République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université Akli Mohand Oulhadj Bouira

FACULTE DES SCIENCES
DEPARTEMENT GENIE ELECTRIQUE

MEMOIRE DE FIN D'ÉTUDES
pour l'obtention du diplôme de
MASTER

Domaine : **Sciences et Techniques**
Filière : **Génie Electrique**
Spécialité : **Systèmes électroniques complexes**

Thème :

Développement d'une Solution au Niveau "Bit" au Problème de la Multiplication par une Constante

Soutenu publiquement le
Par :
Labandji Mourad

Promoteur:
M^{elle} Sadji Aziza
Mr Medjdoub Smail

Encadreur
Dr A. Kamel Oudjida

Jury:
- **Dr. Madi Djemal MCA**
- **Saoud Bilal**
- **boucenna mohamed lamine**

Je dédie ce modeste travail :

A ma très chère mère pour son amour et qui n'a jamais cessé de prier pour moi.

A mon très cher père, pour ses encouragements, son soutien, surtout pour son amour et son sacrifice afin que rien n'entrave le déroulement de mes études.

A Mes très chères sœurs.

A Mes très chers frères.

Toute ma famille,

A tous ceux qui contribues de prés ou de loin à la réalisation de ce travail,

A tous ceux qui proche de mon cœur.

L.MOURAD

Remerciements

Nous remercions tout d'abord, le bon Dieu tout puissant de nous avoir donné la santé, la volonté, la force, le courage, et la puissance pour pouvoir surmonter les moments difficiles, et atteindre nos objectifs et sans lesquels notre projet n'aurait pas pu voir la lumière de ce jour.

J'exprime mes sincères remerciement à mon père, qui était mon soutien jusqu'au dernier jour de sa vie, merci bcp mon père, je remercie ma chère maman, elle est tous ce que j'ai maintenant dans ma vie.

Je souhaite vivement remercier et exprimer ma gratitude à mon encadreur Mr OUDJIDA pour m'avoir proposé ce sujet et qui je suis très reconnaissant pour ses remarques, ses orientations, son suivi pendant toute la réalisation de mon projet.

Je souhaite vivement remercier et exprimer ma gratitude à ma promotrice M^{elle} SADJI qui j'exprime mon profonde reconnaissance pour les temps précieux qu'ils m'a consacré, ses suivis, ses nombreuses interventions qui ont permis l'aboutissement de ce travail et qui je suis très reconnaissant pour ses remarques, ses orientations, et ses conseils pendant toute la réalisation de mon projet et même pour la période d'étude pendant les trois années.

Je tiens également à remercier mon co-promoteur Mr MEJDOUB, de m'avoir orientés significativement dans mon travail.

Je dis mille fois merci, à ma famille, pour leur soutien et leurs encouragements ainsi qu'à tous ceux qui ont contribué de prés ou de loin à la réalisation de ce modeste mémoire soit par un aide, un encouragement ou même un sourire.

Sans oublier de remercie très sincèrement mes chères amies pour tout quatre année d'étude en commun plein de bonheur, son encouragement énorme pour réussir a terminé ce projet.

Finalement, je remercie tout, les membres jury, devant qui j'ai l'honneur d'exposer mon travail, et qui ont pris la peine de lire avec soin ce mémoire pour juger son contenu.

ASIC	:	Application Specific Integrated Circuit
Ath	:	Adder Depth, nombre maximum des opérations d'additions série de l'entré à la sortie
Avg	:	Average number of additions
BASH	:	Bourne Again Shell
BIGE	:	Bounded Inverse Graph Enumeration, algorithme SCM optimal
CLB	:	Configurable Logic Bloc
CSD	:	Canonical Signed Digit
DAG	:	Directed Acyclic Graphs
DBNS	:	Double Base Number System
DSP	:	Digital-Signal-Processor/Processing
FA	:	Full Adder
FPGA	:	Field Programmable Gate Array
FWL	:	Finite Word Length
GCC	:	GNU Compiler Collection
Hcub	:	Cumulative Benefit Heuristic
HDL	:	Hardware Description Language
IOB	:	Input Output Block
LTI	:	Linear Time Invariant
MAC	:	Multiply-And-Accumulate
MCM	:	Multiple Constant Multiplication
MEMS	:	Micro Electro Mechanical Systems
NP-difficile	:	Non-deterministic Polynomial-time, problème complet pour la classe NP
OM	:	Odd Multiple
OS	:	Operating System
PP	:	Partial Product
RISC	:	Reduced Instruction Set Computer
RNS	:	Residue Number System
RTL	:	Register Transfer Level
SCM	:	Single Constant Multiplication
VHDL	:	Very high speed integrated circuit Hardware Description Language

Résumé

Beaucoup d'applications dans le traitement de signal, d'image, des systèmes de contrôle … etc. basés sur les systèmes LTI, ces systèmes sont basés généralement sur des multiplications par des constantes SCM/MCM qui consomme une quantité importante des ressources.

L'objectif de ce travail est le développement d'une solution au niveau bit pour la résolution de ces systèmes avec une optimisation des ressources consommées, la technique basée sur l'arithmétique RADIX-2r qui a des hautes capacités en matière de performance, et réduction du nombre des additions, de plus l'utilisation de cette heuristique au niveau bit permet plus de réduction sur les additionneurs de 1 bit, la solution décrire une description VHDL robuste et optimisé.

La solution développée est une plateforme qui peut être utilisée par tout un système LTI, elle assuré la construction de ce système avec une surface réduite, consommation de puissance réduit, et une vitesse élevée.

Mots clé : RADIX 2^r, SCM, MCM, LTI, filtre RIF.

Abstract

Many applications in signal processing, image processing, control systems, etc. based on LTI systems, these systems are generally based on multiplications by SCM / MCM constants that consume a significant amount of resources.

The objective of this work is the development of a solution at the bit level for the resolution of these systems with an optimization of the resources consumed, the technique based on arithmetic RADIX-2r which has high performance capacities, and reducing the number of additions, moreover the use of this heuristic at the bit level allows more reduction on the 1-bit adders, the solution describing a robust and optimized VHDL description.

The solution developed is a platform that can be used by an entire LTI system, it ensured the construction of this system with a reduced surface area, reduced power consumption, and high speed.

Key words: RADIX 2^r, SCM, MCM, LTI, FIR filter.

الملخص

العديد من التطبيقات في معالجة الإشارات ومعالجة الصور وأنظمة التحكم وما إلى ذلك. تعتمد غالبا علي عمليات الضرب في ثابت أو مجموعة من الثوابت، التي تستهلك الكثير من الموارد .

والهدف من هذا العمل هو تطوير حل على مستوى البتات لهذه الأنظمة مع الاستفادة المثلى من الموارد المستهلكة، اعتمادا علي تقنية RADIX 2^r التي تتميز بالأداء العالي و الحد الأدنى من عمليات الجمع، اضافة الى هذا فإن استخدامها علي مستوى 1 بت تسمح بدرجة أكبر من الخفض، مع ملف وصفي VHDL قوي و أكثر دقة . هذا الحل هو منصة يمكن استعمالها في كل أنظمة LTI، فهو يضمن انشاء نظام ذو مساحة منخفظة، سرعة في الأداء، مع استهلاك أقل للطاقة

الكلمات المفتاحية: RADIX 2^r, SCM, MCM, LTI, FIR filter

Table des figures

List des tableaux

Sommaire

Chapitre III SCM/MCM au niveau bloc d'additionneur

Chapitre IV SCM/MCM au niveau bit

Chapitre V Application du SCM/MCM aux filtres RIF

Introduction générale

Le domaine des applications MEMS a la particularité d'être très sensible au bruit. Afin d'atteindre la précision souhaitée dans la préhension des objets micrométriques, le développement des méthodes avancées de contrôle est nécessaire. Ce dernier se traduit souvent par des lois de commande, des filtres et des algorithmes qui sont exécutés en temps réel sur les plates-formes qui sont volumineux et coûteux et présentent un grand inconvénient d'empêcher toute utilisation embarquée ou mobile (autonomie). Par conséquent, l'intégration matérielle de ces lois de commande et tous les problèmes y afférents constituent un véritable défi.

Notre travail aborde le problème d'optimisation des ressources matérielles au niveau bit de systèmes linéaires invariant dans le temps (Linear Time Invariant (LTI)) avec une longueur de mot finie (Finite Word Length (FWL)). L'effort d'optimisation de notre travail se situe au niveau algorithmique et non pas architecturale. Le plus grand défi est d'assurer un contrôle satisfaisant avec un nombre de ressources hardware le plus réduit possible. A ce stade deux optimisations distinctes sont possible bien que complémentaires : théorie de contrôle et l'arithmétique binaire. Seule cette dernière est associée à notre projet.

L'arithmétique binaire doit être à la fois assez rapide pour faire face à la dynamique rapide des MEMS, à faible consommation d'énergie pour des applications embarquées (utilisation d'une batterie), hautement évolutive (scalable) en terme de performances, et facilement prévisible pour donner une idée précise sur les ressources logiques nécessaires avant la mise en œuvre.

L'exploration d'un certain nombre d'arithmétiques binaires a montré que radix-2^r est le meilleur candidat qui correspond aux exigences susmentionnées. Il a été pleinement exploité à concevoir des noyaux de multiplicateur (×) efficace, qui sont le véritable moteur des systèmes linéaires.

Pour mener bien notre projet, nous avons structuré notre mémoire en Cinq chapitres :

- Le premier chapitre inclut les fondements de l'arithmétique binaire .Tout d'abord les formats de nombre généralement utilisés format à virgule fixe et format à virgule flottante, leur précision et leur gamme dynamique respectives, Nous présentons ensuite les systèmes de numération les plus couramment utilisés. Nous insistons surtout sur les deux opérations arithmétique (+, ×) requis par les systèmes linéaires.

- Le deuxième chapitre est consacré à l'opération de multiplication par une constante. Il présente une formulation des problèmes SCM/MCM (Single Constant Multiplication / Multiple Constant Multiplication) suivi par l'introduction d'une nouvelle heuristique appelée RADIX-2r.

- Le troisième chapitre traite le problème SCM/MCM au niveau block d'additionneurs (adder-bloc). Nous présentons notre solution développée dans le cadre de l'heuristique RADIX-2^r. Nous montrons son efficacité par des résultats expérimentaux.

- Le quatrième chapitre traite par contre le problème SCM/MCM au niveau bit d'additionneur (full-adder). Nous analyserons notre solution par une étude aussi bien analytique qu'expérimentale.
- Dans le chapitre 5, nous appliquons les résultats de la recherche développée dans les chapitres précédents pour les filtres FIR .

A la fin on terminera par une conclusion générale et perspective.

Chapitre I :

L’arithmétique binaire

I.1 Introduction

Ce chapitre aborde les fondements de l'arithmétique binaire. Nous décrivons les représentations les plus utilisés dans les systèmes numériques, leurs précisions, et leurs gammes dynamiques.

Ensuite, nous passerons vers les principaux systèmes de numération qui sont basés sur le format en virgule fixe, notamment le système de numération de résidus et le radix-2^r, pour les opérations arithmétiques l'addition et de multiplication. Nous insisterons sur l'importance de ces deux opérations dans le calcul et la conception hardware des systèmes linéaires.

I.2 L'arithmétique binaire

L'arithmétique binaire est un domaine mathématique basé principalement sur :

- ✓ L'étude des systèmes de représentation des nombres afin d'éliminer la redondance logique dans le recodage ;
- ✓ La recherche des algorithmes qui effectue efficacement les opérations arithmétiques (–, +, ×, /, $\sqrt{\ }$, a^netc.) dans un système numérique donné ;
- ✓ L'exploration des meilleures techniques de mise en œuvre lorsque le système numérique est un DSP, un FPGA ou un ASIC.

En arithmétique binaire, les nombres sont représentés généralement à l'aide des deux symboles : 0 et 1.

Dans les systèmes de représentation des nombres, les symboles sont appelés chiffres, et le nombre de symboles est appelée '' base'' ou ''radix''.

Les nombres binaires sont exprimées en base 2, tandis que par exemple les nombres décimaux sont représentés en base 10, puisque dix chiffres sont utilisés (0, 1, ..., 9).

La notation utilisée couramment pour représenter des nombres est : (x) y, où x est un nombre exprimé dans la base y.

Exemple :

(10) 10 représente le nombre dix dans le système décimal ; (10) 2 représente le numéro deux dans le système binaire

Remarque : *les recodage octal (base 8), hexadécimal (base 16), etc.…, ne servent qu'à faciliter la manipulation de longues chaînes de 0 et de 1. La mise en œuvre est réalisée en binaire (radix-2).*

Il existe différents types d'arithmétique binaire. Les caractéristiques de chaque calcul sont conditionnées par le format utilisé pour représenter les nombres. Les arithmétiques les plus souvent utilisées sont : arithmétique à virgule flottante et arithmétique à virgule fixe. Les implémentations et les optimisations menées dans chaque arithmétique sont différentes [1].

Le choix de l'arithmétique à utiliser est principalement dicté par la précision souhaitée et la gamme dynamique imposé par le problème à résoudre.

En outre, le niveau de complexité de l'arithmétique est déterminé par les opérations requises. Ce dernier dépend fondamentalement du type du système considéré : système linéaire ou non linéaire.

Un système linéaire est un modèle mathématique basé sur des opérations linéaires (-, +, ×, /). Il présente généralement des caractéristiques et des propriétés qui sont beaucoup plus simple et

plus facile à comprendre et à manipuler que le cas non linéaire qui nécessite des opérations complexes, tels que polynôme ou les fonctions trigonométriques.

I.3 Les Formats de représentation des nombres

Il existe différents formats de représentation pour les nombres [4,8]. Les formats les plus utilisés sont résumés comme suit :

• **Format à virgule fixe** : propose une gamme limitée et/ou de précision, mais facile à implémenter. Très pratique pour les applications à haute vitesse et faible consommation de puissance. Il manipule des nombres entiers $x \in I = \{-N,...,N\}$ ainsi que des nombres rationnels de la forme $x = a/2^f$ (''binaire'' rationnel), $a \in I$ et f est un entier positif.
• **Format à virgule flottante :** il s'agit de l'approche la plus courante. Elle offre une gamme dynamique large avec une grande précision pour pouvoir manipuler des nombres extrêmement grands et petits, respectivement.

Toutefois, il est relativement difficile à implémenter. Il gère les nombres de la forme $x \times b^E$, où *x* est un nombre rationnel, *b* est la base en entier naturel, et *E* est un exposant entier naturel.
• **Format logarithmique** : représente les nombres par leurs signes et leurs logarithmes. Attractif pour des applications qui ont besoin d'une faible précision et une plage dynamique étendue.
• **Format rationnel :** se rapproche à une valeur réelle par le rapport de deux entiers. Conduit à des opérations arithmétiques qui sont difficiles à implémenter.

Nous nous limitons aux représentations à virgule fixe et à virgule flottante.

I.3.1 Format à virgule fixe

Les données en format virgule fixe sont représentées comme étant des nombres fractionnaires à virgule fixe (exemple -1.0 à 1.0), ou comme des entiers classiques.Il existe deux types pour ce format, format virgule fixe pour les entiers non signés et le format virgule fixe pour les entiers signés.

I.3.1.1 Format à virgule fixe : entiers non signés

La représentation d'un entier *x* est notée par le vecteur suivant :

$$x = \left(x_{n-1}, x_{n-2}, \ldots, x_1, x_0\right) \qquad \text{(I.1)}$$

L'indexation utilisée à l'origine zéro est concerné le digit le plus à gauche.
Un système de numération pour représenter *x* se compose des éléments suivants :

- ✓ Le nombre de chiffres *n*.
- ✓ Un ensemble de valeurs numériques pour les chiffres. Nous appelons Di l'ensemble des valeurs des x_i.

  Le cardinal de Di est dénoté par l'ensemble de valeurs $|D_i|$ de xi. Par exemple, {0,1, 2, ..., 9} est le chiffre fixé pour les systèmes de numération décimal classiques avec cardinalité 10.

- ✓ Une règle d'interprétation qui correspond à un mappage entre l'ensemble des vecteurs-chiffres et l'ensemble de valeurs des entiers.

L'ensemble des entiers, chacun représenté par un vecteur à n chiffres.

Pour cela la représentation de nombre est comme suit :

$$x = \sum_{i=0}^{n-1} x_i \times w_i \qquad \text{(I.2)}$$

où $w = (w_{n-1}, w_{n-2}, ..., w_1, w_0)$ est le vecteur-poids.

Dans laquelle le vecteur-poids est lié au vecteur- Radix $r = (r_{n-1}, r_{n-2}, \ldots, r_1, r_0)$ comme suit :

$$w_0 = 1; \quad w_i = w_{i-1} \times r_{i-1} \quad (1 \le i \le n-1) \qquad \text{(I.3)}$$

Cela équivaut à :

$$w_0 = 1; \quad w_i = \prod_{j=0}^{i-1} r_j \qquad \text{(I.4)}$$

Les Systèmes de numération de base sont classés selon le vecteur-radix en systèmes fixe-radix et en systèmes mixtes-radix.

Dans un système de radix-fixe, tous les éléments du vecteur-radix ont la même valeur r (la base). Par conséquent, le vecteur-poids comme suit :

$$w = \left(r^{n-1}, r^{n-2}, ..., r^2, r\ 1\right) \qquad \text{(I.5)}$$

Et les jeux de chiffres sont :

$$D_i = D \qquad (1 \le i \le n-1) \qquad \text{(I.6)}$$

et

$$x = \sum_{i=0}^{n-1} x_i \times r^i \qquad \text{(I.7)}$$

Les bases les plus fréquemment utilisées sont des puissances de deux, par exemple 2 (binaire), 4 (quaternaire), 8 (octal), 16(hexadécimal), et ainsi de suite. La gamme correspondante de x représentée avec n chiffres radix-r est :

$$0 \le x \le r^n - 1 \qquad \text{(I.8)}$$

Selon l'ensemble des données numériques (Di), les systèmes de numération de base sont classés en des systèmes redondants et des systèmes non redondants.

Un système de numération est non redondant si chaque vecteur-chiffre représente un autre nombre entier ; autrement dit, si le mappage de représentation est un à un.

Il est redondant s'il existe des entiers qui sont représentée par plus d'un vecteur-chiffre.

I.3.1.2 Format à virgule fixe : entiers signés

Pour les entiers signés (positifs et négatifs) deux représentations courantes :

- **La représentation du signe et magnitude (SM)** : en SM, un entier signé x est représenté par un couple (x_s, x_m), où x_s est le signe et la x_m est la magnitude (entier positif).

Les deux valeurs de signe (+, -) sont représentées par une variable binaire, « 0 » correspond à un plus (+) et « 1 » pour un moins (–).

Si on utilise un système de radix-r classique, la gamme des entiers, pour n chiffres signés, est $0 \le x_m \le r^n - 1$. Notez que zéro a deux représentations : $x_s = 0$, $x_m = 0$ (zéro positif) et $x_s = 1$, $x_m = 0$ (zéro négatif).

- **La représentation par un complément** : dans ce système, il n'y a pas de séparation entre la représentation du signe et la représentation de l'ampleur, mais l'ensemble entier signé est représenté par un entier positif. Les représentations des entiers positifs sont appelées formes véritables et ceux des entiers négatif, les formes de complément. Le complément est exprimé dans la cas général en radix-r, nous ne considérons que le cas particulier de radix-2, appelé les deux représentations du complément.

Dans la représentation du complément à deux, un entier x signé (bit vecteur) avec n chiffres est représenté comme suit :

$$x = -x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i \times 2^i \qquad \text{(I.9)}$$

Pour convertir un vecteur de bits d'une valeur, nous utilisons le fait que le bit le plus significatif (x_{n-1}) de x a un poids négatif, tandis que les bits restants ont des poids positifs.

Exemple :

$$x = (11011) \rightarrow -16 + 8 + 2 + 1 = -5 .$$

La représentation complément à deux (EQ I.9) possède les propriétés suivantes :

• La représentation sous forme de zéro est unique. Zéro est obtenue lorsque tous les chiffres sont mises à zéro.

• La gamme des nombres n'est pas symétrique puisque $x = -2^{n-1}$ est représentable mais $x = 2^{n-1}$ n'est pas représentable. La gamme est $\left[-2^{n-1}, 2^{n-1} - 1\right]$.

I.3.1.3 Arithmétique virgule fixe pour le complément à deux

Avant de décrire les opérations arithmétiques, nous devons d'abord officialiser la représentation des nombres au format à virgule fixe [27].

-Soit β le nombre de chiffres (bits) d'un nombre signé (bit-vector) x, et γ le nombre de chiffres de la partie décimale de x (Fig. I.1).

- Soit α le nombre de chiffres de la partie entière (α = β – γ). Enfin, FPRx désigne le triple (α, β, γ) définissant la représentation à virgule fixe de x. par conséquent, $x = x_{Int} + x_{fr}$ s'écrit $x = \left(x_{\alpha-1} \ldots x_1 x_0 \bullet x_{-1} \ldots x_{-\gamma+1} x_{-\gamma}\right)$, telle que

$$x = -x_{\alpha-1} \times 2^{\alpha-1} + \sum_{i=-\gamma}^{\alpha-2} x_i \times 2^i \qquad \text{(I.10)}$$

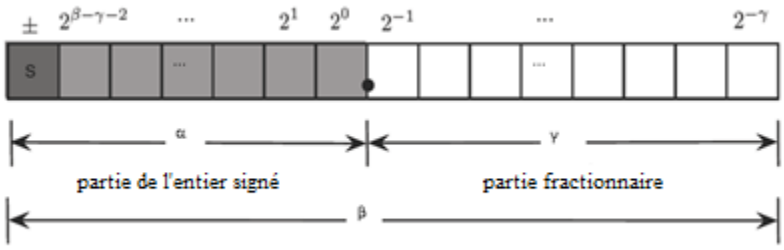


Figure I.1 format virgule fixe pour les nombres signés en complément à deux[8].

Pour convertir un nombre réel x FPR, nous procédons comme suit :

- B est une valeur donnée, puisqu'elle doit être égale à la largeur en bits du chemin de données.
- Le nombre de bits de la partie entière est donné par :

$$\alpha_x = \lfloor \log_2 |x| \rfloor + 2 \qquad \text{(I.11)}$$

Avec $\lfloor a \rfloor$ est la fonction floor qui arrondit au nombre entier inférieur ou égal le plus proche à x. par conséquent, nous déterminons la partie fraction comme $\gamma_x = \beta_x - \alpha_x$, et le FPR de x est donné par : $FPR_x = (\beta_x, \alpha_x, \gamma_x)$

Par contre, dans la représentation en virgule fixe, x est représentée par l'entier Nx tels que :

$$N_x = \lfloor x \times 2^{\gamma_x} \rceil \qquad \text{(I.12)}$$

où $\lfloor a \rceil$ est la fonction qui arrondit au nombre entier le plus proche. Ainsi, le x est approximé par

$$x^{\dagger} = N_x \times 2^{-\gamma_x} \qquad \text{(I.13)}$$

L'addition et la multiplication sont les deux opérations arithmétiques les plus utilisées dans les systèmes linéaires. En format virgule fixe, elles sont gérées comme suit.

I.3.1.4 La Multiplication en virgule fixe

Considérons l'opération $z = x \times y$,avec $(\beta_x, \alpha_x, \gamma_x)$ *et* $(\beta_y, \alpha_y, \gamma_y)$. FPRz est donnée par :

$$FPR_z = (\beta_x + \beta_y, \alpha_x + \alpha_y, \gamma_x + \gamma_y) \qquad \text{(I.14)}$$

et l'opération de multiplication est réalisée par $N_z \leftarrow N_x \times N_y$. Dans l'équation 3.14 la multiplication est effectué en double précision depuis $\beta_z = \beta_x + \beta_y$, mais généralement le bit-largeur (W_{dp}) du chemin de données- est plus petit que $(\beta_x + \beta_y)$. Dans ce cas, laissez-nous désigner $\beta_{op} = W_{dp}$ et donner l'expression générale de FPRz :

$$FPR_z = (\beta_{op}, \alpha_x + \alpha_y, \beta_{op} - \lceil \alpha_x + \alpha_y \rceil) \qquad \text{(I.15)}$$

et l'opération est réalisée par

$$N_z \leftarrow (N_x >> s_x) \times (N_y >> s_y) \qquad \text{(I.16)}$$

où s_x *et* s_y et le bit droit déplacements appliqués sur Nx et Ny telle que $s_x + s_y = (\beta_x + \beta_y) - \beta_{op}$. Mais Si $\beta_{op} = (\beta_x + \beta_y)$, $s_x = s_y = 0$. Les cas spéciaux et généraux, données par l'équation I.14

et I.15, respectivement, sont illustrés par la figure I.2. Un nombre de bits Scst est tronqué de l'opérande Y de tel cette $\beta_{op} = W_{dp}$.

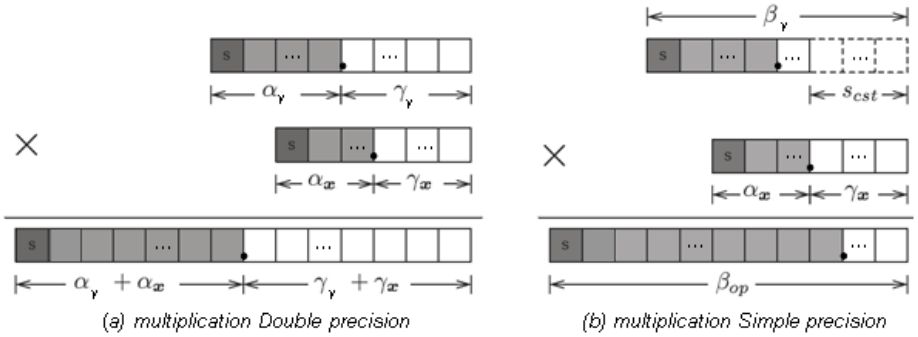


Figure I.2 (a) Double et (b) simple précision[8]

I.3.1.5 Addition en virgule fixe :

L'addition de deux nombres à virgule fixe nécessite la même position du point. La précision complète est donnée par :

$$\begin{cases} \alpha_z = \max(\alpha_x + \alpha_y) + 1 \\ \gamma_z = \max(\gamma_x + \gamma_y) \\ \beta_z = \alpha_z + \gamma_z \end{cases} \quad \text{(I.17)}$$

Le cas général (précision limitée) où est donnée par :

$$\begin{cases} \alpha_z = \max(\alpha_x + \alpha_y) + 1 \\ \gamma_z = \beta_{op} - \max(\alpha_x + \alpha_y) - 1 \\ \beta_z = \beta_{op} \end{cases} \quad \text{(I.18)}$$

et l'opération d'addition est réalisée par

$$N_z \leftarrow (N_x >> s_x) \times (N_y >> s_y) \quad \text{(I.19)}$$

Avec $s_x = \gamma_x - \gamma_z$ *et* $s_y = \gamma_y - \gamma_z$

I.3.1.6 Détection d'un Débordement (overflow)

Dans la représentation du complément à deux, un dépassement de capacité se produit lorsque les opérandes sont de même signe et le résultat de l'addition représente un nombre entier de signe différent. Étant donné que le signe est déterminé par le bit le plus significatif (x_{n-1}), la détection de dépassement de capacité est spécifiée par l'expression suivante :

$$AVF = (\bar{x}_{n-1} \times \bar{y}_{n-1} \times z_{n-1}) + (x_{n-1} \times y_{n-1} \times \bar{z}_{n-1}) \quad \text{(I.20)}$$

I.3.2 Format à virgule flottante

La représentation en virgule flottante est utilisée pour la représentation des nombres réels. Comme dans la représentation en virgule fixe, la représentation en virgule flottante est codée sur un nombre fini de bits. Pour un système spécifique à virgule flottante, un nombre réel qui est exactement représenté dans le système est appelé un nombre en virgule flottante. Dans le cas d'un dépassement de capacité positifs et négatifs les nombres sont représentés par des nombres à virgule flottante qui ont une valeur qui se rapproche du nombre réel. Le processus de rapprochement est appelé arrondissement et génère une erreur.

Un nombre x en virgule flottante est représenté par un triple *(S_x, Mx, E_x)*, tels que :

$$x = (-1)^{S_x} \times M_x \times b^{E_x} \quad \text{(I.21)}$$

où b est une constante appelée la base ; Ex est un exposant entier signé ; MX est la mantisse ; et $S_x \in \{0,1\}$ est le signe de la mantisse.

I.3.2.1 La gamme dynamique

Le but d'utiliser la représentation à virgule flottante est d'augmenter la gamme dynamique, par rapport à une représentation en virgule fixe. Cette gamme dynamique est définie comme le rapport entre le nombre le plus grand et le nombre le plus petit non nuls et positif qui peut être représentée [1]. Pour obtenir une représentation en virgule fixe à l'aide de digits de n radix-r , la gamme dynamique est :

$$DR_{fxpt} = r^n - 1 \quad \text{(I.22)}$$

En revanche, pour la représentation à virgule flottante :

$$DR_{ftpt} = \frac{M_{\max} \times b^{E_{\max}}}{M_{\min} \times b^{E_{\min}}} \quad \text{(I.23)}$$

Par exemple, si les chiffres de n sont partitionnés ainsi que m chiffres sont utilisés pour les chiffres mantisse et n-m pour l'exposant et b = r, nous obtenons :

$$DR_{ftpt} = \left(r^m - 1\right) \times r^{\left(r^{n-m}-1\right)} \quad \text{(I.24)}$$

Par exemple, si n = 32, m = 24 et r = 2, les plages dynamiques correspondantes sont :

$$DR_{fxpt} = 2^{32} - 1 \approx 4.3 \times 10^{9}$$
$$DR_{ftpt} = \left(2^{24} - 1\right) \times 2^{2^{8}-1} \approx 9.7 \times 10^{83}$$

Une large gamme dynamique est requise dans de nombreuses applications pour éviter les dépassements de capacité positifs et négatifs. Si la gamme dynamique de la représentation à virgule fixe n'est pas suffisante, des opérations complexes doivent être incluses dans le programme. Ainsi, un système à virgule flottante est préférable dans de telles applications.

I.3.2.2 Précision

Dans la représentation en virgule flottante, Pour la même plage de valeurs représentables, virgule flottante a tendance à être mieux que virgule fixe en termes de l'erreur moyenne.

Bien que la représentation en virgule flottante fournisse une grande gamme dynamique et une précision supérieure à la représentation en virgule fixe, elle est beaucoup plus coûteuse à mettre en œuvre. Dans les applications des systèmes embarquées tolérant un certain degré d'imprécision, la notation à virgule fixe est la plus utilisées pour augmenter le débit et diminuer la superficie, le retard, et l'énergie.

I.4 Les systèmes de représentation des nombres

Les systèmes de numération sont développés afin de permettre une réduction pour la complexité des opérations arithmétiques. Chaque système de numération présente des propriétés numériques spécifiques, les opérations arithmétiques sont gérées différemment parmi ces systèmes de numération on peut trouver.

I.4.1 La représentation numérique canonique des nombres signés (CSD)

Est la forme de la représentation canonique des entiers signés développée par Avizienis [11,8] en 1961.
Est un radix-fixe redondant (r = 2) sa représentation définie comme suit :

$$x = \sum_{i=0}^{n-1} x_i \times 2^i \quad avec \quad x_i \in D = \{\bar{1}, 0, 1\} et \ \bar{1} = -1 \qquad (I.25)$$

L'avantage principal de ce système est que les opérations d'addition/soustraction peuvent être faites sans rapport de propagation, et avec un traitement accéléré, notamment pour les grands opérandes.

Dans le cas des entiers signés, l'entier « sept », par exemple, possède plusieurs représentations :
$(0111)_2 = 4+2+1 = 7$, $\left(10\bar{1}1\right)_2 = 8-2+1 = 7$, $\left(1\bar{1}11\right)_2 = 8-4+2+1 = 7$, $\left(100\bar{1}\right)_2 = 8-1 = 7$

Le nombre rationnel « 5/8 », par exemple ; peut-être écrit différemment :

$$\frac{5}{8} = 0.625 = (0.101)_2 = \left(1.\bar{1}01\right)_2 = \left(1.\bar{1}1\bar{1}\right)_2 = \left(1.0\bar{1}\,\bar{1}\right)_2$$

La conversion de la SD en CSD se faite comme suit :

- ✓ Convertir une longue chaîne de 1.
- ✓ Fusionner des chiffres adjacents de signes opposés : en utilisant le format de la CSD sur une valeur de n bits, le nombre de chiffres différent de zéro est délimité par les (n + 1) / 2 et il tend asymptotiquement à une valeur moyenne de (n/3) +(1/9). Par rapport à la représentation binaire traditionnelle nécessitant n/2 chiffres sur la moyenne, CSD permet une économie de 33 % en chiffre différent de zéro. Cela signifie que, dans la multiplication constante, 33 % moins d'additions/soustractions sont nécessaires, ce qui conduit à beaucoup compact implémentations de systèmes LTI [2]. C'est la raison pour laquelle les CSD est populaire.

Malgré le fait que le CSD minimise le nombre de chiffres différent de zéro pour la représentation de la constante, c'est loin d'être optimale. Il est possible de décomposer la valeur de n bits pour réduire davantage le nombre d'opérations. Ceci peut être réalisé à l'aide de plusieurs systèmes de nombre complexe.

I.4.2 Système de numération à double Base (DBNS)

L'arithmétique DBNS a été développé par Dimitrov en 1999 [12]. Dans DBNS, un entier x est exprimé à l'aide des bases 2 et 3, comme suit :

$$x = \sum_{i,j} d_{i,j} \times 2^i \times 3^j \quad Avec \quad d_{i,j} \in D = \{0,1\} \tag{I.26}$$

Par exemple, le nombre entier x = $(10603)_{10}$ est écrit en DBNS comme suit :

$$x = (3^2\times2^8)+(3\times2^5)+(3^0\times2^{13})+(3^0\times2^3)+ (3^1\times2^0)= (10603)_{10}.$$

Selon EQ. I.2 et I.3, La représentation en DBNS est extrêmement redondante. La représentation canonique de DBN (CDBNR) qui exprime un entier donné comme une somme d'un nombre minimal de 2-nombres entiers est très complexe à déterminer (problème NP-complet). Ainsi, les opérations arithmétiques dans ce système de numération ne garantissent pas que les résultats qui sont obtenus soient sous la forme minimale.

I.4.3 Système de numération de résidus (RNS)

Le concept de RNS remonte à 1500 ans en Chine [4]. Un nombre x est représenté par le vecteur de ses résidus en ce qui concerne les premiers modules k $m_{k-1} > m_{k-2} > \cdots > m_1 > m_0$. Le résidu xi de x à l'égard de l'i-ème module mi est semblable à un digit, et la représentation de l'ensemble k-résidu de x peut être considérée comme un k-digits. Sans, nous écrire

$$x = \left(x_{k-1} \mid x_{k-2} \mid \cdots \mid x_1 \mid x_0\right)_{RNS\,(m_{k-1}|m_{k-2}|\cdots|m_1|m_0)}$$
$$x_i = x \bmod m_i = \langle x \rangle_{m_i} \quad avec \quad x_i \in \{0,1,2,\ldots, m-1\} \tag{I.27}$$

Le vecteur des modulos $_{RNS(m_{k-1}|m_{k-2}|\cdots|m_1|m_0)}$ peuvent être supprimés de l'indice lorsque nous nous sommes mis d'accord sur un ensemble par défaut. Le produit M des premiers modulos

de k est le nombre de différentes valeurs représentables dans le RNS et est connu comme sa gamme dynamique :

$$M = m_{k-1} \times m_{k-2} \times \cdots \times m_1 \times m_0 \tag{I.28}$$

Par exemple, $M = 8 \times 7 \times 5 \times 3 = 840$ est le nombre total de valeurs distinctes qui sont représentables dans $RNS = (8 \mid 7 \mid 5 \mid 3)$. À cause de l'égalité

$$\langle -x \rangle_{m_i} = \langle M - x \rangle_{m_i} \tag{I.29}$$

Les 840 valeurs disponibles peuvent être utilisées pour représenter les nombres de 0 jusqu'à 839, ou de-420 jusqu'à 419, ou tout autre intervalle de 840 entiers consécutifs. En effet, les nombres négatifs sont représentés à l'aide d'un système de complément avec la constante de complémentation M. Voici quelques exemples de chiffres dans (8 | 7 | 5 | 3) :

(0 | 0 | 0 | 0)$_{\text{RNS}}$ *Represente 0 or 840 or . . .*

(1 | 1 | 1 | 1)$_{\text{RNS}}$ *Represente 1 or 84 1 or . . .*

(2 | 2 | 2 | 2)$_{\text{RNS}}$ *Represente 2 or 842 or . . .*

(0 | 1 | 3 | 2)$_{\text{RNS}}$ *Represente 8 or 848 or . . .*

(5 | 0 | 1 | 0)$_{\text{RNS}}$ *Represente 21 or 861 or . . .*

(0 | 1 | 4 | 1)$_{\text{RNS}}$ *Represente 64 or 904 or . . .*

(2 | 0 | 0 | 2)$_{\text{RNS}}$ *Represente −70 or 770 or . . .*

(7 | 6 | 4 | 2)$_{\text{RNS}}$ *Represente −1 or 839 or . . .*

Représentation RNS n'est pas redondante au sein de l'intervalle choisi par 840 entiers consécutifs.

I.4.3.1 Addition et Multiplication

Le signe d'un nombre RNS peut être changé en complétant indépendamment chacun de ses chiffres en ce qui concerne son modulo. De même, l'addition, la soustraction, et la multiplication peuvent être effectuées en agissant indépendamment sur chaque chiffre. Les exemples suivants pour RNS (8 | 7 | 5 | 3) illustrent le processus d'addition, soustraction et multiplication, respectivement :

(5 | 5 | 0 | 2) RNS représente x = + 5
(7 | 6 | 4 | 2) RNS représente y = −1
(4 | 4 | 4 | 1) RNS x + y :,, etc.
(6 | 6 | 1 | 0) RNS x−y :,, etc.
(3 | 2 | 0 | 1) IA x× y :,, etc.

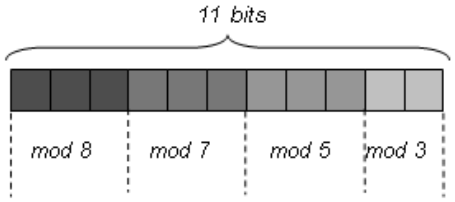


Figure I.3 Format RNS pour le nombre (8|7|5|3)[8]

La vitesse et la simplicité sont les principaux avantages des opérations arithmétiques. Dans le cas de l'addition, par exemple, la propagation de retenu est limité à un seul résidu (quelques segments). Ainsi, la représentation RNS a résolu à peu près le problème de propagation de la retenue. En ce qui concerne la multiplication, un multiplicateur de 4 x 4 par exemple, est quatre fois plus simple qu'un multiplicateur de 16 x 16, en plus d'être beaucoup plus rapide. En fait, étant donné que les résidus sont légers (par exemple 6 bits de largeur), il est tout à fait possible d'appliquer l'addition, la soustraction, et la multiplication par recherche dans la table directement. Avec des résidus de 6 bits, chaque opération nécessite une table de 4 Kx6. Ainsi, à l'exception de la division, un module d'unité arithmétique complète pour un résidu de 6 bits peut être implémenté avec 9 Ko de mémoire.

Malheureusement, ce que nous gagnons en ce qui concerne la vitesse et la simplicité d'implémentation de l'addition, la soustraction, et la multiplication, peut être réduit à néant par la complexité de la division et la difficulté de certaines opérations auxiliaires telles que le test du signe, Comparaison de magnitude et détection de dépassement de capacité.

I.4.4 Système de numération radix-2^r

La représentation de radix-2^r a été développée par Sam en 1990 [13]. en radix-2^r, le complément à deux d'un nombre x est comme suit :

$$x = \sum_{j=0}^{(n/r)-1} \left(x_{rj-1} + 2^0 x_{rj} + 2^1 x_{rj+1} + 2^2 x_{rj+2} + \cdots + 2^{r-2} x_{rj+r-2} - 2^{r-1} x_{rj+r-1} \right) \times 2^{rj}$$

$$= \sum_{j=0}^{(n/r)-1} Q_j \times 2^{rj} \quad \text{Avec} \quad Q_j \in D = \{-2^{r-1}, -2^{r-1}+1, \cdots, -1, 0, 1, \cdots, 2^{r-1}-1, -2^{r-1}\} \qquad \text{(I.30)}$$

Quand $x_{-1} = 0$ et $r \in \mathrm{N}^*$. Pour une simplicité et sans perte de généralité, nous supposons que r est un diviseur de N. En EQ. I.30, de complément à deux de x est divisée en n/r complément à deux segments, chacune des r + 1 bit de longueur. Chaque paire de deux segments continues a un bit qui se chevauchent.

En fait, la représentation SD (EQ. I.25) est un cas particulier de représentation radix-2r (EQ. I.30) pour r = 1.

Le signe de l'expression est donné par le bit x_{rj+r-1} et, avec $k_j \in \{0,1,2,\ldots,r-1\}$ et $m_j \in OM(2^r) \cup \{0\}$ où $OM(2^r) = \{1,3,5,\ldots,2^{r-1}-1\}$. $OM(2^r)$ est l'ensemble des chiffres positifs impairs radix-2r recodage, avec $|OM(2^r)| = 2^{r-2}$. $|Q_j| = 0$ Correspond mj = 0. Enfin, x peut être exprimée comme suit :

$$x = \sum_{j=0}^{(n/r)-1} (-1)^{x_{rj+r-1}} \times m_j \times 2^{rj+k_j} \qquad \text{(I.31)}$$

Exemple illustratif :

Le produit 10599 x X est exprimé en CSD, DBNS, et RADIX-2^r. Pour représenter ce produit en RADIX-2^r, il est d'abord nécessaire de représenter $(10599)_{10}$ en complément à deux, ce qui donne $(010100101100111)_2$. Donc dans la représentation complément à deux la longueur de bits de la constante sera N+1 (14+1=15 pour 10599).

Pour N = 14 correspond à r = 3 pour C= 10599 les équations (I.30) et (III.3) devient respectivement :

$$C = \sum_{j=0}^{4} Q_j \times 2^{3j} \text{ Et } P_{RADIX} = \sum_{j=0}^{4} (-1)^{c_{3j+2}} \times (mj \times X) \times 2^{3j+kj}$$

$$C = \sum_{j=0}^{4} (c_{3j-1} + 2^0 c_{3j} + 2^1 c_{3j+1} + 2^2 c_{3j+2}) \times 2^{rj}$$

La table de vérité équivalant à cette représentation est calculée en utilisant l'équation précèdent, les résultats de calcule des mj, kj illustré dans le tableau I.1.

Tableau I.1 La table de verité pour 10599 en RADIX-2r

Q_j				m_j	k_j
C_{3j+2}	C_{3j+1}	C_{3j}	C_{3j-1}		
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	3	0
0	1	1	0	3	0
0	1	1	1	1	2
1	0	0	0	1	2
1	0	0	1	3	0
1	0	1	0	3	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

Pour déterminer les inconnus c_{3j+2}, m_j, k_j, la table de vérité de RADIX-2^3 (voir tableau I.1) est indexée par les termes Q_j. Les triples (c_{3j+2}, m_j, k_j) correspondent à Q_0, Q_1, Q_2, Q_3, Q_4 sont respectivement (1,1,0), (1,3,0), (1,1,1), (1,3,0), et (0,3,0). Le recodage de C=10599 écrit :

$P_{RADIX} = (3\times X)\times 2^{12} - (3\times X)\times 2^{9} - (1\times X)\times 2^{7} - (3\times X)\times 2^{3} - (1\times X) = (X_0\times 2^{12}) - (X_0\times 2^{9}) - (X\times 2^{7}) - (X_0\times 2^{3}) - X$ Avec $X_0 = (X \times 2) + X$

La segmentation de la constante est illustrée dans la figure ci-dessus

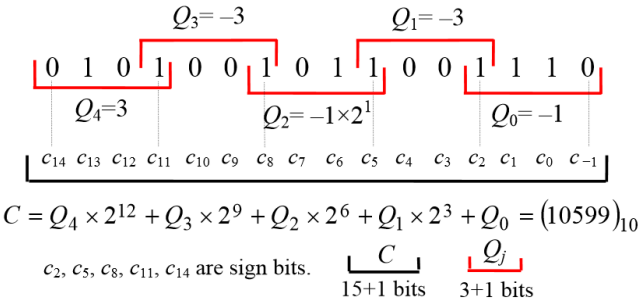


Figure I.4 partitionnement de $(10599)_{10}$ dans RADIX-2^3

I.4.4.1 Représentation canonique Radix-2r

Radix-2r est une représentation redondante puisque d'autres simplifications sont possibles

$$\cdots + 2^{r(j+1)} - 2^{rj+(r-1)} \pm \cdots = \cdots + 2^{rj+(r-1)} \pm \cdots \quad \text{(I.32)}$$

$$\cdots - 2^{r(j+1)} + 2^{rj+(r-1)} \pm \cdots = \cdots - 2^{rj+(r-1)} \pm \cdots \quad \text{(I.33)}$$

Ces deux simplifications se produisent dans l'équation I.31 dans le cas où deux termes avec signes différents de Qj et Qj + 1, exposition (mj, kj) paires de la forme (1, r-1) et (1,0), respectivement.

Si EQ. I.32 et I.33 sont les simplifications uniques qui peuvent être effectuées en radix-2^r a besoin d'une démonstration mathématique. Dans tous les cas, la forme minimale existe, mais la forme canonique (représentation unique) reste un problème de recherche ouvert.

I.4.5 Comparaison entre les systèmes de numération

Après avoir résumé les principales caractéristiques arithmétiques des systèmes de numération binaire, SD, CSD, DBNS, RNS et Radix-2^r, il est prématuré à ce stade de commenter leurs caractéristiques matérielles sans une mise en œuvre efficace d'une fonction arithmétique donnée. Pour cela, nous essayons seulement de donner une idée approximative sur l'implémentation hardware d'une valeur arbitraire, par exemple x = $(10599)_{10}$, dans différents systèmes de numération. La comparaison est basée sur le nombre de digits requis par chaque système de numération (tableau I.2). En règle générale, le nombre de digits, les réductions de ressources matérielles, et la vitesse atteinte.

Tableau I.2 nombre des chiffres requis pour chaque système pour la valeur $(10599)_{10}$

Systeme de numérisation	*Expression arithmetique*	*Nombre des chiffres*
Binary	$x = 2^{13}+2^{11}+2^{8}+2^{6}+2^{5}+2^{2}+2^{1}+2^{0}$	*8*
SD	$x = 2^{13}+2^{12}-2^{11}+2^{9}-2^{8}+2^{7}-2^{5}+2^{3}-2^{0}$	*9*
CSD	$x = 2^{13}+2^{11}+2^{9}-2^{7}-2^{5}+2^{3}-2^{0}$	*7*
DBNS	$x = (3^{2}\times 2^{8})+(3\times 2^{5})+(3^{0}\times 2^{13})+(3^{0}\times 2^{3})-(3^{0}\times 2^{0})$	*5+2=7*
RNS	x *is represented by* $(7 \mid 28 \mid 9)_{RNS(32 \mid 31 \mid 15)}$	*3+3+2=8*
Radix-2^{r}	$x = (3\times 2^{12})-(7\times 2^{8})+(3\times 2^{5})+(7\times 2^{0})$	*4+2=6*

Note que les termes 3, 3^2 , et 7dans DBNS et RADIX-2^r considéré comme des extra-chiffres

Comme nous l'avons déjà mentionné au début de ce chapitre, l'objectif d'un système de numération est l'absorption maximale de logique-redondante dans la représentation des nombres. Cela conduira à moins de digits et par conséquent à moins de ressources matérielles et plus de vitesse. Ce processus d'absorption peut être simple, par exemple dans le cas de CSD, ou très compliquées tels que dans le cas de systèmes DBNS et RNS. Selon le type de fonction arithmétique à mettre en œuvre, le processus d'absorption s'effectue soit en le logiciel ou en matériel (hardware). Par exemple, dans la multiplication par une constante (C × X), le processus de réduction est réalisé en logiciel, alors que dans la multiplication par une variable (X × Y), la réduction est implémentée en matériel. Par conséquent, l'avantage de réduire le nombre de digits peut être compensé par l'effort d'absorption, nécessitant trop de temps-de calcul (logiciel), ou trop de ressources matérielles (tableau I.3).

Tableau I.3 Principales caractéristiques des systèmes numériques

Number System	*Weighted System*	*Radix System*	*Fixed/Mixed Radix System*	*Canonical Form*	*Ease of use & Implementation*	*Frequently Used*	*Hardware Optimization*
Binary	*Yes*	*Yes*	*Fixed*	*Yes*	*A*	*A*	*E*
SD	*Yes*	*Yes*	*Fixed*	*Yes*	*B*	*A*	*D*
CSD	*Yes*	*Yes*	*Fixed*	*Yes*	*C*	*A*	*C*
DBNS	*Yes*	*No*	–	*U**	*E*	*C*	*B*
RNS	*Yes*	*Yes*	*Mixed*	*U**	*F*	*D*	*D*
Radix-2^{r}	*Yes*	*Yes*	*Fixed*	*U**	*D*	*B*	*A*

*A-F : A, le meilleur ; F, le plus mauvais ; U : Inconnu ; * : l'existence de la forme canonique doit être prouvée.*

I.5 Conclusion

L'arithmétique binaire est un vaste sujet. C'est pourquoi nous avons volontairement limité notre étude selon notre besoin pour les principales idées et les concepts essentiels impliqués dans la conception des systèmes linéaires. Nous nous sommes concentrés surtout sur l'arithmétique à virgule fixe et les principaux systèmes de nombre qui lui sont liés. Une attention particulière a été consacrée à l'addition et la multiplication qui sont les deux principales opérations de toute architecture de système linéaire.

Chapitre II :
Multiplication par une constante SCM/MCM

II.1 Introduction

Ce chapitre aborde le problème d'optimisation des ressources nécessaire pour les systèmes linéaires invariants (LTI), en se concentrant principalement au niveau de la multiplication par une constante simple/multiple (SCM/MCM) afin d'elaborer des solutions communes.

II.2 Les systèmes LTI

Un système linéaire est un modèle mathématique basé sur des opérations linéaires. Une opération linéaire se conformer par deux propriétés, à savoir l'additivité et l'homogénéité. Étant donné deux vecteurs x et y et un scalaire c, ces Propriétés sont décrites comme suit :

- ✓ Additivité : $f(x+y) = f(x) + f(y)$ (II.1)
- ✓ Homogénéité : $f(c \times x) = c \times f(x)$ (II.2)

II.2.1 Le rôle des systèmes linéaires invariants

Les Systèmes linéaires ont des rôles comme des abstractions mathématiques pour des modèles de calcul dans un grand nombre d'applications, dont principalement : la théorie du contrôle automatique, traitement du signal et télécommunications.

II.2.2 Formulation des systèmes LTI :

Si X et Y sont des entrées et des sorties d'un type vecteurs, respectivement, et C est une matrice de transformation, le système LTI peut être décrit comme suit :

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1n} \\ C_{21} & C_{22} & \dots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{m1} & C_{m2} & \dots & C_{mn} \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \quad \text{(II.3)}$$

Telque la matrice de transformation C est une matrice m × n, où le C_{ij} représente la constante, le Yi est le signal de sortie de produit de la ligne i de la matrice de transformation C et les échantillons d'entrée n de x :

$$Y_j = \sum_{j=1}^{n} C_{ij} \times X_j \quad \text{(II.4)}$$

II.3 La contrainte de la multiplication par une constante

Le problème de la mise en œuvre de la multiplication par une constante est apparu dans les années 1970. Plusieurs microprocesseurs dans ce temps (tels que les Intel 8008) n'a pas eu des multiplieurs, d'où la multiplication doit être fait avec des additions, soustractions et des décalages. Et même lorsque l'instruction de la multiplication est devenue disponible, l'exécution nécessite généralement plusieurs cycles d'horloge.
Aujourd'hui, et avec la notion de pipeline la résolution de problème de la multiplication par une constante pourrait conduire à une réduction pour le temps d'exécution mais pas comme une solution satisfiable.

SCM résout uniquement le problème de la multiplication d'une variable (X_j) par une constante *(*C_{ij}*)*. Pour être efficacement gérées, la mise en œuvre matérielle du $C_{ij} \times X_j$ doit être faite exclusivement par des additions, des soustractions et des décalages.
Afin de conduire à des avantages considérables en ce qui concerne la durée d'exécution, le domaine, débit et puissance, la complexité du calcul de SCM semble encore inconnue. C'est seulement conjecture (non preuve) d'être NP-difficile [14,8]. Mais parce que la solution-espace à explorer est énorme, il faut utiliser l'heuristique de minimisation de nombres des additions.

Exemples illustratifs :

L'exemple est $45 \times X_j$, de nombreuses solutions possibles, seulement quatre des solutions sont présentées :

$$45 \times X_j = (101101)_2 \times X_j = X_j \times 2^5 + X_j \times 2^3 + X_j \times 2^2 + X_j \quad \text{(II.5)}$$

$$45 \times X_j = (63-18) \times X_j = [(111111)_2 - (010010)_2] \times X_j = (X_j \times 2^6 - X_j) - X_j \times 2^4 - X_j \times 2 \quad \text{(II.6)}$$

$$45 \times X_j = (10\bar{1}0\bar{1}01)_2 \times X_j = X_j \times 2^6 - X_j \times 2^4 - X_j \times 2^2 + X_j \quad \text{(II.7)}$$

$$45 \times X_j = U \times 2^2 + U \text{ avec } U = X_j \times 2^3 + X_j \quad \text{(II.8)}$$

- EQ. II.5 est la méthode la plus simple pour la multiplication à simple constante. elle transforme le constant C_{ij} dans sa représentation binaire, convertit les 1 en décalage basés sur leurs positions, et additionne les valeurs décalées.
  Pour $C_{ij} = 45$, EQ. II.5 nécessite trois additions et trois opérations de décalage. Le nombre des additions à l'aide de la représentation binaire est inférieur au nombre d'instances « 1 ».
- autre méthode (EQ. II.6) utilise les décalages et les soustractions en traduisant les valeurs « 0 » en décalage tout en les soustrayant de la constante de la même longueur, composée seulement des 1. La constante de 45 requiert six bits et la constante correspondante de six bits de tous les 1 (111111) est 63. Le terme $(X_j \times 2^6 - X_j)$ représente le nombre de 63 et suit avec deux méthodes, $X_j \times 2^4 \; et \; X_j \times 2$ représentent les termes 16 et 2, respectivement (16 + 2 = 18).
- La représentation de la CSD (EQ. II.7) encode un nombre constant en utilisant le nombre minimal de chiffres différent de zéro. Par conséquent, lors de la transformation d'une multiplication constante en une séquence de décalage et additions, la représentation de la

CSD donne le nombre minimal des additions. Mais dans ce cas spécial (C_{ij} = 45), ne fournit aucun avantage sur EQ. II.5 et II.6.

- EQ. II.8 permet d'obtenir le nombre minimal d'addition pour C_{ij} = 45. La réduction des additions provient de la mise en commun du terme U = 9 j × X. Cela est bien illustré par la figure. II. 1 b, où les nœuds gris signalent une opération de décalage, et les rouges indiquent une addition. Le nombre total d'opérations est deux additions. Notez que plusieurs solutions avec 2 additions peuvent exister (Fig. II. 1 a, II. 1 b et c de II.1).

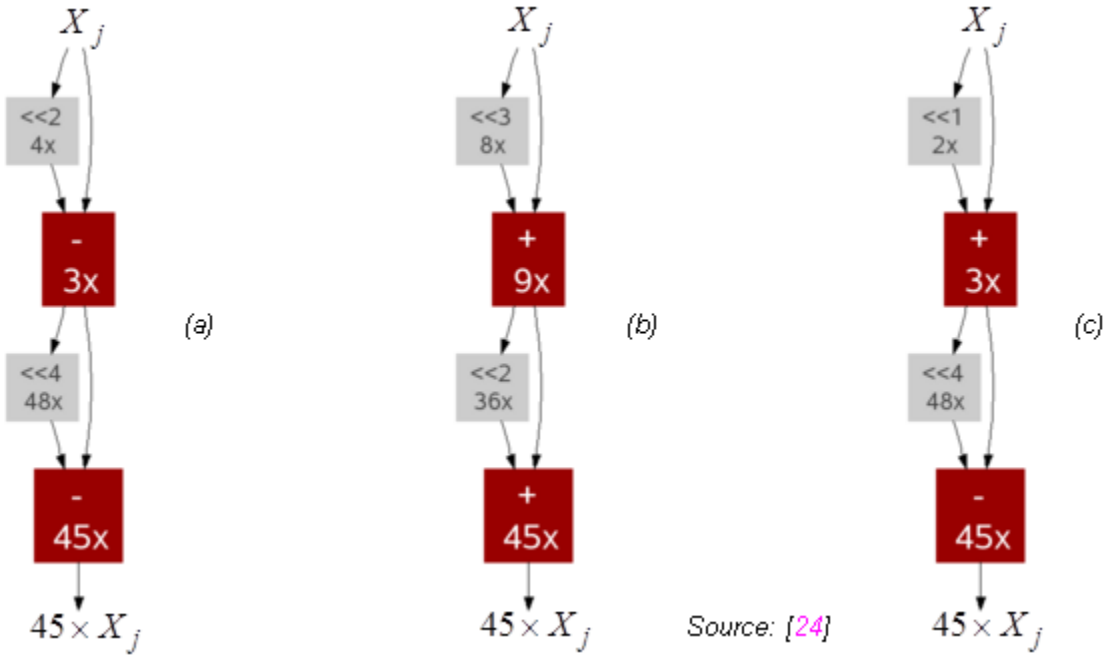


Figure II.1 le nombre minimal d'additions pour 45 j × X.

Les solutions sont données par le site web de spirale (www.spiral.net). Le signe « << a » signifie un décalage d'une position ($\times 2^a$) .

L'heuristique SCM vise à fournir des solutions optimales dans un temps de calcul raisonnable. La prévisibilité est une autre caractéristique importante de l'heuristique de la SCM. Selon la taille des bits de la constante, il permet de connaître à l'avance (avant la mise en œuvre), le nombre maximal d'additions et le couut formant le chemin critique (vitesse).

II.3.1 Multiplication de multiple-Constant (MCM)

Envisager III.3 l'équation linéaire ; MCM est une extension du SCM, où est la seule variable X_j multiplié par l'ensemble des constantes C_{1j}, C_{2j}, C_{3j}, C_{mj} (la colonne entière j). Définis comme tels, le Problème MCM est au moins aussi dur que le problème de la SCM, puisque ce dernier est un sous ensemble du problème SCM ; étant donné que le problème de la multiplication de single-constant a été conjecturé NP-difficile, le problème de la MCM est

également conjecturé NP-difficile. Une solution potentielle au problème MCM est simplement optimisée par SCM indépendamment.

Exemple illustratif

Nous effectuons les deux multiplications suivantes : $81 \times X_j$ et $23 \times X_j$. Fig. II. 2 a et II. 2 b représentent la meilleure optimisation pour chaque cas individuellement. Les nœuds indiquent plus, alors que le spectacle des bords du montant du transfert. Fig. II. 2 a et II. 2 b nécessitent deux additions, entraînant quatre additions à effectuer deux multiplications. Fig. II.2 c montre l'optimisation simultanée des deux variables. Le variables peuvent partager une multiplication commune 9 x, par conséquent, le nombre total des additions pour les deux variables est réduit d'une unité (soit un total de trois additions).

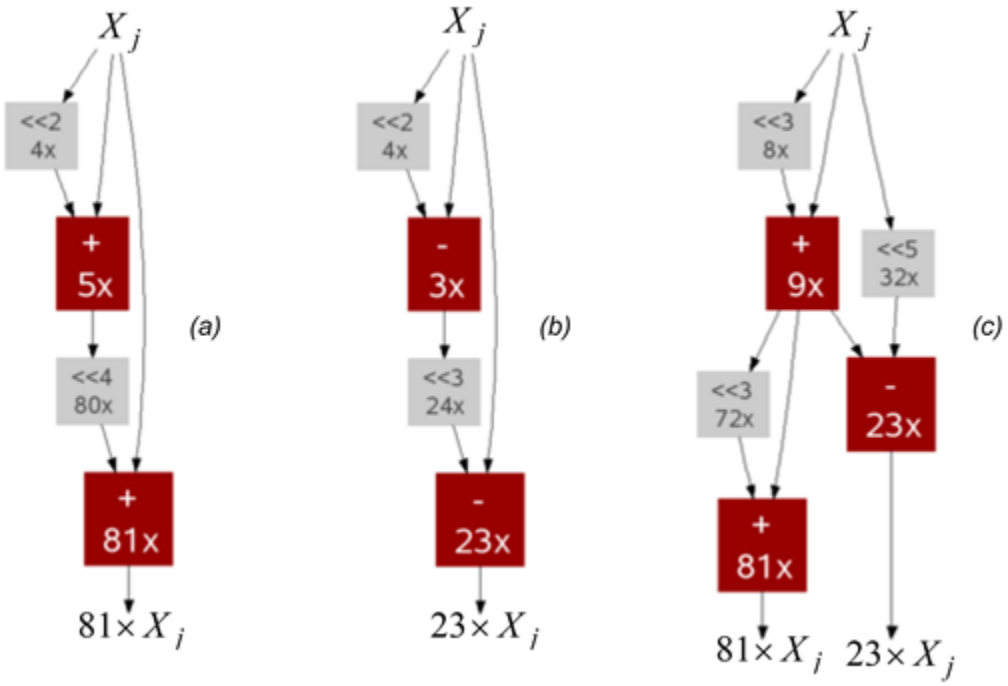


Figure II.2 Multiplication des constantes 81 et 23.

(a), (b) l'optimisation de *chaque variable indépendamment ; Cela nécessite deux additions par constante pour un total de quatre additions. (c) l'optimisation simultanée de ces deux variables. Les variables peuvent partager une addition entraînant trois additions pour la multiplication de ces deux constantes.*

II.3.2 La formalisation du problème SCM

Avant de formaliser le problème de la SCM, nous devons définir clairement :

• Le type constant : positif, négatif, impair ou même. Presque toutes les propositions des heuristiques de SCM gérer seules les constantes impair/positif/négatif peuvent être dérivé simplement à l'aide des négation et des décalages. On note que dans ce cas un traitement extra des constantes est nécessaire.

• Les opérations autorisées : addition, soustraction, décalage vers la gauche, décalage vers la droite, et opération « ou ».

Limiter le nombre d'opérations rend le problème plus difficile à résoudre. La Plupart des projets des heuristiques permettent seulement l'addition, soustraction et décalage vers la gauche. Décalage à droite permet à quelques heuristiques [14,8] puisque il a des inconvénients, tout en apportant peu de valeur : par exemple, ils ne travaille pas avec l'arithmétique modulo 2^k.

La définition formelle suivante s'applique à tous les types de constantes, c.-à-d., positif/négatif et impair/pair, permettant à des additions, soustractions et décalage à gauche seulement.

soit notre constant c, $c \in Z$. Une séquence finie des entiers signés u_0, u_1, u_2, . . . , u_q est acceptable pour c si elle satisfait les propriétés suivantes :

- Valeur initiale : $u_0 = 1$;
- Pour i>0, $u_i = s_i \times u_j \times 2^{a_i} + r_i \times u_k \times 2^{b_i}$ *avec* $j,k < i$; $s_i, r_i \in \{-1,0,1\}$ *et* $a_i, b_i \in \mathrm{N}$;
- Valeur finale : $u_q \times 2^{c_q} = c$ *avec* $c_q \in \mathrm{N}$

Le problème est de trouver un algorithme qui prend le nombre *C* et qui génère une séquence acceptable $(u_i)_{0 \le i \le q}$ aussi court que possible. q est appelé la qualité, ou la longueur. On note que pour retarder les déplacements, nous pouvons limiter bi à 0 (cela casse la symétrie, mais fait moins variable).Ainsi, un nombre arbitraire X étant donné la solution correspondante itérativement calcule $u_i \times X$ dejà calculé les valeurs $u_j \times X$ et $u_k \times X$, jusqu'à obtenir c × X. Note qu'avec cette formulation, on a calculé la valeur permise de réutiliser le tout déjà. C'est pourquoi certaines solutions générées peuvent besoin de stocker des résultats temporaires.
SCM/MCM est un problème fondamental dans le contrôle, DSP et les télécommunications

II.3.3 Les algorithmes SCM/MCM existants

Les algorithmes MCM existants peuvent être divisés en quatre grandes classes :

- ✓ algorithmes de recodage des Chiffres tels que la représentation canonique chiffres signés (CSD) [11,8], stand recodage [15,8], et Dimitrov DBNS recodage [16 ,8] ;
- ✓ algorithme d'élimination des sous-expressions communes (CSE) à l'aide des critères spéciaux effectués après un premier recodage. Des exemples typiques sont Hartley [17,8], [26,8] de Lefèvre et Boullis [17,8] ;
- ✓ algorithme de réalisation des graphes acycliques (DAG) cette catégorie comprend les Bernstein [17,8], MAG [20,8] et Hcub [21,8] ;

- ✓ Algorithmes hybrides combinant CSE et DAG tel que l'algorithme optimal récent BIGE [14,8].

II.3.4 Définition des mesures des algorithmes SCM/MCM

Les algorithmes SCM/MCM vise à produire une implémentation matérielle dans un laps de temps raisonnable en termes de la surface , vitesse et la consommation énergétique.

Étant donné une constante de N bits, les définitions principales des mesures susmentionnées sont :

Définition 2 .1 – la limite supérieure (U_{pb}) : pour chaque constant C_i à N-bit corresponds à A_i additions, pour l'implémentation de $C_i \times X$. $Upb = \max(A_i)$.

Définition de 2 .2 – profondeur d'adder (Ath) : soit D_i le nombre d'adder qui traversent l'entrée à une des sorties du circuit logique de la multiplication par une constante (le chemin le plus long). $Ath = \max(D_i)$.

Définition 2.3 − moyenne (Avg) : pour chaque constante C_i à N-bit corresponds à A_i additions, pour la mise en œuvre de $C_i \times X$, $Avg = \sum_{i=1}^{m} A_i / m$, où m est le nombre total des constantes.

En fait, bien que formellement, la profondeur de l'additionneur est définie par une estimation de chemin le plus long à travers le circuit logique (chemin critique)[9,8].

ll.3.5 Les principales Limitations des algorithmes SCM/MCM existants :

L'optimisation des systèmes linéaires est un sujet essentiel qui a été au centre des efforts continus au cours des dernières années, ce qui entraîne un nombre impressionnant d'algorithmes SCM/MCM. Parmi les principales limitations de ces algorithmes existants on trouve :

- ✓ La Prévisibilité
- ✓ La durée de stockage en mémoire
- ✓ Le risque de débordement
- ✓ La facilité d'utilisation et la mise en œuvre

II.4 Le Nouvel algorithme de recodage (RADIX-2^r)

Il repose sur l'arithmétique radix-2^r ,traite l'heuristique SCM/MCM susmentionneé , facile à utiliser et plus sécurisé en cas d'un dépassement de capacité.

Principe :

Une constante C à N bits est exprimée en radix-2^r comme suit :

$$C = \sum_{j=0}^{(n/r)-1} \left(C_{rj-1} + 2^0 C_{rj} + 2^1 C_{rj+1} + 2^2 C_{rj+2} + \ldots + 2^{r-2} C_{rj+r-2} - 2^{r-1} C_{rj+r-1} \right)$$

$$= \sum_{j=0}^{(n/r)-1} Q_j \times 2^{rj} \qquad \text{(II.9)}$$

Avec $C_{-1}=0$ et r$\in N^*$

En cas d'un nombre signé; il s'agit de la formule suivante :

$$-2^{r-1}c_{rj+r-1} \times 2^{rj} + c_{rj+r-1} \times 2^{r(j+1)} = c_{rj+r-1} \times 2^{rj+r-1}.$$

La fonctionnalité d'algorithme RADIX-2^r a une prévisibilité permet la génération des contrôleurs LTI entièrement capables de satisfaire les différentes exigences, telles que :

- Générer un contrôleur comprenant un nombre minimal d'additions (contrôleur plus compact);
- Générer un contrôleur avec un chemin critique plus court (contrôleur plus rapide) ;
- Activer un compromis entre le nombre d'additionneurs et le nombre d'étapes additionnelles, c'est-à-dire entre la surface et la vitesse
- Génère un contrôleur satisfaisant la contrainte de retard de sorte que le nombre d'additionneurs / soustracteurs soit minimisé ; etc.

l'arithmétique radix-2^r est un outil mathématique simple et puissant qui pourrait être exploré pour la resolution de probleme de la multiplication par une constante ,les chapitres qui suivent montrer l'efficacité de cet algorithme.

II.5 Conclusion

Dans ce chapitre nous avons introduit le probleme de la multiplcation par une constante simple et multiple MCM /SCM et nous avons presenté brevement la nouvelle solution algorithmique radix 2^r pour la resolution de ce probleme

Le chapitre qui suit entamine la multiplication MCM/SCM au niveau bloc d'additionneur on montre l'efficacité de cet resolution algorithmetique radix 2^r

Chapitre III :

SCM/MCM au niveau bloc d’additionneur

III.1 Introduction

Contrairement aux heuristiques qui fait la résolution des problèmes MCM/SCM, RADIX-2^r offre une réduction très importante au nombre d'additions nécessaire pour représenter un problème SCM/MCM. Cette optimisation influt directement sur l'architecture d'un système qui est basé sur la multiplication par multiple constantes concernant la surface, la vitesse, le chemin critique, ainsi que la consommation d'énergie.

Dans ce chapitre on va traiter l'explication du principe de l'algorithme RADIX-2^r pour la multiplication par constantes, puis la solution SCM/MCM développé au niveau bloc d'additionneur.

III.2 RADIX-2^r pour la multiplication par une constante SCM/MCM

L'arithmétique RADIX-2^r est une heuristique developpée pour minimiser le nombre des additions dans la multiplication par une constante. Sa complexité est sous linéaire [22]. Elle permet de générer un recodage d'une constante C afin d'avoir un produit C x X qui utilise un nombre d'additions optimisé.

La représentation RADIX-2r est développée par SAM en 1990 [8]. Dans cette technique un nombre représenté en complément à 2 de N-bits est écrit comme suit:

$$C = \sum_{j=0}^{(N+1)/r-1} (c_{rj-1} + 2^0 c_{rj} + 2^1 c_{rj+1} + 2^2 c_{rj+2} + \ldots + 2^{r-2} c_{rj+r-2} - 2^{r-1} c_{rj+r-1}) \times 2^{rj}$$

$$C = \sum_{j=0}^{(N+1)/r-1} Q_j \times 2^{rj} \qquad \text{(III.1)}$$

Ou $x_{-1} = 0$ et $r \in \mathbb{N}$

A des fins de simplicité et sans perte de généralité, on considère que r est un diviseur de N eq (III.1), la représentation complément à 2 de x est divisée en n+1/r complément a 2 segments (Q_J), représentés sur r bits car il va de 2^0 jusqu'à 2^{r-1}. Toutefois Q_j a besoin d'un bit supplémentaire (c_{rj-1}) égal au bit de poids fort du segment précèdent (Q_{j-1}), et donc chaque 2 segments adjacents ont un bit en commun. L'ensemble des chiffres DS(2^r) correspond à (III.1) tel que :

$$Qj \in DS(2^r) = \left\{-2^{r-1}, -2^{r-1}+1, \ldots, 2^{r-1}-1, 2^{r-1}\right\}$$

Le produit devient

$$C \times X = \sum_{j=0}^{(N+1)/r-1} X \times Q_j \times 2^{rj} \qquad \text{(III.2)}$$

Le signe des termes Q_j correspond au bit C_{rj+r-1} et $|Q_j| = 2^{kj} \times m_j$ avec $kj \in \{1,2,3,....,r-1\}$ et $mj \in OM\{2^r\} \cup \{0,1\}$ où $OM(2^r) = \{3,5, 7, \ldots,2^{r-1}-1\}$, dans RADIX-$2^r$ $OM(2^r)$ est l'ensemble des nombres positifs impaires avec $|OM(2^r)| = 2^{r-2}-1$

Chaque Q_j comprend r+1 bits. Le nombre total des différentes combinaisons des bits est 2^{r+1}, à partir de (III.1) seulement deux combinaisons produisent $Q_j=0$, en cas où tous les r+1 bits sont à '0' ou '1'

Finalement on peut exprimer le produit comme suit :

$$C \times X = \sum_{j=0}^{(N+1)/r-1} (-1)^{c_{rj+r-1}} \times (m_j \times X) \times 2^{rj+kj} \qquad \text{(III.3)}$$

Contrairement à la multiplication par une variable (Y x X) où tous les produits partiels (m_j x X) doivent être précalculés, dans la multiplication par une constante (C x X) seulement les sous-ensembles sont nécessaires. En réalité, le nombre des produits partiels est le nombre de différentes valeurs de m_j générées par le processus de décodage des (N+1)/r segments (les termes Q_j). Donc la génération des produits partiels (PP) consiste premièrement si $m_j \neq 0$, lors du calcul des PP m_j x X *s'il n'est pas précalculer avant*, ensuite il est soumis à un décalage à gauche matériel de r_j+k_j positions. Finalement la négation de $(-1)^{Crj+r-1}$ dépend du bit c_{rj+r-1} .

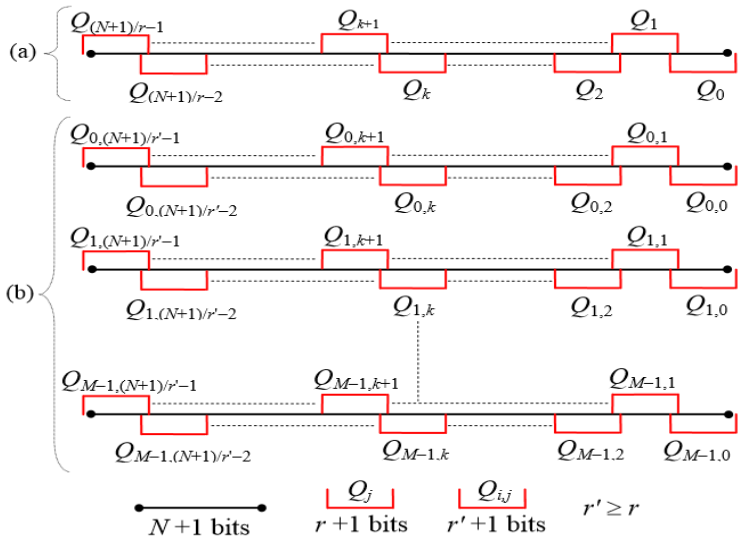


Figure III.1 Partitionnement des segments d'une constante de n bits sous RADIX-2^r[25]

III.2.1 Nombre maximal d'additions pour une constante de N bits

D'un côté il y a (N+1)/r itérations dans (III.3), chaque itération génère un produit partiel, donc le nombre maximal de PP est (N+1)/r qui requiert au max $N_{pp} = \frac{(N+1)}{r} - 1$ additions. D'un autre côté, un maximum de 2^{r-2}-1 PP non triviaux $\{3.X,5.X,\ldots,(2^{r-1}-1).X\}$ peut être invoqué lors du processus de génération des PP. Ils sont construits en utilisant la méthode binaire, du bit de poids le plus faible au le bit de poids le plus fort, et pour cela , les éléments de m_j 3,5,….2^{r-1}-1 sont construits un par l'autre, chaque fois en utilisant une seul addition entre les éléments qui sont déjà construits avec une puissance de 2 .

Le processus de construction des PP non triviaux est illustré par l'exemple suivant ou en prend RADIX-2^6 :

OM(26)= { 1, 3, 5,7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31}

={1}U{2^1 +1=3}U{2^2 +1=5,2^2 +3=7}U{2^3 +1=9, 2^3 +3=11,

2^3 +5 =13,2^3 +7 =15}U{2^4 +1=17, 2^4 +3=19, 2^4 +5 = 2^1,

2^4 +7 =2^3, 2^4 +9=2^5, 2^4 +11=2^7, 2^4 +13=2^9, 2^4 +15=31} .

Alors, les PP (m_j x X) correspondant à OM (2^6) sont calculés ultérieurement dans l'ordre suivant (6-2=4 étapes) :

$$\{3\times X\};\{5\times X,7\times X\};\{9\times X,11\times X,13\times X,15\times X\};\{17\times X,19\times X,21\times X,23\times X,25\times X,27\times X,29\times X,31\times X\}.$$

Donc il est claire que l'EQ (III.3) contient que l'opération d'addition, de soustraction, et de décalage à gauche. La figure III.2 représente tous les détails nécessaires à l'implémentation hardware

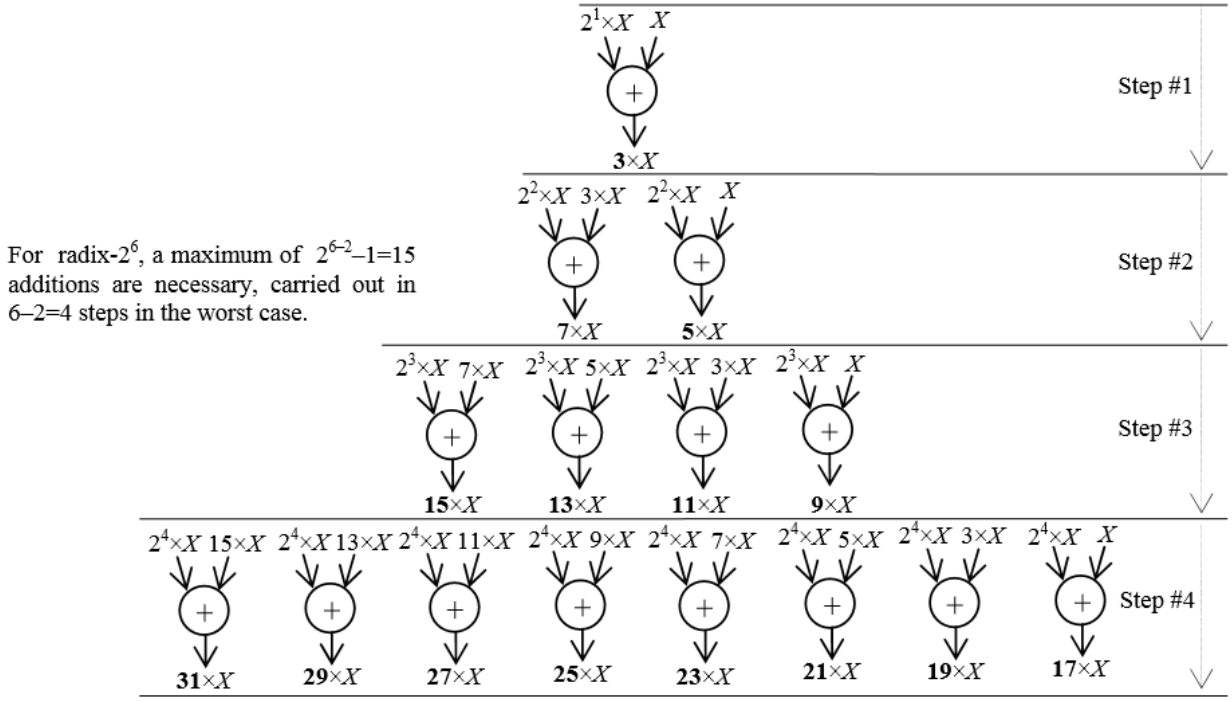


Figure III.2 Ordonnancement sequentiel de l'ensemble des produit partiels necessaire pour RADIX-2^r[22]

Par conséquent, le nombre total d'additions requis par Radix-2r est égal à :

$$Upb(r) = \mathrm{M}_{\mathrm{om}} + \mathrm{M}_{\mathrm{pp}} = \left\lceil \frac{N+1}{r} + 2^{r-2} - 2 \right\rceil \qquad \text{(III.4)}$$

Upb (r) est minimal pour $r = 2.W(\sqrt{(N+1).\log(2)})/\log(2)$ où W est la fonction de Lambert qu'est la réciproque de la fonction de variable complexe f définie par $f(w) = we^w$, c'est-à-dire que pour tous nombres complexes z et w, nous avons :

$$z = we^w \Leftrightarrow w = W(z)$$

Le minimum est obtenu pour un des deux nombres entiers les plus proches de r et les deux nombres doivent être testés. La figure III.3 représente les limites supérieures "*Upper-bounds*" des nombres d'additions pour CSD, DBNS, et RADIX-2^r.

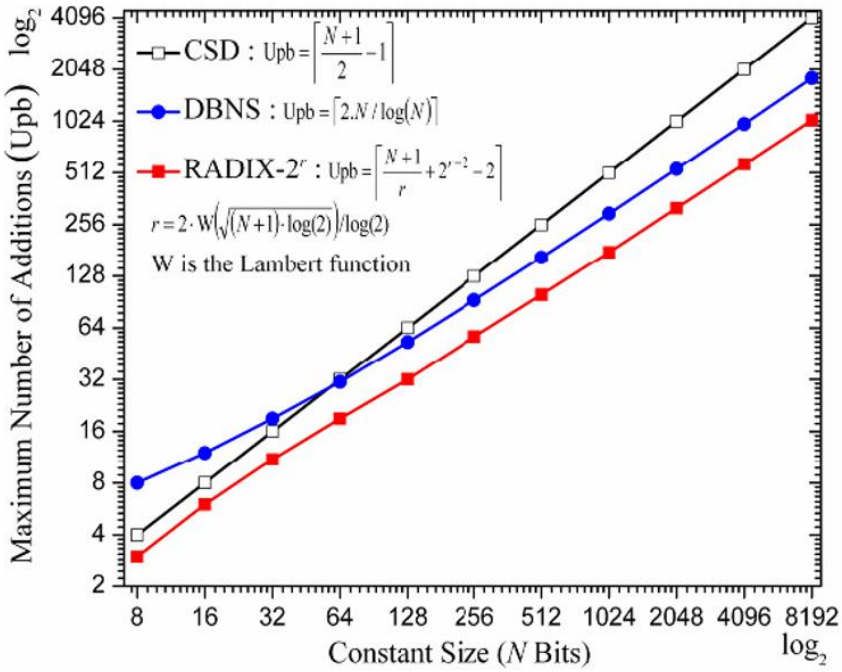


Figure III.3 Comparaison des limites supérieurs pour une constante de N bits

En ce qui concerne le nombre moyen d'additions, il a été calculé de manière exhaustive pour des valeurs de C variant de 0 à 2^N-1, pour N = 8, 16, 24 et 32. Mais pour N = 64, nous avons calculé la moyenne avec 10^5, 10^6, 10^9 et 10^{10} valeurs aléatoires de C uniformément distribuées. Alors que la différence entre les quatres résultats obtenus sont insignifiants (<10-3), la valeur Moyenne est autour de 15.7165 additions [22]. Les résultats sont présentés dans le tableau III.1.

Pour N = 64, RADIX-2r utilise 23,12% en moyennes moins d'additions que CSD. Ce gain semble progresser linéairement pour des valeurs faibles de N.

Tableau III.1 Nombre moyen d'additions de RADIX-2r et CSD[22]

Longueur de bit N de constante	CSD		RADIX-2^r		Economie (Avg,%)
	Avg	Upb	Avg	Upd	
8	1.7882	4	1.8645	3	-4.2668$^+$
16	4.4445	8	4.5127	6	-1.5344$^+$
24	7.1111	12	6.7994	9	4.3832
32	9.7777	16	8.9627	11	8.3352
64	20.4444	32	15.7165*	19	23.1256

La figure III.4 représente les valeurs moyennes des différentes heuristiques

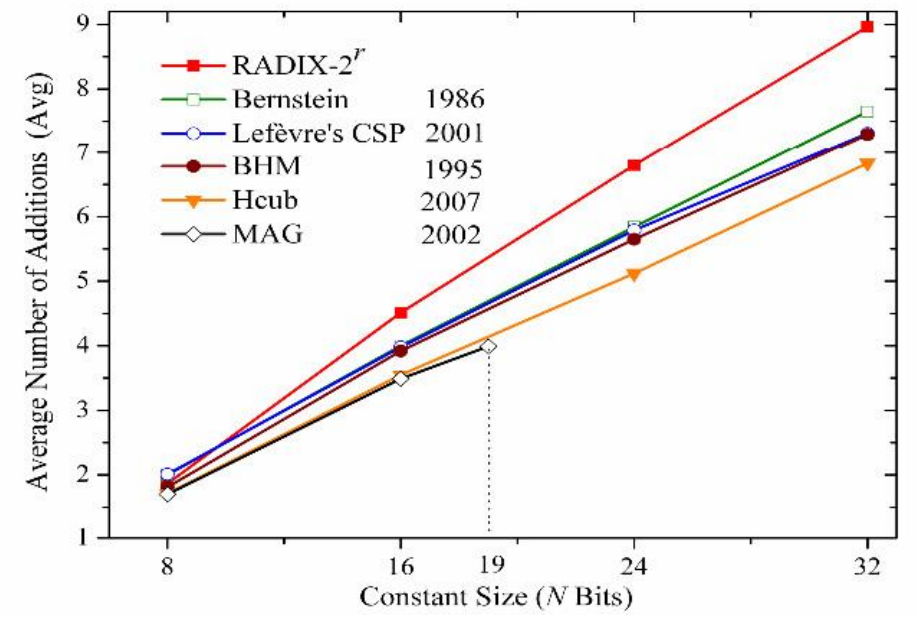


Figure III.4 Longueur de bits N d'une constante[22].

Le tableau III.2 contient les différentes relations de RADIX-2^r pour un nombre M de constantes 5MCM) a longueur de bits variable.

Tableau III.2 Equations de RADIX-2r pour des constantes non négatives de différents longueur de bit[25]

	Equations
Cout d'additionneurs	$Upb(r') = \sum_{i=0}^{M-1} \left\lceil \frac{(N+1)}{r'} \right\rceil + 2^{r'-2} - 1 - M$, avec r'=r1 ou r'=r2
Profondeur d'additionneurs	$Ath = \left\lceil \frac{\max(N_i)+1}{r'} \right\rceil + r' - 3$, avec i=0, …, M-1 et r'=r1 ou r'=r2
La moyenne	$-M + Avg_{pp} + Avg_{om} \le Avg(r') \le -M - 1 + Avg_{pp} + 2^{r'-2}$ avec

	$Avg_{pp} = (1-2^{-r'}) \times \sum_{i=0}^{M-1} \left\lceil \frac{N_i+1}{r'} \right\rceil$, $Avg_{om} = \sum_{j=0}^{\sum_{i=0}^{M-1}\left\lceil \frac{N_i+1}{r'} \right\rceil - 1} \left\{ \sum_{k=1}^{2^{r'-2}-1} P(m_{jk}) \times \left[1-P(m_{jk})\right]^j \right\}$, avec $P(m_{jk}) = \frac{\log_2 \left\lceil \frac{2^{r'-1}}{2\times k+1} \right\rceil}{2^{r'-1}}$ and r'=r1 ou r'=r2
$r1 = 2.W\left[\sqrt{\left(\sum_{i=0}^{M-1}(N_i+1)\right).\log(2)}\right]/\log(2)$	$r2 = W\left[4.\sqrt{\left(\sum_{i=0}^{M-1}(N_i+1)\right).\log(2)}\right]/\log(2)$

Contrairement aux autres algorithmes MCM, chaque constante dans RADIX 2^r est implémentée à part, indépendamment des autres, mais tous les autres constantes utilisent les même PP.

III.2.2 Les caractéristiques de RADIX-2^r

III.2.2.1 Totalement prévisible

La limite supérieure (Upb) en nombre d'additions, de profondeur de l'additionneur (Ath) et la moyenne (Avg), sont connus par des formules analytiques exactes. Cette caractéristique non seulement permet aux concepteurs d'avoir une image avant la mise en œuvre sur la surface, la vitesse, et la consommation de puissance, mais elle permet également aux outils de synthèse de satisfaire rapidement les contraintes d'utilisateur et d'effectuer les compromis nécessaires sans les rétroactions « sans fin » à la recherche de la solution appropriée. Notez qu'aucun des algorithmes de MCM existants est prévisible en Upb, Ath ou Avg [32].

III.2.2.2 Sous-linéaire

Pour un nombre de constantes non négatives donné M avec une longueur de bit N, RADIX-2r présente une complexité sous-linéaire O (M × N / r), où r est une fonction de (M, N). Cela signifie qu'il n'a aucune limitation concernant le couple (M, N). Pour les problèmes de grande complexité (M × N >>), RADIX-2r est très probablement le seul qui soit à même capable de fonctionner. Notez que les meilleures heuristiques MCM ont une complexité polynomiale ou exponentielle[32].

III.2.2.3 Solution haute vitesse et faible consommation de puissance

Profondeur d'additionneur(Ath) n'est pas seulement une mesure de chemin critique (vitesse), mais aussi un bon indicateur de la consommation d'énergie.

Les algorithmes de MCM existants ne peuvent pas rivaliser avec Radix-2r, car il permet une réduction logarithmique de Ath, alors que c'est impossible dans les autres algorithmes en raison des additions partagés [32].

III.3 Les specifications d'entrée du système par l'application RADIX-2^r

La solution RADIX-2r est une application, développée au CDTA, et est basée sur l'arithmerique de multiplication par multiple constantes de Ms A.K Oudjida. Elle prend une matrice de constantes stockées dans un fechier text comme entré puis elle genére dans un fechier le rocodage des constantes du système avec ses specifications comme sortie.

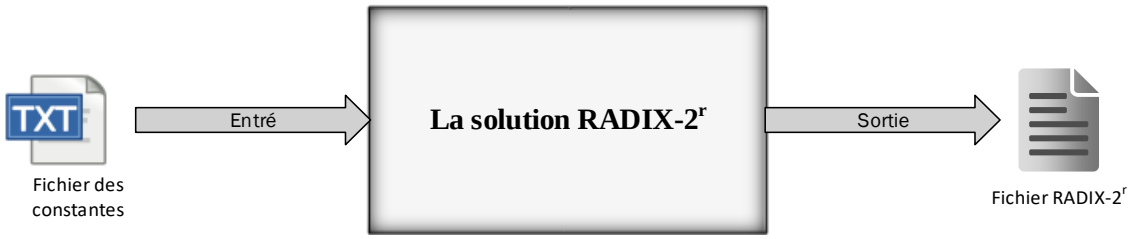


Figure III.5 Le programme RADIX-2r

Cette application donne les resultats suivants :

- Les multiples impaires utilisés aisni que le recodage des constantes selon l'arthimetique RADIX-2^r .
- La limite inferieure en nombre d'additionneurs .
- La limite superieure en cas ou les additionneurs sont connectés en serie ou en structure d'arbre.
- La longueur de bits maximale et minimale des constantes.
- Le nombre minimal de constantes (Hmin Set) utilisées pour la représentation du SCM/MCM (les odd multiples + les constantes) .
- La valeur optimale de r pour avoir une limite superieure minimale.
- La valeur de r pour avoir un nombre minimal d'additionneurs.

Le fichier genéré par le programme RADIX-2^r qui contient le recodage ainsi que les specifications d'un exemple ou on a C = 10599 est illustré par la figure III.6

```
10599

*** Radix-2^r MCM solution of the reduced set (Hmin set) including positive/odd constants only ***

    3 = +1<<1 +1<<0
    7 = +1<<3 -1<<0
10599 = +7<<0 +3<<5 -7<<8 +3<<12

*** Radix-2^r MCM Solution for the user given constants (H set) ***

10599 = +7<<0 +3<<5 -7<<8 +3<<12

With:

    7 = +1<<3 -1<<0
    3 = +1<<1 +1<<0


*** Radix-2^r solution summary ***

Lower-bound in adder-cost   = 3
Upper-bound in Radix-2^r    = 5
Adder-cost in Radix-2^r     = 5

Lower-bound in adder-depth  = 3
Upper-bound in Radix-2^r (adders connected in series)           = 4
Upper-bound in Radix-2^r (adders connected in tree structure)   = 3

Maximum constant bit-size (Nmax) in the Hmin set = 14
Minimum constant bit-size (Nmin) in the Hmin set = 14
Number of constants (M_Hmin) of the Hmin set     = 1
Optimal value of r (Radix-2^r) for upper-bound   = 3
Optimal value of r (Radix-2^r) for adder-cost    = 4

```

Figure III.6 Le recodage de 10599 et sa specification par l'algorithme RADIX-2^r

Cette application permet aussi de générer des filtres FIR d'un MCM donnée en langage verilog au niveau des blocs d'additionneurs.

III.4 representation de la solution developpée au niveau bloc d'additionneurs

III.4.1 Les langages utilisés

- **Langage C**

  Le langage C a été créé en 1972 par Denis Ritchie avec un objectif relativement limité: écrire un système d'exploitation (UNIX). Mais ses qualités opérationnelles l'ont très vite fait adopter par une large communauté de programmeurs[5].

- **BASH**

  Est l'acronyme de *Bourne Again Shell*, est un interpréteur en ligne de commande de type script. Revendiquant ainsi sa filiation forte au shell sh original. D'un autre côté, Bash a aussi intégré de nombreuses fonctionnalités provenant du shell Korn, et même de Tcsh. Il est ainsi devenu un outil très puissant, que la plupart des utilisateurs de Linux emploient comme shell de connexion, on utilise ce langage pour l'ordonnancement des processus de vérification et simulation de système SCM/MCM.

- **VHDL**

  Est un langage de description matériel utilisé pour décrire le comportement et la structure des systèmes numériques, l'acronyme VHDL signifie « VHSIC Hardware Description Language », et VHSIC lui-même signifie « Very High Speed Integrated Circuit », toutefois, VHDL est un langage de description matériel à usage général qui peut être utilisé pour décrire et simulé une grande variété de systèmes numériques [7].

III.4.2 La description logicielle

- **Système d'exploitation linux distribution Ubantu 16.04**

  Ubuntu est un système d'exploitation libre, gratuit, sécurisé et convivial, qui peut aisément remplacer ou cohabiter avec les systèmes actuels (Windows, MacOS, GNU/Linux.). À l'aide d'Ubuntu, on peut naviguer sur Internet, lire et écrire des courriels, créer des documents, des présentations et des feuilles de calculs, gérer la bibliothèque multimédia et bien plus encore [28].

- **Editeur de texte G-edit**

  Gedit est l'éditeur de texte officiel de l'environnement graphique GNOME Shell, et Unity. Ce logiciel propose une interface simple et facile d'utilisation, développée avec l'aide de la bibliothèque GTK [29].

- **Compilateur GCC**

  GCC (GNU Compiler Collection) est une suite de logiciels libres de compilation. On l'utilise dans le monde Linux dès que l'on veut transcrire du code source en langage machine, c'est le plus répandu des compilateurs. La suite gère le C et ses dérivés mais aussi le Java ou encore le Fortran [30].

- **Logiciel de simulation modelsim 12.0 et 13.1**

  ModelSim est un environnement complet de simulation et débogage pour la conception de circuits digitaux en ASIC et en FPGA. Il supporte plusieurs langages de description, dont le Verilog, le VHDL et le SystemC.

III.4.3 Principe

La figure III.7 représente un schéma descriptif de la solution proposée, où le système utilise les specifications d'entreé d'un système proposé. L'application Radix-2^r traduit les specifications d'entrée puis genére une description en language VHDL optimisé équivalente a la sortie du systeme .

Figure III.7 la solution SCM/MCM développée

III.4.4 Stratégie de translation des spécifications du système

III.4.4.1 Détection de la zone des multiples impaires

L'extraction des multiples impaires est basée sur la lecture des caractères d'après le fichier d'entré, la détection de la zone de données désiré fait par les chaines de caractère qui sont au-dessous et au-dessus de la zone, ces chaines sont prises comme références, le diagramme de la figure III.8 decrit le processus de détection des multiples impaires.

III.4.4.2 Translation des caractéres

La zone des données détectée est constituée purement par des chaines de caractères. Les opérations arithmétiques ne peuvent pas effectuer alors l'étape qui suit. Il faut donc la traduction des caractères vers des types de données bien spécifiés. Cette étape se fait par une série de tests pour assurer l'extraction et la conversion de tous les chiffres et les opérations qui créent tous les multiples impairs. Chaque ligne contient un multiple impair et chaqu'un de ces multiples et calculé selon ces coefficients m_j et ces facteurs de décalages kj.

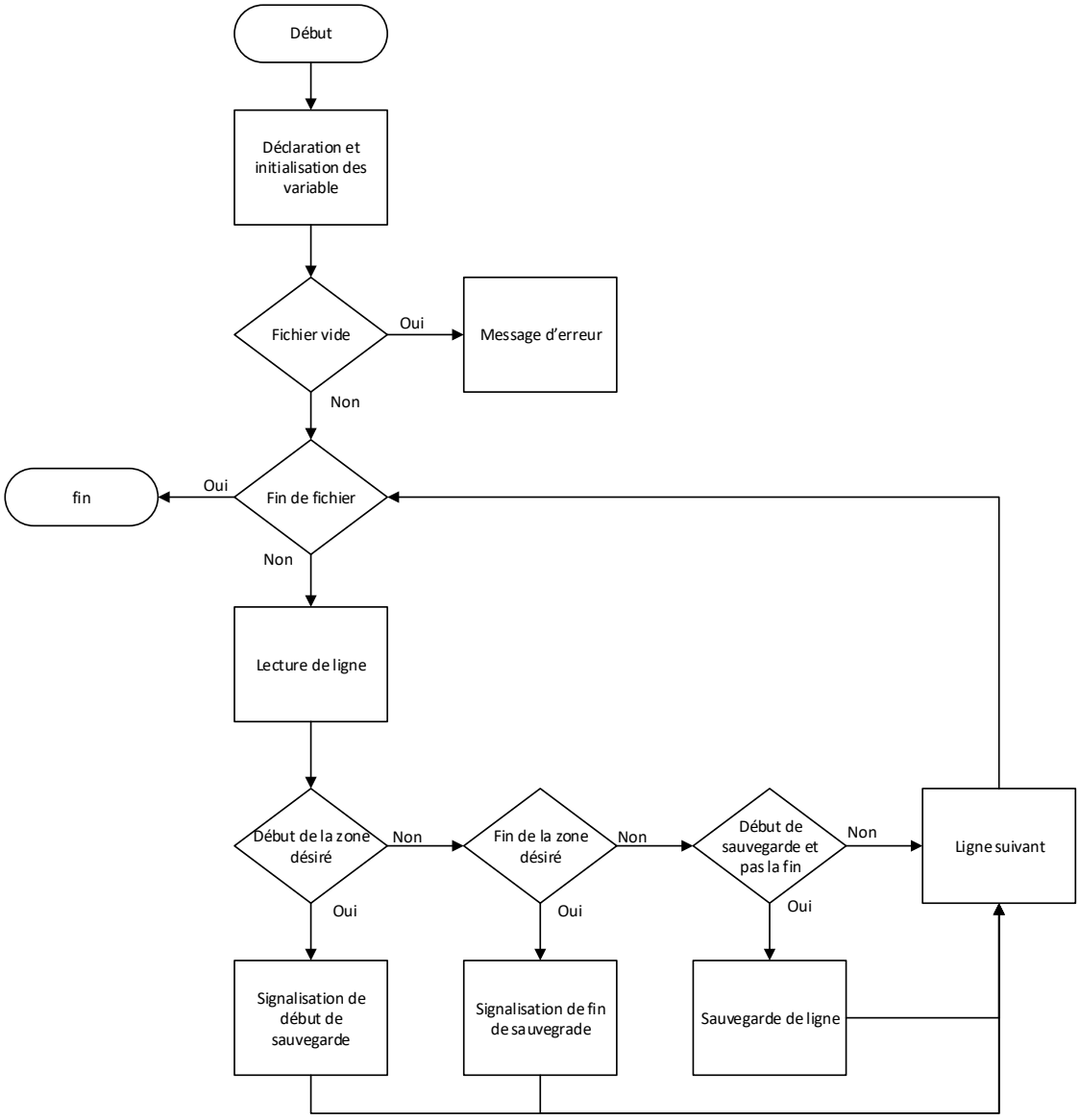


Figure III.8 L'organigramme de la détection de la zone des multiples impaires

III.4.4.3 Le stockage

Le stockage se fait par l'utilisation de listes chainées des données. Cette liste est une collection des données ordonnées, de taille arbitraire, dont la représentation en mémoire est une succession de cellules faites d'un contenu et un pointeur vers une autre cellule comme illustré dans la figure III.9.

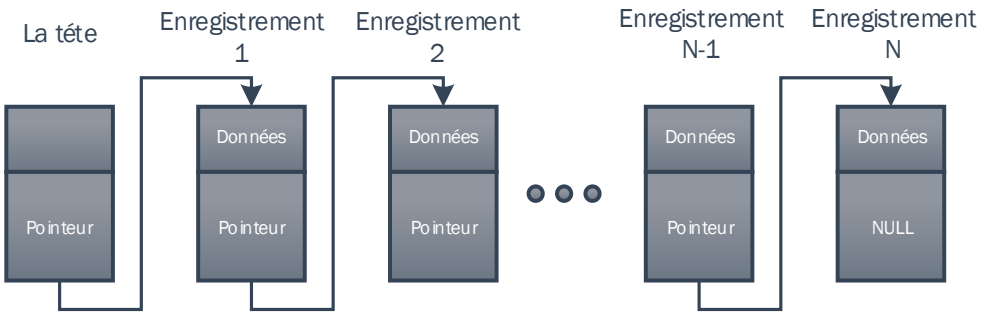


Figure III.9 Le principe de la structure chaine de liste

L'utilisation de ce type de liste permet l'allocation dynamique de mémoire selon le nombre de multiples impairs. Au contraire l'utilisation des enregistrements ou bien des matrices implique une allocation statique ou il exige une taille prédéfinie, le principe de stockage et de conversion des données est présenté sur la figure III.10.

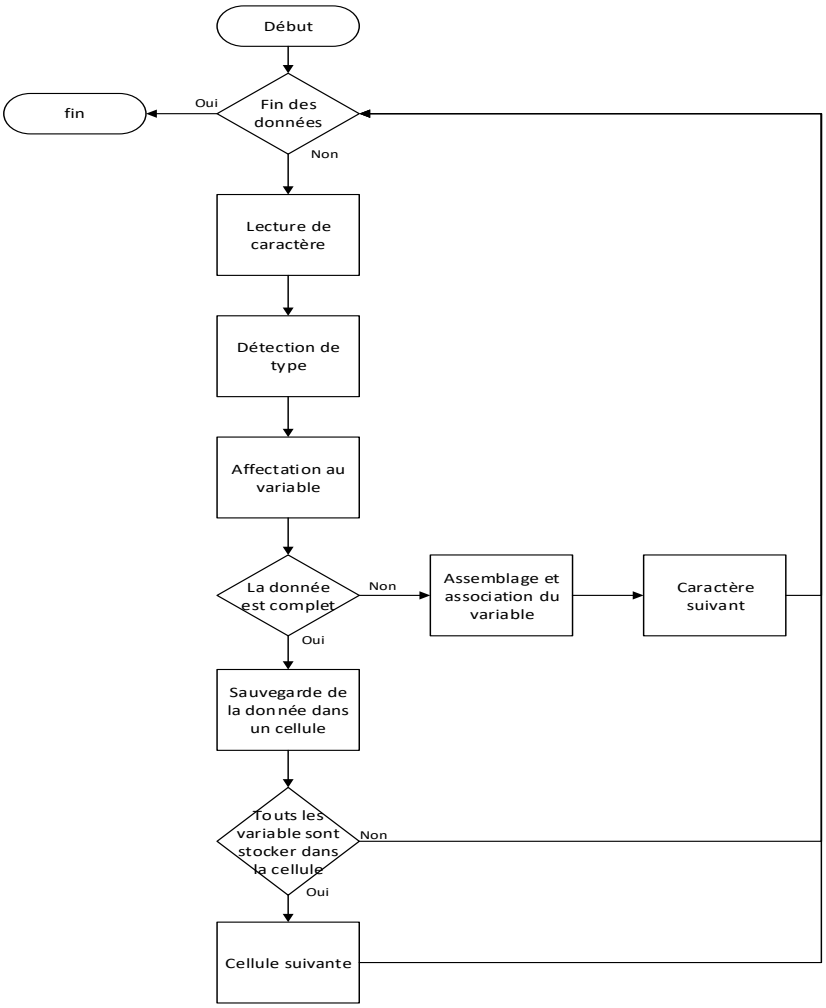


Figure III.10 L'organigramme de conversion et de stockage des coeficients

Après le stockage des m_j et k_j de chaque multiple impair ainsi que le calcule de résultat de chaque ligne, le nombre des bits de chaque bloc d'additionneur lui-même doit être calculé et stocké par la formule :

$$bit\_length = Xb + \log_2 C$$

Ce paramètre joue un rôle très important lors de la génération des signaux intermédiaires, la sortie, ainsi que la description de l'architecture de chaque multiple impair.

III.4.5 Stratégie de génération de la description VHDL

La description matérielle est basée sur la décomposition du système sous des composant élémentaire. Chaque composant représente un bloc d'additionneur d'un multiple impair. Le nombre maximal des composants (bloc d'additionneur) est $N_{IP} = 2^{r-2} - 1$. Dans cet exemple r=5 donc le nombre des composant elementaire ne dépasse pas $2^{5-2} - 1 = 7$.

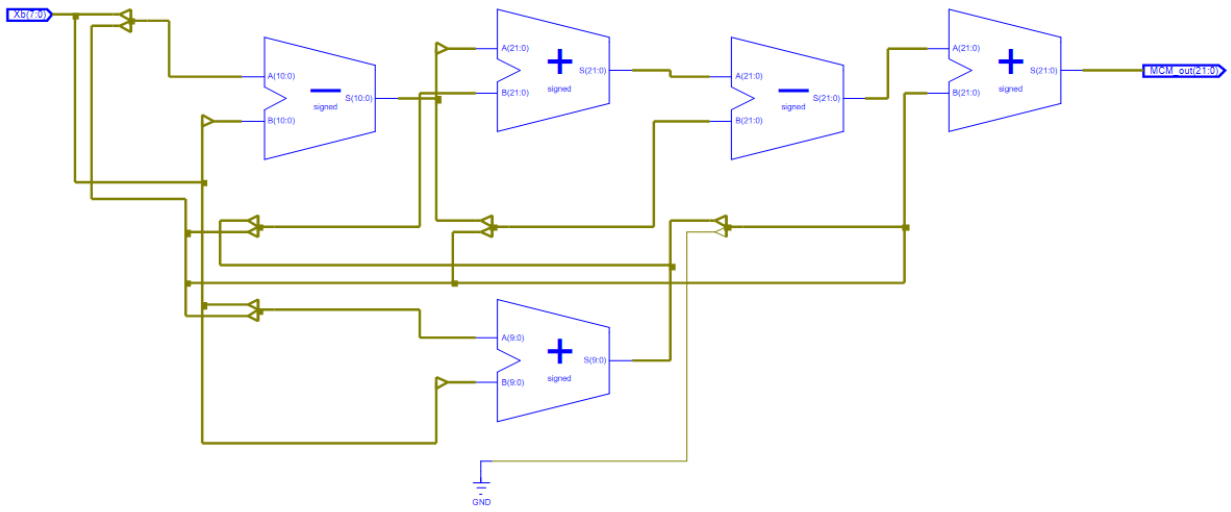


Figure III.11 architecture d'un exemple d'une solution SCM au niveau bloc d'additionneur

Le model VHDL construit est basé sur le type de description flot de donnée « *data flow* », ou on utilise des assignations directes. Tous les instructions sont exécutées en concurrence. On utilise la bibliothèque standard du langage IEEE avec le paquet standard 1164 ainsi que le paquet des nombres signé pour assurer l'addition des nombre positifs et négatifs. Le diagramme de la figure III.12 représente le processus de génération du code VHDL.

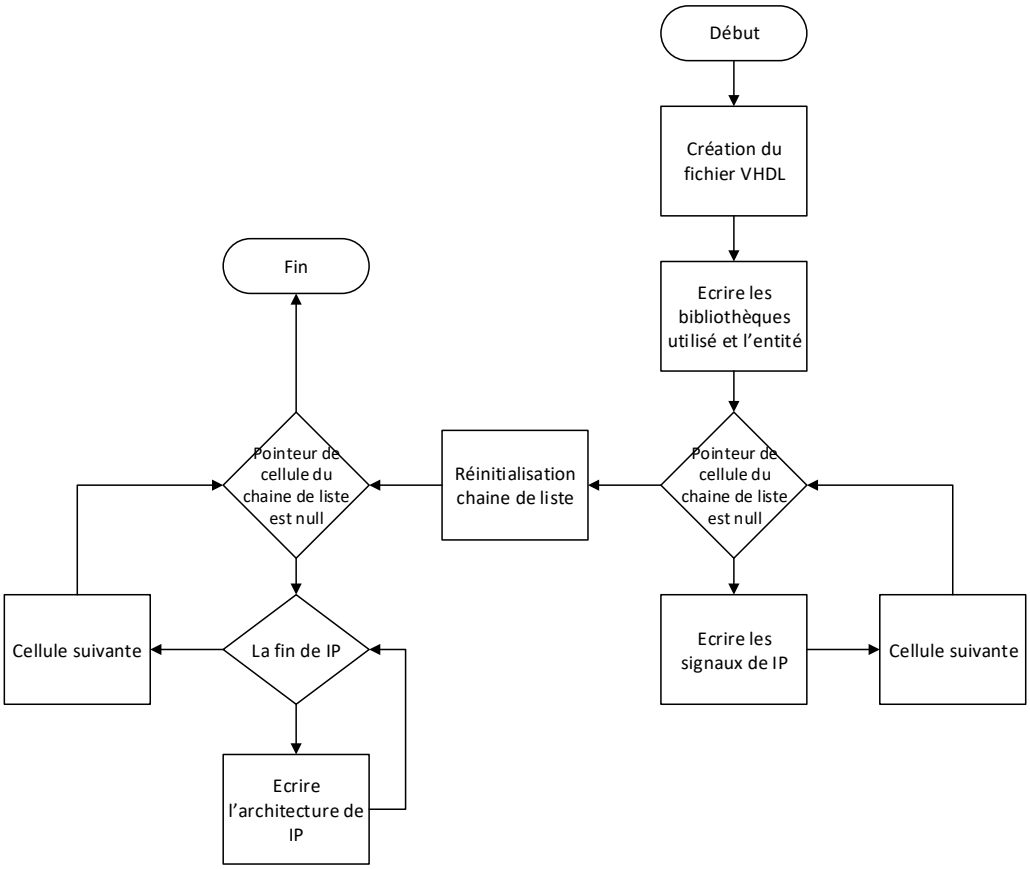


Figure III.12 L'organigramme de génération de la description VHDL

III.5 Tests et simulations

Afin de s'assurer du bon fonctionnement, on a fait des simulations de 5000 constantes différentes. On a testé toutes les possibilités de l'entrée à l'aide d'un t*est bench*. De plus Ce test fait la comparaison du résultat RTL avec la formule mathématique (C x X) comme il est montré au niveau de la figure III.16 (b).

La simulation automatique est faite sous le système d'éxploitation linux pour assurer l'ordonancement et faciliter l'organisation des programmes executés . La figure III.13présente le processus de simulation automatique sous linux.

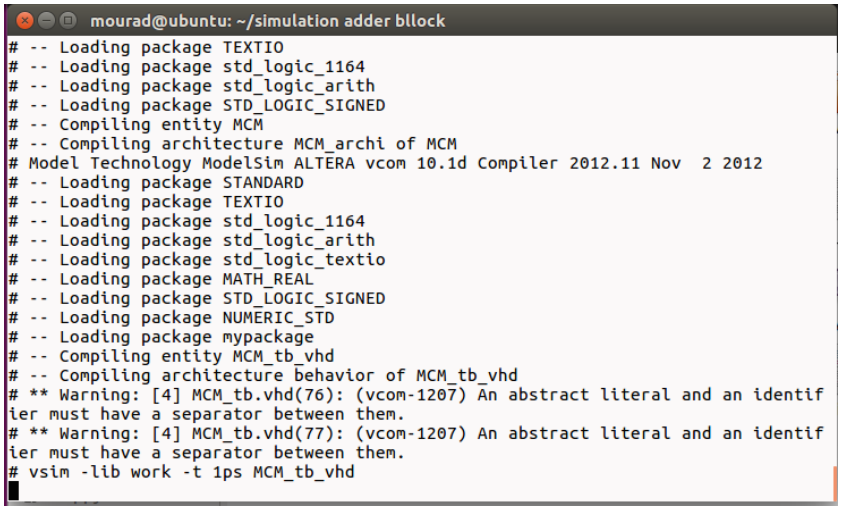


Figure III.13 Le processus de simulation de constante

Exemple illustratif :

On va faire une simulation pour la meme constante utilisé en chapitre 1 C=10599, Le recodage du constante sous le programme RADIX-2^r est :

3 = +1<<1 +1<<0

7 = +1<<3 -1<<0

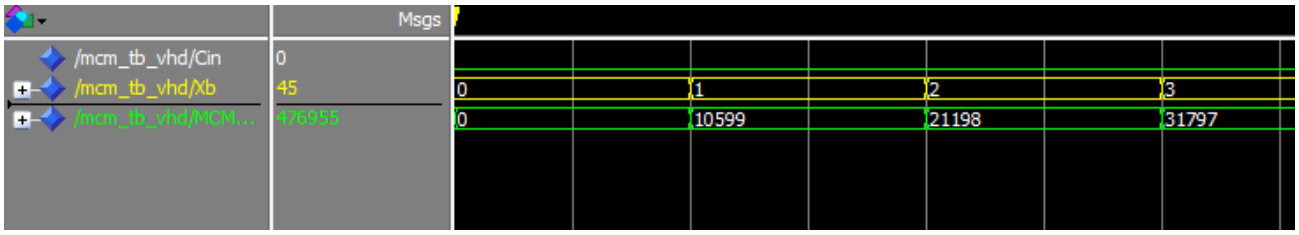
10599 = +7<<0 +3<<5 -7<<8 +3<<12

La figure III.15 représente une partie de la description VHDL équivalente à cette spécification.

```
20
21  architecture MCM_archi of MCM is
22
23  -- Intermediate signal declaration
24  signal sig_1 : std_logic_vector(7 downto 0);
25  signal sig_3 : std_logic_vector(9 downto 0);
26  signal sig_7 : std_logic_vector(10 downto 0);
27  signal sig_10599 : std_logic_vector(21 downto 0);
28  begin
29  sig_1 <= Xb ;
30  sig_3 <=  +(sig_1(7) &
31   sig_1 & '0') +(sig_1(7) &  sig_1(7) &
32   sig_1 );
33  sig_7 <=  +(
34   sig_1 & "000") -(sig_1(7) &  sig_1(7) &  sig_1(7) &
35   sig_1 );
36  sig_10599 <=  +(sig_7(10) &  sig_7(10) &  sig_7(10) &  sig_7(10) &  sig_7(10) &  sig_7(10) &  sig_7(10)
37   sig_7 ) +(sig_3(9) &  sig_3(9) &  sig_3(9) &  sig_3(9) &  sig_3(9) &  sig_3(9) &  sig_3(9) &
38   sig_3 & "00000") -(sig_7(10) &  sig_7(10) &  sig_7(10) &
39   sig_7 & "00000000") +(
40   sig_3 & "000000000000");
41  MCM_out <= Sig_10599 ;
42  end MCM_archi;
43
44  ------------------------------------end description of mcm solution---------------------------
45
```

Figure III.14 La description VHDL de 10599 x X

Pour la simulation on a utilisé le code qui est déjà donné par la figure III.15, ainsi que le testbench qui genère les stimulis d'entré ainsi que la comparaison de resultats RTL avec le model mathématique dans le logiciel de simulation modelsim. Les signaux de simulation ainsi que le rapport sont presenté sur la figure III.16 (a) et III.16 (b) respectivement.



(a)

```
math theory    -74193  =rtl    -74193
math theory    -63594  =rtl    -63594
math theory    -52995  =rtl    -52995
math theory    -42396  =rtl    -42396
math theory    -31797  =rtl    -31797
math theory    -21198  =rtl    -21198
math theory    -10599  =rtl    -10599
simulation succeed
```

(b)

Figure III.15 Le resultat de modelsim et le repport de simulation de 1059 x X

III.6 Conclusion

Dans ce chapitre nous avons présenté l'heuristique RADIX-2^r et ses caractéristiques. On a présenté encore le programme RADIX-2^r qui permet de calculer et générer la spécification particuliere pour chaque SCM/MCM puis on a presenté la solution que nous avons développée afin de construire la description VHDL équivalente pour chaque cas SCM/MCM.

Le chapitre suivant sera consacré à la présentation de l'heuristique RADIX-2^r et ses caractéristiques au niveau bit, puis la solution principale développée dans le cadre de notre projet.

Chapter IV :

SCM/MCM au niveau bit

IV.1 Introduction

Les heuristiques au niveau bit produisent de meilleurs résultats d'optimisation en termes de ressources hardwares que les heuristiques au niveau bloc d'additionneurs, mais au détriment d'un effort excessif de calcul. Cela les disqualifie de faire face au problème MCM de haute complexité. Un modèle au niveau bit précis est utilisé dans la fonction objective d'un algorithme MCM exact pour estimer la superficie de chaque opération.

Dans ce chapitre nous allons presenter RADIX-2^r au niveau bit avec son temps d'execution necessaire, la partie developpement de notre solution optimisée, avec des tests bien simulés qui sont generés d'une manière automatique au niveau bloc d'additionneur et au niveau cellule d'additionneur (1 bit).

IV.2 Le principe de RADIX-2^r SCM/MCM au niveau bit

La multiplication par une constante SCM/MCM sous RADIX-2^r est basée sur l'addition, la soustraction, et le décalage à gauche. Au niveau bloc d'additionneurs (Adder block) l'addition et la soustraction sont faite directement, bien qu'au niveau bit on sait que chaque deux Q_j adjacent (Q_j,Q_{j+1}) sont décalés un par rapport à l'autre par un décalage à gauche K_j. Le résultat de l'addition/soustraction donc soit l'addition de tous les bits de Q_j avec celle de Q_{j+1}, alors qu'on peut bénéficier un nombre très important des additionneurs complets par l'élimination des addition/soustraction qui sont pas nécessaire pour l'opération. On ce qui concerne le bit ou il y a le décalage à gauche, dans la majorité des cas on fait le déplacement de ces bits vers le poids faible du résultat plus une addition/soustraction des bits restants.

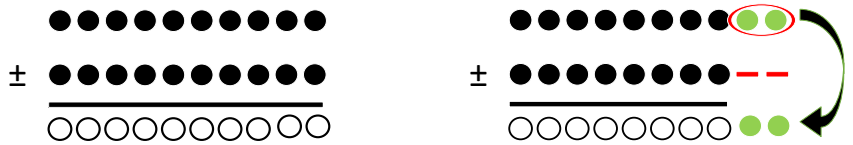


Figure IV.1 le principe d'optimisation au niveau d'additionneurs de 1 bit

IV.2.1 Extensions de signe

Nous considérons le cas général de MCM $\{C_0, C_1, C_2, ..., C_{M-1}\} \times X$, Où Ci est une constante non négative et X représenté au format complément à deux. Cependant, en arithmétique

complément à deux, le signe de tous les opérandes doit être étendu à la longueur en bits du résultat avant toute opération.

Deux produits partiels PPs $\left(...+2^{r(j+1)}\times Q_{j+1}\times X+2^{rj}\times Q_j\times X+...\right)$ exigent une extension de signe SE variant de 1 à 2r-1 bits. Quand $Q_{j+1}< Q_j$ l'extension de signe SE va être minimale et vise versa. Un exemple d'une extension de signe est donné dans la figure 4.2 ou la longueur de bits de X égal a 8, r=3, j=0[24].

$$\begin{cases} Q_j=\pm 1 \Rightarrow 2^{rj}\times Q_j\times X=2^0\times 1\times X=X \\ Q_{j+1}=\pm 2^2 \Rightarrow 2^{r(j+1)}\times Q_{j+1}\times X=2^3\times 2^2\times X=2^5\times X \end{cases}$$

$$\begin{cases} Q_j=\pm 2^2 \Rightarrow 2^{rj}\times Q_j\times X=2^0\times 2^2\times X=2^2\times X \\ Q_{j+1}=\pm 1 \Rightarrow 2^{r(j+1)}\times Q_{j+1}\times X=2^3\times 1\times X=2^3\times X \end{cases}$$

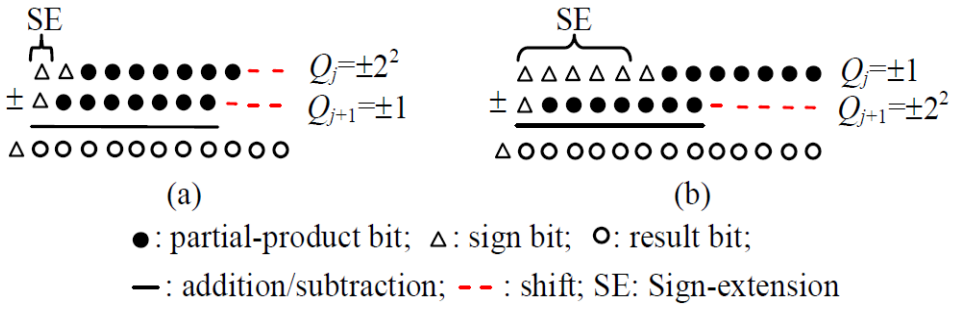


Figure IV.2 la methode d'extension de sign appliqué dans RADIX-2^r[23]

IV.2.2 Le nombre total d'additionneurs de 1 bit

Le produit C x X requis au max $X_b+\lceil \log_2 c \rceil$ bits ou X_b est la longueur de bits de la constante C. d'un côté les PPs exigent $\lceil (N+1)/r \rceil -1$ additions, chaucune consomme au plus X_b+r-1 additionneurs de type FAs. Alors le nombre max de FAs nécessaire pour la matrice des PPs est $FA_{pp}=(\lceil (N+1)/r \rceil -1)\times(X_b+r-1)$ [11]. (IV.1)

D'un autre côté, les odd multiples consomment au plus

$$FA_{om}=X_b\times(2^{r-1}-1)+(r-2)\times 2^{r-3}-1 \quad \text{(IV.2)}$$

Donc C x X a besoin au maximum de FA_{pp} + FA_{om} d'additionneur de type FA.

IV.2.2.1 En cas du MCM

$$FA_{pp} = \sum_{i=0}^{M-1} (\lceil (\mathrm{N}+1)/\mathrm{r} \rceil - 1) \times (X_b + r - 1) \quad \text{(IV.3)}$$

$$FA_{om} = X_b \times (2^{r-1} - 1) + (r-2) \times 2^{r-3} - 1 \quad \text{(IV.4)}$$

$$r = 2.W\left[\sqrt{\left(\sum_{i=0}^{M-1} (N_i + 1) \right) \times \log(2)} \right] / \log(2) \quad \text{(IV.5)}$$

IV.2.3 L'avantage de RADIX-2^r au niveau bit

RADIX-2^r est un algorithme efficace quelque soit la taille en bit (N) de la constante. C'est le seul algorithme qui offre un temps d'exécution sous-linéaire $O(M \times N / r')$ [25] par rapport à la compléxité du problème. Par contre les autres algorithmes offrent des complexités polynomiales ou carrément exponentielles. Quand la constante est très grande, le temps d'exécution sera très important. Le tableau IV.1 représente la complexité des algorithmes MCM.

Tableau IV.1 la complexité de calcul des algorithme SCM/MCM [25]

Hcub	$O(M^4 \times N^5 \times \log(M \times N) + M^3 \times N^6)$
BHM	$O(M^3 \times N^4)$
Lefèvre CSP	$O(M^3 \times N^3)$
RAG$_n$	$O(M^2 \times N^3 \times \log(M \times N))$
MAD$_c$	$\mathrm{O}(\mathrm{M} \times 2^N)$
NAIAD	$\mathrm{O}(\mathrm{M} \times 2^N)$
SIREN	$\mathrm{O}(\mathrm{M} \times 2^N)$
CSD	$\mathrm{O}(\mathrm{M} \times \mathrm{N})$
R3	$\mathrm{O}(\mathrm{M} \times \mathrm{N})$
RADIX-2^r	$O(M \times N / r')$

IV.3 Le developpement d'une solution SCM/MCM au niveau bit

Le principe de la solution au niveau bit est le même qu'au niveau bloc d'additionneurs où on utilise les spécification des constantes du système SCM/MCM générées par RADIX-2^r, puis extraire les coefficients m_j et $k_{j'}$ ainsi que les caractéristiques particulières de chaque constante pour construire une description matérielle en VHDL équivalente aux spécifications utilisées. Le nombre d'additionneurs 1 bit (FA) nécessaires ne dépasse jamais le nombre calculé par la formule. La procédure de détection de la zone des multiples impaires, la conversion des données extraites, ainsi que le stockage des coefficients m_j et k_j sont les mêmes. Le calcul du nombre d'additionneurs 1 bits du système est fait par l'utilisation des coefficients k_j et m_j et est stocké dans la liste chainée des données.

IV.3.1 Stratégie de calcul du nombre des additionneurs 1 bits

Afin d'obtenir de meilleures performances du model en termes de surface occupée, vitesse d'exécution, ainsi que la consommation d'énergie, l'utilisation d'un nombre d'additionneurs exact est nécessaire pour avoir les résultats désirés. C'est pourquoi la procédure de calcul des additionneurs 1 bit utilisé prend une importance majeure lors du développement de la solution

Le calcul du nombre d'additionneurs est basé sur le calcul de la longueur en bits de chaque couple d'opérandes successives par la relation :

$$bit\_length = m_j + \log_2 C \qquad \text{(IV.6)}$$

Puis la comparaison de ces deux longueurs et l'élimination des additionneurs qui font l'addition avec des zéros (les additionneurs ou il y a un décalage). Le seul cas ou l'utilisation d'additionneurs complet sans élimination c'est lorsque $k_j[i] > k_j[i+1]$ et le deuxième opérande est négatif ($m_j[i+1] < 0$), comme le montre la figure IV.3(d), les autre cas (a), (b), (c) permettent l'annulation des additionneurs 1 bit égaux au coefficient de décalage k_j.

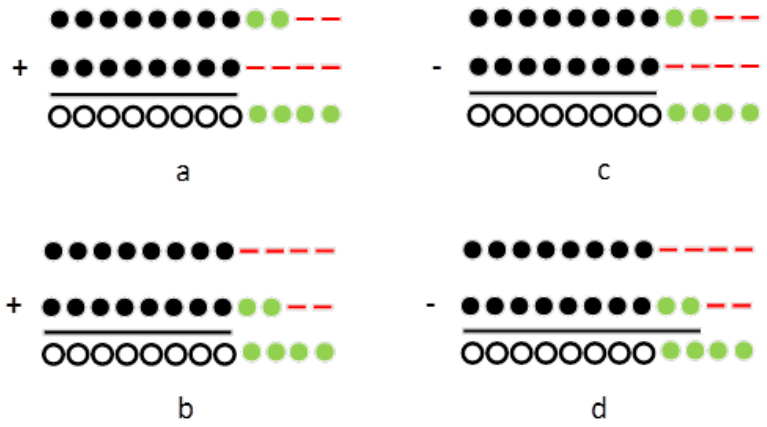


Figure IV.3 les possibilité d'une addition/soustraction en RADIX-2^r

IV.3.1.1 Cas particulier

Si $m_j[0] < 0$ on inverse les signes de tous les opérandes (on prend (-) comme facteur)

$S = -a_0 << k_{j0} + a_1 << k_{j1} - a_2 << k_{j2}$ devient $S = a_0 << k_{j0} - a_1 << k_{j1} + a_2 << k_{j2}$

puis le calcul du nombre d'additionneurs se fait comme les autres cas.

Le diagramme IV.4 représente la technique utilisée pour le calcul des additionneurs.

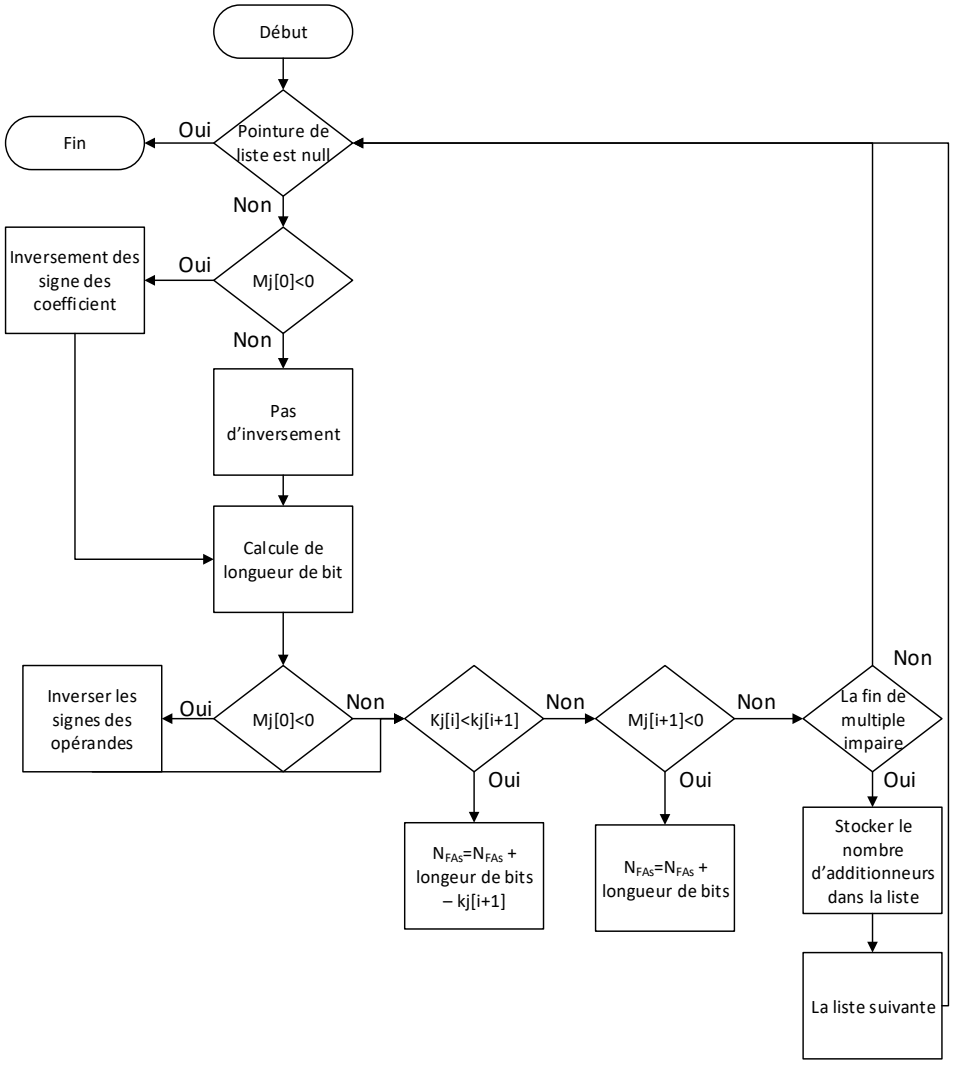


Figure IV.4 l'ogranigramme de calcul du nombre total d'additionneurs 1 bit

IV.3.2 La génération du code VHDL

La manipulation au niveau bit consiste à manipuler les données à l'aide des opérations booléennes ET (AND), OU (OR), OU exclusif (XOR) et NON (NOT), ainsi que les décalages logiques et arithmétiques. La génération du code VHDL au niveau bit passe par plusieurs étapes d'analyse et de traitement des cellules de la liste chainée des données stockées dans le but d'avoir des lignes d'instruction bien précises qui décrivent le système avec une utilisation exact des ressources. On s'appuyant sur une description structurelle où on utilise une description d'un composant (additionneur complet 1 bit) puis instancier ce composant afin d'obtenir un bloc d'additionneur optimisé avec une bon fonctionnement de chaque composant élementaire qui représente une multiplication C x X, le diagramme IV.5 représente les étapes de génération du code VHDL.

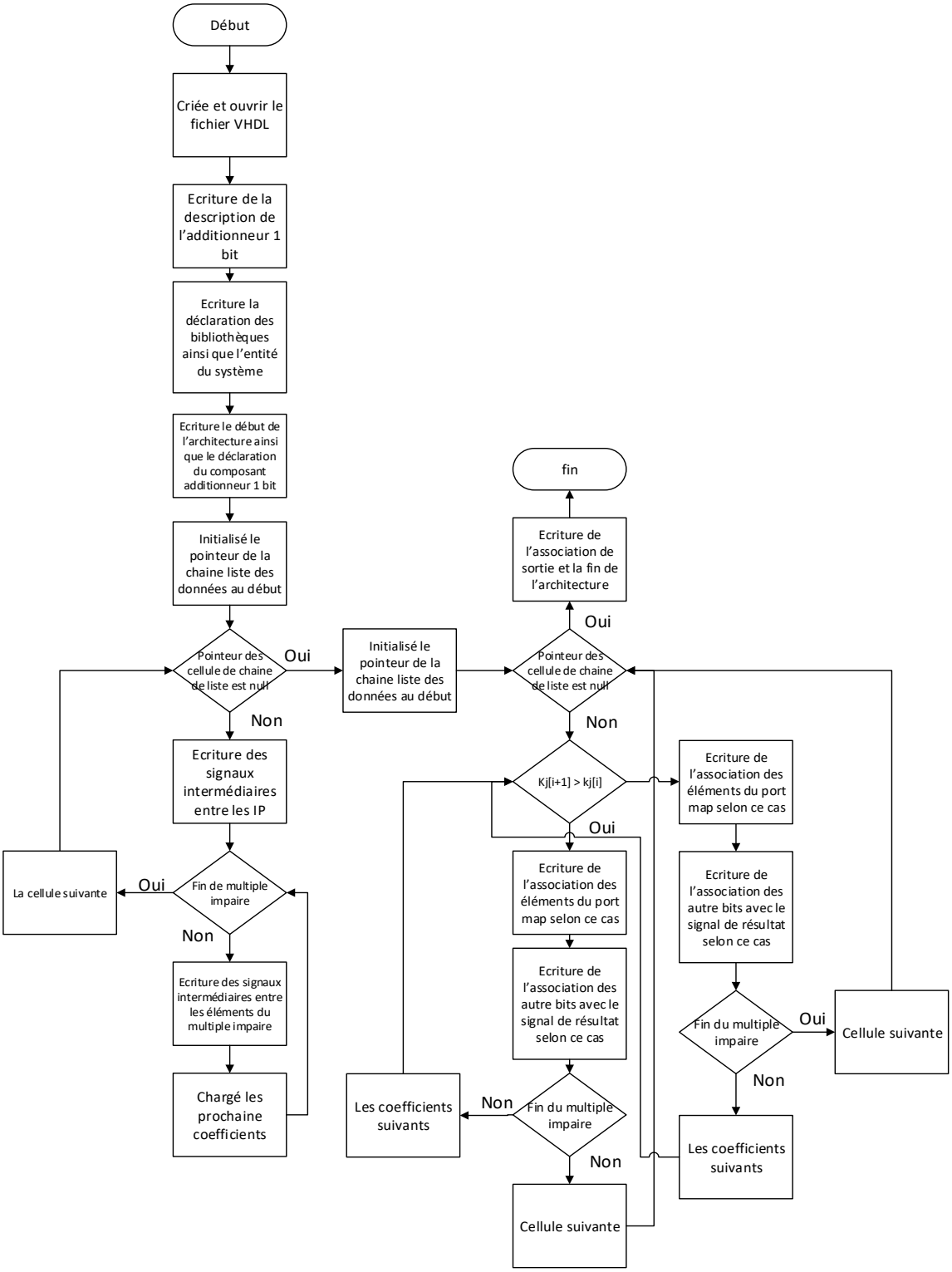


Figure IV.5 organigramme de géneration du code VHDL d'un SCM/MCM

IV.4 La description VHDL du SCM/MCM

IV.4.1 L'additionneur complet 1 bits (FA)

C'est un circuit qui fait l'addition en bit, il a 3 entrées pour le premier bit, le deuxieme bit et le retenu précedent comme un troisieme bits. La sortie est constitué de la somme et de la retenu. Le schéma ci-dessus ainsi que le tableau représente le circuit d'un additionneur complet 1 bit et sa table de vérité respectivement.

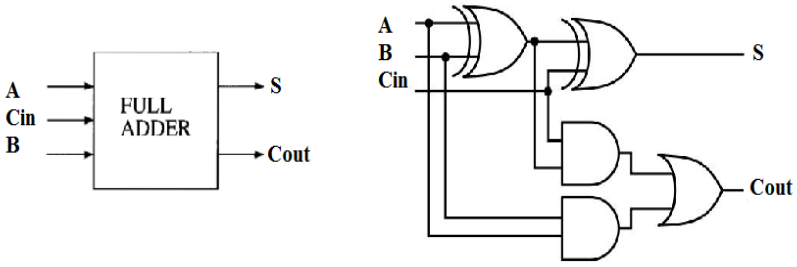


Figure IV.6 un additionneur complet 1 bit

Tableau IV.2 table de vérité d'un additionneur complet 1 bit

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

IV.4.2 L'additionneur soustracteur

L'additionneur/soustracteur est un circuit qui fait les deux opérations (l'addition, la soustraction). La soustraction des deux nombres A et B donnée par $S = A + B' + 1$, ou B' est le

complément à 1 de B, l'implémentation hardware de cette formule fait par une porte XOR ou l'ensemble des bits de B sera inversé puis à l'aide de l'entrée de la retenu on ajout un 1 pour obtenir un conversion complément à 2. La figure IV.7 représente un exemple d'un additionneur soustracteur de 4 bits. La table de vérité (Tableau IV.3) représente l'inversement des bits lorsque on fait une soustraction (Add/sus =1).

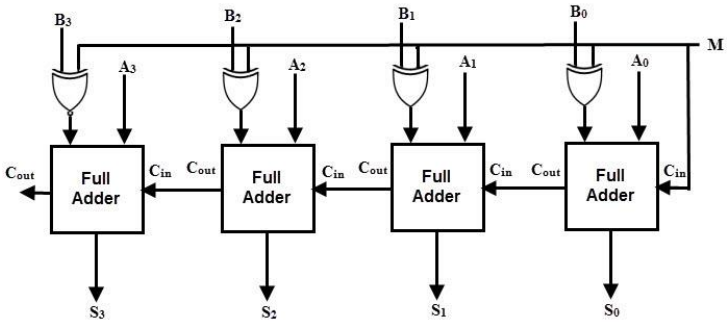


Figure IV.7 le circuit d'un additionneur soustracteur de 4 bits [31]

Tableau IV.3 table de verité d'un inversement par XOR

Add/sus	B	S
0	0	0
0	1	1
1	0	1
1	1	0

IV.4.3 L'architecture d'un SCM/MCM

Au niveau bit on élimine tous les additionneurs de 1 bit dont nous n'avons pas besoin. Ces additionneurs font la somme des bits avec des zéros, alors qu'il suffit juste de l'assignée à la sortie du signal de résultat. Cette opération ne consomme aucune ressource hardware. La figure IV.8 représente l'architecture d'un exemple d'un multiple impaire au niveau bit par l'utilisation de cette solution.

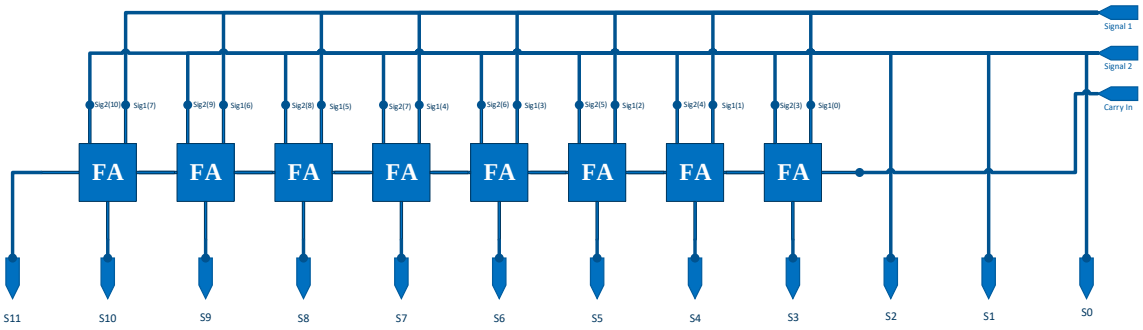


Figure IV.8 L'architecture d'un SCM au niveau bit

On associe les circuits des multiples impaires afin de construire les circuits SCM/MCM montionnées dans les specifications. La IV.9 représente un exemple d'une architecture SCM/MCM.

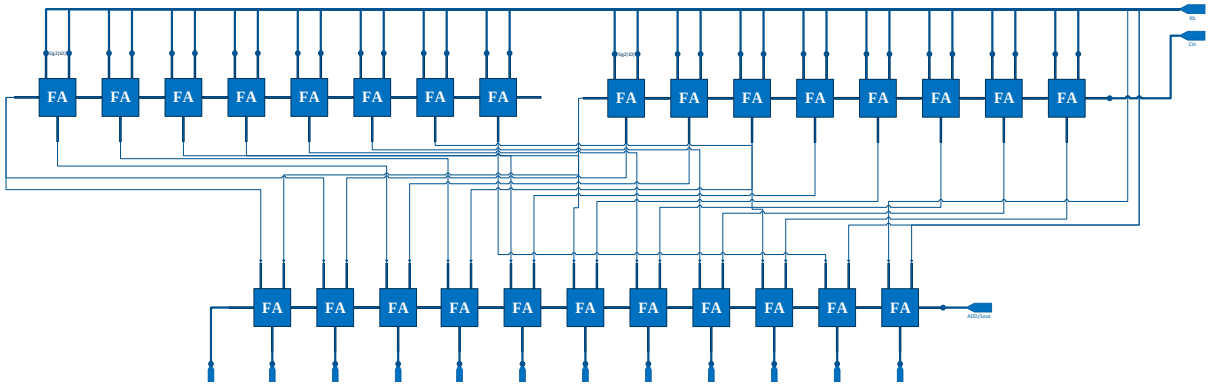


Figure IV.9 l'achitecture d'un exemple SCM/MCM

IV.5 La verification de fonctionnelle

Le but primordial est de prouver que le module va fonctionner dans l'environnement dans lequel il sera intégré. Pour cela, il faut simuler cet environnement, moyennant un *testbench*. Ce dernier permettra de mettre en situation le design, en émulant le mieux possible son environnement. Pour un *testbench* basique, seule l'instanciation du bloc et la génération des stimulis d'entrée sont nécessaires (Figure IV.10).

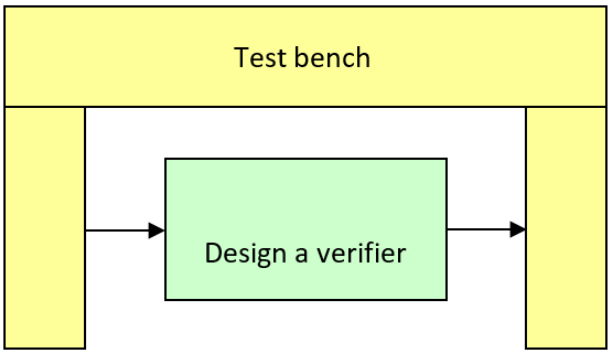


Figure IV.10 test bench

Toutefois, ceci impliquerait une vérification manuelle des résultats, donc une validation incomplète puisqu'une vérification manuelle ne pourra pas couvrir assez de combinaisons d'entrées (des constantes). Les tests effectués à ce niveau ne permettent pas de tester tous les cas possibles d'une part, et d'autre part le concepteur devait lui-même prédire les résultats mathématiques afin de les comparer avec les résultats obtenus au niveau de l'architecture .

Pour cette raison, une vérification fiable ne peut se faire qu'en faisant une comparaison automatique des résultats de sortie théorique et pratique Figure IV.11. Nous pouvons ainsi faire une validation sur un très grand nombre de combinaisons d'entrée, tout en évitant les erreurs humaines.

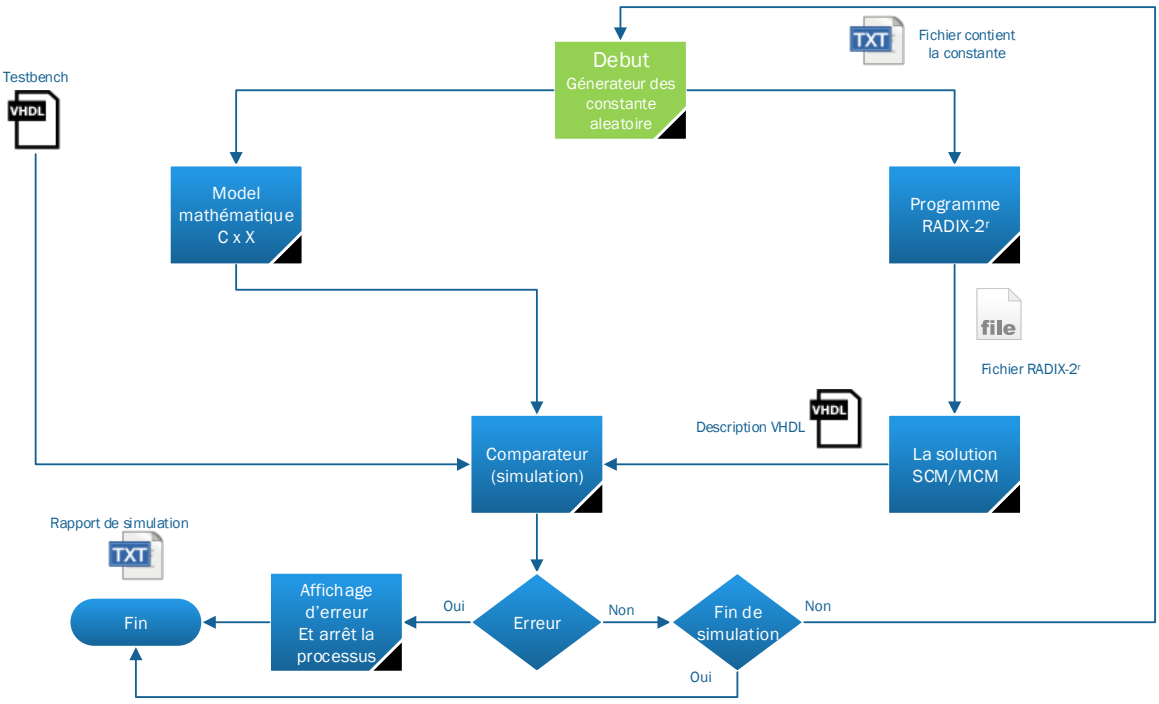


Figure IV.11 principe de la verification fonctionnelle

Le processus de vérification fonctionnelle est ordonnée par un script en shell, il commence comme le montre le diagramme, tout d'abord par un programme en C qui génère des constantes aléatoires en sortie. Cette derniere est reliée directement à l'entrée du RADIX-2^r qui génère lui-même une spécification particulière pour le système C x X désiré. Elle contient le recodage des multiples impairs qui vont être utilisés pour générer la description VHDL équivalente du système par la solution développée.

IV.5.1 Le script shell

Il est responsable de la verification fonctionnelle. il gère les programmes executés, ainsi que le lancement du simulateur Modelsim. Le script associe les entreés de chaque programme avec la valeur equivalente. Il génère encore le rapport de simulation qui contient le debogage de la verification focntionnelle. En cas d'erreur de simulation ou bien de faux resultats, dans chaque programme le script arrete le processus d'execution. Les principales partie du script sont illustrées parla figure IV.12.

```
do
    ./rand_numb # start the constant number generator program
    if [ -e $constfile ]        #verify if the constant file exist and read the constant
    then
        echo "loading constant file ..."
        if [ -s $constfile ]
        then
            echo "reading the value ..."
            read constval < $constfile
            const=$constval
            echo $const
        fi
```

L'éxecution de générateur des constantes aléatoire

```
if [ $equal = false ]
then
    ./radix m constant.txt c | cat          # execute the radix program using constant file in the input
    echo constant.result  8  | ./my_prg     # execute developed program using the result file of radix
    /home/mourad/altera/13.1/modelsim_ase/bin/vsim  -c -do ~/simulation/MCM_tb_vhd.fdo # excute modelsim in CLI mode & excute the fdo scripte
```

L'éxecution de programme RADIX-2^r , la programme développée, ainsi que le simulateur modelsim

```
    fi          # the secced condition

        if [ $sim_suc = true ]
        then
            echo "       ##################################################"
            echo "       #    simulation succeed "$counter " are tested    #"
            echo "       ##################################################"
        else
            echo "there are an error during the simulation please check you log file for more information ."
            break
            kill

        fi
```

Figure IV.12 quelques parties de scipt shell

IV.5.2 Le Test bench

Le *test bench* est un programme non synthétisable écrit en VHDL ou en Verilog. Il invoque le design puis le simule. Il peut aussi faire des fonctions plus complexes. Il contient une logique pour déterminer les stimulus corrects de design ou comparer les résultats par exemple.

IV.5.2.1 Les bibliothèques utilisées pour le test bench

Les bibliothèques

- IEEE
- STD

Les packages :

- IEEE.STD_LOGIC_1164.ALL (le package standard de VHDL)
- IEEE.STD_LOGIC_SIGNED.ALL (la package qui permet la manipulation des nombres signés
- STD.textio.all (le package qui permet la manipulation de fichiers).
- IEEE.MATH_REAL.ALL (contient les fonctions mathématiques).

Le *test bench* crée est composé de deux parties : un paquet contient une fonction qui calcule le nombre de bits de sorties pour chaque constante différente (Figure IV.13a), et la deuxième est le corps du *test bench* qui génère les différentes possibilités de l'entrée, puis il compare les résultats RTL (stimulis de sortie) avec le modèle mathématique C x X. Ce processus termine par le débogage des résultats dans un rapport, en cas d'erreur, la simulation sera arrêtée puis il donne dans le rapport un message d'erreur contenant la partie exacte où se trouve l'erreur, les figures IV.13 b et c contiennent les parties principales du *test bench*.

```
13  package body mypackage is
14  function bit_length(const : integer) return integer is
15
16  file const_file : text open read_mode is "constant.txt";
17    variable c : integer ;
18    variable the_length : integer ;
19    variable const_line : line ;
20
21
22  begin
23
24      while not (endfile(const_file)) loop
25        readline (const_file,const_line);
26        read(const_line,c);
27      end loop ;
28
29      the_length := 7 + integer (ceil(log(real (c))/log(real (2)))) ;
30      return the_length ;
31  end bit_length ;
32
33  end mypackage ;
34
```

(a)

```
51    tb : PROCESS
52    BEGIN
53
54
55      -- Wait 100 ns for global reset to finish
56      for i in 0 to 300 loop
57      wait for 1 ns;
58      Xb <= Xb + '1';
59      end loop;
60
61      -- Place stimulus here
62
63      wait; -- will wait forever
64    END PROCESS tb;
65
```

(b)

```
88      while not (endfile(math_file)) loop
89
90        wait for 1 ns ;
91        readline (math_file,math_line);
92        read(math_line,op);
93
94        math_n := op * c;
95
96        write(logfile_line,string'("math theory   "));
97        write (logfile_line,math_n);
98        write(logfile_line,string'("  =rtl   "));
99        write (logfile_line,conv_integer(MCM_out));
00        writeline(log_file,logfile_line) ;
01
02        if math_n /= conv_integer(MCM_out) then
03          write(logfile_line,string'("error found!   "));
04          write (logfile_line,op);
05          writeline(log_file,logfile_line) ;
06          wait  ;
07
08        end if ;
09      end loop ;
10        write(logfile_line,string'("the code is succesfully simulated"));
11        writeline(log_file,logfile_line) ;
12      wait ;
```

(c)

Figure IV.13 le code test bench

IV.5.3 Compilation, élaboration, et simulation

Avant que le model VHDL soit simulé, on doit premièrement faire la compilation figure IV.14, en cas d'erreur la compilation affichera un message d'erreur approprié. Si le code VHDL est conforme à tous les règles, le compilateur génère un code intermédiaire qui peut être utilisé par le simulateur ou le synthétiseur. On utilise le simulateur Modelesim en mode interface ligne de commande CLI. Cette technique offre un contrôle facile et robuste avec une rapidité d'exécution.

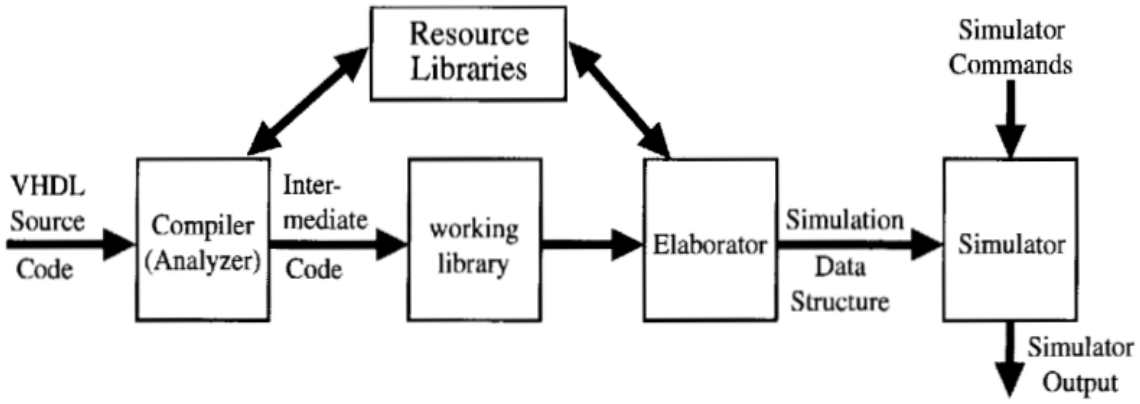


Figure IV.14 compilation, élaboration et simulation du code VHDL [6]

IV.5.4 La simulation par modelsim

Les étapes de simulation sont ordonnées dans Modelsim par un script. Les étapes de simulation sont :

- Création d'une bibliothèque pour la simulation
- Compilation des fichiers VHDL (les packages, la description de système, le *test bench*)
- Lancer la simulation par VSIM
- Afficher les signaux
- Déterminer le temps de simulation

Le script utilisé par modelsim est illustré par la figure IV.15.

```
##
vlib work
vcom -explicit  -93 "mypackage.vhd"
vcom -explicit  -93 "MCM_solution.vhd"
vcom -explicit  -93 "MCM_tb.vhd"
vsim -t 1ps   -lib work MCM_tb_vhd
view wave
add wave *
do {MCM_tb_vhd.udo}
view structure
view signals
run 1000ns
```

Figure IV.15 le script de simulation de modelsim

IV.6 Resultats et discussion

On a résolu les meme exemples de SCM/MCM au niveau bloc d'additionneurs et au niveau bit (additionneur complet 1 bit) tableau IV.4.

Tableau IV.4 le nombre des additionneurs 1 bit consomés par les deux solutions

MCM/SCM	Nombre Maximum d'additionneurs	Niveau bit	Taux de profit (%)
SCM			
10559 SCM	**60**	**50**	**16.66**
Filtres a faible ordre			
FIR_25_13_12	**463**	**391**	**15.55**
FIR4_30_14_13	**586**	**424**	**27.64**
FIR3_30_14_13	**628**	**469**	**25.31**
FIR1_40_19_12	**733**	**620**	**15.41**
FIR2_40_19_13	**775**	**664**	**14.32**
FIR7_40_19_19	**859**	**642**	**25.26**
Filtre a ordre moyen			
FIR6_60_29_14	**1069**	**841**	**21.32**
FIR8_80_36_14	**1069**	**941**	**11.97**
FIR5_80_39_15	**1384**	**1196**	**13.58**
Filtre a haute ordre			
FIR_279_140_24	**5822**	**4784**	**17.82**
FIR_418_208_22	**7156**	**6128**	**14.36**
FIR_516_256_24	**8467**	**7093**	**16.22**
FIR_631_313_23	**9502**	**7929**	**16.55**
FIR_695_345_24	**10399**	**8665**	**16.67**

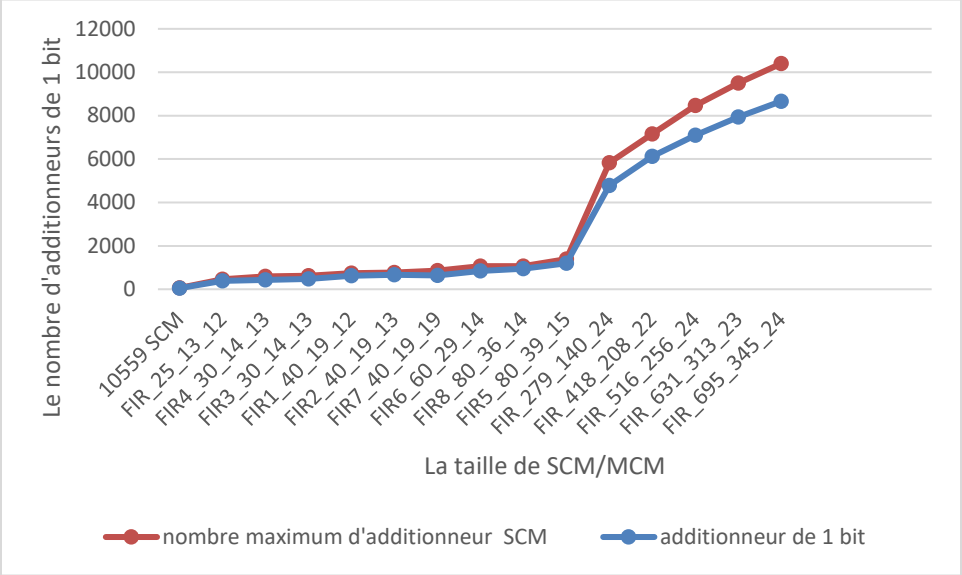


Figure IV.16 Comparaison entre le niveau bloc d'additionneurs et niveau bit

D'apres le graphe la reduction du nombre d'additionneurs est plus grande quand l'ordre des filtres ou bien le nombre des MCM sont plus grands. Le graphe de la figure IV.16 représente la variation du rapport niveau bit / niveau bloc. Il montre un rapport compris entre 0.9 et 0.7, et ça offre une reduction de 14% a 27%.

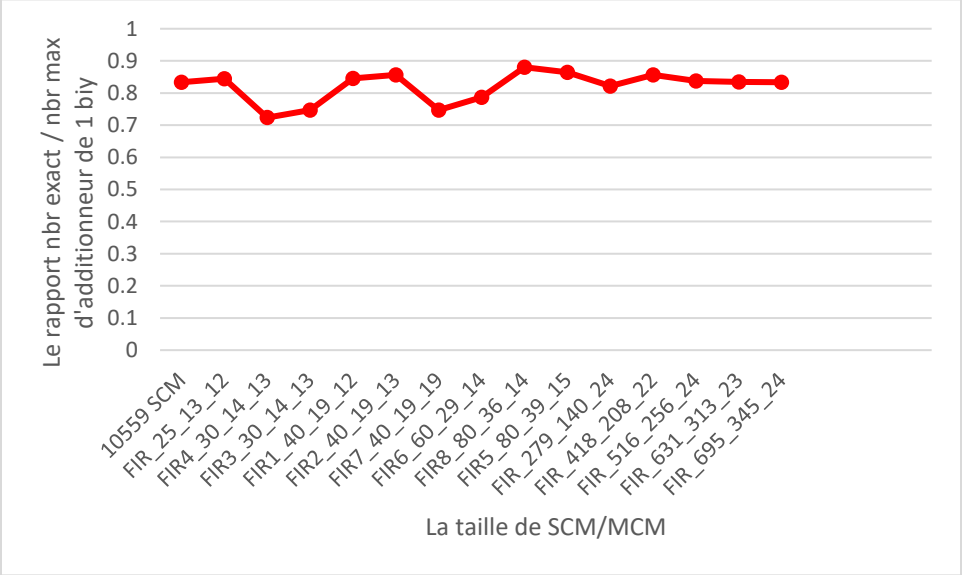


Figure IV.17 Variation d'un nombre d'additionneuurs reduits en differents cas

On vu que le niveau bit offre une optimisation très importante, en terme de nombre total d'additionneurs à l'aide d'élimination des additionneurs de 1 bit qui ne sont pas utilisés. Cette optimisation a une influence sur la surface de la puce utilisé (FPGA/ASIC), la vitesse, et la consommation d'énergie.

IV.7 Conclusion

Dans ce chapitre nous avons expliqué l'heuristique de RADIX-2^r au niveau bit, et présenté l'idée de notre solution développée, ainsi que les étapes de conception, avec des tests pour SCM/MCM au niveau bloc d'additionneurs et au niveau bit,

Nous avons constaté que la solution au niveau bit offre une optimisation très importante en termes de ressources hardware qui influence directement la surface, le temps d'exécution, ainsi que l'énergie consommée.

Le chapitre qui suit sera dédié à l'utilisation de cette solution pour construire des filtres numériques RIF optimisé.

Chapter V :

Application du SCM/MCM aux filtres RIF

V.1 Introduction

Le filtre numérique a une large gamme d'applications, et est un élément clé dans la plupart des systèmes électroniques et plus généralement dand tout dispositif de traitement de signal. La conception matérielle d'un filtre RIF est principalement dominée par des blocs de multiplieurs (SCM/MCM), qui sont implémentés généralement par un réseau d'additionneurs, soustracteurs, et de circuits de décalage (Shifters).

Dans ce chapitre on utilise la solution SCM/MCM au niveau bit développée précédemment pour générer une description matérielle pour un filtre RIF optimisée au niveau de la surface, vitesse, et consommation de puissance.

V.2 Les filtre RIF

Les filtres à réponse impulsionnel finie RIF sont des filtres à temps discret avec une durée finie. Ils sont caractérisés par une réponse uniquement basée sur un nombre fini de valeurs de signal d'entré. Par conséquent, quel que soit le filtre, sa réponse impulsionnelle sera stable et avec une durée finie, dépendant du nombre de coefficients du filtre. Les filtres RIF sont utilisés dans le traitement numérique des signaux [24], Un signal de sortie de N-tap d'un filtre RIF est calculé par:

$$y(n) = \sum_{i=0}^{N-1} h_i \times x(n-i) \qquad \text{(V.1)}$$

Où N est la longueur du filtre, h_i sont les coefficients, et $X_{(n-1)}$ est la précédant entré du filtre. La forme directe et transposée d'un filtre RIF sont illustrées par les figure 1(a) et (b), respectivement.

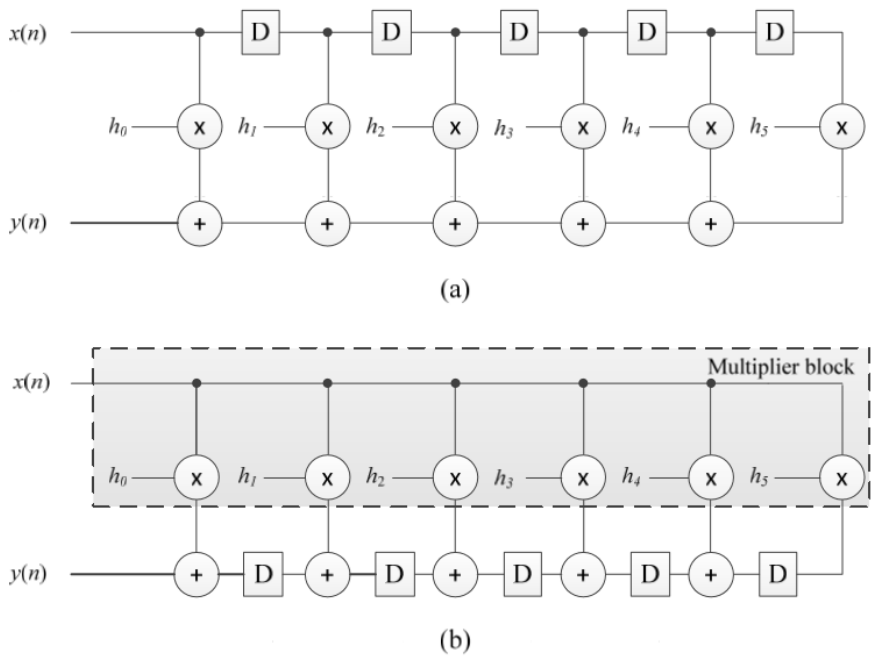


Figure V.1 implementation d'un filtre RIF (a) forme direct (b) forme transposé[24]

Selon la figure V.1 , les deux architectures ont une complexité similaire. Ils ont N multiplieurs et N-1 bloc d'additionneurs, ainsi que des registres composés principalement par des bascules D.

En général, la réalisation préférée est souvent la forme transposée à cause de sa performance et sa consommation d'énergie qui sont relativement meilleurs. La multiplication des coefficients de filtrage avec l'entrée du filtre à un impact significatif sur la complexité et les performances de la conception car un grand nombre de blocks de multiplication par une constante sont nécessaires [24].

Puisque les coefficients de filtrage sont fixés et déterminés préalablement, la réalisation matérielle de multiplieur est couteuse en terme de surface, délai, et énergie consommé. L'opération de la multiplication par une constante est implémentée généralement de maniere sans multiplication (multiplierless) par l'utilisation d'additions, de soustractions, et de circuits de décalage, sachant que l'opération de décalage peut etre implémenter juste par des fils, qui n'exigent aucune ressource matérielle (voir le chapitre précédent). Dans ce cas la complexité des filtres RIF dépend uniquement du nombre des additionneurs/soustracteurs utilisés pour implémenter la multiplication par une constante.

V.3 Les plateformes de réalisation des filtres RIF

La phase de conception d'un filtre numérique consiste à matérialiser l'algorithme de calcul, pour cela, il existe plusieurs possibilités de réalisation [10] :

- Logique câblée (portes logiques, mémoires, …) par l'utilisation des FPGA/ASIC
- Logique programmée (processeur de traitement du signal "DSP : Digital Signal Processing", microprocesseur "ordinateur").

V.3.1 FPGA/ASIC

Un FPGA *Field Programmable Gate Arrays* est un circuit de semi-conducteur basé sur une matrice de blocs élémentaires configurables (CLBs), organisés en lignes et en colonnes et connectés entre eux avec des interconnexions programmables. Les FPGA peuvent etre reprogrammables pour une application désirée ou une fonctionnalité requise avant la fabrication. Cette fonctionnalité distingue les FPGA des ASIC.

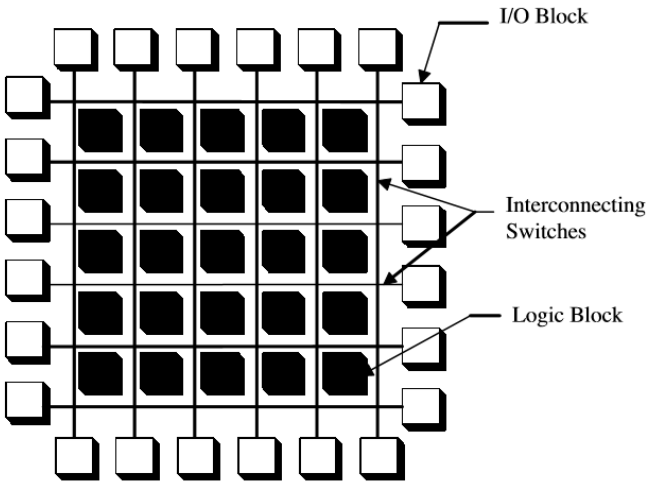


Figure V.2 la structure interne d'une FPGA[7]

ASIC *Application Specific Integrated Circuits* sont des circuit integré fabriqués sur mesure pour des tâches de conception spécifiques.

V.3.1.1 Caractéristiques des FPGA/ASIC

- Temps de réponse très rapide dans les entrées/sorties
- Des capacités de calcul parallèles massives qui peuvent être exploitées en optimisant l'architecture
- Les multiples cellules MAC peuvent être implémentées sur une seule puce.

V.3.2 DSP

Un DSP pour Digital Signal Processor en anglais est un processeur spécialisé dans le traitement numérique du signal. Son architecture est optimisée pour traiter une grande quantité de données en parallèle à chaque cycle d'horloge. Ce mode de fonctionnement est très efficace pour traiter des signaux numériques (filtrage, compression, extraction, etc.) .

V.3.2.1 Caractéristiques des DSP

- Peu couteux
- Basés sur l'architecture RISC (Reduced Instruction Set Computer)
- ont des multiplieurs accumulateurs rapides
- Architecture de pipline multi-stage

Afin de comparer ces deux types de plateforme et pour un meme traitement d'image effectué, le tabelau ci-dessous illustre les résultats de ce traitement :

Tableau V.1 le traitement d'image dans une FPGA et un DSP

	FPGA	DSP
Le nombre des cycles d'horloge	164354 cycles	10770432 cycles
Temps d'exécution (fréquence d'horloge est 50 Mhz pour FPGA, 600 Mhz pour DSP)	3.2 ms	17.9 ms
Temps d'exécution/pixel	0.19 µs	1 µs
Utilisation matériels HW	742 éléments logique	67.2 KB
Nombre de ligne de code	132	429
L'énergie consommé	79.97 Mw	540 mW

Il est claire q'avec une fréquence réduite, une vitesse importante, peu de consommation d'énergie, ainsi qu'un court code on peut implémenter des traitements de signaux surFPGA contrairement au DSP qui sont relativement plus couteux en terme d'énergie avec moins de performance. Donc l'utilisation des FPGA/ASIC garantie une haute performance avec un cout en énérgie moindre.

V.4 La structure d'un filtre RIF

La structure d'un filtre RIF se compose d'une chaine de N SCM (multiplication par constante), N-1 registres pour les SCM [24], le produit est sommé par une chaine de N-1 additionneurs. Le chemin critique le plus long est celui qui est construit par les N-1 additions avec un SCM. Le bloc MCM est la solution développée au niveau bits, la figure V.3 représente l'architecture d'un filtre RIF au forme transposé.

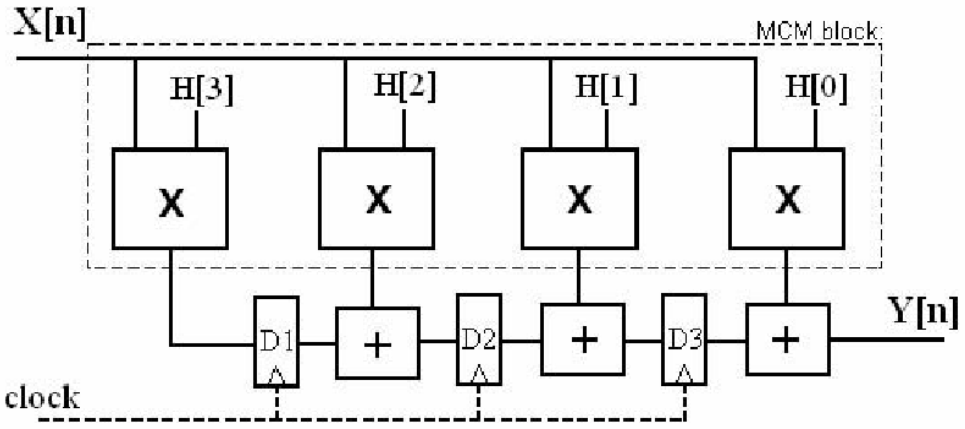


Figure V.3 L'architecture d'un filtre RIF

La génération d'une description materielle de cette architecture est basée sur une strategie qui utilise le bloc MCM et des additionneurs pour assurer la sommation des produits (*sum of product*s), Pour réduire le chemin critique, augmenter la vitesse des échantillons, et réduire la consommation d'énergie nous utilisons ce qu'on appelle le pipelining.

V.4.1 Pipelining

Pipelining est une technique de mise en œuvre dans laquel des instructions multiples chevauchent lors de l'exécution pour l'amélioration de la vitesse du chemin critique dans la plus part des systèmes de traitement de signal DSP, microprocesseurs … etc. Il peut soit augmenter la vitesse d'horloge ou réduire la consommation d'énergie électrique. Le temps d'exécution totale pour chaque instruction individuele n'est pas modifiée par le pipeline, il n'accélère pas le temps d'exécution des instructions, mais il augmente le nombre d'instructions finies par cycle.

Dans le pipeline tout opération dans le chemin critique est divisée en opérations très petites. L'opération est donc plus rapide avec l'utilisation des registres entre les niveaux, afin d'obtenir un chemin critique plus petit.

V.5 La conception du programme générateur code VHDL pour les filtres RIF

Le code VHDL optimisé pour un filtre numérique RIF est basé principalement sur le bloc MCM de notre solution développée précédemment (voire chapitre 4), avec l'addition d'élements nécessaires à la réalisation du filtres (les registres, l'entré d'horloge, l'entré du signal de reset), pour la réinitialisation des registres. Le diagramme ci-après présente les étapes principales de l'algorithme qui génère le filtre RIF optimisé.

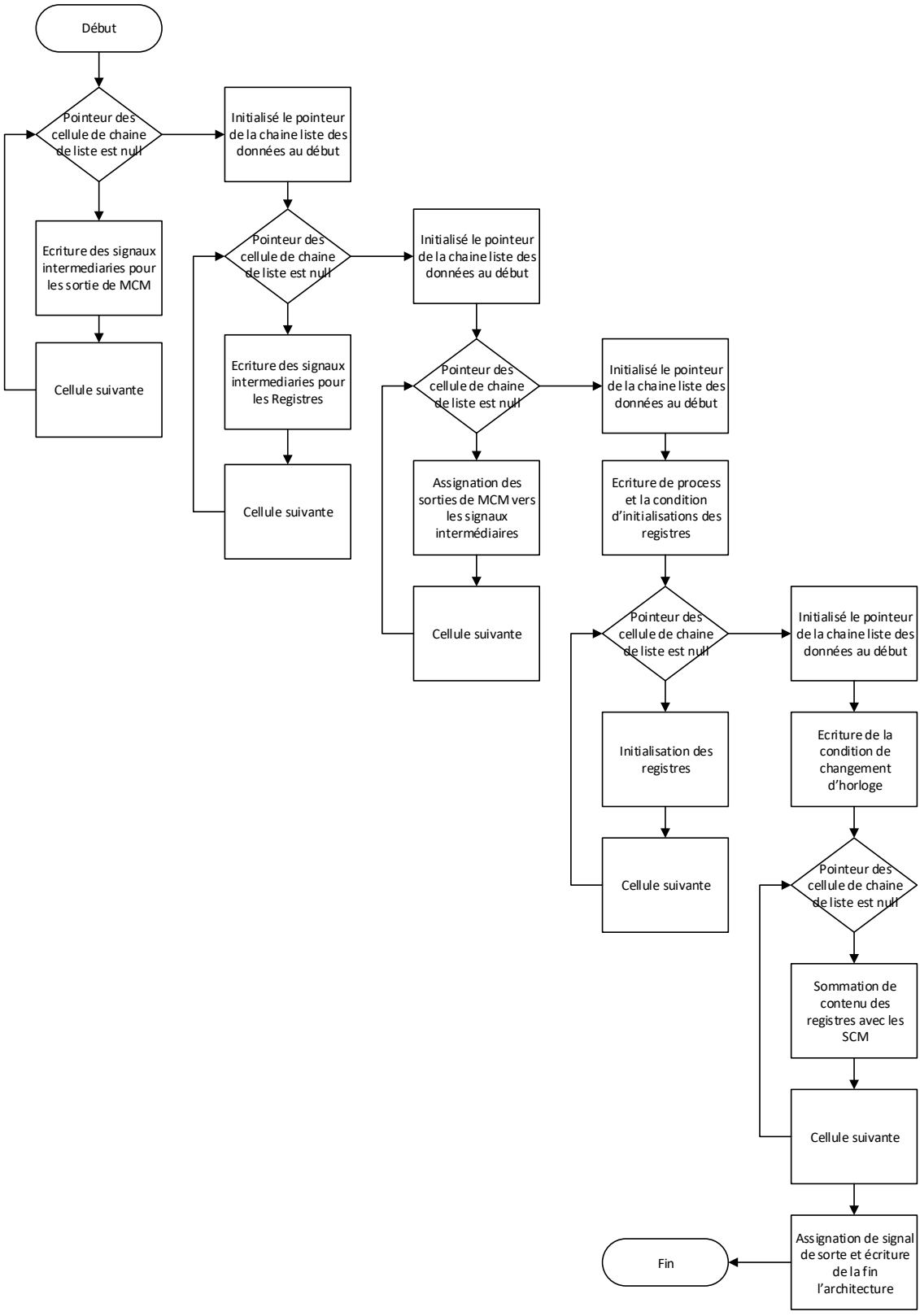


Figure V.4 l'organigramme de generation de description VHDL pour les filtres RIF

V.5.1 Résultat et discusions

Pour tester l'application, on applique notre programme sur un filtre de 24 coefficients. On extrait le recodage équivalant par le programme RADIX-2^r, puis nous l'utilisons. Les coefficients du filtres ainsi que son recodage en RADIX-2^r sont représentés dans la figure V.5.

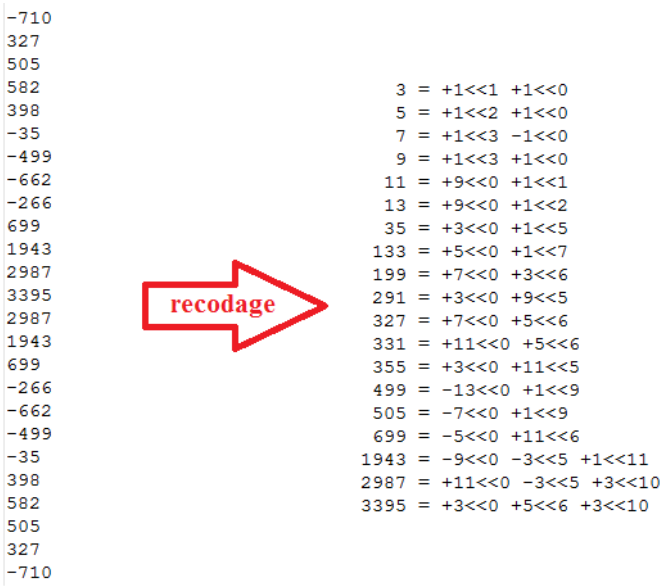


Figure V.5 les coefficients de filtres ainsi que leur recodage en RADIX-2^r

Apres l'exécution de notre programme, il génère une description VHDL optimisé équivalant à partir de recodage précèdent (voire la figure V.6).

```
SCM_15 <= sig_699 ;
SCM_14 <= sig_1943 ;
SCM_13 <= sig_2987 ;
SCM_12 <= sig_3395 ;
SCM_11 <= sig_2987 ;
SCM_10 <= sig_1943 ;
SCM_9 <= sig_699 ;
SCM_8 <= sig_133 & '0';
SCM_7 <= sig_331 & '0';
SCM_6 <= sig_499 ;
SCM_5 <= sig_35 ;
SCM_4 <= sig_199 & '0';
SCM_3 <= sig_291 & '0';
SCM_2 <= sig_505 ;
SCM_1 <= sig_327 ;
SCM_0 <= sig_355 & '0';
fltr : process (clk,reset)
begin
if reset = '1' then

     MAC_24 <= (OTHERS => '0');
     MAC_23 <= (OTHERS => '0');
     MAC_22 <= (OTHERS => '0');
     MAC_21 <= (OTHERS => '0');
     MAC_20 <= (OTHERS => '0');
     MAC_19 <= (OTHERS => '0');
     MAC_18 <= (OTHERS => '0');
     MAC_17 <= (OTHERS => '0');
     MAC_16 <= (OTHERS => '0');
     MAC_15 <= (OTHERS => '0');
     MAC_14 <= (OTHERS => '0');
     MAC_13 <= (OTHERS => '0');
```

Figure V.6 Une partie de la description VHDL de filtre

Pour la simulation de ce filtre sur modelsim, on utilise un fichiers (.do), qui génère des stimulis d'entrée du filtre, le signal de reset, ainsi que le signal d'horloge, comme montrée à la figure V.7.

```
1  force Xb 00000001 0
2  force RESET 0 0,1 10,0 20
3  force CLK 1 0,0 10 -repeat 20
4  force cin 0 0
5  run 300ns
```

Figure V.7 la generation des signaux d'entré pour la simulation

Comme il est apparaît à la figure 5.8, les résultats de simulation du filtre RIF apparaissent après un certain nombre d'oscillation d'horloge, la sortie du filtre sera égale à la somme des produit C_i x X, à cause du pipeline du système qui joue le rôle de délai. La partie inconnue dans ce signal (en rouge) est fait parce que les registres n'étaient pas encore remplis par des valeurs, donc on applique une impulsion de reset qui permet la réinitialisation des registres à 0.

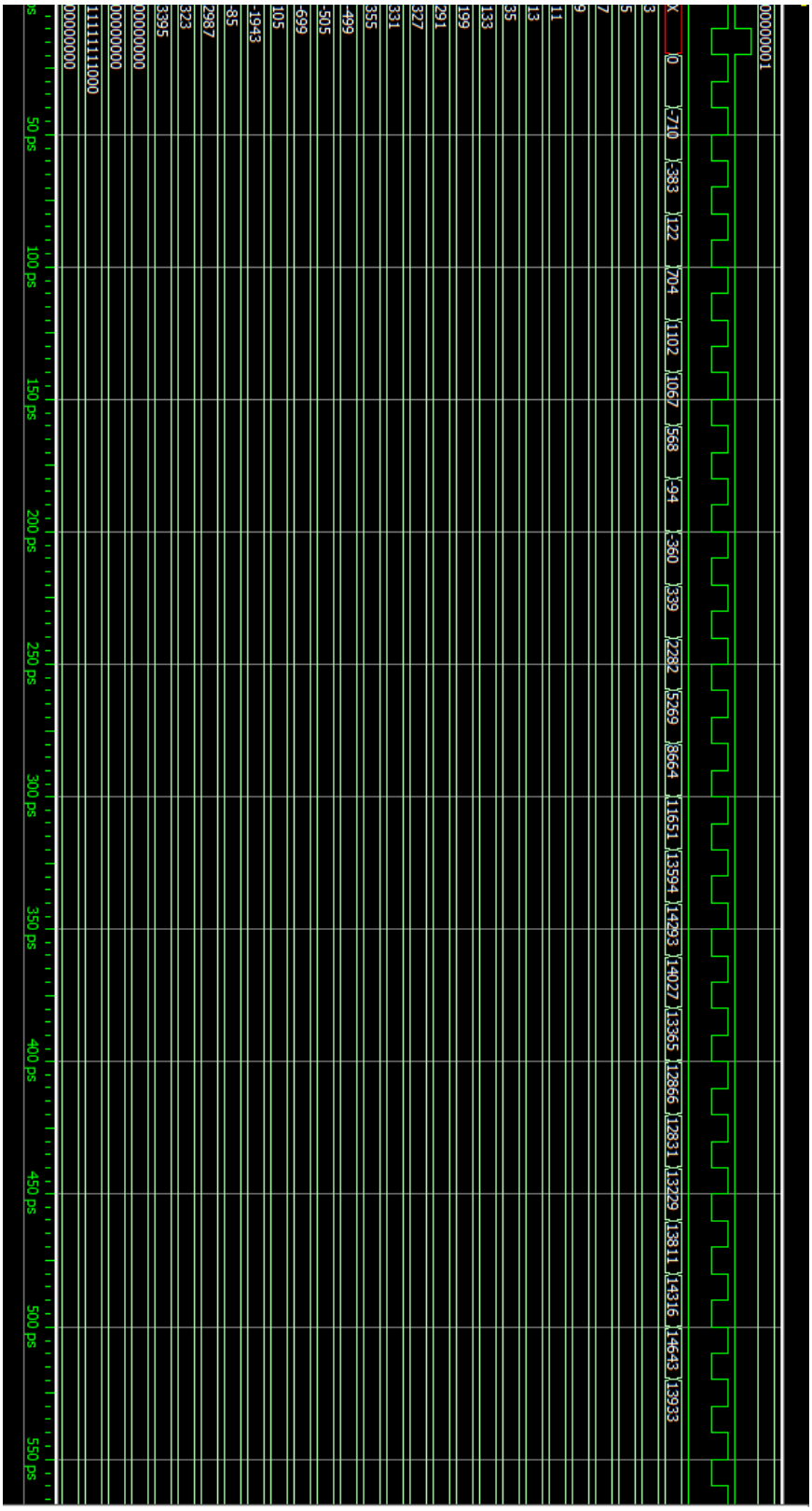


Figure V.8 les résultat de simulation de filtre

V.6 Conclusion

Dans ce chapitre on a appliqué la solution de résolution des problèmes SCM/MCM au niveau d'additionneur de 1 bit, sur les filtres à réponse impulsionnelle finie RIF. Cette application permet de générer une description VHDL très optimisée bit par bit pour tous les filtres RIF. Elle offre une optimisation importante au niveau des ressources consommées dans les blocs de multiplication, avec une architecture interne optimisée basée sur des d'additionneur de 1 bit.

Conclusion générale

L'objectif de notre projet était le développement d'une nouvelle version de RADIX-2^r au niveau "Bit" afin de solutionner plus efficacement le problème de la Multiplication par une Constante. Il s'agit de réduire davantage la consommation de puissance, d'améliorer la vitesse, et de réduire la complexité hardware (ressource logiques). Pour cela la solution développée est basée sur l'heuristique RADIX-2^r avec un algorithme de recodage multi bit .

Nous avons montré que la solution que nous avons développée au nieau bit présente des meilleurs résultats pour l'implémentation hardware de système LTI. La méthodologie adoptée est comme suit:

- Nous avons étudié et comparé les quatre systèmes de recodage : CSD, DBNS, RNS, et Radix-2^r afin de montrer l'efficacité de ce dernier en terme des performances pour la résolution du problème de la multiplication par une constante
- Nous avons développé une solution au niveau bit de l'heuristique Radix-2^r. Cette solution est prévisibile en termes de nombre maximal d'additionneurs binaires (Full Adder).
- Nous avons montré expérimentalement les performances de notre solution développée au niveau bloc d'additionneur et au niveau bit.
- Nous avons appliqué notre solution sur un filtre FIR afin de générer un filtre FIR optimisé.

Enfin, la nouvelle arithmétique Radix-2^r peut être avantageusement appliquée à d'autres domaines numériques, tels qu'en traitement numérique du signal (DSP), traitement d'image, télécommunications et le cryptage. Une idée consiste à appliquer la nouvelle heuristique SMC/MCM aux algorithmes de chiffrement RSA pour cibler des longues clés de chiffrement (plus de 4096 bits).

Livres:

[1] M.D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufman Publishers, an Imprint of Elsevier Science, ISBN: 1-55860-798-6, San Francisco, USA, © 2004.

[2] R. Kastner, A. Hosangadi, and F. Fallah, “Arithmetic Optimization Techniques for Hardware and Software Design,” Cambridge University Press, ISBN-13 978-0-521-88099-2, © 2010.

[3] J.M. Muller et al., "Handbook of Floating-Point Arithmetic," Birkhäuser Boston, a part of Springer Science+Bussiness Media, ISBN: 978-0-8176-4704-9, © 2010.

[4] B. Perhami, "Computer Arithmetic, Algorithms and Hardware Design," Oxford University Press, ISBN: 0-19-512583-5, New-York, USA, © 2000.

[5] C. Delannoy, "programmer en langage c cours et exercices corrigés", Paris : ÉDITIONS EYROLLES, 2009, 281p, ISBN 978-2212140101.

[6] K.C.Chang, "Digital system design with VHDL and synthesis : An integrated appraoch, Los Alamitos", California: IEEE Computer society, 1999, 551p, ISBN 0-7695-0023-4.

[7] j. Charles H. Roth, "Digital system design using VHDL, Boston: PWS Publishing" company, 1997, 476p, ISBN 0-534-95099-X.

Theses:

[8] A. K. Oudjida, «Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications » Université de Franche-Comté, Besançon, 2015.

[9] J. Thong, "New Algorithm for Constant Coefficient Multiplication in Custom Hardware," Master Thesis, No 4292, McMaster University, Hamilton, Ontario, Canada, October 2009.

[10] M. Kamal BOUDJELABA, "contribution à la conception des filtres bidimensionnels non récursifs en utilisant les techniques de l’intelligence artificielle : application au traitement d’images", Thèse de doctorat, Université Ferhat Abbas – Sétif – 1 –, 11 juin 2014.

Articles:

[11] A. Avizienis, “Signed-digit number representation for fast parallel arithmetic,”. IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.

[12] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, “Theory and Applications of the Double-Base Number System,” IEEE Trans. on Computers (TC), vol. 48, No. 10, pp. 1098-1106, October 1999.

[13] H. Sam, and A. Gupta, “A Generalized Multibit Recoding of Two’s Complement Binary Numbers and its Proof with Application in Multiplier Implementation,” IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[14] J. Thong and N. Nicolici, "An optimal and practical approach to single constant multiplication," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1373–1386, September 2011.

[15] Y.E. Kim et al., "Efficient Design of Modified Booth Multipliers for Predetermined Coefficients," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 27172720, Island of Kos, Greece, May 2006.

[16] V.S. Dimitrov, L. Imbert, and A. Zakaluzny, "Multiplication by a Constant is Sublinear," Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH), pp. 261-268, June 2007.

[17] R.I. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, October 1996.

[18] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," IEEE Trans. on Computers (TC), vol. 54, No. 10, pp. 1271-1282, October 2005.

[19] R.L. Bernstein, "Multiplication by Integer Constant," Software– Practice and Experience 16, 7, pp. 641-652, 1986.

[20] O. Gustafsson, A.G. Dempster, and L. Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, pp. I-73 I-76, Scottsdale Arizona, USA, May 2002.

[21] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," ACM Trans. on Algorithms (TALG), vol. 3, No. 2, article 11, pp. 1-38, May 2007.

[22] A. K. Oudjida, N. Chaillet, M. L. Berrandjia «RADIX-2r Arithmetic for Multiplication by a Constant: Further Results and Improvements » *IEEE Trans. on Circuits and Systems II: Express Brief*, vol. 62, pp. 372-376, Avril 2015.

[23] A. K. Oudjida, N. Chaillet «RADIX-2r Arithmetic for Multiplication by a Constant » *IEEE Trans. on Circuits and Systems II: Express Brief,* vol. 61, pp. 349-353, May 2014.

[24] A.Liacha, A. K. Oudjida, F. Ferguene, M. Bakiri, M. L. Berrandjia «Design of High-Speed, Low-Power, and Area-Efficient FIR Filters» *Circuits, Devices and Systems*, n° %15, DOI 10.1049/iet-cds.2017.0058 , 2017.

[25] A. K. Oudjida, A. Liacha, M. Bakiri, N. Chaillet," Multiple Constant Multiplication Algorithm for High-Speed and Low-Power Design" *IEEE Trans on Circuits and Systems II: Express Briefs*, vol. 63, n° %12, pp. 176 - 180, Fevrier 2016.

Rapport:

[26] V. Lefèvre, “Multiplication by an Integer Constant,” INRIA Research Report, No. 4192, Lyon, France, May 2001.

[27] B. Lopez, T. Hilaire and L.S. Didier, “Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation,” Proceedings of the International Conference on Design and Architecture for Signal and Image Processing (DASIP), Karlsruhe, Germany, Oct. 2012.

Page web:

[28] «The leading operating system for PCs, IoT devices, servers and the cloud _ Ubuntu,» Canonical Ltd. Ubuntu, 2017. [En ligne]. Available: https://www.ubuntu.com/. [Accès le 6 septembre 2017].

[29] C. f. d. d'Ubuntu, «gedit - Documentation Ubuntu Francophone,» Communauté francophone d'utilisateurs d'Ubuntu, 2017. [En ligne]. Available: https://doc.ubuntu-fr.org/gedit. [Accès le 6 septembre 2017].

[30] C. f. d. d'Ubuntu, «gcc - Documentation Ubuntu Francophone,» Communauté francophone d'utilisateurs d'Ubuntu, 2017. [En ligne]. Available: https://doc.ubuntu-fr.org/gcc. [Accès le 6 septembre 2017].

[31] «Binary Adder and Subtractor,» ELECTRONICS HUB, 29 juin 2015. [En ligne]. Available: http://www.electronicshub.org/binary-adder-and-subtractor/. [Accès le 7 septembre 2017].

[32] CDTA « Multiple Constant Multiplication Algorithm », © 2015. [En ligne]. Available : http://www.cdta.dz/products/mcm/. [Accés le 16 septembre 2017].