projection, que les entrées/sorties des sous-blocs existent sous forme de signaux dans la description structurelle finale.

Le deuxième niveau de génération va lui assurer la projection vers la cible, en tenant compte des conditions aux interfaces des divers sous-blocs structurels, et des orientations fournies par les critères généraux. La synthèse logique se prête parfaitement à la réalisation de ces multiples projections. L'adaptation au contexte sera assurée par les optimisations booléennes et la projection sera assurée par le *mapping*. Il reste toutefois à déterminer qui assure la description des sous-fonctionnalités et quelle est la forme prise par la description des sous-blocs intermédiaires à l'entrée de la synthèse logique. Classiquement cette entrée est une description comportementale définissant des équations logiques. Toutefois, rien n'interdit de définir la fonctionnalité du sous-bloc étudié, directement par une description structurelle. L'idée va donc être d'utiliser des générateurs bas niveau. Ces derniers vont fournir une description en fonctions logiques élémentaires, que celles-ci existent ou pas dans la cible technologique (multiplexeur à 5 entrées, NON/OU à 6 entrées, etc.). La description obtenue est facilement synthétisable et est compatible avec le cadre de développement. Ces générateurs que nous qualifierons de pseudo-comportementaux seront dédiés à une famille de cibles technologiques.

Dans le cas où l'on ne souhaiterait pas utiliser la synthèse logique et maîtriser intégralement la projection, les générateurs pseudo-comportementaux devront être remplacés par des générateurs assurant une synthèse directive. Cette dernière devra être en prise directe avec les fonctionnalités et les performances des blocs élémentaires de la cible technologique voulue. Nous qualifierons de *dédiés* ces générateurs bas niveau puisque les structures produites se construiront directement en blocs élémentaires de la cible technologique visée. Il en découle qu'à chaque bibliothèque technologique sera associée une bibliothèque de générateurs *dédiés*.

Que la génération des blocs intermédiaires se fasse par synthèse logique ou par générateurs dédiés, dans les deux cas des mécanismes de dialogue avec la bibliothèque décrivant les blocs élémentaires de la technologie cible, sont nécessaires. Les objectifs sont de savoir quels sont les blocs élémentaires existants et quelles sont leurs performances et leur fonctionnalité. L'in-

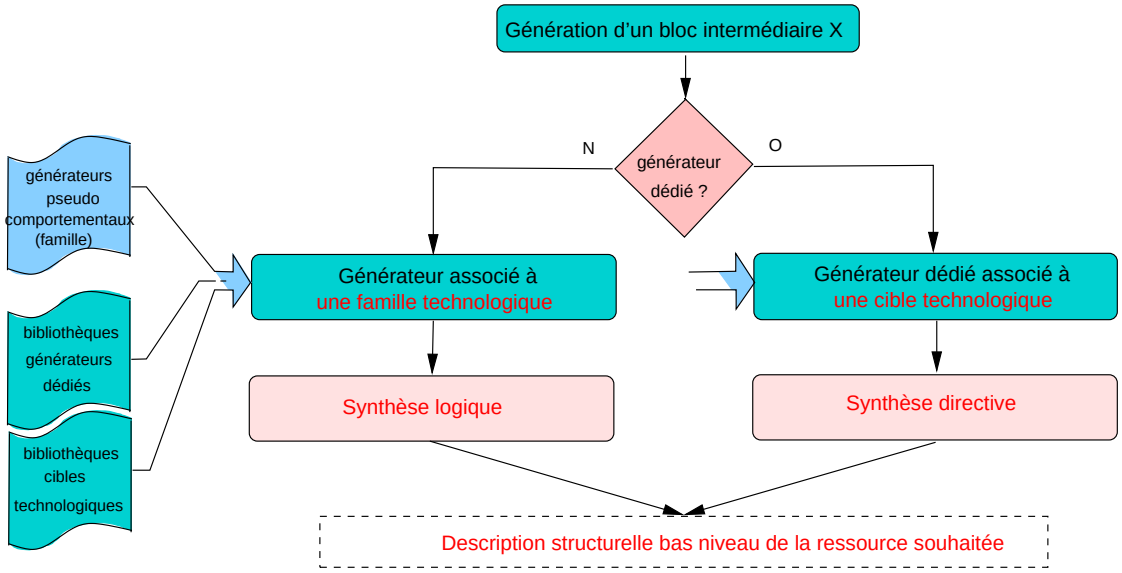


Figure 7.2 – Génération intermédiaire

terrogation des bibliothèques peut aussi servir lors de la synthèse directive, notamment pour déterminer quelle variante d'un algorithme sera utilisée. C'est le cas par exemple, pour l'opérateur somme (section 3.3) dont les paliers de réductions sont fonctions du type d'additionneurs élémentaires utilisés : $3/2$ ou $4/2$. Plus généralement ce mécanisme est à associer au cadre de développement et doit être accessible à tout moment.
Si un générateur bas niveau existe, c'est qu'*a priori* il apporte une solution spécifique. Dans ce cas il sera prioritaire sur la synthèse. Un test de son existence est nécessaire ainsi qu'un mécanisme de rattachement à la fonctionnalité qu'il décrit : figure 7.2.

L'appel à des générateurs intermédiaires ou dédiés implique que la génération est hiérarchique. Ces appels se feront aussi entre générateurs de haut niveau. Dans le cas de la construction d'un multiplieur classique, par exemple, l'apport spécifique du générateur va concerner essentiellement la construction de la matrice de produits partiels. La somme de ces derniers sera faite par l'appel à la génération d'un arbre de Wallace qui lui-même instanciera la génération d'un additionneur standard afin d'assurer la notation classique en sortie.

7.2.2 Génération et cadre de développement

Globalement ce qui nous intéresse ici, c'est de définir un peu plus précisément l'environnement de la génération et les interactions entre les générateurs et le cadre de développement. Un des premiers points à préciser est que, dans le cadre de développement proposé, les notions de

modèle et d'instance rattachées à une netlist vont évoluer respectivement vers les notions de fonctionnalité et de ressource. La différence fondamentale est que ces notions ne définissent plus des blocs dont l'interface et la structure sont statiques. Un générateur ayant pour rôle de construire une fonctionnalité donnée, il représentera un modèle. Une ressource va, elle, prendre la forme d'un contexte de génération (critères généraux, interface contextuelle,...).

Adaptation contextuelle et optimisation locale

L'adaptation contextuelle va se faire en plusieurs étapes. L'étape concernant la synthèse directive, va consister à définir l'algorithme utilisé en fonction des contraintes générales et des propriétés des entrées/sorties de la ressource souhaitée (notation, dynamique, etc.). Si l'on reprend l'exemple du multiplieur classique, le choix va se faire entre les algorithmes *direct* et de *Booth* (section 3.4). Des choix d'algorithmes seront aussi effectués dans les instances internes au multiplieur par transmission des contraintes générales. Par exemple pour la somme, l'idée est de déterminer les paliers de réduction en fonction des blocs élémentaires existants (cellule 4/2 ou 3/2), mais aussi de choisir l'ordre de prise en compte des entrées et des résultats partiels en fonction de leur disponibilité temporelle. Comme il est indiqué plus haut, concernant la synthèse logique, l'adaptation au contexte sera obtenue par la modification des équations logiques, et par le choix des blocs élémentaires utilisés lors de la phase de projection. Dans le cas des générateurs dédiés, l'adaptation contextuelle se fera par une construction des blocs intermédiaires conditionnée par les performances des blocs élémentaires et en fonction des contraintes en E/S.
L'ordre de génération des instances de même hiérarchie - dans le cas du multiplieur matrice puis somme et conversion - dépendra des critères généraux. Il en sera de même pour les hiérarchies inférieures. À chaque niveau de hiérarchie les générations seront itératives ou récursives. La définition des contraintes pour chacun des blocs se fera de la même manière que pour les ressources. Pour utiliser les mêmes méthodes d'optimisations des critères généraux, la construction hiérarchique des diverses instances (niveau ressources et blocs intermédiaires) se fait dans le cadre de développement défini précédemment. L'adaptation contextuelle et par extension la génération contextuelle ne sont possibles qu'à travers le cadre de développement.

Interface générique en entrées/sorties des générateurs

Comme le cadre assure les requêtes entre les divers savoir-faire et méthodologies, il est le seul à exécuter un générateur et donc à passer les paramètres à ce dernier. Il s'agit de concilier cet appel centralisé avec l'hétérogénéité des paramètres nécessaires à chaque générateur.
En fait, le cadre en lui-même n'a aucune action sur les paramètres d'une génération, son rôle

se restreint à fournir et manipuler une représentation du circuit en cours de construction en fonction des divers traitements effectués. Le problème se réduit donc à formater le passage des paramètres, à définir une interface générique. L'interprétation des paramètres se fera à l'intérieur de chacun des générateurs. Les positionnements/modifications des paramètres seront effectués initialement par la conversion du langage mou en entrée puis par le déroulement des divers traitements effectués.
Une solution simple consiste à associer à chaque ressource décrite en mémoire (instance) une liste regroupant tous ses paramètres. Chacun d'eux sera identifié par un code lié aux informations le concernant. Cette solution est valable à la fois pour l'interface de la ressource (entrée/sortie) et pour les paramètres plus spécifiques à la génération.
Assurer la cohérence de cet ensemble suppose de définir des codes standards pour les paramètres communs à un ensemble de générateurs ou plus généralement manipulés par les traitements. Ce sera le cas pour les entrées/sorties, les indications de contextes, les critères généraux d'optimisations. L'interprétation des codes sera faite dans les générateurs. Cette interface non figée, extensible, s'adapte très bien à la notion de langage mou. Les sorties des générateurs vont être de deux types. La première va être la description produite. Cette netlist sera rattachée à l'instance regroupant déjà la définition de l'interface d'entrée. La seconde va correspondre à la gestion des erreurs de génération.

Comportement et fonctions associées à chaque ressource

En plus du générateur lui même, chaque fonctionnalité se verra associer des fonctions assurant le complément des informations minimales nécessaires aux diverses étapes d'optimisations (propagation des dynamiques des entrées vers les sorties, notation obligatoire, ...), et des fonctions d'estimations des critères généraux (temps de propagations, ...). Ces dernières serviront à l'accélération des mécanismes de détermination des contextes. Toutefois les indications données concernant les critères seront moins précises que la génération qu'elles remplacent. De même, des fonctions d'optimisations spécifiques à la ressource comme le retour partiel en notation classique, doivent pouvoir être rattachées à une fonctionnalité. Cette dernière va prendre la forme d'un objet avec ses champs et ses méthodes.

7.2.3 Synthèse

Dans cette section, nous avons présenté une méthodologie de conception de générateurs permettant de passer d'une description haut niveau d'une ressource, à sa description sous forme de *netlist* instanciant les blocs élémentaires de la cible technologique souhaitée. Cette méthodologie se décompose en deux étapes. La première correspond au déroulement d'un ensemble

d'algorithmes, de savoir-faire architectural, aboutissant à une description constituée d'une liste de blocs structurels intermédiaires et de leurs interconnexions. La description structurelle obtenue pour un contexte donné, est déterministe, ce qui nous a amené à qualifier cette première étape de génération de *synthèse d'architecture directive*. La seconde étape assure, elle, la portabilité vers les diverses cibles technologiques existantes, soit par des générateurs dédiés spécifiques à la cible ou à la famille technologique envisagée, soit par synthèse logique. Dans ce second cas, les blocs intermédiaires à synthétiser seront produits par des générateurs pseudo-comportementaux écrits pour répondre aux besoins d'une famille de cibles technologiques précise (bibliothèques de cellules précaractérisées par exemple). Les interactions avec le cadre de développement (interface des générateurs, optimisation contextuelle, etc.), ont aussi été décrites.
La méthodologie de conception de générateurs et de projection présentée, est inspirée de travaux antérieurement [Comp92, Vauc97, Houe97] réalisés au sein du laboratoire d'informatique de Paris VI. Elle en reprend les notions de base que sont le niveau de description intermédiaire virtuelle et l'association à la bibliothèque de blocs élémentaires décrivant une cible technologique d'une bibliothèque de blocs intermédiaires dédiés à cette cible [Vauc97, Houe97].
Les mécanismes décrits dans cette section, sont particulièrement adaptée à l'encapsulation de notre expertise en terme de conception d'architectures d'opérateurs arithmétiques. Excepté l'adaptation contextuelle et l'appel par transparence à la synthèse logique, cette méthodologie de génération et de projection a été validée par la réalisation des générateurs correspondant aux divers opérateurs mixtes proposés dans le chapitre 3.

7.3 Traitements et Optimisations

Cette section a pour objectif de définir une méthodologie d'optimisation ainsi que les diverses contraintes et mécanismes qui permettront d'employer notre expertise arithmétique. En plus de la détermination de la méthodologie d'optimisation, nous nous intéressons à l'évaluation des critères généraux.

7.3.1 Méthodologie générale d'optimisation, traitements associés

La première phase de traitement à assurer est la redistribution des ressources logiques (multiplexeur, registre, mise à zéro, saturation,...) par glissement. Cela va permettre de dégager des portions arithmétiques plus importantes, propices à l'exploitation des qualités des arithmétiques non autonomes. Les ressources logiques déplacées ne devront pas modifier les propriétés des nombres ; plus précisément : soit ne pas avoir d'impact sur les codages (multiplexeur,

registre,...), soit effectuer un traitement complètement uniforme (saturation, mise à zéro,...). Cette première étape permettra de regrouper les ressources assurant une même fonctionnalité (multiplexage, addition,...), soit exploiter plus largement les propriétés d'associativité, de commutativité et de factorisation des ressources de même nature (arithmétique ou logique). La quantité d'enchaînements d'opérateurs arithmétiques va augmenter. Cette première phase s'accompagnera de la délimitation de blocs purement arithmétiques et logiques, ainsi que du marquage des premiers signaux devant être impérativement en notation classique (entrées/sorties des ressources logiques). Ce marquage sera suivi par le complément, au niveau ressource, des informations minimales manquantes à la description d'entrée (calcul et propagation des dynamiques des données, notations impératives en un point, etc.). Pour être optimale, le découpage devrait être précédé par une *mise à plat* au niveau ressource à générer (instances dont les modèles sont définis par des générateurs) de la description d'entrée. Il est souhaitable que la hiérarchie initiale puisse être partiellement ou intégralement respectée. Dans ce cas les traitements seront tout de même effectués à plat mais en garantissant le respect des signaux définissant les interfaces des divers blocs arithmétiques et logiques. Certains générateurs intègrent des appels à d'autres générateurs. Pour garder une certaine cohérence au niveau des opérateurs élémentaires utilisés, leur mise à plat ne sera pas effectuée. La fusion permettra toutefois de contourner le problème. À terme, effectuer la mise à plat de tous les générateurs incluants des appels à des fonctions arithmétiques et logiques élémentaires sera très intéressant. Les registres auront la double nature arithmétique et logique. Le glissement ne leur sera pas appliqué dans cette phase d'optimisation car le niveau d'abstraction ne permet pas de déterminer les implications sur le temps de propagation et sur la surface.

La seconde phase va correspondre à la modification des équations arithmétiques par l'exploitation de la commutativité, de l'associativité et de la distributivité (cette phase et le pendant à la phase d'optimisation des équations booléennes de la synthèse logique). Les règles utilisées auront pour objectifs de supprimer les redondances des équations, de simplifier ces équations et de réutiliser les équations existantes afin de simplifier d'autres équations proches non encore traitées. La modification des équations ira dans trois directions ; suivant les critères généraux, équilibrer les chemins (consommation), limiter les ressources (surface) ou optimiser le nombre de couche d'opérateurs (délai, consommation). Le même travail sera effectué sur les équations logiques (synthèse logique). Nous ne rentrerons pas dans le détail de telles optimisations. Elles ne font pas partie de la problématique que nous souhaitions traiter ; leur étude détaillée reste à faire. Il est toutefois important de les mentionner pour bien préciser la méthodologie d'optimisation proposée. Cette phase d'optimisation se déroule au niveau ressource à générer.

La phase suivante a pour objectif de définir plus précisément les ressources : essentiellement

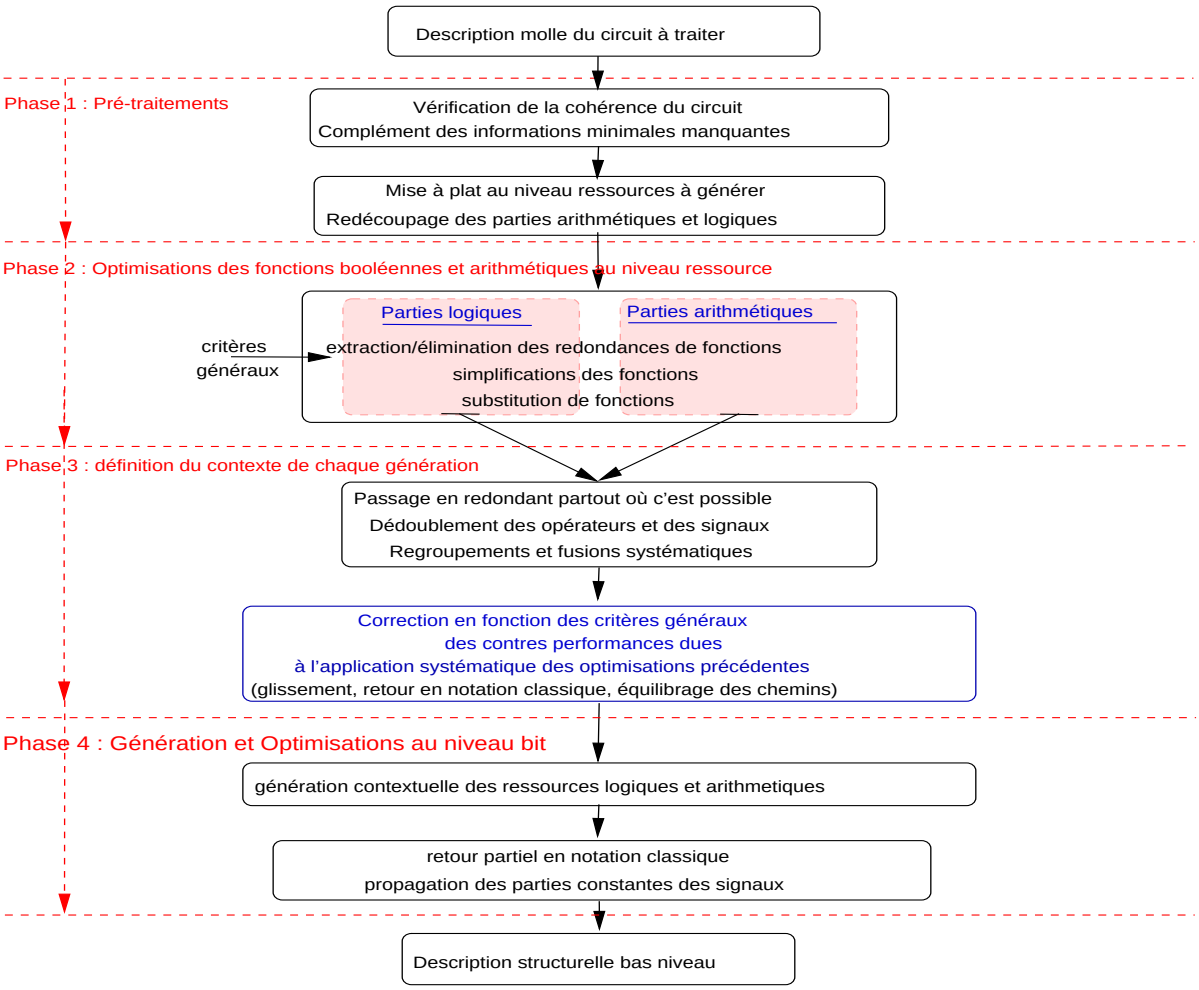


Figure 7.3 – Méthodologie d'optimisation

les notations en E/S et les algorithmes ou plutôt les critères contextuels présidant à la future génération de chacune des ressources. Elle a aussi comme but de regrouper ou de fusionner des ensembles de ressources en des ressources plus complexes dont la génération offrira de meilleures performances. Cette phase comprend les optimisations proposées dans les chapitres sur la redéfinition des enchaînements arithmétiques, le regroupement, la fusion, et le glissement des registres. Les traitements sous-entendus sont fonction des critères généraux d'optimisations et de la famille technologique cible. Ils seront aussi basés sur les fonctions de comportements accompagnant les générateurs. Les mécanismes mis en œuvre sont équivalents à la phase de mapping de la synthèse logique. Nous reviendrons en détail sur cette phase dans la sous-section suivante.

La dernière phase va, elle, effectuer la génération contextuelle des ressources. Elle sera complétée par le retour partiel en notation non redondante. Gérer cette dernière optimisation introduit

deux problèmes. Le premier est : comment appliquer cette optimisation vue que les techniques à adopter varient suivant les fonctionnalités à générer. Le second est : déterminer comment représenter les données en notation redondante partielle. La résolution de ces deux problèmes va consister à associer à chaque générateur une fonction dédiée assurant le retour partiel en notation classique de ses sorties. Cette première fonction sera complétée par une optimisation générale assurant la propagation des composantes constantes des signaux à travers le circuit et les hiérarchies (par exemple par annotation des signaux ou de portions de signaux invariants). La substitution du matériel (dégradation du matériel) dont les E/S sont partiellement ou intégralement constantes, se fera une fois que la structure du circuit sera intégralement définie. Cette solution est aussi adaptée à l'élimination des branches non utilisées (sortie ou entrée générées mais non exploitées). La conservation jusqu'à la fin des structures générées permettra d'éviter de multiples dégradations/régénérations de l'architecture. Il est important de noter que les deux dernières phases sont étroitement liées.

La figure 7.3 reprend l'ensemble des points venant d'être présenté, de la méthodologie d'optimisation proposée.

7.3.2 Application des optimisations arithmétiques

L'objectif de cette sous-section est de détailler, plus avant, l'application des optimisations arithmétiques regroupées dans la troisième phase de traitement et de préciser la méthodologie d'optimisation arithmétique.
Pouvoir optimiser une architecture demande entre autres de connaître ces performances. Cela suppose que l'intégralité des paramètres nécessaires à l'interrogation des fonctions d'évaluation des critères associées aux générateurs, soient connus. Le complément de ces informations va démarrer par la définition des points où les notations non autonomes ne sont pas utilisables (interrogation des propriétés/comportements associés aux générateurs, sorties primaires de boucles, etc.). Il sera complété par un passage automatique en notation redondante partout où cela est possible. Ce second choix s'explique par les meilleures performances intrinsèques des opérateurs mixtes et redondants. Cette étape va être complétée par le dédoublement des signaux (expression d'un signal à la fois en notation classique et en notation redondante) et des opérateurs non mixtes de faible coût (sous-section 5.1.2), et par le regroupement et la fusion des fonctionnalités compatibles (arbre d'additions, ensemble d'équations logiques successives,...). Toutes ces optimisations seront appliquées sans tenir compte de leur impact sur les critères généraux. L'architecture obtenue est probablement sous optimale, notamment à cause du passage des notations redondantes à travers les points mémorisants, de la non redistribution de ces

derniers (glissement de registre non encore effectué), et du ré-ordonnancement des entrées des regroupements/fusions non encore effectué (adaptation contextuelle).À la fin de cette première étape, seules les informations concernant les critères physiques de génération et les contraintes physiques des entrées de chacune des ressources, sont inconnues. Compléter ces informations et apporter une correction aux contre-performances introduites, va être le rôle de la seconde étape de la méthodologie d'optimisation arithmétique présentée.
Comme il est indiqué dans la sous-section 7.1.2, chaque méthode d'optimisation va prendre la forme d'une fonction évaluant le coût de l'application des traitements correspondants, en terme de critères généraux. Le choix de retenir ou non les modifications de l'architecture introduites par la méthode testée, incombera à la méthodologie d'optimisation arithmétique. Cette dernière prend la forme d'une fonction de coût dont les choix s'orienteront selon les critères généraux. Elle aura pour vocation de déterminer en fonction de ces derniers, les critères et les notations des E/S des diverses ressources. La méthodologie retenue va commencer par l'évaluation de l'ensemble des ressources avec les critères physiques généraux comme critères de générations. Les informations physiques, notamment le délai, concernant les E/S seront déduites des comportements associés à chaque générateur et propagées d'une ressource à une autre. À partir de cette première estimation, la seconde étape de la méthodologie d'optimisation arithmétique devra répertorier les points critiques correspondant aux critères généraux, et les minimiser. En fonction des critères choisis cette étape va être réitérés jusqu'à la stabilisation de l'architecture. À ce moment là, l'optimisation d'autres critères pourra être effectuée sur les parties non critiques de l'architecture. Les optimisations exploitées par cette étape de la méthodologie vont essentiellement être le glissement des registres, le retour en notation classique, l'équilibrage des diverses chaînes combinatoires et le ré-ordonnancement des arbres de fonctionnalité comme l'addition.

7.3.3 Optimisation selon les critères généraux

Le traitement en deux temps (critère principal puis critères secondaires) s'appliquera entre autres à la réduction du temps de propagation global. Une fois les délais optimisés, rien n'empêchera de réduire la surface ou la consommation des chaînes combinatoires non critiques, tant que le temps de propagation global est respecté. D'ailleurs ce temps de propagation pourra être indiqué directement comme contrainte à respecter. La réduction des délais suppose de réitérer les traitements, chaîne critique par chaîne critique, jusqu'à stabilisation de l'architecture. L'ajustement de l'architecture à chaque itération reposera sur le déplacement des registres et sur l'introduction de retours en notation classique.
Dans le cas d'une optimisation en surface l'idée de base va être de parcourir tout le circuit et de

choisir l'architecture la moins coûteuse possible à chaque ressource rencontrée. Cette solution est intéressante dans le cadre de l'arithmétique classique où la modification d'une ressource n'a pas d'impact en surface sur son environnement. C'est aussi le cas pour les ressources logiques. En fait, cela est vrai car les interfaces sont toujours les mêmes. L'introduction des notations redondantes invalide cette propriété car quels que soient les opérateurs l'architecture la moins coûteuse en surface est celle dont toutes les entrées sont en notation classique et toutes les sorties en notation redondante (par exemple dans ce contexte l'opérateur additionnant deux termes est virtuel). L'application simultanée de ces deux contraintes est impossible dans le cas d'enchaînements d'opérateurs. Concernant l'arithmétique mixte, pour réduire la surface, l'idée est de partir du constat simple que plus on diminue le nombre d'entrées redondantes d'un opérateur (les notations des sorties étant conservées) plus il sera petit (et accessoirement plus sa consommation sera réduite). Donc, le premier traitement va consister à effectuer un retour en notation classique partout où le coût en surface de l'insertion d'une conversion est compensée par la réduction de surface introduite par la propagation du changement de notation à travers les ressources rattachées à la conversion. La spécification des notations en entrées/sorties des diverses ressources s'accompagne du choix de la solution architecturale la moins coûteuse possible respectant l'interface d'entrée/sortie. Appliquer cette solution demande de parcourir l'architecture des entrées vers les sorties. Le retour ciblé en notation classique va être accompagné du glissement des registres et des blocs n'ayant pas d'influence sur les notations arithmétiques. Il est important de noter que l'additionneur et la conversion les plus intéressants en terme de surface sont obtenus avec le *Carry Ripple Adder*. Cet opérateur va sensiblement dégrader le temps de propagation du circuit. Aussi, il serait souhaitable d'indiquer un temps de propagation maximal lors des optimisations en surface.
Proposer une solution générale pour optimiser la consommation serait un peu hasardeux. Celle-ci dépend à la fois du matériel mais aussi de l'utilisation faites de celui-ci. Nous nous contenterons d'appliquer des solutions simples comme équilibrer les divers chemins combinatoires via le glissement, et chercher à employer des opérateurs dont le matériel est réduit. L'idée est de réduire la quantité de commutations transitoires et la quantité de couches combinatoires qu'elles traversent. À cet effet rester en notation redondante est intéressant car les conversions sont supprimées.

7.3.4 Évaluation des critères généraux

Comme indiqué dans la section 7.1.2, les modèles comportementaux sont généralement moins précis que les blocs générés et introduisent de fait une incertitude dans le choix des ressources et des notations ; notamment les indications de délai vont se faire directement au niveau donnée

(nombre) et non à un niveau plus élémentaire (chiffres ou bits). Il est nécessaire de prendre en compte ce problème dans la méthodologie de conception. Cela va se faire à travers les appels aux évaluations des critères.
Améliorer les estimations va simplement consister à générer les portions d'architecture où les estimations sont critiques, afin d'avoir des indications plus précises. Concernant la méthodologie d'optimisation, la première estimation sera faite au niveau nombre. Les estimations suivantes, plus locales puisqu'elles ont pour objectifs d'aider la méthodologie à traiter un point critique, pourront profiter de ce mécanisme d'estimation plus pointu. L'optimisation de l'architecture va être obtenue par raffinement successif. Ce raffinement ne sera vraiment nécessaire que pour le traitement des points critiques. Il serait intéressant que l'utilisation de ce mécanisme puisse être explicitement indiquée, localement comme les contraintes physiques, dans la description initiale du circuit.
Remarque : Les évaluations et la propagation sous-entendue des informations à travers l'architecture, devront tenir compte des portions invariantes de l'architecture introduites par les retours partiels en notation classique, ainsi que de l'élimination des branches générées mais non utilisées.

7.3.5 Synthèse

Dans cette section nous avons présenté une méthodologie d'optimisation générale intégrant la gestion de notre savoir-faire arithmétique. La base de cette méthodologie est la séparation des parties arithmétiques nécessitant une synthèse directive, des parties logiques réalisables par synthèse logique et/ou par synthèse directive. Concernant l'arithmétique le fil conducteur des optimisations va être de calquer les traitements effectués dans la synthèse logique : optimisation des équations et projection. Bien que séparées, les diverses parties doivent être construites en même temps et en respectant leur interdépendance notamment temporelle.
La méthodologie proposée comprend quatre phases. La première redistribue les parties arithmétiques et logiques dans l'optique d'élargir leurs frontières respectives. Une mise à plat aux niveaux ressources à générer sera effectuée préalablement. La seconde phase correspond à l'optimisation des équations arithmétiques et logiques. Ces deux phases travaillent au niveau ressources. La phase suivante définit plus précisément les ressources, essentiellement les notations en E/S et les critères contextuels présidant à la future génération de chacune des ressources. Cette phase commence par une première spécification intégrale de toutes les ressources. Excepté les retours en notation classique et les glissements, l'ensemble des méthodes d'optimisations présentées dans les chapitres 5 et 6 seront appliquées, mais sans en corriger les effets négatifs comme les contre-performances temporelles et l'augmentation de la surface liée à la

traversée des registres par les notations redondantes. Cette première étape sera complétée par la correction des contre-performances en fonction des critères généraux choisis. La dernière phase va, elle, effectuer la génération contextuelle des ressources. Elle exploite des optimisations au niveau le plus élémentaire comme le retour partiel en notation classique. Les deux dernières phases sont étroitement liées. Elles sont le pendant de la phase de projection de la synthèse logique.
Dernier point, il est important de ne pas oublier que l'application des optimisations dans la deuxième phase est sujette à mutation, car elle ne prend pas encore en compte les systèmes de notation RNS.

7.4 Cadre de développement

L'objectif de cette section est de détailler plus avant le cadre de développement, plus spécifiquement le traitement des problèmes liés à la prise en compte de la multiplication des représentations des nombres, à la méthodologie de projection basée sur la génération et aux implications de la notion de descriptions *molles*.
Comme nous l'avons indiqué en début de ce chapitre, notre objectif se limite à déterminer l'impact de la mise en œuvre, de l'utilisation, de l'arithmétique mixte sur un outil d'aide à la conception de chemins de données. Aussi, le lecteur ne trouvera pas dans cette section de description de structure de données ou le détail des méthodes associées.

7.4.1 Vision générale du cadre de développement

Si l'on résume, les précédentes sections de ce chapitre, le cadre de développement apparaît comme une structure de données représentative de l'architecture du circuit en cours de développement, à laquelle sont associées diverses méthodes. Celles-ci ont pour rôle de construire (compléter), manipuler et parcourir la structure. Le nombre de ces méthodes n'est pas limité. Au-delà des fonctions classiques d'ajout, de retrait et de complément des informations des signaux, des instances ou des modèles, on pourra citer des fonctions de parcours en profondeur, en largeur (estimation des critères) et par coloration (détection de boucles), des fonctions permettant la mise à plat locale ou globale des hiérarchies (élargissement des portions arithmétiques) ou redéfinissant des hiérarchies (redécoupage en parties logiques et arithmétiques). Cette liste n'est bien sure pas exhaustive. Toutefois bien qu'ouvert, le cadre de développement est et doit rester indépendant des traitements liés aux optimisations et à l'évaluation de l'architecture étudiée.
La structure représentative souhaitée correspond à une description structurelle hiérarchique

pouvant s'adapter au concept de langage mou ou plutôt de description molle utilisée en entrée et dans les générateurs. Ce type de description demande de disposer des objets minimums que sont les modèles (générateurs), les instances (ressources) et les signaux (nombre ou plus généralement les données). L'aspect hiérarchique suppose que les modèles et les instances aient une interface assurant les points d'attaches des connexions. Globalement, ces objets composent le minimum nécessaire à la description d'une netlist *molle*. Toutes ces entités sont variables et ajustables en fonction des critères de traitements et des contextes. À cet effet, chacune devra supporter des indications contextuelles.

7.4.2 Impact de l'arithmétique mixte sur les éléments de base de la description

Impact sur les signaux :

Pouvoir gérer en parallèle plusieurs représentations possibles des nombres, demande une évolution sensible de la notion de signal. Une des premières informations à ajouter, est la distinction de la nature des signaux (état, commande, donnée pure) afin de restreindre l'ensemble des notations utilisable par le signal considéré. Cette information servira à marquer les points de l'architecture étudiée, devant être impérativement en notation classique (boucle séquentielle, glissement, opérateur non mixte,...). Elle sera accompagnée par l'indication de la représentation utilisée (CS, BS, NR, RNS, $constante$,...). Ces indications sont le pendant aux limites - détection du signe, dépassements de capacité, opérations logiques impossibles,... - des notations décrivant une arithmétique non autonome (RNS, redondant).
Tous les signaux ne définissent pas la même dynamique. Dans le cas des nombres classiques, cette variable est représentée par une valeur en nombre de bits ou par l'indication des bits de poids fort (MSB) et de poids faible (LSB) ce qui est très pratique pour accéder à un sous-ensemble d'un nombre. Toutefois, les *nouvelles* représentations ne sont pas codées sur des bits mais sur des chiffres et même un ensemble d'entiers dans le cas des RNS. Cette solution doit être adaptée par exemple en indiquant l'espace de définition du nombre considéré. L'indication de la valeur maximale et de sa valeur minimale va répondre à ce besoin. Ces deux indicateurs serviront aussi à exprimer la dynamique des autres types de signaux comme un vecteur d'adresse ou les commandes d'un multiplexeur à K entrées. Enfin il faudra aussi indiquer le signe du nombre.
Un problème connexe introduit par la multiplication des notations est la difficulté d'accéder à une partie d'un signal. Ce cas n'est possible, n'a de sens, que pour les nombres définis par une arithmétique autonome. Par exemple, le découpage direct d'un nombre en deux portions - poids forts/poids faibles - en notation classique et redondante ne donnera pas les mêmes résultats. Pire, suivant les retenues conservées (notations CS et BS) le découpage des divers

codes redondants possibles pour une valeur donnée, ne donnera pas les mêmes valeurs pour les deux ensembles. Dans ce cas la portion d'un nombre va prendre la forme de l'indication des bornes de l'intervalle de bits considérés et ne sera possible qu'avec les notations autonomes.
Le fait de pouvoir utiliser plusieurs notations implique qu'un nombre peut-être représenté suivant plusieurs codages. Chacun de ces codages est spécifique. Leurs composants élémentaires seront tous des bits auxquels seront associés des informations complémentaires comme un temps de propagation ou la notion d'invariance (portions constantes). Le cadre de développement doit donc posséder des fonctions de construction et de manipulation des codages potentiels, et des fonctions assurant la propagation des informations complémentaires.
Il n'est pas rare que dans un circuit, l'indication du signe, de la nullité ou du dépassement de capacité soit nécessaire. Une solution pour gérer tous ces indicateurs liés aux états, est de regrouper leur gestion dans un opérateur dédié. Une des principales propriétés de cet opérateur va être de forcer la représentation du nombre (signal) à son entrée à une notation classique (arithmétique autonome). Dans bien des cas, cet opérateur se résumera à une simple connexion d'un des bits d'entrée. Un autre intérêt de cette gestion explicite des signaux d'états est de permettre de s'adapter facilement aux changements de dynamique des nombres. Nous rajoutons ainsi la notion d'échelle (*scaling*) à la portabilité et la réutilisation.

Impact sur les modèles et les instances :

Un autre point à spécifier est la gestion de la diversité des modèles : blocs élémentaires, hiérarchies et générateurs. Une solution simple va consister à considérer l'ensemble des modèles comme des générateurs en leur associant des fonctions comportementales élémentaires. Les fonctions en question indiqueront seulement que les entrées/sorties d'un modèle seront figées en quantité, en taille ou en notation suivant les cas (complément automatique des tailles et des notations de la description). Le générateur sera statique. Dans ce cas et concernant toute la phase de construction d'un circuit, le cadre s'adressera uniquement à des générateurs et les mécanismes associés seront toujours les mêmes.
Le fait de travailler avec des générateurs permet de disposer de ressources dont le nombre d'entrées est variable comme la somme (addition) ou les fonctions logiques basiques (ET, OU, etc.), soient tous les opérateurs associatifs. Cette notion pourra être étendue aux fonctionnalités offertes par une ressource comme les unités arithmétiques et logiques. Les diverses fonctionnalités seront passées comme paramètres de génération en une ou plusieurs fois.

Remarque plus générale :

Deux des contraintes exprimées dans les sections précédentes sont le respect des hiérarchies et le respect de la notion de nombre (notamment au niveau ressources), afin de garder les infor-

mations utiles à la conservation du développement en cours, pour son éventuelle réutilisation.

7.4.3 Synthèse

Globalement, le cadre de développement est constitué d'une structure évolutive qui servira de représentation des circuits à réaliser, et de méthodes permettant de la manipuler. Ce cadre supporte toutes les contraintes et les indications assurant l'utilisation des diverses représentations des nombres mais aussi la génération contextuelle et l'aspect incomplet de l'architecture. À cet effet, les éléments constitutifs de la structure représentative (instances, modèles, signaux) sont composés d'une partie minimale et de champs complémentaires.
Le rôle du cadre est de manipuler la structure du circuit et de stocker les informations suivant les besoins de son environnement (chargement d'une architecture, optimisation,...).

7.5 Langage de description *mou*

Il ne s'agit pas ici de spécifier complètement un langage de description structurel de circuit, mais plutôt de définir les fonctionnalités minimales correspondantes aux contraintes exprimées à travers l'ensemble des sections précédentes. Nous parlerons d'un *proto-langage*.

7.5.1 Contraintes

Le langage souhaité doit couvrir les besoins nécessaires à la description d'une *netlist* hiérarchique, et à la prise en compte de notre expertise arithmétique (notamment la multiplication des notations des nombres). Classiquement une *netlist* hiérarchique est composée de modèles, de l'instanciation de ces modèles et de la liste des interconnexions entre ces instances. L'aspect hiérarchique suppose que les modèles et les instances aient une interface assurant les points d'attaches des connexions (signaux) entre instances. Concernant le cadre de développement, les générateurs de tout niveau sont associés à la notion de modèle, les ressources à la notion d'instance et les nombres - plus généralement les données - à la notion de signal. L'ensemble de ces entités définit des objets variables et ajustables en fonctions des critères des traitements et des contextes.
À cela s'ajoutent les contraintes introduites par l'aide à la conception souhaitée. La première d'entre elle (sous-section 7.1.1, est que la description initiale d'un circuit doit être la moins possible définie, mais dans le même temps doit permettre de spécifier très précisément une portion donnée de l'architecture souhaitée. Cette remarque nous a amené à définir la notion de *langage mou*. La deuxième est que la nature des données (nombres, commandes, états,...) doit être explicitement manipulable.

7.5.2 Proto-langage

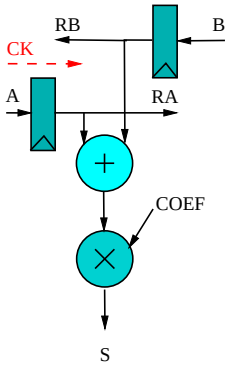
Le proto-langage est décomposable en deux parties. La première correspond à la forme générale prise par la description d'un circuit, soit le langage le moins précis possible permettant de décrire la structure d'une architecture. La seconde partie sert à préciser les champs complémentaires des divers objets assurant la description d'un circuit. La somme de ces deux parties va garantir la notion de langage mou.

Langage élémentaire

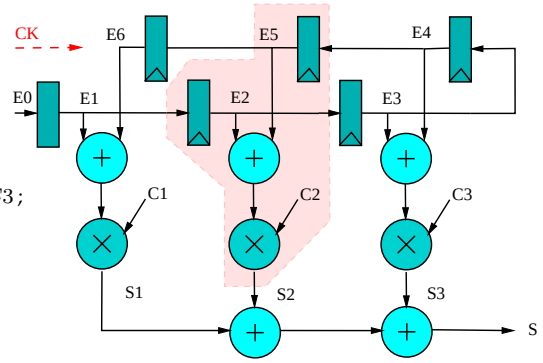
L'objectif ici, est de présenter un langage minimum assurant une description structurelle hiérarchique. La figure 7.4 présente un exemple de description de circuit. L'aspect hiérarchique est assuré par des délimiteurs de début et de fin de description (*DebutModele*, *FinModele*), les ports d'entrées/sorties (*Connecteur*) et par l'instanciation de modèles (de générateurs). Cette dernière peut prendre deux formes : celle d'un appel encapsulé par une fonction dédiée (*Op*) ou d'un appel implicite à un modèle à un générateur en utilisant un code dédié (déclaration des additions $a + b + c$,...). On retrouve dans le passage des paramètres les codes définis dans les sections précédentes. Ces codes sont volontairement explicites.
Comme l'illustre les trois instances $AddMul$, l'assignation des E/S entre modèle et instance, peut se faire par affectation directe du signal externe au signal interne, ou implicitement en suivant l'ordre de déclaration des E/S du modèle. La désignation du type de l'entrée par code, permet de mélanger l'ordre de déclarations, d'effectuer celles-ci par morceau et permet de mêler les deux types d'affectations dans la déclaration d'une même instance. Il est intéressant de noter que le contexte extérieur n'ayant pas été défini, cette architecture peut aussi bien définir un filtre avec des coefficients constants qu'un filtre dont les coefficients sont des variables. La description fournie est directive mais reste suffisamment indéfinie pour assurer la portabilité.
Actuellement l'architecture décrite sur la figure 7.4 n'est pas générable. En effet, il faudrait au minimum indiquer la taille et la nature des entrées du modèle *FIR_SYM_3*. Plus généralement, ces indications ne seront pas systématiquement nécessaires. Elles dépendront des signaux devant être impérativement définis (sorties primaires des boucles, entrées du circuit,...).
Un point important à souligner est l'absence de nom d'instance notamment pour les déclarations implicites de ressources ($a + b$,...) ; ceux-ci peuvent être produit automatiquement lors de l'interprétation de la description. Dans le cas des déclarations explicites le *nom* peut-être ajouté dans la liste des paramètres (*Nom :add1* par exemple).
Les fonctions présentées définissent le minimum utile à la description d'une netlist hiérarchique. D'autres fonctionnalités pourront être ajoutées, par exemple pour manipuler les signaux (typage, fusion, concaténation, etc.).

```
DebutModele(AddMul);
    Connecteur(Entree :COEF,A,B);
    Connecteur(Horloge :CK);
    Connecteur(Sortie :S,RA,RB);
    Op(Modele :registre ;Entree :A ;Horloge :CK ;Sortie :RA);
    Op(Modele :registre ;Entree :B ;Horloge :CK ;Sortie :RB);
    S = (RA + RB ) x COEF ;
FinModele(AddMul);
DebutModele(FIR_SYM_3);
    Connecteur(Entree :C1,C2,C3,E0);
    Connecteur(Horloge :CK);
    Connecteur(Sortie :S);
    Op(Modele :AddMul ;Entree :C1,E0,E5 ;;Sortie :S1,E1,E6);
    Op(Modele :AddMul ;Entree :C2,E1,E4 ;Sortie :S3,E2,E5);
    Op(Modele :AddMul ;Entree :C3 => COEFF ;Entree :E2,E3 ;
    Horloge :CK ;Sortie :S2,E3,E4);
    S=S1 + S2 + S3;
    /* en plus directif */
    /* Op(Modele :addition ;Entree :S1,S2,S3 ;Sortie :S) ;*/
FinModele(FIR_SYM_3);
```

(a) Description élémentaire



(b) Addition multiplication élémentaire



(c) Filtre numérique symétrique à six coefficients

Figure 7.4 – Proto-langage : exemple de description d'un circuit

Indications supplémentaires/spécification des contraintes

Rajouter des informations aux signaux ou aux instances suppose de compléter la description minimale proposée en amont. Une solution simple va consister à rajouter les paramètres à coté de chaque signal ou de chaque instance, comme indiqué sur les exemples suivants :

Signal :	E	$\rightarrow$	E{Notation :NR ;Signe :+ ;...}
Instance implicite :	$+$	$\rightarrow$	$+${Nom :op_x ;Algo :Ripple ;Delai :4ns ;...}

Le codage va reprendre la forme d'un code suivi de l'information voulue. Dans le cas des instances explicites (*Op*), il suffira d'ajouter l'information souhaitée dans la liste des déclarations. Comme on peut le voir sur l'exemple précédent, les contraintes générales peuvent être indiquées directement dans la déclaration d'une instance. Pour les contraintes du circuit, les paramètres pourront être définis lors de la déclaration du modèle (*DebutModele*) ; cette pos-

sibilité permettra ainsi de définir des valeurs par défaut pour les modèles y compris les entrées/sorties :

délémiteur : DebutModele() → DebutModele(Delai :8ns ;Surface :Min ;...)

Une seconde solution consisterait à ajouter au langage d'entrée des fonctions dédiées assurant le rajout ou la modification des informations concernant un signal ou une instance. De telles fonctions sont déjà nécessaires à la construction de la représentation du circuit développé. Dans le cas des instances cela suppose qu'elles possèdent un nom pour l'adresser correctement.
Si cette solution est intéressante pour compléter ou modifier le contexte de génération elle implique aussi la capacité de modifier le port d'entrées/sorties des instances. Dans le cadre de la génération cette fonctionnalité est très intéressante car elle permettra d'employer des générateurs intégrant plusieurs fonctionnalités comme les *unités arithmétique et logique/ALU* mais aussi facilitera la déclaration d'opérateurs ayant un nombre variable d'entrées comme la *somme*. Les fonctions sous-jacentes seront aussi utiles lors de la phase d'optimisation. Une forme possible pour ces déclarations serait :

Signal : Signal(Nom :E ; Notation :CS ;...)
Instance : Op(Nom :op_x ; Surface :Min ;...)

Dans le cas des instances, on retrouve la forme encapsulée de déclaration précédente. La manipulation de ces déclarations devra se faire avec attention.
Dernier point, si dans la description d'entrée, des paramètres sont positionnés c'est que l'utilisateur souhaite qu'ils soient respectés. Une solution simple pour assurer la pérennité de ces paramètres est de jouer sur le code utilisé ; par exemple une notation indiquée en majuscule sera impérativement respectée alors que le même code en minuscule pourra être modifié. Une autre solution est d'ajouter à chaque code un caractère spécifique comme le souligné (*_CS*).

7.5.3 Langage et génération

Assurer la portabilité, et l'optimisation contextuelle des blocs générés, supposent que le langage de description utilisé dans les générateurs offre les mêmes possibilités que le langage de description en entrée de l'aide à la conception. Toutefois les générateurs déroulant des algorithmes, le langage de description doit être en plus procédural. Ajouter cette fonctionnalité va complexifier fortement le langage à développer et son interprétation. Une solution simple va

consister à utiliser le même langage de programmation (C, C++,...) que pour l'aide à la conception, pour écrire les générateurs ; l'aspect procédural et les fonctions élémentaires sont alors assurés par ce langage.
Dans ce cas, le langage de génération est composé, définit, une partie des fonctions de construction liées au cadre de développement. L'interprétation directe d'équation ne sera pas assurée et les instances (ressources) seront explicitement déclarées. L'aspect mou du langage, le passage des indications non obligatoire, prend exactement la même forme que dans la sous-section 7.5.2 proposant un premier langage. Les primitives précédantes vont devenir des fonctions ; le format des paramètres sera quasiment identique puisque les fonctionnalités à fournir seront les mêmes. Une forme possible pour les principales de ces fonctionnalités, est donnée ci-dessous :

DebutModele(Surface :Min)	→	int DebutModele("Surface","Min",...,0)
FinModele()	→	int FinModele()
Connecteur(Entree :*A*,...)	→	int Connecteur("Entree","A",...,0)
Op(Nom :op_x ;Modele :AddMul,...)	→	int Op("Nom","op_x","Modele","AddMul",...,0)
...	→	...

Dans la section 7.2, il est indiqué que l'interface d'un générateur est composée d'une liste de paramètres indifférenciés. Chacun de ces paramètres étant composé d'un code et de l'information voulue. Cette spécificité se retrouve dans les primitives données ci-dessus. La forme prise correspond à une succession de chaînes de caractères.

Remarque : dans le souci de séparer un savoir-faire de son application - ici la génération - la déclaration d'une ressource n'implique pas sa génération immédiate. L'appel à un générateur est encapsulé. Il se fera par le cadre de développement, après détermination du contexte.

7.5.4 Synthèse

Dans cette section nous avons présenté les bases de ce que pourrait être les langages de description en entrées de l'aide à la conception et du langage de génération. Le premier est un sur ensemble des fonctions élémentaires de construction de l'aide à la conception, qu'il faudra interpréter. Le second correspond, est plus proche, des primitives de construction de la représentation manipulable associée au cadre de développement. Il est procédurale.
Les langages proposés couvrent les besoins nécessaires à la description d'une *netlist* hiérarchique, et à la prise en compte de notre expertise arithmétique, notamment la multiplication

des notations des nombres. L'aspect mou ou indéfini, prend la forme de paramètres optionnels pouvant être ajoutés indifféremment aux instances ou aux signaux.
Globalement les deux versions du langage de description structurel proposé, sont ouvertes aux évolutions des méthodologies traitées.

7.6 Vision globale de l'outil de conception

La figure 7.5 présente le principe de l'aide à la conception de chemins de données proposée. Les divers composants développés dans les sections précédentes sont présents comme la génération/projection, les optimisations, l'évaluation des critères, le cadre de développement et le langage *mou*.

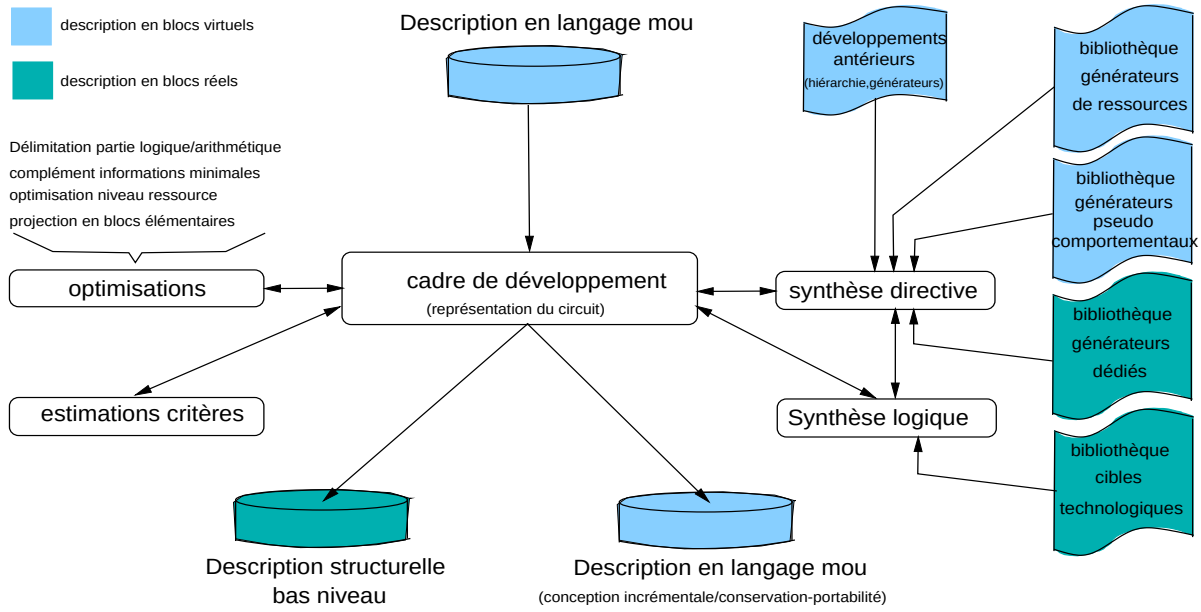


Figure 7.5 – Principe de l'outil d'aide à la conception

À terme, l'objectif de cette aide à la conception de chemin de données est de réaliser la synthèse des portions arithmétiques de circuits ou de blocs fortement calculatoires. Les traitements appliqués se feront parallèlement et de façon interactive avec les blocs purement logiques (génération contextuelle), après séparation des parties logiques et arithmétiques. Comme pour la synthèse logique, la synthèse arithmétique sous-entendue se décompose en deux étapes : traitements des équations arithmétiques et projection. Toutefois, les traitements se font ici au niveau vecteur. La différence de complexité introduite est amplifiée par la prise en compte de la diversité des systèmes de représentations des nombres intégrée dans l'arithmétique mixte. Dans les sections précédentes, nous nous sommes essentiellement intéressés à la seconde étape

de cette synthèse.
En fait le positionnement d'une telle aide à la conception dans le flot de conception va se faire aux côtés, en complément, de la synthèse RTL comme pour la synthèse d'automate. Le langage mou n'est qu'une *extension* d'une description RTL permettant de gérer explicitement les divers systèmes de notations.

7.7 Conclusion

À travers ce chapitre, nous avons présenté la définition d'un outil d'aide à la conception de chemin de données permettant une mise en œuvre automatisée de notre expertise de l'arithmétique en micro-électronique.
Cette aide à la conception est composée d'un cadre de développement fournissant un support à l'exploitation d'un ensemble de traitements et de méthodes d'optimisations des chemins de données. Le cadre proposé prend la forme d'une structure de données assurant la représentation d'un circuit, et d'un ensemble de mécanismes permettant de manipuler à volonté cette dernière. Plus précisément ces mécanismes ont pour rôle d'assurer le dialogue et la propagation des informations à travers la représentation du circuit en cours de développement et entre les divers savoir-faire. Ils permettent aussi la construction du circuit par l'entremise des bibliothèques de cibles technologiques et de générateurs de ressources associés. Volontairement, le cadre proposé est indépendant des traitements et des optimisations afin que les modifications ou les évolutions de la problématique couverte ne nécessite pas sa continuelle adaptation.
Les optimisations prennent la forme, elles, de méthodes évaluant l'impact de leur application sur l'architecture en cours de développement. Le choix de retenir ou non les modifications apportées par une optimisation, incombe à une méthodologie générale d'optimisation. Cette dernière a essentiellement pour rôle d'assurer la cohérence des traitements en fonctions des critères généraux de conception. Cette séparation entre les méthodes et leur(s) application(s) a aussi pour vocation de permettre d'élargir à tout moment la méthodologie à de nouvelles optimisations arithmétiques (gestions des notations RNS, nouveaux opérateurs,...), à d'autres problématiques associées aux chemins de données, et ultérieurement à l'encapsulation de savoir-faire liés à la construction des parties de contrôles.

Les travaux présentés dans ce chapitre ont amenés à la définition d'une méthodologie de conception de circuit basée sur la générécité et sur ce que nous appelons une description *molle* des architectures à réaliser. Cette méthodologie générale s'est accompagnée de la définition de méthodologies plus spécifiques liées à la génération/projection contextuelle de ressource haut niveau comme les opérateurs arithmétiques, et à l'optimisation arithmétique de chemins de

données fortement arithmétiques. Ces concepts ont été exploités et validés à travers la réalisation des divers générateurs d'opérateurs mixtes du chapitre 3 [Dumo00b], et de plusieurs applications comme une transformée cosinus discrète [Chot00], une unité de calcul de distance [Dumo01] ou le filtrage numérique en sortie d'un convertisseur analogique numérique $\Delta\Sigma$ [Abou00, Dumo00a, AbouXX].

Au-delà des optimisations arithmétiques liées à l'utilisation simultanée de plusieurs systèmes de représentations des nombres, la fonctionnalité assurée par l'aide à la conception proposée est proche de la phase de projection interne à la synthèse logique (*mapping*). La conversion *description structurelle haut niveau* $\rightarrow$ *description structurelle bas niveau* proposée incorpore en plus les opérateurs arithmétiques élémentaires. À terme, la segmentation proposée en parties logiques et arithmétiques permettra d'introduire des méthodes d'optimisations des équations arithmétiques élémentaires. Ces méthodes seront le pendant des optimisations booléennes incluses dans la synthèse logique.

Conclusions et perspectives

Conclusions

L'objectif principal de cette thèse était d'étudier l'impact de l'introduction de nouveaux systèmes de représentations des nombres, plus précisément les nombres redondants, dans le flot de conception $VLSI$ des cœurs de calculs. Cette introduction s'est faite en trois étapes.

La première a consisté à mettre en place une arithmétique mixte permettant l'usage conjoint des notations conventionnelles et des notations redondantes. Cette étape a donné lieu au développement d'opérateurs correspondants aux opérations élémentaires minimales nécessaires à l'utilisation de l'arithmétique mixte dans la conception de circuits (addition, somme et multiplication). Afin de démontrer le potentiel de l'arithmétique mixte, ces réalisations ont été complétées par une étude comparative des performances intrinsèques des nouveaux et des anciens opérateurs (respectivement les opérateurs mixtes/redondants et classiques).
Une fois les besoins minimums assurés (cohérence entre notations et disponibilité des opérateurs élémentaires), nous avons étudié l'impact du recours à l'arithmétique mixte dans la conception de chemins de données. Cette deuxième étape nous a permis de confirmer l'intérêt de recourir aux notations redondantes ainsi que les limites de leurs utilisations. Parallèlement cette étude a donné lieu à l'énumération de règles d'optimisations générales et automatisables, liées à l'usage des notations redondantes et à l'arithmétique plus généralement. Ces règles ont pour principales actions de redéfinir les notations et les opérateurs utilisés ; elles s'apparentent à celles employées dans la phase de projection vers une cible technologique interne à la synthèse logique.
La troisième étape a été consacrée à la gestion explicite de plusieurs systèmes de représentations dans la conception de haut niveau et plus spécifiquement là où ils sont employés, soient, dans les chemins de données. Cette étape s'est concrétisée par la proposition d'une *aide à la conception de chemin de données*. Celle-ci prend la forme de la spécification d'un outil incluant notre savoir-faire arithmétique tant au niveau des opérateurs qu'au niveau des règles d'optimisations. Cette spécification est accompagnée d'une méthodologie de conception basée sur la générécité et sur ce que nous appelons une description *molle* des architectures à réaliser. Le cadre de conception présenté, est volontairement plus large que nécessaire. Il est ouvert à l'introduction de nouveaux systèmes de notations (arithmétique ou non) et nous l'espérons à d'autres problématiques comme l'insertion de matériel de test. Cette ouverture doit aussi

permettre de faire évoluer rapidement les règles d'optimisations et d'estimations des divers critères, proposées.
La fonctionnalité assurée par l'aide à la conception proposée est la conversion d'une description structurelle haut niveau en une description structurelle bas niveau à base de blocs élémentaires d'une cible technologique choisie. Cette conversion, s'apparente à la phase de projection interne à la synthèse logique. Elle incorpore en plus les opérateurs arithmétiques élémentaires. Cet ajout passe par la délimitation des portions arithmétiques de l'architecture et par l'application à ces portions de traitements spécifiques (optimisations et synthèse directives). Les portions arithmétiques et logiques sont traitées en parallèle et de façon interactive. À terme, la segmentation proposée en parties logiques et arithmétiques permettra d'introduire des méthodes d'optimisations des équations arithmétiques élémentaires. Ces méthodes seront le pendant des optimisations booléennes incluses dans la synthèse logique.

Perspectives

Arithmétique

Bien que nous l'espérons intéressant, le travail d'introduction de nouveaux systèmes de notations aux cotés des systèmes de représentations classiques effectué s'est limité aux systèmes redondants. Dans les deux chapitres introductifs de cette thèse, il est fait mention d'au moins deux autres systèmes de représentations : les systèmes flottants et les systèmes de codages par résidus. Comme nous l'avons déjà indiqué, les premiers ne répondent pas aux même besoins que les systèmes classiques et redondants (nombres entiers/nombres flottants). Si leur introduction est souhaitable, elle ne semble pas des plus urgentes, d'autant que ces systèmes sont normés ce qui les rend rigides.
Les seconds systèmes sont eux plus intéressants. Ils répondent aux mêmes besoins que les systèmes classiques et redondants, et posent des problèmes similaires à ceux des représentations redondantes (coût meilleures performances/conversions, etc.). L'élargissement de l'arithmétique mixte aux RNS est une des premières choses à effectuer. Un tel travail comprend au moins deux phases : une première correspondant à l'étude de l'arithmétique RNS avec comme point de mire l'élaboration conjointe des opérations élémentaires (addition, somme, multiplication, conversion). Comme les RNS sont composés d'une succession de nombres indépendants, une telle étude implique des opérateurs à la fois localement classiques, mixtes et redondants. La deuxième phase correspond, elle, à l'insertion de l'arithmétique RNS dans l'arithmétique mixte et à l'étude de l'impact du recours aux notations RNS dans divers contextes d'utilisations.

Une autre voie à explorer est celle de l'enrichissement des opérateurs et des structures disponibles. Il serait à la fois intéressant de disposer de nouvelles fonctionnalités mais aussi d'ajouter aux cotés des algorithmes parallèles actuellement utilisés pour les fabriquer, les algorithmes en séries où tout du moins les algorithmes en ligne (avec des calculs s'effectuant sur des mots). De même il serait intéressant de transposer notre travail (générateurs et règles d'optimisation) au monde des $FPGA$.

Un point important que nous n'avons pas développé, est celui de l'impact des notations redondantes sur les mécanismes de tests des circuits. Ce point mérite d'être détaillé car le doublement des registres et l'aspect *opération non encore effectuée* doit avoir un impact non négligeable sur le coût du test que cela soit au niveau matériel (doublement des registres) ou au niveau logiciel (calcul de la valeur des redondants avant vérification).

Aide à la conception

Comme pour l'arithmétique, deux voies sont à explorer. La première concerne le cadre de conception et la seconde les travaux (générateurs, optimisations) que l'on souhaite réaliser dans ce cadre.
Concernant le cadre, la première chose à effectuer est la mise au point d'une première version logicielle stable de l'aide à la conception présentée. Dans la foulée il serait intéressant de mettre en place une connexion vers un outil de synthèse de contrôle afin de compléter les chemins de données élaborés, permettant ainsi d'insérer ce travail dans un flot de conception plus large (synthèse de haut niveau). À plus long terme, une réflexion plus étendue englobant les autres domaines dont le savoir-faire a une forte implication sur les chemins de données, comme le test, devrait être menée (domaines basés, en autres, sur l'encapsulation d'un savoir-faire et sur une méthodologie à base de génération).

Concernant les travaux à effectuer, les évolutions à apporter tournent autour de l'amélioration de l'existant, la transcription des *nouveautés* arithmétiques et l'élargissement des règles d'optimisations. Dans l'immédiat, compléter notre approche multi-représentations par l'introduction et la gestion des notations RNS semblent le plus intéressant. À terme, l'ajout des notations flottantes est incontournable. Dans ce cas, la gestion explicite de mécanismes de contrôle de la précision des calculs (troncature, arrondis, saturation) semble aussi incontournable.
Plus généralement les travaux à effectuer seront l'expression des réflexions faites sur l'arithmétique mais aussi sur les autres domaines influant sur la réalisation d'un chemin de données. Il est alors difficile de les lister ici.

Bibliographie

[Aber95] M. Aberbour, A. Houelle, H. Mehrez, and N. Vaucher. A multiplier by a constant generator using a radix booth algorithm. In *International Conference on Microelectronic (ICM)*, december 1995.

[Aber98] M. Aberbour. Architecture and design methodology of the RBF-DDA neural network. In *Proc. IEEE International Symposium on Circuits and Systems*, 1998.

[Abou00] H. Aboushady, Y. Dumonteix, M.M. Louërat, and H. Mehrez. Efficient polyphase decomposition of comb decimation filter in $\delta\sigma$ analog-to-digital converters. In *Proc. Midwest Symposium on Circuit and System*, august 2000.

[AbouXX] H. Aboushady, Y. Dumonteix, M.M. Louërat, and H. Mehrez. Efficient Polyphase Decomposition of Comb Decimation Filter in $\delta\sigma$ Analog-to-Digital Converters, Accepted for Publication In. *IEEE Trans. on Circuits and Systems–II : Analog and Digital Signal Processing*, XXXX.

[Amar91] A. Amara, A. Greiner, L. Lucas, and F. Petrot. The portable libraries of alliance CAD system. In *IRI*, page VIII Congresso da SBMicro, september 1991.

[Augé97] Ivan Augé, Rajesh K. Bawa, Pierre Guerrier, Alain Greiner, Ludovic Jacomme, and Frédéric Pétrot. User guided high level synthesis. In Ricardo Reis and Luc Claensen, editors, *VLSI : Integrated Systems on Silicon*, pages 464–475, Gramado, Brazil, august 1997. IFIP, Chapman & Hall.

[Augé99] I. Augé, A.Greiner, and F. Pétrot. Fine Grain Scheduling. In *25th Euromicro Conference*, september 1999.

[Aviz62] A. Avizienis. Signed-digit number representation for fast parallel arithmetic. *IRE Trans. Electronic Computers*, 10 :389–400, 1962.

[BOA00] Synopsys. *High-Level Synthesis / Behavioral Compiler Application Notes / Creating High-Speed Data-Path Components*, may 2000.
http ://www.synopsys.com.

[Boot51] A.D. Booth. A signed binary multiplication technique. *Quartely J.Mechanics and Applied Mathematics*, 4(2) :236–240, 1951.

[Bork99] Shekhar Borkar. Design challenges of technology scaling. *IEEE MICRO Chips, Systems, Softwares, and Applications*, pages 23–92, July 1999.

[Boua94] A. Bouaraoua and A. Greiner. A Portable SRAM Generator. In *International Conference on Microelectronic (ICM)*, pages 176–179, september 1994.

[Brig28] H. Briggs. *Arithmetica logarithmica*. Goudae Excudebat Petrus Rammasenius, 1628. http ://gallica.bnf.fr/Catalogue/Notices/IMP/n021044.htm.

[Brig93] W.S. Briggs and D.W. Matula. A 17x69 multiply and add unit with redundant binary feedback and single cycle latency. In *11th Symposium on computer arithmetic*, 1993.

[CAD] Cadence. *Cadence*.
http ://www.cadence.com.

[Camp92] S.G. Campbell. Floating-point operation (1962). *Planning a Computer System : Project Strech*, pages 92–121, W. Buchholz Mc Graw-Hill 1992.

[Cava84] J.J.F. Cavanagh. *Digital Computer Arithmetic, Design and Implementation*. Mc Graw-Hill, 1984.

[Chot00] R. Chotin, Y. Dumonteix, and H. Mehrez. Use of redundant arithmetic on architecture and design of a hight performance dct macro-bloc generator. In *Proc. Design of Circuits and Integrated Systems*, november 2000.

[Cody73] W.J. Cody. Static and Dynamic Numerical Characteristics of Floating-Point Arithmetic. *IEEE Trans. on Computers*, 22(6) :598–601, 1973.

[Comp92] A. Compan, V. Delorme, J.A. François, and H. Mehrez F. Pecheux. F-Risc : A 32 bits floating point integrated in a Risc processor for embedded system. In *International Conference on Microelectronic (ICM)*, december 1992.

[Conw00] R. Conway and J. Nelson. New architecture for RNS FIR filter. In *Proc. International Conference on Signal Processing Applications and Technology*, october 2000.

[Coon84] J.T. Coonen. *Contributions to a Proposed Standard for Binary Floating-Point Arithmetic*. PhD thesis, Berkeley, CA, USA, 1984.

[Dadd65] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34 :549–356, 1965.

[Dadd76] L. Dadda. On parallel digital multipliers. *Alta Frequenza*, 45 :574–580, 1976.

[Daum97] M. Daumas and D.W. Matula. Recoders for partial compression and rouding. In *Research report*, january 1997.
ftp ://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR1997/RR1997-01.ps.Z.

[Daum00a] M. Daumas and D.W. Matula. A booth multiplier accepting both a redundant or a non-redundant input with no additional delay. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2000.

[Daum00b] M. Daumas and D.W. Matula. Further reducing the redundancy of a notation over a minimal redundant digit set. In *Research report 3886, Institut National de Recherche en Informatique et en Automatique*, 2000.

[Dumo00a] Y. Dumonteix, H. Aboushady, H. Mehrez, and M.M. Louërat. Low power comb decimation filter using polyphase decomposition for mono-bit $\delta\sigma$ analog-to-digital converters. In *Proc. International Conference on Signal Processing Applications and Technology*, october 2000.

[Dumo00b] Y. Dumonteix and H. Mehrez. A family of redundant multipliers dedicated to fast computation for signal processing. In *Proc. IEEE International Symposium on Circuits and Systems*, may 2000.

[Dumo01] Y. Dumonteix, Y. Bajot, and H. Mehrez. A fast and low-power distance computation unit dedicated to neural networks, based on redundant arithmetic. In *Proc. IEEE International Symposium on Circuits and Systems*, may 2001.

[Duno99] J. Dunoyer. *Modèles et Méthodes Probabilistes Pour L'évaluation de la Consommation des Circuits Intégrés VLSI*. PhD thesis, Université Pierre et Marie Curie Paris, 1999.
ftp ://asim.lip6.fr/pub/reports/1999/lip6.1999.julien.tgz.

[Dupr91] J. Duprat, Y. Herreros, and S. Kla. New redundant representations of complex numbers and vectors. In *10th Symposium on computer arithmetic*, pages 2–9, 1991.

[Fada93] J. Fadavi-Ardekani. Mxn booth encoded multiplier generator using optimized wallace trees. *IEEE Trans. on VLSI Systems*, 1(2) :120, june 1993.

[Garn59] H.L. Garner. The residue number system. *IRE Trans Electronic Computers*, 8 :140–147, 1959.

[Gold91] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1) :5–47, march 1991.

[Gran99] B. Granado. An embedded system for the real time execution of grbf networks. In *Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, april 1999.

[Grei92] A. Greiner and F. Pecheux. A complet set of CAD Tools for teaching VLSI Design. In *Third EuroChip Workshop on VLSI Design Trainning*, september 1992.
http ://www-asim.lip6.fr/alliance/.

[Grei94] A. Greiner and F. Petrot. A Public Domain High Portable ROM Generator. In *Euromicro Conference*, pages 414–419, september 1994. IEEE Computer Society Press.

[GuyoX1] A. Guyot. Cours d'opérateurs arithmétiques. In , XXX1.
http ://tima-cmp.imag.fr/∼guyot/Cours/Arithmetique/pdffile/multi.pdf.

[GuyoX2] A. Guyot. Réduction d'arbres de wallace. In , XXX2.
http ://tima-cmp.imag.fr/∼guyot/Cours/Exercices/ReducTree.html.

[Houe97] A. Houelle. *GENOPTIM : Un environement d'aide à la conception de générateurs de circuits portables optimisés en performance et en surface*. PhD thesis, Université Pierre et Marie Curie Paris, 1997.
ftp ://ftp.lip6.fr/lip6/reports/1997/lip6.1997.026.ps.tar.gz.

[IEEE85] IEEE. Standard for binary floating-point arithmetic (ansi/ieee std 754-1985). *IEEE Press*, 1985.

[Kaha96] W. Kahan. *Lecture notes on the status of the IEEE standard 754 for binary floating point arithmetic*. Published on the Net, 1996. http ://http.cs.berkeley.edu/∼wkahan/.

[Kant00] V. Kantabutra, P. Corsonello, and S. Perri. Fast, low-cost adders using carry strength signals. In *SSGRR*, july 2000.

[Leis83] C. Leiserson, F. Rose, and J. Saxe. Optimizing synchronous circuitry by retiming. In *3 rd Caltech conf VLSI*, march 1983.

[Metz59] G. Metz and J.E. Robertson. Elimination of carry propagation in digital computers. In *Information Processind'59 (UNESCO Conference)*, 1959.

[Moor65] G.E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[Mull89] J-M. Muller. *Arithmétique des ordinateurs, opérations et fonctions élémentaires*. Masson, 1989.

[Mull97] J-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser Boston, 1997.

[Pali00] V. Paliouras. Novel high-radix residue number system architectures. *IEEE Trans. on Circuits and Systems–II : Analog and Digital Signal Processing*, 47(10) :1059–1073, october 2000.

[Parh99] B. Parhami. *Computer Arithmetic, Algorithms and Hardware Designs*. Oxford University Press, 1999.

[Peyr97] O. Peyran. *Synthèse d'Architectures Intégrées Utilisant des Arithmétiques Redondantes*. PhD thesis, INP Grenoble France, 1997.
ftp ://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD1997/PhD1997-14.ps.Z.

[Rami00] J. Ramirez, A. Garcia, P.G. Fernandez, L. Parrilla, and A. Lloris. A novel QRNS-based 1-D DCT processor over field-programmable logic devices. In *Proc. Design of Circuits and Integrated Systems*, november 2000.

[Skla60] J. Sklansky. Conditional-sum addition logic. *IRE Trans. Electronic Computers*, 9 :226–231, 1960.

[Sorl91] Mac Sorley. High speed arithmetic in binary computers. In *IRI*, 1991.

[Soud00] D.J. Soudris, M.M Dasigenis, and A.T. Thanailakis. Designing RNS and QNRS full-adder based converters. In *Proc. IEEE International Symposium on Circuits and Systems*, may 2000.

[Souf00] H. Souffi-Kebbati, J.P. Blonde, and F. Braun. A cmos standard cell based, parallel 32x32 bits signed multiplier. In *Proc. Design of Circuits and Integrated Systems*, november 2000.

[SYN00] Synopsys. *High-level Synthesis Tools*, may 2000.
http ://www.synopsys.com.

[Szab67] N.S. Szabo and R.I. Takana. *Residue Arithmetic and Its Applications to Computer Technology*. Mc Graw-Hill, 1967.

[Turi00] A. Turier. *Étude, conception et caractérisation de mémoires CMOS , faible consommation, faible tension en technologies submicroniques*. PhD thesis, Université Pierre et Marie Curie Paris, 2000.
ftp ://asim.lip6.fr/pub/reports/2000/lip6.2000.turier.pdf.

[Vauc97] N. Vaucher. *Méthodologie de conception d'architectures VLSI gén'eriques appliquée au traitement numérique*. PhD thesis, Université Pierre et Marie Curie Paris, 1997.
ftp ://ftp.lip6.fr/lip6/reports/1997/lip6.1997.025.ps.tar.gz.

[Vold59] J.E. Volder. The CORDIC trigonometric computer technique. *IRE Trans. Electronic Computers*, 8 :330–334, 1959.

[Wall64] C.S. Wallace. A suggestion for a fast multiplier. *Proc. Techno 90*, 13(11) :14–17, 1964.

[Yohe73] J.M. Yohe. Rounding in floting-point arithmetic. *IEEE Trans. on Computers*, 22(6) :577–586, 1973.

[Zimm99] R. Zimmermann. Computer arithmetic : Principles, architectures and vlsi design. In *Lecture notes on*, 1999.
http ://www.iis.ee.ethz.ch/∼zimmi/publications/comp_arith_notes.ps.gz.

Liste des publications

[Dum00a] Y. Dumonteix and H. Mehrez, A family of redundant multipliers dedicated to fast computation for signal processing, In *Proc. IEEE International Symposium on Circuits and Systems*, may 2000, ftp ://asim.lip6.fr/pub/reports/2000/ar.dum.iscas00.ps.gz.

[Dum00b] H. Aboushady, Y. Dumonteix, M.M. Louërat, and H. Mehrez, Efficient polyphase decomposition of comb decimation filter in $\Delta\Sigma$ analog-to-digital converters, In *Proc. Midwest Symposium on Circuit and System*, august 2000, ftp ://asim.lip6.fr/pub/reports/2000/ar.abou.mwscas00.ps.gz.

[Dum00c] Y. Dumonteix, H. Aboushady, H. Mehrez, and M.M. Louërat, Low power comb decimation filter using polyphase decomposition for mono-bit $\Delta\Sigma$ analog-to-digital converters, In *Proc. International Conference on Signal Processing Applications and Technology*, october 2000, ftp ://asim.lip6.fr/pub/reports/2000/ar.dumo.icspat00.pdf.

[Dum00d] R. Chotin, Y. Dumonteix, and H. Mehrez, Use of redundant arithmetic on architecture and design of a hight performance dct macro-bloc generator, In *Proc. Design of Circuits and Integrated Systems*, november 2000, ftp ://asim.lip6.fr/pub/reports/2000/ar.chot.dct_ip_core.ps.gz.

[Dum01a] Y. Dumonteix, Y. Bajot, and H. Mehrez, A fast and low-power distance computation unit dedicated to neural networks, based on redundant arithmetic, In *Proc. IEEE International Symposium on Circuits and Systems*, may 2001, ftp ://asim.lip6.fr/pub/reports/2001/ar.dumo.iscas2001.pdf.

[Dum01b] H. Aboushady, Y. Dumonteix, M.M. Louërat, and H. Mehrez. Efficient Polyphase Decomposition of Comb Decimation Filter in $\Delta\Sigma$ Analog-to-Digital Converters, Accepted for Publication In *IEEE Trans. on Circuits and Systems–II : Analog and Digital Signal Processing*, XXXX.

Annexes

Annexe A Multiplication par une constante

Cette annexe [1] est consacrée à la réalisation de la multiplication par une constante. Les architectures développées sont basées sur l'algorithme de *Booth*. L'objectif est de profiter de l'aspect invariant de l'entrée constante pour utiliser des bases supérieures à quatre lors de son encodage.

A.1 Introduction

La multiplication par un coefficient constant est une opération très fréquemment rencontrée en traitement du signal. Elle est présente dans beaucoup d'algorithmes comme les filtres numériques (FIR, IIR, etc.) ou certaines transformées (FFT, DCT, etc.).
Disposer d'un tel opérateur permet à la fois de réduire la surface, le délai et la consommation par rapport à un multiplieur variable par variable. Les gains liés à ces trois critères pouvant avoir une incidence importante sur les choix d'implémentation d'un algorithme.
Comme nous l'avons vu dans la section 3.4, un multiplieur est décomposable en deux parties : le calcul des produits partiels et leur sommation. Dans le cas d'un multiplieur par une constante, la nature invariante d'une des deux entrées permet de supprimer la quasi totalité du matériel nécessaire à la réalisation de la matrice [Mull97, Parh99]. Ainsi, la multiplication par une constante se résume à une somme. Améliorer ces performances repose alors, presque exclusivement sur la réduction du nombre de produits partiels à additionner.
L'algorithme de *Booth* en base 4, utilisé dans la conception de multiplieurs *variable* · *variable* classiques, sous-section 3.4.1, se prête bien à cet objectif. A lui seul, il assure la division par deux du nombre de produits partiels. Dans l'article [Aber95], les auteurs vont plus loin dans la réduction du nombre de termes à sommer, en utilisant l'algorithme de *Booth* en base 8. Dans cet esprit, nous proposons de généraliser l'utilisation des bases supérieures à quatre, lors de l'application de cet algorithme. Deux solutions sont envisagées pour recoder une même constante : en premier, user d'une base unique supérieure à 4, et en second utiliser simultanément plusieurs bases.

[1] Ces travaux ont été effectués en collaboration avec Hazem Moussa et Hervé-Robert Charlery lors de leur stage de maîtrise.

A.2 Algorithme de Booth

Mac Sorley [Sorl91] proposa en 1961 une méthode de recodage de l'algorithme de *Booth* prenant en compte 3 bits ou plus. Cette technique consiste à réécrire un nombre binaire en complément à deux sur M bits tel que ce nombre X en base 2^M soit :

$$X = -2^{M-1} \cdot X_{M-1} + \sum_{i=0}^{M-2} (X_i \cdot 2^i) \tag{A.1}$$

En s'appuyant sur cette méthode, nous allons chercher à généraliser l'usage des bases supérieures à 4.

A.2.1 Algorithme de Booth en base 2^M, M constant

La technique employée se décompose en deux étapes. La première consiste à appliquer l'algorithme de *Booth* pour obtenir les chiffres composant l'expression de la constante dans la base 2^M choisie. La deuxième a pour tâche de rendre interprétable ces chiffres, en vue du calcul des produits partiels. L'algorithme de *Booth* étant appliqué à la constante, déterminer les chiffres ne pose aucune difficulté. En fait, toute la problématique réside dans la seconde étape.

Encodage de la constante

En base 4, les chiffres issus de l'encodage de la constante, sont éléments de {-2,-1,0,1,2}. Leur interprétation est basée sur la correspondance en binaire, du décalage et de la multiplication par deux. Cette propriété permet de faire coïncider à chaque produit *chiffre encodé* $\cdot$ *variable*, un seul bit. La réduction du nombre de produits à sommer est de deux. Nous garderons cette valeur comme référence de comparaison. Cette correspondance entre un chiffre et un décalage reste vraie tant que le chiffre encodé est une puissance de deux. Pour un chiffre égal à 2^i, le décalage est de i.
En base 2^M avec $M > 2$, tous les chiffres issus de l'encodage, ne sont pas des puissances de deux (au coefficient -1 près), comme le montre les ensembles définissant les bases 8 : {-4,-3,-2,-1,0,1,2,3,4}, et 16 : {-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8}. Leur nombre augmente même avec les bases : 1 en base 8 ($\pm$3), 4 en base 16 ($\pm$7,$\pm$6,$\pm$5,$\pm$3), etc.
Aussi, chercher à garder la correspondance 1 chiffre/1 bit pour obtenir une réduction par M du nombre de produits partiels, implique de pré-calculer les produits lignes : *variable par chiffre* $\neq \pm 2^i$ avant de les sommer. Cette solution n'est pas réaliste. Le coût matériel, serait trop onéreux et conduirait à une contre-performance. C'est d'autant plus vrai que le nombre de ces chiffres augmente avec la base. La solution envisagée et réalisée, consiste à décomposer ces chiffres en

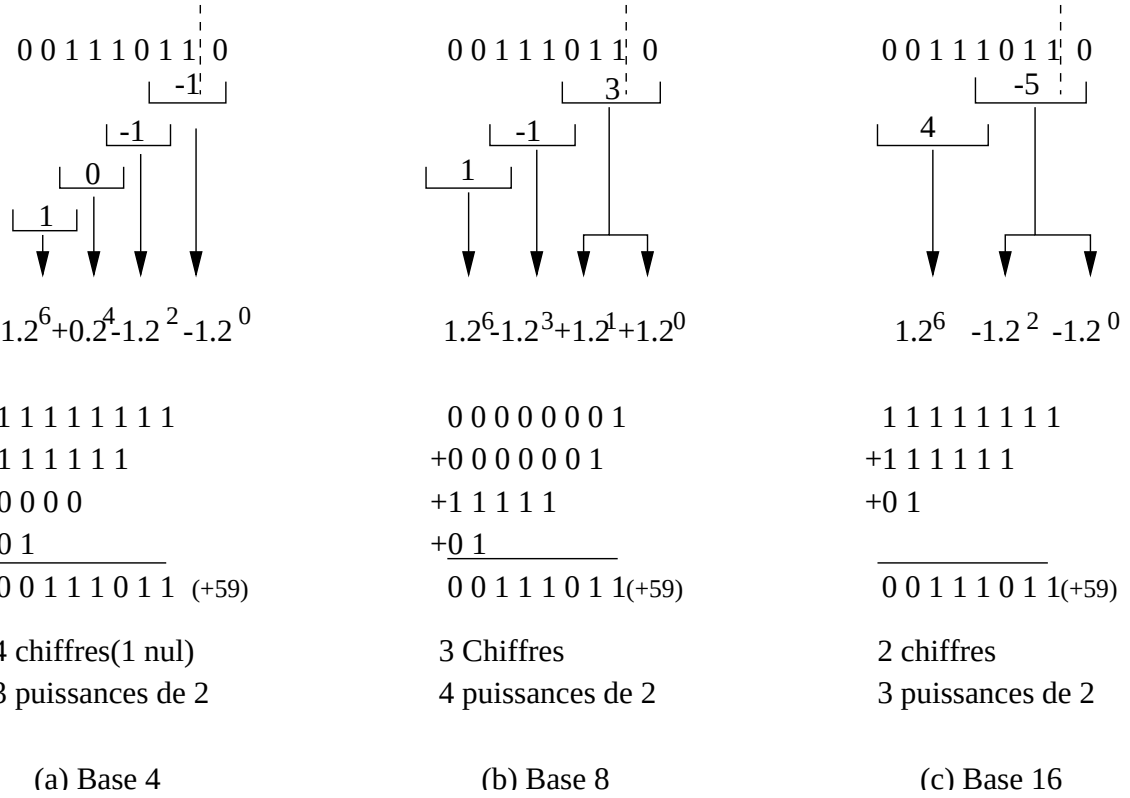


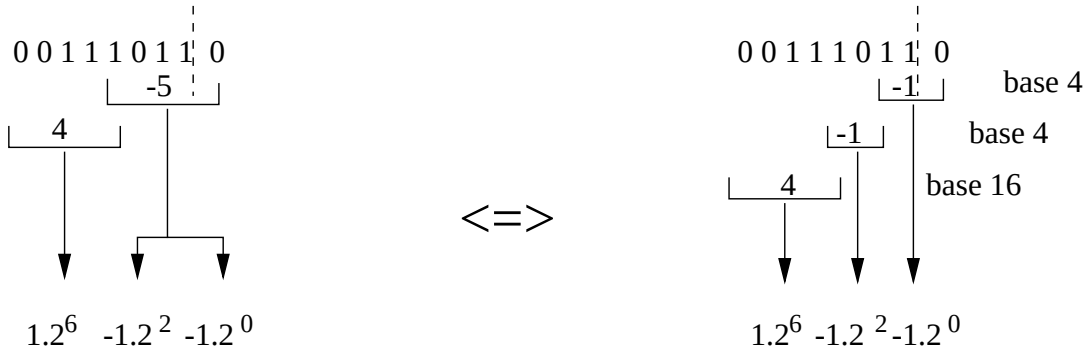
Figure A.1 – Décomposition en puissance de deux de $59_{(00111011)}$

puissances de deux au coefficient -1 près, avant d'effectuer et de sommer les produits lignes. Pour illustrer cette technique, la figure A.1 présente la décomposition de la constante 59 suivant les bases 2^M égales à 4, 8 et 16. M est respectivement égal à 2, 3 et 4.
La correspondance *1 chiffre / 1 bit* n'est pas respectée. Le nouveau rapport dépend de la valeur du chiffre considéré et de la base choisie. Grâce à l'utilisation conjointe de 2^i et des -2^i avec $i \leq M$, le nombre de puissances de deux obtenu est borné par $(M+1)/2$ (division entière). Dans ces conditions, garantir une réduction minimale par deux, référence de comparaison, nécessite juste que M soit pair. Les exemples proposés figure A.1 illustrent ces propos.

A.2.2 Nombre variable de bases

Vers l'utilisation simultanée de plusieurs bases

En analysant la technique de décomposition en puissances de deux, il apparaît que la solution choisie consiste à utiliser *localement* des bases différentes. Reprenons l'exemple en base 16 (2^4), de la figure A.1, et détaillons-le : figure A.2. Comme le montre le cas A.2.(b), la décomposition



(a) base 16 (b) bases multiples

Figure A.2 – Vers l'usage simultané de plusieurs bases

du chiffre -5, revient à utiliser localement deux fois la base 4. Parallèlement, les bits de poids forts sont encodés avec la base 16.

En généralisant, la décomposition en puissance de deux des chiffres non égaux à ces puissances, revient à employer parallèlement plusieurs bases 2^i tel que $(i < M)$. Plus M est grand, plus le nombre de *sous bases* utilisable est important et plus la réduction potentielle est importante. Les meilleurs résultats sont obtenus pour M tendant vers l'infini, soit M *égal* à la dynamique de la constante.

Algorithme de Booth en bases multiples

Nous donnons ci-dessous une solution pour déterminer les puissances de 2 décomposant une constante codée sur M bits. L'algorithme proposé est simple : figure A.3. Il démarre en prenant en compte l'ensemble des bits composant la constante. La première étape consiste à chercher le premier sous-ensemble défini par un chiffre égal à une puissance de deux (au signe près). Cette première étape effectuée, l'opération est répétée mais cette fois en limitant l'ensemble de départ considéré, aux bits de la constante non encore exprimés. Le processus est réitéré jusqu'à ce que la constante entière soit encodée.

Pour que le résultat soit juste, il faut tenir compte du codage différentiel implicite à l'algorithme de *Booth*, en réintégrant le signe (repport) du chiffre venant juste d'être trouvé, dans le calcul du suivant.

Cet algorithme est un peu brutal, mais il donne le résultat escompté. La figure A.4 présente un exemple de décomposition en base variable.

repport=0, rang=0, K=constante, M=taille de K
repeter
    i=-1
    répeter
        i=i+1
        d=premier_chiffre(K,base 2^{M-i},repport)
    tant que ($|d| \neq$ puissance de 2)
    repport=K ET 2^{M-i-1}
    K=K div 2^{M-i}
    rang=rang+M-i
    M=i
tant que (K=0)

Avec :
premier_chiffre=-(K ET 2^{M-i})+(|K| mod 2^{M-i})+repport
chiffre=$d \cdot 2^{rang}$

Figure A.3 – Algorithme de Booth en base variable

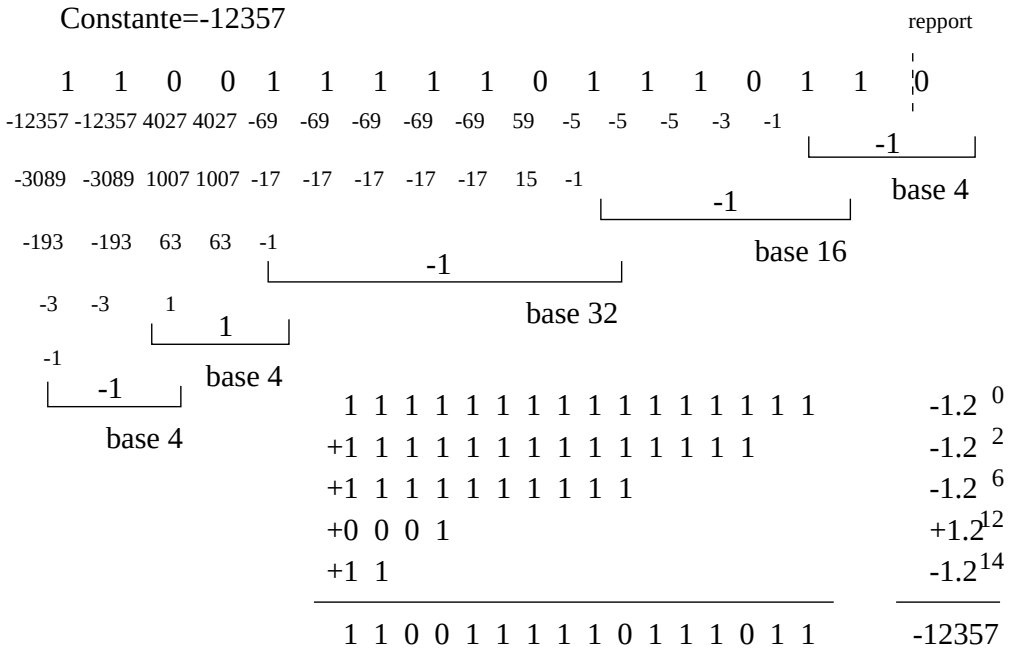


Figure A.4 – Exemple de décomposition en base multiple

A.3 Multiplieurs mixtes par une constante

Malheureusement, l'usage de l'algorithme de *Booth* n'est pas complémentaire à la technique (basée sur les décalages) utilisée lors de la réalisation des multiplieurs mixtes et redondants. Ce constat est vrai que l'encodage soit en base 4 ou en bases multiples.
Leur usage conjoint est certes possible, mais les décalages induisent des produits partiels contigus, alors que l'algorithme de *Booth*, n'implique un produit, au minimum, que tous les deux bits. Dans ces conditions, le nombre de produits partiels est de deux par produit $chiffre \cdot chiffre$. Cela implique le doublement de la quantité de produits partiels, et équivaut à deux multiplieurs $NR \cdot constante$. L'addition préalable des deux termes composant le nombre redondant est préférable.
Dans le cas de la multiplication $BS \cdot NR$, le décalage est remplacé par le choix entre -1 et 1 sous-section 3.4.3 figure 3.25. *A priori*, les deux techniques sont alors exploitables sans augmentation du nombre de produits partiels à sommer. Le problème s'est déplacé. Maintenant, chaque produit partiel peut prendre les trois valeurs de l'ensemble {-1,0,+1}, contre les deux des ensembles {-1,0} ou {0,+1} précédents. De nouveau, deux bits sont nécessaires au codage d'un produit partiel $chiffre \cdot chiffre$.

A.4 Architecture

A.4.1 Architecture globale

Comme nous l'avons indiqué dans la section 3.4, un multiplieur se décompose en deux grosses parties : la matrice des produits partiels et la somme qui les additionne. L'architecture globale figure A.5, reprend ce modèle en l'appliquant au cas de la multiplication par une constante.

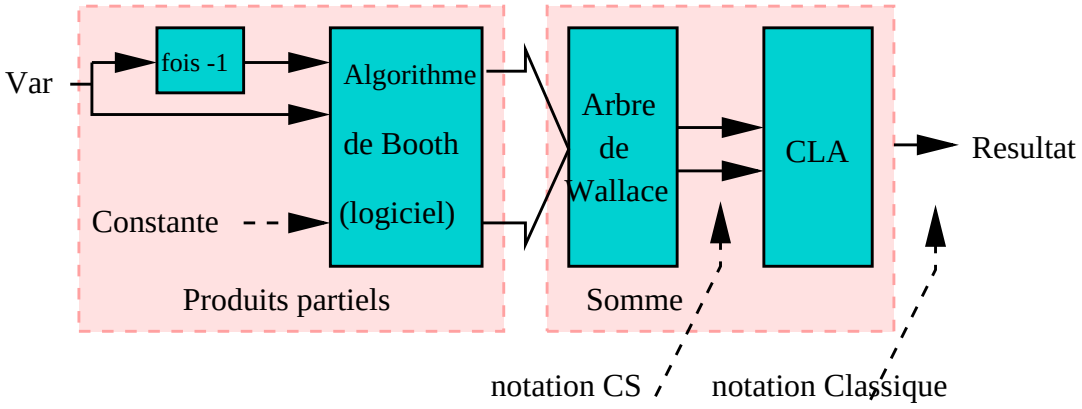


Figure A.5 – Multiplication par une constante : architecture globale

A.4.2 Matrice des produits partiels

La nature constante d'une des deux entrées et le choix de ne garder que les chiffres égaux aux puissances de deux impliquent que le matériel nécessaire à la matrice des produits partiels est inexistante : 2^i en binaire équivaut à un décalage de i bits, lequel est constant et ne correspond alors qu'à un bus. Le calcul des chiffres est lui, fait par logiciel.
Toutefois, l'algorithme de *Booth* impose de disposer de la variable et de sa valeur multipliée par -1. Nous utiliserons à cet effet le complément à deux ($-X = \overline{X} + 1$). Le complément à 1 ($\overline{X}$) est obtenu par une ligne d'inverseurs. Il est distribué en fonction des puissances de deux négatives de la constante encodée. Les *plus 1* ($+1$) sont eux, additionnés en tenant compte des décalages, et sont injectés sous forme d'une constante dans la somme.

A.4.3 Somme

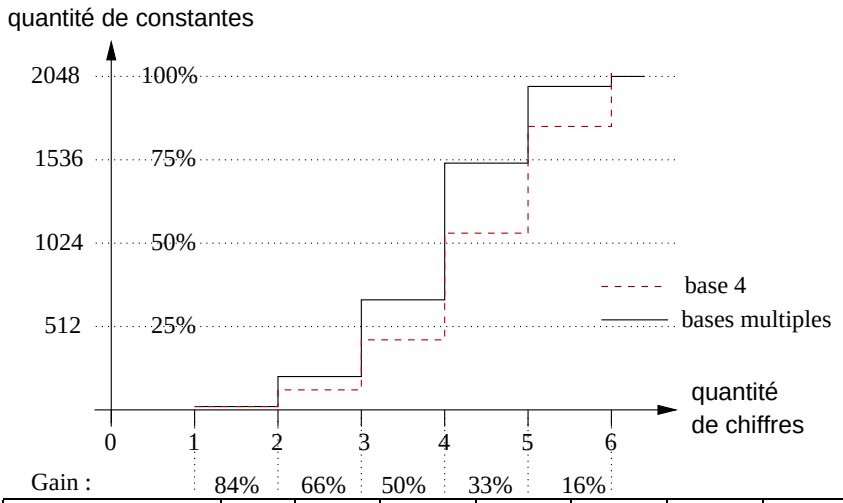
Comme pour les multiplieurs mixtes, section 3.4, la somme des produits partiels est réalisée avec un réducteur de Wallace. Le résultat est en notation Carry-Save. Pour obtenir le produit final en notation classique, il faut additionner les deux composantes du CS. Ici aussi nous utilisons un additionneur rapide tel que le *Carry Lookahead Adder - CLA*.
Dans le cas particulier de la multiplication par une constante, les produits partiels ne forment pas de produit ligne où tous les bits sont contigus, et nombre d'entre eux sont égaux à -1. Ces particularités nous ont amené à considérer chaque bit distinctement et à traiter les bits égaux à -1 comme des bits de signes lors de leur introduction dans la somme. Leur traitement suivra la méthode établie dans la section 3.3.

A.5 Résultats

Cette section est découpée en deux parties. La première est dédiée à l'évaluation de l'algorithme proposé. La seconde présente la réalisation matérielle de plusieurs exemples de multiplieurs par une constante. Dans les deux cas, les performances des multiplieurs constants en bases multiples, sont comparées à celles des multiplieurs $variable \cdot variable$ et $constante \cdot variable$ utilisant l'algorithme de *Booth* en base 4.

A.5.1 Algorithme

Pour estimer les performances de l'encodage en bases multiples, nous avons étudié les écarts de quantité de chiffres obtenus, pour l'encodage de toutes les constantes définies par une taille en bits donnée.



Quantité de chiffres	0	1	2	3	4	5	6
Base 4	1	16	105	360	675	648	243
Base multiple	1	21	162	560	840	432	32

Figure A.6 – Comparaison avec un multiplieur $variable \cdot variable$

Comparaison avec un multiplieur $variable \cdot variable$

La figure A.6 présente la répartition de toutes les constantes représentables sur une dynamique de 12 bits, en fonction de la quantité de puissances de deux les composant. L'étude porte à la fois sur les algorithmes en base 4 et en base multiple. Dans ce contexte, six chiffres maximums sont nécessaires pour encoder une $variable$ de 12 bits avec l'algorithme de *Booth* en base 4. Ce nombre correspond à un multiplieur $variable \cdot variable$. Il sert de point de référence.
Dans 88% des cas pour l'encodage en base 4, et dans 98.5% des cas pour un encodage en bases multiples, la quantité de chiffres est inférieur à celui du multiplieur variable équivalent. Les gains sont compris entre 16% et 86%. Les gains en terme de quantité de chiffres, sont très significatifs (>33%) pour 56% des constantes avec un encodage en base 4, et pour 77% des constantes avec un encodage en bases multiples.
Ces résultats montrent tout l'intérêt d'utiliser des multiplieurs par une constante. Cette remarque est d'autant plus vraie, que les gains présentés ne tiennent pas compte de l'absence de matériel nécessaire à la réalisation de la matrice de produits partiels et de la réduction de la somme qui découle de la réduction du nombre de produits partiels.

Comparaison entre les versions en base 4 et en bases multiples de l'algorithme de *Booth*

La figure A.7 présente, pour l'ensemble des constantes codables sur 12 bits, les écarts de quantité chiffres entre les réalisations en base 4 et en bases multiples.
Deux points importants sont à noter. En premier, quelle que soit la constante, le gain est tou-

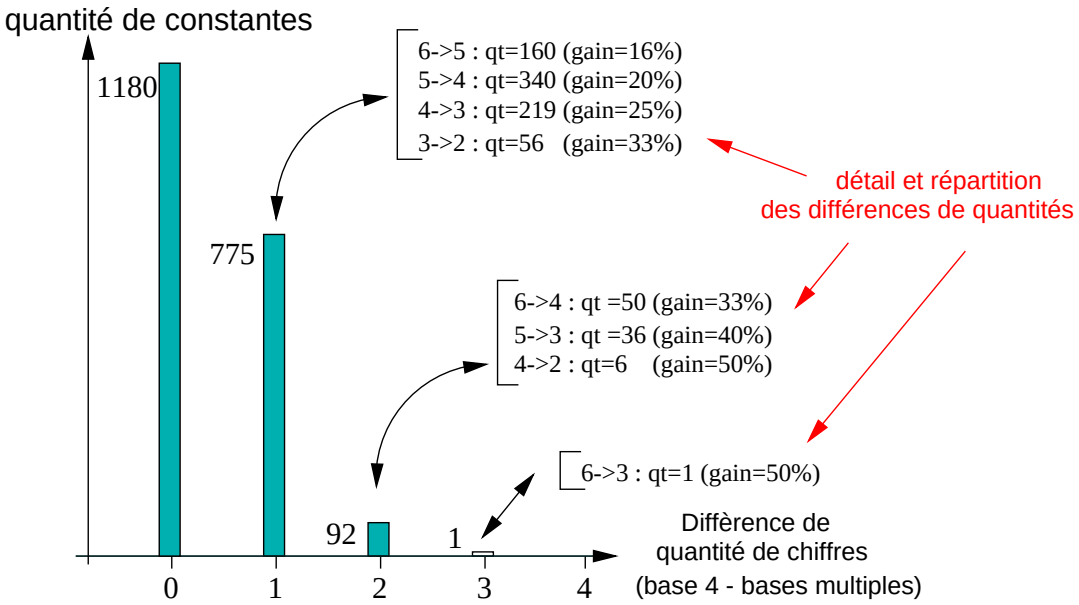


Figure A.7 – Différence de quantité de chiffres entre les encodages en base 4 et en bases multiples

jours positif. Au pire il sera nul. Deuxièmement, dans presque la moitié des cas, l'encodage en bases multiples introduit une réduction supplémentaire de la quantité de chiffres comprise de un à trois chiffres. Les gains engendrés étant compris eux, entre 16% et 50%. Ces deux points illustrent parfaitement l'intérêt de l'usage simultané de plusieurs bases, lors de l'application de l'algorithme de *Booth* à une constante.

A.5.2 Réalisations matérielles

Plutôt que de présenter des exemples dit *représentatifs*, nous avons orienté notre étude comparative vers des coefficients correspondant à une application réelle. Notre choix s'est porté sur la réalisation *directe* de la transformée cosinus discrète (DCT).

La formule mathématique de la DCT mono dimensionnelle est donnée par :

$$DCT(i) = \frac{2}{\sqrt{2N}} c(i) \sum_{x=0}^{N-1} pixel(x) \cos \frac{(2x+1)i\pi}{2N} \tag{A.2}$$

où $c(x) = \frac{1}{\sqrt{2}}$ si $x = 0$ et $c(x) = 1$ si $x \neq 0$, et $pixel(x)$ la valeur du pixel à la coordonnée x.

Cet algorithme est exploitable sous forme matricielle. En simplifiant et factorisant cette matrice et avec $N = 8$ et $\theta = \frac{\pi}{16}$, on arrive à l'expression matricielle suivante :

$$\begin{pmatrix} Y0 \\ Y2 \\ Y4 \\ Y6 \end{pmatrix} = \begin{pmatrix} \cos 4\theta & \cos 4\theta & \cos 4\theta & \cos 4\theta \\ \cos 2\theta & \cos 6\theta & -\cos 6\theta & -\cos 2\theta \\ \cos 4\theta & -\cos 4\theta & -\cos 4\theta & \cos 4\theta \\ \cos 6\theta & -\cos 2\theta & \cos 2\theta & -\cos 6\theta \end{pmatrix} * \begin{pmatrix} X0 + X7 \\ X1 + X6 \\ X2 + X5 \\ X3 + X4 \end{pmatrix}$$

$$\begin{pmatrix} Y1 \\ Y3 \\ Y5 \\ Y7 \end{pmatrix} = \begin{pmatrix} \cos \theta & \cos 3\theta & \cos 5\theta & \cos 7\theta \\ \cos 3\theta & -\cos 7\theta & -\cos \theta & -\cos 5\theta \\ \cos 5\theta & -\cos \theta & \cos 7\theta & \cos 3\theta \\ \cos 7\theta & -\cos 5\theta & \cos 3\theta & -\cos \theta \end{pmatrix} * \begin{pmatrix} X0 - X7 \\ X1 - X6 \\ X2 - X5 \\ X3 - X4 \end{pmatrix}$$

Cette matrice comprend 7 coefficients constants différents (au signe près) et 32 multiplieurs. La dynamique des constantes est sur 12 bits, celle des variables sur 9 bits dont 1 bit de signe. Ces 9 bits, correspondent à une application réelle [Chot00].
Dans l'exemple choisi, chaque multiplication est additionnée à 3 autres. Pour améliorer les performances de cette seconde étape de calcul, la sortie de chacun des multiplieurs sera en notation *Carry-Save*. L'additionneur (CLA) : figure A.5, présent en fin de chaque multiplieur disparaît. Nous respecterons cette contrainte pour estimer les performances.

Performances

L'étude des performances porte uniquement sur la réalisation des différents multiplieurs par une constante de la DCT. Tous ont été réalisés à la fois avec un encodage en bases 4 et un encodage en bases multiples. Parallèlement, un multiplieur $variable \cdot variable$ utilisant l'algorithme de *Booth* en base 4, a aussi été réalisé ; ses performances vont servir de référence au comparatif. La sortie de chacun des opérateurs réalisés est en notation *Carry-Save*. L'ensemble des résultats est présenté dans la table A.1.
Pour chaque valeur réelle exprimant un délai, une consommation ou une surface, nous indiquons le gain en % par rapport à la référence. Les pourcentages entre parenthèses, donnés pour l'encodage en bases multiples, correspondent à la différence de gain entre cette méthode et l'encodage en base 4. Comme pour les opérateurs élémentaires en arithmétique mixte (chapitre 4) tous les circuits présentés ont été placés, routés et évalués, à l'aide des chaînes de conceptions *Cadence* [CAD] et *Alliance* [Grei92].

Comparaison avec la référence

Quelque soit le critère ou l'algorithme utilisé, les gains sont très importants : de 71% à 89% en surface, de 31% à 68% en délai et de 75% à 91% en consommation. Les meilleures performances

			base 4				**bases multiples**			
Constante	codage décimal	codage binaire	surface μM^2		delai ns		Consommation $\mu W/Mhz$		surface μM^2		delai ns		Consommation $\mu W/Mhz$	
Multiplieur $Variable \cdot Variable$			158642	réf.	4.77	réf.	139	réf.						
$\pm\cos(4\pi/16)$	0.707107	001011010100	36939	77%	3.00	37%	29	79%	36939	77%	3.00	37%	29	79%
$\pm\cos(2\pi/16)$	0.92388	001110110010	35546	78%	2.83	41%	30	78%	25836	84% (27%)	2.21	54% (22%)	21	85% (30%)
$\pm\cos(6\pi/16)$	0.382683	000110001000	24071	85%	2.63	45%	20	86%	18417	88% (24%)	1.53	68% (42%)	13	91% (35%)
$\pm\cos(\pi/16)$	0.980785	001111101100	16838	89%	1.61	66%	14	90%	16838	89%	1.61	66%	14	90%
$\pm\cos(3\pi/16)$	0.83147	001101010011	46387	71%	3.29	31%	35	75%	46387	71%	3.29	31%	35	75%
$\pm\cos(5\pi/16)$	0.55557	001000111001	37264	77%	2.73	43%	29	79%	24530	85% (34%)	1.93	60% (29%)	17	88% (41%)
$\pm\cos(7\pi/16)$	0.19509	000011001000	25323	84%	2.37	43%	20	86%	16811	89% (34%)	1.55	68% (35%)	12	91% (40%)

TAB. A.1 – Exemples d'applications des encodages en base 4 et en bases multiples

sont obtenues par la technique utilisant simultanément plusieurs bases. Ces résultats illustrent parfaitement l'intérêt de la dégradation des multiplieurs quand une des deux entrées est une constante. Ils recoupent parfaitement ceux obtenus lors du comparatif au niveau algorithme.

Comparaison entre l'algorithme de Booth en base 4 et en bases multiples

Comme l'illustre les performances des multiplieurs par $\pm cos\frac{4.\pi}{16}$, $\pm cos\frac{\pi}{16}$ et $\pm cos\frac{3.\pi}{16}$, l'utilisation de plusieurs bases n'introduit pas un gain systématique. Les résultats de ces trois opérateurs sont toutefois identiques à ceux des multiplieurs ayant un encodage uniquement en base 4, car la technique adoptée garantit d'utiliser au pire la base 4. Pour les autres multiplieurs, l'amélioration des performances est significative. Quel que soit le critère, elle est comprise entre 25% à 45%. Le gain entre les deux techniques apparaît comme variable et très dépendant de la constante considérée.
Plus généralement, l'assurance d'un gain au pire nul, implique que l'encodage en bases multiples peut-être systématiquement utilisé dans le cas de multiplication par une constante.

A.6 Conclusion

Dans cette annexe, nous avons présenté une technique permettant de généraliser l'utilisation des bases supérieures à 4, lors de l'emploi de l'algorithme de *Booth* dans la conception de multiplieurs par une constante.
La garantie d'utiliser au pire la base 4 assure que l'extension aux bases supérieures, implique un gain toujours positif (au pire nul) par rapport à la technique d'encodage reposant uniquement sur la base 4.
Malheureusement, cette technique n'est pas extensible aux multiplieurs mixtes et redondants car les techniques de décalages et l'algorithme de *Booth* ne sont pas complémentaires ; leurs gains respectifs ne peuvent pas être additionnés.

Annexe B Carré et calcul de distance

Cette annexe est consacrée à la réalisation de l'opérateur de carré (A^2) en arithmétique mixte. L'objectif est double : premièrement élargir le champ des opérateurs mixtes au *carré* et deuxièmement illustrer l'intérêt de détourner les opérateurs mixtes et redondants afin d'obtenir des fonctionnalités nouvelles en arithmétique classique (ici le calcul de distance $(A-B)^2$ et la somme de distance $\sum_i (A_i - B_i)^2$).

B.1 Introduction

L'opération *carrée* est un cas particulier de la multiplication. Sa fréquence d'apparition dans les algorithmes de traitements du signal est plutôt faible. Généralement son calcul est effectué à l'aide d'un multiplieur. C'est le cas dans les circuits où les ressources sont banalisées : les processeurs par exemple. Toutefois certaines applications sont très gourmandes en *carré*. C'est le cas des réseaux de neurones RBF (Radial Basis Function) [Aber98, Gran99], et plus généralement des applications effectuant des calculs de distances. Dans ce contexte il devient alors intéressant de s'attarder sur la spécificité de cette opération et de développer l'opérateur correspondant.

B.2 Architectures associées au *Carré*

B.2.1 Opérateur *Carré* : Principe

Comme indiqué précédemment le *carré* est un cas particulier de la multiplication où les deux opérandes sont égales. L'exploitation de cette particularité permet de réduire sensiblement le matériel nécessaire au développement de l'opérateur *carré* [Mull97, Parh99].
Considérons un multiplieur classique et l'opération A^2. L'égalité des deux entrées revient à dire que les produits élémentaires $a_i \cdot a_j$ et $a_j \cdot a_i$ sont égaux ($i \neq j$). La matrice des produits partiels est donc symétrique selon sa diagonale, et le nombre de produits partiels peut-être réduit pratiquement de 50% en remplaçant chaque pair $a_i \cdot a_j$ et $a_j \cdot a_i$ par le terme unique

$2.a_j \cdot a_i$. Seuls les termes $a_i \cdot a_j$ avec $i = j$ restent inchangés. La figure B.1 illustre l'exploitation de cette particularité pour A sur 5 chiffres.

					a_4	a_3	a_2	a_1	a_0
				x	a_4	a_3	a_2	a_1	a_0
					$a_4.a_0$	$a_3.a_0$	$a_2.a_0$	$a_1.a_0$	$a_0.a_0$
				$a_4.a_1$	$a_3.a_1$	$a_2.a_1$	$a_1.a_1$	$a_0.a_1$	
			$a_4.a_2$	$a_3.a_2$	$a_2.a_2$	$a_1.a_2$	$a_0.a_2$		
		$a_4.a_3$	$a_3.a_3$	$a_2.a_3$	$a_1.a_3$	$a_0.a_3$			
	$a_4.a_4$	$a_3.a_4$	$a_2.a_4$	$a_1.a_4$	$a_0.a_4$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

(a) Multiplication $A \cdot A$

					a_4	a_3	a_2	a_1	a_0
				x	a_4	a_3	a_2	a_1	a_0
				$a_4.a_0$	$a_3.a_0$	$a_2.a_0$	$a_1.a_0$	-	a_0
			$a_4.a_1$	$a_3.a_1$	$a_2.a_1$	-	a_1		
		$a_4.a_2$	$a_3.a_2$	-	a_2				
	$a_4.a_3$	-	a_3						
	a_4								
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

(b) Réduction de la matrice

Figure B.1 – Réduction de la matrice de produits partiels

La réduction de la matrice est effective que les a_i représentent des bits ou des chiffres (bs_i ou cs_i). Il devient alors intéressant de proposer une réalisation matérielle de A^2 prenant en compte les notations redondantes, et d'étudier son impact dans une unité de calcul de distance.

B.2.2 Opérateur *carré* en notation classique

La réalisation matérielle du carré en notation classique, ne pose pas de problème. En utilisant l'algorithme *Direct*, sous-section 3.4.1, chaque produit partiel $2.a_i.a_j$ avec $(i \neq j)$ et $a_i.a_j$ avec $(i = j)$ correspond à une simple porte ET. Comme pour tous les autres multiplieurs classiques, la somme est effectuée avec un arbre de Wallace et la conversion $CS \rightarrow NR$ est réalisée avec un additionneur classique.

B.2.3 Carré en notation redondante

De manières similaires aux notations classiques, l'opérateur *carré* en notation redondante est basé sur le multiplieur traitant les mêmes notations, ici le multiplieur redondant présenté dans la sous-section 3.4.2.
L'utilisation de la propriété $a_i \cdot a_j = a_j \cdot a_i$ avec $i \neq j$, introduit une modification de la topologie de la matrice des produits partiels. Il se pose alors la question de la pérennité des règles de distributions des sous produits entre cellules. Pour mémoire, dans les multiplieurs redondants chaque produit partiel est décomposé en trois sous-produits ($a_i = 1$, $a_j = 1$, $a_i = 2$, $a_j = 1$ et $a_i = 1$,$a_j = 2$) qui sont distribués en fonction de leur poids sur les cellules voisines. Du point de vue structurel ces distributions correspondent à des décalages selon les lignes et les colonnes. Elles répondent à la volonté de réduire à un unique bit l'expression de chacun des produits partiels (sous-section 3.4.2).
Le remplacement des pairs de produits $a_i \cdot a_j$ et $a_j \cdot a_i$ par $2.a_i.a_j$ se traduit par la suppression des termes $a_j \cdot a_i$ et le décalage à droite des $a_i \cdot a_j$. Dans ce contexte, les décalages suivent la même règle et ne nécessitent aucune modification dans leur gestion. C'est vrai car le placement relatif des cellules est respecté. Par contre dans le cas des produits uniques $a_i.a_j$ ($i = j$), le placement relatif n'est pas respecté. Une modification des règles de distributions des sous-produits est nécessaire. Ces produits sont définis par l'ensemble de valeurs {0,1,4}. Les cellules $a_i.a_i$ n'ayant pas subit de décalage, le rapport entre la valeur des cellules $a_i.a_i$ et $a_{i+1}.a_i$ est passé de deux à quatre. La solution consiste donc à réduire les $a_i.a_i$ en deux sous-produits : $a_i.a_i = 1$ et $a_i.a_i = 4$, et de distribuer le second sur la cellule $a_{i+1}.a_i = 1$.
Globalement la réduction de la matrice n'entraîne qu'une légère modification de la distribution des sous-produits partiels. Pour illustrer ce propos, la figure B.2 présente comme exemple, la partie calcul des produits partiels de l'opérateur *carré redondant*, pour une entrée en notation CS codée sur six chiffres. La structure proposée reprend le nécessaire recodage de l'entrée, les cellules d'en-têtes et détaille la topologie des cellules composant la matrice des produits partiels ainsi que les décalages selon les lignes et les colonnes (flèches unidirectionnelles). Suivant le recodage utilisé, dans l'exemple proposé de 1^{er} type (figure 3.20), le CS recodé CS^r présente un certain nombre de chiffre $CS_i^r \in$ {0,1}. Cette particularité permet de dégrader un certain nombre de cellules. La dégradation est symbolisée sur la figure B.2 par l'absence de décalage.
Comme pour tous les autres multiplieurs, la somme des produits partiels est effectuée avec un arbre de Wallace. Le résultat est en notation CS.

Deux remarques :

1. Comme pour les multiplieurs redondants, dans le cas d'une entrée BS le complément à

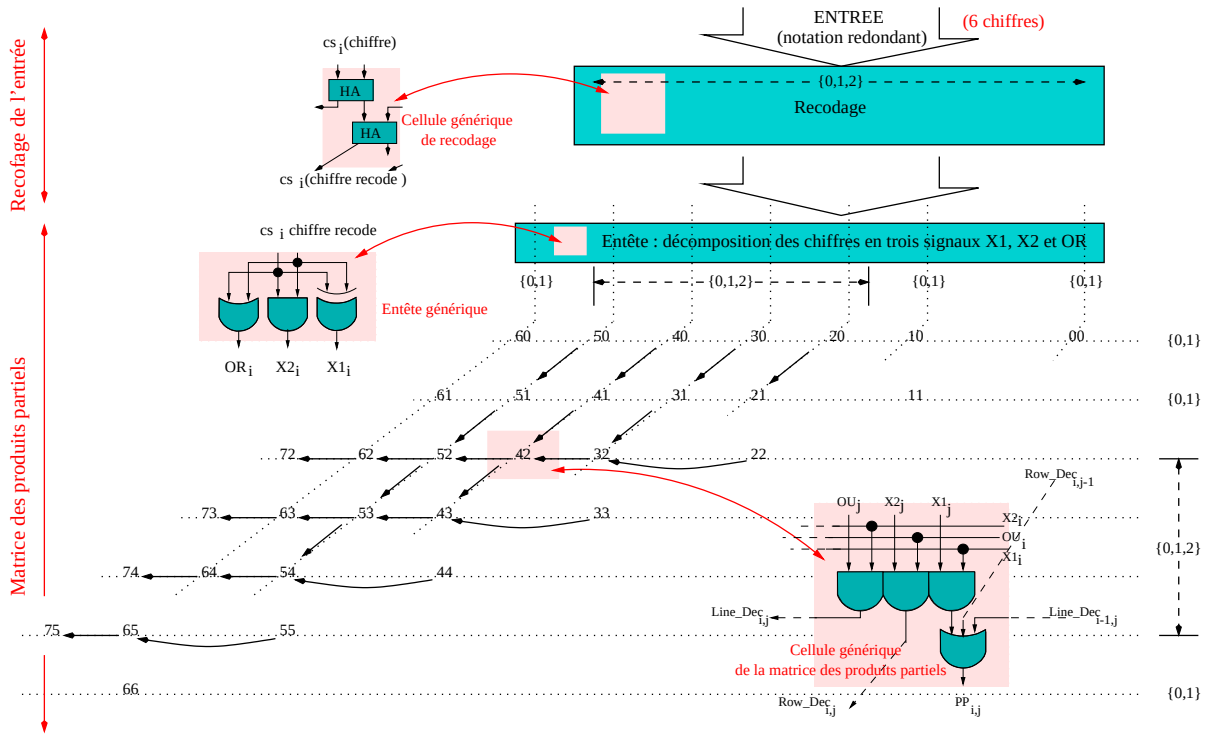


Figure B.2 – Carré redondant : détail du calcul des produits partiels pour une entrée sur 6 chiffres

deux du terme négatif BS^- est effectué en amont du recodage (figure 3.23).

2. En arithmétique classique la quantité et la topologie des produits partiels sont les mêmes exceptées pour les produits $7x$. Ceux-ci sont dus aux décalages.

B.3 Unité de calcul de distances (DCU)

Au-delà de l'opérateur carré, l'intérêt de cette annexe est de fournir un exemple d'une autre utilisation d'un opérateur redondant ou semi-redondant. Notre choix s'est porté sur le calcul de distances. La figure B.3.(a) présente l'architecture correspondant à l'unité de calcul de distances $\sum_i (A_i - B_i)^2$. Elle se décompose en deux parties. La première calcule les distances et la deuxième réalise la somme par accumulation.

Afin d'améliorer un peu plus les performances de la DCU, le résultat de l'accumulation est directement injecté dans le calcul de la somme des produits partiels du *carré* (section 6.3 sur la fusion). Une analyse rapide de l'architecture permet de constater que la première partie équivaut au carré d'un nombre en notation Borrow Save : $(A_i - B_i)^2 = (BS^+ - BS^-)^2$, à la différence près que la sortie est en notation non redondant. En substituant la première partie de la DCU

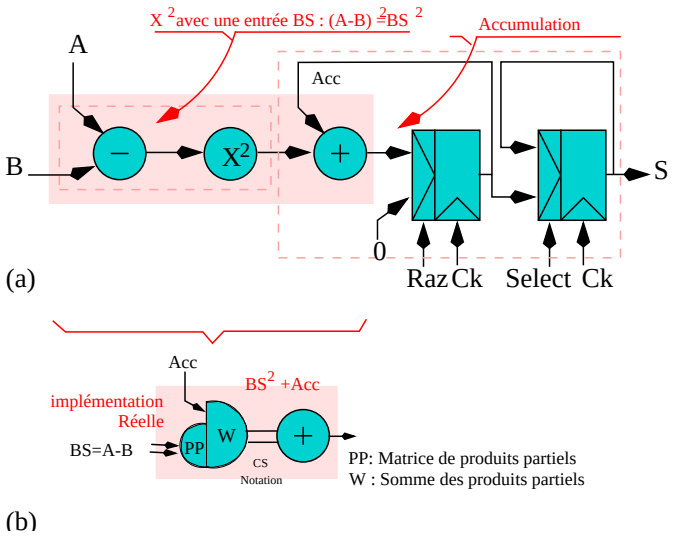


Figure B.3 – Unité de calcul de distance (DCU)

par le carré redondant et en injectant le résultat de l'accumulation dans le carré, on obtient l'architecture effectivement réalisée et présentée figure B.3.(b).

B.4 Résultats et comparaison

Cette partie est axée sur l'analyse de performance des diverses architectures proposées. L'analyse porte à la fois sur les réalisations du carré et de l'unité de calcul de distance en arithmétique classique et en arithmétique redondante.
Dans un premier temps et afin de situer les performances des diverses architectures réalisées, l'analyse porte sur un comparatif avec un multiplieur *direct* classique : $NR \cdot NR \rightarrow NR$, utilisé comme référence. Dans un deuxième temps, les réalisations classiques et redondantes des carrés et des DCU sont comparées.
Les valeurs de surface, de consommation de puissance et de délai sont regroupés dans la table B.1. Elles sont fonction de la taille des entrées. Pour chacune d'elle nous indiquons le gain en % par rapport à la référence. Les pourcentages entre parenthèses correspondent aux gains des circuits redondants face aux circuits classiques.

B.4.1 Performance des opérateurs $A^2_{NR,R}$

Comparaisons avec la référence :

Concernant le carré en notation classique et suivant la taille de l'entrée A, les gains sont compris entre 36% et 46% pour la surface, entre 26% et 15% pour le temps de propagation et entre 58% et

48% pour la consommation. Dans le cas du carré en notation redondante, les gains sont encore plus importants. En surface, le gain moyen est de 45%, en délai il est de 31%. La consommation elle, varie de 58% à 48%. Quel que soit le système de notation utilisé, l'ensemble de ces gains est très important. Ils démontrent l'intérêt de développer un opérateur dédié au carré dans le cas d'applications gourmandes de cette opération.

Comparaison entre carrés classiques et redondants :

Globalement la surface et la consommation sont équivalentes entre les deux réalisations. Selon ces deux critères et jusqu'à 24 bits, le carré redondant est légèrement plus intéressant. Pour un nombre sur 32 bits, les écarts de performances ne sont pas significatifs : environ 4%. La différence de performance entre ces deux réalisations réside essentiellement dans le temps de propagation. Le gain introduit par le carré redondant est compris entre 7% et 19% (sa moyenne est de 14%). Le carré redondant apparaît comme plus intéressant que le carré classique.

B.4.2 Performance des opérateurs de calcul de distances

Comparaisons avec la référence :

L'intérêt de cette comparaison est de situer les performances de l'unité de calcul de distances. En arithmétique classique, le délai et la puissance consommée sont supérieurs à ceux du multiplieur de référence, respectivement de 11% à 23% et de 18% à 23%. Dans le même temps, la surface est, elle, réduite de 30%. Dans le cas de la DCU redondante, le temps de propagation et la consommation sont pratiquement identiques à ceux de la référence. Parallèlement la surface décroît de 4% à 28% suivant la dynamique de l'entrée.

Comparaison entre les DCU classique et redondante :

La réalisation de l'unité de calcul de distance en arithmétique redondante offre de meilleures performances que celle en arithmétique classique. Le gain est compris entre 10% et 15% en délai et entre 12% et 21% en puissance consommée. Dans le même temps, l'augmentation de la surface est quasi constante et est proche des 16%. La différence d'architecture entre les carrés et les DCUs sont essentiellement l'ajout de l'additionneur d'entrée en arithmétique classique, et l'ajout de l'additionneur de sortie en arithmétique redondante. Les modifications des écarts de performances des carrés sont essentiellement dues à ces deux additionneurs. L'augmentation de la surface entre les $DCUs$ classique et redondante vient de la différence de taille (N) de leurs entrées. L'évolution du délai est aussi rattachée à cette valeur N puisque les deux additionneurs ont la même complexité $O(log_2(N))$ (additionneur *Sklansky*).

		Arithmétique classique			Arithmétique redondante		
Opérateur	taille entrée	Surface μM^2	Délai ns	Consommation $\mu W/Mhz$	Surface μM^2	Délai ns	Consommation $\mu W/Mhz$
$A \cdot B$	8	6890 (ref)	8.7 (ref)	119 (ref)	-	-	-
$A \cdot B$	16	26288 (ref)	12.9 (ref)	399 (ref)	-	-	-
$A \cdot B$	24	65929 (ref)	15.2 (ref)	809 (ref)	-	-	-
$A \cdot B$	32	137886 (ref)	17.0 (ref)	1448 (ref)	-	-	-
A^2	8	4410 -36%	6.4 -26%	50 -58%	3773 -45% (-14.4%)	6.0 -31% (-6.3%)	42 -65% (-16.0%)
A^2	16	15092 -43%	9.6 -26%	197 -51%	14822 -44% (-1.8%)	8.5 -34% (-11.5%)	171 -57% (-13.2%)
A^2	24	36027 -45%	12.5 -18%	433 -46%	37515 -43% (+4.1%)	10.2 -33% (-18.4%)	397 -51% (-8.3%)
A^2	32	74130 -46%	14.5 -15%	759 -48%	77175 -44% (+4.1%)	12.0 -30% (-17.2%)	785 -46% (+3.4%)
$(A-B)^2$	8	6259 -9%	10.7 +23%	147 +23%	7166 +4% (+14.5%)	9.9 +14% (-7.5%)	130 +9% (-11.5%)
$(A-B)^2$	16	19293 -27%	14.3 +11%	478 +20%	22491 -14% (+16.6%)	13.0 +1% (-9.0%)	386 -3% (-19.2%)
$(A-B)^2$	24	44222 -33%	16.9 +11%	961 +19%	51480 -22% (+16.4%)	14.8 -3% (-12.5%)	760 -6% (-20.9%)
$(A-B)^2$	32	86350 -37%	20.4 +20%	1713 +18%	99849 -28% (+15.6%)	17.5 +3% (-14.2%)	1426 -2% (-16.8%)

TAB. B.1 – Performances du carré et de l'unité de calcul de distance

B.5 Conclusion

Dans cette annexe nous avons présenté le développement de l'opération carré en arithmétique redondante, ainsi que la réalisation d'une unité de calculs de distances.
Disposer du carré en redondant permet d'augmenter le champ d'application de l'arithmétique mixte. Au-delà de cet opérateur l'objectif visé était d'illustrer le propos section 3.5 énonçant que les opérateurs redondants et semi-redondants, en plus de l'opération réalisée, offrent de nouvelles fonctionnalités en arithmétique classique.

Dans les deux cas, carré et unité de calcul de distance, les performances obtenues sont meilleures qu'en arithmétique classique. Ces résultats illustrent parfaitement l'intérêt d'utiliser les notations redondantes mais aussi l'intérêt de mêler/dissoudre les arithmétiques classique et redondante, en une seule arithmétique : l'arithmétique mixte.