République Algérienne Démocratique Populaire

Ministère De L'Enseignements et de la Recherche Scientifique

Université Saad Dahlab Blida (USDB)

Faculté des Sciences

Mémoire De Fin D'études

Pour l'obtention du diplôme de Master en informatique

Thème :

Développement d'un programme d'optimisation de calcul de la multiplication à coefficient constant en vue une implémentation VLSI

Réalisé par :

Mr: Baid Mohammed Amin

Mr: Bharnoute Ouissem Eddine

Encadre Par:

Mme: S.Bensttiti *(USDB)*

Mr: A.k Oudjida *(CDTA)*

Soutenu le :23/09/2013

Nom Président du jury Mr : ZAIR.

Nom examinateur1 Melle : TOUBALINE

Nom examinateur2 Melle : ATTAF

Promotion 2012/2013.

Remerciement

Je voudrais tout d'abord exprimer ma profonde reconnaissance à mon dieu pour toutes donations.

Je voudrais exprimer mes remerciement en premier cite à mes encadreurs Madame ***S.Bensettiti*** et Monsieur :***A.K Oudjida*** qui ont dirige mon travail , ses conseils et ses commentaire de progresser dans mes études .

Je voudrais également exprimer mes remerciement à sincères à mes parents mes sœur et Monsieur ***M.L. Berrandjia***, qui a pas ces expérience et ces enthousiasme m'a aussi donnée beaucoup de proposition tout au long de ce projet

Tout mon travail s'est déroulé au sien de centre de développement des technologies avance et je tien a remercier tous les membres de l'équipe des administrateurs, collègue pour leur conseil leur sympathique ainsi que leurs idées.

Et en fin, comme je voudrai remercier mes amis ***Daoudi Omar*** , ***Chadouli Mohamed*** et ***Belaidi Amirouche*** qui m'on aide de réussir a mon travail .

Mohamed .Amine. Bensaid

Dédicace

Tous les lettres ne sauraient trouver les mots qu'il faut .....

Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance

Aussi c'est tout simplement que : je dédie cette thèse a .....

A mon très cher père Monsieur Bensaid Mohamed et ma tendre mère Larachiche Malika

Merci pour tes sacrifies le long des ces années

Merci pour ta présence rassurante merci pour tout l'amoure que vous procurez a notre petite famille.

A mes sœur Wafia , Samira, Hayet

Vous été les sœurs idéales pour moi, vous avez énormément de qualité que je ne pourrais pas tout les citer

A mes ange Rayen , Asma, Anes et Hadjer

Vos présence dans ma vie est pour moi un encouragement et un soutien vos joie et vos gaieté me comblent a bonheur

A mes cher amis : Chadouli Mohamed, Daoudi Omar et Tertiri Sif Eddine

Je vous dédie cette thèse en témoignage de ma grande affection et en souvenir des agréables moments passe ensemble

Mohamed .Amine. Bensaid

Notation

ASIC : Application Spécific Integer Circuit (Application spécifier des circuit intégré)

CAN : Convertisseur Analogique Numérique.

CDTA : Centre de Développement des Technologie Avancées.

CI : Circuit Intégré.

CSD : Canonical Signed Digit(canonique signé chiffres).

CSE : Commun Sub Expression .(élimination des sous expression commune).

DAG : Directed Asyclic Graphe (Graphe Oriente Acyclique).

DB NS : Double Base Numbering System (double système en base de numération).

DFT : Discret Fourier Transformation(transformation discret de fourier).

DSP : Digital Signed Processing (traitement du signal numérique).

FPGA : Field Programmable Gate Arrays.

LIT : Linéaire Invariante Time (temps linéaire invariant).

LSI : Large Size Intégration (integration a grand echelle).

MSI : Meduim Size Intégration (integration a moyen echelle).

SSI : Single Size Intégration (integration a faible echelle).

VLSI : Verry Large Size Intégration (integration a tres grand echelle).

Résume :

Le traitement de signal numérique est basé sur le recodage et la multiplication à une constante. Plusieurs algorithmes existent pour assure la rapidité des applications du traitement de signal comme algorithme de BENSTEIN, MAG……etc.

Un nouveau Algorithme de multiplication à une constante a été développé. Il permet de réduire considérablement la consommation de puissance tout en augmentant la vitesse de calcul comme il permet aussi d'assure la complexité linéaire qui est un des avantages principale de ce nouveau algorithme.

L'algorithme développé est basé sur la combinaison de deux algorithmes : algorithme de Radix-2^r et l'algorithme de Dimitrov. Et a fin d'augmenter d'avantage les performances, l'idée de combiner ce nouveau algorithme avec celui de V.Lefvre qui est un algorithme d'élimination des sous expression communes.

Mots Clés :

Double système en base de numérotation (DBNS), Système Linéaire invariant dans le temps (LTI), sans multiplicateur unique / multiple Multiplication Constant (SCM / MCM), Arithmétique Radix-2^R.

Summarizes:

The digital signal processing is based on the coding and the multiplication with a constant. Several algorithms exist to ensure the timeliness of applications such as signal processing algorithm BENSTEIN,MAG ..... etc..

A new algorithm for multiplication constant was developed. It can significantly reduce power consumption while increasing computing speed as it allows also ensures linear complexity which is one of main advantages of the new algorithm.

The developed algorithm is based on the combination of two algorithms: Radix-2r algorithm and the algorithm of Dimitrov. And end to increase the performance advantage, the idea of combining this new algorithm with that of V.Lefvre which is an algorithm for disposal of common expression..

Keywords:

Double numbering system base (DBNS), linear time-invariant system (LTI), no single multiplier / Multiple Constant Multiplication (SCM / MCM), Arithmetic Radix-2^r.

Introduction générale

Le traitement de signal est une discipline indispensable de nos jours. On trouve ses applications sur plusieurs appareils comme le radar, la télévision à haute définition, le modem câblé.... .....etc .

Il a pour objet l'élaboration et l'interprétation des signaux porteurs des informations, et l'extraction de ces derniers sous forme de codes binaires (code binaire, Gray, DCB,..)

La multiplication à une constante unique ou multiple est une opération courante dans beaucoup d'algorithmes de traitement de signaux numériques. Elle est considérée comme une opération chère par rapport aux autres opérations (addition, soustraction,...) car elle prend plus de temps pour l'exécution des mêmes ressources logiques que les autres opérations et elle garantit une vitesse d'exécution et une complexité minimal des algorithmes.

La multiplication à une constante pose plusieurs problèmes. Ces problèmes sont liés à la complexité des algorithmes, aux nombres d'additionneurs utilisés et au de temps d'exécution des algorithmes.

Pour résoudre ces problèmes, plusieurs algorithmes ont été développés comme l'algorithme de BENSTIEN, l'algorithme de MAG.... qui sont de complexité élevée. Un nouvel algorithme a été développé qui est une combinaison entre deux algorithmes : l'algorithme Radix-2^r et l'algorithme de Dimitrov dont le but est de minimiser le nombre maximum d'additionneurs et d'exécuter l'algorithme avec une complexité linéaire.

Dans ce mémoire, on a partagé notre travail en quatre chapitres pour assurer un bon partage des informations. Nous avons commencé au premier chapitre par la présentation des généralités sur le traitement de signal et la définition des circuits intégrés et avec quel niveau d'intégration on implémente ce type d'algorithme. Nous passerons au deuxième chapitre en définissant la multiplication à une constante et les problèmes que cette dernière pose et on présente les algorithmes proposés pour résoudre ces problèmes. Au chapitre trois nous présentons en détails l'algorithme de Radix-2^r avec une présentation de quelques comparaisons avec des algorithmes similaires. A la fin, au quatrième chapitre nous vous proposons les étapes du programme développé.

Sommaire

Sommaire

Sommaire

Chapitre 02 : Multiplication par une constante

Sommaire

Chapitre 03 : Algorithme Radix-2^r

Chapitre 04 : Programmation d' algorithme Radix-2^r

Liste des Tableaux

Liste des Figures

Chapitre 01 : Généralité sur le traitement de signal

I. Définition :

I.1. Signal [1]:

Un signal est la représentation physique de l'information, qu'il convoie de sa source à son destinataire.

La description mathématique des signaux est l'objectif de la théorie de signal, elle offre les moyens d'analyser, de concevoir et de caractériser des systèmes de traitement de l'information.

I.2. Bruit [1]:

Un bruit correspond à tout phénomène perturbateur gênant la transmission ou l'interprétation d'un signal.

I.3. Rapport signal/bruit [1]:

Le *rapport signal / bruit* mesure la quantité de bruit contenue dans le signal.

Il s'exprime par le rapport des puissances du signal ($\mathbb{P}_S$) et du bruit ($\mathbb{P}_N$). Il est souvent donné en décibels (**dB**).

Le rapport du signal par le bruit est calculé avec la formule suivante

$$\left(\frac{S}{N}\right) = 10 \log \frac{Ps}{Pn}$$

II. Traitement de signal [2]:

Le traitement de signal est une discipline indispensable de nos jours.

Il a pour objet l'élaboration ou l'interprétation des signaux porteurs d'informations Son but est donc de réussir à extraire un maximum d'information utile sur un **signal** perturbé par du **bruit** en s'appuyant sur les ressources de l'électronique et de l'informatique.

---

[1] : **T.Dumartin** [Rappel Traitement Du Signal].2004
[2] : **A.Oumnad** [Traitement Numerique Du Signal].

III. Classification des signaux [1]:

On peut envisager plusieurs modes de classification pour les signaux suivant leurs propriétés

III.1 Classification phénoménologique :

On considère la nature de l'évolution du signal en fonction du temps. Il apparaît deux types de signaux :

III.1.a) Les signaux déterministes :

Ou signaux certains, leur évolution en fonction du temps peut être parfaitement modélisé par une fonction mathématique. On retrouve dans cette classe les signaux périodiques, les signaux transitoires, les signaux pseudo-aléatoires,.... etc

III.1.b) Les signaux aléatoires :

Leur comportement temporel est imprévisible, il faut faire appel à leurs propriétés statistiques pour les décrire. Si leurs propriétés statistiques sont invariantes dans le temps, on dit qu'ils sont stationnaires.

III.2 Classification énergétique :

On considère l'énergie des signaux. On distingue :

III.2.a) Les signaux à énergie finie :

Il possède une puissance moyenne nulle et une énergie finie.

III.2.b) Les signaux à puissance moyenne finie :

Il possède une énergie infinie et sont donc physiquement irréalisable.

III.3 Classification morphologique :

On est alors amené à distinguer deux types de signaux :

- Le Signal Analogique
- Le Signal Numérique

III.3.a) Signal analogique [3]:

Les signaux analogiques sont des signaux à temps continu. Ils seront représentes par des fonctions mathématiques d'une variable réelle.

Les signaux analogiques sont en général les signaux naturels mesures à l'aide de divers capteurs physiques.

On considérera deux grandes classes des signaux : ***Les Signaux Périodiques Et Les Signaux A Temps Fini.***

- ***Signaux périodique :***

Un signal s périodique de période T est défini comme une fonction $s : \mathbb{R} \rightarrow \mathbb{C}$ telle que pour tout $t, s(t + T) = s(t)$.

Ces signaux ne sont pas physiquement réalisables car à temps infini. Cependant, ils sont d'une importance majeure pour la modélisation et la compréhension des signaux rencontres.

- ***Signaux à temps fini :***

Les signaux a temps fini sont les signaux rencontres en pratique. Ce sont des signaux a support borne.

III.3.b) Signal numérique [3] :

Les signaux numériques sont obtenus à partir des signaux analogiques par *Echantillonnage, Quantification et Codage.*

Ces signaux ont une importance particulière car ce sont ceux-là qui pourront être manipules sur un ordinateur.

Ils sont modélise en mathématiques par des suites à valeur dans R ou C.

$$S : Z \rightarrow C$$
$$k \rightarrow s[k] = sk$$

---

[3] : **Michel Terré** [Traitement Numérique Du Signal] 2011.

IV. Comparaison entre les deux signaux analogique & numérique[1]

L'analogique et le numérique sont deux procédés pour transporter et stocker des données de type audio, photo, vidéo… .

L'analogique est né avec le début de l'électricité tandis que le numérique est apparu plus récemment avec l'ère de l'informatique.

Le principe de l'analogique est de reproduire le signal à enregistrer audio, vidéo… sous forme similaire sur un support (magnétique en général) par exemple lorsque l'on enregistre un signal audio sur un système analogique le signal présent sur la bande magnétique suivra les mêmes amplitudes, " la même courbe ", que l'onde sonore avec plus ou moins de fidélité : les variations de pression acoustique caractéristiques d'une onde sonore seront traduites en variations d'un signal électrique.

Ainsi l'amplitude électrique du signal analogique sera l'image plus ou moins fidèle du Signal à enregistrer audio, vidéo… En numérique le signal analogique à enregistrer est converti en numérique grâce à un convertisseur analogique/numérique.

La tâche du convertisseur analogique/numérique est de traduire le signal en une séquence de nombres binaires.

Après cette conversion le signal n'est plus qu'une suite de " **0** " et de " **1** " c'est à dire un signal à deux amplitudes au lieu d'une infinité en analogique.

Après un transport et un stockage en numérique tout signal (vidéo ou audio) devra revenir à sa forme analogique de départ. Par exemple un signal audio sera reconverti de numérique en analogique pour ensuite être amplifié.

En effet nos oreilles ne savent pas entendre en numérique, Il faut bien garder à l'esprit que le numérique ne sert dans le domaine de l'audio et de la vidéo qu'au transport, au stockage et au traitement des données

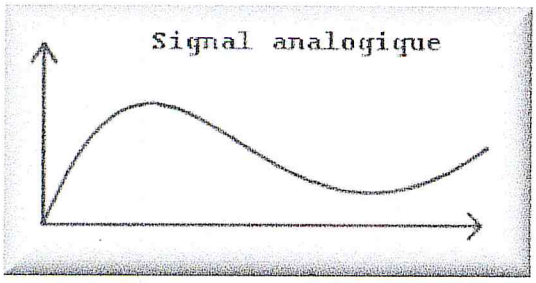


Figure 01 : Représentation de Signal Analogique

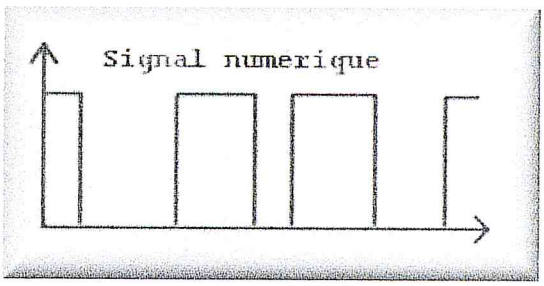


Figure 02 : Représentation de Signal Numérique

Un signal numérique est beaucoup plus facile à reproduire qu'un signal analogique.

La copie d'une cassette audio provoque des pertes de la qualité du signal, alors que la copie numérique produit un clone parfait de l'original.

En numérique, on peut en théorie faire une infinité de copies en ayant strictement la même qualité que l'original.

V. Numérisation du signal [1]:

L'importance des systèmes numériques de traitement de l'information ne cesse de croître (radio, télévision, téléphone, instrumentation…).

Ce choix est souvent justifié par des avantages techniques tels que la grande stabilité des paramètres, une excellente reproductibilité des résultats et des fonctionnalités accrues.

Le monde extérieur étant par nature analogique, une opération préliminaire de conversion analogique numérique est nécessaire.

La conversion analogique numérique est la succession de trois effets sur le signal analogique de départ :

- **L'Echantillonnage** pour rendre le signal discret
- **La Quantification** pour associer à chaque échantillon une valeur
- **Le Codage** pour associer un code à chaque valeur.

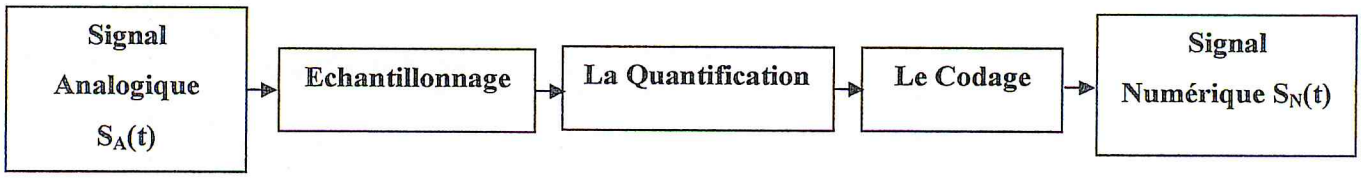


Figure 03 : La Chaine des Effets de Conversion Analogique Numérique.

V.1. Echantillonnage [4] :

L'échantillonnage consiste à prélever à des instants précis, le plus souvent équidistants, les valeurs instantanées d'un signal.

Le signal analogique $s(t)$, continu dans le temps, est alors représenter par un ensemble de valeur discrète : $se(t) = s(n.Te)$

Avec n : entier

Te : période d'échantillonnage.

---

[4] : **I.Zambettazis** [Numerisation Du Signal].

Cette opération est réalisée par un échantillonneur souvent symbolisé par un interrupteur.

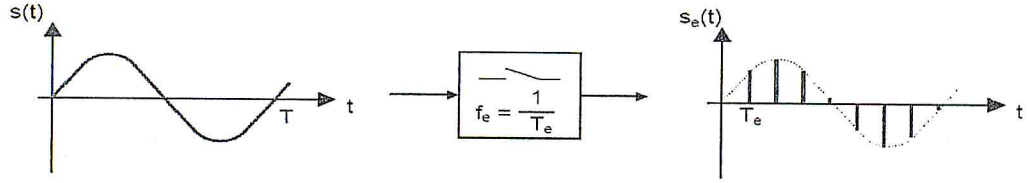


Figure 04 : Echantillonnage du Signal.

On distingue trois types d'échantillonnage :

V.1.a) Echantillonnage idéal :

L'échantillonnage idéal est modélisé par la multiplication du signal continu ***s(t)*** et d'un peigne de Dirac de période ***Te***.

V.1.b) Echantillonnage réel :

En pratique, l'échantillonnage s'effectue en commandant un interrupteur par un train d'impulsions étroites.

Il est donc impossible d'obtenir des échantillons de durée quasiment nulle. La modélisation de l'échantillonnage par un peigne de Dirac est donc erronée.

En fait, chaque impulsion va avoir une durée très courte τ. L'échantillonnage peut donc être modélisé par la multiplication du signal par une suite de fonction rectangle de largeur τ.

V.1.c) Echantillonnage-blocage :

En pratique, on n'échantillonne pas un signal pour le reconstruire juste après.

L'échantillonnage est utilisé pour prélever le signal à des instants multiples de ***Te*** et ensuite convertir les échantillons sous forme d'un code binaire ***(8, 12, 16 bits, ...)***.

Cette conversion est effectuée par l'intermédiaire d'un convertisseur analogique-numérique (*CAN*). Cette conversion n'est pas instantanée. Si le signal à convertir varie trop rapidement, il est nécessaire de procéder au blocage du signal pour avoir une conversion sans erreur.

On utilise donc un échantillonneur-bloqueur qui mémorise la tension à convertir et la maintient constante pendant toute la durée de conversion.

V.2. Quantification [4]:

La quantification consiste à associer à une valeur réelle x quelconque, une autre valeur x_q appartenant à un ensemble fini de valeurs et ce suivant une certaine loi : arrondi supérieur, arrondi le plus proche, etc…

L'écart entre chaque valeur x_q est appelé ***pas de quantification***.

Le fait d'arrondirai la valeur de départ entraîne forcément une erreur de quantification que l'on appelle le bruit de quantification.

V.3.Codage [4]:

Le codage consiste à associer à un ensemble de valeurs discrètes un code composé d'éléments binaires.

Les codes les plus connus : code binaire naturel, code binaire décalé, code complément à 2, code DCB, code Gray.

Exemple sur 4 bits :

Nombre	Binaire	Binaire Décale	DCB	Gray	Complément a 2
-8	/	0000	/	/	1000
-3	/	0101	/	/	1101
0	0000	1000	0000	0000	0000
1	0001	0001	0001	0001	0001
5	0101	0101	0101	0111	0101
10	1010	/	0001 0000	1111	/
15	1111	/	0001 0101	1000	/

Tableau 01 : Défirent type de codage du variable .

VI. Filtrage numérique :

VI.1.Définition du filtrage numérique :

En électronique, un filtre numérique est un élément qui effectue un filtrage à l'aide d'une succession d'opérations mathématiques sur un signal discret.

C'est-à-dire qu'il modifie le contenu spectral de signal d'entrée en atténuant ou éliminant certaines composantes spectrales indésirées. Contrairement aux filtres analogiques, qui sont réalisés à l'aide d'un agencement de composantes physiques (résistance, condensateur, inductance, transistor, etc.), les filtres numériques sont réalisés soit par des circuits intégrés dédiés, des processeurs programmables (FPGA, microprocesseur, DSP, microcontrôleur, etc.), soit par logiciel dans un ordinateur.

VI.2.Définition de système [5] :

On peut définir les systèmes comme les modèles mathématiques des diverses transformations subies par les signaux : par exemple une onde qui se propage est modifiée suivant les caractéristiques de transmission du milieu qu'elle traverse ;

Une quantité physique mesurée par un capteur subit une certaine altération, qui traduit l'effet de la < réponse > du capteur.

Plus généralement, on caractérise la relation d'entrée-sortie d'un système quelconque par un operateur mathématique qui associe `a un signal d'entrée $x(t)$ un signal de sortie $y(t)$.

On s'intéressera plus spécifiquement aux cas particuliers des filtres linéaires homogènes, dont la caractéristique d'entrée-sortie est un operateur linéaires invariant dans le temps.

Ces filtres possèdent de nombreuses propriétés mathématiques et fournissent souvent un premier niveau de description satisfaisant (par exemple, pour des petites variations du signal d'entrée) de systèmes physique beaucoup plus compliques.

Un système est un dispositif représenté par un modèle mathématique de type Entrée/Sortie qui apporte une déformation au signal (Ex: modulateur, filtre, etc…).

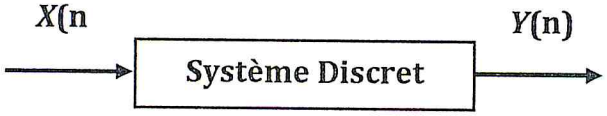
---

[5] ;**S.Tisserant** [Traitement Du Signal]2008

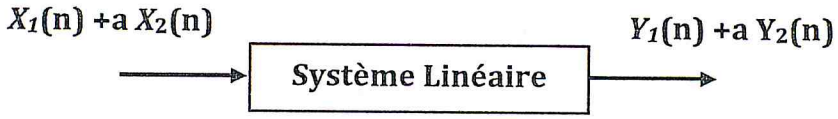
VI.3. Les Systèmes Linéaires Discrets Invariants Dans Le Temps [3]:

VI.3.a) Définitions :

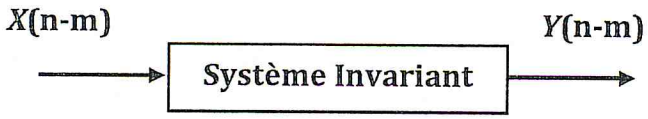
- Un système est ***Discret***, si à la suite d'entrée discrète ***x(n)*** correspond une suite de sortie discrète ***y(n).***



- Un système est ***Linéaire***, si à la suite ***x1 (n) + a x2 (n)*** correspond la suite ***y1 (n) + a y2 (n)***.



- Un système est ***Invariant Dans Le Temps***, si à la suite ***x(n-m)*** correspond la suite ***y(n-m).***



- Dès lors si $\delta(n)$ est la suite unitaire $\begin{cases} \delta(0) = 1 \\ \delta(n) = 0 \ \forall n \neq 0 \end{cases}$ alors toute suite *x(n)* peut s'écrire:

$$x(n) = \sum_{m=-\infty}^{m=+\infty} x(m)\delta(n-m)$$

- si *h(n)* est la réponse d'un système discret linéaire et invariant dans le temps à la suite $\delta(n)$ alors :

$$x(n) \rightarrow y(n) = \sum_{m=-\infty}^{m=+\infty} x(m)h(n-m) = \sum_{m=-\infty}^{m=+\infty} h(m)x(n-m)$$

• on reconnaît alors une équation de convolution: $x(n) = h(n) * x(n)$

Ainsi dès qu'un système peut être considéré comme linéaire, discret et invariant dans le temps, il en découle qu'il est :

1) Régi par une équation de convolution

2) Entièrement déterminé par la réponse *h(n)* qu'il fournit lorsqu'il est excité par la suite impulsion elle $\delta(n)$ Cette suite *h(n)* constituant la réponse impulsionnelle du système.

VI.3.b) Condition de stabilité :

Un système discret, linéaire et invariant dans le temps (***LIT***) est stable si à toute suite d'entrée bornée correspond une suite de sortie bornée.

Une condition nécessaire et suffisante pour qu'un système soit stable est que la somme des valeurs absolues de sa réponse impulsionnelle soit bornée.

$$\sum_{m=-\infty}^{m=+\infty} |h(m)| < +\infty$$

Preuve de la condition nécessaire :

Soit *x(n)* la suite d'entrée bornée définie par : $x(-n) = \delta(h(n))$ alors, par définition de l'équation de convolution régissant le système $y(0) = \sum_{m=-\infty}^{m=+\infty} |h(m)|$ Donc si $y(0) = \sum_{m=-\infty}^{m=+\infty} |h(m)|$ n'est pas $< \infty$ la suite de sortie n'est pas bornée et la condition de stabilité n'est pas respectée.

Preuve de la condition suffisante :

Soit *x(n)* une suite d'entrée bornée, c'est à dire: $\forall n \exists \because |x(n)| < M$

Alors :

$$|y(n)| \leq \sum_{m=-\infty}^{m=+\infty} |h(m)x(n-m)| \leq \sum_{m=-\infty}^{m=+\infty} |h(m)|M$$

Et si $\sum_{m=-\infty}^{m=+\infty} |h(m)| < \infty$, la suite y(n) est alors bornée

VII. Les Circuits intégrés :

VII.1. Présentation [3]:

Les fonctions logiques peuvent être matérialisées par l'intermédiaire de circuits intégrés, ces circuit sont caractérisé par :

VII.1.a) Tension d'alimentation :

Les circuits de technologie ***CMOS*** peuvent être aliments entre 3 et 18 volts, par contre les circuits de technologie ***TTL*** doivent impérativement être alimentés sous 5 volts.

VII.1.b) Vitesse de commutation :

C'est le temps que met le signal pour traverser l'opérateur logique ; on l'appelle aussi temps de propagation.

VII.1.c) Règles d'association :

D'usage, on ne mélange pas plusieurs technologies entre elles. Cependant il existe des structures d'interface permettant de relier deux technologies différentes. Ces structures peuvent être matérialisées par l'intermédiaire de transistors.

Les entrées non utilisées doivent être connectées soit à l'alimentation soit à la masse. Ne jamais relier deux sorties entre elles sauf le cas de sortie à collecteur ouvert.

VII.1.e) Immunité au bruit :

L'immunité au bruit est le degré avec lequel une porte logique peut supporter les variations d'entrées, sans que cela entraîne de modification du niveau de sortie. Ces circuits sont classés selon leurs caractéristiques et leur domaine d'emploi.

Ce classement est représenté selon l'arborescence ci-dessous :

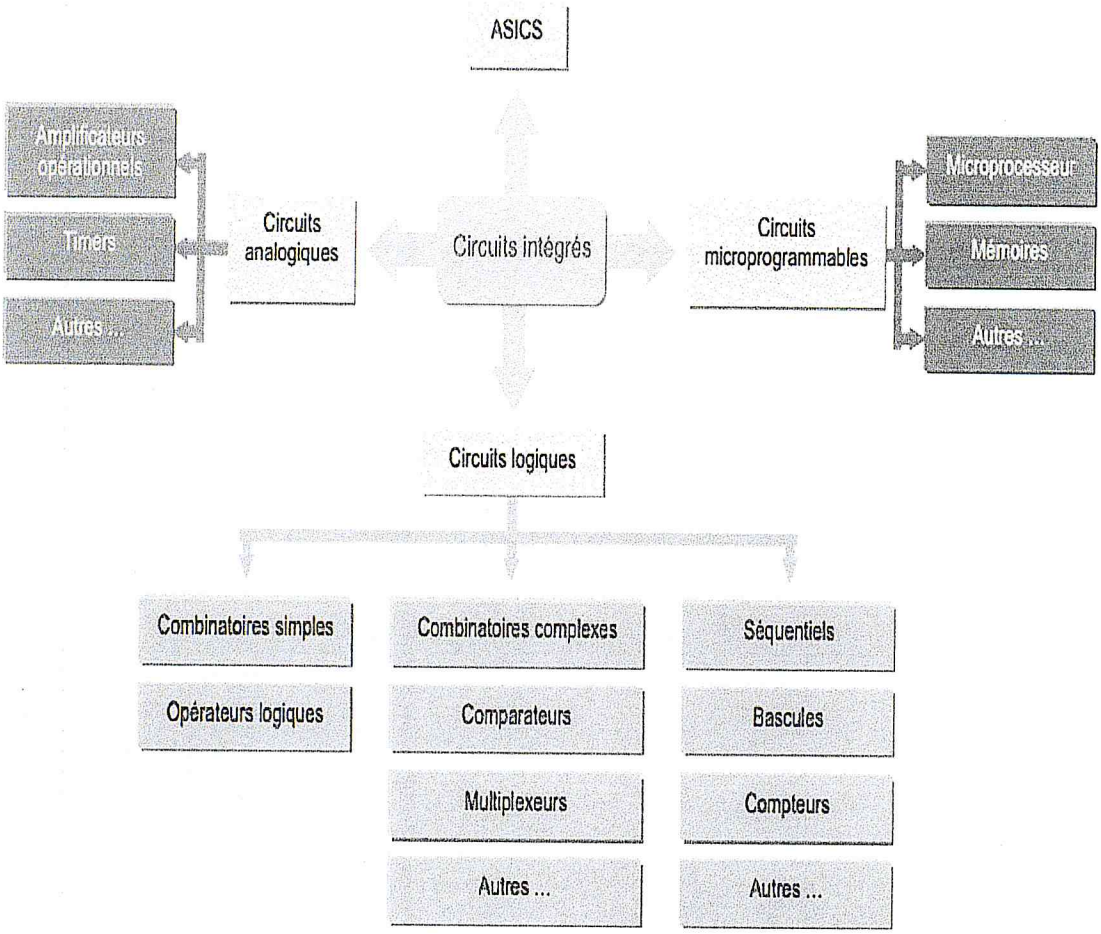


Figure 05 : Classification des Circuits Intégré

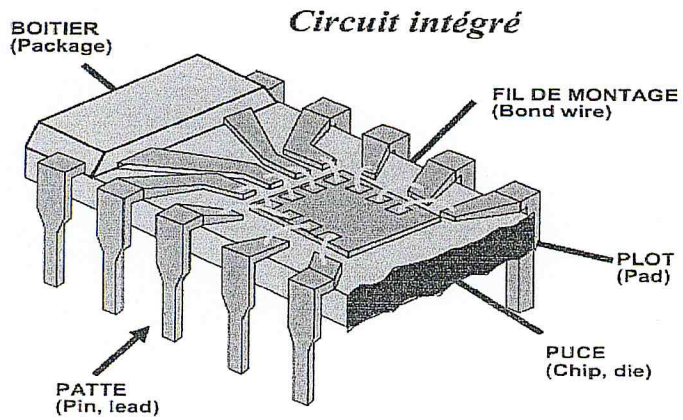


Figure 06 : Les Composent d'un circuit Intègre

VII.2. Circuit intégré analogique :

Les composants les plus simples peuvent être de simples transistors encapsulés les uns à côté des autres sans liaisons entre eux, jusqu'à des assemblages complexes pouvant réunir toutes les fonctions requises pour le fonctionnement d'un appareil dont il est le seul composant.

Les amplificateurs opérationnels sont des représentants de moyenne complexité de cette grande famille où l'on retrouve aussi des composants réservés à l'électronique haute fréquence et de télécommunication. Comme exemple de circuit analogique : l'ampli op LM741 et de nombreux cousins.

VII.3. Circuit intégré numérique :

Les circuits intégrés numériques les plus simples sont des portes logiques (et, ou et non), les plus complexes sont les microprocesseurs et les plus denses sont les mémoires. On trouve de nombreux circuits intégrés dédiés à des applications spécifiques (ou ASIC pour Application-spécifique intégrâtes circuit), notamment pour le traitement du signal (traitement d'image, compression vidéo...) on parle alors de processeur de signal numérique (ou DSP pour Digital Signal Processor).

Une famille importante de circuits intégrés est celle des composants de logique programmable (FPGA, CPLD). Ces composants sont amenés à remplacer les portes logiques simples en raison de leur grande densité d'intégration.

VII.4. Niveau d'intégration :[6]

Avec l'évolution technologique, la miniaturisation de plus en plus poussée permet d'augmenter le nombre de composants au millimètre carré. On distingue les niveau d'intégration suivant :

SSI : Single Size Intégration, intégration à faible échelon (< 10 portes) .

---

[6] : **Mr.M.V.Sathish, Mrs Sailaja** [VLSI architecture of parallel multiplier– accumulator based on Radix-2 modified booth algorithm].2007

MSI : Medium Size Intégration, intégration à moyenne échelle (entre 10 et 100 portes).

LSI : Large Size Intégration, intégration à grande échelle (entre 100 et 1 000 portes) .

VLSI : Very Large Size Intégration, intégration à très grande échelle (>1 000 portes).

VII.4.a) Intégration a très grand échelle [6]:

L'intégration à très grande échelle (*VLSI* - Very-Large-Scale Integration) est une technologie de circuit intégré (*CI*) dont la densité d'intégration permet de supporter plus de 100 000composants électroniques sur une même puce.

Elle a été réalisée pour la première fois dans les années 1980, dans le cadre du développement des technologies des semi-conducteurs et des communications.

Les premières puces à semi-conducteurs supportaient un seul transistor chacune. Avec les progrès subséquents, on est arrivé à ajouter de plus en plus de transistors, et en conséquence, de plus en plus de fonctions ou de systèmes individuels ont pu, avec le temps, être intégrés. Un microprocesseur est un dispositif VLSI.

La première *génération* d'ordinateurs utilisait des tubes à vide. Puis, vinrent les dispositifs à semi-conducteurs, suivis des circuits intégrés. Les CI intègrent plusieurs dispositifs sur la même puce: des diodes, des transistors, des résistances et des condensateurs voire des inductances, ce qui rend possible la fabrication d'une ou de plusieurs portes logiques sur un même circuit. La quatrième génération a, ainsi, consisté en une intégration à grande échelle (LSI - Large-Scale Intégration), c'est-à-dire en des circuits comportant de 1 000 à 10 000 composants. La technologie VLSI a succédé naturellement à la LSI.

Les techniques actuelles dépassent de loin ce niveau, de sorte que les microprocesseurs fabriqués aujourd'hui comportent plusieurs millions de portes. Issus de cette technologie, les réseaux logiques programmables (*FPGA* - *Field-Programmable Gate Arrays*) sont des composants VLSI entièrement reconfigurables, ce qui permet de les reprogrammer à volonté afin d'accélérer considérablement des calculs complexes

Conclusion :

Le traitement de signal à une grande importance au réussir de fonctionnement de plusieurs machine et produit électrique qui ce base à des circuits intégré a sont fonctionnement.

Ainsi que le traitement de signal a plusieurs avantages comme la flexibilité de reconfiguration des opérations de traitement des signaux et ça mise en œuvre quelle été pas couteuse ….etc. Mais ce reste toujours pas parfait a cause de sont matériel qui est trés trés chère et limite de convertir toute les signaux que ce soit a cause de vitesse de signal ou bien sa faiblesse.

Mais comme le traitement du signal à jouer des grands rôles a fonctionnement des machine les operateurs arithmétique sont jouent un rôle pour optimise de la dure et la complexité et même sa puissance et on ce voix plus de détails au prochaine chapitre de notre mémoire.

Chapitre 02:
Multiplication par une constante

I. Applications de la multiplication à une constante [7] :

La multiplication par un jeu de constantes se produit lors de la multiplication par un vecteur constant ou une matrice constante par exemple, le produit scalaire **a . b** donne la projection scalaire de ***a*** sur ***b***.

Multiplication par une matrice constante n'est rien de plus effectuant le produit scalaire entre les plusieurs vecteurs constantes (*qui forment collectivement une matrice*) et un vecteur de variables (*les éléments de ce vecteur sont les entrées*).

Multipliant par une matrice constante peut ainsi être considéré comme une transformation linéaire des coordonnées, qui est utilisé dans de nombreuses applications par exemple, la conversion du **RGB** (*rouge, vert, bleu*) d'espace de couleur de l'espace de couleur **YUV** (*Y représente la luminosité, U et V représenter chroma*) implique une multiplication par une matrice 3x3 constant.

Parce que l'œil humain est plus sensible à la luminosité de la coloration (*chroma*), on peut comprimer les informations contenues dans les composantes U et V avec seulement une faible distorsion perçue.

Ceci est exploité en JPEG et MPEG pour la compression des images et des vidéos, respectivement.

Beaucoup des signaux linéaire discrète transformé impliquer la multiplication par des constantes, telles comme la transformée de Fourier discrète (**DFT**) et d'autres transformées de Fourier connexes comme la transformée en cosinus discrète.

Le signal d'entrée a été fourni par une sinusoïde à la fréquence f (*pour différentes f*). Ces Les transformations peuvent également être considérées comme une transformation linéaire de coordonnées et en tant que tel, ils peuvent être utilisés pour la compression de signaux.

L'objectif est de transformer l'entrée signal dans une base de coordonnées de telle sorte que la majorité de l'énergie (*ou informations*) dans le signal est concentrée dans seulement quelques composants.

---

[7] **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.

Nous pouvons alors compresser ou ignorer même complètement les composants de faible énergie avec seulement un minimum de distorsion.

Dans les communications, la compression du signal est essentielle pour minimiser la quantité d'énergie nécessaire à la transmission d'informations. Ceci est particulièrement important dans les appareils mobiles, en qui vie de la batterie est une préoccupation majeure.

Comme exemple de signaux compression, de nombreuses images de la vie réelle (*par opposition au hasard pixels générés*) sont typiquement composés de contenu en fréquences basses. Par exprimant une image dans son domaine de fréquence, on peut comprimer la haute fréquence d'informations sans trop de distorsion. Ceci est également exploité dans JPEG et MPEG.

De même, l'oreille humaine est moins sensible aux sons à haute fréquence. Exploits MP3 cela en utilisant moins de compression pour les basses fréquences et plus de compression pour une grande fréquence.

Lors de la décompression pour écouter l'audio, après que nous défaire la compression, il faut transformer à partir du domaine fréquentiel au domaine temporel, ce qui implique à nouveau un signal linéaire transformer avec la multiplication constante.

Les filtres **FIR** et **IIR** sont sans doute les moyens les plus simples d'effectuer un filtrage numérique (*Et donc probablement la moins chère à mettre en œuvre*).

Le filtrage est utilisé pour atténuer ou à gagner chaque fréquence dans le signal d'entrée d'une quantité souhaitée, il peut donc être utilisé pour modifier les caractéristiques de réponse de fréquence dans un système. Par exemple, une communication canal peut atténuer certaines fréquences plus que d'autres, mais en ayant une estimation de la réponse en fréquence du canal, nous pouvons appliquer un filtre avec l'atténuation inverse de chaque fréquence à la sortie du canal afin de rétablir le signal d'origine.

Dans les communications, les filtres **FIR** / **IIR** sont utilisés pour l'égalisation, suppression d'écho, etc

Les filtres **FIR** sont généralement utilisés pour sur échantillonnage et sous-échantillonnage des signaux numériques. En les deux cas, un filtre d'interpolation est généralement utilisé pour lisser le signal nouvellement échantillonnée.

Cela a de nombreuses applications dans le traitement des fichiers audio et vidéo par exemple, redimensionnement d'une reproduire de l'audio ou vidéo à des vitesses différentes nécessite un changement de la fréquence d'échantillonnage.

Les filtres **FIR** sont également utilisés dans le traitement de l'image. Par exemple, l'opérateur de Laplace peut être utilisé pour la détection de bord, et l'un des moyens les plus simples d'application du présent est avec un filtre FIR à 2 dimensions additionneurs.

II. La Multiplication à une constante [8] :

II.1 Généralités sur la multiplication:

Nous pouvons supposer que tous les coefficients constants seront des nombres entiers, comme nombres réel peuvent être déplacées jusqu'à ce qu'ils deviennent un entier (ce qui ne nécessite pas de coût, et ce processus est réversible). C'est pourquoi nous allons seulement envisager multiplication entier.

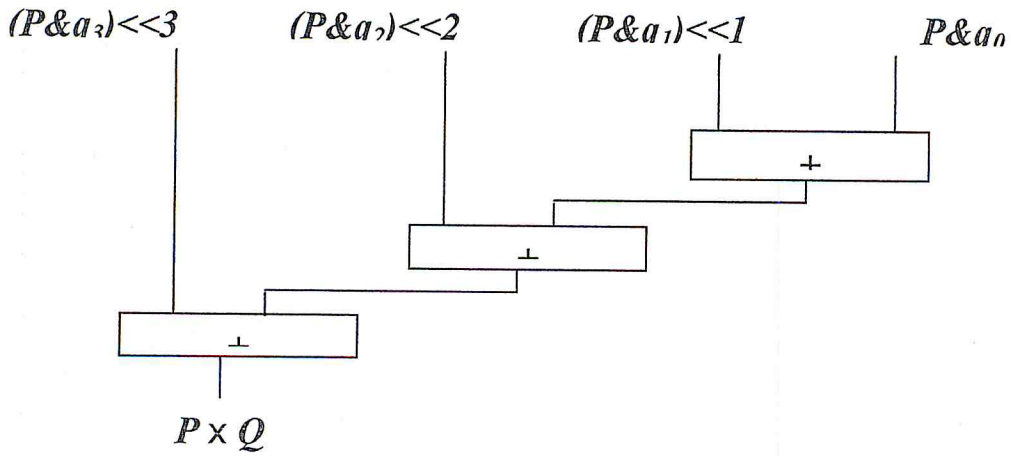


Figure 7: Un général multiplicateur 4 bits.

Un décalage vers la gauche par n bits est notée << n.

Q a une représentation binaire de $q_3 q_2 q_1 q_0$, donc $q_i \in \{0;1\}$ pour i = {0 ; 1 ; 2 ; 3}.

Notez que $P \& q_i$ est une représentation simplifiée d'utilisation d'une porte **ET** entre q_i et chaque bit de P, donc si P est **m** bits alors $P \& q_i$ est aussi **m** bits.

---

[8] :**Y. Voronenko and M. P¨uschel** [Multiplier les Multiple Constant Multiplication] 2007.

En base binaire, chaque chiffre est égal à **0** ou **1**, de sorte que le multiplicande n'est jamais multiplié par **0** ou **1** ,par conséquent, en base 2, la multiplication peut être décomposé en un ensemble des ajouts et des changements de gauche .

Par exemple la multiplication générale de deux arbitraires binaires entiers **P** et **Q**

Supposons **Q** peut être représentée sur N bits, et donc Q a une représentation binaire de $q_{N-1}\ q_{N-2}\ q_{N-3}\ ..\ ..q_3\ q_2\ q_1\ q_0$.

$(Q)_{10} = (\ q_{N-1}\ q_{N-2}\ q_{N-3}\ ..\ ..q_3\ q_2\ q_1\ q_0)_2$ où chaque bit $q_i \in \{0;1\}$.

Soit $\mathbf{R} = \mathbf{P} * \mathbf{Q} = \sum_{i=0}^{N-1}(P \times q_i) \ll i$.

alors **R** =. Évidemment $\mathbf{P} * q_i = 0$ si i = 0 et $\mathbf{P} * q_i = P$ si i = 1.

Pour mettre en œuvre, une porte **ET** est utilisée entre q_i et chaque bit de **P** un général N multiplicateur binaire doit avoir $N * 1$ Additionneur

II.2. Utilisation de la technique de la soustraction à la multiplication [7] :

Une logique plus petite peut être obtenue en considérant la soustraction. À titre d'exemple, considérons multipliant un nombre arbitraire x de 7 (111 en binaire).

De la multiplication générale, il s'ensuit que $7x = (x << 2) + (x << 1) + x$. Considérons maintenant $7x$ Décompose comme suit $8x - x = (x << 3) - x$.

Cette dernière décomposition est mieux depuis addition et la soustraction exigent à peu près la même quantité de ressources logiques Pour simplifier la discussion sur ressources logiques.

Rappel de la multiplication générale de la définition de $\mathbf{R} = \mathbf{P} * \mathbf{Q} = \sum_{i=0}^{N-1}(P * q_i) \ll i$

Dans une représentation binaire, $q_i \in \{0,1\}$ est appliquée en soi, mais si nous laissons $q_i = -1$, Soustraction sera utilisé depuis $\mathbf{P} * q_i$, serait alors égale -**P**. Ajout d'un terme négatif en somme est équivalente à l'utilisation de la soustraction.

Dans la suite de cette thèse, nous allons utiliser $\bar{1}$ pour représenter -1. Permettre $q_i \in \{\bar{1}, 0, 1\}$ facilite l'utilisation des deux additions et des soustractions.

Si une constante Q peut être représentée par n chiffres non nuls (où chaque chiffre $q_i \in \{\bar{1}, 0, 1\}$), puis multiplication par Q nécessite $n - 1$ additionneurs (pour ajouter les n termes que les chiffres non nuls n représentent).

En permettant également la soustraction, nous pourrions être en mesure de réduire le nombre de non nuls chiffres Q. Une série de signaux consécutifs dans la représentation binaire de la constante nécessite de nombreux ajouts, mais une seule soustraction remarquez que $\sum_{i=0}^{N-1} 2^i = 2^N - 1$ pour exemple, 111 peut être remplacé en $100\,\bar{1}$.

De même, $1111 \rightarrow 1000\bar{1}$, $11111 \rightarrow 10000\bar{1}$, et ainsi de suite. Evidemment $1 \rightarrow 1\bar{1}$ est une valable transformer, mais il n'est d'aucune utilité, car elle augmente le nombre de chiffres non nuls.

La représentation résultant signé chiffres est connu comme le canon signé chiffres formulaire (*CSD*). Un exemple avec Q = **7523** est illustrée à la **figure08** Remarquez la transformation du produit de droite à gauche. À chaque étape, la plus à droite du groupe de signaux consécutifs est éliminé. Chaque 1 nouvellement crée a toujours pas de côté Chiffres non nuls (nous ne considérons pas $1\bar{1} \rightarrow 1$ et il est impossible de créer $\bar{1}\bar{1} \rightarrow \bar{1}1$)

$$1110101100011$$

$$1110101100010\bar{1}$$

$$1110110\bar{1}00010\bar{1}$$

$$1110\bar{1}0\bar{1}00010\bar{1}$$

$$1000\bar{1}0\bar{1}0\bar{1}00010\bar{1}$$

Figure 08: Le chiffre canonique signé (CSD) transformé appliqué à la constante Q = 7523.

La transformation de $011.....111 \rightarrow 100....00\ \bar{1}$ est appliqué à plusieurs reprises de droite vers la gauche jusqu'à ce qu'il n'y chiffre non nuls consécutifs ou plus demeurent.

Avec le $011....111 \rightarrow 100.....\bar{1}$ Transformé, Le prochain groupe de ceux consécutifs à transformer est toujours située à gauche du groupe actuel, si un seul passage à travers la constante est nécessaire. Ainsi, la transformation de CSD nécessite moment de l'exécution linéaire par rapport à la largeur de bit de la constante. La transformation se produit de manière déterministe, donc une représentation unique CSD existe pour chaque numéro.

III. Définition du problème de multiplication constante [7] :

Dans cette thèse, | **R**| désigne la cardinalité de l'ensemble **R** alors que |r| désigne l' absolu de la valeur de l'élément **r** .

Le problème de la multiplication constante étant donné un ensemble **T** unique et différent de zéro des coefficients constants est de trouve un ensemble **R** ={ $r_0, r_1, r_2, \ldots\ldots, r_n$}tels que **T** C **R** , r_0 = 1 , et pour chaque $r_k \in$ R avec **1 < k < n** , il existe des éléments r_i , $r_j \in$ **R** avec 0 <= i, j <k qui satisfaire $r_k \in$ A (r_i , r_j) . L'objectif est de minimiser |**R** | .

En plus générale de A (x, y) , un additionneur-fonctionnement de l' éléments x et y , est l'ensemble de tous les nombres possibles qui peuvent être créés en utilisant les x , y , et un `Additionneur , Soustraction et / ou des changements peuvent ou non être autorisés par l `additionneur".

Nous n'avons pas délibéré a définir explicitement l'additionneur-opération A (x, y) car légèrement différents problèmes (mais quelque peu liés) peuvent être obtenus simplement en permettant ou en interdisant la soustraction et / ou des changements.

IV. La multiplication à une constant unique (SCM)[8] :

La multiplication **y = t x** d'une variable ***x*** par un nombre entier ou connue du point fixe constants **t** peuvent être décomposés en additions (ajoute), soustractions (soustrait), et décalages binaires mais le problème qui ce pose est de trouver la décomposition avec le moins des opérations est connus comme une seule multiplication constante (SCM).

Sans perte de généralité, nous supposons que les constantes sont des nombres entiers, car une multiplication en virgule fixe est équivalente à une multiplication par un nombre entier suivi d'un décalage vers la droite.

Le problème de SMC est lié à la constante mais il est différent du problème de la chaîne d'ailleurs qui multiplie que par une constante ajoute. L'autorisation des quarts de plaisir modifie fondamentalement le problème et les stratégies pour sa solution.

La méthode simple pour décomposer la multiplication en ajoute et quarts traduit 1 dans la représentation binaire de la constante ***t*** en quarts, et additionne les entrées décalées.

Par exemple, pour T = 71, $71x = 1000111_2 x$ = x «6 + x« 2 + x «1 + x, ce qui nécessite 3 ajoute.

Alternativement, la multiplication peut être décomposé en soustrait et décale en traduisant des 0 en quarts, et en soustrayant de la plus proche composé constante de 1 de seulement (c.-à-de la forme 2^n - 1): $71x = 1000111_2 x$ = (x «7 - x) - x« 5 - x «4 - x« 3,

Prenant le meilleur de ces deux méthodes rendements dans le pire et dans le cas d'une moyenne de solution avec ***b/2 + O (1)*** ajoute / soustrait, où **b** est la largeur de bit de ***t***.

V. La Multiplication A Plusieurs Constante (MCM)[8]:

Une extension de SCM est le problème de la multiplication d'une variable x par plusieurs constantes $t_1, \ldots, T_n$ en parallèle à un bloc que l'on appelle multiplicateur représenté sur la Figure 07.

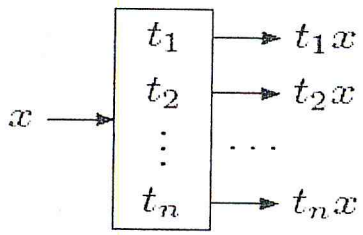


Figure 09 : bloc multiplicateur.

Depuis les résultats intermédiaires des compositions constantes peuvent être partagés, un multiple constant « bloc multiplicateur » peut être décomposé en moins d'opérations que la somme de fonctionnement des comptes des décompositions simples constants.

VI. Problèmes Liés Au SCM Et MCM [7]:

Nous allons brièvement décrire quelques-uns des problèmes liés à la SMC et MCM. Ce sont variantes du problème SCM ou MCM dans laquelle un objectif secondaire ont été ajoutées et / ou une métrique de minimisation légèrement différente est utilisée.

VI.1. Profondeur contraindre [7] :

Un des problèmes secondaires courants consiste à appliquer une contrainte profondeur dans la solution à un SMC ou problème MCM.

La profondeur de l'additionneur est définie comme étant le nombre maximum des additionneurs qui on traverse le long d'une trajectoire de l'entrée vers l'une des sorties dans le circuit logique de multiplication constant.

La profondeur de l'additionneur est une estimation du plus long trajet à travers le circuit logique, qui est connu comme le chemin critique. En raison de la construction physique, des portes logiques ont un retard de propagation, qui est la longueur de temps entre le moment où les entrées sont stables affirmé au moment où toutes les sorties deviennent stables.

En plus des portes logiques sont placées en série entre l'entrée et la sortie, le chemin critique devient plus long et le circuit logique doit être cadencé à un rythme plus lent vitesse, ce qui conduit à un débit de calcul plus faible.

Le chemin critique est aussi une fonction d'autres comme le retard de chaque porte et le temps de passage le long des fils, Mais nous allons faire abstraction de cela dans le but de résoudre de taille réelle SCM et MCM problèmes dans des quantités raisonnables de temps. La plupart (sinon la totalité) des travaux dans ce domaine de recherche utilise la profondeur de l'additionneur afin d'estimer la longueur du chemin critique.

Compte tenu de la même instance de problème, le nombre de couleuvres augmente à mesure que la profondeur contrainte est rendue plus petite. Une solution peut ne pas exister si la profondeur est trop limitée.

VI.2. Minimisation des vipères seules bit [7]:

Pour les petites tailles de problème, il est possible d'utiliser une mesure plus précise (moins abstraite) métrique d'estimer la quantité de ressources logiques nécessaires pour mettre en œuvre coefficient constant multiplication. Comme on a présente précédent que le nombre d'additions ou des soustractions est utilisé comme mesure de la plupart des travaux dans ce domaine de recherche.

Afin de minimiser la logique nécessaire, il est supposé que additionneurs report en cascade sont utilisés. Une extension de report en cascade est un définir des additionneurs sur un seul bit reliés en série.

Notez que lorsque nous calculons outre par main , il suffit de calculer une somme des chiffres et un bagage à la fois (un seul bit additionneur calcule cela en binaire). Chaque transport est passée de droite à gauche, d'où le Nom report en cascade Additionneur.

Pour le même problème SCM ou MCM, un algorithme peut trouver plusieurs solutions" `` meilleurs en termes d'ajouts ou des soustractions.

Dans ce cas, il serait bénéfique pour briser l'égalité en prenant en compte un plus précis métrique, telles que le nombre d'additionneurs à un seul bit.

VII. Cadre algorithmique [8]:

Cependant, il est nécessaire d'introduire d'abord les cadres algorithmiques que la plupart des algorithmes existants utilisent.

La solution d'un problème de multiplication constante (la décomposition addition - soustraction-de décalage module) peut être représenté de différentes manières. Chaque cadre algorithmique fournit une représentation qui permet algorithmes pour rechercher efficacement la solution. nous allons illustrer les propriétés bénéfiques de chaque cadre et donner un aperçu sur la façon dont un algorithme pourrait rechercher une solution en exploitant ledit propriétés, mais nous n'allons pas discuter de tout algorithme particulier.

Dans cette partie , nous allons illustrer comment représenter multiplication de coefficient constant avec des graphiques orientés acycliques (*DAG*) et l'élimination des sous-expressions communes (*CSE*).

VII.1. Réaliser de Graphe Acyclique [8]:

Supposons que R est une solution valable au problème de la constante de multiplication et donc satisfait les contraintes vue au problème de multiplication Sauf pour $r_0 = 1$, un additionneur est nécessaire pour construire chaque élément de R (rappelons un additionneur-opération peut impliquer la soustraction).

Pour chaque indice $k \geq 1$, l'interprétation de la contrainte que $r_k \in$ A (r_i, r_j) où i, j <k, c'est que quand un nouvel élément r_k est ajouté à R, nous devons utiliser deux existante éléments dans R (r_i et r_j dans ce cas) que les opérandes de la vipère qui construira r_k.

Il ya donc une dépendance unidirectionnelle entre les éléments de R.

Les dépendances dans R sont faciles à englober un graphe acyclique dirigé. Chaque nœud dans le *DAG* est marqué par la valeur de l'élément en R qu'il représente. Sauf pour $r_0 = 1$, chaque r_k élément a une dépendance sur deux éléments r_i et r_j. c'est représenté par deux bords dirigés de la r_i et r_j nœuds au nœud r_k (si $r_i = r_j$, il y aura deux bords dirigées de r_i de nœud à nœud r_k). Seul le $r_0 = 1$ nœud a pas de bords venant à lui.

Ceci est connu comme le nœud source depuis le 1er multiplié par le multiplicande vient gratuitement. Si $r_k \in A(r_i, r_j)$, alors il existe des entiers n et m tels que $r_k = |2^m r_i \pm 2^n r_j|$.

Les changements dans le fonctionnement d'additionneur (qui sont exprimés en multiplication par une entière puissance de 2) sont également entourés par le ***DAG***.

Le bord dirigé depuis r_i à r_k est marqué par le 2^m de valeur et le bord de r_j de r_k est marqué avec $\pm 2^n$.

exemple :

$$7x = 8x - x = (x<<3) - x$$
$$113x = 16(7x) + x = ((7x)<<4) + x$$
$$53x = 0{,}5(113x) - 0{,}5(7x) = ((113x) - (7x))>>1$$
[Expression Mathématique]

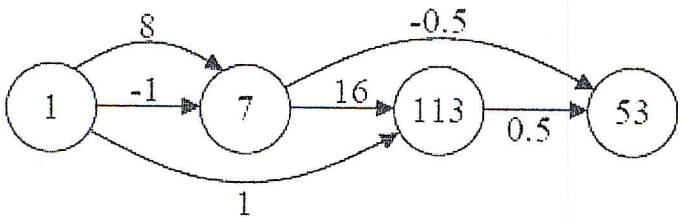


Figure 10 : Le DAG correspondant à l'expression mathématique

VII.2. Notation d'Elimination des Sous Expression Communes [8]:

La notation pour l'élimination de la sous-expression commune est simplement une extension de ***SD*** notation, Avec notation ***CSE*** , un nouveau chiffre est utilisé pour définir un modèle (un ensemble d'exister chiffres). Chaque instance du modèle peut alors être remplacé par ce nouveau chiffre.

Nous allons illustrer par un exemple. Considérez multiplication par 45, qui a un ***CSD*** représentation de $10\bar{1}0\bar{1}01$. Permettez A représentent le multiplicande arbitraire que nous voulons à multiplier par 45 , puis $45 * A = A0\bar{A}0\bar{A}0A = (A << 6) - (A << 4) - (A << 2) + A = (2^6 - 2^4 - 2^2 + 2^0) * A$. en avant nous avons introduit $\bar{1}$pour désigner - 1 , ce qui signifie le terme correspondant sera déduit de la somme .

L'extension à d' autres chiffres , $\bar{A}$ représente -A , par exemple. Nous allons définir un nouveau modèle $B = A0\bar{A} = (A << 2) - A = 3 * A$.

Ainsi, nous avons aussi B = A0$\overline{A}$ = -3 * A. En général, la forme négative de un motif est obtenue en inversant le signe de chaque chiffre de la configuration. Notez le modèle A0$\overline{A}$ ou son négatif apparaît deux fois dans A0$\overline{A}$0$\overline{A}$0A , donc B ou $\overline{B}$ peuvent être substitués à deux endroits pour obtenir 00B000$\overline{1B}$ (B = A0$\overline{A}$ équivaut à 00B = A0$\overline{A}$) .

En substituant C = B000$\overline{B}$, nous obtenons 000000C , donc 45 * A = C = (B<< 4) -B = 15 * B = 15 * 3 * A.

Notez que les zéros dans un modèle sont des espaces réservés, à savoir que nous pourrions substituer D = A000$\overline{A}$ en A0$\overline{A}$0$\overline{A}$0A pour obtenir 0000D0$\overline{D}$.

Modèles numériques signés sont une façon de représenter les termes existants qui sont décalés et ajouté à créé le nouveau terme. Par exemple, B = $\overline{A}$000A moyen B = A - (A << 4).

Un nouveau modèle doit être composé de chiffres existants et avoir au moins 2 chiffres non nuls. Remarque A0A0A qui ne contient qu'une seule occurrence de A0A car il ne peut être remplacé une fois , même si cela peut se faire de deux manières différentes. Si B = A0A , nous pourrions obtenir A000B ou 00B0A , mais clairement 00B0B =/= A0A0A .

Compte tenu de ***n*** chiffres non nuls dans la représentation ***CSE*** d'une constante , ***n*** -1 additionneur sont nécessaire d'ajouter les ***n*** termes . De même , il en coûte **m**-1Additionneur pour créer un modèle avec **m** chiffres non nuls .

L'objectif dans les problèmes ***SCM*** et ***MCM*** est de minimiser l' nombre de couleuvres , de sorte qu'il est bénéfique pour faire un modèle si elle peut être assez substitué fois de sorte que le nombre d'additionneurs enregistrés (par réduction du nombre de chiffres différents de zéro) est au moins aussi grand que le nombre d'additionneurs nécessaires à la construction du modèle

Exemple T={45,75,211} .

$$15x = 16x - x = (x<<4) - x$$
$$45x = 4(15x) - (15x) = ((15x)<<2) - (15x)$$
$$75x = 4(15x) + (15x) = ((15x)<<2) + (15x)$$
$$211x = 256x - (45x) = (x<<8) - (45x)$$

[Expression Mathématique]

Substitution	*45*A*	*75*A*	*211*A*
	$A0\,\overline{A}0\overline{A}0A$	$A0A0\overline{A}0\overline{A}$	$A0\overline{A}0A0A0\overline{A}$
$B=A000\overline{A}$	$0000B0\overline{B}$	$0000B0B$	$A00000\overline{B}0B$
$C=B0\overline{B}$	$000000C$	$0000B0B$	$A0000000\overline{C}$
$D=B0B$	$000000C$	$000000D$	$A0000000\overline{C}$
$E=A0000000\overline{C}$	$000000C$	$000000D$	00000000^{E}

[Substitutions CSE]

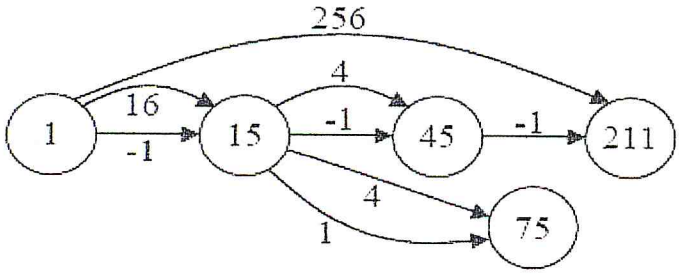


Figure 11 : *Trois Représentations Equivalentes d'une Solution Possible à la MCM Problème avec T = {45, 75, 211}.*

VIII. Algorithmes Existant [9] :

Dans cette section , nous allons donner un aperçu des techniques existantes pour résoudre le ***SMC, MCM*** et d'autres problèmes étroitement liés.

Étant donné que les algorithmes existants couvrent plus de deux décennies, et qui ont proposé les approches fondamentale et des idées qui ont été fréquemment utilisés et améliorés par des algorithmes plus tard.

VIII.1.L'Algorithme MAG :

Dempster et Macleod a proposé le premier optimale Algorithme SMC en 1994, qui a été nommé l'algorithme de graphe d'addition réduite ***(MAG).*** Il vérifie si oui ou non limitativement une solution avec un additionneur existe.

S'il est trouvé, les arrêts de l'algorithme, sinon il vérifie exhaustive pour 2 vipères, puis 3 additionneurs, et ainsi de suite. En cochant exhaustive dans l'ordre croissant de coût

---

[9] :**Florent De Dinechins, Vincent Lefevre**[Constant Multipliers of FPGAs]May 2000

d'addition, nous sommes garantie de trouver la solution optimale (avec le nombre minimum de vipères).

Avec ***n*** additionneurs, il ya seulement tellement de façons possibles d'organiser changement de soustraction ajouter unités dans un arbre d'additionneurs. Nous pouvons donc énumérer tous les correspondantes possibles graphiques topologies.

Pour chaque graphe de topologie, on calcule l'ensemble des valeurs possibles que pourrait être construit au niveau du nœud se termine dans la ***DAG***. Nous pouvons alors comparer ces des valeurs avec le **SMC** donnée cible **T** pour déterminer si la topologie de graphe peut construire ***t***.

Puisque nous cherchons dans l'ordre croissant de coût d'addition, si nous calculons la mesure du possible

Les valeurs pour l'ensemble de la courbe des topologies de ***n*** additionneurs et nous ne pouvons toujours pas construire ***t***,

c'est une preuve suffisante que ***t*** nécessite au moins ***n + 1*** Additionneur.

En raison de la `` première construction, puis vérifiez l'encontre de l'objectif'' nature du algorithme MAG, il peut être utilisé pour trouver la solution optimale SMC pour plusieurs cibles à la en même temps.

En fait, Dempster et Macleod utilisé l'algorithme pour générer le MAG Les solutions de SCM optimales pour toutes les constantes jusqu'à une largeur de bit de 12 ans. L'algorithme crée une table de consultation afin que plus tard, quand un problème SMC doit être résolu pour une cible donné ***t***, la solution peut être consulté à partir de la table. La recherche a été limitée à 4 additionneurs en raison de la puissance de calcul limitée des ordinateurs de bureau à l'époque.

VIII.2 Les algorithmes simples [9] :

L'algorithme le plus simple consiste à écrire la ou les constantes en base 2, et à utiliser la distributivité de la multiplication. Ainsi, si l'écriture binaire de la constante **c** contient ***n*** occurrences du chiffre 1, la multiplication par **c** se fera en ***n - 1*** additions. Ainsi, dans notre exemple, le calcul se fait en 11 additions:

111463x = x<<16 + x<<15 + x<<13 + x<<12 + x<<9 + x<<8 + x<<6 + x<<5 + x<<2 + x<<1 + x

Une amélioration simple consiste à utiliser le recodage canonique de BOOTH de la constante.

Il s'agit d'un recodage en base 2, avec l'ensemble de chiffres $\{\overline{1};0;1\}$, où $\overline{1} = -1$, qui minimise le nombre de chiffres non nul.

Pour notre exemple, on a $111463 = 11011001101100111_2 = 100\overline{1}0\overline{1}0100\overline{1}0\overline{1}0100\overline{1}_2$ et le calcul se fait alors en 8 additions:

111463x = x<<17 - x<<14 - x<<12 + x<<10 - x<<7 - x<<5 + x<<3 – x.

VIII.3 L'algorithme de BERNSTEIN [10] :

Avec les algorithmes précédents, la variable d'entrée est décalée plusieurs fois (multipliée par des puissances de 2), puis les résultats de ces décalages sont tous additionnés.

Chaque valeur calculée ne sert donc qu'une seule fois. Un progrès serait donc de permettre la réutilisation de valeurs intermédiaires. Par exemple, pour calculer $q = 165x = 101001012$ x, il serait avantageux de calculer $z = 5x = 101_2$ x puis $q = 33z = 100001_2$ z, réduisant ainsi le nombre d'additions de 3 à 2.

L'algorithme de BERNSTEIN permet des recodages de ce type. Son principe est de rechercher, par une exploration d'arbre, le meilleur recodage d'une constante, en utilisant des opérations élémentaires.

Le coût de chacune de ces opérations peut être précisé, pour orienter les résultats en fonction de paramètres technologiques.

Les opérations permises sont les suivantes: .

- Décalage ti+1 = ti <<k;.
- Addition de la variable d'entrée ti+1 = ti * x;
- Addition d'un nombre avec lui-même après décalage ti+1 = ti <<k * ti.

Pour notre exemple p = c * x, l'algorithme de BERNSTEIN trouvera la solution suivante, en 5 additions:

t1 = ((x<<3 □ x)<<2) - x

t2 = t1<<7 + t1

p = (((t2<<2) + x)<<3) - x

---

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

Malheureusement, cet algorithme ne permet pas de partage de sous-expressions communes entre plusieurs résultats.

En outre, la solution construite reste très linéaire: une valeur calculée n'est utilisée que pour calculer la valeur suivante, ceci peut limiter la capacité de l'algorithme à trouver les meilleurs recodages, et ne permet pas d'utiliser le parallélisme inhérent aux implantations matérielles. En outre, la recherche du meilleur recodage est une opération très longue, probablement trop pour être applicable aux opérateurs linéaires.

VIII.4 L'algorithme de LEFÈVRE [11] :

L'algorithme de LEFÈVRE permet de trouver des recodages où une même valeur peut être réutilisée . En outre, cet algorithme est directement conçu pour la multiplication d'une variable par un jeu de constantes, en permettant des partages de sous-expressions communes entre les différentes constantes.

Avant d'exposer l'algorithme, quelques définitions préliminaires sont nécessaires. Un motif est une suite de chiffres $\{0,1,\overline{1}\}$. On dit qu'un motif apparaît dans une constante avec un décalage k si, en décalant le motif de k vers la gauche, on a un 1 dans la constante en face de chaque 1 du motif et un $\overline{1}$ en face de chaque $\overline{1}$.

On définit aussi le poids d'un motif comme le nombre de chiffres non nuls du motif. Par exemple, le motif $1000\overline{1}$ est de poids 2, et il apparaît deux fois dans la constante $84 = 1010\overline{1}0\overline{1}_2$ (avec des décalages de 0 et de 2). En outre, un motif peut apparaître dans une constante sous sa forme complémentée. Ainsi, le motif 101 apparaît aussi deux fois dans la constante $84 = 1010\overline{1}0\overline{1}_2$ (décalé de 4, et sous forme négative décalé de 0).

Le principe de l'algorithme de LEFÈVRE est de rechercher un motif commun, de poids maximal, dans les écritures des différentes constantes. ce motif doit apparaître plusieurs fois, soit à l'intérieur d'une même constante, soit dans des constantes différentes. On parle alors de motif maximal. Ce motif est alors soustrait des endroits où il apparaît, et rajouté comme nouvelle constante. L'algorithme recommence alors jusqu'à ce qu'il n'y ait plus de motif de poids supérieur ou égal à 2.

---

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

Sur notre exemple, en partant de la constante 111463 = 11011001101100111_2 = $100\bar{1}0\bar{1}0100\bar{1}0\bar{1}0100\bar{1}_2$, l'algorithme de LEFÈVRE trouve tout d'abord le motif $100\bar{1}0\bar{1}$, avec des décalages de 5 et 12. Après soustraction du motif, il reste les deux constantes 7 = $100\bar{1}_2$ et 26 = $100\bar{1}0\bar{1}_2$. L'algorithme trouve ensuite le motif commun $100\bar{1}$, et finit avec les deux constantes 7 = $100\bar{1}_2$ et 1 = 1_2. Ainsi, l'algorithme de LEFÈVRE trouve cette décomposition, en seulement 4 additions:

t1 = (x<<3) - (x<<0)

t2 = (t1<<2) - (x<<0)

p = (t2<<12) + (t2<<5) + (t1<<0)

VIII.5 CSD[9] :

L'efficacité de la méthode de Horner peut encore être améliorée en utilisant le canonique Signé chiffres (CSD) format pour représenter le multiplicateur ou diviseur.

Le format de la CSD vise à réduire le nombre d'opérations d'ajout lors de la multiplication et la division. Le format de la CSD a un ternaire définir par opposition à un ensemble binaire en nombre représentation. Les symboles utilisés dans ce format sont $\{0, 1, \bar{1}\}$, $\bar{1}$ est la représentant de -1.

L'objectif est de regrouper 1 consécutifs et les modifier pour une représentation ternaire de la représentation binaire. Ceci est fait à partir de la droite 1 et de procéder à gauche jusqu'à ce que le dernier 1.

En agissant ainsi, la représentation finale CSD n'a jamais 1s $\bar{1}$s ou adjacent. Cette représentation est efficace quand il ya beaucoup 1s adjacent au représentation binaire

Le format de la CSD peut être utilisé ici pour regrouper les 1 consécutifs. En partant du bit le plus à droite, le premier 1 a lieu à la position 2^{-12} .

Ce 1 n'a pas un côté 1 à sa gauche, et donc pas modifié. la prochaine 1 à la position 2^{-9}

a cinq 1s adjacente à la gauche de celui-ci. Ces six 1s sont combinés et 1 est placé à la position de bit le plus à droite (2^{-9} Zéros), au niveau des positions restantes (2^{-8} à 2^{-4}) Et un 1 à la position de bit 2^{-3} (un poste laissé à gauche 1 de la séquence originale). Ce processus est répété pour d'autres groupes de 1s que l'on rencontre.

La représentation CSD devient maintenant:

0,12345 = $0.00100000\bar{1}001$CSD

La méthode de Horner en utilisant le format CSD réduit le nombre d'opérations d'ajout au cours de multiplication.

Conclusion :

L'importance de la multiplication et sa sensibilité du rôle quelle va le jouer nous oblige de recherche toujours de ce qui est optimal et rapide et de moins couts .

Malgré que la performance de ces algorithme et méthodes vu a ce chapitre mais malheureusement sont des complexités très haute ce qui nous permette de perdus beaucoup des unité de temps

Mai heureusement d'âpres les recherche aujourd'hui on peut dire que on a eu un algorithme 'Radix-2^r' très performent que ces dernier et qui est moins couteux que ces derniers et ce algorithme on va le voir au prochaine chapitre.

Chapitre 03: Algorithme Radix-2^r

Chapitre 03 : Algorithme Radix-2^r

Introduction :

De nombreuses applications dans DSP et de contrôle, telles que le temps linéaire invariant (LTI) filtres / régulateurs, impliquent le calcul d'un grand nombre de multiplications d'une variable par un ensemble de constantes.

Pour être traitée efficacement, la mise en œuvre doit être sans multiplicateur, c'est, en utilisant exclusivement des ajouts, soustractions et décalages.

Ce problème est connu comme unique / multiple multiplication constante (SCM / MCM) et a eu de nombreux algorithme pour le résoudre comme a vue au chapitre précédent mai d'âpres les recherche qui ont fait en collaboration entre le membre des chercheure de centre algérien CDTA et institut France FEMTO-ST on réussit a nous offrir une nouvelle heuristique nécessitant une moyenne de 24,16% et 04,38% de moins que la norme ajouts canonique signé représentation chiffres (CSD) et le double fond Système de nombre (DBNS), respectivement, pour les coefficients de 64 bits .

A notre chapitre on va vous présente ce nouveau algorithme qui est nome Radix-2^r selon une version ancienne et qui modifier plusieurs fois âpres cette présentations.

I. Récapitulation sur les algorithmes existant :

Un grand nombreux d'heuristique proposer pour résoudre le problème qui est nome a unique/ multiple multiplication a une constante *(SCM/MCM)* a cela on classe ces heuristique on quatre catégories qui sont :

- Heuristiques Digit-recodage comme la CSD et DBNS .
- Élimination de sous-expression commune (CSE) en utilisant modèle correspondant. Les exemples sont Lefèvre.
- Des algorithmes Mise en scène graphique acyclique (DAG) comme Hcub H(k) , et MAG ;
- Algorithmes mixtes associant des CSE et DAG comme BIGE récente de l'algorithme optimal .

Malgré le grand nombre d'heuristiques proposées, la grande majorité des optimisations système LTI utiliser la CSD représentation de codage constant a cause de sa facilite mise en œuvre et ça complexité quelle est connu ce qui est diffère au autre heuristique

Dans la CSD le nombre de additionneurs est délimitée par *(N +1) /2 -1* et tend asymptotiquement à une valeur moyenne de *(N / 3) -8 / 9*, ce qui donne à 33% d'économiser plus de l'additionneur naïve et une approche d'équipe.

Pincez a été le premier à prouver que la multiplication par une constante est sous linéaire: O *(N / (log (n) α))* avec $\alpha < 1$. Basé sur DBNS arithmétique , Dimitrov a montré que la condition $\alpha < 1$ dans la complexité de pincement n'est pas nécessaire, ce qui réduit donc la limite supérieure de *O (N / (log (n)))*. Bien plus, en 2011, Dimitrov ont évalué la constante cachée dans la notation grand-O comme étant égal à 2. Depuis lors, *2.N/Log (N)* est considéré comme le plus analytique de la limite supérieure connue jusqu'ici. D'autre part, Gustafsson dans son algorithme optimal (MAG) pour SCM ont montré que 5 additionneurs suffisent à exprimer toutes les constantes jusqu'à 19 bits. String a fait mieux avec l'algorithme de BIGE exact, comme il a conjecturé (aucune preuve) que 7 ajouts sont assez jusqu'à 32 bits. Bien que MAG et BIGE garantie optimalité via un recherche exhaustive, ils nécessitent un temps d'exécution et le stockage exponentielle par rapport à N . Cela les rend peu pratique pour des valeurs élevées de N, au moins pour la puissance de calcul actuelle. Néanmoins, avec BIGE nous pouvons observer combien tout heuristique est loin d'optimalité jusqu'à 32 bits.

II. Recodage Radix-2^r :

II.1. Présentations D'algorithme :

Une constante C de N bits est exprimé en Radix-2^R comme suit:

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

$$=\sum_{j=0}^{(\frac{N}{r})-1} Q_j *2^{rj} \qquad (1)$$

où $c_{-1} = 0$ et $r \in N^*$. Par souci de simplicité et sans perte de généralité, nous supposons que r est un diviseur de N.

En équation **(1)**, la représentation de deux de complément de constante *C* est divisé en *N*/r complément tranches de deux (*Qj*), chacun des ***R +1*** longueur de bits. Chaque paire de deux tranches contiguës a un peu de chevauchement. Le chevauchement n'a aucun effet sur le complément à deux notations depuis:

$$-2^{r-1}c_{rj+r-1} * 2^{rj} + c_{rj+r-1} * 2^{r(j+1)} = c_{rj+r-1}*2^{rj+r-1}$$

Pour équation (1) correspond un ensemble chiffres DS (2^r) de telle sorte que $Q_j \in (2^r)= \{-2^{r-1},-2^{r-1}+1,\ldots..,\ 1,\ 0,\ 1,\ldots\ldots..,\ 2^{r-1}-1\ ,2^r-1\}$

ainsi, le produit *C* de *X* devient: $$C *X=\sum_{j=0}^{(\frac{N}{r})-1} X * Q_j *2^{rj} \qquad (2)$$

Le signe du terme *Q*j est donnée par C_{rj+r-1} bit, et $|Qj| =2^{Kj} *m_j$ avec $kj \in \{0,1,2, \ldots, r-1\}$ et $m_j \in OM(2^r) = \{1,3, 5,\ldots, 2^{r-1}-1\} \cup \{0\}$.

$OM(2^r)$ est l'ensemble requis de multiples impairs dans le recodage radix-2^r ,avec $OM(2^r) = 2^{r-2}$. Dans le cas où $Qj = 0$, $m_j = 0$.

Enfin, le produit peut être exprimée comme suit:

$$C*X=\sum_{j=0}^{(\frac{N}{r})-1} (-1)^{crj+r-1}*(m_j*X)*2^{rj+kj} \qquad (3)$$

Contrairement à la multiplication par une variable (Y × X) où l'ensemble définir des impair-multiples (mj × X) doit être pré calculée, dans le cas la multiplication par une constante (C × X) seulement un sous-ensemble est nécessaire.

En fait, le nombre des multiples impairs est égal au nombre de différentes valeurs m_j induites par le procédé de codage des N / R tranches (termes Qj). Par conséquent, la génération de produits partiels (PP) se compose d'abord, si mj ≠ 0, dans le calcul du impair multiple mj × X s'il n'a pas été pré calculées avant, qui est ensuit soumis à un changement câblé de positions rj + kj, et enfin, conditionnellement complémentée (-1) $c_{rj\ r-1}$ en fonction du signe bit C_{rj+r-1} de Qj.

II.2 Nombre maximal des additionneurs pour une constante de N bit :

En équation . (3), il ya **N / r** itérations. Chaque itération génère un PP. Ainsi, le nombre maximal de PP est **N / r**, exige un maximum de la $\mathbf{N_{pp} = N/r-1}$ ajouts.

Un maximum de $|OM(2^r)| - 1 = 2^{r-2} - 1$ impairs multiples

$\{3 \times X,\ 5 \times X, 7 \times X, ..., (2^{r-1}-1) \times X\}$

peuvent être invoqués, pendant le processus de génération de PP, ce qui peut être simplement construit un après l'autre, à chaque fois en utilisant un Outre unique entre un élément qui a déjà été construits et une puissance de deux.

Étant donné que chaque élément a besoin exactement un plus, cela donne $2^{r-2}-1$ additions.

Cette naïf méthode conduit à un additionneur profondeur importante (2^{r-2}-1 de cascade Additionneur) qui a été démontré qu'elle affecte fortement le pouvoir consommation, car le nombre de pépins augmente avec augmentation additionneur approfondie.

Afin de minimiser l'addition approfondie sans augmenter le coût additionneur ($2^{r-2}-1$), nous annonçons le théorème suivant:

Théorème 1.

Dans Radix-2^r, le pré calcul de l'ensemble des impairs multiples $\{3 \times X, 5 \times X, 7 \times X, ..., (2^{r-1}-1) \times X\}$ nécessite un additionneur-coût et un additionneur-profondeur de $2^{r-2}-1$ et r-2, respectivement.

Preuve : Nous exploitons le fait que $OM(2^r)$ peut être intégré dans tout seule étape en utilisant uniquement (2^{r-1}), comme indiqué ci-après :

$$OM(2^r) = \{1,3,5, ..., 2^{r-1}-1\}$$
$$= \{1,3,5, ..., 2^{r-2}-1, 2^{r-2}+1, 2^{r-2}+3, 2^{r-2}+5, ..., 2^{r-1}-1\}$$
$$= \{1,3,5, ..., 2^{r-2}-1\} \cup \{2^{r-2}+1, 2+3, 2^{r-2}+5^{r-2}, ..., 2^{r-1}-1\}$$
$$= OM(2^{r-1}) \cup \{2^{r-2}+1, 2^{r-2}+3, 2^{r-2}+5, ..., 2^{r-1}-1\}$$
$$= OM(2^{r-1}) \cup \{2^{r-2}+1, 2^{r-2}+3, 2^{r-2}+5, ..., 2^{r-2}+(2^{r-2}-1)\}$$

$= OM(2^{r-1}) \cup S_{r-2}$

$= OM(2^{r-1}) \cup S_{r-3} \cup S_{r-2}$

$= OM(2^{r-1}) \cup S_1 \cup S_2 \cup \ldots\ldots \cup Sr_{-3} \cup S_{r-2}$

Avec $|OM(2^{r-1})| = |S_{r-2}| = 1/2 \ |OM(2^r)|$ et $|S_{i-1}| = 1/2 \ |S_i|$

Ainsi, le nombre total des ajouts nécessaires est égal à:

$N_{om} = |S_1| + |S_2| + \ldots\ldots\ldots\ldots + |S_{r-3}| + |S_{r-2}|.$

$$= \frac{1}{2^{r-2}} 2^{r-2} + \frac{1}{2^{r-3}} 2^{r-2} + \ldots\ldots\ldots + \frac{1}{2^2} 2^{r-2} + \frac{1}{2} + 2^{r-2}$$

$$= 2^0 + 2^1 + \ldots\ldots\ldots + 2^{r-4} + 2^{r-3}$$

$$= 2^{r-2} - 1$$

En ce qui concerne le nombre maximum des étapes, des ensembles de Si sont effectuées séquentiellement, à partir de S_1 à S_{r-2}, ce qui nécessite r-2 étapes .

Nous illustrons le concept avec l'exemple radix -2^6 suivants:

$OM(2^6) = \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\}$

$= \{1\} \cup \{2^1 + 1 = 3\} \cup \{2^2 + 1 = 5, 2^2 + 3 = 7\} \cup \{\cup 2^3 + 1 = 9, 2^3 + 3 = 11,$
$23 + 5 = 13, 23 + 7 = 15\} \cup \{2^4 + 1 = 17, 2^4 + 3 = 19, 2^4 + 5 = 21,$
$2^4 + 7 = 23, 2^4 + 9 = 25, 2^4 + 11 = 27, 2^4 + 13 = 29, 2^4 + 15 = 31\}$.

Ainsi, les multiples impairs ($m_j \times X$) correspondant à l'OM (2^6) sont ensuite calculée dans l'ordre suivant (6-2 = 4 étapes) :

S1 = {3 × X} ;

S2 = {5 × X , 7 × X} ;

S3 = { 9 × X , 11 × X , 13 × X , 15 × X} ;

S4 = { 17 × X , 19 × X , 21 × X , 23 × X , 25 × X , 27 × X , 29 × X , 31 × X} .

A la figure 10 on va fournit tous les détails nécessaires au matériel mise en œuvre. En conséquence, le nombre total des ajouts requis par radix-2 r est égale à: En conséquence, le nombre total des ajouts requis par Radix - 2^r est égale à :

Pour Radix-2^6, un maximum de 2^6-2-1 = 15 ajouts sont nécessaires, menées en 6-2 = 4 étapes dans le pire des cas.

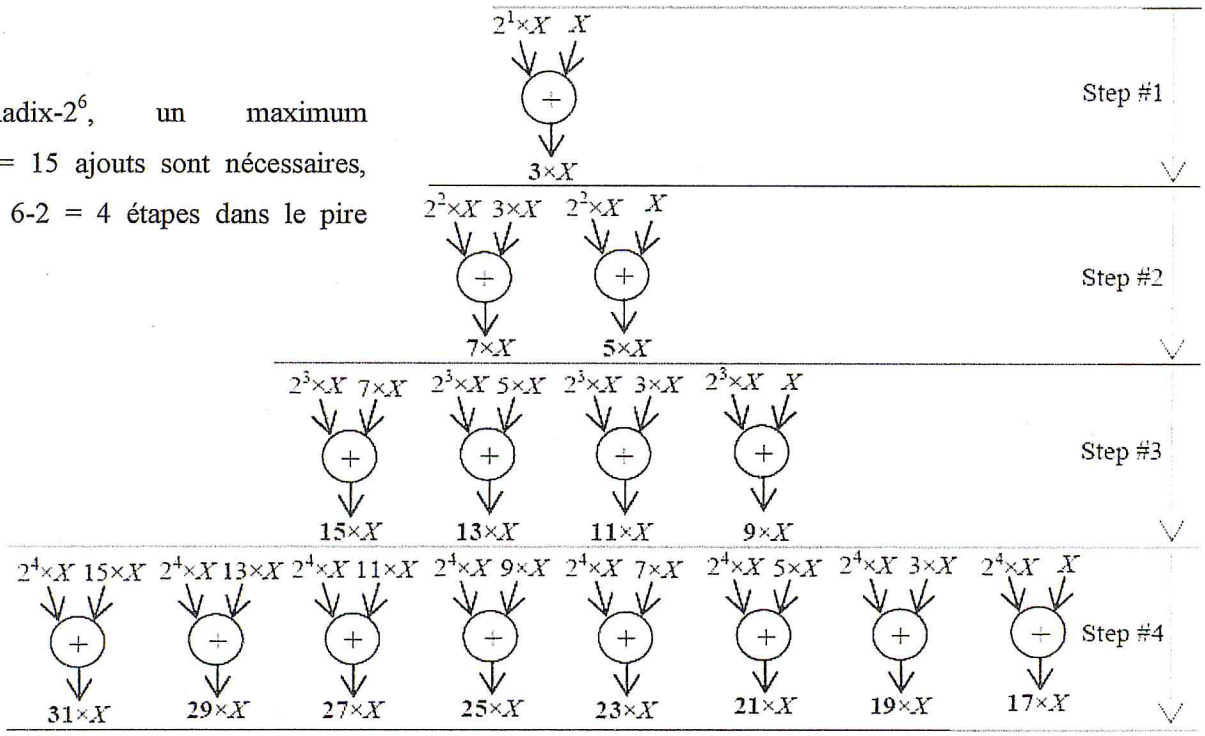


Figure 12 : Ordre séquentiel de calcul de l'ensemble des multiples impairs nécessaires par radix-2^6

$$U_{pb}(r)=N_{pp}+N_{om}=\left\lceil \frac{N}{r}+2^{r-2}-2 \right\rceil$$

Où $\lceil\ \rceil$ est la fonction de plafond (ex $\lceil 5.29 \rceil = 6$).

U_{pb} (r) est minime pour $r=2.w(\sqrt{N.\log(2)}\ /\log(2)$ où W est la fonction de Lambert .

Le tableau 02 donne les valeurs de r qui conduisent à nombre minimum d'additions pour N de 8 à 8192, tandis que Figure.11 représente le coin supérieur limites en nombre d'ajouts pour la CSD , DBNS et RADIX -2^r .

N	8	16	32	64	128	256	512	1024	2048	4096	8192
R	4	4	4	4	5	6	6	7	8	8	9
$U_{pb}(r)$	3	6	10	18	32	57	100	177	318	574	1037

Tableau 02 : Valeurs Limite Supérieure Et r Pour Le N-Bit Constante A L'aide Radix-2^r

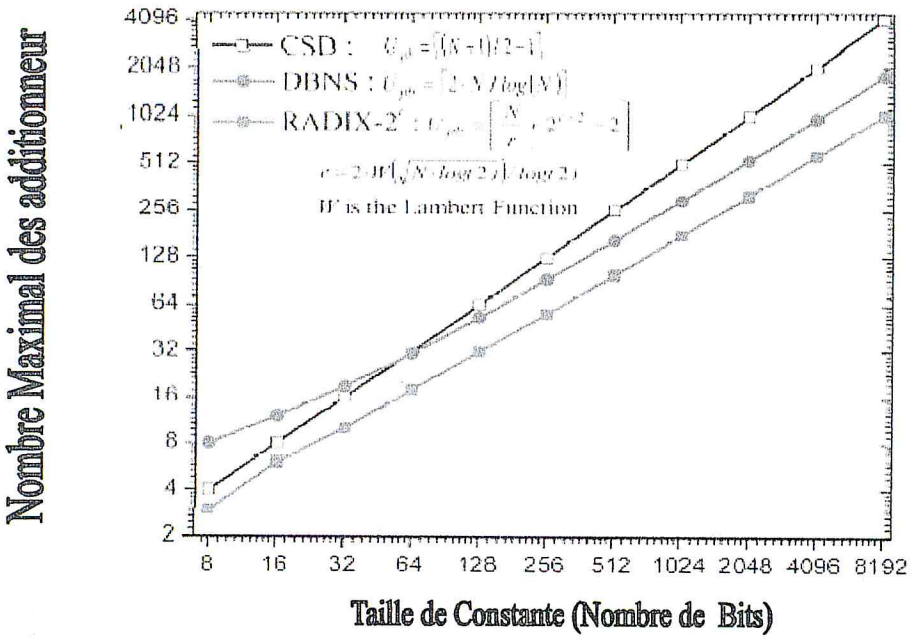


Figure 13 : Comparaison du U_{pb} selon la taille de constante .

II.3. La Moyenne D'additionneur AVG :

En ce qui concerne le nombre moyen d' additions , il a été limitativement calculé pour les 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 en utilisant 10^5, 10^6, 10^9 et 10^{10} uniformément répartie les valeurs de C aléatoires .

Bien que la différence entre les quatre les résultats obtenus sont négligeables (< 10-3) , le moyen diminue que le nombre de valeurs C augmente , et converge à 15,5037 additions . Les résultats sont présentés dans le tableau 03. Pour N = 64 , RADIX - 2^r utilise 24.17 % de moins que les additions CSD. L'économie semble croître de façon linéaire pour les faibles valeurs de N.

Langueur de constante N bit	CSD		Radix-2^r		Econome de Moyenne(%)
	Moyenne	U_{pb}	Moyenne	U_{pb}	
8	1.788	4	1.784	3	0.218
16	4.444	8	4.251	6	4.3356
24	7.111	12	6.531	8	8.152
32	9.777	16	8.685	10	11.170
64	20.444	32	15.503	18	24.166

Tableau 03: RADIX-2^r VERSUS CSD: Nombre moyen d'ajouts et Upb Et supérieure (U_{pb}).

En ce qui concerne DBNS, Dimitrov calculé moyenne et U_{pb} à partir de 10^5 répartis uniformément constantes aléatoires, pour 32 et 64 bits seulement (tableau 04). Notez que DBNS U_{pb} sera plus élevé si le pire des cas ne sont pas atteints par le modèle de 105 constantes.

Langueur de constante N bit	DBNS		Radix-2^r		Econome de Moyenne(%)
	Moyenne	U_{pb}	Moyenne	U_{pb}	
32	≈9.05+*	13*	8.6855	10	4.0276
64	16.2151*	21*	15.5037	18	4.3872

Tableau 04 : RADIX-2^r VERSUS DBNS: Nombre moyen d'ajouts (Moy) Et supérieure (Upb).

Nous avons également comparé RADIX-2^r à certains non-recodage heuristiques (CAS et DAG) basé sur des programmes et numérique

Données fournies gracieusement par Lefèvre et Voronenko. Tandis que la figure. 14 indique les valeurs Moyenne inférieurs pour heuristiques non recodage que s'attendre en raison d'une plus grande exploration de l'espace de solution, Le tableau 05 présente des valeurs de l'UPB plutôt élevés. Significative conclusion: une moyenne inférieure ne garantit pas un Upb inférieur.

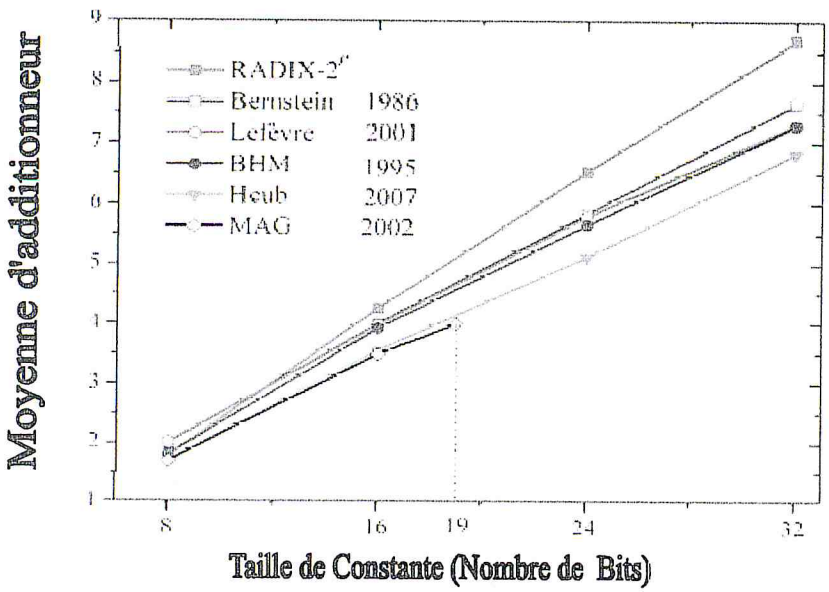


Figure 14 : Comparaison du Moyenne d'additionneur selon la taille de constante .

Algorithme	(84AB5)H N=20	(64AB55)H N=24	(5959595B)H N=32	Temps D'exécution
BENSTEIN	8^G	7	8	$O(2^N)$
Hcub*	6	9^G	_	$O(N^6)$
BHM*	5	7	_	$O(N^4)$
Lefèvre	4	6	11^G	$O(N^3)$
Radix-2r	5	7	10	O(N)

Tableau 05 : RADIX-2^r par rapport aux non-RECODAGE Algorithme: DUREE Complexité et le nombre d'ajouts de certains cas particuliers

N: Constant bit-taille;

G: Supérieur à RADIX-2^r Upb;

RADIX-2^r Upb = 7, 8, et 10 pour N = 20, 24 et 32, respectivement;

Un autre indicateur de performance de l'enregistrement est le plus petite valeur qui nécessite q additions, pour q variant de 1 à la limite supérieure de l'enregistrement.

Le tableau 6 résume ça ces informations sur 32 bits constant.

Notez qu'à partir de q = 7, des valeurs plus élevées sont donnés par RADIX-2^r par rapport à la CSD.

Nombre d'addition(q)	CSD	Radix-2^r	Lefevre*
1	3	3	3
2	11	11	11
3	43	43	43
4	171	139	213
5	683	651	1703
6	2731	2699	13623
7	10923	33491	174903
8	43691	526491	1420471
9	17473	8422027	13479381
10	699051	134744219	-
11	2796203	-	-
12	11184811	-	-
13	44739243	-	-
14	178956971	-	-
15	715827883	-	-

Tableau 06: RADIX-2r VERSUS CSD ET LEFEVRE: PETIT Valeurs jusqu'à ce que constante de 32 bits

III. Déroulement d'un exemple en algorithme Radix-2^r :

Le produit 10599 × X est d'abord calculé en CSD , DBNS , et RADIX -2^r .

Notez que $(10599)_{10} = (10100101100111)_2$.

$P_{CSD} = (X \times 213) + (X \times 211) + (X \times 29) - (X \times 27) - (X \times 25) + (X \times 23) - X$.

$P_{DBNS} = ((X1 \times 21) + X1) + (X \times 213) + (X \times 23) - X$

avec $X1 = ((X0 \times 21) + X0) + (X \times 25)$ et $X0 = (X \times 28)$.

Pour exprimer le produit dans PRADIX , un complément à deux représentation de (10599) $_{10}$ est nécessaire , ce qui est $(010100101100111)_2$. Ainsi, en notation complément à deux, la taille constante est égal à N = 15 , ce qui correspond à la valeur R égal à 4 (tableau 2) . Comme 15 n'est pas un multiple de quatre , le bit de signe (0 dans ce cas) est prolongé par une position de sorte que N = 16 .

Pour C = 10599 , equations . (1) et (3) deviennent respectivement :

$$P_{Radix}=\sum_{j=0}^{3} Q_j \times 2^{4j} \quad , P_{Radix}=\sum_{j=0}^{3}(-1)^{c4j+3} \times (mj \times X) \times 2^{4j+kj}$$

La détermination des quatre termes Qj est décrite dans Figure .4 . Pour déterminer les valeurs inconnu c4j +3 , MJ, et kj , le radix -2^4 table de consultation (tableau 07) est indexée par des termes Qj .

Se référant au tableau 07, les triplets ($c4_j$ +3 , m_j , k_j) correspondant à Q_0 , Q_1, Q_2 et Q_3 sont (0,7,0) , (0,3,1) , (1,7,0) et (0,3,0) , respectivement . Le recodage de C = 10599 nécessite le près computation de deux impairs multiples uniquement {3 × X , 7 × X} , tandis que radix -2^4 recodage peut invoquer un ensemble de trois éléments au maximum {3 × X , 5 × X , 7 × X} .

Q_j					mj	Kj
C4j+3	C4j+2	C4j+1	C4j	C4j-1		
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	1	0	1	0
0	0	0	1	1	1	1
0	0	1	0	0	1	1
0	0	1	0	1	3	0
0	0	1	1	0	3	0
0	0	1	1	1	1	2
0	1	0	0	0	1	2
0	1	0	0	1	5	0
0	1	0	1	0	5	0
0	1	0	1	1	3	1
0	1	1	0	0	3	1
0	1	1	0	1	7	0
0	1	1	1	0	7	0
0	1	1	1	1	1	3
1	0	0	0	0	1	3
1	0	0	0	1	7	0
1	0	0	1	0	7	0
1	0	0	1	1	3	1
1	0	1	0	0	3	1
1	0	1	0	1	5	0
1	0	1	1	0	5	0
1	0	1	1	1	1	2
1	1	0	0	0	1	2
1	1	0	0	1	3	0
1	1	0	1	0	3	0
1	1	0	1	1	1	1
1	1	1	0	0	1	1
1	1	1	0	1	1	0
1	1	1	1	0	1	0
1	1	1	1	1	0	0

Tableau 07 : RADIX-2r TABLE DE CONSULTATION

Par conséquent , nous pouvons écrire :

$PRADIX = ((3 \times X) \times 2^{12}) - ((7 \times X) \times 2^8) + ((3 \times X) \times 2^5) + (7 \times X)$.

$= (X_0 \times 2^{12}) - (X_1 \times 2^8) + (X_0 \times 2^5) + X_1$

avec $X0 = (X \times 2) + X$ et $X_1 = (X \times 22) + X_0$.

Notez que pour C = 10599 , P_{CSD} et P_{DBNS} nécessitent à la fois 6 additions , tandis que P_{RADIX} nécessite 5.

Le naïf décalage et d'addition approche aurait fallu 7 additions . Nous supposons que addition et la soustraction ont la même aire / vitesse coûts, et ce changement ne coûte rien , car il peut être réalisé sans portes en utilisant le câblage dur.

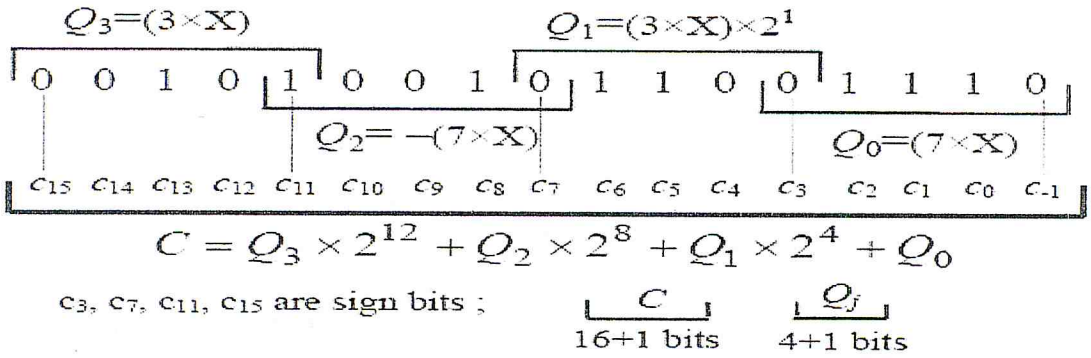


Figure 15 : Partitionnement de $(10599)_{10}$ en Radix-2^4

Simplifications intérieur équation. (3) sont possibles dans le cas où deux Q_j et Q_{j+1} mandats consécutifs de signes opposés présentent (m_j , k_j) des couples de la forme (1, r - 1) et (1,0) , respectivement . cette est illustré par les deux possibilités suivantes :

$\ldots\ldots\ldots\ldots + X * 2^{r(j+1)} - X * 2^{rj+(r-1)} \pm \ldots\ldots = \ldots\ldots\ldots + X * 2^{rj+(r-1)} \pm \ldots\ldots\ldots\ldots$

$\ldots\ldots\ldots\ldots - X * 2^{r(j+1)} + X * 2^{rj+(r-1)} \pm \ldots\ldots\ldots = \ldots\ldots\ldots - X * 2^{rj+(r-1)} \pm \ldots\ldots\ldots\ldots$

Une autre idée intéressante est d'inclure une redondance dans m_j termes d'égalisation. (3) . Ces deux tours vont diminuer la moyenne nombre d'additions en base - 2^r (Tableau 3, 4, et la Fig 14) .

En plus des capacités de compression plus élevés de Radix -2^r par rapport à la CSD et DBNS , sa complexité d'exécution est linéairement proportionnelle à N comme indiqué par l'équation (1). , et le nécessaire espace mémoire est très petite (pour 8192 correspond constants bits une table de consultation de $2^{9+1} = 1024$ entrées) .

Ces deux caractéristiques font RADIX - 2^r très utile pour de grosses constantes (<< N).

Depuis l'introduction de H (k) En 2004, le CST heuristiques ont surperformé DAG au SMC . Ceci a été réalisé par l'application du CST à chaque signé chiffres forme possible

(SD) de la constante . De même, l'espace de recherche du CST peut être étendu considérant RADIX - 2^r recodage lieu représentation SD.

Pour un tel objectif (SCM / MCM) , Lefèvre heuristique se présente comme le meilleur candidat CSE pour sa complexité computationnelle inférieur O (N^3) par rapport à ses homologues du CST .

De nombreuses techniques de conversion de unsigned ou deux compléter le numéro de sa forme CSD sont proposées pour réduire la complexité du matériel et d'augmenter la vitesse de la variable multiplicateurs.

Basé sur RADIX -2^r , nous avons proposé plusieurs techniques de conversion et déterminé le plus efficace.

Conclusion :

Sur la base de calcul radix-2^r, nous avons développé un nouveau recodage temps linéaire (base-2^r) accompagné de la borne superieure complexité. Ce dernier est le plus bas de la limite supérieure connu à ce jour pour la multiplication par une constante.

Non seulement RADIX-2^r réalise un meilleur taux de compression que DBNS et CSD, mais réduit également la vipère approfondie qui donne moins de puissance et plus de vitesse. plus loin des améliorations sont possibles en utilisant la redondance dans RADIX-2^r recodage.

Expressions analytiques exactes de l additionneur maximale ainsi que le nombre moyen d'ajouts requis par RADIX-2^r sont encore à déterminer.

Chapitre 04:
Implimentation de l'algorithme Radix-2^r

Chapitre 04 : Implémentation de l'algorithme Radix-2r

Introduction :

Comme on a déjà vu au chapitres précédents que le problème de multiplication à constante a eu plusieurs solution mais sont pas performante et sont couteuse car elle prend beaucoup d'unité de temps et sont très complique (ça complexité et très élevé) par contre l'algorithme de Radix-2^r a minimise cette complexité il a rendu linéaire comme a nous aide a gagner plusieurs unités de temps puis ce que il est plus rapide au rendement des résultats si est programme de manier correcte et optimal .

A ce chapitre on vous presente notre programme qu'on le fait pour le développement de algorithme Radix-2.

I. Langage Utiliser :

C est un langage de programmation impératif et généraliste. comme il est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour être compilée en un nombre d'instructions machine assez prévisible en termes d'occupation mémoire et de charge de calcul.

Il propose un éventail de types entiers et flottants conçus pour pouvoir correspondre directement aux types supportés par le processeur. Il fait en outre un usage intensif de la notion de pointeur. Il a une notion de type composé, mais ne propose aucune opération qui traite directement des objets de plus haut niveau (fichier informatique, chaîne de caractères, liste…), Ces types plus évolués doivent être traités en manipulant des pointeurs et des types composés.

De même, le langage ne propose pas en standard la gestion de la programmation orientée objet, ni de système de gestion d'exceptions. Il existe des fonctions standards pour gérer les entrées-sorties et les chaînes de caractères, mais contrairement à d'autres langages, aucun opérateur spécifique pour améliorer l'ergonomie.

Ceci rend aisé le remplacement des fonctions standards par des fonctions spécifiquement conçues pour un programme donné.

Ces caractéristiques en font un langage privilégié quand on cherche à maîtriser les ressources utilisées, le langage machine généré par les compilateurs étant relativement prévisible et parfois même optimal sur les machines d'architecture RISC à grand nombre de registres.

Ce langage est donc extrêmement utilisé dans des domaines comme la programmation embarquée sur microcontrôleurs, les calculs intensifs, l'écriture de systèmes d'exploitation et tous les modules où la rapidité de traitement est importante.

Il constitue une bonne alternative au langage d'assemblage dans ces domaines, avec les avantages d'une syntaxe plus expressive et de la portabilité du code source. Le langage C a été inventé pour écrire le système d'exploitation UNIX, et reste utilisé pour la programmation système. Ainsi le noyau de grands systèmes d'exploitation comme Windows et Linux sont développés en grande partie en C.

En contrepartie, la mise au point de programmes en C, surtout s'ils utilisent des structures de données complexes, est plus difficile qu'avec des langages de plus haut niveau.

En effet, dans un souci de performance, le langage C impose à l'utilisateur de programmer certains traitements (libération de la mémoire, vérification de la validité des indices sur les tableaux…) qui sont pris en charge automatiquement dans les langages de haut niveau.

Dépouillé des commodités apportées par sa bibliothèque standard, C est un langage simple, et son compilateur l'est également. Cela se ressent au niveau du temps de développement d'un compilateur C pour une nouvelle architecture de processeur : Kernighan et Ritchie estimaient qu'il pouvait être développé en deux mois car « on s'apercevra que les 80 % du code d'un nouveau compilateur sont identiques à ceux des codes des autres compilateurs existant déjà. »[1].

II. Résume de L'Algorithme Radix-2^r :

Comme on vue Algorithme de Radix-2^r au chapitre précédent on a le prend et on a le partage en quelque partie pour facilite sont développement comme on a le résume comme suite :

L'algorithme radix-2^r est un des algorithme qui oriente a minimise la complexité de résolution d'un problème de multiplication alors il est base a utilisation des constante et des variable quelle prend la forme binaire automatiquement puis ce que il est exécuté sur machine après cette conversion l'utilisateur de ce algorithme il va récupère la taille ou il va représente sa forme binaire de sa variable du tableau 02 sans oublions de rajoute un 0au bits de position c_{-1} après sa on récupère la valeur de nombre bits qu'on va les sélectionne a chaque groupe (Nombre de Bit Sélectionner= r (du tableau02)+1) sont oublions que a chaque groupe on chevauche d'un bit entre chaque deux groupe en fur a mesure de sélection du groupe on prend chaque groupe et on le compare avec la table de consultation quelle est mentionne déjà au tableau 07 pour qu'on récupère les K_j et M_j pour qu'on écrire les Q_j tel que $Q_j = M_j X^* 2^{K_j}$. a la fin on récrire la forme de la variable C

sous la nouvelle forme telle que $C = \sum_{j=0}^{(\frac{N}{r})-1} Q_j * 2^{rj}$ quand on arrive a ce point d'avancement on calcule le nombre de additionneur existant.

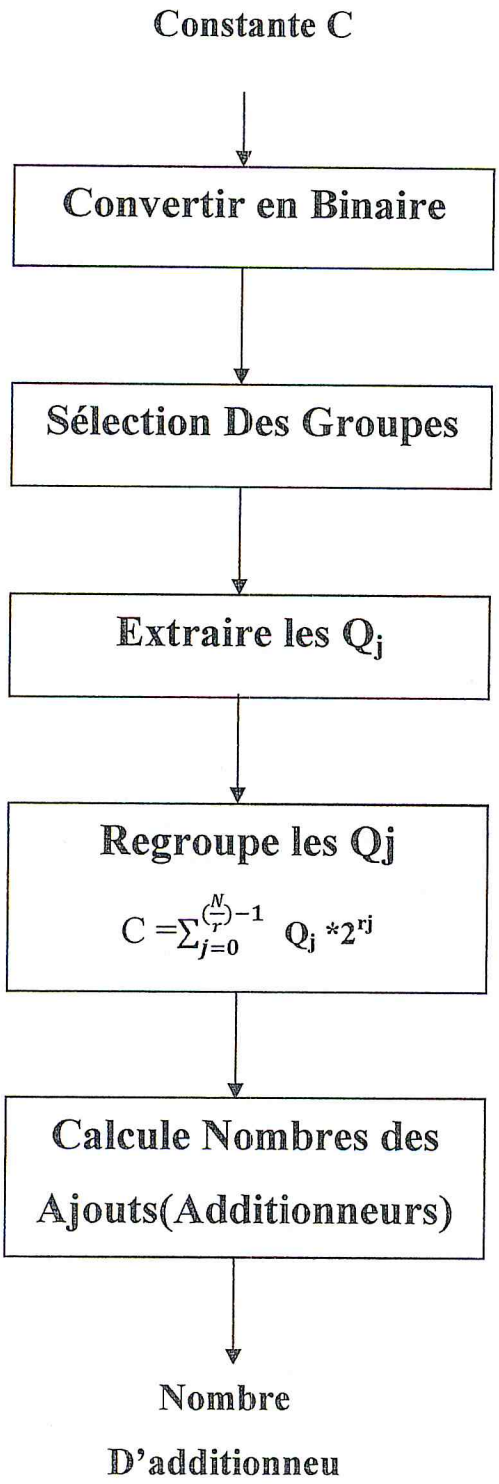


Figure 16 : Graphe représentatif de algorithmes Radix-2r

III. Implémentation de l'Algorithme Radix-2^r :

Pour le développement d'algorithme Radix-2^r on a choisi le langage C++ puis ce qu'il est le plus utilisable a ce domaine de développement.

A notre développement on a le partage on étapes telle que chaque étape et totalement indépendante a autre pour évite tout les obstacles et les sous problèmes.

III.1 Convertion Binaire :

La conversion en binaire a comme objectif de prendre une constante et la récrire a la forme d'une chaine des 0 et des 1

Par exemple $(10599)_{10}=(10100101100111)_2$

A notre programme on a partage cette étape en deux sous étapes : la première est de extraire (Calcule)la taille de la variable et la deuxième de fait les division a 2jusqu'au arrive a un contions=0 e voici le code source qui apartien a cette partie

```
// 01- Extraire La Tailles en Bits.
unsigned long int Constante=0;
unsigned long int ConstanteG=0;
int Taille;

printf("                -Partie 01: Convertir en Binaire.\n");

printf(" Introduit La Valeure Que Vous Voullez La Traite :");
scanf("%d",&Constante);
ConstanteG=Constante;
Taille=TaillesBits(ConstanteG);
printf("\n La Taille de Cette Variable [ %d ] est  %d Bits.\n",Constante,Taille);

// 02- Recupaire L'ecriture Binaire.
int Binaire[Taille];
int TabBin=0;

printf("\n L'Ecriture Binaire de La Valeur %d Est :\n  ",Constante);
printf("\n La lecture -->\n  ");

while (TabBin<Taille)
{// Dubet de Boucle While
```

```
Binaire[TabBin]=Reste(ConstanteG);
ConstanteG= Contion(ConstanteG);
printf("%d ",Binaire[TabBin]);
TabBin++;
}// Fin de Boucle While.

ConstanteG=Constante;
```

III.2 Sélection du groupe :

Cette partie et le cœur de programme puis ce que a cette partie on extraire les groupes et qu'on dit extraire du groupe alors on parle sur les **Qj,** a cause de sa importance on a la traite prudemment et on la partage en deux grande étapes quelle on elle-même partage en sous étapes quelle sont (écriture sur la nouvelle taille et la sélection des groupe).

III.2.a) Extraction de la nouvelle taille :

A ce niveau de programme on va extraire la nouvelle taille de la variable selon le tableau des taille qu' il est déjà défini au tableau 02 on parallèle on extraire le nombre de groupe et le pas de groupe (Nombre des bits a chaque groupe) qui conviens a la nouvelle taille est ça toujours du tableau 02.

```
int TailEquivalante=Taille;
int PasGroupement=0;
int NombreGroupement=0;

printf("              -Partie 02: Nouvelle Taille.\n");
// 01- Extraire La Nouvelle Tailles en Bits.
TailEquivalante=ExtraireTaille (Taille);
// 02- Extraire Le Nompre de Groupe.
PasGroupement=ExtaireNombreGroupe(Taille);
// 03- Extraire Le Pas de Groupe.
NombreGroupement=TailEquivalante/(PasGroupement-1);

printf(" La Nouvelle Taille est :%d  \n Le Pas de Groupement : %d  \n Le Nombre de Groupe: %d
",TailEquivalante-1,PasGroupement,NombreGroupement);
```

III.2.b) Ecriture selon la nouvelle taille :

A l'écriture de résultat binaire en nouvelle taille est de prend la chaine et l'affecte dans un tableau de tailles N+1 avec la case de indice 0 toujours égal a 0 et on commence affectation du case de indice1.

```
int TabEquival[TailEquivalante];
 int TabEqui=1;
 TabBin=0;
 ConstanteG=Constante;
 // 01- Ajout du bit (-1)
 TabEquival[0]=0;

 printf("                -Partie 03: Nouvelle Ecriture Binaire.\n");
// 02- Convertion Binaire sur une nouvelle taille
while(TabEqui<TailEquivalante)
{// Dubet de La boucle While.
 TabEquival[TabEqui]=Reste(ConstanteG);
 ConstanteG= Contion(ConstanteG);
 TabEqui++;
 }// Fin de La boucle While.

 printf(" La Nouvelle Ecriture de La Constante  %d  Est :\n ",Constante);
TabEqui=0;

while( TabEqui<TailEquivalante)
{//Dubet de La boucle while
 printf(" %d ",TabEquival[TabEqui]);
 TabEqui++;
}//Fin de La boucle while
 ConstanteG=Constante;
```

III.2.c) La sélection et extraire du groupe:

La sélection du groups c'est l'étapes la plus sensible et délecte a cette étapes on sélection chaque fois r +1 bit avec un chevauchement d'un bit .

```
int NbrGroupe=0;
int PosArretAnc=0;
int PosDemarre=0;
int PosArret=1;
int VarPos=0;
int TabGroupe[TailEquivalante+NombreGroupement-1];
int demar=PasGroupement;
int i=0;
int cpt=1;

printf("                -Partie 04: Les Groupes Selectionne .
printf("\n  Les Groupes Selectioner de Ecriture Binaire :\n");

while (NbrGroupe<=NombreGroupement)
{//Dubet de la boucle while
if (NbrGroupe==0)
{//Dubet de if
  PosArretAnc=PasGroupement-1;
 PosDemarre=0;
 PosArret=PosDemarre+(PasGroupement-1);
 for(VarPos=PosDemarre; VarPos<=PosArret; VarPos++)
 {//Debut de la boucle for
   TabGroupe[VarPos]=TabEquival[VarPos];
   printf(" %d ",TabGroupe[VarPos]);
   }//Fin   de la boucle for

 NbrGroupe++;
}//fin   de if
 else if (NbrGroupe!=0)
{//Debut de else

PosArretAnc=PosArret;
PosDemarre=PosArretAnc ;
PosArret=PosDemarre+ (PasGroupement-1);
 for(VarPos=PosDemarre; VarPos<=PosArret; VarPos++)
{//Dubet de la boucle for
TabGroupe[VarPos+NbrGroupe]=TabEquival[VarPos];
printf(" %d ", TabGroupe[VarPos+NbrGroupe]);
}//fin  de la boucle for
```

```
}//fin  de else
NbrGroupe++;
printf("\n");
}//fin   de la boucle while

while (cpt<NombreGroupement)
{// Dubet de la boucle While.
for(i=cpt*demar; i<cpt*demar+(PasGroupement-1)+cpt; i++)
{// Dubet de la boucle for.

TabGroupe[i]=TabGroupe[i+1];
}// fin de la boucle for.
cpt++;
}// fin de la boucle for.

printf("\n");

for(VarPos=0; VarPos<TailEquivalante+PasGroupement-2; VarPos++)
{//Dubet de la boucle for
 if (TabGroupe[VarPos]!=1) TabGroupe[VarPos]=0;
}// FIn de la boucle for

printf("\n  Le Tableau 'TabGroupe Contien' Tout Les Groupes :\n");
VarPos=0;


while (VarPos<TailEquivalante+PasGroupement-2)
{//Dubet de la boucle for
printf(" %d ", TabGroupe[VarPos]);
VarPos++;
}//Fin  de la boucle for

printf("\n\n----------\n");
```

III.2.d) Extraction de Qj :

$$Qj=M_j X * 2^{Kj}$$

Extraire de Qj est Extraire de Kj et Mj et les sauvegarde dans un tableau, telle que extraire de Kj et Mj se fait par la recherche dans la table de consultation quelle est définis au tableau 07 , on prend chaque groupe sélectionne et compare la chaine des bits de groupe avec la chaines des bits de la table de consultation et extraire le kj et Mj équivalent a cette dernier

```
int ColTab=0,LignTab=0;
int TabGrouN[NombreGroupement+1];
int VarVer=0;
int Demare=0;
int Arret=-1;
int Temp=0;
int puis=1;
int Cos=0;
int KM[NombreGroupement*2];
int KMG[NombreGroupement];
int l=0;
int P=0;
int pui=0;
NbrGroupe=0;

// colone 1 Kj          // colone 2 Mj
    TableVerite[0][0]=0;      TableVerite[1][0]=0;
    TableVerite[0][1]=0;      TableVerite[1][1]=1;
    TableVerite[0][2]=0;      TableVerite[1][2]=1;
    TableVerite[0][3]=1;      TableVerite[1][3]=1;
    TableVerite[0][4]=1;      TableVerite[1][4]=1;
    TableVerite[0][5]=0;      TableVerite[1][5]=3;
    TableVerite[0][6]=0;      TableVerite[1][6]=3;
    TableVerite[0][7]=2;      TableVerite[1][7]=1;
    TableVerite[0][8]=2;      TableVerite[1][8]=1;
    TableVerite[0][9]=0;      TableVerite[1][9]=5;
    TableVerite[0][10]=0;     TableVerite[1][10]=5;
    TableVerite[0][11]=1;     TableVerite[1][11]=3;
    TableVerite[0][12]=1;     TableVerite[1][12]=3;
```

```
TableVerite[0][13]=0;    TableVerite[1][13]=7;
TableVerite[0][14]=0;    TableVerite[1][14]=7;
TableVerite[0][15]=3;    TableVerite[1][15]=1;
TableVerite[0][16]=3;    TableVerite[1][16]=1;
TableVerite[0][17]=0;    TableVerite[1][17]=7;
TableVerite[0][18]=0;    TableVerite[1][18]=7;
TableVerite[0][19]=1;    TableVerite[1][19]=3;
TableVerite[0][20]=1;    TableVerite[1][20]=3;
TableVerite[0][21]=0;    TableVerite[1][21]=5;
TableVerite[0][22]=0;    TableVerite[1][22]=5;
TableVerite[0][23]=1;    TableVerite[1][23]=2;
TableVerite[0][24]=1;    TableVerite[1][24]=2;
TableVerite[0][25]=0;    TableVerite[1][25]=3;
TableVerite[0][26]=0;    TableVerite[1][26]=3;
TableVerite[0][27]=1;    TableVerite[1][27]=1;
TableVerite[0][28]=1;    TableVerite[1][28]=1;
TableVerite[0][29]=0;    TableVerite[1][29]=1;
TableVerite[0][30]=0;    TableVerite[1][30]=1;
TableVerite[0][31]=0;    TableVerite[1][31]=0;
printf("          -Partie 05: Extraire de Kj et Mj .\n");

// 01- Extraire du Kj & Mj .
while(NbrGroupe<NombreGroupement)
{// Dubet de la Boucle while
 Demare=Arret+1;
 Arret=Demare+PasGroupement-1;
 ColTab=2;
 LignTab=0;
 printf("\n");
 VarVer=Demare;
 while( VarVer<=Arret)
 {//Dubet de la boucle while
  TabGrouN[VarVer]=TabGroupe[VarVer];
  printf(" %d ",TabGroupe[VarVer]);
  VarVer++;
 }//Fin de la Boucle while
 for(Temp=Demare;Temp<=Arret;Temp++)
 {//Dubet de la boucle for
  if(Temp==Demare) puis=puis;
  else
```

```
{// Dubet de Else
 puis=puis*2;
 Cos=Cos+(puis*TabGrouN[Temp]);
}// Fin de Else
}//Fin de la Boucle for

 if (TabGrouN[Demare]==1) Cos=Cos+1;
 printf(" Ligne %d ",Cos);
  if(l<=NombreGroupement*2)
 {// Dubet de If
  KM[l]= TableVerite[1][Cos];
  KMG[P]=TableVerite[1][Cos];
  l=l+1;
  P++;
  KM[l]= TableVerite[0][Cos];
  l=l+1;
 }// Fin de If
 printf ("  Mj= %d,  Kj= %d  Pj= %d",TableVerite[1][Cos], TableVerite[0][Cos], NbrGroupe);
 puis=1;
 Cos=0;
 NbrGroupe++;
}//Fin  de la Boucle while
 printf("\n \n Les Couple (Mj,Kj) Existant:\n");

for (l=0; l<NombreGroupement*2; l++)
{// Dubet de la boucle for.
 printf(" %d /",KM[l]);
 }// Fin de la boucle for.
 l=0;
printf(" \n \n L'enseble des Qj equivalant a la constante ( %d ) sont les suivant \n",Constante);
 pui=1;
  // 02- Ecriture de Qj
 while(l<NombreGroupement*2)
{// Dubet de la boucle while.
  if (((l%2)==0)& (KM[l]==0)) {l++; pui++;}
 else if (((l%2)==0)& (KM[l]!=0)){printf("\n Q %d : %d (X)",pui++/2,KM[l]); l++;pui++;}
  if (((l%2)!=0)& (KM[l]>0)) {printf(" * 2 puis %d ",KM[l]*((l-1)/2));l++;}
 else l++;
}// Fin de la boucle While.
```

III.2.e) Calcule de nombre additionneur :

le nombre d'additionneur est le nombre des operateur d'ajout existant entre les Qj non nul avec ajout des additionneur supplémentaires comme on la déjà nome ces dernier sont des additionneur on les rajoute quand Mj=3 ou bien Mj =5 ou bien Mj=7 ma a condition ne compte pas plusieurs fois les ajout supplémentaires sauf une seule fois

```
int gh=0;
 int eti=0;
 int v1=0;
 int v2=0;
 int df=0;
int KMG1[NombreGroupement];
int ne=0;
int cptg=0;
int vaf=0;
int pat=0;
 printf("                -Partie 06: Extraire Des Paternes Et Ces Etiration .\n");
 printf("-------------------------------------------------------------------------\n");
 printf("\n");
 P=0;
  // 01- Extraire les Paterne existant
while (P<NombreGroupement)
{// Dubet de La Boucle While.
 if (KMG[P]>0) cptg++;
 P++;
}
 P=0;
while (P<NombreGroupement)
{// Dubet de La Boucle While.
 printf(" %d ",KMG[P]);
 P++;
}
P=0;
while (P<NombreGroupement)
{// Dubet de La Boucle While.
 KMG1[P]=KMG[P];
 P++;
}// Fin de La Boucle While.
```

```
P=0;
while(P<NombreGroupement)
{// dubet de la boucle while.
 v1=KMG1[P];
 ne=P+1;
 df=0;
 while (ne<NombreGroupement)
 {//dubet de boucle while.
  if (v1==KMG1[ne]) {KMG1[ne]=0; ne++;}
  else ne++;
 }//fin de La boucle while
P++;
}//fin de la boucle while.

printf ("\n Les Paterne exiwtant ");
P=0;
while(P<NombreGroupement)
{// dubet de la boucle while.
printf(" %d ",KMG1[P]);
P++;
}// Fin de La boucle while.

 // 02- Calcule Nombre d'etiration de ces paterne
P=0;
df=0;
while (P<NombreGroupement)
{// dubet de la boucle while
if (KMG1[P]>0) {df++;P++;}
else P++;
}// fin de la boucle while
 printf(" \n \n Le nombre d'paterne du couples Kj & Mj %d  %d ",df, cptg);
 int Pater[df*2];
 P=0;
 ne=0;
while (P<NombreGroupement)
{// dubet de while
 if (KMG1[P]==0) P++;
 else {// dubet de else
      vaf=KMG1[P];
      ne=0;
```

```
        eti=0;
        while (ne<NombreGroupement)
        {// dubet de while
         if (vaf==KMG[ne]) { eti++; ne++; }
         else ne++;
        }// fin de while
        Pater[pat]=vaf;
        pat++;
        Pater[pat]=eti;
        pat++;
        P++;
        }//fin de else
}// fin de while

printf("\n");
pat=0;
while (pat<df*2)
{// dubet de while.
 printf (" %d +",Pater[pat]);
 pat++;
}// fin de while.
```

Conclusion :

A ce chapitre on a vous présente un développement simple et facile de algorithme de radix-2r mai notre groupe a considère que ce travail comme la premier version de développement de ce algorithme qui est toujours en développement et qui prouve jour après jour qu'il est le plus efficace.

Au prochaine travail qu'il est maintenant on laboratoire au sein de CDTA est de extraire une nouvelle version plus efficace que celle-ci quelle va voir le soleil le plus tôt possible .

Conclusion générale

Plusieurs appareils utilisés dans notre vie courante utilisent le traitement de signal et les circuits intégrés. La rapidité du traitement de signal est assurée par la rapidité des algorithmes de multiplication utilisés et du nombre d'additionneurs existant. On rencontre plusieurs propositions et plusieurs algorithmes mais ils sont toujours d'une complexité très élevée et où le nombre d'additionneurs est élevé ce qui implique moins de rapidité d'exécution.

Dans le travail que nous avons présenté dans ce mémoire, on introduit la discipline du traitement de signal et le rôle joué par la multiplication pour assurer la rapidité du traitement. On a présenté aussi une nouvelle solution qui est formalisée sous forme d'un algorithme qui est le résultat de la combinaison de deux autre algorithmes : l'algorithme Radix-2r et l'algorithme de V.lefevre. Cette combinaison nous permet d'assurer un nombre d'additionneur minimal à 25% par rapport aux algorithmes similaires et une complexité linéaire.

A la fin de ce mémoire on a pu présenter les résultats obtenus à partir de notre programme implémenté en langage c++ et comparer les résultats obtenus par les autres algorithmes.

Notre algorithme est plus performant car il présente plusieurs avantages que ce soit le gain de 25% du nombre additionneur, la vitesse d'exécution ou bien la complexité linéaire de l'algorithme. Mais les recherches sont toujours en cours pour avoir des résultats plus performants et plus efficaces.

On espère qu'on a atteint notre but dans ce mémoire qui ne représente en réalité qu'une goutte d'eau dans la mer du domaine

Annexe

Annexe

I. Présentation du CDTA :

Notre stage ce déroule au sein du Centre de Développement des Technologies Avance (CDTA) quelle est situe au Baba Ahssen Alger.

Le Centre de Développement des Technologies Avancées (**CDTA**) a été crée au sein du Commissariat aux Energies Nouvelles en 1982 en tant que Centre de Développement des Techniques Avancées. Le Centre était composé des laboratoires suivants :

Architecture des Systèmes,

Robotique

Laser

Plasma

Fusion thermonucléaire

Etude spatiale des rayonnements

22 mars 1988 : Création du ''Centre de Développement des Technologies Avancées'' par décret présidentiel n° 88-61.

28 décembre 1988 : Rattachement de l'Unité de Développement de la Technologie du Silicium au CDTA par arrêté ministériel.

29 novembre 1988 : L'organisation interne du CDTA fixée par arrêté interministériel :

- Départements de support à la recherche :

Formation et information scientifique

Administration et Moyens

Etudes et Moyens Techniques

- Départements de recherche et développement:

Cybernétique

Milieux Ionisés

Microélectronique

01 décembre 2003 : Décret exécutif n° 03-457 modifiant et complétant le décret N° 88-61 du 22 mars 1988 portant création du Centre de Développement des Technologies

Avancées et passage du Centre au statut d'Etablissement Public à Caractère Scientifique et Technologique.

02 septembre 2006 : Arrêté interministériel portant organisation interne du Centre de Développement des Technologies Avancées :

- Départements administratifs et techniques:

Ressources humaines et relations extérieures

Finances, comptabilité, des moyens et de la gestion des projets

Information scientifique et technique, des équipements scientifiques et de la valorisation des résultats de la recherche

- Divisions de recherche:

Architecture des systèmes et multimédia

Microélectronique et nanotechnologie

Milieux ionisés et laser

Productique et Robotique

08 novembre 2007 : Arrêté interministériel complétant l'arrêté interministériel du 02 septembre 2006 portant organisation interne du Centre de Développement des Technologies Avancées et relatif aux activités de l'Unité de Développement de la Technologie du Silicium.

15 Mars 2011 : Arrêté Ministériel N° 143 portant création d'une unité de recherche en photonique et optique, sise à Sétif et rattachée au Centre de Développement des Technologies Avancées.

II.2.organisation des département procède technologique avance et gestion des projets :

III. Le Codage Binaire :

III.1. Présentation du Binaire :

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide de « contacteurs » (interrupteurs) fermés pour « vrai » et ouverts pour « faux » il était possible d'effectuer des opérations logiques en associant le nombre 1 pour « vrai » et 0 pour « faux ».

Ce codage de l'information est nommé **base binaire**. C'est avec ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états (représentés par les chiffres 0 et 1) pour coder les informations.

L'homme calcule depuis 2000 ans avant Jésus-Christ avec 10 chiffres (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), on parle alors de base décimale (ou base 10). Toutefois dans des civilisations plus anciennes ou pour certaines applications actuelles d'autres bases de calcul ont et sont toujours utilisées :

- base sexagésimale (60), utilisée par les Sumériens. Cette base est également utilisée dans le système horaire actuel, pour les minutes et les secondes ;
- base vicésimale (20), utilisée par les Mayas ;
- base duodécimale (12), utilisée par les anglo-saxons dans leur système monétaire jusqu'en 1960 : un « pound » représentait vingt « shilling » et un « shilling » représentait douze « pences ». Le système d'heure actuel fonctionne également sur douze heures (notamment dans la notation anglo-saxonne) ;
- base quinaire (5), utilisée par les Mayas ;
- base binaire (2), utilisée par l'ensemble des technologies numériques.

III.2 Bits :

Le terme **bit** (*b* avec une minuscule dans les notations) signifie « **binary digit**», c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique. Il est possible de représenter physiquement cette information binaire :

- par un signal électrique ou magnétique, qui, au-delà d'un certain seuil, correspond à la valeur 1 .

IV. Code Gray :

Le **code de Gray**, également appelé **binaire réfléchi**, est un type de codage binaire permettant de ne modifier qu'un seul bit à la fois quand un nombre est augmenté d'une unité. Le nom du code vient de l'ingénieur américain Frank Gray qui déposa un brevet sur ce code en 1953 sont intre est Le fait de modifier plusieurs bits lors d'une simple incrémentation peut mener, selon le circuit logique, à un état transitoire indésirable dû au fait que le chemin logique de chaque bit dispose d'un délai différent. Ainsi, lors du passage de la valeur "01" à la valeur "10" en binaire naturel, il est possible d'observer un état transitoire "00" si le bit de droite commute en premier ou "11" dans le cas contraire. Si le circuit dépendant de cette donnée n'est passynchrone, l'état transitoire peut perturber les opérations en faisant croire au système qu'il est passé par un état normalement non atteint à ce stade. Ce code permet de contourner cet aléa en forçant la commutation d'un seul bit à la fois, évitant ainsi les états transitoires.

V. Les ressources Logiques :

Les nombres réels ont une mantisse de taille finie, donc sans perte de généralité, lorsque nous cherchons décompositions de multiplication constants, on peut supposer que tout le constant donné coefficients seront entiers parce que tout nombre fractionnaire avec précision finie peut être gauche décalée jusqu'à ce qu'il devienne un entier.

Ce processus n'encourt pas de coût et est réversible, c'est à dire après la multiplication constante a été mis en place, nous pouvons changer le résultat de corriger tout facteur de 2^n, où n est un nombre entier.

VI. Microélectronique :

La **micro-électronique** est une spécialité du domaine de l'électronique

Tel que son nom le suggère, la micro-électronique s'intéresse à l'étude et à la fabrication de composants électroniques à l'échelle micronique.

Ces composants sont fabriqués à partir de matériau à semi-conducteurs (comme le Silicium) au moyen de diverses technologies dont la photolithographie. Cette technologie permet l'intégration de nombreuses fonctions électroniques sur un même morceau de Silicium (ou autre semi-conducteur) et donc à un prix plus bas. Les circuits ainsi réalisés sont appelés puces ou circuits intégrés. Ils peuvent être standards ou

spécifiques à une application (ils sont alors nommés "ASIC" : Application Spécifique intégrâtes Circuit). Tous les composants électroniques discrets : les transistors, les condensateurs, les inductances, les résistances, les diodes et, bien sûr, les isolants et les conducteurs, ont leur équivalent en micro-électronique.

VII. Complexité algorithmique :

VII.1. Algorithme :

Un algorithme est suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème.

VII.2 Complexité algorithmique :

La complexité d'un algorithme est la mesure du nombre d'opérations fondamentales qu'il effectue sur un jeu de données. La complexité est exprimée comme une fonction de la taille du jeu de données.

◆Nous notons Dn l'ensemble des données de taille n et C(d) le coût de l'algorithme sur la donnée d.

Bibliographie

Bibliographie :

[01] **T.Dumartin** [Rappel Traitement Du Signal].2004

[02] **A.Oumnad** [Traitement Numerique Du Signal].

[03] **Michel Terré** [Traitement Numerique Du Signal]2011.

[04] **I.Zambettazis** [Numerisation Du Signal].

[05] **S.Tisserant** [Traitement Du Signal]2008

[06] **Mr.M.V.Sathish, Mrs Sailaja** [VLSI architecture of parallel multiplier– accumulator based on Radix-2 modified booth algorithm].2007

[07] **J. Thong and N. Nicolici**, [An optimal and practical approach to single constant multiplication] , September 2011.

[08] **Y. Voronenko and M. P¨uschel** [Multiplierless Multiple Constant Multiplication] 2007.

[09] **Florent De Dinechins, Vincent Lefevre**[Constant Multipliers of FPGAs]May 2000

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

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