

RÉPUBLIQUE ALGERIENNE DÉMOCRATIQUE ET POPULAIRE

» 0 «

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

» 0 «

ECOLE NATIONALE POLYTECHNIQUE

» 0 «

Département : ÉLECTRONIQUE Option : TÉLÉCOMMUNICATIONS

المدرسة الوطنية المتعددة التقنيات  
BIBLIOTHÈQUE  
Ecole Nationale Polytechnique

THESE DE MAGISTER

THEME

CONCEPTION D'ARCHITECTURES DIGITALES  
UNIFIÉES TCD / TCDI 2-D :  
ETUDE COMPARATIVE DU MODE DE CALCUL  
HALF-LINE ET DE L'ARITHMÉTIQUE DISTRIBUÉE

Présentée par : Mr M. TOUNSI

Ingénieur d'état en Electronique, ENP.

Devant le Jury :

Mr D. BERKANI, Professeur à l'E.N.P..... Président  
Mr A. FARAH, Professeur à l'E.N.P..... Rapporteur  
Mr A. BEL OUCHRANI, Docteur d'état à l'E.N.P ..... Examineur  
Mr R. SADOON, Chargé de cours à l'E.N.P ..... Examineur  
Mr Z. TERRA, Docteur-Ingénieur, Chargé de cours à l'E.N.P ..... Examineur

# REMERCIEMENTS



De prime abord, je tiens à remercier vivement mon directeur de thèse, le Professeur **A. FARAH** de l'E.N.P, pour toute son assistance, ses orientations clairvoyantes et ses encouragements répétés qu'il m'a prodigués tout au long de la préparation de cette thèse. Qu'il trouve ici, l'expression de ma grande reconnaissance et l'assurance de ma parfaite considération pour ses multiples enseignements.

Je remercie **Mr D. BERKANI**, Professeur à l'E.N.P, de me faire l'honneur de présider mon jury. Qu'il trouve ici, l'expression et l'assurance de ma parfaite reconnaissance à ses précieux encouragements et ses riches enseignements.

Mes plus vifs remerciements vont à **Mr A. BELLOUCHRANI**, Docteur d'état à l'E.N.P, **Mr Z. TERRA**, Docteur-Ingénieur chargé de cours à l'E.N.P, et **Mr R. SADOUN**, chargé de cours à l'E.N.P, d'avoir bien voulu accepter d'examiner mon travail de thèse et pour tout l'intérêt qu'ils lui ont accordé.

Mes vifs remerciements vont également à **Mr E.M AIT NOURI**, du centre de développement des Techniques Avancées (C.D.T.A) pour son aide précieuse et tous ses encouragements à mener à bien mon travail.

Je remercie aussi toute ma famille, particulièrement ma Mère, mes frères, neveux et nièces, mon Oncle Hachemi, sa femme et ses adorables enfants, tous mes amis (ies) ainsi que tous mes collègues et responsables de la direction « Recherches-Développement » de SONELGAZ pour leurs encouragements et leur soutien.

# DÉDICACE



JE DÉDIE CE PRÉSENT TRAVAIL À LA MÉMOIRE DE MON PÈRE

ET À LA MÉMOIRE DE CELUI QUI M'A INITIÉ AUX ÉTUDES

ET QUI AURAIT VOULU QUE JE SOIS DOCTEUR,

MON FRÈRE AÎNÉ ABDELKADER.

## ملخص

إن استعمال محولة تجب المنفصلة (TCD) بشفرة الصور الرقمية في وقتنا الحاضر، مقرر في مختلف القواعد الدولية . الهدف هو إيجاد تطبيقات وخدمات مثل المحاضرة-البصرية و HDTV بحيث لا يمكن تحقيق ذلك إلا بزرع TCD و تحويلها العكسية في الدارات المتكاملة (ASICs) نظرا لاحتلالها الطبقات الأكثر عملا. بالرغم من أن معظم حساباتها تستنتج بالدمج التسلسلي، فإن دمجها الحسي أصبح هو العنصر المفتاح في VLSI لضغط الصور.

سنيين في هذا البحث أن هناك خفة عالية في التوافق بين الحسابات و الأشكال بحيث تبقى دائمة باستعمال دراسة مقارنة و أدوات حسابية فعالة كالحساب الموزع و طريقة الحساب بالخط "ON LINE" التي تسمح باشتقاق الحساب المجهين من أجل تكوين شكل موحد

## RÉSUMÉ

*L'utilisation de la transformée cosinus discrète (T.C.D) en codage d'images numériques, est de nos jours adoptée par les diverses normes internationales. L'objectif d'offrir des applications et des services tels que la visioconférence et la HDTV, reporté sur le codec, ne pourrait être atteint que par l'implantation de la TCD et sa transformée inverse sur des circuits intégrés dédiés (ASICs), vu qu'elles occupent les blocs les plus opérationnels. Quoique la majorité de leurs algorithmes, sont dédiés à une implémentation software, leur implémentation hardware est devenue l'élément-clé en VLSI pour la compression d'image. Dans notre travail de thèse, nous montrons avec une étude comparative, qu'une grande souplesse d'adaptation algorithmes-architectures existe avec l'utilisation d'outils arithmétiques efficaces tels l'arithmétique distribuée et le mode de calcul half-line qui permettent de dériver un algorithme hybride TCD/TCDI 2-D pour la conception d'une architecture unifiée.*

## ABSTRACT

*Transform coding utilising the discrete cosine transform (DCT) has been adopted in various standards of image compression. As it is the most operative bloc of the encoder's range, the modern digital image systems require fast DCT computational algorithms, especially dedicated to VLSI implementation. In this work, we show that an adequation algorithms-architectures exists and we present the derivation and the choice of an hybrid algorithm DCT/IDCT 2-D, based on the distributed arithmetic and the half-line mode of computation, in order to obtain a regular unified architecture TCD/TCDI 2-D.*

## LISTE DES MOTS-CLES

Synthèse architecturale

Algorithmique rapide

Transformée orthogonale Cosinus Discrète T.C.D

Circuits intégrés dédiés ( ASICs )

Codage d'images

Norme JPEG

Standard H.261

Standard MPEG

Niveaux de description d'un circuit intégré

Systèmes redondants d'écriture des nombres

Arithmétique sérielle ON-LINE et HALF-LINE

Arithmétique distribuée

Adéquation algorithme - architecture

Mémoire ROM à double buffer

Additionneur à retenue conservée ( CSA )

Additionneur à retenue anticipée ( CLA )

Additionneur à propagation de retenue ( CPA )

Arbre additionneur multi-niveaux

Modèle de comparaison des ressources architecturales

Modèle de comparaison de Kai-Hwang

## SOMMAIRE

PAGES

INTRODUCTION GENERALE.....	1
<b>CHAPITRE I : GENERALITES SUR LES TECHNIQUES DE CODAGE D'IMAGES</b>	
1 - INTRODUCTION.....	5
2 - NUMERISATION, CODAGE ET COMPRESSION D'IMAGES.....	5
2.1 - Techniques de compression d'images.....	6
2.1.1 - Techniques avec distorsions.....	6
2.1.1.1 - Le codage prédictif.....	6
2.1.1.2 - Le codage par transformées orthogonales.....	7
2.1.1.3 - Le codage hybride.....	10
2.1.1.4 - Le codage par quantification vectorielle.....	10
2.1.1.5 - La technique de troncature de bloc.....	10
2.1.1.6 - Le codage par interpolation.....	10
2.1.2 - Techniques sans distorsions.....	11
2.1.2.1 - Technique des plages.....	11
2.1.2.2 - Technique des plans de bits.....	11
2.1.2.3 - Techniques statistiques.....	11
2.2 - Standardisation en codage d'images.....	11
2.2.1 - Aperçu sur la norme JPEG.....	12
2.2.1.1 - Propriétés.....	12
2.2.1.2 - Le codage par la transformée TCD.....	12
2.2.1.2.a - Codage par TCD séquentielle.....	13
2.2.1.2.b - Codage par TCD progressive.....	13
2.2.1.2.c - Propriétés.....	13
2.2.1.3 - Image à plusieurs composants.....	14
2.2.2 - Aperçu sur le standard H.261.....	15
2.2.2.1 - Codeur.....	15
2.2.2.2 - Codage par transformée.....	15
2.2.2.3 - Quantification.....	16
2.2.2.4 - Types de balayages.....	16
2.2.2.5 - Estimation de mouvement.....	16
2.2.2.6 - Filtre de la boucle.....	17
2.2.2.7 - Codage à longueur variable ( CLV ).....	17
2.2.3 - Aperçu sur le standard MPEG.....	17
2.2.3.1 - Réduction de la redondance temporelle.....	18
2.2.3.2 - Compensation de mouvement.....	18
2.2.3.3 - Estimation de mouvement.....	19
2.2.3.4 - Réduction de la redondance spatiale.....	19
3 - CONCLUSION.....	20

**CHAPITRE II : LA TRANSFORMEE COSINUS DISCRETE T.C.D.**

1 - INTRODUCTION.....	21
2 - ALGORITHMIQUE RAPIDE DE LA T.C.D.....	21
2.1 - Définitions.....	21
2.1.1 - T.C.D unidimensionnelle.....	21
2.1.2 - T.C.D bidimensionnelle.....	22
2.2 - Propriétés.....	22
2.3 - Algorithmes T.C.D 2-D.....	23
2.3.1 - Approche Ligne / Colonne.....	23
2.3.2 - Approche Directe.....	24
2.3.2.1 - Algorithme rapide TCD 2-D de N.I.Cho et S.U.Lee.....	24
2.3.2.2 - Discussion.....	31
2.4 - Algorithmes T.C.D 1-D.....	31
2.4.1 - Décomposition en radical 2.....	32
2.4.1.1 - Algorithme récursif de Hou.....	32
2.4.1.2 - Algorithme de Chen.....	37
2.4.1.3 - Algorithme par T.F.D.....	40
2.4.2 - Décomposition en radical 4.....	43
2.4.3 - Décomposition en radical 8.....	44
2.4.4 - Discussion.....	44
3 - CONCLUSION.....	45

**CHAPITRE 3 : OUTILS ARITHMETIQUES : MODE HALF-LINE ET**

**ARITHMETIQUE DISTRIBUEE**

1 - INTRODUCTION.....	46
2 - INCOMMODITES DE L'ARITHMETIQUE CLASSIQUE.....	46
2.1 - Incommodité de la circulation parallèle des opérandes.....	46
2.2 - Incommodité de la propagation de la retenue.....	47
2.3 - Incommodité de la simultanéité.....	47
3 - ARITHMETIQUE « SERIE » OU MODE DE CALCUL SERIEL.....	47
4 - LES SYSTEMES REDONDANTS D'ECRITURE DES NOMBRES.....	48
5 - ARITHMETIQUE ON-LINE ET HALF-LINE.....	50
5.1 - Le mode de calcul On-line.....	50
5.2 - Le mode de calcul Half-line.....	50
5.3 - Spécificités des modes On-line et Half-line.....	50
5.4 - Evaluation de la TCD par le Half-line.....	51
5.4.1 - Evaluation de l'expression généralisée de la T.C.D.....	51
5.4.2 - Algorithme de l'addition half-line.....	54
5.5 - Structure générale d'un algorithme half-line.....	55

5.6 - Implémentation de l'expression généralisée de la T.C.D.....	56
5.6.1 - Le bloc de normalisation .....	56
5.6.1.1 - Proposition d'une ROM.....	56
5.6.1.2 - Circuits de décodage de la ROM.....	57
5.6.1.3 - Additionneur à retenue conservée (CSA).....	58
5.6.1.4 - CSAs multi-niveaux .....	59
5.6.1.5 - Architecture du bloc de normalisation pour le radical 8.....	59
5.6.1.6 - Architecture du bloc de normalisation pour le radical 8.....	60
5.6.2 - Le bloc de sélection .....	61
5.6.2.1 - Additionneur à Retenue Anticipée (CLA).....	63
5.6.2.2 - Circuit de sélection du bit résultat.....	63
5.6.3 - Architecture pipe-linée du bloc de sélection.....	65
5.6.4 - Discussion .....	65
6. ARITHMETIQUE DISTRIBUEE (D.A).....	66
6.1 - Dérivation d'un calcul « arithmétique distribué ».....	66
6.2 - Implémentation .....	66
6.2.1 - Réduction de la taille de la ROM.....	67
6.2.2 - Amélioration de la vitesse de traitement.....	70
6.2.3 - Limites de la conception classique .....	71
6.3 - Optimisation de l'algorithme distribuée de la TCD.....	71
6.3.1 - Principe de l'algorithme.....	71
6.3.2 - Implémentation de l'expression généralisée de la T.C.D.....	72
6.3.3 - Etage de Normalisation.....	73
6.3.3.1 - Architecture du bloc de Normalisation pour la T.C.D à radical 8.....	73
6.3.3.2 - Architecture du bloc de Normalisation pour la T.C.D à radical 4.....	74
6.3.4 - Etage d'Accumulation.....	74
6.3.4.1 - Additions parallèles.....	75
6.3.4.2 - Propagation de la retenue.....	76
6.4 - Discussion .....	77
6.5 - Remarque.....	77
7. CONCLUSION.....	78

## **CHAPITRE 4 : COMPARAISON DES ARCHITECTURES**

1 - INTRODUCTION.....	79
2 - MODELES DE COMPARAISON.....	79
2.1- Le modèle de comparaison des ressources architecturales .....	79
2.2. Le modèle de comparaison de Kai Hwang.....	79
3. LA COMPARAISON SPACIALE.....	80
3.1 - L'espace mémoire.....	80
3.2 - L'espace opératif.....	81



4. LA COMPARAISON TEMPORELLE.....	81
4.1. Temps de propagation des ressources architecturales .....	81
4.2. La fréquence (le chemin critique).....	81
5. COMPARAISON DES LA PRECISION .....	82
5.1 - Nombre de bits du signal d'entrée (pixels).....	83
5.2. Nombre de bits des noyaux cosinus ou constantes A.....	83
6. COMPARAISON DES ARCHITECTURES .....	83
6.1. Application et comparaison en mode Half-Line .....	84
6.1.1 - Les ressources architecturales .....	84
6.1.1.1 - L'espace mémoire.....	84
6.1.1.2 - L'espace opératif .....	84
6.1.2 - Performances temporelles.....	85
6.1.2.1 - Estimation du temps de propagation global.....	85
6.1.2.2 - Discussion .....	86
6.1.2.3 - Estimation de la période (fréquence) .....	86
6.1.3. L'étude de précision.....	87
6.2. Application et comparaison en arithmétique distribuée.....	89
6.2.1 - Les ressources architecturales .....	89
6.2.1.1 - L'espace mémoire.....	89
6.2.1.2 - L'espace opératif .....	89
6.2.2 - Performances temporelles.....	90
6.2.2.1 - Estimation du temps de propagation global.....	90
6.2.2.2 - Discussion .....	91
6.2.2.3 - Estimation de la période (fréquence) .....	91
6.2.3. L'étude de précision.....	91
6.3. Comparaison et choix final .....	93
7. CONCLUSION .....	93

**CHAPITRE 5 : CONCLUSION GENERALE .....** 95

**ANNEXE A : RAPPELS SUR LES ADDITIONNEURS PARALLELES**

**ANNEXE B : PROGRAMMATION EN LANGAGE C++**

**BIBLIOGRAPHIE**

المدرسة الوطنية المتعددة التقنيات  
BIBLIOTHEQUE — المكتبة  
Ecole Nationale Polytechnique

**INTRODUCTION**  
**GÉNÉRALE**

## INTRODUCTION GÉNÉRALE

**L**e traitement numérique du signal connaît de nos jours une utilisation et un développement considérables dans les diverses disciplines d'ingénierie. Sa puissance réside dans la multitude d'outils mathématiques rapides et efficaces utilisés pour résoudre et modéliser des applications effectives. Il s'en trouve que l'importance et les performances élevées attendues de certaines applications modernes, ont conduit à développer une nouvelle branche de ces techniques numériques, à savoir l'« Algorithmique Rapide » qui vise à développer des algorithmes rapides de calcul, exploitant au maximum les spécificités du problème traité.

Ces algorithmes rapides gagneraient alors à être implémentés sur des processeurs de calcul pour réaliser les fonctions voulues. Malheureusement, les relations entre algorithmes rapides et architectures de traitement de signal ont évolué pratiquement de manière non-concourante et l'on a même cru, les progrès rapides de la technologie aidant, que le recours à des algorithmes rapides était devenu inutile à cause de la perte de régularité qu'ils introduisent dans leurs graphes de fluence. Aussi, la conception et l'implémentation classiques, efficaces encore pour certaines applications à bat débit, sont vite dépassées et le recours à de nouvelles techniques est devenu incontournable. C'est ainsi que le développement de circuits intégrés spécifiques du type ASIC (en Anglais : Applied Specific Integrated Circuits), est venu répondre aux préoccupations architecturales des applications de traitement en temps réel. La technique et la méthodologie de conception, basées sur la recherche d'une adéquation algorithmes-architectures de l'application considérée, permettent d'arriver à des algorithmes très efficaces une fois implantés.

Le traitement et la transmission d'images sont des applications qui ont bénéficié du développement des techniques numériques et des technologies des semi-conducteurs mais suscitent toujours beaucoup d'intérêt à cause des exigences modernes de traitement de l'information. En effet, des applications comme la visiophonie, la visioconférence, la TVHD ainsi que la consultation des banques de données et d'images, sont des marchés en grande diffusion vu l'importance des services qu'ils offrent. Ces applications présentent certaines difficultés liées en majeure partie au débit des images numérisées traitées. En ce sens que la quantité d'information produite par la numérisation dépassait largement les capacités de stockage et les débits des réseaux existants en dépit des efforts des chercheurs et des organismes de normalisation dans ce domaine.

La réduction du débit et le codage des images de la façon la plus appropriée, sont des domaines de la recherche qui connaissent chaque jour, une innovation. Mais, il reste que l'objectif d'offrir ces services à des prix raisonnables n'est pas entièrement réglé : le problème du coût est surtout reporté sur le CODEC et la solution ne pourra venir que du développement de circuits intégrés ASICs pour ces applications.

Le codage par la transformée orthogonale Cosinus discrète T.C.D est souvent recommandé par les diverses normes internationales de codage d'images. Le développement d'algorithmes rapides de calcul de cette transformée et leur implémentation sur circuits ASICs, sont devenus les éléments-clés de la VLSI de compression d'image. Cette thèse est justement consacrée à la conception d'architectures digitales d'un circuit dédié pour la transformée TCD 2-D et sa transformée inverse TCDI 2-D.

La conception d'architectures est la première phase d'un processus qui doit aboutir à la fabrication du circuit intégré considéré. C'est dans cette phase de conception que s'élaborent les plans du circuit sur la base de ses spécifications fonctionnelles. Dans la méthodologie générale de conception, il y'a trois (03) niveaux de description majeurs [1] du circuit à savoir :

- le niveau comportemental s'attachant au fonctionnement du circuit;
- le niveau structurel s'attachant à son architecture;
- et le niveau physique à sa réalisation matérielle.

Ces niveaux d'abstraction peuvent être représentés selon les trois (03) branches de la représentation en Y de la figure 1, proposée par D.D.Gajski [2].

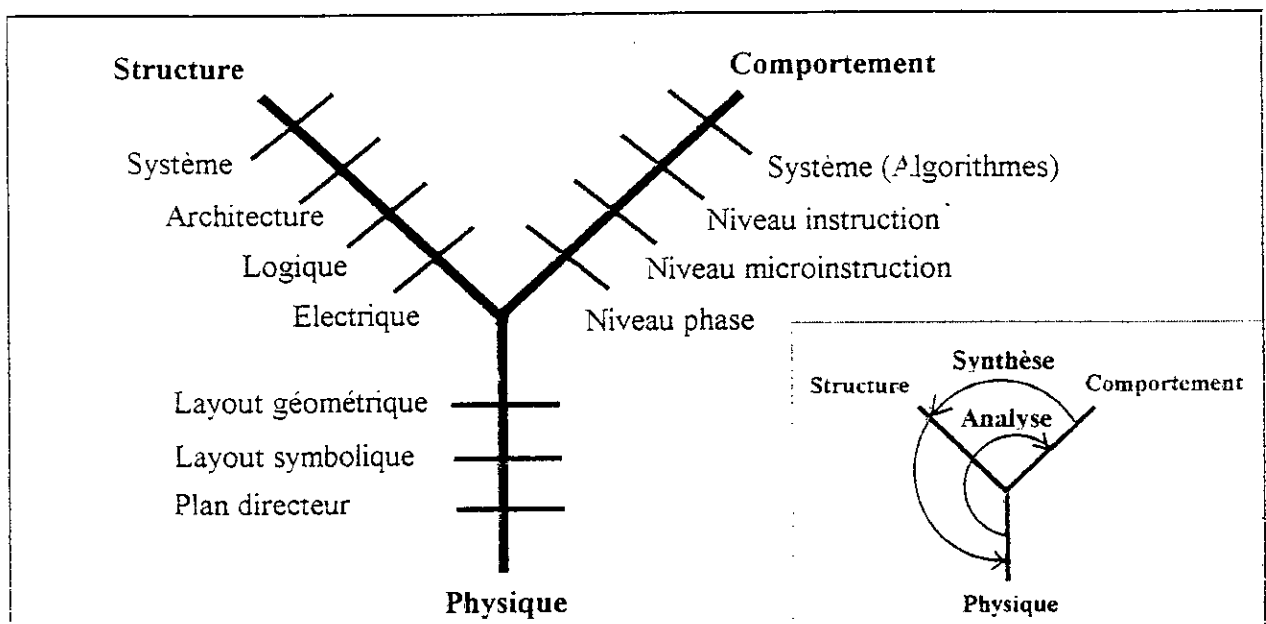


Figure 1 - Représentation en Y des niveaux de description d'un circuit intégré.

Notre travail dans cette thèse, s'inscrit donc dans le passage du premier au second niveau de description c'est-à-dire du niveau comportemental au niveau structurel. Notre objectif est de concevoir une architecture pour un ASIC unifié TCD 2-D / TCDI 2-D sans pour autant arriver au niveau électrique du circuit. C'est un travail de synthèse entre le niveau comportemental « algorithmes » et le niveau structurel « architecture ». Ce travail est finalisé par une étude comparative des architectures obtenues en fonction du mode de calcul adopté : l'arithmétique distribuée et le mode de calcul « half-line ».

Cette thèse est composée de cinq (04) chapitres élaborés comme suit :

- Le chapitre I est un rappel des généralités sur des techniques de compression d'images, et un résumé des spécifications des standards internationaux dans ce domaine comme la norme JPEG, la norme H.261 et la norme MPEG, pour montrer l'utilité et les avantages de l'utilisation de la transformée Cosinus discrète T.C.D en codage d'images.
- Le chapitre II est une revue générale des algorithmes bidimensionnels et unidimensionnels de la TCD et son inverse TCDI. Nous y montrons très clairement, que ces algorithmes ont été proposés sans grande préoccupation architecturale aussi bien avec l'approche de calcul direct TCD 2-D qu'avec l'approche conventionnelle ligne/colonne de calcul TCD 1-D, l'objectif étant de minimiser le nombre de multiplications, quels que soient les autres coûts associés. Les algorithmes de décomposition en radical 2, 4 et 8 de cette transformée, y sont sélectionnés pour trouver des artifices d'adaptation et d'adéquation algorithmes-architectures pour arriver à des architectures cibles unifiées possibles pour la TCD 2-D/TCDI 2-D. Nous y montrons aussi que le calcul de ces transformées se résume et ne peut se faire pour l'objectif assigné dans cette thèse, qu'avec une expression paramétrée complexe en radical 4 et 8.
- Le chapitre III montre que le processus de calcul des transformées orthogonales discrètes est caractérisée par une structure parallélo-séquentielle dont les performances d'exécution sont dues à son aspect parallèle. La diminution des contraintes inhérentes au traitement séquentiel, n'a pas été suffisamment approchée dans les diverses études menées sur le calcul des transformées orthogonales. Ce chapitre montre que toutes les possibilités algorithmiques sont conditionnées par les modes de calcul utilisés par les opérateurs arithmétiques, et que l'amélioration du calcul de ces transformées peut être obtenue en utilisant des arithmétiques adéquates comme le mode de calcul « half-line » et l'arithmétique distribuée, et révèle les architectures obtenues.

- Le chapitre IV est une comparaison des architectures obtenues avec les outils arithmétiques utilisés, tant du point de vue des ressources architecturales utilisées, le temps de propagation global, la fréquence de traitement ainsi que la précision de chaque algorithme. Des modèles de comparaison y sont donnés avec des résultats de programmation et la discussion du choix final de l'architecture unifiée à adopter compte tenu de l'étude comparative présentée.
- La conclusion générale et les recommandations de notre travail.
- Enfin, la bibliographie utilisée ainsi que deux annexes, sont données à la fin cette thèse :
  - l'annexe A traitant des additionneurs parallèles ;
  - et l'annexe B donnant la programmation en langage C++ des algorithmes de la TCD avec le mode de calcul half-line et l'arithmétique distribuée.

# CHAPITRE I



## GÉNÉRALITÉS SUR LA COMPRESSION D'IMAGES

## 1 - INTRODUCTION

Le traitement numérique du signal est l'une des techniques les plus largement utilisée dans diverses activités de l'ingénierie et connaissant aujourd'hui un essor considérable dans les disciplines recherchant des outils mathématiques rapides et efficaces. L'« Algorithmique Rapide » qui a fini par constituer une branche en soi du traitement du signal, s'attache à utiliser au maximum la spécificité du problème traité. Les transformées rapides de signal constituent essentiellement la base de cette nouvelle branche, notamment les transformées orthogonales telles que la transformée Cosinus Discrète T.C.D développée par N. Ahmed et al. [3] en 1974. utilisée en compression de données et réduction de débit. **D'où l'intérêt de présenter dans ce chapitre, les techniques de compression d'images dans une thèse consacrée à la T.C.D et son inverse T.C.D.I.**

Le traitement et la transmission d'images suscitent toujours beaucoup d'intérêt mais présentent certaines difficultés liées en majeure partie au débit de l'image numérisée traitée. En effet, la quantité d'information produite par la numérisation d'image dépassait largement les capacités de stockage et les débits des réseaux existants. Il est devenu donc nécessaire de réduire le débit et de chercher à coder l'image de la façon la plus appropriée. Comme nous le verrons plus loin, la T.C.D est très utilisée dans ce domaine.

Par ailleurs, l'homogénéité et la compatibilité des représentations devaient passer par la standardisation du codage, du stockage, de l'échange et de la restitution des informations manipulées. Ceci a fait que l'activité des organismes internationaux de normalisation a été très intense notamment au cours des quinze (15) dernières années en compression d'image dans le but d'offrir des services à bas débit. En effet, des applications comme la visiophonie, la visioconférence, la TVHD, le stockage sur CD-ROM ainsi que la consultation des banques de données et d'images sont des marchés de grande diffusion, estimés très importants.

Il reste que l'objectif d'offrir ces services à des prix raisonnables n'est pas entièrement réglé : le problème du coût est reporté sur le codec. La solution ne pourra venir que du développement de circuits intégrés ASICs dédiés aux fonctions de ce codec.

## 2 - NUMERISATION, CODAGE ET COMPRESSION D'IMAGES

La numérisation des images a été introduite pour le stockage d'une copie fidèle de l'image originale afin de permettre des traitements plus ou moins complexes. Numériser une image revient à la mettre sous forme d'un tableau de « pixels ». A chaque pixel est affectée une valeur numérique codée généralement sur 8 bits.



L'image digitale standard est définie dans la norme CCIR comme étant composée de trois signaux électriques : Y est le signal de luminance, CB et CR sont les signaux de différence de couleurs. Ces signaux sont générés par la matrice des signaux primaires y, r, g et b tels que :

$$\begin{cases} y = 0.299r + 0.587g + 0.114b \\ Y = 219y + 16 \\ CR = 112(r - y) / 0.701 + 128 \\ CB = 112(b - y) / 0.886 + 128 \end{cases}$$

La fréquence d'échantillonnage est de 13.5 MHz pour la luminance Y, et de 6.75 MHz pour les signaux CB et CR. Ainsi pour tout système, le nombre d'échantillons par ligne a été fixé à 720 pixels pour la luminance, 360 pixels pour les signaux CB et CR; chaque pixel étant codé sur 8 bits.

Mais, Il est vite apparu que la quantité d'informations produite par la numérisation n'était pas en adéquation avec les capacités de stockage et les débits de transmission des réseaux existants. La mise en œuvre d'algorithmes de compression est devenue alors une nécessité.

## 2.1 - TECHNIQUES DE COMPRESSION D'IMAGES

Les techniques de compression sont classées (Fig. I.1) selon deux critères [4] [5] :

Le premier critère est lié à la distorsion apportée à l'image originale, il s'agit :

- des techniques sans distorsion restituant exactement les pixels de l'image source;
- des techniques qui apportent une distorsion aux images reconstruites, indétectable à l'oeil nu.

Le second critère est lié au domaine de leur application, selon que nous avons :

- des techniques spatiales appliquées directement aux pixels de l'image;
- des techniques transformées travaillant dans le domaine « transformé » après l'application d'une transformation orthogonale aux pixels.

### 2.1.1 - TECHNIQUES AVEC DISTORSIONS

#### 2.1.1.1 - CODAGE PREDICTIF [6] [7]

La technique MICD, modulation à impulsion codée différentielle, est le codage de type prédictif le plus classique. Le principe est qu'un prédicteur établit une estimation de chaque pixel à partir de ses voisins immédiats (le pixel précédent en général).

L'écart  $\varepsilon$  entre la luminance Y du pixel et sa valeur de prédiction  $Y_{pr}$  :  $\varepsilon = Y - Y_{pr}$ , est quantifiée puis transmise à la sortie du codeur telle que :  $S = \varepsilon \cdot Q = \varepsilon_Q$  où  $\varepsilon_Q$  est l'erreur de prédiction quantifiée. Le nombre de ces valeurs quantifiées étant restreint, un plus petit nombre de bits est utilisé pour leur codage. C'est à ce niveau que se fait la réduction du débit. A la réception, le décodeur reçoit l'erreur de prédiction et l'ajoute à son propre prédicteur (le même que celui du

codeur) :  $Y' = \epsilon_Q + Y_{pr}$ . Le résultat ne diffère de la valeur exacte que de l'erreur de quantification. C'est la quantification de ces écarts qui est à la base de la réduction du débit nécessaire au codage de ces erreurs, et de la réduction du débit numérique de l'image. Bien sûr, cette méthode n'est pas sans effet sur la qualité de l'image transmise, mais avec 16 niveaux de quantification, les défauts de codage sont tolérés par l'œil humain.

### 2.1.1.2 - CODAGE PAR TRANSFORMEES ORTHOGONALES [8] [9] [10]

Cette technique consiste en la représentation de l'image dans un domaine où la séparation de l'information pertinente de l'information redondante est plus facile. Les transformations orthogonales s'appliquent sur des blocs d'image, de dimension  $N \times N$ . La taille de ces blocs doit être suffisante pour que la corrélation entre les points les plus éloignés du bloc, soit négligeable. Elle est dite « longueur de corrélation », et est fixée généralement à  $8 \times 8$  ou  $16 \times 16$  pixels.

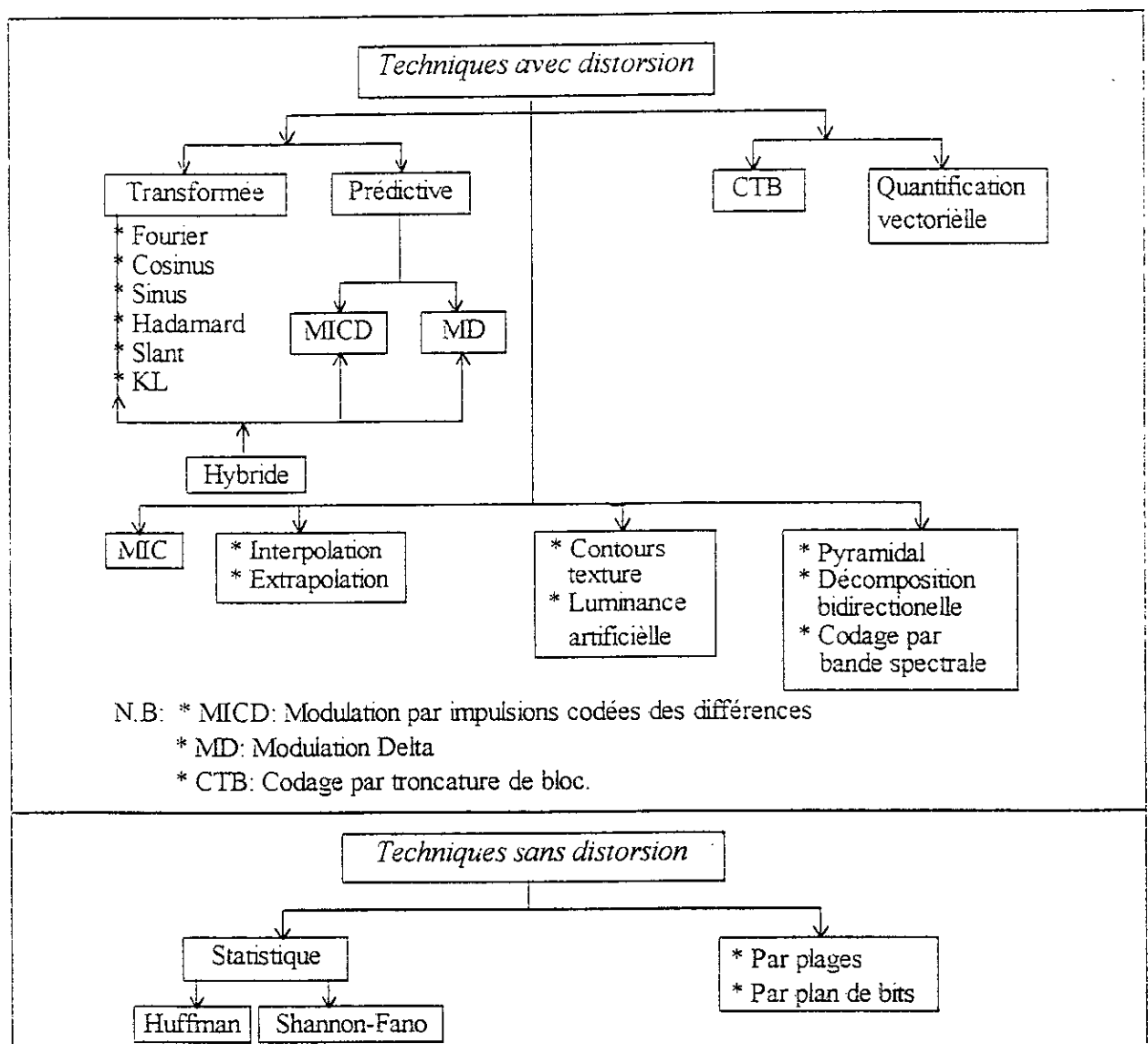


Figure I.1- Classification des techniques de compression d'images.

La transformée orthogonale optimale au sens statistique [11] en codage d'image est la transformée de Karhunen-Løve (TKL). Malheureusement, son implémentation n'est pas envisageable en temps réel vu qu'elle dépend de l'image traitée. Les autres transformées sont dites sous optimales parce qu'elles ne décorrelent pas totalement la séquence de données mais permettent une implémentation hardware.

La figure I.2 donne la classification de ces transformées en deux catégories:

1. Transformations optimales;
2. Transformations sous optimales

La catégorie des transformées sous-optimales est répartie en deux catégories :

- type 1 dont les fonctions de base reposent sur le cercle unité;
- type 2 dont les fonctions de base sont soit des fonctions sinusoidales, soit des fonctions non-sinusoidales.

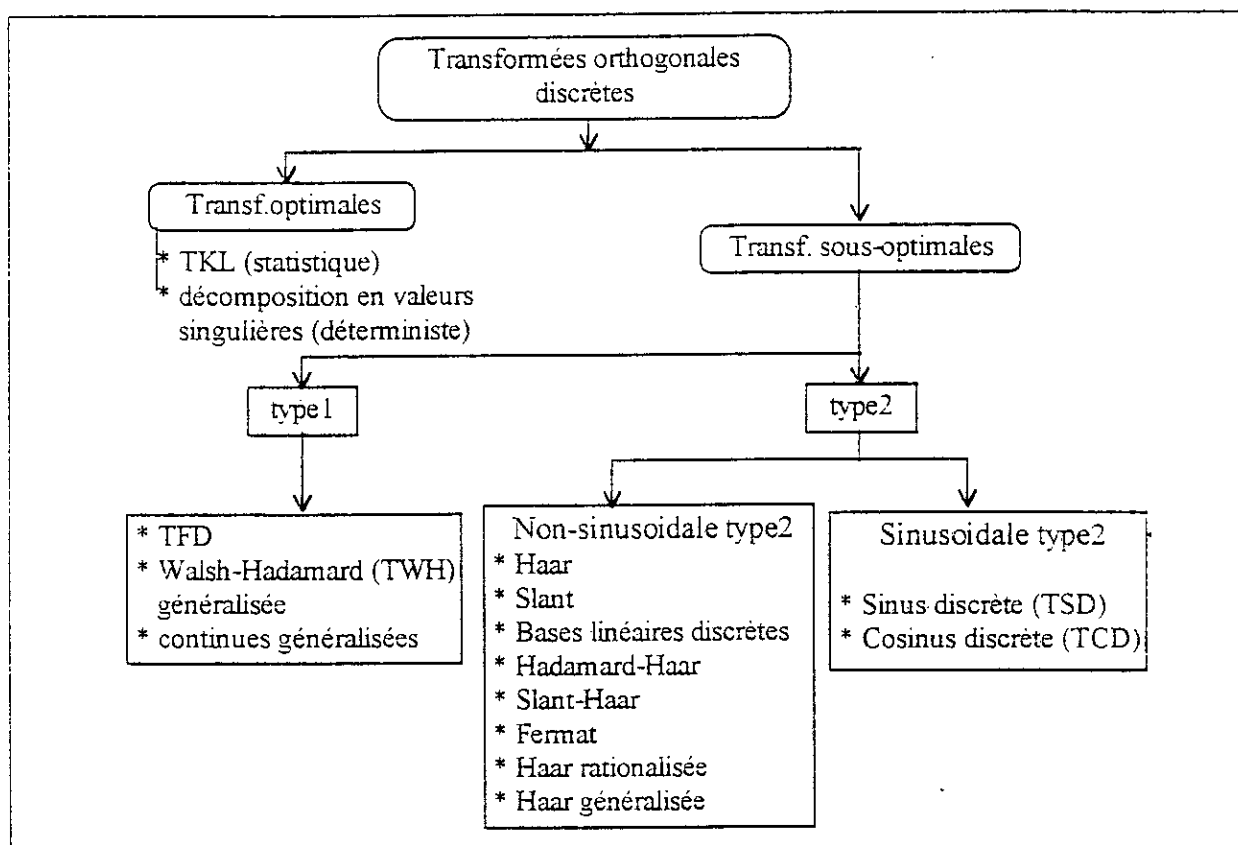


Figure I.2 - Classification des transformées orthogonales discrètes [11].

Parmi toutes ces transformées, les plus utilisées sont :

- La transformée de Hadamard-Walsh (TWH);
- La transformée de Fourier discrète (TFD);
- La transformée de Karhunen-Løve (TKL);
- La transformée Cosinus discrète (TCD).

En effet, ces transformées ont les propriétés communes suivantes:

- Les coefficients obtenus peuvent être interprétés comme des fréquences. L'information relative à la texture et au contour des objets, est contenue dans les basses fréquences et l'information relative à l'éclairage général de la scène, est contenue dans les hautes fréquences.
- Ces transformées ont plus ou moins la capacité de décorréler les pixels du bloc initial. Ceci revient à interpréter chaque coefficient en fonction de tous les pixels de l'image initiale de telle sorte que ces coefficients soient statistiquement indépendants. Ce qui permet de coder indépendamment les coefficients et d'éliminer la redondance.
- Dans le plan transformé, l'énergie est concentrée aux faibles fréquences spatiales.
- Le coefficient  $X(0,0)$  est appelé coefficient continu, et a la plus grande valeur dans un bloc.

Après la transformation, les coefficients subissent des opérations de quantification et de codage. L'ensemble {transformateur, quantificateur, codeur} ainsi que les opérations inverses constituent la chaîne de transmission appelée « CODEC » illustrée par la figure suivante :

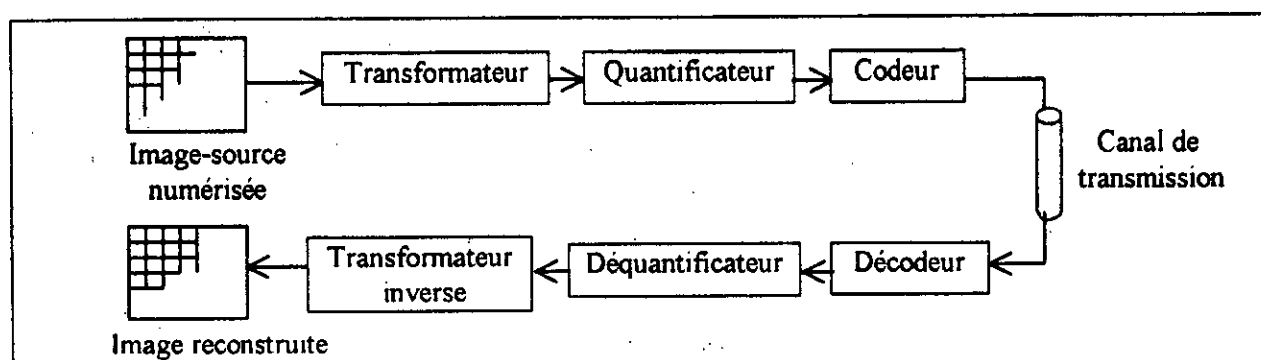


Figure I.3 - Chaîne de transmission CODEC à base de transformations orthogonales.

Le bloc transformateur opère le changement de représentation de l'image du domaine temporel vers le domaine fréquentiel spatial. Il ne fait pas de réduction de débit en lui-même mais permet de séparer l'information utile de l'information redondante. Ses performances dépendent du choix de la transformée à adopter à savoir par ordre croissant la TWH, la TFD, la TCD, la TKL. Quoique optimale, la TKL est restée à l'état de concept théorique car ses fonctions de base sont les vecteurs propres de la matrice de covariance de la séquence traitée. Quant à la TFD, elle est utilisée pour calculer la TCD.

Il reste que la TCD est la transformée adoptée par les différentes normes de compression d'image car elle est asymptotiquement équivalente à la TKL. Elle exprime chaque pixel en fonction d'une somme des coefficients multipliés par des fonction de base trigonométrique qui sont une classe des polynômes Chébyshev discrets. Son application ne dépend pas de l'image considérée et la connaissance de ses coefficients suffit pour reconstituer l'image source.

### 2.1.1.3 - CODAGE HYBRIDE [12] [13]

Cette technique combine les deux précédents : le codage prédictif et le codage par TCD.

La figure suivante montre les différentes étapes suivies.

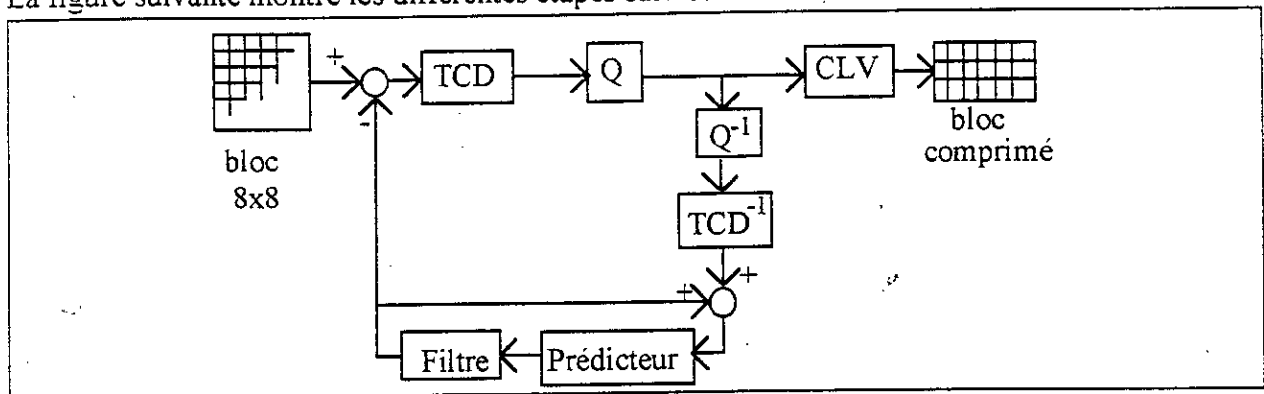


Figure I.4 - Codeur hybride avec compensation de mouvement.

La notion de « compensation de mouvement » signifie que le codec consistera à implémenter un codage prédictif « temporel » et un codage par transformée « spatial ». Cette notion sera plus explicitée dans la norme H.261 adoptée pour la visiophonie. Notons que ce codage nécessite l'utilisation de la transformée TCD et son inverse TCDI.

### 2.1.1.4 - CODAGE PAR QUANTIFICATION VECTORIELLE [14] [15]

L'image est découpée en petits blocs de pixels. Lors de la transmission, chaque bloc est comparé à un ensemble de vecteurs prédéfinis. Cet ensemble de vecteurs (dictionnaire) est présent au niveau du décodeur, et l'adresse du vecteur le plus proche est transmise. La taille des blocs utilisés, est 4x4 ou 8x8.

La difficulté de cette technique réside dans le développement d'un dictionnaire de vecteurs précis. Il est construit en choisissant statistiquement un ensemble de blocs représentatifs d'un ensemble fini d'images. L'algorithme le plus connu est l'algorithme LBG [16].

Cette technique présente plusieurs atouts, et peut être combinée avec d'autres schémas de codage pour donner un codeur plus efficace.

### 2.1.1.5 - TECHNIQUE DE TRONCATURE DE BLOC [17]

L'image est décomposée en blocs de pixels, qui seront codés indépendamment. Dans chaque bloc, la moyenne « m » et l'écart-type « e » sont calculés. Ces derniers sont quantifiés puis transmis avec une matrice de signes indiquant pour chaque point du bloc s'il se trouve au dessus ou au dessous de cette moyenne. La valeur restituée est (m-e) ou (m+e) selon les cas.

### 2.1.1.6 - CODAGE PAR INTERPOLATION [4]

Cette technique envoie seulement un sous-ensemble de pixels de l'image. A la réception, une interpolation est faite pour retrouver les pixels non transmis et ainsi reconstruire l'image tout entière.

## **2.1.2 - TECHNIQUES SANS DISTORSIONS**

### **2.1.2.1- TECHNIQUE DES PLAGES [18] [19]**

Sur une ligne de la matrice représentant une image, plusieurs échantillons peuvent avoir la même valeur. L'ensemble de ces échantillons est appelé « plage ». Cette technique consistera à décrire ces suites de pixels identiques par leur longueur et leur valeur. Par exemple, une suite de 10 pixels noirs sera décrite par deux nombres : 10 et 0.

Cette méthode est d'autant plus efficace que le nombre de niveaux de gris possibles, est faible.

### **2.1.2.2 - TECHNIQUE DES PLANS DE BITS [18] [20]**

Dans cette technique, les pixels d'une image numérique  $N \times N$  sont codés sur  $M$  bits. Le  $j^{\text{ème}}$  bit de chaque échantillon est isolé pour obtenir  $M$  images binaires de dimension  $N \times N$ , appelées « plans de bits ». On pourra appliquer à chacune d'elles, des méthodes de compression spécialement développées pour des images binaires.

### **2.1.2.3 - TECHNIQUES STATISTIQUES [7] [21]**

Ces techniques dites ponctuelles font associer à chaque pixel de l'image un mot de code dont la longueur dépend de la probabilité d'apparition du niveau de gris correspondant. Les techniques de codage les plus performantes sont celles de Huffman et Shannon-Fano.

Ces méthodes sont connues sous le nom de méthodes VLC ( variable length cod ). Elles peuvent être combinées aux méthodes de codage prédictif ou par transformée.

## **2.2 - STANDARDISATION EN CODAGE D'IMAGES**

Les travaux de normalisation peuvent être classés en quatre parties suivant les services voulus et les applications :

- Transmission et stockage des images fixes;
- Visiophonie, visioconférence et visioreunion;
- Applications T.V. et stockage sur support numérique;
- Réseaux de communications multimédias .

Notons qu'actuellement, beaucoup de laboratoires ne travaillent plus sur le côté algorithmique mais axent leurs recherches sur l'implémentation d'algorithmes existants répondant aux spécifications de normalisation, sur des architectures intégrées. On trouve essentiellement trois (03) normes adoptées pour la compression d'image :

- JPEG pour la compression des images fixes;
- H.261 pour l'algorithme de codage pour la visiophonie;
- MPEG pour la compression d'images animées.

## 2.2.1 - APERÇU SUR LA NORME JPEG [22]

La norme JPEG ( Joint Photographic Experts Group ) est destinée pour le stockage et la restitution des images fixes, monochromes ou couleurs, sur les supports numériques pour des applications comme le fac simili, la vidéotex, la conférence audiographique ...etc. Ses spécifications portent sur les éléments suivants :

- les procédures de conversion de l'image source en image comprimée;
- les procédures de conversion de l'image comprimée en image reconstruite;
- un aperçu sur l'implémentation de ces procédures sur le plan architectural;
- les différentes méthodes à choisir pour le codage et le décodage à savoir :
  - le codage par la transformée orthogonale discrète TCD;
  - le codage prédictif « sans pertes ».

Cette norme considère qu'une image numérique est constituée de plusieurs composants; l'image couleurs en possède trois : un composant pour la luminance et deux pour la chrominance; tandis que l'image monochrome n'a qu'un seul composant. Pour le codage à base de TCD, un échantillon est une matrice carrée de 8x8 pixels, dite bloc. Tandis que pour le codage prédictif, les échantillons sont simplement les pixels.

Dans ce qui suit, nous considérons une image monochrome et pour l'image multi-composants, nous donnerons à la fin la procédure de son codage.

### 2.2.1.1 - Propriétés

- Un pixel est un entier à valeur prise dans le domaine  $[0, 2^{P-1}]$  avec une précision P de 8 bits, soit un domaine de valeurs possibles des pixels :  $[0, 255]$ .
- Le balayage de l'image (du composant) se fait de gauche à droite et de haut en bas.

### 2.2.1.2 - Le codage par la transformée TCD

La norme JPEG spécifie deux (02) méthodes basées sur la TCD :

- Le codage séquentiel ;
- Le codage progressif .

Les figures suivantes donnent les chaînes de codage et de décodage par la TCD.

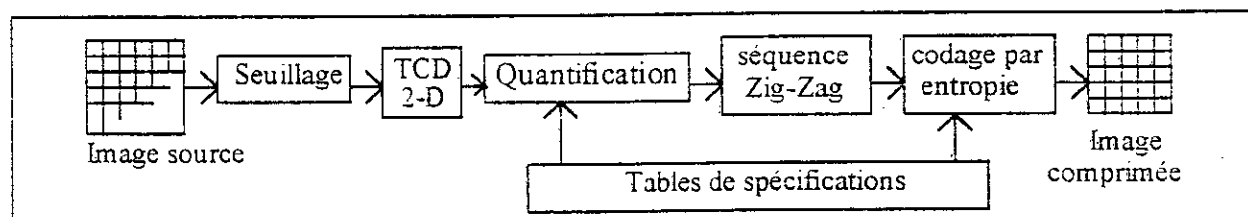


Figure I.5 a - Chaîne de codage par TCD

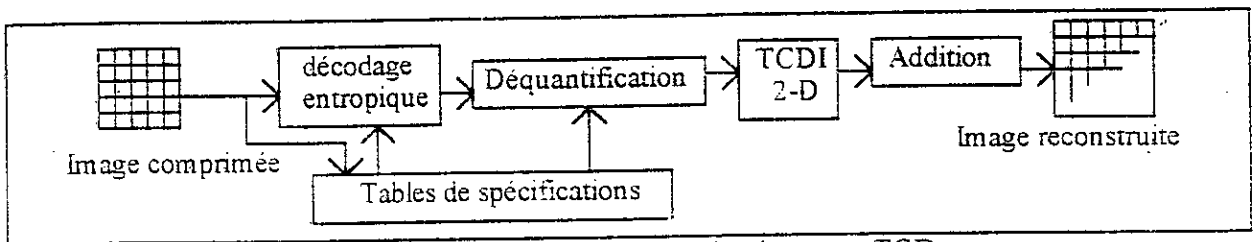


figure I.5.b - Chaîne de décodage par TCD

### 2.2.1.2.a - Codage par TCD séquentielle

Les échantillons par blocs 8x8, sont introduits de gauche à droite, rangée par rangée et de haut en bas. Après que le bloc ait été quantifié, les 64 coefficients peuvent être codés immédiatement puis transmis.

### 2.2.1.2.b - Codage par TCD progressive

Les blocs sont introduits de la même façon que précédemment, mais le codage se fait en plusieurs balayages. Ceci est fait avec l'insertion entre le quantificateur et le codeur d'un buffer pour mémoriser l'image.

Il existe deux (02) procédures de codage des coefficients quantifiés dans le buffer :

- Une sélection « spatiale » tenant compte de la répartition fréquentielle des coefficients, est faite dans le buffer. Une bande précise du spectre de fréquence, est envoyée à la fois vers le codeur.
- Les coefficients quantifiés de la bande de fréquence sélectionnée, sont partiellement codés en commençant par les bits MSB. Les bits LSB sont codés ultérieurement lors de balayages multiples.

### 2.2.1.2.c - Propriétés

- **Le seuillage** : il consiste à décaler avant le codage, les valeurs du domaine  $[0, 2^P]$  au domaine  $[-2^{P-1}, 2^P]$  en soustrayant la valeur  $2^{P-1}$  à chaque coefficient. Cette opération est d'un grand intérêt du fait qu'il introduit une représentation signée qui sera mise à contribution lors du codage entropique. Au décodage, la quantité  $2^{P-1}$  est bien sûr rajoutée aux valeurs sortantes de la TCDI pour restituer l'image source.
- **Le balayage** : L'orientation dans un bloc se fait de gauche à droite, et de haut en bas.
- **La quantification** : A la sortie du bloc TCD, chacun des 64 coefficients  $Y(m,n)$  obtenus, se présente à la quantification. Celle-ci est uniforme pour tous les blocs de l'image. A chaque coefficient  $Y(m,n)$ , correspond un pas de quantification  $Q_{m,n}$ .

Les coefficients quantifiés  $Y_q(m,n)$  sont définis par :  $Y_q(m,n) = \text{Round} \{ Y(m,n) / Q_{m,n} \}$  où « Round » est la fonction « entier le plus près ». L'opération de quantification peut être subdivisée en deux (02) phases :

- Normalisation des coefficients pour ajuster leur dynamique :  $Y_N(m,n) = Y(m,n) / Q_{m,n}$  ;
- Application de la fonction « Round » : entier  $[ Y_N(m,n) + 0.5 ]$  .



- Au décodage, la déquantification se fera comme suit :  $R(m,n) = Y_N(m,n) * Q_{m,n}$  où  $R(m,n)$  sont les coefficients déquantifiés.
- **Le codage différentiel de la composante continue DC :** On note par DC, la première valeur  $Y(0,0)$  du bloc  $8 \times 8$  transformé. Cette valeur représente la composante continue du signal variant dans l'espace ( luminance ou chrominance ). Après la quantification, le coefficient DC quantifié (  $Y_q(0,0)$  ) est traité séparément des 63 autres coefficients du bloc. notés AC pour composantes alternatives. Le codage de  $Y_q(0,0)$  se fera de manière différentielle. En effet, vu que le coefficient DC varie peu d'un bloc à un autre, il est préféré de coder la différence par la technique MICD.
  - **La séquence zigzag :** Du fait de la propriété de la TCD de concentration du maximum d'informations aux basses fréquences, les 63 coefficients AC quantifiés sont convertis en une séquence zigzag notée ZZ, pour les positionner dans un vecteur ligne et les préparer au codage entropique. Ainsi on peut avoir des séquences de zéros qui faciliteront le codage.
  - **Le codage par entropie :** Les séquences de codage différentiel pour le DC et RUN-LENGTH pour les AC, sont comprimées à l'aide du codage par entropie. La norme JPEG spécifie deux (02) sortes de codage qui permettent de réduire l'entropie:
    - Le codage par la méthode de Huffman;
    - Le codage arithmétique.

Le codage de Huffman est préféré au second du fait de sa simplicité d'implémentation. Chaque valeur prête au codage par entropie, possède deux (02) informations : sa valeur exacte et la longueur de ses bits. Une étude statistique permet de regrouper les valeurs spécifiques au nombre de leurs bits, c-à-d que deux valeurs au même nombre de bits, sont d'une même catégorie. La catégorie, ayant la plus grande probabilité d'apparition, est codée par le minimum de bits, et celle qui a la faible probabilité d'apparition, est codée par un maximum de bits. Deux algorithmes génèrent deux tables de codage: le code de la catégorie noté Huff size, et celui de la longueur associé noté Huff code.

### 2.2.1.3 - Image à plusieurs composants

La procédure d'échantillonnage illustrée ci-dessous, est dite « interchangeable ». En effet, pour certaines images, nous ne pouvons pas coder complètement un composant puis passer à un autre. L'échantillonnage interchangeable signifie qu'on code un échantillon du composant A, puis on passe à un échantillon similaire du composant B, puis on revient à un autre échantillon de A, et ainsi de suite. Il faudra que les composants constituant l'image, aient la même dimension, le même nombre de lignes, et le même nombre de colonnes. Les tables de spécifications étant différentes d'un composant à un autre.

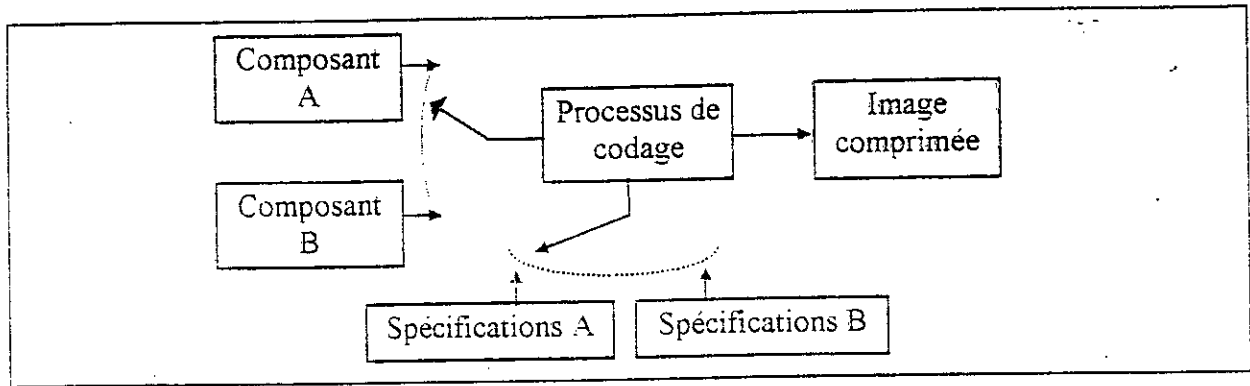


Figure I.6 - Procédure d'échantillonnage d'une image multi-composants.

## 2.2.2 - APERCU SUR LE STANDARD H.261 [23] [24]

L'un des premiers standards pour la représentation de l'image animée, est celui du CCITT H.120 qui décrit les versions 625 lignes 50 trames/s et 525 lignes 60 trames/s du codec pour la vidéoconférence. Le H.261 est le standard destiné pour le codage d'images animées pour la visioconférence, connu sous le nom de standard à  $p \times 64$  Kbit/s pour ses débits variant de 64 Kbit/s à 2 Mbit/s par pas de 64 Kbit/s. Il utilise un codage dit « hybride », qui combine les techniques de codage par TCD pour réduire la redondance spatiale (codage intra-trames), et le codage prédictif pour réduire la redondance temporelle (codage inter-trames).

### 2.2.2.1 - Codeur

Le codeur hybride donné précédemment par la figure I.4, réalise les fonctions suivantes :

- L'acquisition numérique de l'image (changement de standard, sous-échantillonnage temporel,...)
- La prédiction avec compensation de mouvement et filtrage dans la boucle. Les images animées ont en effet une forte corrélation temporelle, la prédiction estime la valeur du signal à transmettre afin de n'en coder que la partie innovatrice.
- Les transformées TCD et TCDI.
- La quantification des coefficients transformés et le codage entropique des données (pour n'en conserver que ceux qui sont significatifs).
- Le multiplexage vidéo des informations (synchronisation, types de codage, vecteurs de mouvement, coefficients...).

### 2.2.2.2 - Codage par transformée

Dans le modèle de référence adopté par le groupe XV du CCITT, une TCD 2-D est utilisée avec des blocs 8x8 pixels. La normalisation de l'implémentation de l'algorithme TCDI est nécessaire vu les problèmes de non-correspondance de la transformée, dus à l'utilisation de deux circuits différents pour la TCD dans le codeur et le décodeur. Pour les circuits de calcul de la TCD, 9 bits sont utilisés en entrée et 12 bits en sortie.

### 2.2.2.3 - Quantification :

La quantification, étape importante après la TCD, sert à réduire la redondance et à représenter les coefficients par un mot binaire de longueur finie, en tenant de :

- la définition d'un critère convenable pour obtenir les valeurs de quantification.
- l'exploitation de la corrélation résiduelle parmi les variables.
- la sélection des paramètres de quantification suivant des critères psychovisuels.
- le moyen de contrôler la quantité de données produites par les variations de tels paramètres.

Cette opération fait que les coefficients quantifiés sont tronqués, produisant ainsi un bruit. La réduction de ce bruit et la minimisation de la distorsion sont nécessaires.

### 2.2.2.4 - Types de balayages

Après la quantification, les coefficients doivent être transmis avec un moyen efficace. La puissance des coefficients est usuellement concentrée dans les séquences d'ordre faible. Le bloc quantifié comprend des coefficients nuls et des coefficients non nuls. Un moyen pour éliminer la dernière chaîne de composantes nulles consiste à appliquer une méthode prédéfinie pour transmettre les coefficients, appelée « balayage » en combinaison avec un code de « fin de bloc ».

Il existe plusieurs types de balayages:

- Balayage en zigzag qui est le plus prometteur;
- Balayage vertical;
- Balayage horizontal;
- Balayage diagonal.

Deux balayages sont fait pour limiter le nombre de coefficients à transmettre :

- Le premier balayage donne une indication sur le nombre de coefficients nuls et non nuls;
- Le second ( dans le sens inverse) retrouve le dernier coefficient non nul. Un flag est mis à 1 pour indiquer que le dernier coefficient non nul est retrouvé, et commencer le bloc suivant.

### 2.2.2.5 - Estimation de mouvement

L'estimation de mouvement est basée sur l'hypothèse que des pixels adjacents ont les mêmes paramètres de déplacement. Un vecteur de mouvement est obtenu en faisant correspondre une fenêtre de mesure rectangulaire, comportant un certain nombre de pixels adjacents avec une fenêtre de mesure correspondante dans l'image suivante. La correspondance est réalisée en recherchant l'extrême d'un critère par exemple la moyenne de la différence absolue de l'image déplacée. L'efficacité de l'estimation dépend de la taille des fenêtres, conjointement avec la quantité actuelle du mouvement. Pour traiter de grands déplacements, il faut des fenêtres de

grandes taille. La méthode adoptée n'est pas normalisée, seule la procédure de transmission des vecteurs de mouvement, est normalisée.

Un bon algorithme d'estimation de mouvement est caractérisé par une bonne précision, un faible effort de calcul, un vrai vecteur de mouvement, et une faible quantité d'information à transmettre. En principe pour bien estimer le mouvement, deux objectifs sont à considérer :

- Une prédiction compensée de mouvement;
- Une interpolation (ou extrapolation) compensée de mouvement.

Dans le premier cas, la précision de l'estimation par des techniques basées sur la correspondance de blocs et un post-traitement (filtre) sont raisonnables. Cependant pour l'interpolation des images « sautées », il est important d'utiliser des « vrais » vecteurs qu'on peut obtenir en utilisant une procédure de correspondance de blocs.

#### 2.2.2.6 - Filtre de la boucle

Afin de réduire l'erreur de prédiction, un filtre passe-bas est introduit dans la boucle de codage après la compensation de mouvement. Ceci pour réduire les effets de haute fréquence introduits par la compensation de mouvement, et diminuer le bruit de quantification dans la boucle de retour.

#### 2.2.2.7 - Codage à longueur variable ( CLV )

Le CLV est un outil efficace qui utilise la redondance statistique dans le signal transmis. Il utilise l'algorithme de Huffman. La TCD, la compensation de mouvement, et le filtre de la boucle sont conçus de façon à générer un signal avec une redondance statistique suffisante pour que le CLV produise un débit binaire faible.

### 2.2.3 - APERCU SUR LE STANDARD MPEG [25]

Les efforts des équipes du CCITT pour le H.261, sont la base du développement du standard de codage d'images animées par l'ISO: MPEG pour « Moving Picture Experts Group ». Contrairement au H.261, en 1<sup>ère</sup> phase, MPEG est destiné au codage d'images animées, en vue de leur stockage sur des supports numériques, d'où des contraintes plus souples.

La 1<sup>ère</sup> version MPEG-1 spécifie une compression vidéo à un débit de 1.5 Mbit/s. Deux autres versions visent à améliorer la qualité du codage vidéo avec une augmentation de débit : MPEG-2 est destiné à des débits de l'ordre de 10 Mbit/s, et MPEG-3 destiné à la TVHD à des débits de 30 à 40 Mbit/s. La dernière version MPEG-4 concerne le codage d'images animées à très bas débit, l'objectif étant d'obtenir un « standard » tenant compte de toutes les contraintes liées aux applications envisageables.

L'algorithme de compression vidéo MPEG se base sur deux techniques :

- La compensation de mouvement basée sur des blocs pour réduire la redondance temporelle;
- La compression basée sur la TCD pour la réduction de la redondance spatiale.

Les techniques de compensation de mouvement sont appliquées avec des prédicteurs causaux (codage prédictif pur) et des prédicteurs non-causaux (codage interpolatif). L'erreur de prédiction est comprimée avec la TCD; et l'information relative au mouvement est basée sur des blocs 16x16, comprimée avec des CLVs, et transmise avec l'information spatiale.

### 2.2.3.1 - Réduction de la redondance temporelle.

L'une des contraintes du MPEG, est l'accès aléatoire aux séquences d'images. Ceci a conduit à spécifier trois (03) types d'images (Figure I.7) :

- des images Intras (**I**) : qui fournissent des points d'accès aléatoires toutes les 10 à 15 images;
- des images Prédites (**P**) : codées avec référence à des images (I) ou (P) passées, et utilisées en général comme référence pour des images (P) futures.
- Des images bidirectionnelles (**B**) : prédites bidirectionnellement avec à la fois deux références passée et future de type (I) ou (P), et ne servent jamais de référence.

La quantité d'informations contenues dans chacune de ces images décroît du type (I) à (P) à (B) à l'inverse de la compression subite.

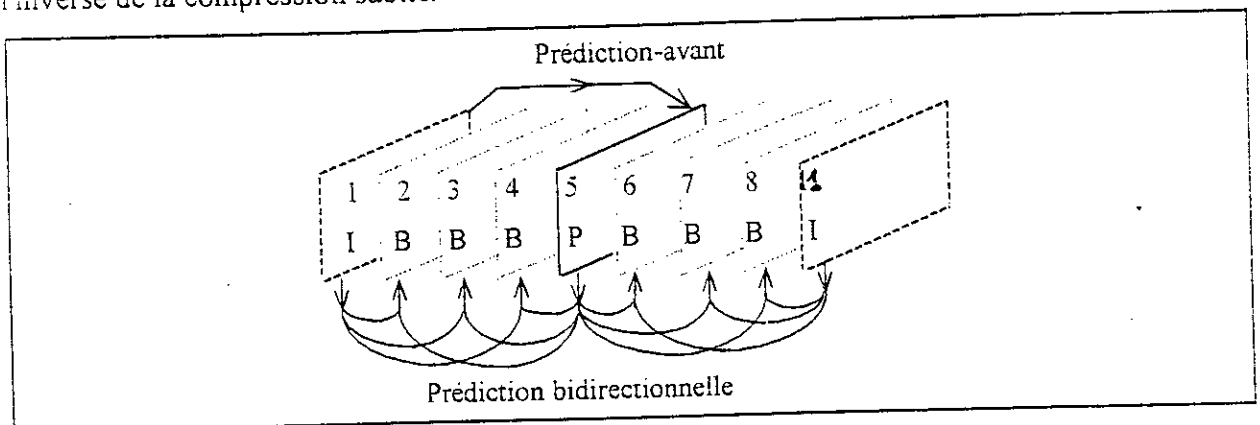


Figure I.7 - Codage intertrame.

### 2.2.3.2 - Compensation de mouvement

- **Compensation par prédiction** : Cette technique assume que « localement » l'image courante peut être modélisée par la translation de l'image qui la précède. Localement signifie que l'amplitude et la direction du déplacement ne sont pas nécessairement les mêmes partout dans l'image. Vu son importance, l'information du mouvement doit être codée adéquatement.
- **Compensation par interpolation** : Cette technique améliore l'accès aléatoire aux séquences d'images, réduit l'effet des erreurs et contribue de façon significative à une bonne qualité de l'image. Elle est dite aussi prédiction bidirectionnelle et possède nombre d'avantages :

- Le taux de compression obtenu est élevé;
- Elle a de bonnes propriétés statistiques permettant en particulier de diminuer l'effet de bruit par un moyennage entre l'image précédente et l'image qui suit;
- Elle permet un découplage entre la prédiction et le codage ( pas d'erreur de propagation );
- Le protocole d'échange associé à la fréquence des images de type (B) est le suivant:
 

« En augmentant le nombre d'images (B) entre les images de références, on diminue la corrélation entre ces dernières et les images (B), ainsi que la corrélation entre les références elles-mêmes. Pour une large classe d'images, les combinaisons sont du type: IBBPBBPBB...IBBPBB ».

### 2.2.3.3 - Estimation de mouvement

Le compromis entre le gain du codage de l'information du mouvement seul et le coût associé, a conduit au choix de macro-blocs (M-bloc) 16x16 comme unité de compensation de mouvement. Dans le cas général d'une image(B), la prédiction des M-blocs dépend des références passées ou futures et des vecteurs de mouvement.:

MPEG spécifie comment représenter l'information de mouvement : un (01) vecteur pour les M-blocs de prédiction avant ou arrière, et deux (02) vecteurs pour les M-blocs de prédiction bidirectionnelle. Les vecteurs de mouvement sont calculés en minimisant une fonction coût mesurant la différence entre un bloc et son prédicteur. L'information de mouvement associée à tout M-bloc 16x16 est codée différentiellement avec celle du bloc précédent pour subir ensuite un codage CLV pour plus d'efficacité.

### 2.2.3.4 - Réduction de la redondance spatiale

On utilise la technique de compression intra-trame par TCD avec une combinaison d'une quantification visuelle adaptative à un codage RUN-LENGTH. L'opération consiste en 3 étapes:

- calcul des coefficients TCD;
- quantification des ces coefficients;
- conversion des coefficients en paires {run - amplitude} après leur réorganisation zigzag.

#### • Quantification :

La quantification adaptative assure une bonne qualité visuelle. Elle peut être grossière aux hautes fréquences et presque uniforme aux faibles fréquences avec une différence de comportement du quantificateur autour de zéro (0) selon le type de blocs. Pour les blocs codés intras, le quantificateur ne présente pas de zone-limite (i.e. la région quantifiée au niveau zéro, est plus petite que le pas de quantification), par contre pour les blocs non-intras (différentiels), la zone-limite est large. Le pas de quantification pouvant être modifié aux limites des blocs.

• **Codage entropique :**

Pour augmenter la compression par TCD et réduire l'impact de l'information de mouvement sur le taux de compression, un codage CLV est spécifié. Une table de Huffman pour les coefficients TCD, est utilisée pour coder les paires d'événements {run - amplitude}. Seuls les événements à grande probabilité d'apparition sont codés par le CLV.

Le schéma du codec MPEG est donné par la figure suivante :

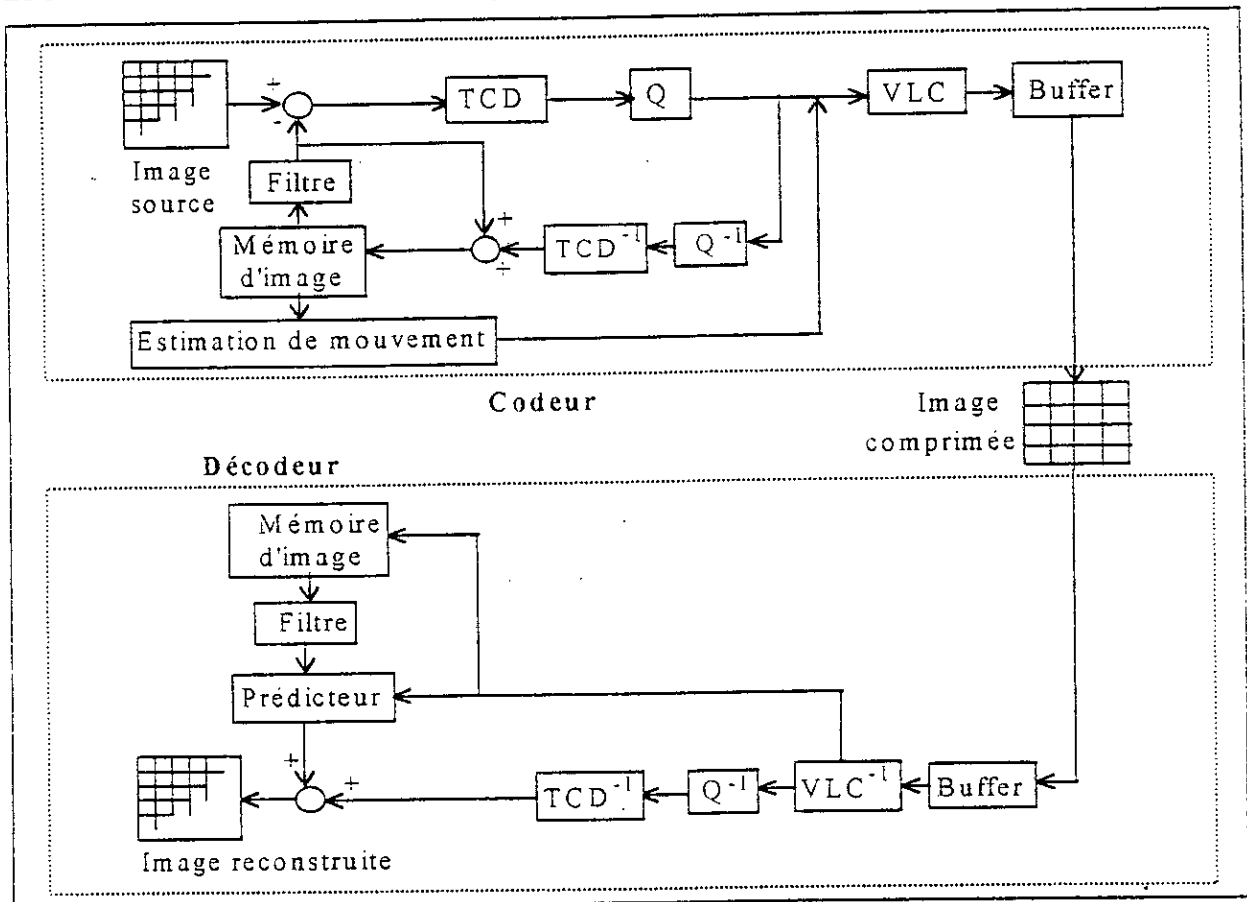


Figure I.8 - CODEC MPEG

**3 - CONCLUSION**

Dans ce chapitre, nous avons présenté l'essentiel des techniques de compression d'images illustrées par des figures explicatives pour mieux cerner leurs performances et leurs particularités. La technique de codage par transformée orthogonale TCD permet de grands et meilleurs taux de compression, et résiste mieux aux erreurs [5]. Les techniques prédictives quant à elles, sont efficaces pour des images à faibles évolutions temporelles et/ou spatiales.

Cependant, l'inconvénient majeur des systèmes de compression par transformée TCD réside dans leur complexité d'implémentation vu qu'ils nécessitent des opérations de transformation directe et inverse avant et après le codage selon les différents standards dans ce domaine.

La suite de notre travail sera orientée vers la dérivation d'un algorithme hybride TCD/TCDI et la conception d'une architecture unifiée pour l'implémentation de la transformée orthogonale cosinus discrète et son inverse.

## CHAPITRE II



LA TRANSFORMÉE  
COSINUS DIRECTE  
T.C.D



## 1 - INTRODUCTION

Les applications modernes utilisant l'imagerie, adoptent un codage par la transformée orthogonale TCD opérant sur des blocs d'image, bidimensionnels dont la taille est appréciée à 8x8 pixels. L'importance et les performances élevées attendues de telles applications, qui sont des marchés en grande diffusion, exigent le développement d'algorithmes rapides de calcul de la TCD, tenant compte de toutes les spécifications des standards de normalisation, dans le sens que leur implémentation est devenue l'élément-clé de la VLSI de compression d'image.

Dans ce qui suit, nous présentons l'essentiel des algorithmes TCD proposés avec diverses techniques de dérivation réduisant la complexité des calculs et augmentant la vitesse d'exécution. Notre but est de dériver et/ou choisir un algorithme efficace de calcul hybride TCD/TCDI 2-D nous permettant la conception d'une architecture unifiée TCD/TCDI.

## 2 - ALGORITHMIQUE RAPIDE DE LA TCD

Il convient de rappeler que la plupart des algorithmes rapides des transformées de signal telles que la TFR, la TCD, la WHT...etc, sont basées sur une technique dite de « Divide and Conquer » (soit diviser pour régner), qui consiste à transformer le problème initial en plusieurs problèmes du même type, plus petits. Des outils mathématiques jusqu'ici peu utilisés en traitement de signal : algèbre des corps finis, mathématiques discrètes dont le principal promoteur est Winograd, permettent de minimiser le nombre de multiplications à implanter et dont le coût est beaucoup plus important que celui des additions.

### 2.1 - DÉFINITIONS [3] [11] [26] [27]

#### 2.1.1 - TCD unidimensionnelle

Pour une séquence de données entrées  $X(i)$ ,  $i = 0, \dots, N-1$ , la séquence de coefficients TCD est donnée par :

$$Y(m) = (2 / N) \cdot u(m) \cdot \sum_{i=0}^{N-1} X(i) \cdot \cos[(2i + 1)m\pi / 2N]; \quad m = 0, \dots, N-1. \quad (II.1)$$

$$\text{où : } u(m) = \begin{cases} 1 / \sqrt{2} & \text{si } m = 0 \\ 1 & \text{sinon} \end{cases}$$

Les coefficients de la transformée inverse TCDI sont donnés par :

$$X(i) = (2 / N) \cdot \sum_{m=0}^{N-1} u(m) \cdot Y(m) \cdot \cos[(2i + 1)m\pi / 2N]; \quad i = 0, \dots, N-1. \quad (II.2)$$

**2.1.2 - TCD bidimensionnelle**

Pour une séquence de données entrées  $X(i,j)$ ,  $i, j = 0, \dots, N-1$ , la séquence de coefficients TCD 2-D  $Y(m,n)$ ,  $m,n = 0, \dots, N-1$ , est donnée par :

$$Y(m,n) = (2 / N)^2 \cdot u(m) \cdot u(n) \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X(i,j) \cdot \text{Cos}[(2i + 1)m\pi / 2N] \cdot \text{Cos}[(2j + 1)n\pi / 2N] \quad (\text{II.3})$$

avec :  $u(m), u(n) = \begin{cases} 1 / \sqrt{2} & \text{si } m, n = 0 \\ 1 & \text{sinon} \end{cases}$

Les coefficients  $X(i,j)$ ,  $i, j = 0, \dots, N-1$ , de la transformée inverse TCDI 2-D sont donnés par :

$$X(i,j) = (2 / N)^2 \cdot \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} u(m) \cdot u(n) \cdot Y(m,n) \cdot \text{Cos}[(2i + 1)m\pi / 2N] \cdot \text{Cos}[(2j + 1)n\pi / 2N] \quad (\text{II.4})$$

**2.2 - PROPRIETES**

- La TCD est une bonne approximation de la TKL car leurs fonctions de base sont très proches;
- Les coefficients obtenus par la TCD sont hautement décorrélés, et le maximum d'information est concentré dans les coefficients de faibles fréquences (premier quart du bloc obtenu ).
- La TCD est une transformée linéaire;
- La TCD 2-D est séparable en TCDs 1-D :

En effet, l'équation (II-3) peut être réécrite comme suit :

$$Y(m,n) = (2 / N) \cdot u(m) \sum_{i=0}^{N-1} \left\{ (2 / N) \cdot u(n) \sum_{j=0}^{N-1} X(i,j) \cdot \text{cos}((2j + 1)n\pi / 2N) \right\} \cdot \text{cos}((2i + 1)m\pi / 2N) \quad (\text{II-5-a})$$

$$= (2 / N) \cdot u(m) \sum_{i=0}^{N-1} X'(i,m) \cdot \text{cos}((2i + 1)m\pi / 2N)$$

où  $X'(i,m) = (2 / N) u(n) \sum_{j=0}^{N-1} X(i,j) \cdot \text{cos}((2j + 1)n\pi / 2N) \quad (\text{II-5-b})$

$X'(i,m)$  représente la TCD 1-D de la séquence  $\{x(i,j) / i \text{ fixe}; j = 0 : N-1\}$ .

Cette séquence représente la ligne  $i$  du bloc  $N \times N$  pixels. Nous obtenons donc  $N$  vecteurs  $X'(i,m)$ ,  $i = 0 : N-1$ , chacun de  $N$  éléments ( $m = 0 : N-1$ ).

En faisant une transposition  $X'(i,m)^T = X'(m,i)$ , nous obtenons :

$$Y(m,n) = (2 / N) u(m) \sum_{i=0}^{N-1} X'(m,i) \cdot \text{cos}((2i + 1)m\pi / 2N) \quad (\text{II-6})$$

Cette relation donne la TCD 1-D des lignes obtenues après transposition des colonnes  $X'(i,m)$ .

Le graphe de fluence de la TCDI est le même que celui de la TCD mais parcouru dans le sens inverse. En effet, soit l'écriture matricielle de la TCD 1-D:  $Y = (A) \cdot X$  où  $A$  est la matrice TCD  $N \times N$  pixels. La transformée inverse TCDI 1-D est définie par :  $X = (A)^{-1} Y$ .

Or la TCD est une transformée orthogonale à noyaux réels [11], d'où :  $(A)^{-1} = (A^t)^* = (A)^t$  (II-7) les indices « t » et « \* » signifient respectivement « transposée » et « complexe conjuguée ».

Par suite, la matrice inverse TCDI n'est que la matrice directe transposée.

### 2.3 - ALGORITHMES TCD 2-D

Plusieurs algorithmes TCD 2-D ont été proposés pour réduire la complexité des calculs et augmenter la vitesse d'exécution, certains basés sur une approche conventionnelle dite de *ligne colonne*, et les autres, plus récents basés sur *une approche directe*.

Mais, il reste que les performances de ces algorithmes implantés sur des architectures, reposent comme nous le verrons plus loin sur l'efficacité des calculs faits à l'aide d'algorithmes TCD 1-D.

#### 2.3.1 - APPROCHE LIGNE / COLONNE

Cette approche repose sur la propriété de séparabilité de la TCD 2-D en TCDs 1-D. Elle considère les données entrées rangée par rangée comme N vecteurs de N pixels chacun. Ces vecteurs sont fournis à une première unité de traitement TCD 1-D pour calculer leurs transformées dans l'ordre des lignes. Les résultats intermédiaires sont transposés dans une RAM, et fournies à une seconde unité de traitement TCD 1-D et traitées dans l'ordre des colonnes.

La TCDI est établie dans le sens inverse du fait que son graphe de fluence est le même que celui de la TCD, mais parcouru dans le sens inverse. La configuration générale de cette approche est donnée par la figure suivante:

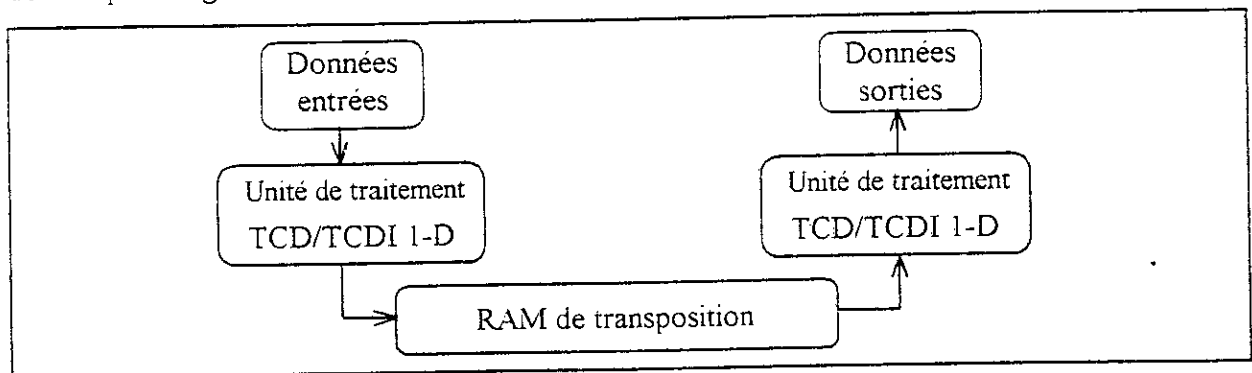


Figure II.1 - Configuration générale de l'approche ligne/colonne.

Nous constatons que :

- pour calculer une TCD  $N \times N$  points, cette méthode requiert  $2 \times N$  TCDs  $N$  points;
- les performances de cette méthode dépendent de la réalisation des calculs TCDs 1-D;
- Cette méthode sied bien à l'implantation d'algorithmes TCD 1-D et son inverse TCDI 1-D sur un même circuit vu qu'elles partagent le même graphe de fluence.

Beaucoup d'algorithmes adoptent cette approche en lui associant différentes arithmétiques telles que l'arithmétique distribuée [28] et les modes de calcul On-line et Half-line [29].

**2.3.2 - APPROCHE DIRECTE**

Pour une implémentation parallèle et un calcul plus efficace de la TCD 2-D, des algorithmes travaillant directement sur l'ensemble des données 2-D ont été proposés [26], [27], [30]-[33]. Ces algorithmes réduisent plus ou moins bien le nombre de multiplications mais la procédure générale de leur implémentation reste plus compliquée que celle de l'approche conventionnelle.

L'un des algorithmes TCD 2-D les plus efficaces est l'approche polynomiale directe proposée par P. Duhamel [33] dans laquelle le nombre de multiplications est réduit de 50% de celui de l'approche conventionnelle. Les algorithmes de M.A. Haque [30] et C. Ma [31] en requièrent 75% et l'approche indirecte utilisant la transformation polynomiale FFT et des rotations, proposée par Vetterli [32], en exige entre 50 et 75%.

Mais, il reste que l'algorithme rapide TCD 2-D le plus attrayant de cette catégorie, est celui proposé par N.I.Cho et S.U.Lee [34] du fait qu'en plus de réduire le nombre de multiplications, il offre une structure de calcul hautement plus régulière. Pour le calcul d'une TCD NxN-points, cet algorithme présenté ci-après, ne requiert que N modules TCDs N-points, sinon N/2 modules si l'on utilise des étages de multiplexage.

**2.3.2.1 - ALGORITHME RAPIDE TCD 2-D DE N.I.CHO ET S.U.LEE [34]**

Cet algorithme ne sera pas retenu pour un calcul hybride TCD/ TCDI pour les raisons que nous verrons plus loin. Néanmoins, il nous a paru utile de le présenter pour sa régularité et son efficacité dans les calculs.

Pour une séquence d'entrées 2-D  $\{x(i, j) : i, j = 0, \dots, N-1\}$  donnée, la séquence TCD 2-D  $\{Y(m, n) : m, n = 0, \dots, N-1\}$  s'écrit :

$$Y(m, n) = (2/N)^2 u(m)u(n) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cdot \cos((2i + 1)m\pi / 2N) \cdot \cos((2j + 1)n\pi / 2N) \quad (\text{II-8})$$

$$\text{où : } u(m), u(n) = \begin{cases} 1 / \sqrt{2} & \text{si } m, n = 0 \\ 1 & \text{sinon} \end{cases}$$

En négligeant le facteur d'échelle  $(2/N)^2 u(m)u(n)$ , la forme dénormalisée de  $Y(m, n)$  s'écrit :

$$y(m, n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cdot \cos((2i + 1)m\pi / 2N) \cdot \cos((2j + 1)n\pi / 2N) \quad (\text{II-9-a})$$

$$\text{soit : } Y(m, n) = (2/N)^2 \cdot u(m) \cdot u(n) \cdot y(m, n) \quad (\text{II-9-b})$$

L'idée principale est que la TCD NxN peut être décomposée en deux transformées  $A(m, n)$  et  $B(m, n)$  en utilisant la relation suivante:

$$\cos(2i + 1)m\pi / 2N \cdot \cos(2j + 1)n\pi / 2N = 1/2 \cdot \left[ \cos\{((2i + 1)m + (2j + 1)n)\pi / 2N\} + \cos\{((2i + 1)m - (2j + 1)n)\pi / 2N\} \right] \quad (\text{II-10})$$

Nous écrivons alors:

$$y(m, n) = (1/2) \cdot (A(m, n) + B(m, n)) \quad (II-11)$$

$$\text{avec : } \begin{cases} A(m, n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cdot \cos\{((2i+1)m + (2j+1)n)\pi / (2N)\} & (II-12-a) \\ B(m, n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cdot \cos\{((2i+1)m - (2j+1)n)\pi / (2N)\} & (II-12-b) \end{cases}$$

Pour prouver que ces deux transformées peuvent être exprimées en termes de N TCDs 1-D, il faut montrer que le terme  $(2j+1)$  est soit un multiple de  $(2i+1)$  modulo  $(2N)$ , soit  $(2N)$  moins un multiple de  $(2i+1)$  modulo  $(2N)$  i.e.:

$$(2j+1) = \begin{cases} p \cdot (2i+1) \text{ mod}(2N) & (II-13-a) \\ \text{ou} & \\ 2N - p \cdot (2i+1) \text{ mod}(2N) & (II-13-b) \end{cases}$$

où  $p$  est un entier impair allant de 1 à  $N-1$ , car toute valeur de  $p$  en dehors de cet intervalle donne la même valeur de  $j$  que celle produite par l'une des valeurs de  $p$ , prise dans l'intervalle spécifié.

La relation (II-13) est équivalente à :

$$j = \begin{cases} p \cdot i + (p-1)/2 \text{ mod}(2N) & (II-14-a) \\ \text{ou} & p = 1, 3, 5, \dots, N-1. \\ N-1 - p \cdot i - (p-1)/2 \text{ mod}(2N) & (II-14-b) \end{cases}$$

Les  $N$  séquences de valeurs de  $j$ , obtenues par la relation (II-14) quand  $i$  varie de 0 à  $(N-1)$ , sont mutuellement différentes. Donc, l'ensemble des données entrées 2-D peut être groupé en  $N$  sous-ensembles différents de données dont les indices de chacun satisfont la relation (II-14).

Pour distinguer les séquences (II-14-a) et (II-14-b), nous adoptons la notation suivante :

$$\begin{cases} j(p, a) = p \cdot i + (p-1)/2 \text{ mod } N & (II-15-a) \\ \text{ou} & p = 1, 3, 5, \dots, N-1 \text{ et } i = 0, 1, 2, \dots, N-1. \\ j(p, b) = N-1 - p \cdot i - (p-1)/2 \text{ mod } N & (II-15-b) \end{cases}$$

Par suite, nous pouvons grouper la séquence d'entrées 2-D  $\{x(i, j) : i, j = 0, 1, \dots, N-1\}$  en  $N$  séquences 1-D  $\{x(i, j(p, a)) : i = 0, \dots, N-1\}$  et  $\{x(i, j(p, b)) : i = 0, \dots, N-1\}$  notées :

$$R_p^a = \{ x(i, j(p, a)) : i = 0, \dots, N-1 ; j(p, a) = p \cdot i + (p-1)/2 \text{ mod } N \} \quad (II-16-a)$$

$$R_p^b = \{ x(i, j(p, b)) : i = 0, \dots, N-1 ; j(p, b) = N-1 - p \cdot i - (p-1)/2 \text{ mod } N \} \quad (II-16-b)$$

Il reste à connaître le résultat exact de la division  $(pi+(p-1)/2) / N$  : en introduisant une nouvelle séquence entière  $q_{pi}$ , quotient de cette division, (II-16) est réécrite sans «mod» comme suit:

$$R_p^a = \{ x(i, j(p, a)) : i = 0, \dots, N-1 ; j(p, a) = p \cdot i + (p-1)/2 - N q_{pi} \} \quad (II-17-a)$$

$$R_p^b = \{ x(i, j(p, b)) : i = 0, \dots, N-1 ; j(p, b) = N-1 - p \cdot i - (p-1)/2 + N q_{pi} \} \quad (II-17-b)$$

En utilisant ces groupements de données,  $A(m, n)$  et  $B(m, n)$  peuvent s'écrire :

$$\begin{cases} A(m, n) = \sum_{i=0}^{N-1} \left\{ T_p^a(m, n) + T_p^b(m, n) \right\} & \text{(II-18-a)} \\ B(m, n) = \sum_{i=0}^{N-1} \left\{ S_p^a(m, n) + S_p^b(m, n) \right\} & \text{(II-18-b)} \end{cases}$$

où  $T_p^a(m, n)$ ,  $T_p^b(m, n)$ ,  $S_p^a(m, n)$  et  $S_p^b(m, n)$  sont définies par :

$$T_p^a(m, n) = \sum_{x(i, j) \in R_p^a} x(i, j) \cdot \cos\left\{((2i+1)m + (2j+1)n)\pi / (2N)\right\} \quad \text{(II-19-a)}$$

$$T_p^b(m, n) = \sum_{x(i, j) \in R_p^b} x(i, j) \cdot \cos\left\{((2i+1)m + (2j+1)n)\pi / (2N)\right\} \quad \text{(II-19-b)}$$

$$S_p^a(m, n) = \sum_{x(i, j) \in R_p^a} x(i, j) \cdot \cos\left\{((2i+1)m - (2j+1)n)\pi / (2N)\right\} \quad \text{(II-19-c)}$$

$$S_p^b(m, n) = \sum_{x(i, j) \in R_p^b} x(i, j) \cdot \cos\left\{((2i+1)m - (2j+1)n)\pi / (2N)\right\} \quad \text{(II-19-d)}$$

A présent, nous pouvons écrire :

$$y(m, n) = \sum_{\substack{p=1 \\ (p \text{ impair})}}^{N-1} (1/2) \left\{ T_p^a(m, n) + T_p^b(m, n) + S_p^a(m, n) - S_p^b(m, n) \right\} \quad \text{(II-20)}$$

Il reste à montrer que  $T_p^a(m, n)$ ,  $T_p^b(m, n)$ ,  $S_p^a(m, n)$  et  $S_p^b(m, n)$  peuvent s'exprimer en termes de TCDs 1-D. Substituons alors (II-17-a) dans (II-19-a) :

$$T_p^a(m, n) = \sum_{i=0}^{N-1} x(i, j_{(p, a)}) \cdot \cos\left\{((2i+1)m + (p(2i+1) - 2N \cdot q_{p,i})n)\pi / (2N)\right\} \quad \text{(II-21)}$$

qui peut être séparée selon que  $n$  est pair ou impair :

$$T_p^a(m, n) = \begin{cases} \sum_{i=0}^{N-1} x(i, j_{(p, a)}) \cdot \cos\left\{(2i+1)(m+n \cdot p)\pi / (2N)\right\} & \text{si } n \text{ pair} & \text{(II-22-a)} \\ \sum_{i=0}^{N-1} (-1)^{q_{p,i}} x(i, j_{(p, b)}) \cdot \cos\left\{(2i+1)(m+n \cdot p)\pi / (2N)\right\} & \text{si } n \text{ impair} & \text{(II-22-b)} \end{cases}$$

de la même façon, en substituant (II-17-b) dans (II-19-b), nous aurons :

$$T_p^b(m, n) = \sum_{i=0}^{N-1} x(i, j_{(p, b)}) \cdot \cos\left\{((2i+1)(m-np) + 2N(1+q_{p,i}))\pi / (2N)\right\} \quad \text{(II-23)}$$

séparée aussi selon les valeurs de  $n$ , telle que :

$$T_p^b(m, n) = \begin{cases} \sum_{i=0}^{N-1} x(i, j_{(p, a)}) \cdot \cos\left\{(2i+1)(m-n \cdot p)\pi / (2N)\right\} & \text{si } n \text{ pair} & \text{(II-24-a)} \\ -\sum_{i=0}^{N-1} (-1)^{q_{p,i}} x(i, j_{(p, b)}) \cdot \cos\left\{(2i+1)(m-n \cdot p)\pi / (2N)\right\} & \text{si } n \text{ impair} & \text{(II-24-b)} \end{cases}$$

La substitution de (II-17-a) et (II-27-b) respectivement dans (II-19-c) et (II-19-d), nous donne :

$$S_p^a(m,n) = \begin{cases} \sum_{i=0}^{N-1} x(i, j_{(p,a)}) \cdot \cos\{(2i+1)(m-n \cdot p)\pi / (2N)\} & \text{si } n \text{ pair} & \text{(II-25-a)} \\ \sum_{i=0}^{N-1} (-1)^q p^i x(i, j_{(p,b)}) \cdot \cos\{(2i+1)(m-n \cdot p)\pi / (2N)\} & \text{si } n \text{ impair} & \text{(II-25-b)} \end{cases}$$

$$S_p^b(m,n) = \begin{cases} \sum_{i=0}^{N-1} x(i, j_{(p,a)}) \cdot \cos\{(2i+1)(m+n \cdot p)\pi / (2N)\} & \text{si } n \text{ pair} & \text{(II-26-a)} \\ -\sum_{i=0}^{N-1} (-1)^q p^i x(i, j_{(p,b)}) \cdot \cos\{(2i+1)(m+n \cdot p)\pi / (2N)\} & \text{si } n \text{ impair} & \text{(II-26-b)} \end{cases}$$

En reportant ces relations dans (II-20),  $y(m, n)$  s'écrira :

$$y(m,n) = \begin{cases} (1/2) \sum_{p=1}^{N-1} \left[ \sum_{i=0}^{N-1} (x(i, j_{(p,a)}) + x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m+n \cdot p)\pi / (2N)\} + \sum_{i=0}^{N-1} (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m-n \cdot p)\pi / (2N)\} \right] & \text{si } n \text{ pair} & \text{(II-27-a)} \\ (1/2) \sum_{p=1}^{N-1} \left[ \sum_{i=0}^{N-1} (-1)^q p^i (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m+n \cdot p)\pi / (2N)\} + \sum_{i=0}^{N-1} (-1)^q p^i (x(i, j_{(p,a)}) + x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m-n \cdot p)\pi / (2N)\} \right] & \text{si } n \text{ impair} & \text{(II-27-b)} \end{cases}$$

Nous pouvons remarquer que les expressions :

$$\sum_{i=0}^{N-1} (x(i, j_{(p,a)}) + x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m+n \cdot p)\pi / (2N)\} \quad \text{(II-28-a)}$$

$$\sum_{i=0}^{N-1} (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1)(m-n \cdot p)\pi / (2N)\} \quad \text{(II-28-b)}$$

correspondent à des TCDs 1-D de la séquence  $\{x(i, j_{(p,a)}) + x(i, j_{(p,b)})\}$  dépendant de  $m$  et  $n$ .

Ceci en définissant :

$$f(p, l) = \sum_{i=0}^{N-1} (x(i, j_{(p,a)}) + x(i, j_{(p,b)})) \cdot \cos\{(2i+1) \cdot l \cdot \pi / (2N)\} \quad \text{(II-29)}$$

Les relations (II-28-a) et (II-28-b) correspondent à  $+f(p, l)$  ou  $-f(p, l)$  pour  $l=0, \dots, N-1$ .

Dans le cas de la relation (II-27-b), en définissant :

$$g(p, l) = \sum_{i=0}^{N-1} (-1)^q p^i (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1) \cdot l \cdot \pi / (2N)\} \quad \text{(II-30)}$$

nous pouvons voir que les relations suivantes :

$$\sum_{i=0}^{N-1} (-1)^q p^i (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1) \cdot (m+n \cdot p)\pi / (2N)\} \quad \text{(II-31-a)}$$

$$\sum_{i=0}^{N-1} (-1)^q p^i (x(i, j_{(p,a)}) - x(i, j_{(p,b)})) \cdot \cos\{(2i+1) \cdot (m-n \cdot p)\pi / (2N)\} \quad \text{(II-31-b)}$$

correspondent à  $\pm g(p, l)$  pour  $l=0, \dots, N-1$ .

Il en résulte que le calcul de la séquence TCD 2-D  $\{y(m, n) : m, n = 0, \dots, N-1\}$ , ne nécessite que des séquences  $\{f(p, l) : l = 0, \dots, N-1\}$  et  $\{g(p, l) : l = 0, \dots, N-1\}$  pour  $p = 1, 3, 5, \dots, N-1$ . Autrement dit, le calcul de la TCD  $N \times N$  requiert seulement le calcul de  $N$  TCDs 1-D. Les relations (II-29), (II-31) et (II-32) signifient que les  $y(m, n)$  sont exprimés en termes de sommations des  $f(p, l)$  et  $g(p, l)$  en sachant que:

$$\sum_{i=0}^{N-1} (x(i, j_{(N-p, a)}) + x(i, j_{(N-p, b)})) \cdot \cos\{(2i+1)(m-(N-p))\pi / (2N)\} = \pm f((N-p), l) \quad \text{(II-32-a)}$$

$$\sum_{i=0}^{N-1} (-1)^{q_{pi}} (x(i, j_{(N-p, a)}) - x(i, j_{(N-p, b)})) \cdot \cos\{(2i+1)(m-(N-p))\pi / (2N)\} = \pm g((N-p), (N-l)) \quad \text{(II-32-b)}$$

Ces deux relations nous permettent d'organiser les sommations en étages « papillons ».

La figure II.2 présente le diagramme-bloc de cet algorithme :

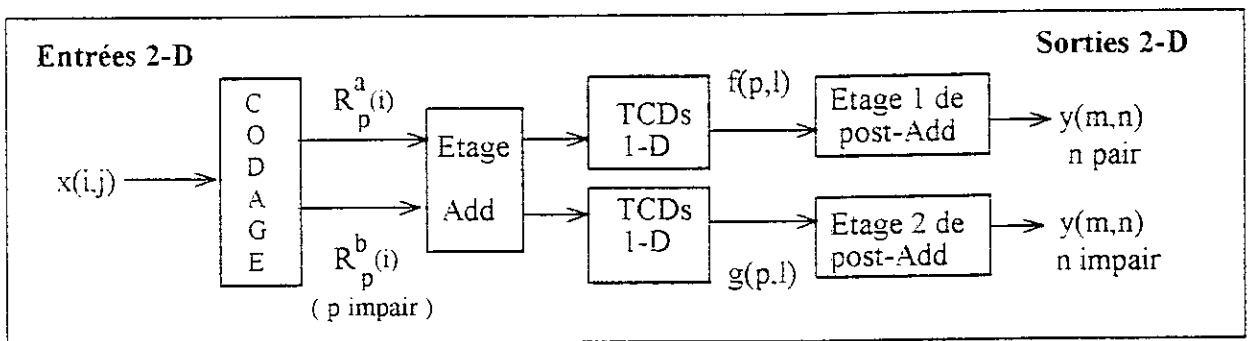


Figure II.2 - Diagramme-bloc de l'algorithme rapide TCD 2-D [34]

Cette structure exige  $N$  modules TCDs 1-D. Ce nombre peut être réduit de moitié si l'on utilise des modules de multiplexage et de démultiplexage comme illustré par la figure suivante:

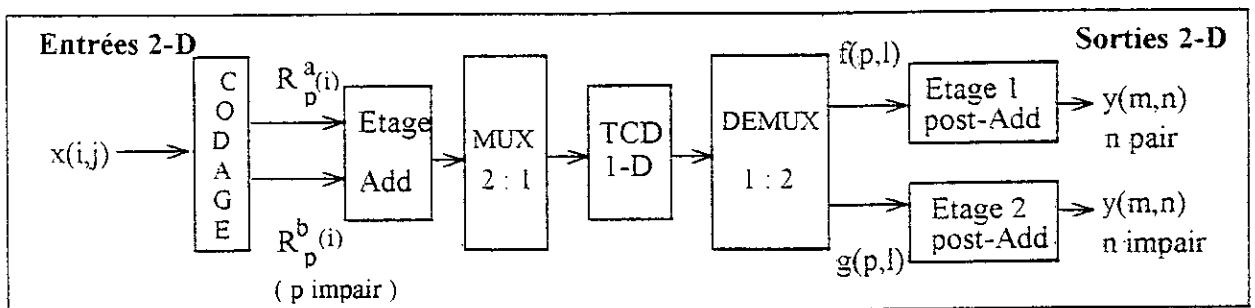


Figure II.3 - Diagramme-bloc de l'algorithme avec des étages MUX/DEMUX.

Une structure plus régulière des étages de post-additions, présentée dans [35] pour le cas de notre application  $N = 8$ , est montrée par les figures (II.4.a) et (II.4.b) :



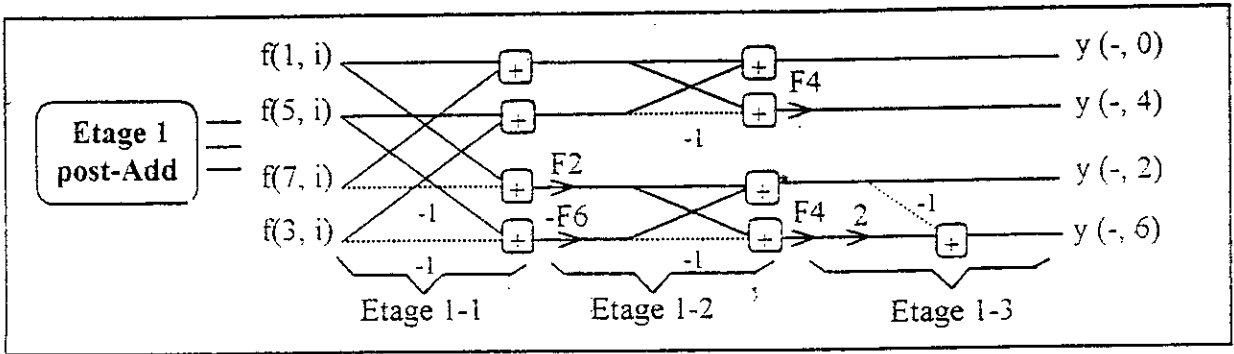


Figure II.4.a - Graphe de fluence des  $f(p,i)$  vers les  $y(m,n)$ ,  $n$  pair.

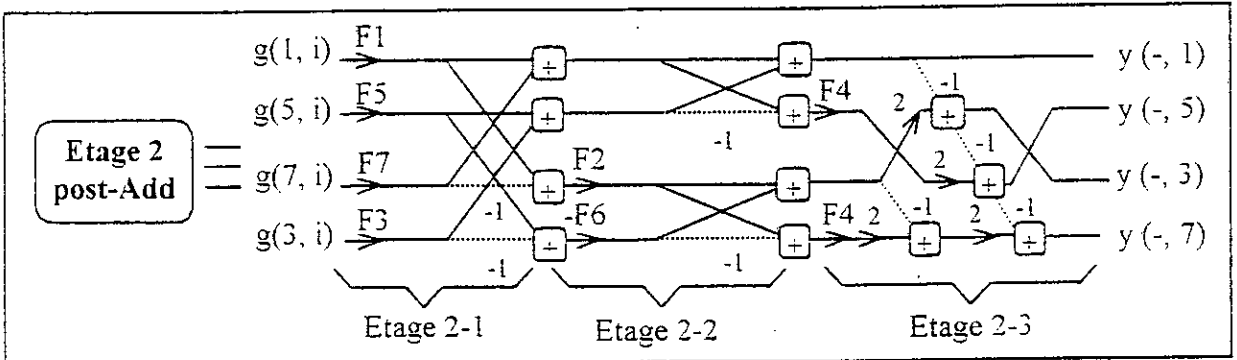


Figure II.4.b - Graphe de fluence des  $g(p,i)$  vers les  $y(m,n)$ ,  $n$  impair.

Où :

- 1 ligne  $\equiv$  8 lignes ( ligne brisée  $\equiv$  transfert de facteur  $(-1)$  );  $\longrightarrow$   $\equiv$  une multiplication;
- $y(-, n) = [ y(0, n), y(1, n), y(2, n), y(3, n), y(4, n), y(5, n), y(6, n), y(7, n) ]^T$ ;
- $f(p, i) = [ f(p, 0), f(p, 1), f(p, 2), f(p, 3), f(p, 4), f(p, 5), f(p, 6), f(p, 7) ]^T$ ,  $p = 1, 3, 5, 7$  ;
- $g(p, i) = [ g(p, 0), g(p, 1), g(p, 2), g(p, 3), g(p, 4), g(p, 5), g(p, 6), g(p, 7) ]^T$ ,  $p = 1, 3, 5, 7$  ;
- Les quantités  $F_i$ ,  $i = 0, \dots, N$  sont des matrices carrées  $N \times N$  ( $8 \times 8$  dans notre cas) dont les éléments sont donnés [35] par :

$$F_i(m, k) = \begin{cases} 1/2 & \text{si } \begin{cases} \cos(m+i)\pi / (2N) = \cos k\pi / 2N \\ \text{ou } \cos(m-i)\pi / (2N) = \cos k\pi / 2N \end{cases} \\ -1/2 & \text{si } \cos(m+i)\pi / (2N) = -\cos k\pi / 2N \\ 1 & \text{si } \cos(m+i)\pi / (2N) = \cos(m-i)\pi / 2N = \cos k\pi / 2N \\ 0 & \text{sinon} \end{cases} \quad \text{(II-33)}$$

Nous remarquons que chacun des étages 1 et 2 de post-additions est une succession de trois (03) sous-étages d'additions. Or, nous savons que le graphe de fluence de la TCD inverse est exactement le chemin inverse du graphe de fluence direct. Par conséquent, pour permettre une implémentation sur une même puce de ces deux transformées, il est souhaitable de « symétriser » cet algorithme c'est à dire essayer de parvenir à un diagramme-bloc avec autant d'étages d'additions en entrée qu'en sortie.

Considérons la représentation plus détaillée des étages de post-additions :

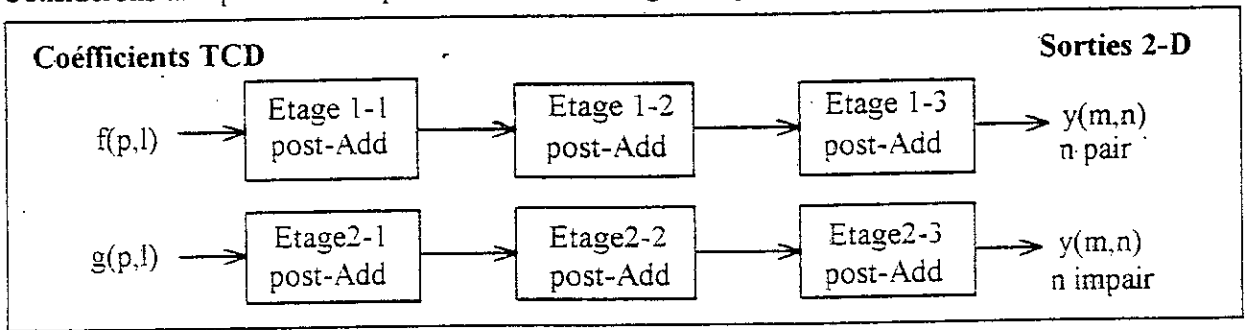


Figure II.5 - Les étages de post-additions.

Mathématiquement, la formulation de la « symétrisation » de l'algorithme s'écrit :

$$x(i, j) \xrightarrow{F} \begin{matrix} f(p, l) \\ g(p, l) \end{matrix} \xrightarrow{G} y(m, n)$$

avec :  $y(m, n) = (G \circ F)(x(i, j)) = L(x(i, j))$

où les fonctions F et G représentent respectivement le calcul TCDs 1-D et les post-additions.

La solution reviendrait à chercher d'autres fonctions F' et G' telles que :

$$x(i, j) \xrightarrow{F'} \begin{matrix} f(p, l) \\ g(p, l) \end{matrix} \xrightarrow{G'} y(m, n)$$

avec :  $y(m, n) = (G' \circ F')(x(i, j)) = L'(x(i, j))$

F' serait une fonction réalisant les post-additions et G' une fonction calculant les TCDs 1-D, c'est à dire équivalente à la fonction F. Le problème se résumerait alors à trouver F' telle que :

$$L'(x(i, j)) = L(x(i, j))$$

Nos différentes investigations n'ont permis d'aboutir, hélas, à aucune solution permettant cette équivalence. La meilleure solution serait de ramener les sous-étages 1-1 et 2-1 de post-additions à l'entrée de l'étage de calculs TCDs 1-D comme le montre la figure suivante:

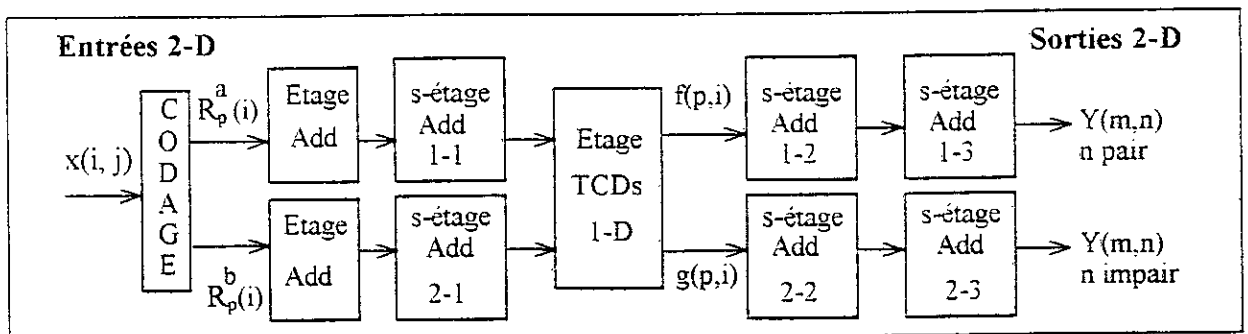


Figure II.6 - Symétrisation du diagramme-bloc.

Cette « symétrisation » serait possible pour l'étage supérieur des Y(m, n), n pair du fait que nous n'avons pas de multiplications des coefficients f(p, i) par des matrices Fi. Par contre, elle est impossible pour l'étage des coefficients Y(m, n), n impair à cause des matrices F1, F5, F7, F3 qui introduisent des non-linéarités dans le sous-étage de post-additions 2-1.

D'autres solutions utiliseraient seulement des signaux de contrôle pour l'acheminement des données sur le diagramme-bloc original comme illustré par la figure II.7, mais la communication interne de la puce serait plus complexe et difficile à contrôler.

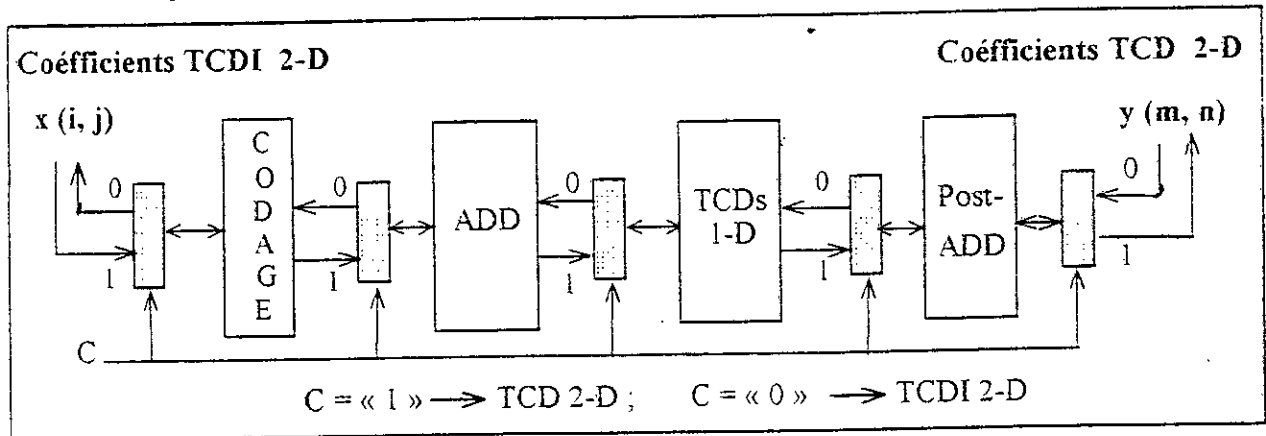


Figure II.7 - Utilisation de signaux de contrôle pour le calcul direct et inverse.

### 2.3.2.2 - DISCUSSION

L'algorithme présenté dans cette section, offre la meilleure structure modulaire pour le calcul de la TCD 2-D avec seulement  $N/2$  modules TCDs 1-D, mais ne se prête pas à une implantation sur un ASIC unifié TCD 2-D/TCDI 2-D. Autrement dit, il apparaît impossible de le « symétriser »; ceci nous ramène à considérer les algorithmes basés sur l'approche conventionnelle de ligne / colonne, dont les performances dépendent de la réalisation des calculs TCDs 1-D.

## 2.4 - ALGORITHMES TCD 1-D

Il existe plusieurs algorithmes de calcul de la TCD 1-D utilisant diverses techniques de dérivation. Ils reprennent tous l'idée « Divide and Conquer » et sont basés sur des décompositions matricielles. En effet, soit la formulation matricielle d'une transformée orthogonale discrète unidimensionnelle :

$$Y_N = (1/N)A_N \cdot X_N ; N = 2^m, m \text{ entier} \tag{II-34}$$

où :  $Y_N = [Y(0), Y(1), \dots, Y(N-1)]^T$ ;  
 $X_N = [X(0), X(1), \dots, X(N-1)]^T$ ;  
 $A_N = [A(i, j)] ; i, j = 0, \dots, (N-1)$ .

Les éléments  $A(i,j)$  sont les fonctions orthogonales de base de la transformée.

Dans [36], on montre que tout algorithme de calcul rapide d'une transformée orthogonale est basé sur la décomposition de la matrice  $A_N$  en un produit de  $\log_r(N)$  matrices creuses, composée chacune de  $(r \times N)$  éléments,  $r$  étant le radical de la décomposition. Le nombre d'éléments à manipuler est ainsi réduit de  $(N \times N)$  à  $(r.N.\log_r(N))$  et l'on peut exprimer toute transformée orthogonale à travers des transformées de moindres dimensions.

La factorisation de la matrice  $A_N$  est basée sur deux outils mathématiques : les matrices de permutation et les produits de Kronecker. Ceci conduit à une représentation récursive du processus de calcul rapide :

$$Y = (1/N)(S(m) \cdot M(m) \cdots (S(i) \cdot M(i) \cdots (S(1) \cdot M(1) \cdot C \cdot X) \cdots) \cdots) \quad (\text{II-35-a})$$

$$\text{ou : } Y = (1/N)(C \cdot S(1) \cdot M(1) \cdots (S(m-i) \cdot M(m-i) \cdots (S(m) \cdot M(m) \cdot X) \cdots) \cdots) \quad (\text{II-35-b})$$

où  $M$ ,  $S$  et  $C$  sont respectivement des matrices de multiplication, de sommation et de permutation.

Le radical de décomposition  $r$  est le nombre de coefficients mis en combinaison après la décomposition (c'est-à-dire la taille des sous-matrices). Pour le cas de la TCD, le nombre d'itérations donné par la relation :  $i = (N/r)$ , est consigné dans le tableau suivant :

Radical (r)	Nombre d'itérations ( i )	Nombre d'éléments
2	4	2
4	2	4
8	1	8

Tableau II.1 - Nombre d'itérations en fonction du radical  $r$ .

Nous présenterons ici les algorithmes les plus attrayants en efficacité de calcul, en régularité et en modularité de leurs structures. L'étude sera illustrée par des graphes de fluence de façon à pouvoir suivre, avec facilité, le processus général de calcul de la TCD, et à élaborer des critères de comparaison. Notre objectif reste bien sûr, de dégager l'algorithme le plus adapté à une architecture unifiée.

#### 2.4.1 - DECOMPOSITION EN RADICAL -2

Nous présentons les algorithmes les plus utilisés dans cette catégorie avec leurs graphes de fluences, structurés en modules de transformation  $r = 2$  pour voir s'ils offrent une symétrie à même de permettre une implémentation unifiée pour le calcul direct et inverse de la TCD.

##### 2.4.1.1 - ALGORITHME RECURSIF DE HOU [37]

Cet algorithme sert à générer des TCDs d'ordres élevés à partir de TCD d'ordres plus faibles. Cette génération récursive est très intéressante dans le cas d'ordres très élevés.

L'expression matricielle de la TCD 1-D donnée par l'équation (II-1), s'écrit comme suit :

$$Y = (2/N) \cdot T(N) \cdot X \quad (\text{II-36})$$

où  $X$  et  $Y$  sont les séquences d'entrées et de coefficients TCD, arrangées respectivement dans un ordre naturel,  $T(N)$  est la matrice TCD d'ordre  $N$  et  $(2/N)$  un facteur de normalisation.

Pour mettre en évidence certaines propriétés de la matrice  $T(N)$ , nous donnons quelques exemples en convenant de négliger le facteur de normalisation et le rétablir en fin de calcul. Le second ordre ( $N=2$ ) de la TCD est donné comme suit:

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \alpha & -\alpha \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}; \quad \alpha = 1/\sqrt{2} \quad (\text{II-37})$$

Le quatrième ordre ( $N=4$ ) de la TCD est donné comme suit:

$$\begin{bmatrix} Y_0 \\ Y_2 \\ Y_1 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \alpha & -\alpha & \alpha & -\alpha \\ \beta & -\delta & -\beta & \delta \\ \delta & \beta & -\delta & -\beta \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_2 \\ x_1 \\ x_3 \end{bmatrix}; \quad \begin{aligned} \text{où : } & \alpha = 1/\sqrt{2}, \\ & \beta = \text{Cos}(\pi/8) \\ & \text{et } \delta = \text{Sin}(\pi/8). \end{aligned} \quad (\text{II-38})$$

La TCD d'ordre  $N=8$  est donnée comme suit:

$$\begin{bmatrix} Y_0 \\ Y_4 \\ Y_2 \\ Y_6 \\ Y_1 \\ Y_5 \\ Y_3 \\ Y_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha & \alpha & -\alpha \\ \beta & -\delta & -\beta & \delta & \beta & -\delta & -\beta & \delta \\ \delta & \beta & -\delta & -\beta & \delta & \beta & -\delta & -\beta \\ \lambda & \mu & -\nu & -\gamma & -\lambda & -\mu & \nu & \gamma \\ \mu & \nu & -\gamma & \lambda & -\mu & -\nu & \gamma & -\lambda \\ \gamma & -\lambda & \mu & \nu & -\gamma & \lambda & -\mu & -\nu \\ \nu & \gamma & \lambda & \mu & -\nu & -\gamma & -\lambda & -\mu \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \\ x_7 \\ x_5 \\ x_3 \\ x_1 \end{bmatrix} \quad \begin{aligned} \text{où : } & \alpha = 1/\sqrt{2}, \\ & \beta = \text{Cos}(\pi/8), \\ & \delta = \text{Sin}(\pi/8), \\ & \lambda = \text{cos}(\pi/16), \\ & \gamma = \text{cos}(3\pi/16), \\ & \mu = \text{sin}(3\pi/16), \\ & \text{et } \nu = \text{sin}(\pi/16). \end{aligned} \quad (\text{II-39})$$

Ces exemples permettent d'observer la propriété récursive de la génération des matrices TCD donnée comme suit:

$$\hat{T}(N) = \begin{bmatrix} \hat{T}(N/2) & \hat{T}(N/2) \\ \hat{D}(N/2) & -\hat{D}(N/2) \end{bmatrix} \quad (\text{II-40})$$

où  $\hat{T}(N)$  est la matrice TCD ( $T(N)$ ) avec permutation de ses rangées et colonnes dans un ordre spécifique et  $\hat{D}(N/2)$  une matrice que nous expliciterons plus loin. Ce développement est basé sur les propriétés intrinsèques suivantes de la TCD :

• **Propriété 1**

La TCD peut être convertie en TFD avec une phase variable :

$$Y_m = \sum_{i=0}^{N-1} \tilde{X}_i \cdot \text{Cos}(\theta_m + 2\pi m i / N) \quad \text{avec : } \begin{cases} \theta_m = \pi m / 2N \\ \tilde{X}_i = X_{2i} \text{ et } \tilde{X}_{N-i-1} = X_{2i-1} \\ i = 0, \dots, (N/2)-1. \end{cases} \quad (\text{II-41})$$

La séquence de données  $X_i$  a été réordonnée en accord avec la relation suivante :  $\tilde{X} = P \cdot X$  où  $P$  est une matrice de permutation donnée par :

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 & 0 & 0 \\ \dots & \dots & \dots & 0 & 1 & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \tag{II-42}$$

• **Propriété 2**

La matrice TCD peut être partitionnée en quatre cadrans tels que :

- \* pour les lignes d'ordres pairs, le cadran gauche et le cadran droit sont les mêmes;
- \* pour les lignes d'ordres impairs, le cadran gauche et le cadran droit sont de signes opposés.

Les noyaux cosinus sont donnés par :  $\tilde{t}_{i,m} = \text{Cos}(\theta_m + 2\pi m i / N)$  où :  $i, m = 0, \dots, (N-1)$ .

En divisant le rang des indices  $i$  en deux intervalles :  $i'$  :  $(0, N/2)$  et  $i'' = i' + N/2$  :  $(N/2, N-1)$ , nous aurons :

$$\begin{cases} \tilde{t}_{m i''} = \tilde{t}_{m i'} & \text{pour } m \text{ pair : } (0, N/2) \\ \tilde{t}_{m i''} = -\tilde{t}_{m i'} & \text{pour } m \text{ impair : } (1, N-1) \end{cases} \tag{II-43}$$

La matrice TCD prend la forme suivante :  $\tilde{T}(N) = \begin{bmatrix} \tilde{E}(N/2) & \tilde{E}(N/2) \\ \tilde{D}(N/2) & -\tilde{D}(N/2) \end{bmatrix}$  (II-44)

où  $\tilde{T}$  et  $\tilde{E}$  sont des matrices carrées  $(N/2 \times N/2)$  qui seront définies ultérieurement.

La séquence de sortie donnant les coefficients TCD devient :

$$Y = \{Y_0, Y_2, Y_4, \dots, Y_{N-2}, Y_1, Y_3, Y_5, \dots, Y_{N-1}\} \tag{II-45}$$

• **Propriété 3**

La moitié supérieure de la matrice TCD d'ordre  $N$  correspond à 2 matrices TCD d'ordre  $N/2$ .

• **Dérivation de l'algorithme**

Maintenant, nous allons établir la relation entre  $\hat{T}(N)$  et  $\hat{D}(N/2)$  pour dériver l'algorithme récursif rapide de la TCD. Commençons par réécrire l'expression des noyaux tq :

$$\tilde{t}_{i,m} = \text{Cos}(m\phi_i) \quad \text{avec: } \phi_i = (4i + 1)\frac{\pi}{2N}; \quad i = 0, \dots, N-1 \tag{II-46}$$

En se référant à cette équation pour  $M=N/2$  et en utilisant la propriété 2, nous aurons :

$$\tilde{D}(M) = \begin{bmatrix} \cos\phi_0 & \cos\phi_1 & \dots & \cos\phi_{M-1} \\ \cos 3\phi_0 & \cos 3\phi_1 & \dots & \cos 3\phi_{M-1} \\ \vdots & \vdots & \dots & \vdots \\ \cos(N-1)\phi_0 & \cos(N-1)\phi_1 & \dots & \cos(N-1)\phi_{M-1} \end{bmatrix} \tag{II-47}$$

et :

$$\tilde{E}(M) = \begin{bmatrix} 1 & & & & 1 \\ \cos 2\phi_0 & & & & \cos 2\phi_{M-1} \\ & & & & \\ & & & & \\ & & & & \\ \cos (N-2)\phi_0 & \cos (N-2)\phi_1 & \dots & \dots & \cos (N-2)\phi_{M-1} \end{bmatrix} \quad (II-48)$$

En utilisant répétitivement l'identité suivante :

$$\cos[(2m+1)\phi_i] = 1/(2 \cos \phi_i) \cdot [\cos(2m\phi_i) + \cos(2m+2)\phi_i] \quad (II-49)$$

nous obtenons :  $\tilde{D}(M) = L(M) \cdot \tilde{E}(M) \cdot \text{diag}[1/(2 \cdot \cos \phi_i)]$ ,  $i = 0, 1, 2, \dots, M-1$  (II-50)

$L(M)$  est une matrice triangulaire supérieure donnée par :

$$L(M) = \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & 0 & \dots & 1 & 1 \\ 0 & \dots & \dots & 0 & \dots & 0 & 1 \end{bmatrix} \quad (II-51)$$

la séquence de coefficients TCD devient maintenant:  $\begin{bmatrix} Y_e \\ Y_o \end{bmatrix} = \begin{bmatrix} \tilde{E}(M) & \tilde{E}(M) \\ \tilde{D}(M) & -\tilde{D}(M) \end{bmatrix} \cdot \begin{bmatrix} \tilde{x}_p \\ \tilde{x}_r \end{bmatrix}$  (II-52)

où  $Y_e$  et  $Y_o$  sont respectivement les moitiés paire et impaire de la séquence des coefficients TCD, chacune arrangée en une séquence naturelle, et  $\tilde{x}_p$  et  $\tilde{x}_r$  sont les moitiés précédentes et récentes de la séquence de données entrées pour laquelle  $\tilde{x}$  est arrangé dans un ordre naturel :

$$Y = [Y_e, Y_o]^T = [Y(0), Y(2), \dots, Y(N-2), Y(1), Y(3), \dots, Y(N-1)]^T \quad (II-53)$$

$$\tilde{x} = [\tilde{x}_p, \tilde{x}_r]^T = [x(0), x(2), \dots, x(N-2), x(N-1), x(N-3), \dots, x(1)]^T \quad (II-54)$$

Pour formuler l'algorithme récursif pour la TCD, les indices des séquences  $Y_e$  et  $Y_o$  sont arrangés en bits inversés tq :  $\hat{Y}_e = R \cdot Y_e$  et  $\hat{Y}_o = R \cdot Y_o$ , où  $\hat{Y}_e$  et  $\hat{Y}_o$  sont les moitiés paire et impaire de la séquence TCD, arrangées en bits inversés;  $R$  est la matrice de permutation.

En accomplissant l'inversion des bits et en utilisant la propriété 3 et l'identité trigonométrique précédente, nous obtenons :

$$\begin{bmatrix} \hat{Y}_e \\ \hat{Y}_o \end{bmatrix} = \begin{bmatrix} \hat{T}(M) & \hat{T}(M) \\ \hat{D}(M) & -\hat{D}(M) \end{bmatrix} \cdot \begin{bmatrix} \tilde{x}_p \\ \tilde{x}_r \end{bmatrix} \quad (II-55)$$

avec :  $\hat{T}(M) = R \cdot \tilde{E}(M)$  (II-56)

$$\hat{D}(M) = R \cdot L(M) \cdot \tilde{E}(M) \cdot \text{diag}[1/(2 \cdot \cos \phi_i)]$$
,  $i = 0, \dots, M-1$  (II-57)

Enfin, l'algorithme récursif pour la TCD en décimation fréquentielle, est donné comme suit :

$$\begin{bmatrix} \hat{Y}_e \\ \hat{Y}_o \end{bmatrix} = \frac{2}{N} \cdot \begin{bmatrix} \hat{T}(N/2) & \hat{T}(N/2) \\ K \cdot \hat{T}(N/2) \cdot Q & -K \cdot \hat{T}(N/2) \cdot Q \end{bmatrix} \cdot \begin{bmatrix} \tilde{x}_p \\ \tilde{x}_r \end{bmatrix} \quad (II-58)$$

avec :  $K = R \cdot L(N/2) \cdot R^T$  (II-59)

$Q = \text{diag}[1 / (2 \cdot \text{Cos} \phi_i)], i = 0, \dots, (N/2) - 1$  (II-60)

Comme la matrice R est symétrique, K devient :  $K = R \cdot L(N/2) \cdot R$  (II-61)

L'algorithme en décimation temporelle s'obtient en intervertissant l'indexage entre l'entrée et la sortie et en prenant la transposée de  $\hat{T}$  :

$$\begin{bmatrix} \tilde{Y}_p \\ \tilde{Y}_r \end{bmatrix} = \frac{2}{N} \cdot \begin{bmatrix} \hat{T}'(N/2) & Q \cdot \hat{T}'(N/2) \cdot K' \\ \hat{T}'(N/2) & -Q \cdot \hat{T}'(N/2) \cdot K' \end{bmatrix} \cdot \begin{bmatrix} \hat{x}_s \\ \hat{x}_o \end{bmatrix}$$
 (II-62)

avec :  $K' = R \cdot U(N/2) \cdot R$  et :  $\hat{T}' = \hat{T}^T$  et  $U = L' = L^T$  (II-63)

L'implémentation de cet algorithme en décimation fréquentielle est donnée par ce schéma-bloc :

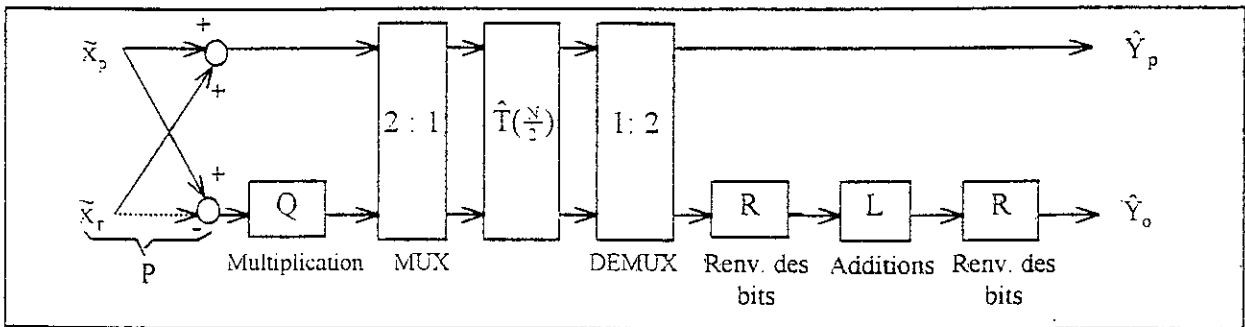


Figure II.8 - Schéma-bloc d'implémentation de la TCD en décimation fréquentielle.

Ce schéma-bloc nous permet de dresser le graphe de fluence pour l'ordre  $N = 8$  :

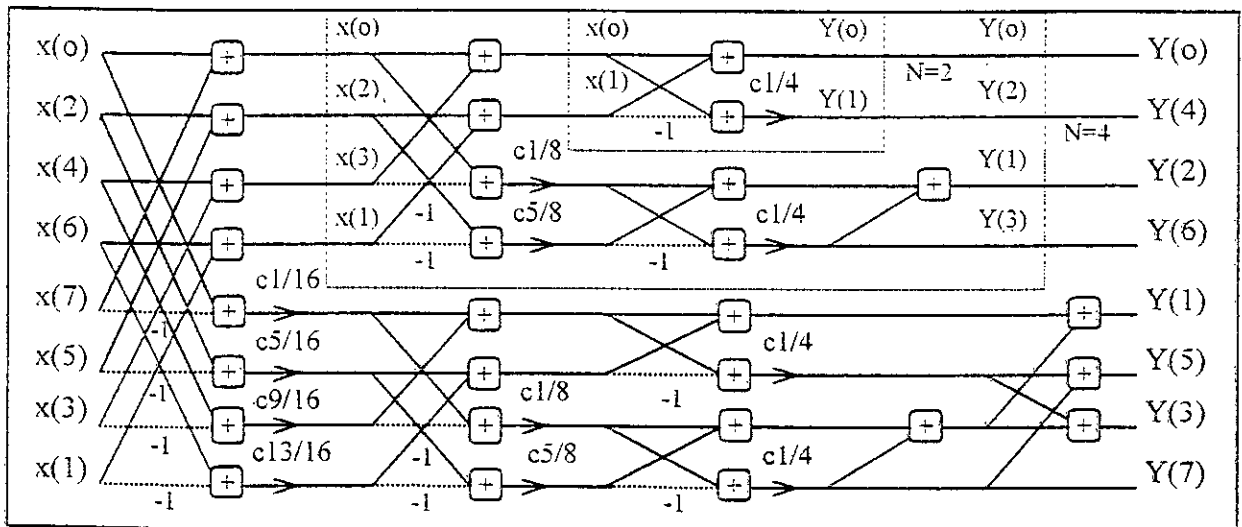


Figure II.9 - Graphe de fluence de la TCD (N=8) par l'algorithme de HOU avec:  $c \ a/b = 1/(2 \cos \pi \ a/b)$

Ce graphe de fluence est ordonné ci-après en modules matriciels de transformations d'ordre 2 :



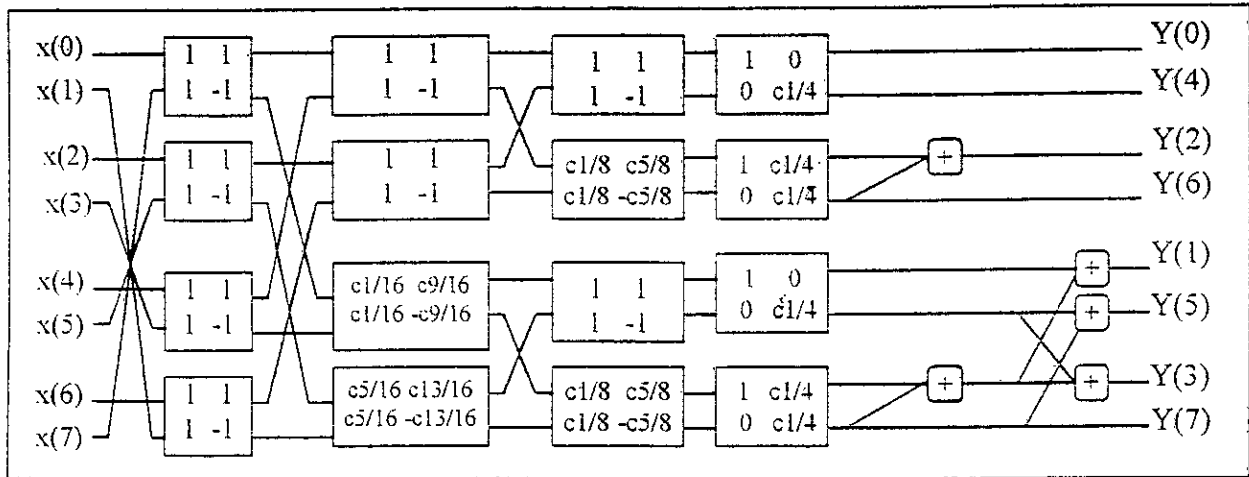


Figure II.10 - Graphe de fluence de la TCD (N=8) par l’algorithme de HOU, organisé en modules matriciels de transformations d’ordre 2.

**2.4.1.2 - ALGORITHME DE CHEN [38]**

Cet algorithme se base sur la décomposition de la matrice TCD, T(N), en l’écrivant sous la forme suivante:

$$[T(N)] = [P(N)] \cdot \begin{bmatrix} T(N/2) & 0 \\ 0 & R(N/2) \end{bmatrix} \cdot [B(N)] \tag{II-64}$$

où :  $[R(N/2)] = \left[ u(m) \cdot \cos(2i + 1)(2m + 1) \frac{\pi}{2N} \right]$ ;  $i, m = 0, 1, \dots, N-1$ . (II-65)

$[P(N)]$ : matrice NxN de permutation pour passer de l’ordre de bits inversés à l’ordre naturel;

$$[B(N)] = \begin{bmatrix} I(N/2) & \bar{I}(N/2) \\ \bar{I}(N/2) & -I(N/2) \end{bmatrix} \text{ où } \begin{cases} [I(N/2)] \text{ est la matrice identité d'ordre } N/2 \\ [\bar{I}(N/2)] \text{ est sa matrice diagonale opposée} \end{cases}$$

la matrice opposée de  $[B(N)]$  est :  $[\bar{B}(N)] = \begin{bmatrix} -I(N/2) & \bar{I}(N/2) \\ \bar{I}(N/2) & I(N/2) \end{bmatrix}$  (II-66)

La matrice TCD est générée récursivement à partir de l’ordre 2 :  $[T(2)] = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$  (II-67)

et de la décomposition de la matrice  $[R(N/2)]$ :

$$[R(N/2)] = [M_1] \cdot [M_2] \cdot [M_3] \times \dots \times [M_{(2\log_2 N - 3)}] \tag{II-68}$$

Soient les matrices suivantes :

$$[S_k^m] = (\text{Sin}(m\pi / k)) \cdot [I_{N/2k}], \quad [\bar{S}_k^m] = (\text{Sin}(m\pi / k)) \cdot [\bar{I}_{N/2k}] \tag{II-69-a}$$

$$[C_k^m] = (\text{Cos}(m\pi / k)) \cdot [I_{N/2k}], \quad [\bar{C}_k^m] = (\text{Cos}(m\pi / k)) \cdot [\bar{I}_{N/2k}] \tag{II-69-b}$$

Les  $M_{(2\log_2(N-3))}$  matrices de la décomposition de  $[R(N/2)]$  sont de quatre types :

- Type 1 :  $[M_1]$ , la première matrice formée de sous-matrices  $[S_{2N}^{a_j}]$ ,  $[\bar{S}_{2N}^{a_j}]$ ,  $[C_{2N}^{a_j}]$ , et  $[\bar{C}_{2N}^{a_j}]$  où les valeurs de  $a_j$  sont la représentation en bits inversés de  $N/2 + j - 1$  pour  $j=1, 2, \dots, N/2$ ;
- Type 2 :  $[M_{(2\log_2(N+3))}]$ , la dernière matrice donnée dans ce qui suit;
- Type 3 :  $[M_i]$ , les matrices numérotées impaires :  $[M_3], [M_5], \dots$ , formées de deux diagonales:
  - La diagonale principale: formée du haut au milieu par la répétition des séquences de matrices  $[I_{N/2k}]$ ,  $[C_k^{m_j}]$ ,  $[S_k^{m_j}]$  et  $[I_{N/2k}]$  où  $k=N/2^{(q-1)/2}$  pour  $j=1, 2, \dots, k/8$ ; et du milieu à la partie inférieure par la séquence de matrices  $[I_{N/2k}]$ ,  $[C_k^{m_j}]$ ,  $[S_k^{m_j}]$  et  $[I_{N/2k}]$  pour  $j=(k/8)+1, \dots, k/4$ .
  - La diagonale opposée: formée similairement par la répétition des séquences de matrices  $[0_{N/2k}]$ ,  $[\bar{S}_k^{m_j}]$ ,  $[\bar{C}_k^{m_j}]$  et  $[0_{N/2k}]$  du haut au milieu, et par  $[0_{N/2k}]$ ,  $[-\bar{S}_k^{m_j}]$ ,  $[\bar{C}_k^{m_j}]$  et  $[0_{N/2k}]$  du milieu à la partie inférieure ( $[0_i]$  est la matrice nulle  $1 \times 1$ );
- Type 4 :  $[M_p]$ , les matrices numérotées paires :  $[M_2], [M_4], \dots$ , matrices diagonales formées par  $[B]$  et son opposée  $[\bar{B}]$ .

Cette décomposition est faite alors comme suit :

$$[M_1] = \begin{bmatrix} S_{2N}^{a_1} & & & \bar{C}_{2N}^{a_1} & \bar{C}_{2N}^{a_1} \\ & S_{2N}^{a_2} & & \bar{C}_{2N}^{a_2} & \\ & & S_{2N}^{a_{N/4}} & \bar{C}_{2N}^{a_{N/4}} & \\ & & \bar{S}_{2N}^{a_{N/4}} & C_{2N}^{a_{N/4}} & \\ -\bar{S}_{2N}^{a_{N/2}} & -\bar{S}_{2N}^{a_{N/2}-1} & & C_{2N}^{a_{N/2}-1} & C_{2N}^{a_{N/2}} \end{bmatrix}, [M_2] = \text{diag}([B_2], [\bar{B}_2], \dots, [B_2], [\bar{B}_2]) \quad (\text{II-70})$$

$$[M_3] = \begin{bmatrix} 1 & & & & & & & 0 \\ & -C_{N/2}^{b_1} & & & & & & \bar{S}_{N/2}^{b_1} \\ & & -S_{N/2}^{b_1} & & & & & -\bar{C}_{N/2}^{b_1} \\ & & & 1 & & 0 & & \\ & & & & \ddots & & & \\ & & & & & 1 & & \\ & & & & & & 0 & 1 \\ & & & & & & -\bar{S}_{N/2}^{b_{N/8}} & C_{N/2}^{b_{N/8}} \\ & & & & & & & S_{N/2}^{b_{N/8}} \\ 0 & \bar{C}_{N/2}^{b_{N/8}} & & & & & & \\ & & & & & & & 1 \end{bmatrix}, [M_4] = \text{diag}([B_4], \dots, [B_4], [\bar{B}_4]) \quad (\text{II-71})$$



Cette factorisation conduit au graphe de fluence suivant :

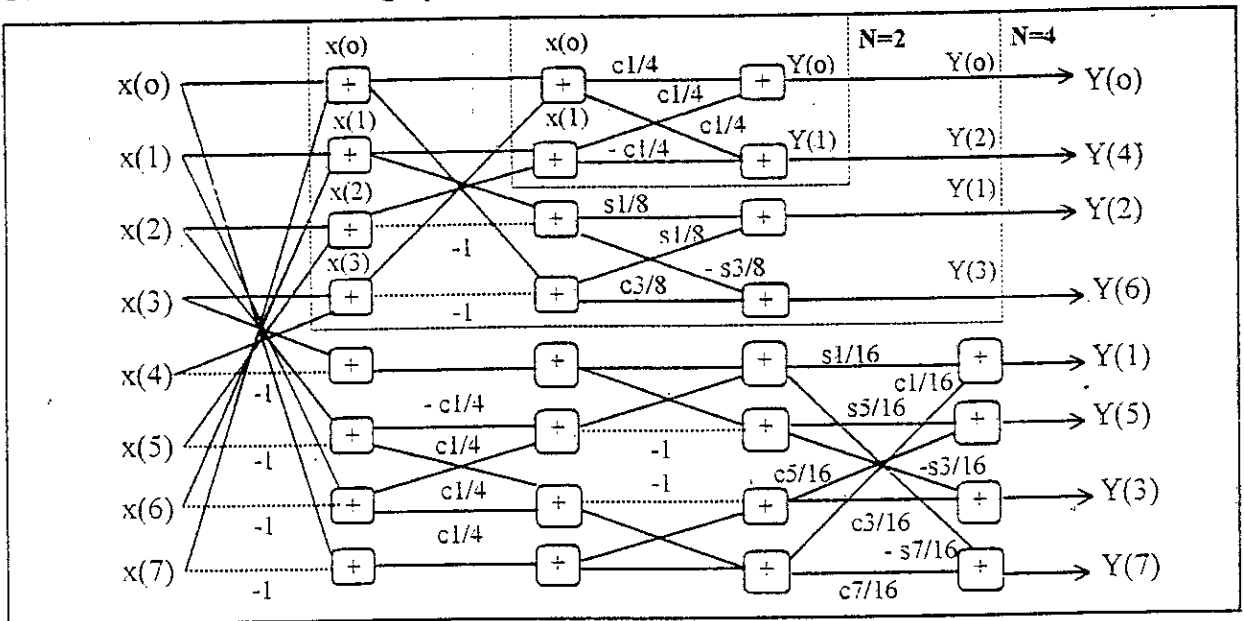


Figure II.11 - Graphe de fluence de la TCD (N=8) par l'algorithme de CHEN  
( où :  $c\ a/b = \cos(\pi a/b)$  et  $s\ a/b = \sin(\pi a/b)$  )

La figure suivante donne ce graphe de fluence organisé en modules de transformations d'ordre 2

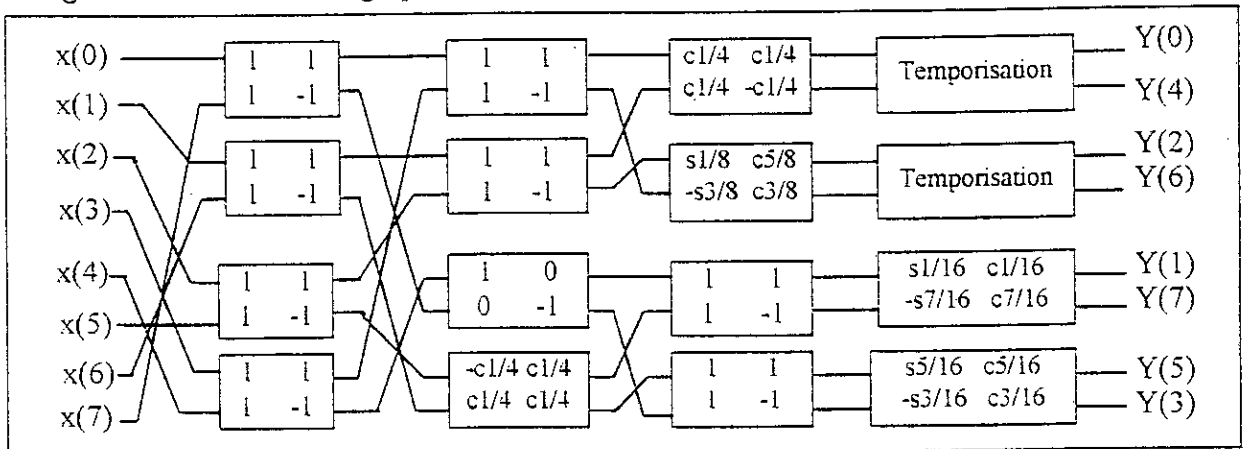


Figure II.12 - Graphe de fluence ordonné de la TCD (N=8) par l'algorithme de CHEN  
( où :  $c\ a/b = \cos(\pi a/b)$  et  $s\ a/b = \sin(\pi a/b)$  )

**2.4.1.3 - ALGORITHME PAR TFD [26],[39]-[41]**

La transformée de Fourier est un outil très puissant en traitement numérique du signal pour ses diverses performances. Dans notre cas, la TFD est utilisée pour accélérer l'évaluation de la TCD. Le calcul de la TCD N-points est simple à partir d'une TFD 2N-points mais nous verrons aussi qu'il est possible de ramener la taille de cette dernière à N-points.

• **Calcul de la TCD à partir d'une TFD 2N-points**

Nous pouvons réécrire l'expression de la TCD comme suit :

$$Y(m) = (2/N)u(m) \sum_{k=0}^{N-1} X(k) \cdot \cos(2k+1)m\pi/2N \tag{II-79}$$

$$= (2/N)u(m) \cdot \text{Re} \left[ \text{Exp}(-j\pi m/2N) \cdot \sum_{k=0}^{N-1} X(k) \cdot W_{2N}^{-mk} \right]$$

où  $\text{Re}$  signifie « partie réelle de » et  $W_{2N}^{-mk} = \text{Exp}(-j2\pi mk/(2N))$  et  $j^2 = -1$ . (II-80)

La quantité  $(\sum_{k=0}^{2N-1} X(k) \cdot W_{2N}^{-mk})$  est une TFD 2N-points. Ceci nous permet donc de calculer la TCD N-points à partir d'une TFD 2N-points avec des entrées  $X(k) = 0$  pour  $k=N, N+1, \dots, 2N$  en suivant le diagramme-bloc suivant :

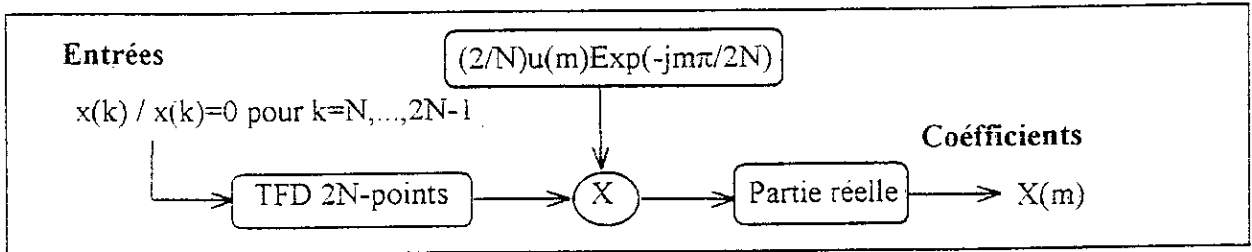


Figure II.13 - Processus de calcul de la TCD par la TFD.

• Calcul de la TCD à partir d'une TFD N-points

Soit la quantité suivante :  $H(m) = \sum_{k=0}^{N-1} X(k) \cdot \cos(2k+1)m\pi/2N$  (II-81)

La TCD est donnée alors par :  $Y(m) = (2/N) \cdot u(m) \cdot H(m)$  (II-82)

La parité de N nous permet d'écrire:

$$H(m) = \sum_{k=0}^{(N/2)-1} X(2k) \cdot \cos\{(4k+1)m\pi/2N\} + \sum_{k=0}^{(N/2)-1} X(2k+1) \cdot \cos\{(4k+3)m\pi/2N\}$$
 (II-83)

posons alors :  $\begin{cases} y(k) = x(2k) \text{ et } y(k') = x(2k+1), k' = N-1-k \\ k = 0, \dots, (N/2)-1 \text{ et donc : } k' = (N/2), \dots, N-1 \end{cases}$  (II-84)

d'où :  $H(m) = \sum_{k=0}^{(N/2)-1} y(k) \cdot \cos\{(4k+1)m\pi/2N\} + \sum_{k'=N/2}^{N-1} y(k') \cdot \cos\{(4(N-1-k')+3)m\pi/2N\}$  (II-85)

Considérons le second terme de H(m) :

$$\sum_{k'=N/2}^{N-1} y(k') \cdot \cos\{(4(N-1-k')+3)m\pi/2N\} = \sum_{k'=N/2}^{N-1} y(k') \cdot \cos\{2m\pi - (4k'+1)m\pi/2N\}$$
 (II-86)

Or la fonction « Cos(x) » est paire, périodique de période  $(2m\pi)$ , d'où :

$$H(m) = \sum_{k=0}^{(N/2)-1} y(k) \cdot \cos\{(4k+1)m\pi/2N\} + \sum_{k'=N/2}^{N-1} y(k') \cdot \cos\{(4k'+1)m\pi/2N\}$$
 (II-87)

$$= \sum_{k=0}^{N-1} y(k) \cdot \cos\{(4k+1)m\pi/2N\}$$

Soit :  $R(m) = \text{Exp}(-jm\pi/2N) \cdot \sum_{k=0}^{N-1} y(k) \cdot \text{Exp}(2mk\pi/2N)$  (II-88)

Nous pouvons vérifier que :

$$R(m) = \sum_{k=0}^{N-1} y(k) \cdot \text{Exp}\{-j(4k+1)m\pi / 2N\} \quad (\text{II-89-a})$$

$$H(m) = \text{Re} [R(m)] \quad (\text{II-89-b})$$

Il en résulte que les coefficients TCD sont donnés par :

$$\text{TCD} : Y(m) = (2 / N) \cdot u(m) \cdot \text{Re} [R(m)] \quad (\text{II-90})$$

Le calcul de la TCD revient donc, à calculer R(m) puis à prendre sa partie réelle.

L'expression R(m) dans (II-89-a) est un produit de deux membres imaginaires Z<sub>1</sub> et Z<sub>2</sub> tq :

$$Z_1 = \text{Exp}\{-j m \pi / 2N\} = A + j B, \text{ où } A = \cos(m \pi / 2N) \text{ et } B = -\sin(m \pi / 2N) \quad (\text{II-91})$$

$$Z_2 = \sum_{k=0}^{N-1} y(k) \cdot \text{Exp}\{-j 2mk\pi / 2N\} = C + j D, \text{ où } C = \text{Re}[Z_2] \text{ et } D = \text{Im}[Z_2] \quad (\text{II-92})$$

Z<sub>2</sub> est l'expression d'une TFD N-points.

$$\text{Par suite : } \text{Re} [R(m)] = \text{Re} [Z_1 * Z_2] = A C - B D$$

(II-93)

Cette dernière relation, que nous appellerons « opération de base », signifie que le calcul de la TCD se fait en trois (03) étapes comme illustré par le diagramme-bloc suivant :

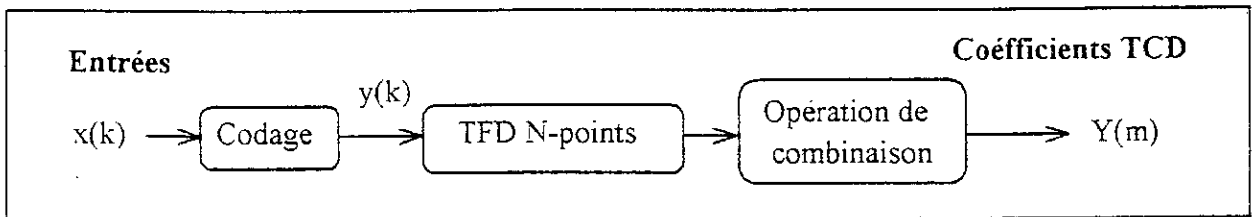


Figure II.14 - Processus de calcul de la TCD par la TFD.

• **Dérivation de l'algorithme TCD utilisant la TFD**

Dans [42] est dérivé un algorithme TCD utilisant les propriétés de la TFD et son algorithme rapide TFR tout en exploitant la parité de la dimension N et le fait que les échantillons entrées et les noyaux cosinus sont réels.

Le processus de calcul général obtenu, est représenté par le graphe de fluence suivant structuré en modules de transformée N/4. Ce processus est orienté vers une implantation hardware et sa conception architecturale se réduit à celle du module matriciel de base (transformée N/4) et sa reproduction 16 fois avec les différentes connexions.

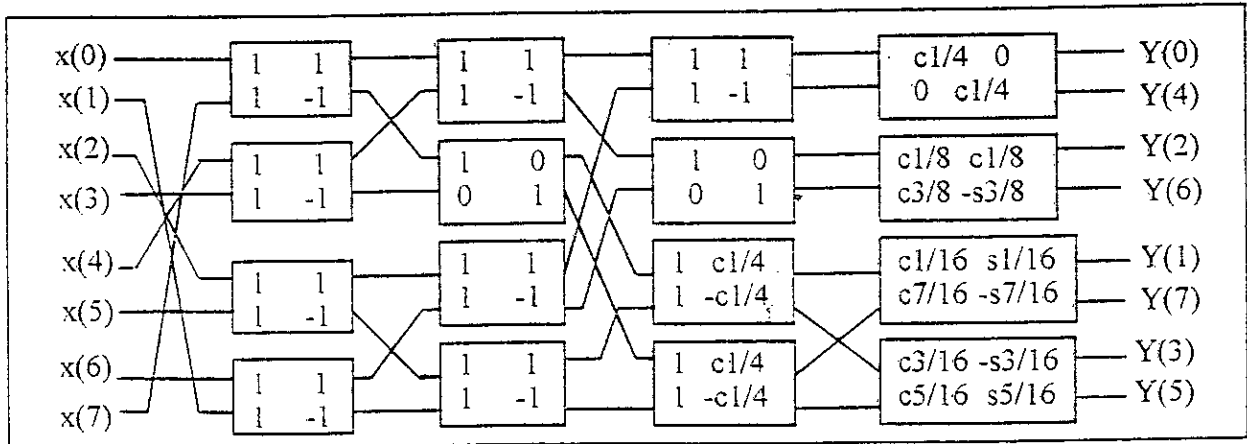


Figure II.15 - Processus de calcul matriciel de la TCD en utilisant la TFD

avec :  $c\ a/b = \cos(\pi a/b)$  et  $s\ a/b = \sin(\pi\ a/b)$ .

### 2.4.2 - DECOMPOSITION EN RADICAL 4

La décomposition en radical 4 a été utilisée dans beaucoup de travaux d'implémentation de la TCD [28]. Le principe est basé directement sur la propriété de symétrie de la TCD. En effet, grâce à un codage approprié des entrées/sorties, la matrice TCD peut être subdivisée en 4 parties symétriques 2 à 2. En rassemblant les pixels à leur tours 2 à 2, on aboutit à l'équation suivante régissant l'algorithme en radical 4 :

$$\begin{cases} Y(2k) = [A] (X(k) + X(7-k)) \\ Y(2k+1) = [B] (X(k) - X(7-k)) \\ k = 0, 1, 2, 3. \end{cases} \quad \text{où : } [A] = \begin{bmatrix} a & a & a & a \\ a & -a & -a & a \\ b & c & -c & -b \\ c & -b & b & -c \end{bmatrix} \quad \text{et } [B] = \begin{bmatrix} e & f & g & h \\ g & -e & -h & f \\ f & -h & -e & -g \\ h & -g & f & -e \end{bmatrix} \quad \text{(II-94)}$$

où :  $a = \cos(\pi/4)$ ,  $b = \cos(\pi/8)$ ,  $c = \cos(3\pi/8)$ ,  $e = \cos(\pi/16)$ ,

$$\text{et } f = \cos(3\pi/16), g = \cos(5\pi/16) \text{ et } h = \cos(7\pi/16) \quad \text{(II-95)}$$

La transformation inverse est donnée par les matrices transposées :

$$\begin{cases} X(k) = (1/2)[A^T] Y(2k) + (1/2)[B^T] Y(2k+1) \\ X(7-k) = (1/2)[A^T] Y(2k) - (1/2)[B^T] Y(2k+1) \\ k = 0, 1, 2, 3. \end{cases} \quad \text{(II-96)}$$

Nous savons qu'il faut 2 itérations pour exécuter l'algorithme à radical 4. Ceci est vérifié par l'équation de la TCD où dans la première itération, est effectuée l'addition des pixels 2 à 2, et dans la seconde la multiplication vecteur-matrice. Pour rendre l'algorithme hybride, il est nécessaire de créer dans le chemin direct de l'algorithme le cheminement évaluant l'équation de la TCDI tout en sachant que dans la première itération pour la TCDI, est effectué la multiplication vecteur-matrice, et dans la seconde l'addition des pixels 2 à 2. A cet effet, une architecture unifiée doit obligatoirement prendre en considération cette différence entre la TCD et la TCDI.

La figure suivante illustre le principe de l'architecture hybride :

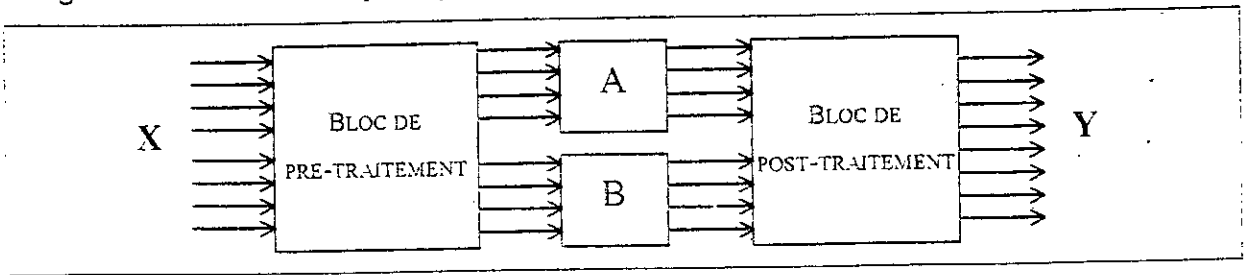


Figure II.16 - Structure générale de l'algorithme pour  $r = 4$

Les deux blocs pré-traitement et post-traitement sont identiques. Ces blocs se configurent en fonction d'un signal de commande externe indiquant la transformation à effectuer.

### 2.4.3 - DECOMPOSITION EN RADICAL 8

Dans ce cas, le nombre d'itération est égal à un, donc il n'y a aucune décomposition, et la matrice TCD sera une matrice  $8 \times 8$  donnée par :

$$A = 1/2 \begin{bmatrix} a & a & a & a & a & a & a & a \\ d & e & f & g & -g & -f & -e & -d \\ b & c & -c & -b & -b & -c & c & b \\ e & -g & -d & -f & f & d & g & -e \\ a & -a & -a & a & a & -a & a & -a \\ f & -d & g & e & -e & -g & d & -f \\ c & -b & b & -c & -c & b & -b & c \\ g & -f & e & -d & d & -e & f & -g \end{bmatrix} \quad \text{où : } \begin{aligned} a &= \text{Cos } \pi/4, & b &= \text{Cos } \pi/8, \\ c &= \text{Sin } \pi/8, & d &= \text{Cos } \pi/16, \\ e &= \text{Cos } 3\pi/16, & f &= \text{Sin } 3\pi/16, \\ \text{et } g &= \text{Sin } \pi/16. \end{aligned} \quad (\text{II-97})$$

Cette structure monobloc procure automatiquement une similitude totale entre les acheminements direct et inverse des données, à savoir la TCD et la TC DI.

### 2.4.4 - DISCUSSION

D'après les graphes de fluence obtenus, il résulte que les structures modulaires de calcul de la TCD 1-D en radical 2 ne permettent aucune symétrie entre les étages d'entrée et de sortie pour entrevoir une implémentation unifiée TCD/TC DI 2-D. A l'inverse, les calculs en radical 4 et 8 permettent de l'envisager.

L'augmentation du radical permet certes de venir à bout des exigences d'une structure hybride, à savoir la reconfigurabilité entre la TCD et la TC DI, mais introduit un problème d'ordre algébrique. En effet, pour les décompositions en radical 4 ou 8, l'algorithme est basé sur une multiplication vecteur-matrice qui génère une expression algébrique du genre :

$$Y(n) = \sum_{m=1}^r A(n,m) \cdot X(m), \quad \text{avec : } n,m = 1 \dots r \text{ et } r = 4 \text{ ou } 8. \quad (\text{II-98})$$

Le challenge se réduit finalement à trouver l'outil arithmétique adéquat pour arriver à notre architecture, comparer et choisir entre les décompositions en radical 4 ou 8.



### **3 - CONCLUSION**

D'une façon générale, les relations entre algorithmes rapides et architectures de traitement de signal ont évolué pratiquement de manière non-concourante depuis le développement explosif du traitement numérique. Les progrès rapides de la technologie aidant, on a même cru que le recours à des algorithmes rapides était devenu inutile, sinon nuisible à cause de la perte de régularité qu'ils introduisent dans le graphe de fluence de l'algorithme.

De plus, les algorithmes rapides sont, dans leur majorité, proposés sans grande préoccupation architecturale. L'objectif était de minimiser le nombre de multiplications, quels que soient les autres coûts associés (addition, régularité, taille des blocs).

Il reste que la prise en compte des influences mutuelles existant entre ces deux domaines, permet d'obtenir des algorithmes très efficaces une fois implantés. En effet, les algorithmes rapides fournissent un outil très utile pour travailler sur les architectures : des bornes de complexité. Si par exemple, on est proche de l'optimum avec un algorithme présentant une structure régulière, le challenge pourra se limiter à la recherche de variantes de cet algorithme qui s'implantent facilement sur l'architecture cible.

En résumé, une grande souplesse d'adaptation existe pour obtenir des implantations efficaces d'algorithmes rapides. La démarche générale à suivre est [43] :

1. Tout d'abord, « travailler » les algorithmes de base pour que leur structure mathématique devienne apparente;
2. Introduire de la redondance si nécessaire en regroupant des calculs à un niveau raisonnable;
3. Connaître des bornes de complexité pour le problème considéré : savoir de combien on se pénalise par rapport à l'algorithme le plus efficace, et avoir un éclairage efficace sur la structure mathématique du problème;
4. Analyser l'architecture cible : quelles sont les opérations élémentaires les plus efficaces? quelles sont les opérations à éviter le plus possible ?
5. Trouver l'ensemble des algorithmes rapides, expurgant la redondance introduite dans 2 qui soit le plus large possible.

## CHAPITRE III



### OUTILS ARITHMÉTIQUES

- \* MODE DE CALCUL HALF-LINE
- \* ARITHMÉTIQUE DISTRIBUÉE

## 1 - INTRODUCTION

Dans les applications numériques modernes telles que le traitement d'images, le traitement en temps réel est très déterminant. L'exigence de performances de plus en plus élevées, a conduit à chercher et développer de meilleures solutions technologiques et structurelles combinant les techniques de parallélisme, de pipe-line qui offrent une plus grande rapidité de calcul, plus de précision et une plus grande intégration avec une variété de méthodes de conception.

Cependant, toutes ces possibilités algorithmiques sont conditionnées par les modes de calcul utilisés par les opérateurs arithmétiques. L'arithmétique classique qui utilise toujours une propagation de retenue et une circulation parallèle des opérandes, ne satisfait plus aux performances recherchées et se trouve dépassée. L'apparition des modes de calcul On-Line et Half-Line et l'arithmétique distribuée, a apporté beaucoup de soulagements et d'améliorations.

## 2 - INCOMMODITES DE L'ARITHMETIQUE CLASSIQUE

L'arithmétique classique est le mode de calcul de base de toute l'informatique moderne, des petites calculatrices aux grands ordinateurs vectoriels, où la circulation des bits se fait des moins significatifs (LSB) au plus significatifs (MSB). Les calculs à réaliser, quels que soient leur volume et leur type, se ramènent presque toujours aux quatre opérations de base : addition, soustraction, multiplication, division. Les valeurs manipulées sont réelles et le traitement se ramène à des calculs sur des entiers. Le challenge se réduit à réaliser ces opérations de base le plus rapidement possible en choisissant les modes de représentations les mieux adaptés.

### 2.1 - Incommodité de la circulation parallèle des opérandes

En arithmétique classique, le traitement des opérandes se fait de façon parallèle. Sur le plan architectural, ce parallélisme se traduit par une grande surface et une grande complexité de la circuiterie engendrées par la taille des circuits de traitement. Cette complexité augmente linéairement avec la taille des opérandes.

Parmi les solutions souvent adoptées en circuiterie, on trouve les deux techniques suivantes :

- Le multiplexage des données dans le circuit avec un seul bus pour un traitement multi-opérandes. Ceci est efficace mais augmente le hardware et le temps d'exécution. Ce principe est utilisé dans des microprocesseurs pour le transfert des données dans des cases mémoires (cycle de lecture).
- la technique d'entrées sérielles qui s'adapte aux circuits spécialisés utilisant des registres à décalage entrée série/sortie parallèle, où toutes les données sont introduites bit par bit. Après

un temps de lecture égal au nombre de bits des données, celles-ci seront toutes en parallèles, prêtes à être utilisées. Cette méthode coûte aussi en temps et en espace.

### 2.2 - Incommodité de la propagation de la retenue

Les principes de « génération » et de « propagation » repris dans [44] sont basés sur des constatations très simples : lors de l'addition de deux nombres A et B écrits en numération simple de position ou en complément à 2, si à un certain pas de calcul  $i$  :

- $A_i = B_i$  alors il est inutile d'attendre le calcul de la retenue  $R_i$  pour calculer la retenue  $R_{i-1}$  et la somme  $S_{i-1}$  ( si  $A_i = B_i = 0$  alors  $R_{i-1} = 0$ , et si  $A_i = B_i = 1$  alors  $R_{i-1} = 1$ )
- $A_i \neq B_i$  alors  $R_{i-1} = R_i$ .

On peut tout de suite entrevoir le fait que l'utilisation combinée de ces deux principes, nous permettra d'accélérer le processus d'addition. Malheureusement, l'arithmétique classique n'exploite pas de telles occurrences, et le temps total d'une addition reste toujours une fonction linéaire de la taille des opérandes à traiter.

### 2.3 - Incommodité de la simultanéité

Le traitement en simultanéité a été introduit pour augmenter la vitesse des calculs. Il exige que les données soient présentes en même temps, ce qui est rarement le cas à cause des retards considérables résultant de l'attente des opérandes qui sont calculés en amont.

Il apparait d'après ce qui précède que l'idéal serait d'avoir une arithmétique capable d'engendrer des circuits de complexité moindre, indépendamment de la taille des opérandes et présentant des temps de calcul constants. Beaucoup de travaux d'investigations ont été fournis et continuent de l'être pour la recherche de telles arithmétiques. Et c'est ainsi que l'arithmétique série est venue apporter quelques possibilités très intéressantes.

## 3 - ARITHMETIQUE « SERIE » OU MODE DE CALCUL SERIEL

Les opérateurs arithmétiques se distinguent par la manière dont ils reçoivent leurs opérandes et fournissent leur résultat. Les *opérateurs parallèles* reçoivent au même instant tous les bits des opérandes et restituent en même temps les bits du résultat. Ce sont en général, les opérateurs les plus rapides mais ils prennent souvent beaucoup de place sur un circuit. Les *opérateurs série*, eux, reçoivent les bits des opérandes et fournissent les bits du résultat un par un, de façon séquentielle. On appelle *délai* d'un opérateur série le nombre de bits des opérandes nécessaires au calcul du premier chiffre du résultat [44].

Dans la plupart des additionneurs et multiplieurs série, les bits circulent en commençant par les poids faibles (LSB en tête : sens naturel de propagation des retenues). Par contre, pour la division et la comparaison, il faut faire circuler les bits en commençant par les poids forts (MSB en tête), ce qui nécessite le choix de systèmes particuliers d'écriture des nombres : les systèmes d'Avizienis. Les opérateurs série dans lesquels les bits circulent en commençant par les poids forts, sont appelés : *opérateurs en lignes*.

Il arrive souvent qu'un calcul puisse se décomposer en plusieurs tâches qui doivent s'exécuter séquentiellement. Si ce même calcul doit se répéter plusieurs fois pour des données différentes, il est avantageux de l'effectuer en *pipe-line*. Pour ceci, on commence par ralentir les tâches les plus rapides (en mémorisant temporairement les résultats intermédiaires dans des mémoires tampons) afin que toutes les tâches aient la même durée qu'on adoptera comme unité de temps. On commence par effectuer la première tâche sur un premier jeu de données, puis on effectue la seconde tâche sur ce premier jeu de données tout en effectuant la première tâche sur un second jeu de données, et ainsi de suite. On notera que le calcul en mode série n'est en fait qu'un cas particulier de calcul en pipe-line, où la décomposition en tâches se fait au niveau du bit.

En résumé, le mode de calcul sériel donne la possibilité d'organiser les opérations aussi bien à même d'apporter une solution aux besoins de précision qu'à ceux de rapidité des calculs. Il permet entre autre :

- le pipe-line et les calculs massifs favorisant un haut débit de calcul avec une surface réduite et l'implantation d'un parallélisme massif.
- une meilleure précision sans contraintes physiques, c'est-à-dire, puisque les opérands viennent en série, il suffit alors de faire entrer des bits et produire des bits résultats autant que les bits rentrent [45], [46], [47].

#### **4 - LES SYSTEMES REDONDANTS D'ECRIURE DES NOMBRES :**

Les systèmes de numérotation redondants ont été introduits en 1961 par Avizienis [48]. Les nombres qui y sont représentés, ont la particularité d'avoir plusieurs représentations vu qu'ils ont la possibilité d'avoir des *chiffres signés*, c'est à dire qui peuvent être *negatifs*. Nous conviendrons de noter les chiffres négatifs à l'aide d'une *barre*, afin d'éviter tout risque de confusion avec le signe des nombres négatifs ou le signe de soustraction.

Cette représentation introduite par Avizienis, utilise deux théorèmes importants que nous rappelons ici.

**Théorème 1** : Soient  $S_{b, w, q, n}$  le système d'écriture des nombres en base  $b$  avec  $n$  chiffres pris dans l'ensemble  $\{w, w+1, w+2, \dots, w+q\}$  (où  $w$  est un entier quelconque -éventuellement négatif- et  $q$  un entier strictement positif) et  $G_{b, w, q, n}$  l'ensemble des nombres que l'on peut écrire avec le système  $S_{b, w, q, n}$ .

- Si  $q < b-1$ , les éléments de  $G_{b, w, q, n}$  ne sont pas consécutifs : entre deux de ses éléments, il peut y avoir un entier n'appartenant pas à  $G_{b, w, q, n}$ . Le système  $S_{b, w, q, n}$  ne peut donc être utilisé efficacement comme système d'écriture de nombres.
- Si  $q = b-1$ ,  $G_{b, w, q, n} = \{ \text{entiers } p \text{ tq : } w(b^n - 1)/(b-1) \leq p \leq (w + b-1) \cdot (b^n - 1)/(b-1) \}$ . De plus, l'écriture de chacun de ces entiers dans  $S_{b, w, q, n}$  est unique (c'est la numération simple de position).
- Si  $q > b-1$ ,  $G_{b, w, q, n} = \{ \text{entiers } p \text{ tq : } w(b^n - 1)/(b-1) \leq p \leq (w + q) \cdot (b^n - 1)/(b-1) \}$ . L'écriture d'un entier dans le système  $S_{b, w, q, n}$  n'est plus nécessairement unique : un tel système est dit *redondant*.

Il paraît évident que la redondance entraîne un certain « gâchis », car pour représenter un même nombre d'entiers différents il faudra un plus grand nombre de bits qu'avec un autre système de notation; toutefois, les systèmes redondants sont particulièrement intéressants car ils permettent d'effectuer des additions de manière totalement parallèle et exempte de toute propagation de retenue. Grosso modo, cela est dû au fait que localement, on peut réécrire d'une autre manière des chaînes de chiffres qui sans cela auraient généré une retenue. Pour cela, Avizienis s'est intéressé à une classe particulière de systèmes  $S_{b, w, q, n}$  redondants dits *systèmes quasicanoniques* caractérisés par les exigences suivantes:

- **Exigence de symétrie** :  $S_{b, w, q, n}$  doit permettre la représentation d'entiers négatifs soit  $w < 0$ . De plus, un tel système ne sera réellement utilisable que si l'on peut facilement passer de l'écriture d'un nombre  $x$  à celle de son opposé, en prenant les opposés de chacun des chiffres de  $x$ . Il faut donc que  $S_{b, w, q, n}$  soit *symétrique*, c'est-à-dire que l'ensemble des chiffres utilisés soit de la forme :  $\{ -\rho, -\rho+1, -\rho+2, \dots, 0, 1, \dots, \rho-2, \rho-1, \rho \}$  ( $\rho = -w > 0, q = 2 \cdot \rho$ ).
- **Exigence de comparativité** : Pour que  $S_{b, -\rho, 2 \cdot \rho, n}$  soit réellement utilisable, il faut qu'il permette aisément de comparer des entiers ou de décider du signe d'un entier. Avizienis a démontré que pour que tout nombre écrit dans ce système, ait le signe de son premier chiffre non nul à partir de la gauche est que :  $\rho \leq b-1$ . Un tel système sera dit *comparable*.

- **Addition parallèle dans les systèmes d'Avisienis :**

**Théorème 2** : Tous les nombres entiers peuvent être représentés en base  $b \geq 2$  avec des chiffres pris dans l'ensemble  $\{-a, \dots, -1, 0, 1, \dots, a\}$  si et seulement si  $2a + 1 \geq b$ . De plus, si  $2a - 1 \geq b$ , alors il existe un algorithme pour effectuer des additions de manière complètement parallèle sans propagation de retenue.

## **5 - ARITHMETIQUE ON-LINE ET HALF-LINE**

Dans cette section, nous tenterons d'apporter une solution à l'implémentation de la TCD pour un traitement en temps réel, en utilisant un outil arithmétique très puissant.

### **5.1 - LE MODE DE CALCUL ON-LINE**

Les premiers algorithmes du mode On-line ont été démontrés en 1977 par Trividi et Ercegovic [49]. Ce mode de calcul est caractérisé [50] [51] [52] par :

- Les opérandes sont introduits dans l'opérateur, bit par bit, MSB en tête ( le calcul est entamé dès la connaissance des premiers bits de chaque opérande);
- Le résultat est aussi obtenu bit par bit, MSB toujours en tête, mais avec un retard  $p$  tel qu'au pas  $j$ , quand le  $j^{\text{ième}}$  bit des opérandes est introduit, le  $(j - p)^{\text{ième}}$  bit du résultat est généré;
- Le retard  $p$  est très petit devant la taille des opérandes;
- Les opérandes et le résultat sont représentés dans un système redondant.

### **5.2 - LE MODE DE CALCUL HALF-LINE**

Ce mode représente le cas particulier du mode On-line où un ensemble des opérandes est connu d'avance. Les opérandes connus sont introduits entièrement dès le lancement de l'opération, tandis que les autres opérandes sont introduits bit par bit, MSB en tête. Le résultat est obtenu bit par bit après un retard  $p$  spécifique à l'opération.

### **5.3 - SPECIFICITES DES MODES ON-LINE ET HALF-LINE**

- Par définition, le retard  $p$  est le nombre de chiffres nécessaires à l'opérateur pour pouvoir générer le premier chiffre du résultat. A la fin du calcul, il faut décaler la valeur calculée de  $p$  bits pour retrouver la valeur réelle du résultat.

- **Equations de récurrence :**

Soit un opérateur On-Line noté OP et défini comme suit :  $Z = OP(X, Y)$  (III-1)

A un pas  $i$  du calcul, les équations de récurrence sur les opérandes X et Y connus bits par bits et sur le résultat Z de l'opérateur, sont données par :

$$X[i] = X[i-1] + x_i \cdot b^{-i}; \quad Y[i] = Y[i-1] + y_i \cdot b^{-i}; \quad Z[i] = Z[i-1] + z_i \cdot b^{-i} \quad (\text{III-2})$$

où  $x_i, y_i$  et  $z_i \in \{-p, -p+1, \dots, 0, 1, \dots, p-1, p\}$  et  $b/2 \leq p \leq b-1$ ; ( $2 \cdot p < b-1$ ).

- Par définition, on appelle *résidu partiel* à un pas  $i$  du calcul l'expression suivante :

$$R[i] = b^i \cdot [b^{-p} \cdot OP(X[i], Y[i]) - Z[i]] \quad (\text{III-3})$$

- De la même façon, on appelle *résidu complet* l'expression suivante :

$$H[i] = R[i] + z_i \quad (\text{III-4})$$

- *Condition de convergence* de l'opérateur :

A tout pas de calcul  $i$ , le résidu partiel doit être compris dans un intervalle tel que :

$$\forall i / 0 \leq i \leq n : |R[i]| \leq \varepsilon \quad \text{avec } 0 < \varepsilon < 1 \quad (\text{III-5})$$

ce qui nous donne une *condition d'arrondissement* sur le résultat : [46]

$$\forall i / 0 \leq i \leq n : (z_i - \varepsilon) \leq H[i] \leq (z_i + \varepsilon) \quad (\text{III-6})$$

cette dernière relation signifie que la valeur du bit résultat est déterminée par une procédure de sélection en fonction de la valeur du résidu complet

#### 5.4 - EVALUATION DE LA TCD PAR LE HALF-LINE

Le calcul de la T.C.D par le mode Half-Line est organisé comme suit :

- Développement de l'algorithme Half-Line de l'expression généralisée de la T.C.D;
- Développement pour la TCD à radical 4, de l'algorithme Half-Line des additions (et soustractions) induites par les étages de pré-traitement et post-traitement.

##### 5.4.1 - Evaluation de l'expression généralisée de la T.C.D

L'expression de la T.C.D est donnée sous la forme généralisée des transformées

orthogonales :

$$Y = \sum_{k=0}^{N-1} A_k \cdot X_k \quad (\text{III-7})$$

où  $X_k$ ,  $Y$  et  $A_k$  représentent respectivement les pixels de l'image traitée, les coefficients transformés T.C.D à calculer, et les noyaux cosinus connus avant le lancement de l'opération.

L'exécution de cette expression paramétrée complexe avec le mode parallelo-séquentiel, le digit le moins significatif en tête peut se faire en  $N$  parallèles multiplications et  $(N-1)$  parallèles additions en  $\log_2 N$  pas [52]. Aussi, une opération d'un certain pas ne peut être entamée avant que celles de l'étape précédente soient terminées.

Pour y remédier, nous adoptons le mode de calcul Half-Line :

$X_k$ ,  $Y$  et  $A_k$  sont écrits dans un système redondant symétrique comparable  $S_{b, -p, 2p, n}$  tq :

- $X$  a une précision de  $n$  bits, connus bit par bit, MSB en tête;
- $Y$  a une précision de  $n' = n + p$  bits,  $p$  étant le retard de l'opération;
- $A_k < b^{-r}$  et  $r \geq 0$ ,  $b$  étant la base du système redondant.



Au pas  $i$  du calcul, les  $i$  premiers bits de  $X$  sont introduits et les  $(i-p)$  bits du résultat  $Y$  sont générés, l'opération de base est alors :

$$Y[i] = b^{-p} \cdot \sum_{k=0}^{N-1} A_k X_k[i] \quad (III-8)$$

Les équations de récurrence sur les entrées et les sorties sont données par :

$$X_k[i] = X_k[i-1] + x_{ki} \cdot b^{-i} \quad \text{et} \quad Y[i] = Y[i-1] + y_i \cdot b^{-i} \quad (III-9)$$

où :  $x_{ki}, y_i \in \{-\rho, \dots, -1, 0, 1, \dots, +\rho\}$  ;  $b/2 \leq \rho \leq b-1$

Le Résidu partiel  $R[i]$  est donné au pas de calcul  $i$  par :

$$R[i] = b^i \cdot [ b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot X_k[i] - Y[i] ], 0 \leq i \leq n \quad (III-10)$$

en y reportant les équations de récurrence, nous obtenons :

$$\begin{aligned} R[i] &= b^i \cdot [ b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot X_k[i-1] + \sum_{k=0}^{N-1} A_k \cdot x_{ki} \cdot b^{-i} - Y[i-1] - y_i \cdot b^{-i} ] \\ &= b \cdot ( b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot X_k[i-1] \cdot b^{i-1} - Y[i-1] \cdot b^{i-1} ) + b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot x_{ki} \cdot b^i - y_i \cdot b^i \end{aligned} \quad (III-11)$$

posons :

$$L[i] = b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot x_{ki} \quad (III-12)$$

$L[i]$  représente une accumulation partielle de multiplications bit-valeur  $(A_k \cdot x_{ki})$ .

Nous obtenons une équation de récurrence sur le résidu :

$$R[i] = b \cdot R[i-1] + L[i] - y_i \quad (III-13)$$

$$R[i-1] = b^{i-1} \cdot [ b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot X_k[i-1] - Y[i-1] ] \quad (III-14)$$

L'équation de récurrence sur le résidu complet est obtenue comme suit :

$$\begin{aligned} H[i] &= R[i] + y_i = b \cdot R[i-1] + L[i] \\ &= b \cdot H[i-1] + L[i] - b \cdot y_{i-1} \end{aligned} \quad (III-15)$$

Les conditions de convergence et d'arrondissement s'écrivent respectivement :

$$|R[i]| < \varepsilon \quad \text{et} \quad (y_i - \varepsilon) \leq H[i] \leq (y_i + \varepsilon), \quad 0 < \varepsilon < 1; \quad 0 \leq i \leq n \quad (III-16)$$

Ce qui nous amène à la fonction de sélection notée  $S(H[i])$ , générant le bit-résultat  $y_i$  suivant les valeurs du résidu  $H[i]$ , telle que :

$$y_i = S(H[i]) = \begin{cases} -\rho & \text{si} & -\rho - \varepsilon \leq H[i] < -\rho + \varepsilon \\ \dots & \dots & \dots \\ -1 & \text{si} & -1 - \varepsilon \leq H[i] < -1 + \varepsilon \\ 0 & \text{si} & -\varepsilon \leq H[i] < +\varepsilon \\ +1 & \text{si} & 1 - \varepsilon \leq H[i] < 1 + \varepsilon \\ \dots & \dots & \dots \\ +\rho & \text{si} & \rho - \varepsilon \leq H[i] \leq \rho + \varepsilon \end{cases} \quad (III-17)$$

Puisque nous utilisons un système redondant symétrique, il convient de vérifier la condition d'arrondissement par l'une des valeurs maximales (valeurs sup) soit :  $H[i]_{\max} \leq \rho + \varepsilon \quad (III-18)$

Nous savons que : 
$$\begin{cases} |R[i]| \leq \varepsilon \\ -(\varepsilon + \rho) \leq H[i] \leq (\varepsilon + \rho) \quad \text{pour } 0 \leq i \leq n \\ H[i] = b \cdot R[i-1] + L[i] \end{cases} \quad \text{(III-19-a)}$$

Par suite : 
$$-\varepsilon(1-b) - \rho \leq L[i] \leq -\varepsilon(1-b) + \rho \quad \text{(III-19-b)}$$

ou encore : 
$$\rho(1 - \varepsilon/k) \leq L[i] \leq \rho(1 + \varepsilon/k) \quad \text{(III-19-b)}$$

où k est un facteur de redondance défini par :  $k = \rho / (b - 1)$

d'où : 
$$L[i]_{\max} \leq \rho \cdot (1 - \varepsilon/k) \quad \text{(III-20-a)}$$

or : 
$$L[i]_{\max} = \max |L[i]| \cdot b^{-p} \quad \text{(III-20-b)}$$

Nous aurons alors : 
$$b^{-p} \leq [\rho \cdot (1 - \varepsilon/k)] / \max |L[i]| \quad \text{(III-21)}$$

Par ailleurs : 
$$\max |L[i]| = \max \left( \sum_{k=0}^{N-1} A_k \cdot x_{ki} \right) = \sum_{k=0}^{N-1} \max(A_k) \cdot \max(x_{ki}) \quad \text{(III-22)}$$

puisque la somme est faite sur N opérands, alors : 
$$\max |L[i]| = N \cdot b^{-r} \cdot \rho \quad \text{(III-23)}$$

Enfin, nous obtenons des bornes sur ε et sur le retard p de notre opération :

$$b^{-p} \leq \frac{(1 - \varepsilon/k) \cdot b^r}{N} \quad \text{et} \quad \varepsilon \leq k(1 - N \cdot b^{-p-r}) \quad \text{(III-24)}$$

Dans [53], l'arithmétique des intervalles est exploitée pour comparer le résidu H[i] sur un nombre limité de bits, noté ψ. Cette arithmétique offre une condition de continuité entre les intervalles pour y confiner le résidu H :

$$\varepsilon \geq (1/2) - b^{-\psi-1} \quad \text{(III-25)}$$

Par suite, la valeur minimale de p est donnée par : 
$$b^{-p} \leq \frac{b^r}{N} \left( 1 - \frac{1/2 - b^{-\psi-1}}{K} \right) \quad \text{(III-26)}$$

Le tableau suivant donne les valeurs du retard p et du nombre de bits ψ de H en fonction des valeurs de b, r et N :

	Radical 4	Radical 8
N	4	8
b	2	2
r	2	3
ε	1/2	1/2
p	2	3
ψ	1	1

Tableau III-1 - Retard p et nombre de bits ψ de H en fonction de b, r et N.

Enfin, l'algorithme half-line de l'expression généralisée de la T.C.D : 
$$Y = \sum_{k=0}^{N-1} A_k X_k[i]$$
 est donné ci-après en radical 8 pour une base b = 2. En radical 4, seules les valeurs de N et p changent.

Début

$N = 8; p = 3; R = 0; Y = 0;$

Pour  $i = 0$  à  $N + p - 1$  :

Faire

$L[i] = 0;$

Pour  $k = 0$  à  $N - i$  :

Faire

$L[i] = L[i] + 2^{-p} \cdot x_{ki} \cdot A_k;$

Fait

$H = 2 \cdot R + L[i];$

$$y_i = S(H) = \begin{cases} -1 & \text{si } -3/2 \leq H < -1/2 \\ 0 & \text{si } -1/2 \leq H < 1/2 \\ 1 & \text{si } 1/2 \leq H \leq 3/2 \end{cases}$$

$R = H - y_i;$

$Y = Y + y_i \cdot 2^{-i};$

Fin

$Y = Y \cdot 2^p;$

Fin

5.4.2 - Algorithme de l'addition Half-line

Cette opération est utilisée dans l'étage de pré-traitement et l'étage de post-traitement de la figure II.16, de la T.C.D en radical 4.

L'addition On-Line s'écrit comme suit :  $Z = OP(X, Y) = X + Y$  (III-27)

le résultat Z et les opérandes X et Y sont représentés dans un système redondant symétrique et comparable  $S_{2, -1, 2, n}$  (base  $b = 2$  et  $p = 1$ ). Les équations de récurrence sont les suivantes :

$$X[i] = X[i-1] + x_i \cdot 2^{-i}; \quad Y[i] = Y[i-1] + y_i \cdot 2^{-i} \quad \text{et} \quad Z[i] = Z[i-1] + z_i \cdot 2^{-i} \quad \text{(III-28)}$$

où :  $x_i, y_i$  et  $z_i \in \{-1, 0, +1\}$ . Le résidu partiel s'écrit au pas de calcul  $i$  :

$$\begin{aligned} R[i] &= 2^i \cdot [2^{-p} \cdot (X[i] + Y[i]) - Z[i]], \quad 0 \leq i \leq n \\ &= 2^i \cdot [2^{-p} \cdot (X[i-1] + Y[i-1] + 2^{-i} \cdot (x_i + y_i)) - Z[i-1] - z_i \cdot 2^{-i}] \\ &= 2 \cdot (2^{i-1} \cdot [(X[i-1] + Y[i-1]) - Z[i-1]] \cdot 2^{i-1}) + 2^p \cdot (x_i + y_i) - z_i \\ &= 2 \cdot R[i-1] + L[i] - z_i \end{aligned} \quad \text{(III-29)}$$

$$\text{avec : } R[i-1] = 2^{i-1} \cdot [2^{-p} \cdot (X[i] + Y[i-1]) - Z[i-1]] \quad \text{et} \quad L[i] = 2^p \cdot (x_i + y_i) \quad \text{(III-30)}$$

L'équation de récurrence sur le résidu complet est obtenue comme suit :

$$H[i] = R[i] + z_i = 2 \cdot R[i-1] + L[i] = 2 \cdot H[i-1] + L[i] - 2 \cdot z_{i-1} \quad \text{(III-31)}$$

Les conditions de convergence et d'arrondissement s'écrivent respectivement :

$$-1/2 \leq R[i] \leq +1/2 \quad \text{et} \quad (z_i - 1/2) \leq H[i] \leq (z_i + 1/2) \quad \text{(III-32)}$$

Ce qui nous amène à la fonction de génération du bit-résultat  $z_i$  :

$$z_i = S(H) = \begin{cases} -1 & \text{si } -3/2 \leq H < -1/2 \\ 0 & \text{si } -1/2 \leq H < 1/2 \\ +1 & \text{si } 1/2 \leq H \leq 3/2 \end{cases} \quad \text{(III-33)}$$

Le retard  $p$  est donné par la condition de convergence :

$$(|R[i]| \leq \varepsilon) \Leftrightarrow (2.\varepsilon + |L[i]|_{\max} + \rho \leq \varepsilon) \tag{III-34}$$

or :  $|L[i]|_{\max} = 2^{-p} . (x_{i \max} + y_{i \max}) = 2^{1-p} . \rho \tag{III-35}$

ce qui donne un retard :  $p_{\min} = 2 \tag{III-36}$

Ce retard est le même pour la convergence de l'algorithme de soustraction On-Line, dont le développement est le même que celui de l'addition avec la différence d'inverser les bits de  $Y$ .

L'algorithme Half-Line de l'addition se présente alors comme suit :

```

Début
  p = 2; R = 0; Z = 0;
  Pour i = 0 à n - p - 1 :
    Faire
      H = 2 . R + 2-p . (xi + yi);
      zi = S(H) = { -1 si -3/2 ≤ H < -1/2
                  { 0 si -1/2 ≤ H < 1/2
                  { 1 si 1/2 ≤ H ≤ 3/2
      R = H - yi;
      Z = Z + zi . 2-i;
    Fin
  Z = Z . 2p;
Fin
    
```

**5.5 - STRUCTURE GENERALE D'UN ALGORITHME HALF-LINE :**

Le développement d'un algorithme Half-line se fait comme dans les deux cas que nous avons traités, en deux étapes :

- Calcul de  $L[i]$ ;
- Calcul de  $H[i]$ ,  $S(H[i])$  et génération des bits résultats.

Ainsi, la structure de l'algorithme se subdivise en deux blocs :

- *Bloc de normalisation* opérant le calcul de  $L[i]$ ;
- *Bloc de sélection et génération* calculant  $H[i]$ ,  $S(H[i])$  et générant les bits résultats.

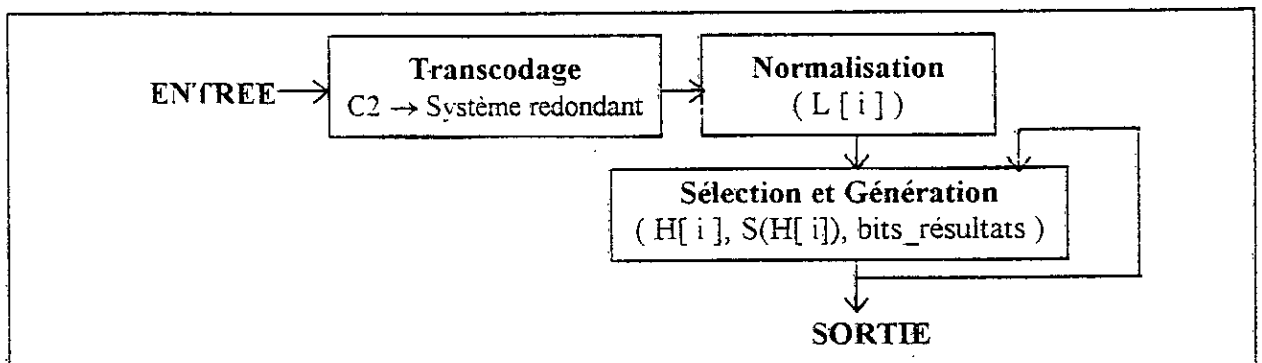


Figure III-1 - Structure générale d'un algorithme Half-line.

**5.6 - IMPLEMENTATION DE L'EXPRESSION GENERALISEE DE LA T.C.D [54]**

L'implémentation de l'algorithme d'évaluation de l'expression généralisée de la T.C.D, obéit à la structure générale présentée ci-dessus. Il s'agit de développer et de concevoir une architecture adaptée à chaque bloc de l'algorithme.

**5.6.1 - LE BLOC DE NORMALISATION**

Ce bloc opère le calcul de l'expression (III-12) :  $L [ i ] = b^{-p} \cdot \sum_{k=0}^{N-1} A_k \cdot x_{ki}$

L'évaluation de la somme des multiplications bit-valeur  $L [ i ]$  peut se faire par l'utilisation de mémoires ROMs. L'idée serait de prédire les valeurs de  $L [ i ]$  en fonction des entrées  $x_{ki}$ , en les prélevant d'une ROM où toutes les valeurs possibles de  $L [ i ]$  sont pré-stockées. Le décodage de cette ROM se fera par les entrées  $x_{ki}$ .

Puisque  $x_{ki} \in \{-1, 0, +1\}$ , le nombre de combinaisons possibles des valeurs de  $L [ i ]$  est :  $C_L^{x_{ki}} = 3^N$ . La taille d'une telle ROM sera de 6561 mots pour  $N = 8$ , et de 81 mots pour  $N=4$ . Pour alléger les contraintes de temps d'accès et de décodage d'adresses de ces mémoires, il convient de réduire leurs capacités en réécrivant la somme  $L [ i ]$  comme suit :

$$L [ i ] = 2^{-p} \left\{ \sum_{k=0}^{N/Q-1} A_k \cdot x_{ki} + \dots + \sum_{k=(Q-1)N/Q}^{N-1} A_k \cdot x_{ki} \right\}, Q = 2, 4 \text{ ou } 8 \quad \text{(III-37)}$$

Cette relation montre que la réduction de la capacité mémoire par la décomposition de  $L [ i ]$ , s'accompagne d'opérations d'addition. Le tableau suivant donne le nombre de mémoires obtenues et le nombre d'additionneurs à retenue conservée CSA (Carry Save Adders) [55] que nous utiliserons pour éviter la propagation des retenues et assurer un traitement en temps réel :

	Nombre de ROMs	Taille des ROMs	Nombre d'additionneurs CSAs
T.C.D en radical 4	1	81 mots	0
	2	9 mots	1
	4	3 mots	2
T.C.D en radical 8	1	6561 mots	0
	2	81 mots	1
	4	9 mots	2
	8	3 mots	6

Tableau III-2 : Choix du nombre de ROMs suivant les CSAs

**5.6.1.1 - Proposition d'une ROM [28]**

En considérant des ROMs de 9 mots, les coefficients  $A_i$  seront pris par couples  $(A_i, A_{i+1})$ ,  $i$  allant de 0 à  $N-2$ . Dans chaque ROM, on stocke les différentes combinaisons de ces couples, soit :  $0, A_i, -A_i, A_{i+1}, -A_{i+1}, (A_{i+1} + A_i), -(A_{i+1} + A_i), (A_i - A_{i+1}), -(A_i - A_{i+1})$ .

En supprimant la case mémoire qui contient la valeur zéro et en exploitant la symétrie de ces combinaisons, nous adoptons la structure suivante :

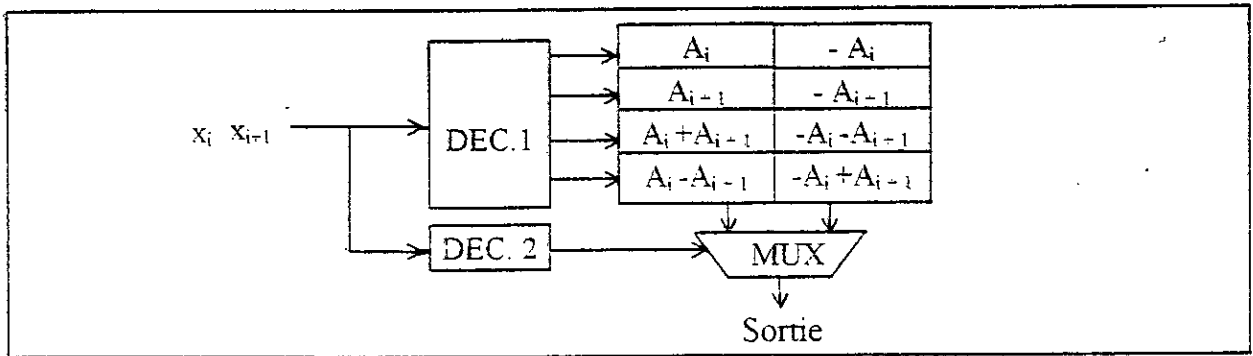


Figure III.3 - ROM de 4 mots réalisant la T.C.D.

Les éléments des deux colonnes de cette ROM sont de signes opposés. Un élément et son opposé seront décodés par une même adresse et c'est le décodeur DEC.2 qui validera l'une des entrées du multiplexeur (la valeur ou son opposé) en fonction des bits d'entrées  $x_i$  et  $x_{i-1}$ .

L'utilisation d'une commande « C », mise à 0 lors de la TCD et à 1 lors de la TC DI, optimisera plus la conception de cette ROM et l'architecture de notre ASIC. En effet, en stockant sur la même ligne de la ROM avec la même configuration les combinaisons des coefficients TC DI,  $(A_i)^t$  donnés par l'équation II.7, avec leurs correspondantes de la TCD, nous obtenons la ROM à double-buffer [28] de 4 mots et 4 colonnes, de la figure III-4. L'évaluation de  $L[i]$  avec de telles mémoires, nécessite des additionneurs CSAs pour sommer les sorties des ROMs utilisées.

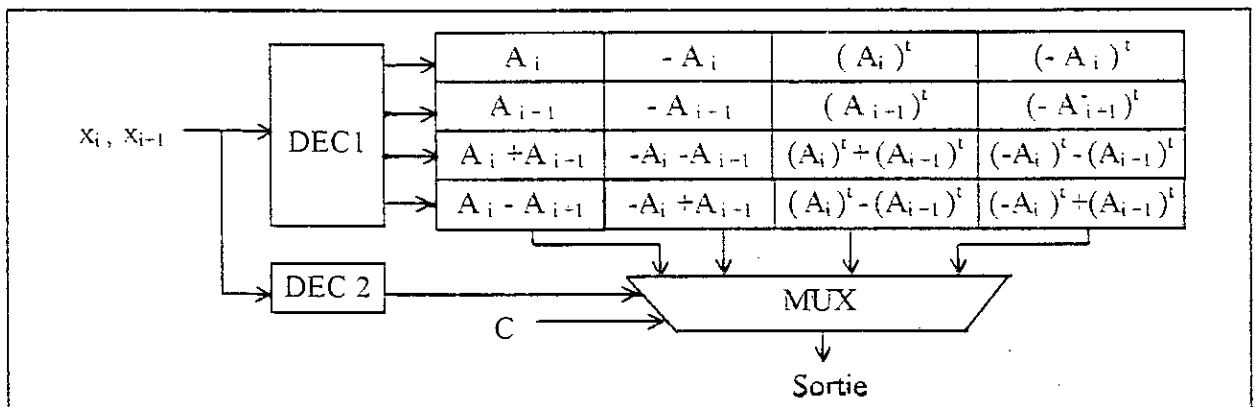


Figure III.4 - ROM de 4 mots réalisant la T.C.D et la T.C.D.I.

**5.6.1.2 - Circuits de décodage de la ROM**

Le décodage ligne et colonne de la ROM est assuré par les décodeurs DEC1 et DEC2 par la combinaison des bits d'entrées  $x_i$  et  $x_{i-1}$  que nous écrirons en représentation « signe-valeur » :  $(x_{i:s}, x_{i:v})$  et  $(x_{i+1:s}, x_{i+1:v})$ . Puisque  $x_i$  et  $x_{i-1}$  sont pris dans l'ensemble  $\{-1, 0, +1\}$ , ils auront trois représentations possibles (1, 1), (0, 0) et (0, 1).

La commande « C » aiguille la sortie du multiplexeur pour un calcul TCD ou son inverse TC DI.

Après simplification, les équations logiques en sortie des deux décodeurs s'écrivent :

• **Décodeur DEC.1 :**

$$E_{C0} = \bar{x}_{i+1:s} \cdot x_{i:v} \tag{III-38-a}$$

$$E_{C1} = x_{i+1:v} \cdot x_{i:v} \tag{III-38-b}$$

$$E_{C2} = x_{i-1:v} \cdot x_{i:v} \cdot (\overline{x_{i-1:s} \oplus x_{i:s}}) \tag{III-38-c}$$

$$E_{C3} = x_{i-1:v} \cdot x_{i:v} \cdot (x_{i-1:s} \oplus x_{i:s}) \tag{III-38-d}$$

• **Décodeur DEC.2 :**

$$E_L = x_{i-1:v} + x_{i:v} \cdot (\overline{x_{i-1:s} \oplus x_{i:s}}) \tag{III-39}$$

Ces équations sont réalisées par le circuit suivant :

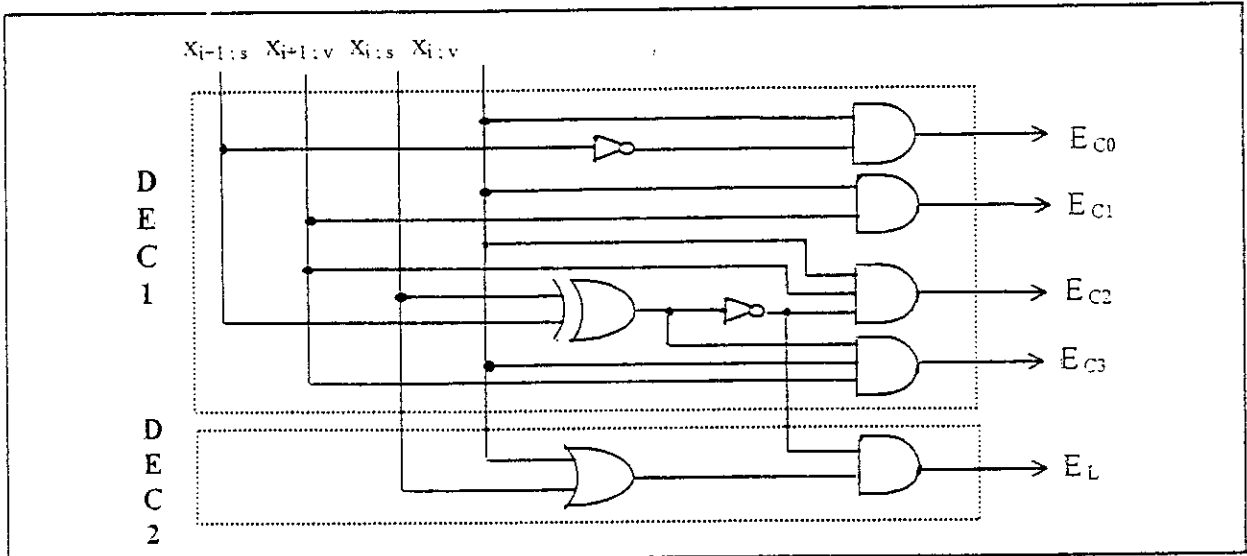


Figure III-5 - Le circuit de décodage de la ROM .

**5.6.1.3 - Additionneur à retenue conservée (CSA) [55]**

Nous avons vu précédemment que la réduction de la capacité mémoire par décomposition de l'accumulation partielle  $L[i]$ , s'accompagne des additions des différentes valeurs en sortie des mémoires utilisées.

Ces additions multiples sont réalisées en parallèle par des additionneurs à retenue conservée CSA (Carry Save Adder), attractifs par leur nature parallèle, leur faible exigence hardware et leur sauvegarde de la retenue jusqu'à ce que toutes les additions soient exécutées pour compléter la propagation.

Ces additionneurs sont décrits dans l'annexe A où nous explicitons l'utilisation :

- d'un additionneur conventionnel dit à propagation de retenue CPA (Carry-propagate Adder) pour sommer à la dernière étape les vecteurs « Somme » et « Retenue » du CSA ;
- ainsi que la logique de retenue anticipée des additionneurs CLA (carry-look ahead) dans le CPA pour accélérer la propagation de la retenue finale.

**5.6.1.4 - CSAs multi-niveaux**

Pour réaliser des additions multi-opérandes, nous pouvons utiliser un nombre de CSAs interconnectés comme un arbre additionneur multi-niveaux. La figure III-6 montre des structures d'arbres CSAs pour l'addition de  $k=3, 4, 8$  nombres. Deux entrées bouclées dans chaque cas, sont nécessaires pour accumuler la somme partielle et la retenue partielle décalée à gauche d'un bit. La flèche sur les retenues sortantes, indique qu'elles sont décalées à gauche d'un bit avec l'introduction d'un « 0 » de l'extrémité droite avant d'être introduites dans les CSAs.

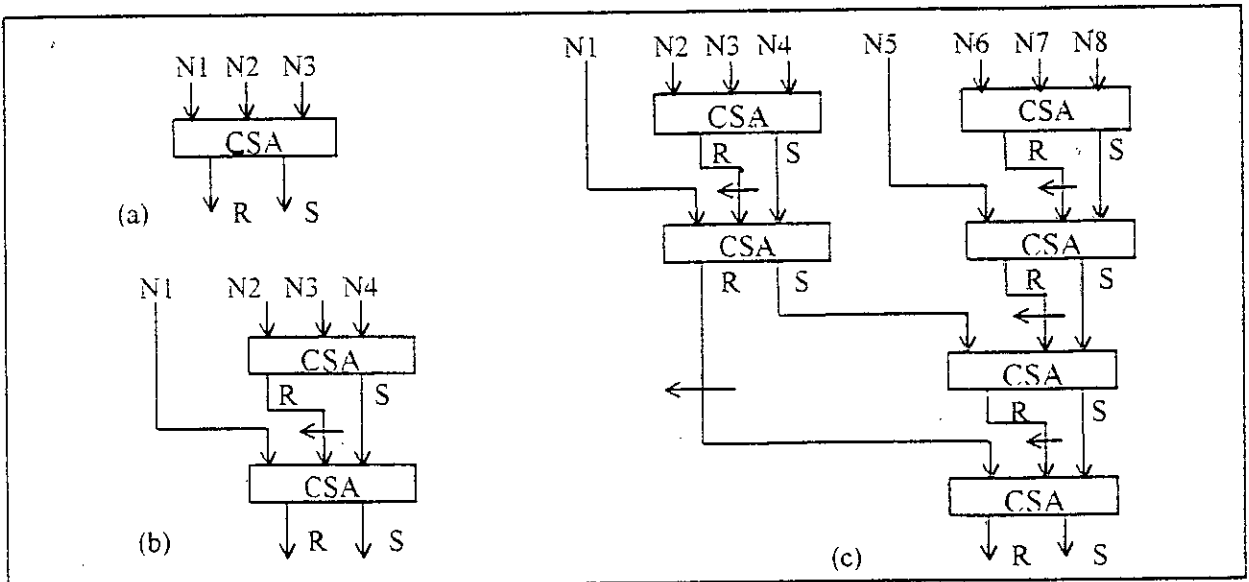


Figure III-6 - Arbres CSAs : (a) : à 3 entrées (CSA 3-2), (b) : à 4 entrées (CSA 4-2), (c) : à 8 entrées (CSA 8-2).

**5.6.1.5 - Architecture du bloc de normalisation de la TCD/TCDI en radical 8**

L'architecture du bloc de normalisation en radical 8 est donnée par la figure suivante :

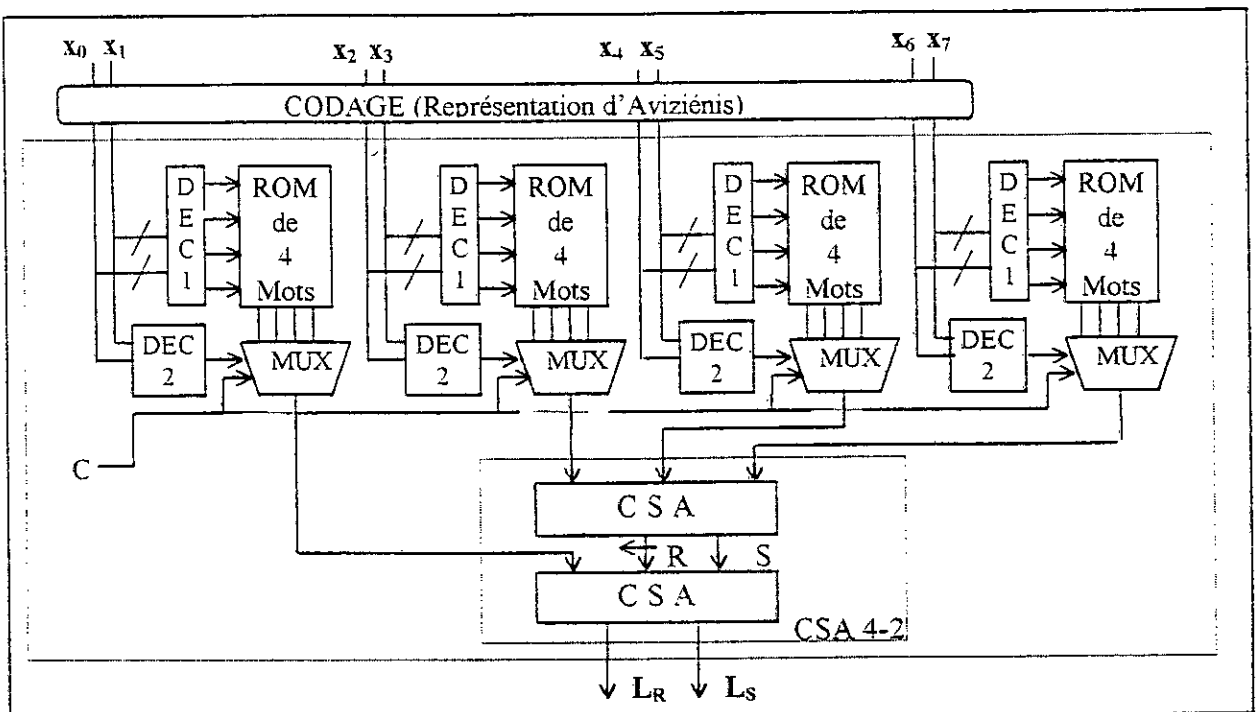


Figure III-7- Architecture du bloc de Normalisation TCD/TCDI en radical 8.



**5.6.1.6 - Architecture du bloc de normalisation de la TCD/TCDI en radical 4**

Comme illustré par la figure II.16 du chapitre précédent, la structure de l'architecture hybride de la TCD/TCDI en radical 4, présente un bloc de pré-traitement et un bloc de post-traitement avant et après le bloc de calcul de la transformée considérée, la TCD ou son inverse.

Le bloc de pré-traitement est un bloc additionneur/soustracteur dans le cas de la TCD, et un suiveur de données dans le cas de la TC DI. A l'inverse, le bloc de post-traitement est un suiveur de données dans le cas de la TCD, et un bloc additionneur/soustracteur dans le cas de la TC DI.

Ces deux blocs étant similaires, nous donnons ci-après seulement la structure du bloc de pré-traitement avec des étages de multiplexeurs/démultiplexeurs pour un aiguillage adéquat selon une commande externe pour réaliser le calcul voulu, direct ou inverse.

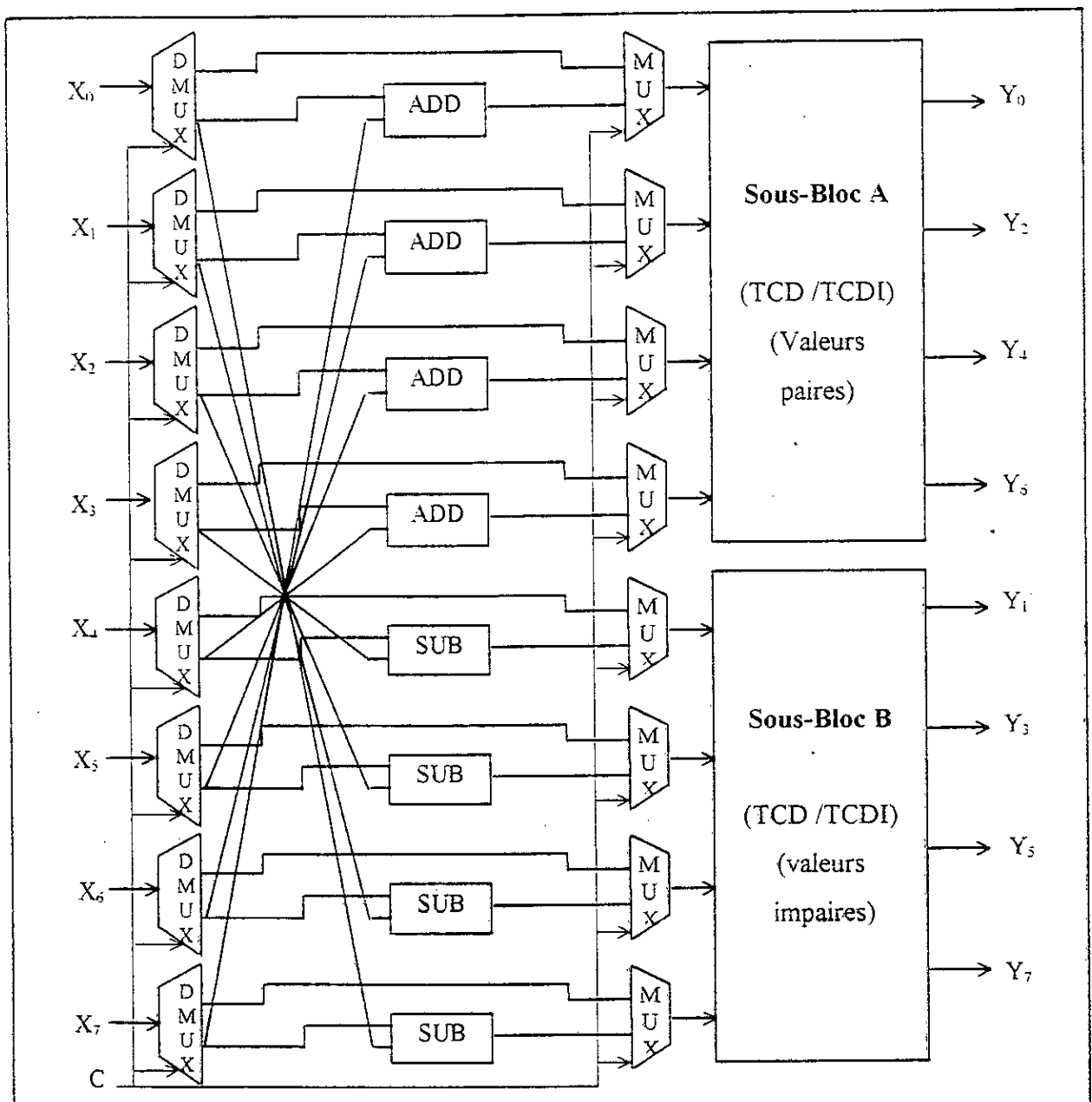


Figure III-8 - Structure du bloc de pré-traitement pour la TCD/TCDI en radical 4.

A présent, nous allons proposer une architecture pour les additions et soustractions de ces blocs.

Il convient de rappeler que le mode de calcul Half-line a été introduit pour des calculs complexes comme dans notre cas, pour l'évaluation de l'expression généralisée de la T.C.D. Nous pensons que pour les opérations de base d'addition et de soustraction en mode Half-line, il n'est pas intéressant de concevoir une architecture sérielle spécifique d'après leur algorithme mais, il nous suffit d'utiliser l'additionneur/soustracteur séquentiel en complément à deux (ripple-carry adder) de la figure suivante avec une commande externe pour le choix de l'opération à réaliser.

La représentation en C2 des opérands sied d'ailleurs bien à notre application où les pixels prennent leurs valeurs dans l'intervalle ]-128, 128].

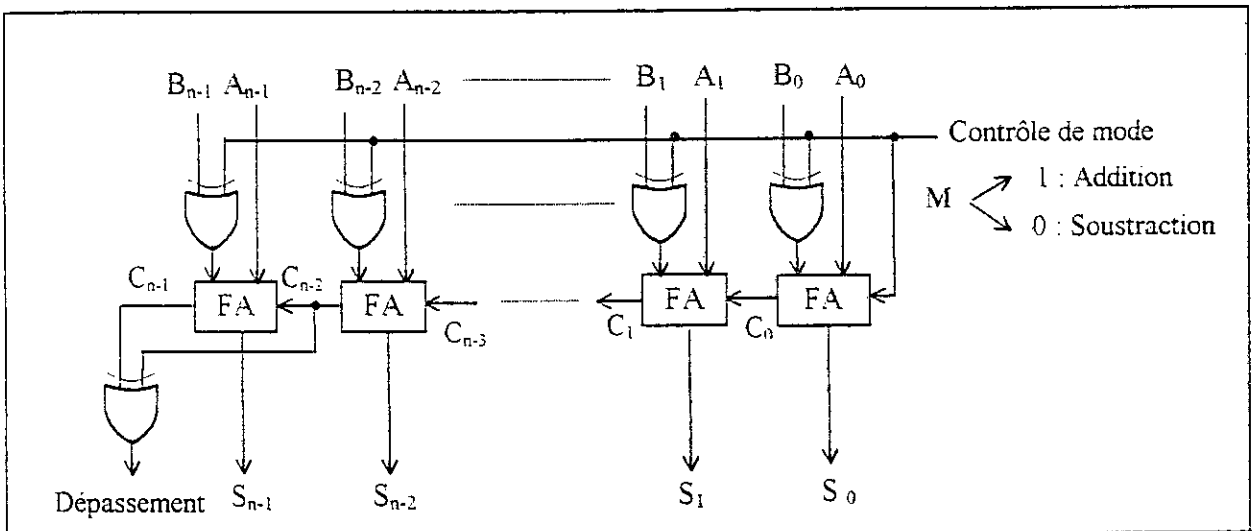


Figure III.9 - Additionneur/Soustracteur en C2 utilisé dans les blocs de pré-traitement et post-traitement du calcul TCD/TCDI en radical 4.

L'architecture complète du bloc de normalisation en radical 4, est donnée par la figure III.10.

**5.6.2 - LE BLOC DE SELECTION**

Ce Bloc opère la sélection et la génération des bits résultats. Cette procédure est la partie la plus critique dans les algorithmes On-line et Half-line pour sa récurrence totale. Le temps de calcul dans ce bloc doit être optimisé pour le choix de la fréquence de travail et l'introduction d'un pipe-line dans les traitements.

Les équations que nous devons implémenter sont :

$$\begin{cases} H [i] = 2 \cdot H [i - 1] + L [i] - 2 \cdot y_{i-1} \\ y_i = S ( H [i] ) : \text{procédure de sélection.} \end{cases} \tag{III-40}$$

Réécrivons l'expression de H comme suit :

$$\begin{cases} H [i] = \overline{H} [i] - 2 y_{i-1} \\ \overline{H} [i] = 2 H [i - 1] + L [i] \end{cases} \tag{III-41}$$

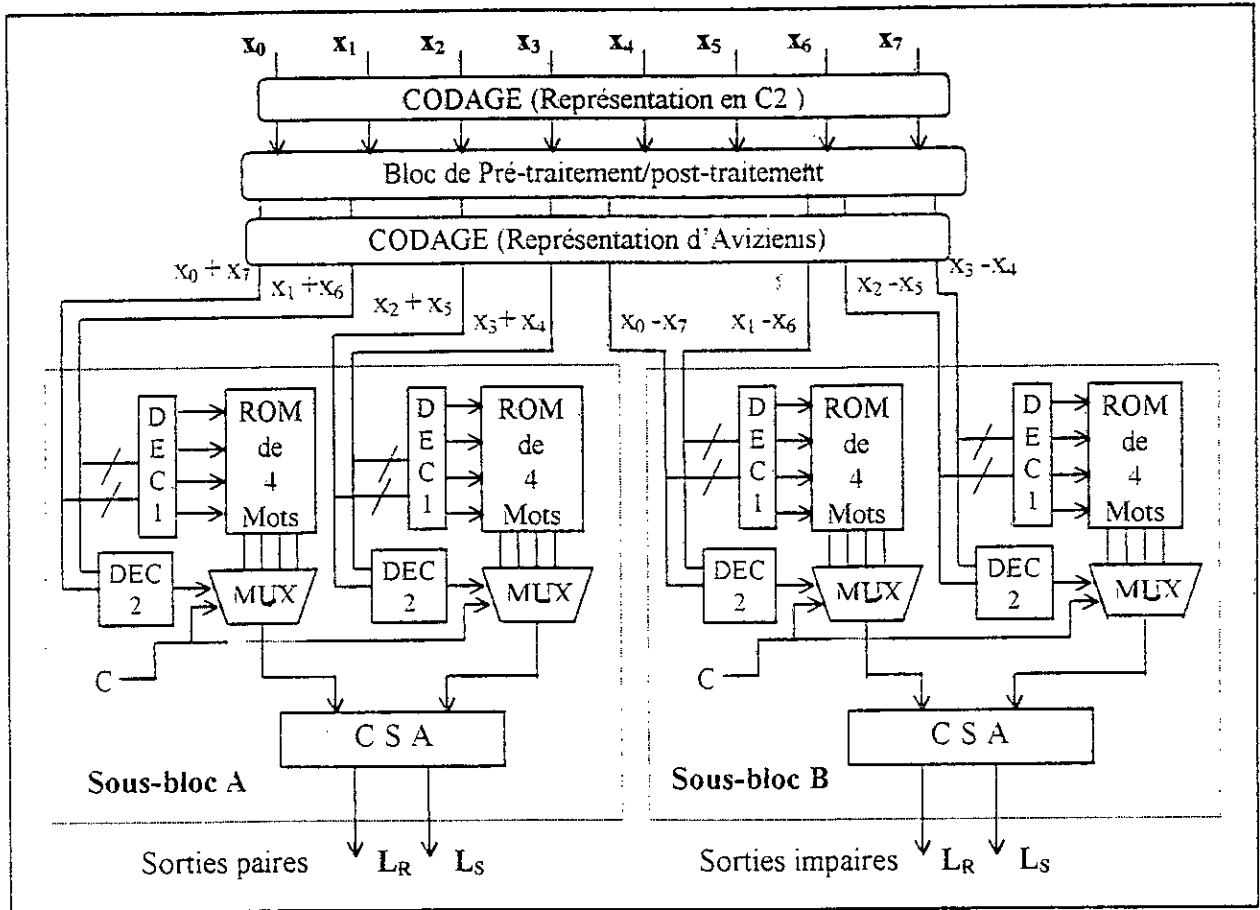


Figure III.10 - Architecture du bloc de Normalisation TCD/TCDI en radical 4.

Nous savons que :

- $L[i]$  est calculé à travers des CSAs donc :  $L[i] = L_S[i] + 2 L_R[i]$ ;
  - $H[i-1]$  l'est aussi à cause de la récursivité des calculs donc :  $H[i-1] = H_S[i-1] + 2 H_R[i-1]$
- les indices « S » et « R » sont respectivement relatifs à l'opérande somme et l'opérande retenue.

Par suite : 
$$\bar{H}[i] = 2 H_S[i-1] + 4 H_R[i-1] + L_S[i] + 2 L_R[i] \tag{III-42}$$

$\bar{H}[i]$  sera composé de deux opérandes:  $\bar{H}_S[i]$  pour la somme et  $\bar{H}_R[i]$  pour la retenue.

Cette expression peut être implémentée en utilisant un additionneur parallèle CSA 4-2 :

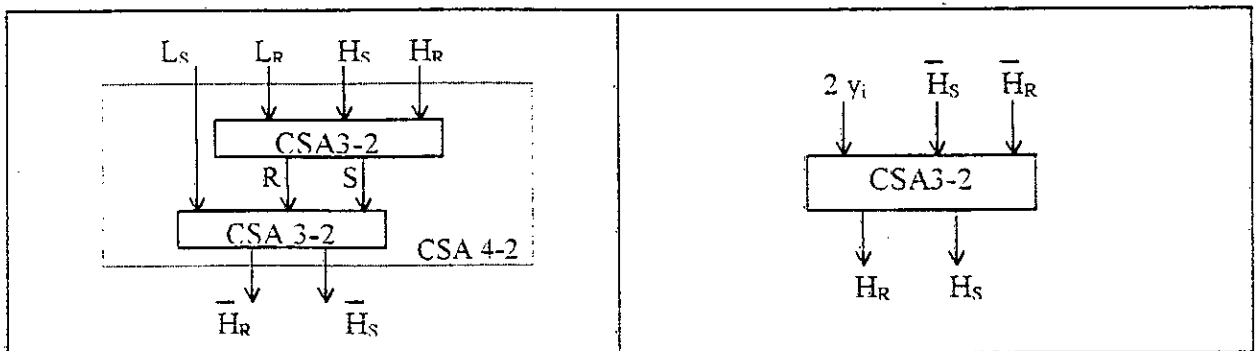


Figure III.11 - Implantation de l'expression : 
$$\bar{H}[i] = 2 H_S[i-1] + 4 H_R[i-1] + L_S[i] + 2 L_R[i]$$

Figure III.12- Implantation de l'expression : 
$$H[i] = \bar{H}_S[i] + 2 \cdot \bar{H}_R[i] - 2 y_{i-1}$$

Le résidu complet H est donné par la seconde équation, soit :

$$H[i] = \overline{H}[i] - 2 y_{i-1} = \overline{H}_S[i] + 2 \cdot \overline{H}_R[i] - 2 y_{i-1} \quad (III-46)$$

Cette équation est réalisée en utilisant l'additionneur parallèle CSA 3-2 de la figure III-12.

Pour la procédure de sélection S(H[i]), il suffit de propager la retenue sur  $\psi$  bits de H [53], (pour une base  $b=2$ ,  $\psi=1$  quelque soit le radical). De plus,  $-3/2 \leq H \leq 3/2$ , le résidu H sera alors considéré sur 3 bits : le bit signe  $h_s$  et les bits de poids « 0 » et « 1 » :  $h_0$  et  $h_1$ .

Donc au lieu de faire l'addition sur toute l'opérande H, on ne l'effectue que sur 3 bits tel que :

$$H^* = \overline{\overline{H}}_S + 2 \cdot \overline{\overline{H}}_R - 2 y_{i-1} \quad \text{avec : } H^* = (H_S^*, H_R^*) \quad (III-47)$$

$\overline{\overline{H}}_S$  et  $\overline{\overline{H}}_R$  sont les représentations de  $\overline{H}_S$  et  $\overline{H}_R$  sur 3 bits.

Ainsi, la retenue sera propagée sur  $H_S^*$  et  $H_R^*$ . Et pour accélérer le calcul de H, on utilisera un additionneur à retenue anticipée CLA (Carry Look-ahead Adder).

### 5.6.2.1 - Additionneur à Retenue Anticipée CLA [44]

Plutôt que de propager les retenues, ces additionneurs grâce à des circuits logiques supplémentaires, cherchent à calculer à l'avance leurs valeurs. Un rappel sur ces additionneurs est donné en annexe A.

Dans notre cas, il s'agit d'additionner les 2 nombres  $H_S^*$  et  $H_R^*$  :  $H = H_S^* + H_R^*$  (III-48)

Cet additionneur sera symbolisé comme suit :

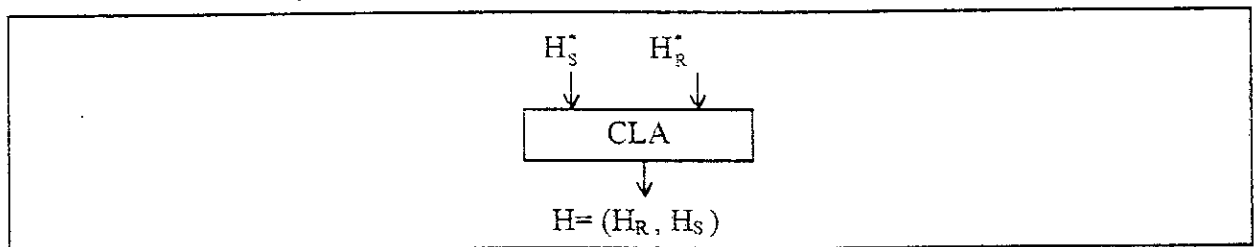


Figure III.13 - Symbole d'un additionneur CLA.

### 5.6.2.2 - Circuit de sélection du bit-résultat

La sélection du bit résultat est assurée sur les 3 bits MSB de H par la seconde partie de l'algorithme de la procédure de sélection, à savoir :  $y = S(H[i])$ , car on a :

$$\frac{-3}{2} \leq H \leq \frac{3}{2} \quad \text{et} \quad H = -2 \cdot h_s + h_0 \cdot 2^0 + h_1 \cdot 2^{-1} \quad (III-49)$$

Alors, il suffit de vérifier seulement  $h_s, h_0, h_1$  pour la sélection du bit résultat  $y_i$  qui sera écrit en notation signe-valeur :  $y_i = -2 y_{i s} + y_{i 0}$ ,  $y_j$  peut prendre une et une seule valeur appartenant à l'ensemble  $\{-1, 0, 1\}$ .

La table de vérité suivante donne les équations logiques des bits  $y_s$  et  $y_0$  :

Valeur de H	$h_s$	$h_0$	$h_1$	Valeur de $y_i$	$y_{is}$	$y_{i0}$
0	0	0	0	0	0	0
0.5	0	0	1	1	0	1
1	0	1	0	1	0	1
1.5	0	1	1	1	0	1
$\phi$	1	0	0	$\phi$	$\phi$	$\phi$
-1.5	1	0	1	-1	1	1
-1	1	1	0	-1	1	1
-0.5	1	1	1	0	0	0

Tableau III-3 : Table de vérité du circuit combinatoire génération de  $y_i$ .

Les équations logiques de  $y_s$  et  $y_0$  sont données par :

$$\begin{cases} y_s = h_0 \overline{h_s} h_1 \\ y_0 = h_0 \oplus h_s + h_0 \oplus h_1 \end{cases}$$

Ces équations sont réalisées par le circuit combinatoire suivant :

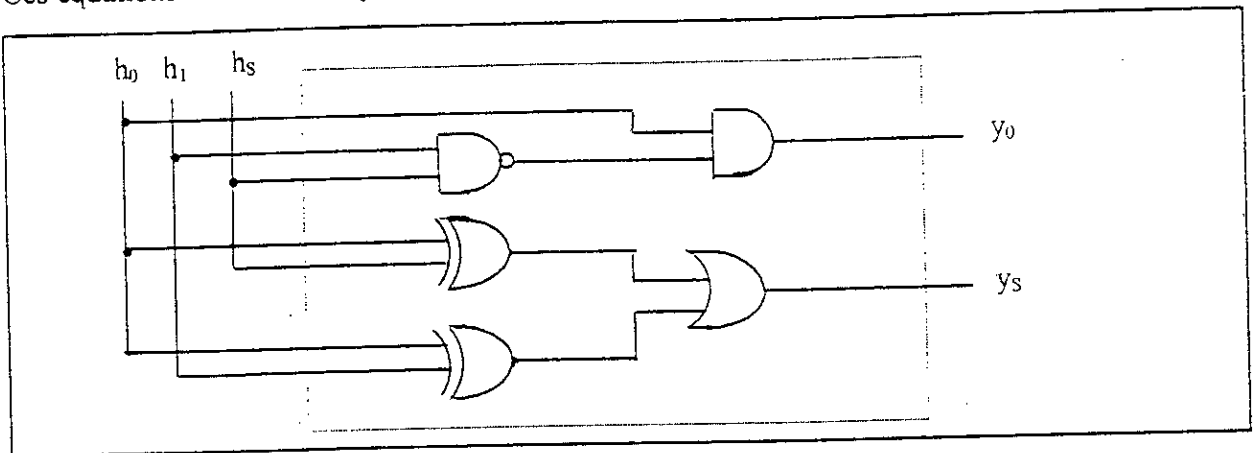


Figure III.14 - Le circuit de sélection du résultat .

Enfin , l'architecture complète du bloc de sélection et génération est la suivante :

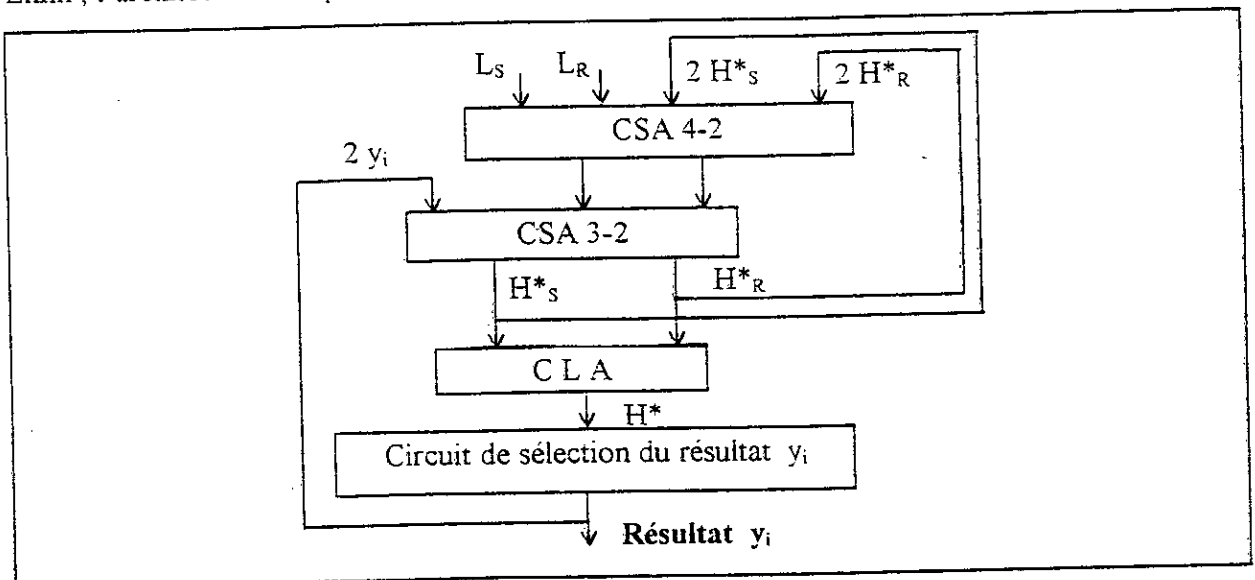


Figure III-15 - Architecture du bloc de sélection et génération.

**5.6.3 - Architecture pipe-linée du bloc de sélection**

Dans l'architecture proposée dans la section précédente, le chemin critique est celui d'un additionneur CSA 3-2, d'un additionneur CLA et du circuit de sélection du résultat  $y$ . Or, ce temps critique doit être le plus petit possible, nous proposons de le réduire en organisant toute l'architecture en deux parties, comme illustré par la figure suivante :

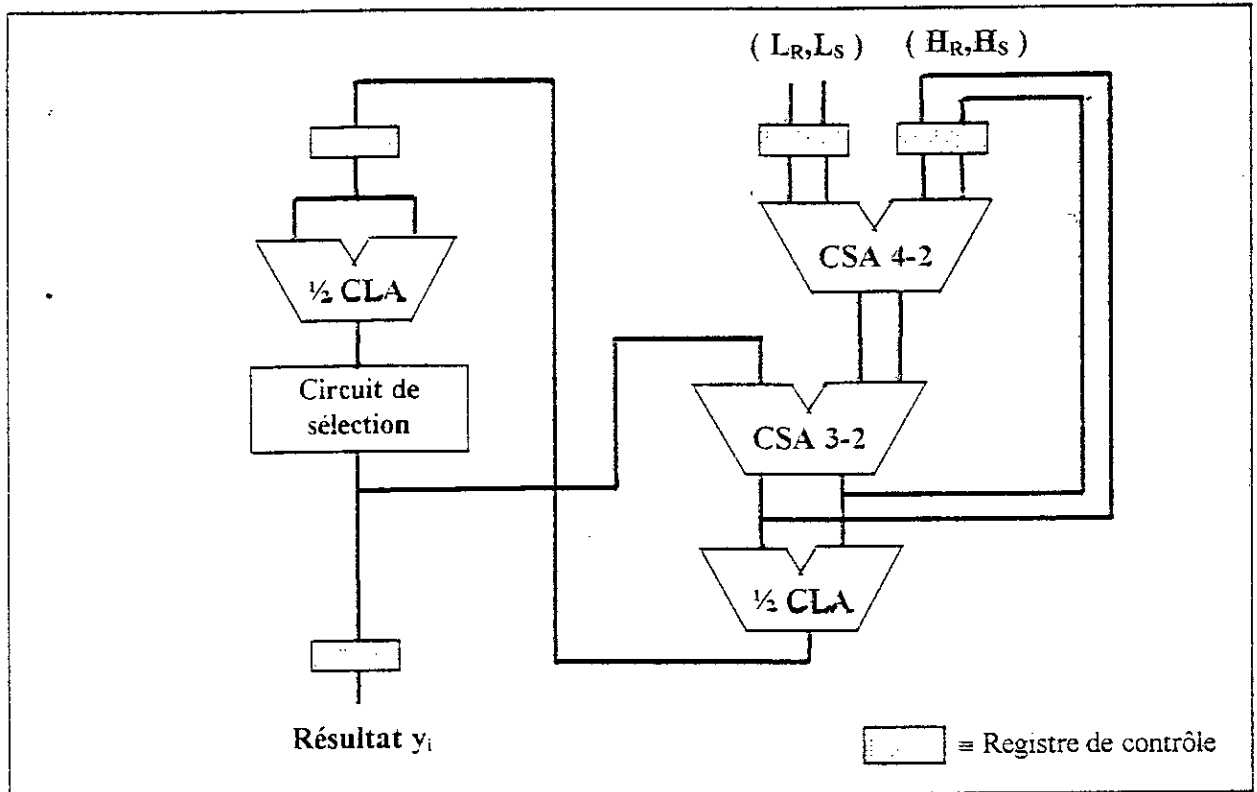


Figure III.16 - Architecture pipe-linée du bloc de sélection.

**5.6.4 - DISCUSSION**

Le calcul en mode Half-line est très efficace pour des calculs complexes car il permet de générer des structures très régulières, adaptées à l'implémentation VLSI. Pour notre application, il nous a permis d'arriver à des architectures intéressantes. L'avantage principal de ce mode, est le fait que la fréquence de travail est constante car indépendante de la taille des opérandes d'entrée. Néanmoins, il présente à notre sens un inconvénient majeur, qui est la représentation des opérandes dans un système redondant. Cette contrainte rend son utilisation inadéquate dans un ensemble de processeurs classiques.

Dans la section suivante, nous allons utiliser un autre mode de calcul et tenter de dériver un algorithme aussi efficace qu'en mode half-line pour l'évaluation de l'expression généralisée de notre transformé afin de concevoir de meilleures architectures pour notre application.

## 6. ARITHMETIQUE DISTRIBUEE (D.A) : [56]

L'arithmétique distribuée est ainsi nommée parce que les opérations arithmétiques de traitement de signal (par exemple l'addition, la multiplication) ne sont pas traitées et organisées de façon confortable mais plutôt distribuées de façon non identifiée sur plusieurs modules.

La première description largement connue de cette arithmétique a été présentée en 1974 par A. Peled et Bede Liu [57]. Un travail antérieur a été présenté en France par A. Croisier et al. [58], mais l'invention de cette arithmétique a été faite par S.Zohar vers 1968 [59].

La motivation principale dans l'utilisation de l'arithmétique distribuée est l'extrême efficacité des calculs. Ses avantages sont exploités dans la conception de circuits, et la forme du hardware est souvent configurée de façon à faciliter son exécution. Avec une conception rigoureuse, l'on peut réduire le nombre total de portes d'une unité arithmétique de traitement de signal de 50 à 80%.

### 6.1 - DERIVATION D'UN CALCUL « ARITHMETIQUE DISTRIBUE »

L'arithmétique distribuée est fondamentalement un mode de calcul bit-sériel, et c'est cette nature qui fait que son désavantage majeur réside dans sa lenteur apparente. Cependant, cette contrainte n'est pas réelle si l'on introduit les techniques de partitionnement des entrées ou de parallélisme.

La forme de calcul la plus répandue en traitement numérique de signal est une somme de produits, c'est aussi le calcul exécuté le plus efficacement par cette arithmétique.

Pour un calcul distribué de la T.C.D, soit l'expression généralisée (III-7):  $Y = \sum_{k=1}^N X_k A_k$ .

Les noyaux cosinus  $A_k$  ( $k=1,N$ ) sont connus d'avance et la séquence  $X_k$  ( $k=1,N$ ) regroupe les  $N$  pixels du bloc d'image à traiter, connus bit par bit et représentés en complément à deux (C2) :

$$X_k = (-x_{k0} + \sum_{j=1}^{M-1} x_{kj} 2^{-j}) \quad (\text{III-50})$$

où les bits  $x_{kj} \in \{0,1\}$  et le bit  $x_{k0}$  représente le bit signe.

Par suite :

$$Y = \sum_{j=1}^{M-1} \left[ \sum_{k=1}^N A_k \cdot x_{kj} \right] 2^{-j} + \sum_{k=1}^N A_k (-x_{k0}) \quad (\text{III-51})$$

Cette équation définit un calcul arithmétique distribué.

### 6.2 - IMPLEMENTATION

Soit le terme entre parenthèses :  $(\sum_{k=1}^N A_k x_{kj}, j = 1 \text{ à } M-1)$ .

Puisque chaque  $x_{kj} \in \{0,1\}$ , le produit  $(A_k \cdot x_{kj})$  peut avoir seulement  $2^N$  valeurs possibles. Au lieu de les calculer, l'on peut les déterminer avant le lancement de l'opération et les stocker dans

une ROM. Les données entrées peuvent être utilisées pour adresser directement la mémoire, et le

résultat  $\sum_{k=1}^N A_k x_{kj}$  peut être obtenu dans un accumulateur en  $N$  cycles.

Cette mémoire doit contenir toutes les valeurs positives possibles et leurs opposées afin d'y charger le second terme ( $\sum_{k=1}^N A_k (-x_{k0})$ ). Notre équation nécessite donc une ROM de  $2 \times 2^N$  mots.

### 6.2.1 - Réduction de la taille de la ROM :

L'idée d'une ROM réduite, vient du fait que les deux termes de l'expression (III-51), sont des quantités identiques qui ont la même séquence de décodage, mais de signes opposés. Pour éviter de stocker les valeurs négatives, un signal bit-signe  $T_s$  est utilisé pour contrôler l'arrivée bit à bit des données, LSB en tête. Les bits de signe ( $x_{k0}$ ) arrivent simultanément en dernier, en un cycle appelé : « cycle des bits-signe ». Durant ce cycle,  $T_s=1$  sinon  $T_s=0$ .

En modifiant l'additionneur en additionneur/soustracteur (ADD/SUB) et en utilisant  $T_s$  comme un signal de contrôle ADD/SUB, nous pouvons réduire la taille de la ROM de moitié. Par exemple pour  $N=4$ , le tableau III-4 donne le contenu de la ROM de 32 mots. La figure III-17.a montre une conception simple pour notre produit, avec un switch qui reste en position « 1 » sauf pendant le cycle des bits-signe où il bascule sur la position « 2 » afin de former le résultat final. La symétrie contenue dans ce tableau permet de réduire la ROM de moitié comme montré en figure III-17.b où  $T_s$  est utilisé comme signal de contrôle (ADD/SUB) autour d'une ROM de 16 mots.

La taille de la ROM peut encore être réduite de moitié. Pour cela, nous supposons que  $X_k$  n'est plus codé par une suite de bits  $x_k \in \{0, 1\}$  mais par une suite de bits  $x_k \in \{-1, 1\}$ .

Réécrivons  $X_k$  sous la forme suivante :  $X_k = 1/2 [X_k - (-X_k)]$  (III-52)

et rappelons qu'en notation en complément à deux (C2), le nombre négatif ( $-X_k$ ) s'écrit :

$$-X_k = -\bar{x}_{k0} + \sum_{j=1}^{M-1} \bar{x}_{kj} \cdot 2^{-j} + 2^{-(M-1)} \quad (III-53)$$

Par suite :  $X_k = 1/2 [-(x_{k0} - \bar{x}_{k0}) + \sum_{j=1}^{M-1} (x_{kj} - \bar{x}_{kj}) 2^{-j} 2^{-(M-1)}]$  (III-54)

Dans le but de simplifier notre notation, il convient de définir de nouvelles variables :

$$\begin{cases} c_{k0} = -(x_{k0} - \bar{x}_{k0}) \\ c_{kj} = x_{kj} - \bar{x}_{kj} ; j \neq 0 \end{cases} \quad \text{où } c_{kj} \in \{-1, 1\} \quad (III-55)$$

d'où :  $X_k = 1/2 \left[ \sum_{j=0}^{M-1} c_{kj} 2^{-j} - 2^{-(M-1)} \right]$  (III-56)



Code d'entrée					Contenu de la ROM de 32 Mots
T <sub>s</sub>	x <sub>1k</sub>	x <sub>2k</sub>	x <sub>3k</sub>	x <sub>4k</sub>	
0	0	0	0	0	0
0	0	0	0	1	A <sub>4</sub>
0	0	0	1	0	A <sub>3</sub>
0	0	0	1	1	A <sub>3</sub> +A <sub>4</sub>
0	0	1	0	0	A <sub>2</sub>
0	0	1	0	1	A <sub>2</sub> +A <sub>4</sub>
0	0	1	1	0	A <sub>2</sub> +A <sub>3</sub>
0	0	1	1	1	A <sub>2</sub> +A <sub>3</sub> +A <sub>4</sub>
0	1	0	0	0	A <sub>1</sub>
0	1	0	0	1	A <sub>1</sub> +A <sub>4</sub>
0	1	0	1	0	A <sub>1</sub> +A <sub>3</sub>
0	1	0	1	1	A <sub>1</sub> +A <sub>3</sub> +A <sub>4</sub>
0	1	1	0	0	A <sub>1</sub> +A <sub>2</sub>
0	1	1	0	1	A <sub>1</sub> +A <sub>2</sub> +A <sub>4</sub>
0	1	1	1	0	A <sub>1</sub> +A <sub>2</sub> +A <sub>3</sub>
0	1	1	1	1	A <sub>1</sub> +A <sub>2</sub> +A <sub>3</sub> +A <sub>4</sub>
1	0	0	0	0	0
1	0	0	0	1	- A <sub>4</sub>
1	0	0	1	0	- A <sub>3</sub>
1	0	0	1	1	- (A <sub>3</sub> +A <sub>4</sub> )
1	0	1	0	0	- A <sub>2</sub>
1	0	1	0	1	- (A <sub>2</sub> +A <sub>4</sub> )
1	0	1	1	0	- (A <sub>2</sub> +A <sub>3</sub> )
1	0	1	1	1	- (A <sub>2</sub> +A <sub>3</sub> +A <sub>4</sub> )
1	1	0	0	0	- A <sub>1</sub>
1	1	0	0	1	- (A <sub>1</sub> +A <sub>4</sub> )
1	1	0	1	0	- (A <sub>1</sub> +A <sub>3</sub> )
1	1	0	1	1	- (A <sub>1</sub> +A <sub>3</sub> +A <sub>4</sub> )
1	1	1	0	0	- (A <sub>1</sub> +A <sub>2</sub> )
1	1	1	0	1	- (A <sub>1</sub> +A <sub>2</sub> +A <sub>4</sub> )
1	1	1	1	0	- (A <sub>1</sub> +A <sub>2</sub> +A <sub>3</sub> )
1	1	1	1	1	- (A <sub>1</sub> +A <sub>2</sub> +A <sub>3</sub> +A <sub>4</sub> )

Tableau III-4 - Le contenu de la ROM de 32 mots.

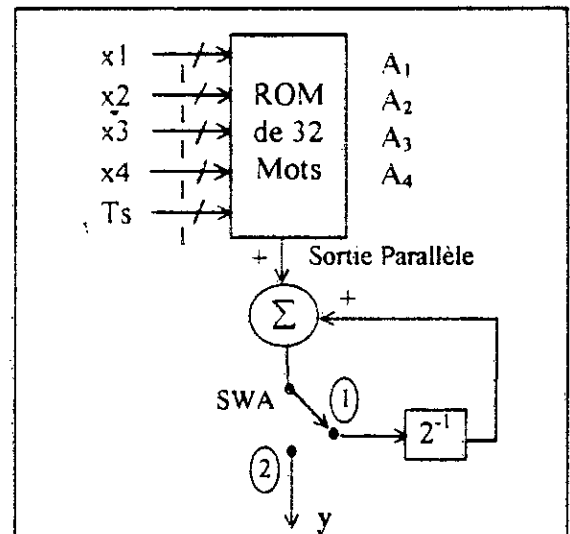


Fig. III.17 a - Additionneur et mémoire complète (32 mots).

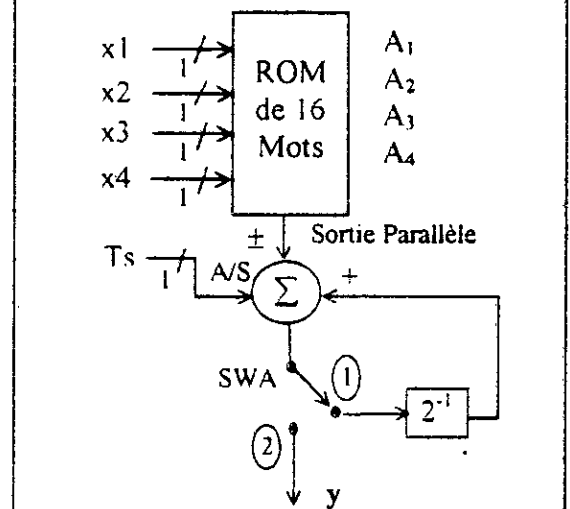


Fig. III.17.b - Additionneur /Soustracteur et mémoire réduite (16 mots).

Figure III.17 - Implémentation bit-sérielle

par la D.A de  $Y = \sum_{k=1}^N X_k A_k$  pour  $N=4$ .

le résultat Y devient :

$$Y = \frac{1}{2} \sum_{k=1}^N A_k \left( \sum_{j=0}^{M-1} c_{kj} 2^{-j} - 2^{-(M-1)} \right) = \sum_{j=0}^{M-1} Q(x_j) 2^{-j} + 2^{-(M-1)} Q(0) \tag{III-57}$$

avec :  $Q(x_j) = \sum_{k=1}^N \frac{A_k}{2} c_{kj}$  et  $Q(0) = \sum_{k=1}^N \frac{A_k}{2}$  (III-58)

La quantité  $Q(x_j)$  a seulement  $2^{N-1}$  valeurs possibles avec un signe donné par la combinaison instantanée des bits. Le calcul de  $Y$  est obtenu en utilisant une ROM de  $2^{N-1}$  mots, un registre de condition initiale pour  $Q(0)$ , et un additionneur/Soustracteur parallèle avec des portes logiques de contrôle. Ceci est montré en figure III-18 pour l'exemple précédent ( $N=4$ ) avec une ROM de 8 mots contenant les valeurs de  $Q(x_j)$ . Le tableau suivant donne les  $2^{N-1}$  valeurs  $Q(x_j)$  possibles.

Code d'entrée				Contenu de la ROM de 8 Mots, Q
$x_{1k}$	$x_{2k}$	$x_{3k}$	$x_{4k}$	
0	0	0	0	$-1/2 (A_1 + A_2 + A_3 + A_4)$
0	0	0	1	$-1/2 (A_1 + A_2 + A_3 - A_4)$
0	0	1	0	$-1/2 (A_1 + A_2 - A_3 + A_4)$
0	0	1	1	$-1/2 (A_1 + A_2 - A_3 - A_4)$
0	1	0	0	$-1/2 (A_1 - A_2 + A_3 + A_4)$
0	1	0	1	$-1/2 (A_1 - A_2 + A_3 - A_4)$
0	1	1	0	$-1/2 (A_1 - A_2 - A_3 + A_4)$
0	1	1	1	$-1/2 (A_1 - A_2 - A_3 - A_4)$
1	0	0	0	$1/2 (A_1 - A_2 - A_3 - A_4)$
1	0	0	1	$1/2 (A_1 - A_2 - A_3 + A_4)$
1	0	1	0	$1/2 (A_1 - A_2 + A_3 - A_4)$
1	0	1	1	$1/2 (A_1 - A_2 + A_3 + A_4)$
1	1	0	0	$1/2 (A_1 + A_2 - A_3 - A_4)$
1	1	0	1	$1/2 (A_1 + A_2 - A_3 + A_4)$
1	1	1	0	$1/2 (A_1 + A_2 + A_3 - A_4)$
1	1	1	1	$1/2 (A_1 + A_2 + A_3 + A_4)$

Tableau III-5 - Contenu Q de la ROM réduite.

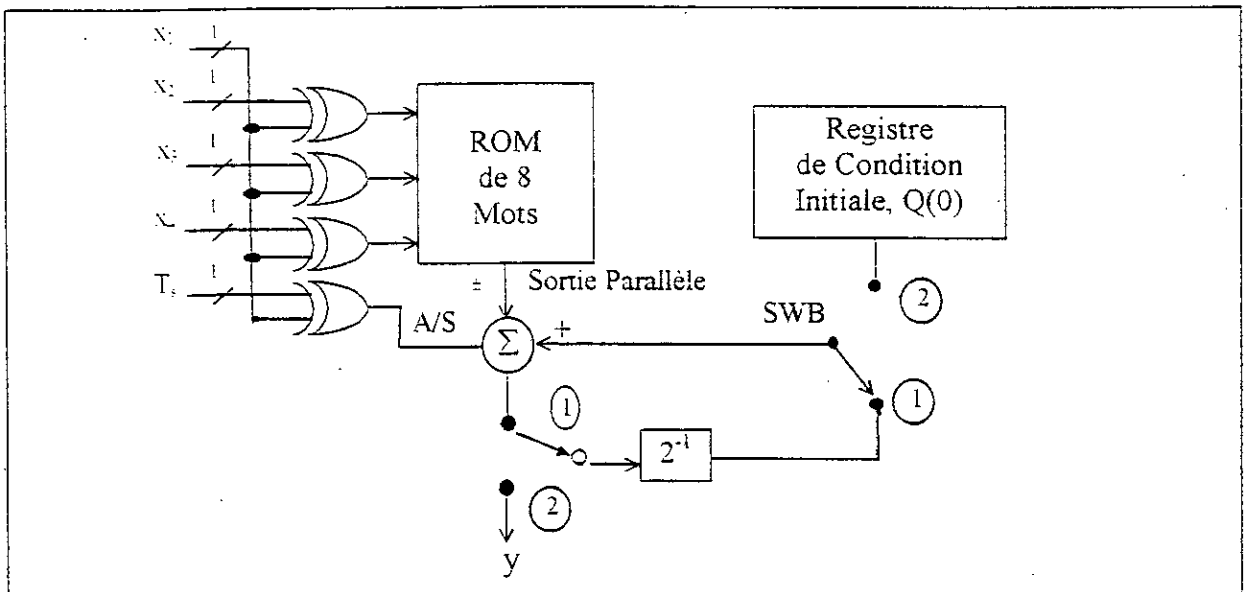


Fig.III-18 - Nouvelle implémentation par la D.A de  $Y = \sum_{k=1}^N X_k A_k$  ( $N=4$ ) avec une ROM de 8 mots.

La ROM utilisée contient les valeurs  $Q(x_{kj})$  de la moitié supérieure du tableau III-5 car l'autre moitié n'est que son image-miroir mais de signe opposé. Cette symétrie fait que l'adressage de la ROM se fait par  $x_{1j}$  combiné à travers des portes «EXOR» avec  $x_{2j}$ ,  $x_{3j}$  et  $x_{4j}$ . Pour avoir le signe des valeurs  $Q(x_{kj})$  et contrôler l'ADD/SUB,  $x_{1j}$  et  $Ts$  sont combinés à travers une porte «EXOR».

La valeur  $Q(0)$  est mémorisée dans un registre de condition initiale. Quand la ROM est adressée par les bits LSBs, la valeur fournie doit être corrigée de  $Q(0)$  à travers un switch SWB, qui opère de façon synchrone avec le switch SWA. Ces deux switches sont en position 2 seulement pendant le cycle des bits-signes quand  $Ts = 1$ . Pendant le premier cycle, la première donnée  $Q(x_{M-1})$  sortie de la ROM, est sommée avec  $Q(0)$ ; le résultat est décalé à droite durant le second cycle et sommé avec  $Q(x_{M-2})$  pour produire  $\{Q(x_{M-2}) + [Q(x_{M-1}) + Q(0)]. 2^{-1}\}$ ; durant le cycle suivant le résultat produit est  $\{Q(x_{M-3}) + Q(x_{M-2}). 2^{-1} + [Q(x_{M-1})+Q(0)]. 2^{-2}\}$ , et ainsi de suite jusqu'au dernier cycle où l'on somme  $Q(x_0)$  au résultat précédent décalé à droite, pour produire enfin la quantité :

$$Q(x_0)+Q(x_1).2^{-1} + Q(x_2)2^{-2} + \dots + Q(x_{M-2}).2^{-(M-2)} + [Q(x_{M-1})+Q(0)].2^{-(M-1)} \quad (III-59)$$

**6.2.2 - Amélioration de la vitesse de traitement**

Il est clair que malgré la réduction de la taille mémoire, l'injection sérielle des données, bit par bit, conduit à une lenteur dans les calculs. La vitesse peut être augmentée d'un facteur L si l'on partitionnait chaque entrée (mot de M bits) en L sous-mots (L doit être un diviseur de M). Nous pourrions utiliser L mémoires avec un accumulateur multi-opérandes. Cette approche est illustrée par la figure III-19.

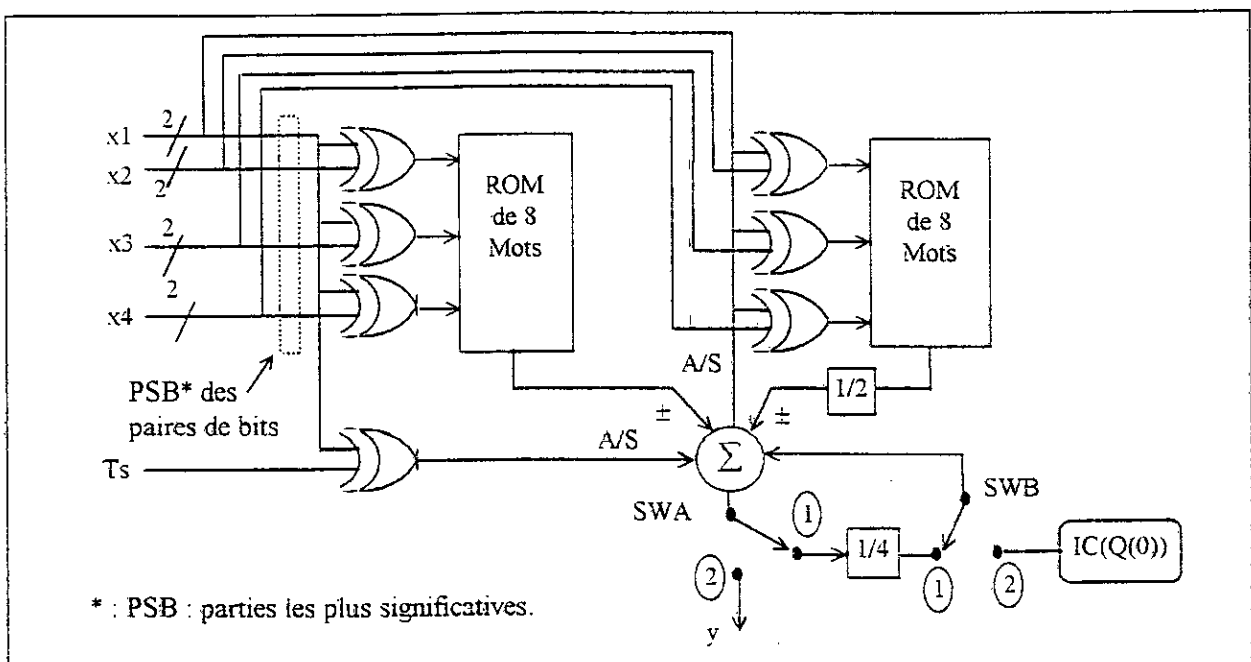


Fig. III-19 - Nouvelle implémentation avec 2 ROMs et partitionnement en paires de bits.

Les ROMs contiennent les valeurs  $Q(x_{kj})$  de la moitié supérieure du tableau III-5. Leur adressage se fait par la combinaison à travers des portes «EXOR», par paires de bits de  $x_1$  avec  $x_2$ ,  $x_3$  et  $x_4$ . Pour contrôler l'ADD/SUB,  $x_1$  et  $T_s$  sont combinés à travers une porte «EXOR». La valeur  $Q(0)$  est mémorisée dans un registre de condition initiale, et les deux switches SWB et SWA opèrent de façon synchrone.

Il est à constater que malgré l'amélioration apportée en vitesse de calcul, l'architecture ci-dessus reprise de la référence [56] reste classique et ne permet pas un circuit dédié très performant.

### 6.2.3 - Limites de la conception classique

L'arithmétique distribuée ainsi présentée est dédiée à une implémentation pouvant utiliser les circuits classiques. En effet, l'implémentation de l'algorithme peut se faire en choisissant les composants des figures III-17, 18 et 19, parmi les circuits MSI. Ce mode d'implémentation n'est pas intéressant vu les problèmes inhérents aux capacités mémoires et à l'accumulation.

De plus, notre but étant une implémentation VLSI de la T.C.D et son inverse sur un ASIC, il devient donc nécessaire d'optimiser l'algorithme distribué de la T.C.D.

### 6.3 - OPTIMISATION DE L'ALGORITHME « DISTRIBUEE » DE LA T.C.D.

L'Arithmétique distribuée telle que utilisée dans la section précédente, ne nous donne pas la possibilité d'exploiter toutes ses performances. En effet, les relations déjà développées ne nous permettaient pas d'introduire un pipe-line ou un parallélisme dans la structure des calculs, d'autant plus qu'elles ne pouvaient résoudre les problèmes inhérents à la taille de la ROM et la propagation de la retenue dans les additionneurs.

Le challenge se réduit à proposer un nouveau modèle de cette arithmétique pour arriver à une facilité dans la conception et assurer de hauts débits dans les calculs. Nous proposons dans ce qui suit une optimisation de l'algorithme afin de tirer profit de l'arithmétique distribuée.

#### 6.3.1 - Principe de l'algorithme

Soit la séquence de pixels  $X_k$  ( $k=1, N$ ) représentés en C2, et connus bit par bit, LSB en tête.

D'après l'expression (III-56) :

$$X_k = \frac{1}{2} \left[ \sum_{j=0}^{M-1} c_{kj} 2^{-j} - 2^{-(M-1)} \right], \quad c_{kj} \in \{-1, 1\}$$

où le premier bit est du poids  $(M-1)$  correspondant à  $c_{k,M-1}$ .

Au pas  $j$ , alors que les  $j$  premiers bits de  $X_k$  [ $k$ ] sont introduits, un signal partiel noté  $X_k$  [ $j$ ] est formé tel que :

$$X_k[j] = \frac{1}{2} \left[ \sum_{j=M-1}^{M-j} c_{M-j} \cdot 2^{-(M-j)} - 2^{-(M-1)} \right], \quad (k = 1 \dots N) \quad \text{(III-60)}$$

L'équation de récurrence sur ce signal est donnée par :

$$X_k[j] = X_k[j-1] + \frac{c_{M-j}}{2} \cdot 2^{-M+j} \tag{III-61}$$

A chaque bit entrant, un accumulateur générera un produit partiel noté P[j] tel que :

$$P[j] = \sum_{k=1}^N A_k \cdot X_k[j] \cdot 2^{M-j} \tag{III-62}$$

La combinaison des deux dernières expressions nous donne :

$$P[j] = \frac{1}{2} P[j-1] + \sum_{k=1}^N A_k \cdot \frac{c_{M-j}}{2} \tag{III-63}$$

Nous définissons une accumulation partielle notée L[j], donnée par l'expression :

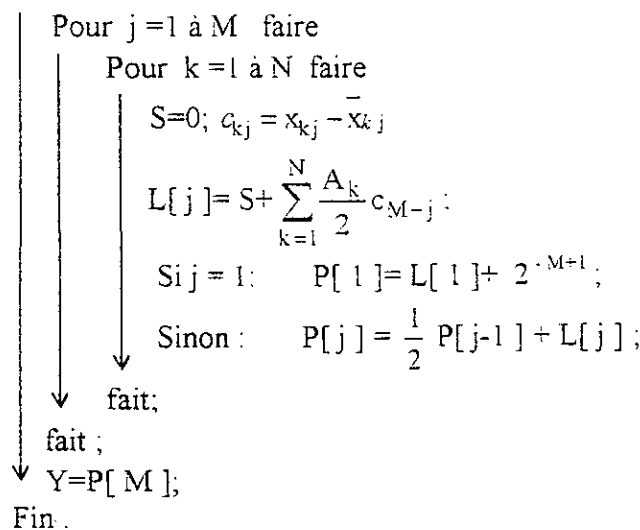
$$L[j] = \sum_{k=1}^N \frac{A_k}{2} \cdot c_{M-j} \tag{III-64}$$

l'équation de récurrence sur ce produit partiel est donnée par :

$$P[j] = \frac{1}{2} P[j-1] + L[j] \tag{III-65}$$

Alors, l'algorithme optimisé de l'arithmétique distribuée sera donné comme suit :

Début



### 6.3.2 - Implémentation de l'expression généralisée de la T.C.D

Le nouvel algorithme possède une structure générale composée de deux étages, comme illustré par la figure III-20 :

- le premier étage est une étape de « Normalisation » donnant les accumulations partielles L[j] utilisant des ROMs;
- et le second est une étape « d'Accumulation » des différentes valeurs normalisées avec une rétroaction des produits partiels.

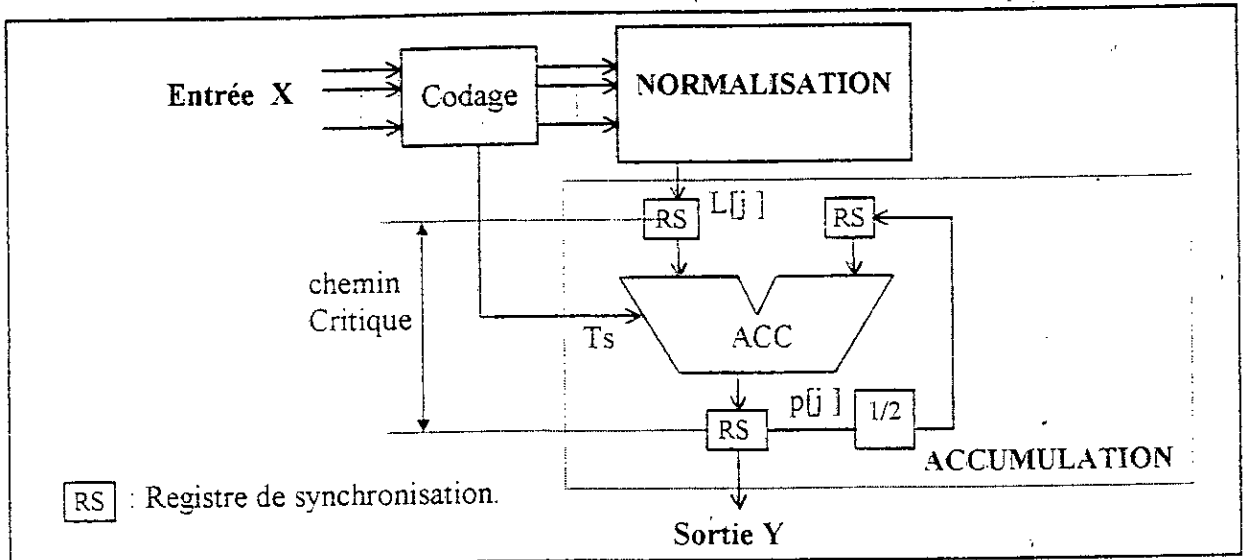


Fig. III-20 - Structure générale de l'algorithme optimisé de l'arithmétique distribué.

Pour arriver à une implémentation VLSI sur un ASIC dédié à nos transformées TCD/TCDI, nous proposons d'introduire deux techniques permettant la réduction de la complexité de la ROM dans l'étage de normalisation et l'utilisation des additions parallèles dans l'étage d'accumulation.

### 6.3.3 - Etage de normalisation

Dans cet étage, nous utilisons la technique de réduction de la taille mémoire basé sur le partitionnement de l'accumulation partielle  $L[j]$  en  $n$  groupes de  $r$  opérandes, tel que :

$$L[j] = \sum_{j=1}^n \left( \sum_{k=1}^r \frac{A_k}{2} c_{k, M-j} \right), \quad n = N/r. \tag{III-66}$$

Ce partitionnement conduit à la distribution de la ROM principale de  $2^{N-1}$  mots, en  $(N/n)$  ROMs de moindre dimension,  $2^{n-1}$  mots. La valeur finale de  $L[j]$  est obtenue en utilisant des additionneurs à retenue conservée CSA afin d'accélérer le calcul.

Le tableau suivant donne les différents cas :

	Nombre de ROMs	Taille des ROMs	Nombre d'additionneurs CSA
Radical 4	1	16 mots	0
	2	4 mots	1
	4	2 mots	3
Radical 8	1	256 mots	0
	2	16 mots	1
	4	4 mots	3
	8	2 mots	7

Tableau III-6 : Choix du nombre de ROMs suivant les CSAs

#### 6.3.3.1 - Architecture du bloc de normalisation de la TCD/TCDI en radical 8

L'architecture du bloc de normalisation de la TCD/TCDI en radical 8, est la suivante :

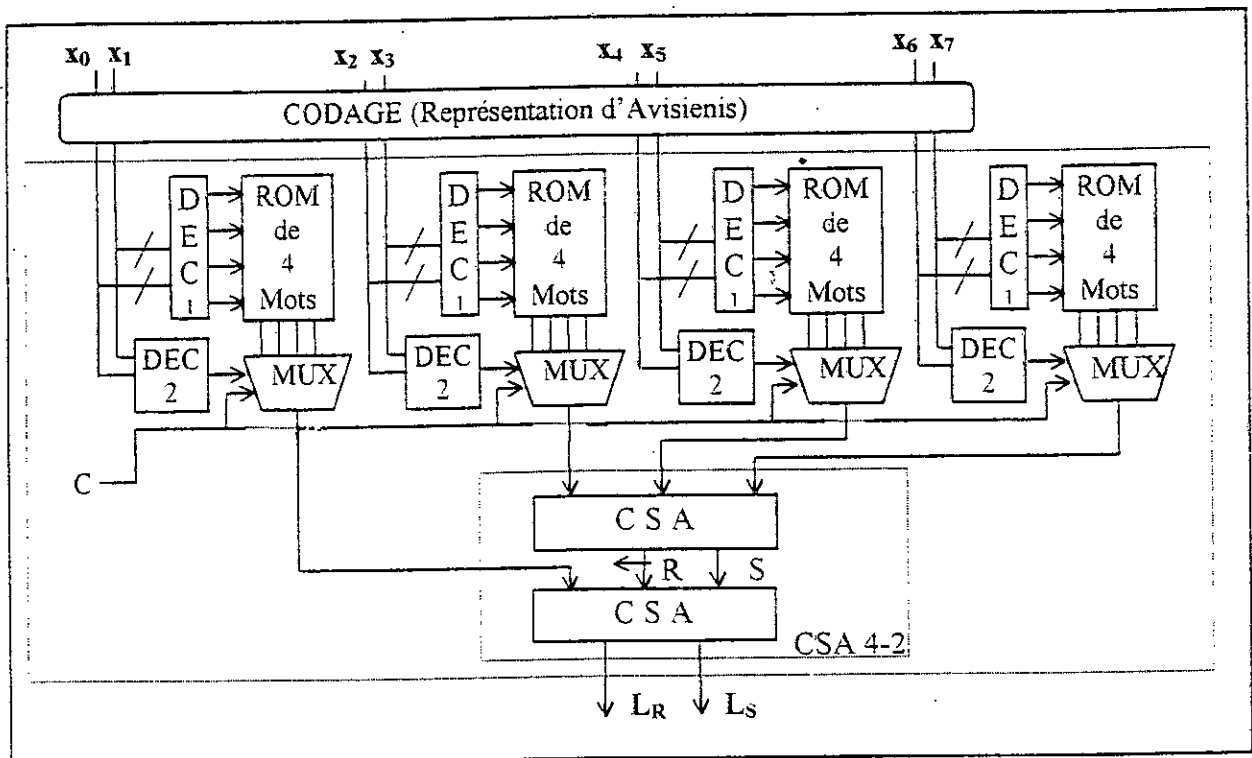


Figure III-21- Architecture du bloc de Normalisation TCD/TCDI en radical 8.

**6.3.3.2 - Architecture du bloc de normalisation de la TCD/TCDI en radical 4**

La structure de l'architecture hybride de la TCD/TCDI en radical 4, présente un bloc de pré-traitement et un bloc de post-traitement avant et après le bloc de calcul de la transformée. Ces deux blocs comme illustré par la figure III-8, introduisent des additions et soustractions. Avec l'arithmétique distribuée, la représentation des nombres est faite en complément à 2, il suffit donc d'utiliser l'additionneur/soustracteur séquentiel de la figure III.9 pour réaliser les opérations de ces blocs avec une commande externe pour le mode « addition » ou mode « soustraction ».

L'architecture complète du bloc de normalisation de la TCD/TCDI en radical 4, est montrée alors par la figure III.22.

**6. 3. 4 - Etage d'accumulation**

La propagation de la retenue pose problème dans l'étape d'accumulation où le traitement se fait avec rétroaction à l'inverse des additionneurs utilisés dans l'étape de normalisation qui effectuent des opérations de manière parallèle. Dans ce qui suit, nous proposons pour éviter la propagation de la retenue, l'utilisation des additionneurs à retenue conservée (CSA).

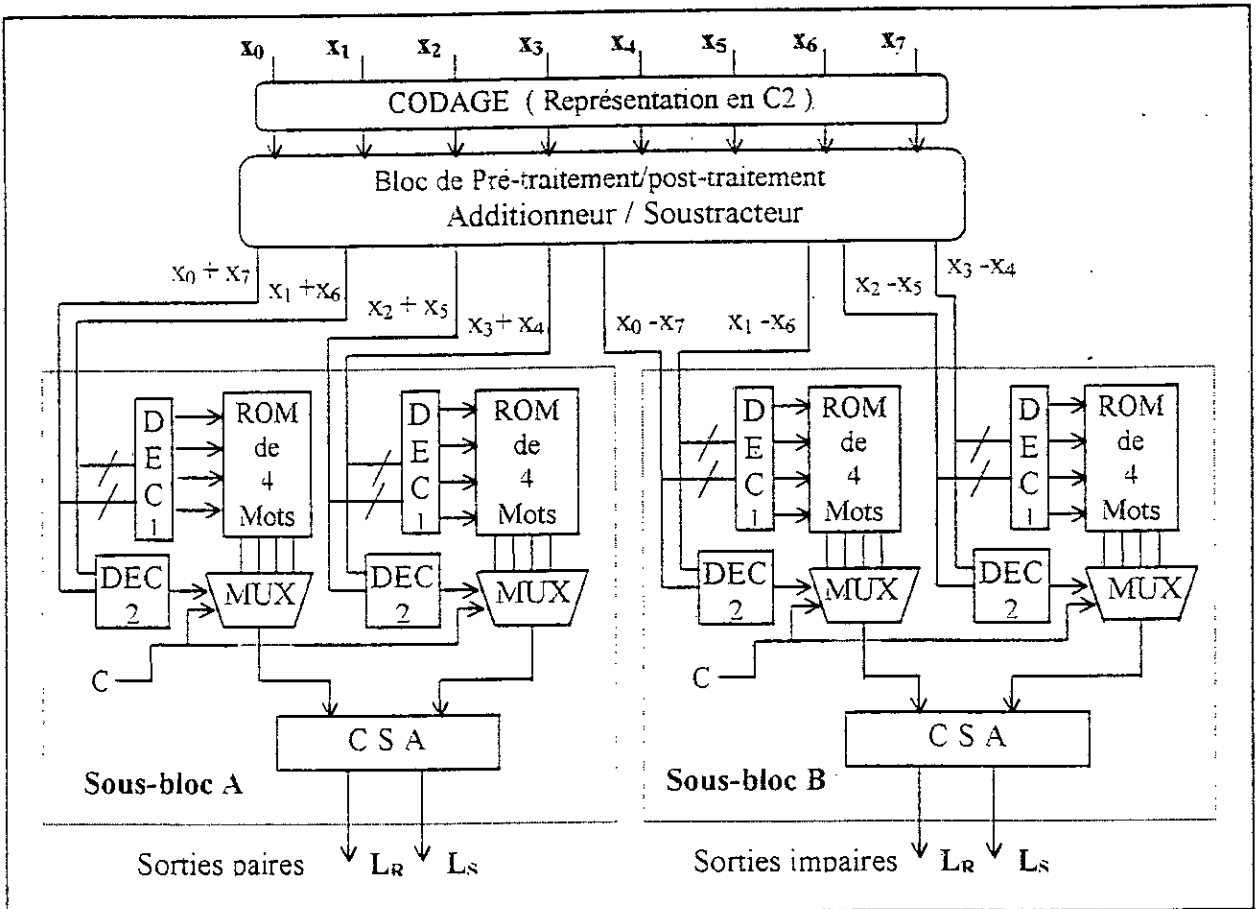


Figure III.22 - Architecture du bloc de Normalisation TCD/TCDI en radical 4.

**6. 3. 4.1 - Additions parallèles**

Du fait de l'utilisation des additionneurs, la réduction de la taille de la ROM a introduit, comme inconvénient, plus de surface Hard. Le tableau III.6 donne le nombre de ces additionneurs en fonction des paramètres N, n, r.

L'accumulateur effectue le calcul du produit partiel P[j]. Les différents additionneurs introduits par la relation (III-66) pourraient être pipe-linés à un niveau très bas si ce n'était la contrainte de chemin critique imposée par l'accumulateur. La solution est d'utiliser des additionneurs à retenue conservée (CSA) afin de réduire la longueur de ce chemin critique et d'accélérer les additions par des arbres réducteurs.

A chaque pas de calcul, l'additionneur CSA génère deux opérandes représentant respectivement les bits « somme » et « retenue » :

$$\begin{cases} L[j] = L_R[j] + L_S[j] \\ P[j] = P_R[j] + P_S[j] = \frac{1}{2} \{ R[j-1] + P_S[j-1] \} + (L_S[j] + L_R[j]) \end{cases} \quad (III-67)$$

Les indices s et r représentent respectivement « somme » et « retenue ».



Ainsi, l'accumulateur sera formé seulement d'un CSA 4-2. La profondeur minimum du chemin critique est égale à 2 cellules Full-Adder (FA). Au pas M, après que les M bits du signal soient introduits, l'accumulateur génère le résultat Y, lui aussi représenté sur 2 opérandes : la somme  $P_s[M]$  et la retenue  $P_r[M]$  :

$$Y = P_s[M] + P_r[M] \tag{III-68}$$

Pour obtenir le résultat final Y, l'utilisation d'un additionneur à propagation de retenue CPA est nécessaire en sortie du CSA 4-2. La propagation de la retenue finale peut encore être accélérée par la logique de retenue anticipée (carry-look ahead) dans un additionneur CPA, ou bien par un circuit en pipe-line à base de cellules Full-Adder (FA).

**6. 3. 4.2 - Propagation de la retenue**

Puisque les opérandes de somme et de retenue formant le résultat Y se présentent en parallèle, nous pouvons réaliser la propagation de la retenue un pipe-line à base de cellules Full-Adder (FA) avec un cycle aussi petit que l'on veut. Ceci entraîne une seule conséquence, à savoir l'augmentation de la latence de l'architecture et le nombre de registres contrôlant la propagation .

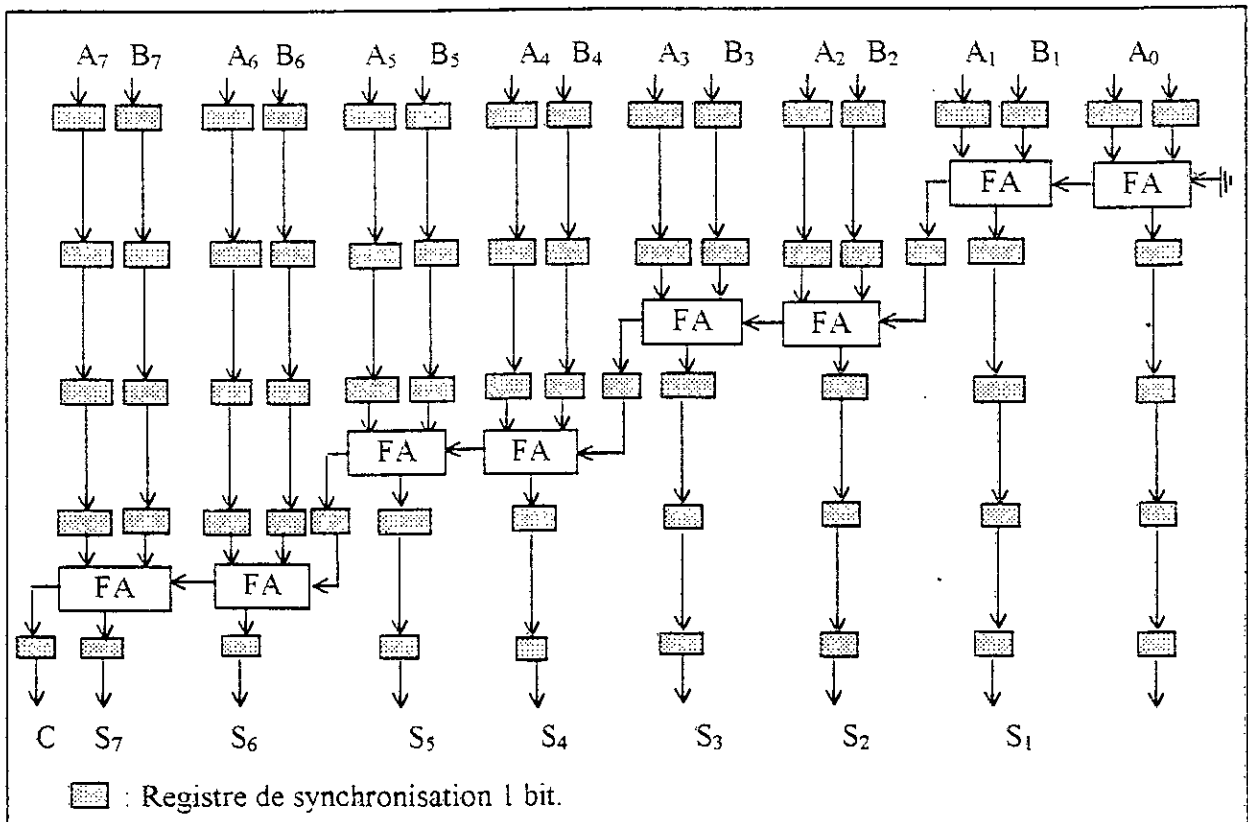


Figure III.23 - Principe de propagation de la retenue en pipeline ( période 2 F-A).

Cette figure illustre le principe d'un additionneur à propagation de retenue tel que :

- les deux opérandes à additionner sont sur 8 bits.
- le pipeline est introduit pour que le chemin critique de base soit égal à 2 cellules F-A.

La structure de l'architecture complète du bloc d'accumulation de la TCD/TCDI est donnée par la figure suivante:

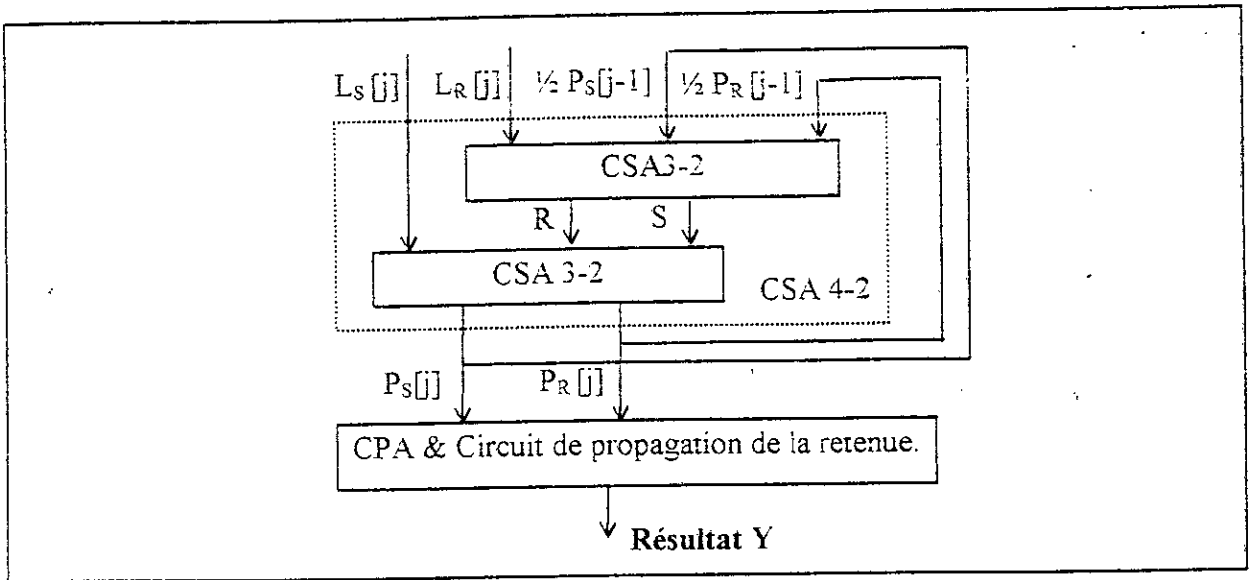


Figure III.24 - Architecture de l'étage d'accumulation de la TCD/TCDI.

#### 6.4 - DISCUSSION

L'arithmétique distribuée s'avère d'après notre étude, être un outil très adaptée au calcul des transformées orthogonales. Elle a été utilisée dans beaucoup d'opérations du traitement de signal et beaucoup de circuits DSP classiques ont été implémentés en utilisant cette arithmétique.

Le problème inhérent à cette arithmétique est la fréquence de travail non élevée de ces circuits et la solution pour arriver à de meilleures performances, n'a été cherchée que du côté technologique, à savoir implémenter au niveau transistor.

L'optimisation de l'algorithme et les techniques introduites dans notre travail, comme la diminution du chemin critique de l'équation de récurrence associée à la distribution de la ROM, et l'utilisation d'additionneurs sans propagation de retenue, sont des arguments valables qui nous permettent de parler de performance temporelle des architectures que nous avons proposées.

#### 6.5 - REMARQUE :

Le travail, décrit dans la section 6.3 qui a aboutit à l'optimisation de l'algorithme de calcul de la TCD et la proposition d'une architecture d'implémentation, a été réalisé en association avec Mr EM. Aït Nouri précédemment ingénieur-chercheur au niveau du centre de développement des techniques avancées CDTA.

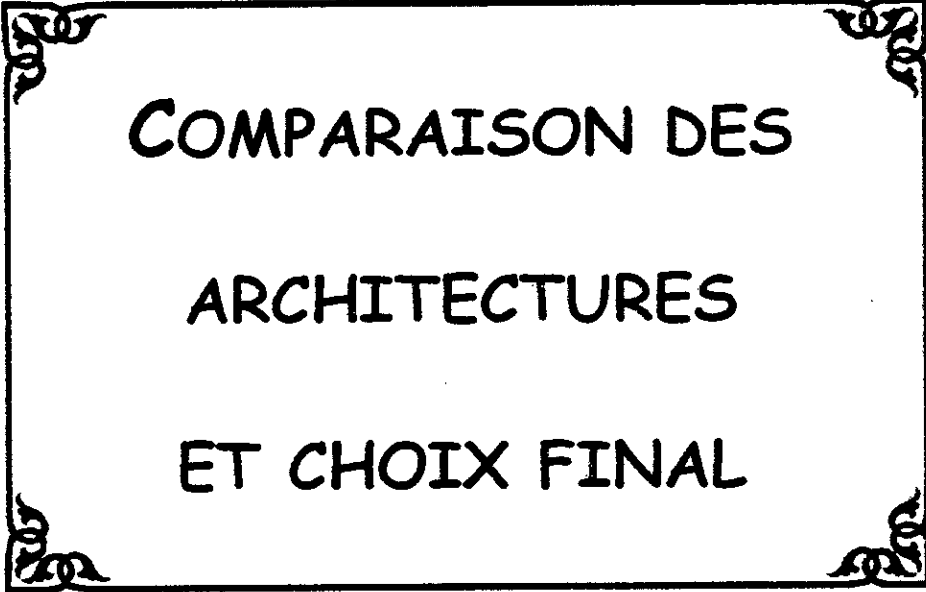
## 7. CONCLUSION

Dans le chapitre précédent, nous avons montré que l'adéquation algorithmes-architectures pour notre application, exigeait une décomposition matricielle de notre transformée. Encore que toutes les possibilités algorithmiques sont conditionnées par les modes de calcul utilisés.

C'est dans ce sens que nous avons appliqué dans le présent chapitre, les modes de calcul On-Line et Half-Line et l'arithmétique distribuée qui s'avèrent être des outils très puissants permettant la dérivation d'algorithmes de calcul efficaces et la conception d'architectures très structurées avec des mémoires réduites permettant une reconfigurabilité TCD/TCDI et l'utilisation de réseaux additionneurs totalement parallèles et la technique pipe-line pour une grande rapidité des calculs.

La comparaison en performances en vue d'un choix final de l'une des architectures obtenues avec les deux arithmétiques utilisées, fera l'objet du chapitre prochain.

## CHAPITRE IV



COMPARAISON DES  
ARCHITECTURES  
ET CHOIX FINAL

## **1. INTRODUCTION**

Les algorithmes de calcul de la T.C.D dérivés, dans le chapitre précédent, par l'application du mode de calcul Half-line et l'arithmétique distribuée, présentent des structures générales identiques. Dans le présent chapitre, nous abordons une étude comparative en vue d'un choix final de l'une des architectures obtenues à partir de ces algorithmes.

Nous précisons qu'à défaut d'une implémentation hardware de nos architectures qui permettrait une comparaison approfondie par l'évaluation précise de leurs performances effectives, nous axerons l'étude comparative de nos architectures sur l'examen des paramètres habituels de synthèse architecturale à savoir les ressources architecturales utilisées pour estimer l'espace occupé, le temps d'exécution et la précision de calcul effectué pour l'application considérée.

La comparaison en espace consistera à évaluer les ressources fonctionnelles et les ressources de stockage de chaque architecture pour connaître son espace mémoire et son espace opératif. Pour la comparaison temporelle, nous assumerons une implémentation en logique câblée de nos architectures pour évaluer le temps de propagation de leurs différentes ressources et estimer la fréquence du traitement. Quant à la précision assurée par chacune de nos architectures, nous l'assumerons à celle de leurs algorithmes de dérivation, en fonction du nombre de bits du signal d'entrée et la taille des noyaux cosinus de la transformée T.C.D.

## **2. MODELES DE COMPARAISON**

Les modèles existants dans la littérature pour la comparaison de circuits électroniques, dépendent fondamentalement du type d'implémentation adopté. Dans notre cas, nous utiliserons deux modèles choisis en fonction des paramètres à comparer.

### **2.1 - LE MODELE DE COMPARAISON DES RESSOURCES ARCHITECTURALES [60]**

Ce modèle s'attelle à comparer les ressources utilisées dans chaque architecture en termes d'unités fonctionnelles ou opératives comme les additionneurs, les multiplieurs, et en termes d'unités de stockage et d'unités de transfert des données. La connaissance de toutes les ressources d'un circuit permet d'apprécier approximativement l'espace qu'il occupe.

### **2.2 - LE MODELE DE COMPARAISON DE KAI HWANG [55]**

Dans son étude sur l'arithmétique des ordinateurs [55], Kai Hwang adopte un modèle de comparaison d'architectures en termes de rapidité des différentes unités arithmétiques utilisées. Il y définit une unité de temps de propagation  $\Delta$  correspondant à un niveau de circuit logique. La valeur  $\Delta$  est donnée par le temps de propagation d'une porte NAND ou d'une porte NOR.

Ce modèle sied beaucoup à la comparaison de circuits conçus à base de cellules. Il suffit de décrire ces cellules en portes logiques simples et d'évaluer le temps de propagation des différents étages de l'architecture et ainsi estimer son temps d'exécution. Le tableau suivant donne les temps de propagation des portes logiques typiques en termes de multiple de l'unité  $\Delta$  :

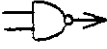

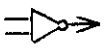




Porte	Symbole logique	Temps de propagation
NAND		$\Delta$
NOR		$\Delta$
NOT		$\Delta$
AND		$2 \Delta$
OR		$2 \Delta$
XOR		$3 \Delta$
XNOR		$3 \Delta$

Tableau IV.1 - Symboles logiques et temps de propagation de portes logiques.

### 3. LA COMPARAISON SPATIALE

La comparaison spatiale dépend du type d'implémentation adopté dans les circuits. Dans notre cas, nous allons comparer l'espace de chacune de nos architectures en termes des ressources qu'elles utilisent. En effet, les ressources de conception en synthèse architecturale, servent à implémenter en hardware différents types de fonctions, et peuvent être classées comme suit :

- Les ressources fonctionnelles de traitement des données, servant à l'implémentation des fonctions arithmétiques ou logiques et pouvant être groupées en deux sous-classes :
  - les ressources primitives qui sont les unités conçues avec beaucoup d'attention du fait qu'elles sont fortement utilisées dans le circuit;
  - les ressources spécifiques qui sont les unités conçues pour une tâche particulière.
- Les ressources de stockage des données comme les registres et les mémoires;
- Les ressources interfaces de transfert des données comme les bus.

D'après ce classement, nous définissons pour nos besoins de comparaison deux (02) types d'espaces de comparaison, à savoir l'espace mémoire et l'espace opératif.

#### 3.1 - L'espace mémoire

La comparaison en espace mémoire revient à évaluer le nombre et la taille des mémoires ROMs utilisées pour le stockage des données dans chacune de nos architectures, ainsi que le nombre de leurs circuits de décodage-lignes et décodage-colonnes.

### 3.2. L'espace opératif

La comparaison en espace opératif revient à évaluer le nombre de ressources utilisées dans la partie opérative de chacune de nos architectures. Il s'agit donc du nombre d'additionneurs utilisés dans les blocs de normalisation et d'accumulation et dans les blocs de sélection.

## 4. LA COMPARAISON TEMPORELLE

La comparaison temporelle de nos architectures consiste à évaluer le temps de propagation des différents étages de ressources utilisées afin d'estimer le temps d'exécution global et la fréquence de nos architectures.

### 4.1. Temps de propagation des ressources architecturales

Le temps d'exécution d'un étage est défini comme étant le nombre d'unités de temps nécessaire pour générer le premier résultat. A défaut d'une implémentation réelle dans notre cas, nous l'assumerons au temps de propagation dans cet étage, obtenu en décrivant ses ressources en fonction de portes logiques ou de cellules typiques aux temps de réponse connus. Ces cellules typiques sont les cellules full-adders FA autour desquelles sont construits les additionneurs des différents étages de nos architectures.

En assumant une implémentation en logique câblée, le tableau suivant donne le temps de propagation des cellules FA et des principales ressources de nos architectures [55] :

Ressources architecturales	Temps de propagation
Cellule Full-Adder (FA)	$2 \Delta$
Multiplexeurs et démultiplexeurs	$2 \Delta$
Additionneur/Soustracteur en C2 (M bits)	$2 (M+2) \Delta$
Additionneur à retenue anticipée CLA (8 bits)	$8 \Delta$
Additionneur à retenue conservée CSA 3-2	$2 \Delta$
Additionneur à retenue conservée CSA 4-2	$4 \Delta$

Tableau IV.2 - Temps de propagation des ressources opératives de nos architectures [55].

### 4.2. La fréquence ( le chemin critique )

Le concept de « chemin critique » dans une structure donnée comme celle de la figure IV.1, doit bénéficier d'une considération particulière puisque c'est lui qui fixe en quelque sorte la fréquence de traitement. La latence d'une telle structure, détermine elle, le nombre d'unités de temps entre deux itérations.

Nous rappelons qu'une structure dite à haut débit est celle qui a une fréquence de traitement élevée. Par suite, nous considérons dans notre comparaison uniquement la fréquence.

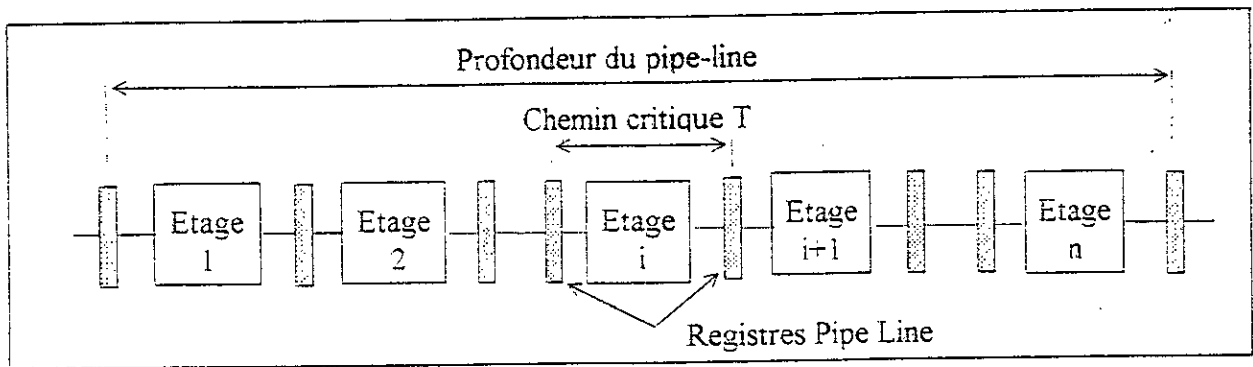


Figure IV.1 - Structure générale d'une architecture pipe-line.

Dans notre cas, le chemin critique est localisé pour l'algorithme Half-line et celui de l'arithmétique distribuée respectivement dans le bloc de sélection et le bloc d'accumulation où nous avons une boucle imbriquée. Ce paramètre est localisé précisément dans la partie opérative, et dépend du nombre de cellules Full-Adders (FA) qui y sont utilisées.

Le chemin critique est donné par :

$$T = t_{FA} \cdot N_{FA} \quad (IV-1)$$

où  $t_{FA}$  est le temps de réponse d'une cellule FA et  $N_{FA}$  le nombre de cellules FA utilisées.

En considérant l'implémentation en logique câblée, ce chemin critique est égal à :

$$T = (2 \cdot N_{FA}) \cdot \Delta \quad (IV-2)$$

## 5. LA COMPARAISON DE LA PRECISION

La précision d'un résultat donné est étroitement liée à celle de l'algorithme de calcul utilisé. Puisque dans le cas de nos algorithmes avec le mode Half-line et avec l'arithmétique distribuée, le calcul se fait d'une manière sérielle, nous pouvons penser qu'il suffit d'introduire un nombre de bits important pour garantir la précision voulue. Cette approche naïve nous mènera de toute évidence à deux inconvénients majeurs : une implémentation plus complexe et une latence élevée de nos algorithmes.

En pratique, il faut admettre et réaliser certains compromis pour aboutir à une étude rigoureuse. Par suite, nous devons optimiser un compromis rapidité/précision. C'est à dire au lieu d'introduire un nombre de bits important pour garantir la précision voulue, nous devons trouver le nombre de bits minimum réalisant une précision suffisante.

Pour cela, nous allons considérer deux niveaux de précision dans notre application, à savoir celle apportée par le nombre de bits des pixels d'entrée, et celle apportée par le nombre de bits des noyaux cosinus de la T.C.D.

Nous confirmerons nos résultats par rapport à la norme CCITT H.261 [23], [28] qui spécifie les valeurs d'erreurs suivantes pour un calcul sur 10 000 blocs de données prises entre -256 et 256 :



Erreur	Valeur spécifiée
Erreur maximum sur les pixels	1
Erreur quadratique moyenne maximum	0.06
Erreur quadratique moyenne totale	0.02
Erreur moyenne maximum	0.015
Erreur moyenne totale	0.0015

Tableau IV.3 - Spécifications CCITT pour 10 000 blocs de données prises dans  $[-256, 255]$ .

### 5.1 - Nombre de bits du signal d'entrée (pixels)

Dans les applications de traitement du signal, le nombre de bits du signal d'entrée est généralement préconisé par l'application considérée car la précision de ce signal est un paramètre déterminant pour l'application. Par exemple, pour un signal issu d'un convertisseur analogique-numérique, sa précision doit être la même que le nombre de bits du convertisseur.

Dans le cas de nos algorithmes, même si les modes de calcul utilisés sont considérés très flexibles quant au choix d'une précision donnée par la convergence de leurs résultats vers la valeur réelle avec le nombre d'itérations, nous ferons varier le nombre de bits du signal d'entrée pour assurer la précision requise par notre application.

### 5.2. Nombre de bits des nouveaux cosinus ou constantes A

Le nombre de bits des constantes A est, lui aussi, un paramètre influent sur la précision de nos algorithmes de calcul et même sur l'espace qu'ils requièrent. En effet, la représentation de ces constantes sur un nombre de bit restreint, nous assure un gain considérable en espace de stockage et en espace opératif, mais ce nombre de bits doit garantir la précision exigée par l'application.

En résumé, l'étude de la précision de nos algorithmes exige de tenir compte des deux paramètres cités ci-haut afin de gagner en espace d'implémentation et en rapidité des calculs.

## 6. COMPARAISON DES ARCHITECTURES

Dans cette section, nous appliquerons progressivement, pour comparer nos architectures, les modèles et les paramètres décrits précédemment selon l'ordre suivant :

- Comparaison et choix entre les architectures dérivées des algorithmes « Half-line » de la T.C.D en radical 4 et en radical 8. L'architecture retenue sera dite architecture « Half-line »;
- Comparaison et choix entre les architectures des algorithmes « Arithmétique distribuée » de la T.C.D en radical 4 et en radical 8. L'architecture retenue sera dite architecture « Distribuée »;
- et enfin comparaison de l'architecture « Half-line » et l'architecture « Distribuée » pour arriver au choix de l'architecture finale.

**6.1. APPLICATION & COMPARAISON EN MODE HALF-LINE**

**6.1.1. LES RESSOURCES ARCHITECTURALES**

Nous allons estimer l'espace mémoire, l'espace opératif et l'espace de contrôle des architectures du chapitre III, dérivées des algorithmes de calcul de la T.C.D en radical 4 et radical 8 avec le mode Half-line. Nous nous référerons aux figures III.7, III.8, III.10, et III.16.

**6.1.1.1 - L'espace mémoire**

Pour évaluer l'espace mémoire de nos deux architectures, nous considérerons que :

- les noyaux cosinus ou constantes A sont représentées chacun sur  $M_H$  bits;
- les mémoires ROMs sont de capacité de stockage de 9 mots.

Le nombre de bits,  $M_H$ , est le plus petit nombre de bits des constantes A qui assure une précision suffisante de l'algorithme T.C.D en mode Half-line. Ce nombre sera déterminé dans l'étude de précision de l'algorithme.

Les blocs de normalisation des figures III.7 et III.10 donnés au chapitre III, respectivement pour la T.C.D en radical 8 et 4 ainsi que le tableau III.2, nous permettent de dresser le tableau suivant donnant l'espace mémoire des deux architectures en mode half-line :

Architecture en mode half-line	Espace mémoire occupé
T.C.D EN RADICAL 4	2* (2 ROMs de 4 mots) 2* ( 2 décodeurs-lignes) 2* ( 2 décodeurs-colonnes)
T.C.D EN RADICAL 8	4 ROMs de 4 mots 4 décodeurs-lignes 4 décodeurs-colonnes

Tableau IV.4 - Espace mémoire des architectures Half-line de la T.C.D.

D'après ce tableau, nous constatons que l'espace mémoire occupé par les deux architectures en mode half-line ne dépend pas du radical de décomposition de notre transformée.

**6.1.1.2 - L'espace opératif**

Nous allons évaluer et comparer les ressources opératives dans les différents blocs de nos architectures en mode half-line :

- Les blocs de normalisation des figures III.7 et III.10 respectivement de la T.C.D en radical 8 et 4, utilisent le même nombre d'additionneurs CSA mais en radical 4, un bloc de pré-traitement formé d'un additionneur/soustracteur des entrées, est exigé.
- Quant au bloc de sélection donné en figure III.16 pour les des deux algorithmes, sa structure est la même quelque soit la décomposition de T.C.D, mais il sera doublé pour le radical 4 pour avoir en parallèle les coefficients T.C.D pairs et impairs.

Les parties opératives pour les deux algorithmes, sont alors listées dans le tableau suivant :

Espace opératif	T.C.D en radical 4	T.C.D en radical 8
<b>Bloc de Normalisation</b> (Figures III.7 et III.10)	<ul style="list-style-type: none"> <li>• 1 Bloc de pré-traitement formé par 1 Additionneur/Soustracteur</li> <li>• 2 Additionneurs CSA</li> </ul>	<ul style="list-style-type: none"> <li>• 2 Additionneurs CSA</li> </ul>
<b>Bloc de Sélection</b> ( Figure III.16 )	<ul style="list-style-type: none"> <li>• 2 * ( 3 Additionneurs CSA ) ;</li> <li>• 2 * ( 1 Additionneur CLA ) ;</li> <li>• 2 * ( 1 circuit de sélection ).</li> </ul>	<ul style="list-style-type: none"> <li>• 3 Additionneurs CSA;</li> <li>• 1 Additionneur CLA;</li> <li>• 1 circuit de sélection.</li> </ul>

Tableau IV.5 - Espace opératif des architectures Half-line de la T.C.D.

Nous constatons que l'espace opératif augmente avec la diminution du radical de décomposition de la T.C.D. Autrement dit, l'architecture en radical 8 exige moins de ressources opératives.

### 6.1.2. PERFORMANCES TEMPORELLES

Nous allons à présent, déterminer les performances temporelles de nos architectures.

#### 6.1. 2. 1 - Estimation du temps de propagation global

L'estimation du temps de propagation de nos architectures passe par la sommation des temps de réponse de leurs différents étages de ressources pour lesquelles nous assumerons une implémentation en logique câblée.

Le tableau ci-dessous donne le temps de propagation global des ressources opératives de nos architectures pour lesquelles nous assumerons que :

- le temps de propagation des ressources mémoire est le temps de réponse des circuits de décodage et de multiplexage des données qui y sont adressées et sélectionnées;
- les temps de propagation des circuits de décodage des mémoires et du circuit de sélection du résultat, sont le maximum des sommes des temps de réponse des portes qu'ils utilisent, soit :
  - $(3 \Delta + \Delta + 2 \Delta) = 6 \Delta$  pour les circuits de décodage des mémoires;
  - $(3 \Delta + 2 \Delta) = 5 \Delta$  pour le circuit de sélection du résultat.
- pour toutes les autres ressources utilisées, leurs temps de propagation pour une implémentation en logique câblée sont considérés.

La contrainte de non connaissance du temps de propagation dans le cas d'une implémentation en logique câblée, des circuits de décodage mémoire, ne faussera pas les résultats de la comparaison car toutes nos architectures utilisent les mêmes ressources mémoire.

Blocs des Architectures Half-line de la T.C.D	Temps de propagation des ressources utilisées	
	T.C.D en radical 4	T.C.D en radical 8
Bloc de Pré-traitement	Additionneur/Soustracteur 2 (M+2) Δ	-----
Bloc de Normalisation	<u>Mémoires ROMs :</u> * Décodage : 6 Δ * Multiplexage : 2 Δ <u>Accumulations</u> * 1 Additionneur CSA 3-2 : 2 Δ	<u>Mémoires ROMs :</u> * Décodage : 6 Δ * Multiplexage : 2 Δ <u>Accumulations</u> * 1 Additionneur CSA 4-2 : 4 Δ
Bloc de sélection	3 Additionneurs CSA : 6 Δ 1 Additionneur CLA : 8 Δ 1 circuit de sélection : 5 Δ	3 Additionneurs CSA : 6 Δ 1 Additionneur CLA : 8 Δ 1 circuit de sélection : 5 Δ
	<b>Temps de réponse global :</b> ( 33 + 2M ) Δ	<b>Temps de réponse global :</b> 31 Δ

Tableau IV.6 - Temps de propagation des architectures Half-line de la T.C.D.

**6.1.2. 2 - Discussion**

Les tableaux comparatifs des ressources opératives et des temps de propagation des architectures obtenues des algorithmes Half-line de la T.C.D, nous permettent dès à présent, d’opter pour l’algorithme en radical 8 pour son meilleur temps de réponse et le faible espace occupé par l’architecture par rapport à celui de l’architecture en radical 4.

Les autres paramètres de comparaison n’influent pas sur ce choix pour les raisons suivantes :

- d’une part, la fréquence et le chemin critique sont les mêmes pour les deux architectures puisque le bloc de sélection dont ils dépendent, a la même structure dans les deux cas ;
- d’autre part, le Half-line est un mode de calcul puissant assurant une précision suffisante pour les deux algorithmes et de ce fait, la précision influe moins sur la comparaison devant les ressources architecturales et le temps de réponse global.

Dans ce qui suit, seule l’architecture de la T.C.D en radical 8 sera retenue et sera dite désormais architecture « Half-line ». Nous allons à présent, estimer sa fréquence de traitement.

**6.1. 2. 3 - Estimation de la période (fréquence)**

D’après l’architecture finale de l’algorithme Half-line élaborée au chapitre III, le chemin critique est localisée dans le bloc de sélection qui est une procédure constante ne dépendant ni de la taille N de la séquence d’entrée, ni de la taille  $M_H$ , des constantes A.

Autrement dit, l'algorithme Half-line de la T.C.D travaille à fréquence constante qui d'après la conception du bloc sélection, est équivalente au temps de réponse d'un additionneur CSA 4-2, d'un additionneur CSA 3-2 et du premier niveau d'un additionneur CLA, soit :

$$\text{Temps global} = \text{Temps de réponse ( CSA 4-2 + CSA 3-2 + } \frac{1}{2} \text{ CLA )} \quad (\text{IV-11})$$

donc l'équivalent de 4 cellules FA :  $T = 4 (12 \Delta) = 48 \Delta$  (IV-12)

### 6.1.3 - L'ETUDE DE PRÉCISION

L'influence du nombre de bits du signal d'entrée (pixels) et du nombre de bits des noyaux cosinus A sur le résultat de l'algorithme de la TCD en mode Half-line a été fait par le calcul en langage C++, des erreurs spécifiées par la norme H.261 par rapport au calcul réel, pour un échantillon de dix milles (10 000) blocs de données générées aléatoirement.

Les tableaux suivants dressent respectivement les résultats du calcul des erreurs en fonction du nombre de bits du signal d'entrée et du nombre de bits des noyaux cosinus A :

NOMBRE DE BITS DU SIGNAL	ERREUR QUADRATIQUE MOYENNE MAXIMUM	ERREUR QUADRATIQUE MOYENNE TOTALE	ERREUR MOYENNE MAXIMUM	ERREUR MOYENNE TOTALE
9	0.05043	0.02062	0.01862	0.00071
10	0.04774	0.01956	0.01666	0.00072
11	0.04450	0.01823	0.01501	0.00066
12	<b>0.04130</b>	<b>0.01692</b>	<b>0.01363</b>	<b>0.00061</b>
13	0.03837	0.01572	0.01246	0.00057
14	0.03574	0.01462	0.01148	0.00053

Tableau IV.7 - Calcul d'erreurs en fonction du nombre de bits du signal d'entrée (pixels).

NOMBRE DE BITS DES NOYAUX COSINUS	ERREUR QUADRATIQUE MOYENNE MAXIMUM	ERREUR QUADRATIQUE MOYENNE TOTALE	ERREUR MOYENNE MAXIMUM	ERREUR MOYENNE TOTALE
10	0.04957	0.02030	0.01683	0.00073
11	0.04506	0.01845	0.01506	0.00067
12	<b>0.04130</b>	<b>0.01692</b>	<b>0.01363</b>	<b>0.00061</b>
13	0.03813	0.01512	0.01244	0.00057
14	0.03540	0.01404	0.01145	0.00053

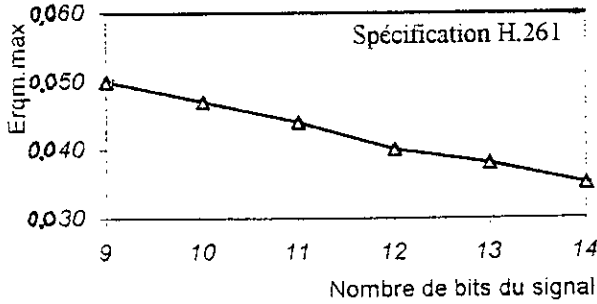
Tableau IV.8 - Calcul d'erreurs en fonction du nombre de bits des noyaux cosinus A.

Ces tableaux montrent que l'algorithme TCD avec le mode de calcul Half-line, exige au respect des spécifications de la norme H.261, une représentation du signal d'entrée ( $M_H$  bits) et des noyaux cosinus sur 12 bits (valeurs d'erreurs en gras sur les tableaux).

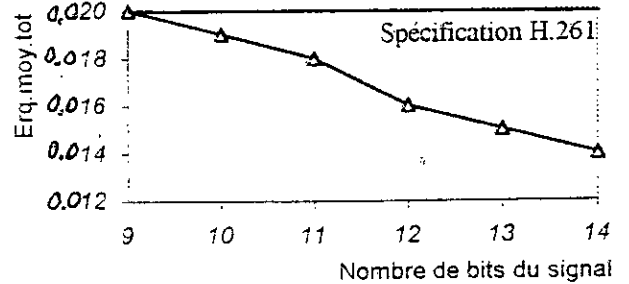
Les courbes suivantes explicitent mieux la variation de l'erreur quadratique moyenne maximum, de l'erreur quadratique moyenne totale et de l'erreur moyenne maximum en fonction du nombre de bits du signal d'entrée et de celui des noyaux cosinus.

*Courbes de variation des erreurs en mode half-line*

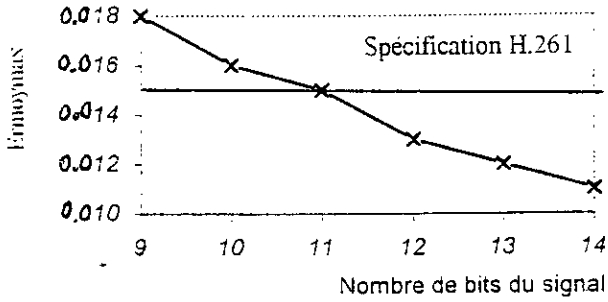
**Courbe 1** - Variation de l'erreur quadratique moyenne maximale en fonction du nombre de bits du signal d'entrée.



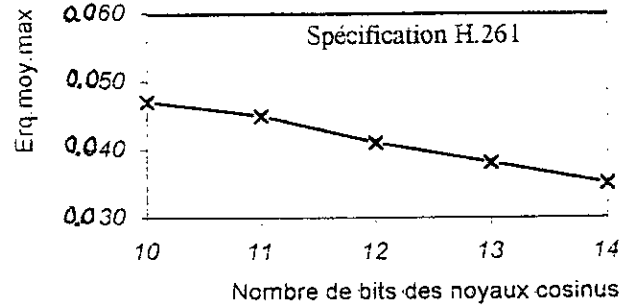
**Courbe 2** - Variation de l'erreur quadratique moyenne totale en fonction du nombre de bits du signal d'entrée.



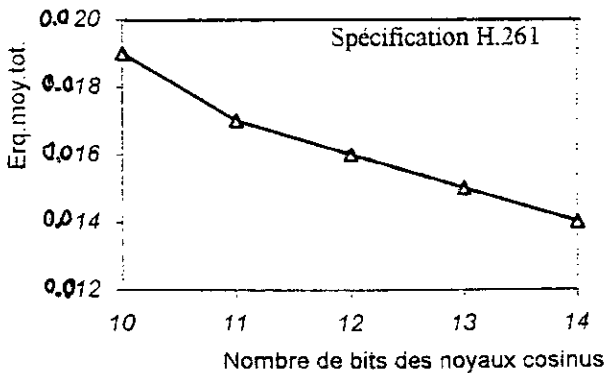
**Courbe 3** - Variation de l'erreur moyenne maximale en fonction du nombre de bits du signal d'entrée.



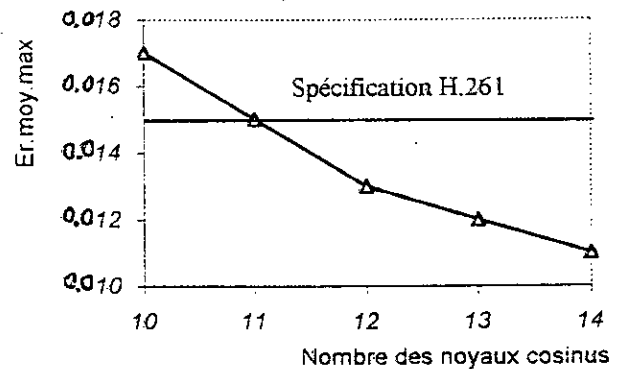
**Courbe 4** - Variation de l'erreur quadratique moyenne maximale en fonction du nombre de bits des noyaux cosinus.



**Courbe 5** - Variation de l'erreur quadratique moyenne totale en fonction du nombre de bits des noyaux cosinus.



**Courbe 6** - Variation de l'erreur moyenne maximale en fonction du nombre de bits des noyaux cosinus.



## **6.2. APPLICATION & COMPARAISON EN ARITHMETIQUE DISTRIBUEE**

De la même manière, nous allons procéder à la comparaison des performances des deux architectures obtenues avec les algorithmes de calcul de la T.C.D en radical 4 et radical 8 avec l'arithmétique distribuée.

### **6.2.1. LES RESSOURCES ARCHITECTURALES**

Nous allons estimer l'espace mémoire, l'espace opératif et l'espace de contrôle des deux architectures obtenues au chapitre III et données par les figures III-21, III-22 et III-24.

#### **6.2.1.1 - L'espace mémoire**

Comme pour le Half-line, la partie stockage dépend de la taille des noyaux cosinus A. Aussi, nous considérerons que :

- Les noyaux cosinus A sont représentés sur  $M_D$  bits ;
- Les ROMs ont une capacité de 9 mots.

Le nombre de bits,  $M_D$ , est le plus petit nombre de bits des noyaux cosinus A qui assurent une précision suffisante de l'algorithme T.C.D avec l'arithmétique distribuée. Ce nombre sera déterminé dans l'étude de précision de l'algorithme.

Le tableau suivant donne alors l'espace mémoire dans le cas de la T.C.D en radical 4 et radical 8 :

<b>Architecture</b>	<b>Espace mémoire occupé</b>
<b>T.C.D en radical 4</b>	2* (2 ROMs de 4 mots) 2* ( 2 décodeurs-lignes) 2* ( 2 décodeurs-colonnes)
<b>T.C.D en radical 8</b>	4 ROMs de 4 mots 4 décodeurs-lignes 4 décodeurs-colonnes

Tableau IV.9 - Espace mémoire des architectures T.C.D avec l'arithmétique distribuée.

#### **6.2.1. 2. L'espace Opératif**

Les figures III-21 et III-22 du chapitre III, montrent que le nombre d'additionneurs CSA est le même dans le bloc de normalisation des architectures T.C.D en radical 8 et 4. La seule différence réside dans le fait qu'en radical 4, un additionneur/soustracteur est utilisé dans le bloc de pré-traitement. Les opérations existant dans les blocs de normalisation, sont les additions sur  $M_D$  bits des différentes accumulations partielles issues des ROMs.

Quant aux blocs d'accumulation et de propagation des retenues des deux architectures, ils sont les mêmes quelque soit le radical de T.C.D, mais ils sont doublés pour le radical 4 pour avoir en parallèle les coefficients T.C.D pairs et impairs.

Les parties opératives pour ces architectures de la T.C.D, sont listées dans le tableau suivant :

Espace opératif	T.C.D en radical 4	T.C.D en radical 8
<b>Bloc de Normalisation</b> (Figures III.21 et III.22)	<ul style="list-style-type: none"> <li>• 1 Bloc de pré-traitement formé par 1 Additionneur/Soustracteur</li> <li>• 2 Additionneurs CSA en parallèle</li> </ul>	<ul style="list-style-type: none"> <li>• 2 Additionneurs CSA en série</li> </ul>
<b>Bloc d'accumulation</b> ( Figure III.24 )	<ul style="list-style-type: none"> <li>• 2 * ( 1 Additionneur CSA 4-2 )</li> </ul>	<ul style="list-style-type: none"> <li>• 1 Additionneur CSA4-2</li> </ul>
<b>Bloc de propagation des retenues</b>	<ul style="list-style-type: none"> <li>• 2 * (M<sub>D</sub> /2 cellules FA )</li> </ul>	<ul style="list-style-type: none"> <li>• M<sub>D</sub> /2 cellules FA.</li> </ul>

Tableau IV.10 - Espace opératif des architectures T.C.D avec l'arithmétique distribuée.

**6.2.2. PERFORMANCES TEMPORELLES**

Maintenant, nous allons déterminer les performances temporelles de nos architectures, à savoir leurs temps de propagation et leurs fréquences.

**6.2. 2. 1 - Estimation du temps de propagation global**

L'estimation du temps de propagation global de nos architectures consiste à sommer les temps de propagation de leurs différentes ressources pour lesquelles nous assumerons une implémentation en logique câblée.

Les tableaux suivants donnent le temps de propagation de ces architectures en fonction de leurs ressources. Nous assumerons que le temps de réponse des ressources mémoire est le temps de réponse de leurs circuits de décodage et de multiplexage des données.

Blocs des Architectures de la T.C.D	Temps de propagation des ressources utilisées	
	T.C.D en radical 4	T.C.D en radical 8
<b>Bloc de pré-traitement</b>	Additionneur/Soustracteur 2 (M+2) Δ	-----
<b>Bloc de normalisation</b>	<u>Mémoires ROMs :</u> * Décodage 6 Δ * Multiplexage 2 Δ <u>Accumulations</u> * 1 Additionneur CSA 3-2: 2 Δ	<u>Mémoires ROMs :</u> * Décodage 6 Δ * Multiplexage 2 Δ <u>Accumulations</u> * 1 Additionneur CSA 4-2: 4 Δ
<b>Bloc d'accumulation</b>	1 Additionneur CSA 4-2 : 4 Δ	1 Additionneur CSA 4-2 : 4 Δ
<b>Bloc de propagation des retenues</b>	M <sub>D</sub> /2 Cellules FA : M <sub>D</sub> Δ	M <sub>D</sub> /2 Cellules FA : M <sub>D</sub> Δ
	<b>Temps de réponse global :</b> ( 2(M+2) + 14 + M <sub>D</sub> ) Δ	<b>Temps de réponse global :</b> (16 + M <sub>D</sub> ) Δ

Tableau IV.11 - Temps de propagation des architectures T.C.D avec l'arithmétique distribuée.



### 6.2.2. 2 - Discussion

Les tableaux comparatifs des ressources opératives et des temps de propagation des architectures obtenues des algorithmes T.C.D avec l'arithmétique distribuée, nous permettent d'opter pour l'algorithme en radical 8 pour son meilleur temps de réponse et le faible espace occupé par l'architecture par rapport à celui de l'architecture en radical 4.

Les autres paramètres de comparaison n'influent pas sur ce choix pour les raisons suivantes :

- d'une part, la fréquence et le chemin critique sont les mêmes pour les deux architectures ;
- d'autre part, l'arithmétique distribuée est un mode de calcul puissant assurant une précision suffisante pour les deux algorithmes et de ce fait la précision influe moins sur la comparaison, devant les ressources architecturales et le temps de réponse global.

Dans ce qui suit, nous allons alors ne retenir que l'architecture de la T.C.D en radical 8 que nous appellerons désormais architecture « distribuée ».

### 6.2. 2. 3 - Estimation de la période (fréquence)

L'algorithme de l'arithmétique distribuée établit la barre limite un chemin critique de deux cellules FA, mais pour une meilleure comparaison avec le mode Half-line, nous choisissons un chemin critique de 04 cellules FA donc une période de :

$$T = 4 (12 \Delta) = 48 \Delta \quad (\text{IV-18})$$

### 6.2.3. L'ETUDE DE PRÉCISION

La précision de l'algorithme TCD avec l'arithmétique distribuée dépend essentiellement du nombre de bits des noyaux cosinus. En effet, le résultat est généré de façon parallèle à partir d'accumulations partielles, sur un même nombre de bits que les noyaux cosinus, après un nombre d'itérations égal au nombre de bits du signal d'entrée. Le nombre de bits du signal d'entrée influe donc très peu sur la précision du résultat, vu que les accumulations partielles ne sont complétées qu'après un nombre total d'itérations.

Ce constat est vérifié par les résultats du calcul des erreurs spécifiées par la norme H.261, qui ne varient pas en fonction du nombre de bits du signal d'entrée. Le tableau suivant donne reprend les résultats obtenus seulement pour un nombre de 09 bits :

NOMBRE DE BITS DU SIGNAL	ERREUR QUADRATIQUE MOYENNE MAXIMUM	ERREUR QUADRATIQUE MOYENNE TOTALE	ERREUR MOYENNE MAXIMUM	ERREUR MOYENNE TOTALE
9	0.03155	0.01102	0.00223	0.001322

Tableau IV.12 - Calcul d'erreurs pour un nombre de 9 bits du signal d'entrée.

Quant aux résultats du calcul d'erreurs en fonction du nombre de bits des noyaux cosinus A, ils sont dressés dans le tableau suivant :

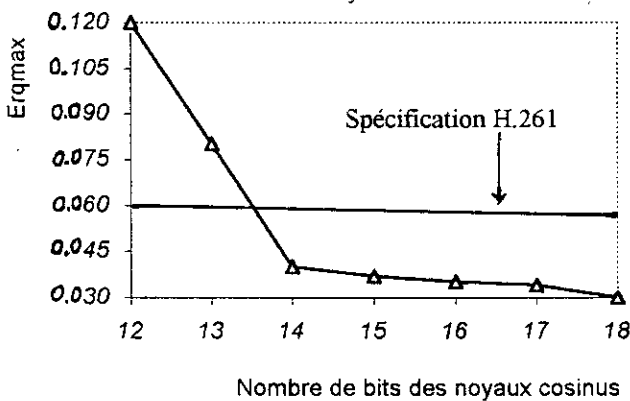
NOMBRE DE BITS DES NOYAUX COSINUS	ERREUR QUADRATIQUE MOYENNE MAXIMUM	ERREUR QUADRATIQUE MOYENNE TOTALE	ERREUR MOYENNE MAXIMUM	ERREUR MOYENNE TOTALE
12	0.1210	0.06278	0.00436	0.00310
13	0.0807	0.03456	0.00342	0.00232
14	0.0404	0.01776	0.00265	0.00168
15	0.0374	0.01509	0.002531	0.00155
16	<b>0.0357</b>	<b>0.01289</b>	<b>0.00239</b>	<b>0.00143</b>
17	0.0349	0.01209	0.00236	0.00138
18	0.0309	0.01115	0.00192	0.00138

Tableau IV.13 - Calcul d'erreurs en fonction du nombre de bits des noyaux cosinus A.

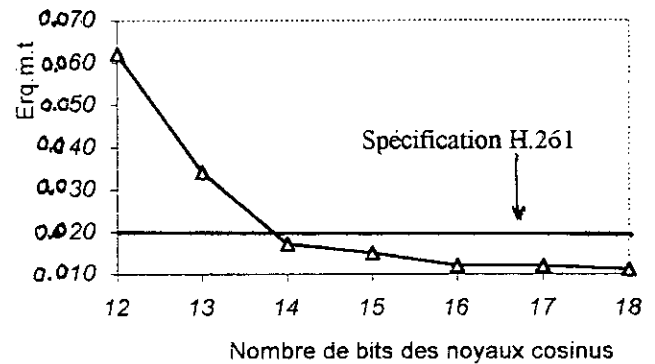
Ce tableau montre que l'algorithme TCD avec l'arithmétique distribuée exige, pour une conformité aux spécifications de la norme H.261, une représentation des noyaux cosinus sur 16 bits ( $M_D = 16$ ) (valeurs d'erreurs en gras sur les tableaux). Nous conviendrons de représenter le signal d'entrée (pixels) sur 9 bits ( $M = 9$ ).

Les courbes suivantes reprennent la variation de l'erreur quadratique moyenne (max. et totale) et de l'erreur moyenne max. en fonction du nombre de bits des noyaux cosinus.

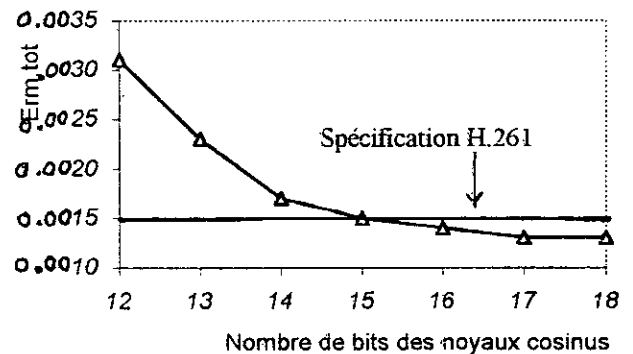
Courbe 7 -Variation de l'erreur quadratique moyenne maximale en fonction du nombre de bits des noyaux cosinus.



Courbe 8 -Variation de l'erreur quadratique moyenne totale en fonction du nombre de bits des noyaux cosinus.



Courbe 9 - Variation de l'erreur moyenne totale en fonction du nombre de bits des noyaux cosinus.



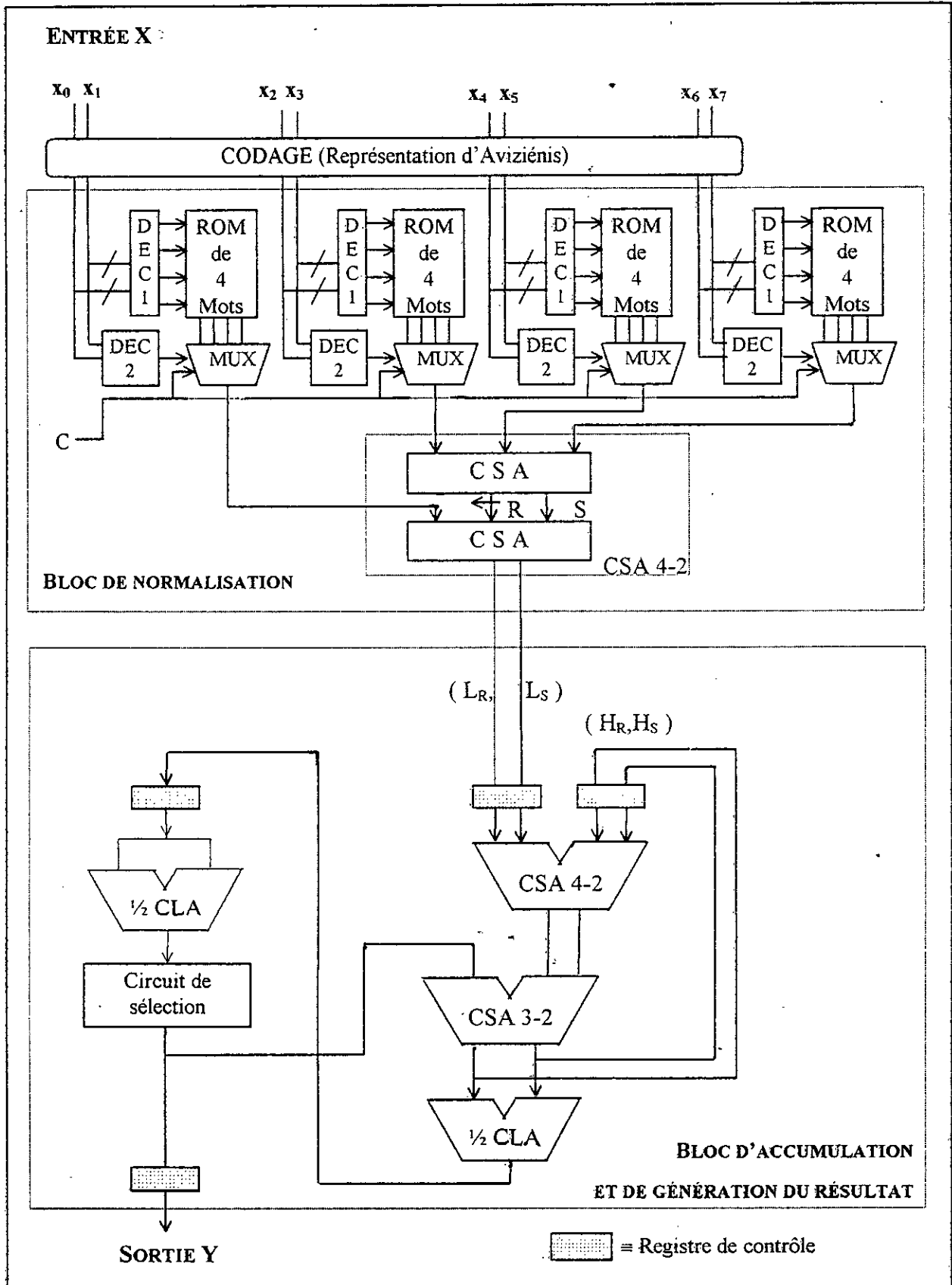


Figure IV.2- Architecture TCD/TCDI « Half-line » en radical 8.

### **6. 3. COMPARAISON DES ARCHITECTURES « HALF-LINE » ET « DISTRIBUÉE » ET CHOIX FINAL**

Dans les sections de comparaison des architectures TCD obtenues, à radical 4 et radical 8 avec chaque mode de calcul, le mode half-line et l'arithmétique distribuée, nous avons montré que les architectures à radical 8 offrent de meilleures performances temporelles et spatiales.

Le choix entre les architectures de la TCD à radical 8 retenues, s'est basé sur la comparaison de leurs temps de propagation global, des ressources architecturales qu'elles utilisent et de la précision de leurs résultats. Cela nous a révélé que l'architecture « half-line », offre par rapport à l'architecture « distribuée », un meilleur temps de propagation global ( $31\Delta$ ), un plus faible espace mémoire et une meilleure précision dans les calculs. De plus, cette architecture est plus structurée avec plus de facilités de contrôle et un meilleur pipe-line.

Poursuite, l'architecture finale, retenue pour notre application, est l'architecture TCD « half-line » (TCD à radical 8) donnée ci-après par la figure IV.2 donnée en page précédente.

## **7. CONCLUSION**

Dans ce chapitre, nous avons utilisé des modèles de comparaison des architectures dérivées des algorithmes TCD avec les deux modes de calcul utilisés, à savoir le mode half-line et l'arithmétique distribuée.

La comparaison et le choix entre les différentes architectures obtenues, a reposé sur beaucoup de critères à savoir le choix du radical de décomposition pour une meilleure reconfigurabilité TCD/TCDI, leurs performances en termes de ressources architecturales qu'elles utilisent, de leur fréquence de traitement et la précision des calculs en fonction de la taille du signal d'entrée et des noyaux cosinus.

L'architecture TCD « half-line » retenue gagnerait beaucoup à être implémentée sur un circuit dédié tant elle a de bonnes performances, une structure très régulière, et utilisant des réseaux d'additionneurs totalement parallèles, des mémoires réduites et des structures pipe-lignées pour un haut débit de calcul.



**CONCLUSION**

**GÉNÉRALE**

## CONCLUSION GÉNÉRALE

**L**a synthèse architecturale est une discipline qu'il faut absolument maîtriser de nos jours. Ceci pour pouvoir bénéficier du rapide développement des technologies d'intégration permettant la réalisation des processeurs et d'opérateurs de traitement numérique très efficaces dans les applications numériques modernes aux services très recherchés. La demande des utilisateurs étant toujours en grande expansion, le challenge se résume en la capacité de réaliser et d'implémenter des fonctions de plus en plus complexes, occupant le moins d'espace possible, assurant beaucoup d'économie et fonctionnant avec une grande rapidité.

C'est dans ce contexte que s'insère notre travail dans cette thèse consacré à la transformée cosinus discrète T.C.D. En effet, cette transformée est recommandée par diverses normes internationales, pour le codage d'images. Ses performances notamment très approuvées en traitement numérique et en transmission d'images qui sont d'ailleurs des marchés en grande diffusion, font que le développement d'algorithmes rapides de calcul de cette transformée et leur implémentation hardware est devenue une exigence chez les professionnels du codage d'images.

La méthodologie de synthèse que nous avons suivie tout au long de cette thèse, est basée sur la recherche d'une adéquation algorithmes-architectures de l'application que nous avons considérée, afin d'arriver à des algorithmes très efficaces une fois implantés. Car, quoique le traitement numérique d'images a bénéficié du développement des techniques numériques et des progrès technologiques les relations entre algorithmes rapides et architectures de traitement de signal ont évolué comme nous l'avons précisé, de manière non concurrente.

La prise en compte des influences mutuelles existant heureusement entre ces deux domaines, nous a permis de dériver des algorithmes très efficaces basés sur la technique de décomposition matricielle de notre transformée. Par ailleurs, nous avons montré que toutes les possibilités algorithmiques qui s'offraient à nous, sont conditionnées par les modes de calcul utilisés. Pour remédier à cette contrainte, nous avons appliqué des modes de calcul très puissants à savoir le mode Half-Line et l'arithmétique distribuée, qui nous ont permis de dériver et optimiser des algorithmes de calcul efficaces et la conception d'architectures très structurées avec des

mémoires réduites permettant l'utilisation de réseaux additionneurs totalement parallèles et la technique pipe-line pour une grande rapidité des calculs.

La comparaison en performances des architectures obtenues avec les deux arithmétiques utilisées, en vue d'un choix final, a consisté à évaluer leurs différents ressources architecturales en termes d'espace mémoire et d'espace opératif requis, leurs performances temporelles en temps de propagation global et fréquence de traitement, ainsi que la précision apportée par les algorithmes adoptés.

À défaut d'une implémentation hardware réelle pour l'évaluation effective des performances des architectures, nous précisons que les modèles d'estimation que nous avons utilisés pour l'évaluation des espaces occupés par les architectures et du temps de propagation global, ont été soigneusement présentés avec beaucoup de clarté et de simplicité pour un meilleur éclaircissement sur le choix final de l'architecture unifiée TCD/TCDI 2-D à adopter. Certes, l'évaluation du temps de propagation global n'a pas tenu compte des pipe-lines utilisés dans les architectures, mais nous faisons remarquer que nous avons utilisé une même approche d'estimation de ce paramètre, pour toutes les architectures, pour faciliter la comparaison.

Il va sans dire que nous espérons que notre présent travail de synthèse, soit complété par une implémentation hardware afin de connaître les performances réelles de l'architecture finale, et que les modes de calcul que nous avons utilisés et largement présentés, soient exploités pour être utilisés dans d'autres applications à même de conduire à de meilleurs résultats pour les problèmes traités.

Il conviendra de noter aussi que ce travail nous a permis une bonne investigation dans beaucoup de domaines, récompensée par un large enrichissement de nos connaissances en techniques de compression d'images, en dérivation d'algorithmes rapides, en arithmétique, et en synthèse et conception architecturales.

# ANNEXE A



## RAPPEL SUR LES ADDITIONNEURS PARALLÈLES



## A.1 - ADDITIONNEUR A RETENUE CONSERVEE (CSA) [55]

Les additionneurs traditionnels sont conçus pour sommer deux opérandes. Mais il existe des opérations arithmétiques qui exigent que l'addition se fasse sur un large nombre d'opérandes. Nous présentons dans ce qui suit un type d'additionneur multi-opérandes, qui peut être utilisé pour réaliser des additions multiples, des multiplications et quelques fonctions spéciales de génération.

Quand des additions multiples sont accomplies tout en sauvegardant la propagation de la retenue jusqu'à ce que toutes les additions soient complétées, et ensuite prendre un cycle final (ou un nombre de cycles finaux) pour compléter la propagation de la retenue, on obtient un additionneur à retenue conservée CSA (Carry Save Adder).

Ces additionneurs sont construits autour de cellules élémentaires « full-adder » décrites ci-après.

### A.1.1 - Rappel sur la cellule full-adder (FA)

Une cellule full-adder 1-bit (FA) additionne deux bits  $A_i$  et  $B_i$ , et une retenue entrante  $R_i$  pour produire une somme  $S_i$  et une retenue sortante  $R_{i+1}$  données par les relations suivantes :

$$S_i = R_i \oplus A_i \oplus B_i \quad (\text{A-1})$$

$$R_{i+1} = A_i \cdot B_i + R_i \cdot (A_i \oplus B_i) \quad (\text{A-2})$$

L'implémentation de ces deux relations, donne un additionneur séquentiel très simple.

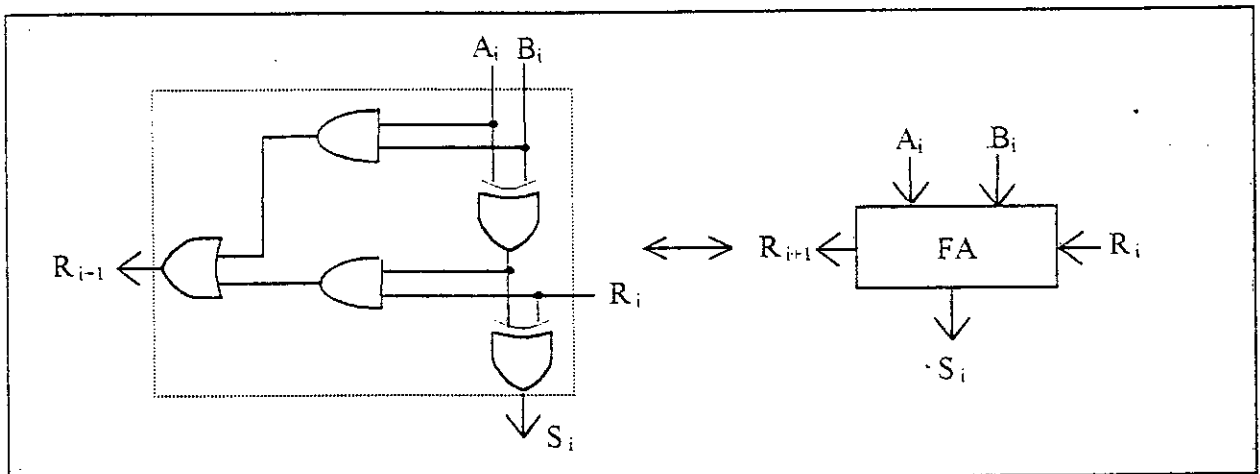


Figure A-1. Schéma élémentaire et représentation d'une cellule full-adder.

### A.1.2 - Additionneur CSA n-bit

Un CSA n-bit consiste en n cellules full-adder (FA) dont les sorties « retenue » vont à un registre intermédiaire appelé : registre de sauvegarde de retenue (CSR). Les retenues entrantes des n FAs forment un troisième vecteur d'entrées à l'additionneur. Les trois ensembles d'entrées

désignés par A, B et C, peuvent être utilisés avec interchangeabilité sans affecter les sorties. Les deux vecteurs de sortie sont désignés respectivement par S et R pour la somme et la retenue. Les indices  $i$ ,  $i=0$  à  $n-1$ , indiquent les positions des bits des nombres.

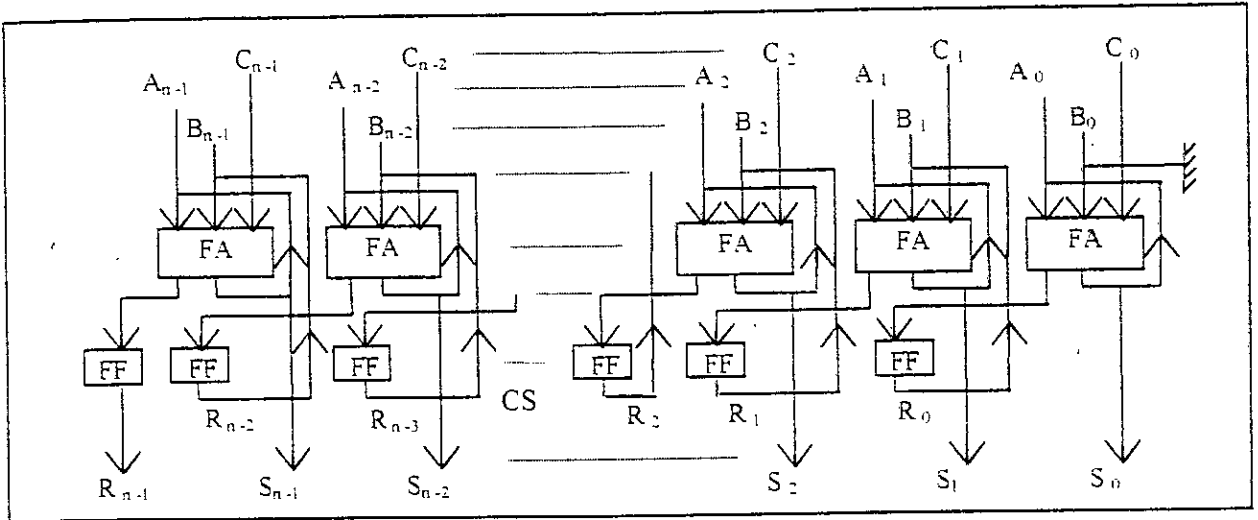


Figure A.2 - Schéma logique d'un CSA n-bits avec CSR.

Les CSAs sont attractifs pour leur faible exigence hardware. Cependant, l'étape finale de la propagation de la retenue est faite de façon séquentielle (ripple-carry), qui peut rendre le processus total de l'addition lent surtout si la longueur  $n$  des mots est grande. Une amélioration en vitesse peut être obtenue en utilisant un additionneur à deux entrées séparées à la dernière étape pour sommer le vecteur «Sommes» et le vecteur «Retenue» du CSA en une somme finale. Cet additionneur conventionnel est dit à propagation de retenue CPA (Carry-propagate Adder).

L'interconnexion du CSA avec le CPA est montré en figure A-3, et la propagation de la retenue finale peut être accélérée par la logique de retenue anticipée (Carry Look Ahead) dans le CPA.

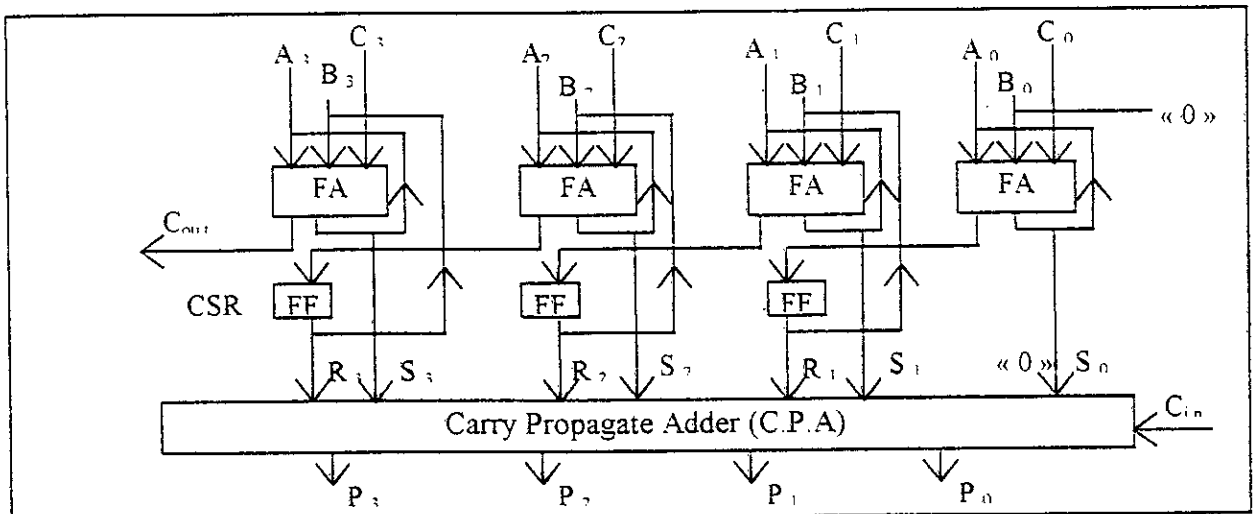


Fig. A.3 - Schéma d'un CSA/CPA (n=4 bits) pour une addition multi-opérandes en C2.

$$A + B + C \xrightarrow{\text{(CSA)}} S + R \xrightarrow{\text{(CPA)}} P$$

### A.1.3 -CSAs MULTI-NIVEAUX

Les CSAs multi-niveaux sont utilisés pour réaliser des additions multi-opérandes car il est possible d'interconnecter un nombre de CSAs comme un arbre additionneur multi-niveaux. Les deux exemples suivants permettent de suivre l'addition de 3 et 4 opérandes à la fois par des structures d'arbres CSAs.

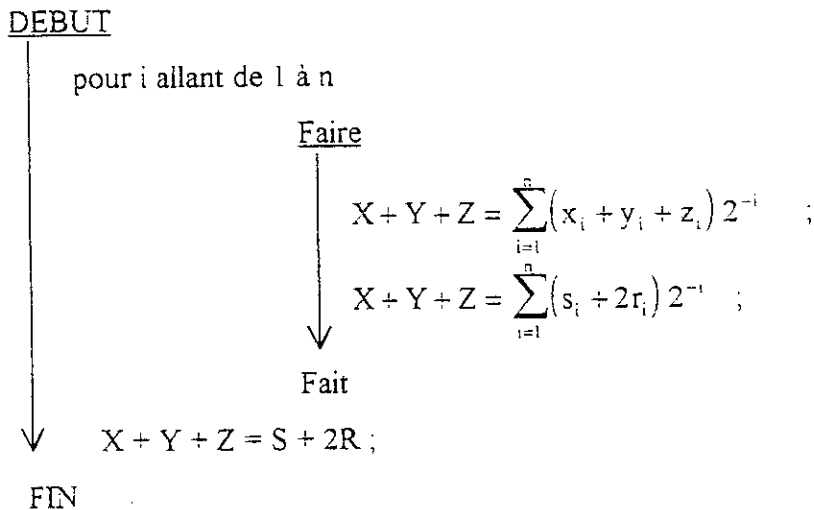
#### A.1.3.1 - Additionneur CSA 3-2

Cet additionneur sert à sommer en parallèle 3 opérandes à la fois.

En effet, soit à additionner 3 opérandes X, Y et Z en utilisant un additionneur CSA, avec:

$$X = \sum_{i=1}^n x_i 2^{-i}, \quad Y = \sum_{i=1}^n y_i 2^{-i} \quad \text{et} \quad Z = \sum_{i=1}^n z_i 2^{-i} \quad (\text{A-3})$$

L'algorithme réalisant cette addition est donné comme suit :



La schématisation de cet additionneur est montrée par la figure A.4.(a).

#### A.1.3.2 -Additionneur CSA 4-2

Cet additionneur sert à sommer en parallèle 4 opérandes à la fois.

En effet, soit à additionner 4 opérandes A, B, C et D tel que :

$$A = \sum_{i=1}^n a_i 2^{-i}, \quad B = \sum_{i=1}^n b_i 2^{-i}, \quad C = \sum_{i=1}^n c_i 2^{-i} \quad \text{et} \quad D = \sum_{i=1}^n d_i 2^{-i} \quad (\text{A-4})$$

$$A + B + C + D = S + 2R \quad (\text{A-5})$$

Cet additionneur utilise un additionneur CSA 3-2 pour calculer en premier la quantité (A+B+C) puis la somme et la retenue résultante avec l'opérande D avec un second additionneur CSA 3-2. donc l'opération est faite en deux étapes, comme le montre l'algorithme suivant.

DEBUT

Pour i allant de 1 à n

Faire

$$A + B + C + D = \sum_{i=1}^n (a_i + b_i + c_i) 2^{-i} + D ;$$

$$A + B + C + D = \sum_{i=1}^n (s'_i + 2r'_i + d_i) 2^{-i} ;$$

$$A + B + C + D = \sum_{i=1}^n (s_i + 2r_i) 2^{-i} ;$$

Fait

$$A + B + C + D = S + 2R ;$$

FIN

La schématisation de cet additionneur est montrée par la figure A.4.(b). La flèche sur les retenues sortantes du premier CSA, indique qu'elles sont décalées à gauche d'un bit avec l'introduction d'un « 0 » de l'extrémité droite avant d'être introduites dans le second CSA .

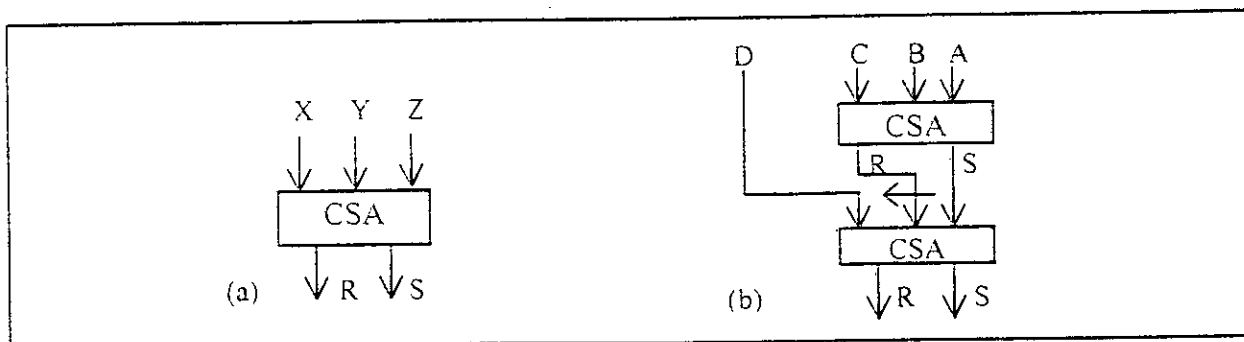


Figure A.4 - Arbres CSAs : (a) : à 3 entrées (CSA 3-2) . (b) : à 4 entrées (CSA 4-2).

### A.2 - ADDITIONNEUR A RETENUE ANTICIPEE CLA [44]

Au lieu de propager ou de sauvegarder les retenues, cet additionneur calcule à l'avance leurs valeurs grâce à des circuits logiques supplémentaires.

Nous rappelons que pour l'addition de deux nombres A ( $A_{n-1} A_{n-2} \dots A_0$ ) et B ( $B_{n-1} B_{n-2} \dots B_0$ ) écrits en numération simple de position ou en complément à deux C2, le comportement de la retenue est déterminé par les quantités suivantes :

$$\begin{cases} P_i = A_i \oplus B_i & (\text{si } P_i = 1, \text{ la retenue est propagée par l'étage } i) \\ G_i = A_i \cdot B_i & (\text{si } G_i = 1, \text{ l'étage } i \text{ génère une retenue}) \end{cases}$$

En somme, le principe des additionneurs à évaluation anticipée de la retenue, consiste à évaluer de manière totalement parallèle, en 3 étapes :

- Les couples ( $P_i, G_i$ );
- Les retenues  $C_i$  :  $C_i = G_{i-1} + G_{i-2} \cdot P_{i-1} + G_{i-3} \cdot P_{i-2} \cdot P_{i-1} + \dots + G_0 \cdot P_1 \cdot P_2 \dots P_{i-1} + C_0 \cdot P_1 \cdot P_2 \dots P_{i-1}$ ;
- Les bits de la somme :  $S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$ .

La formule de calcul de la retenue  $C_i$  est très lourde. Ceci montre que ce principe ne s'applique correctement qu'à de petits additionneurs (4 à 8 bits). Par suite, les CLA sont souvent utilisés comme petits blocs constitutifs de plus gros additionneurs.

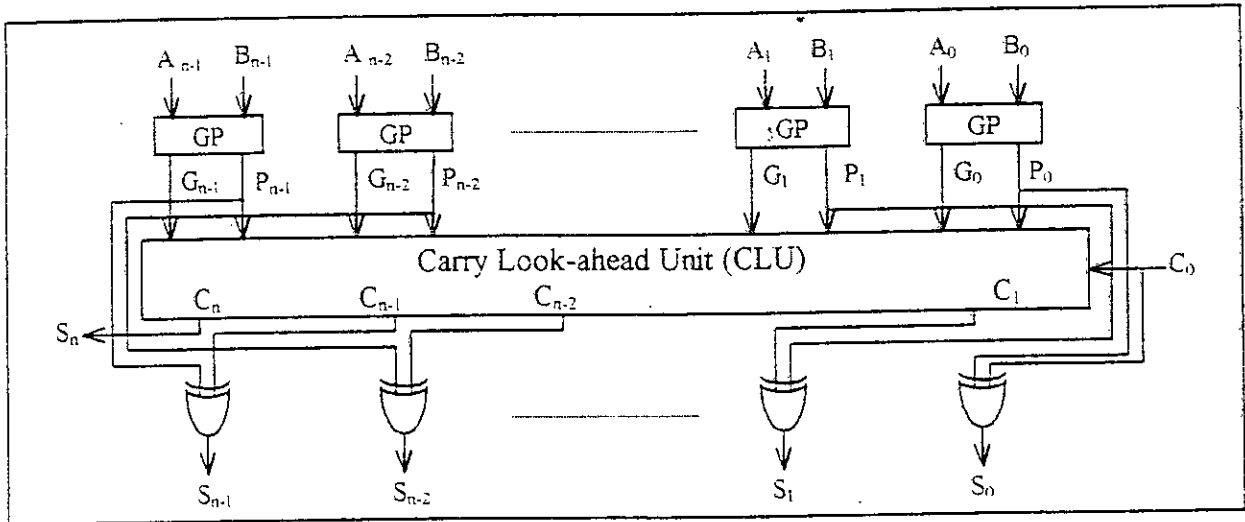


Figure A.5 - Schéma logique d'un additionneur CLA n-bits.

L'étage CLU (carry look ahead unit) est l'unité de calcul des retenues  $C_i$ . La figure suivante présente cette unité dans le cas d'un CLA 4-bit :

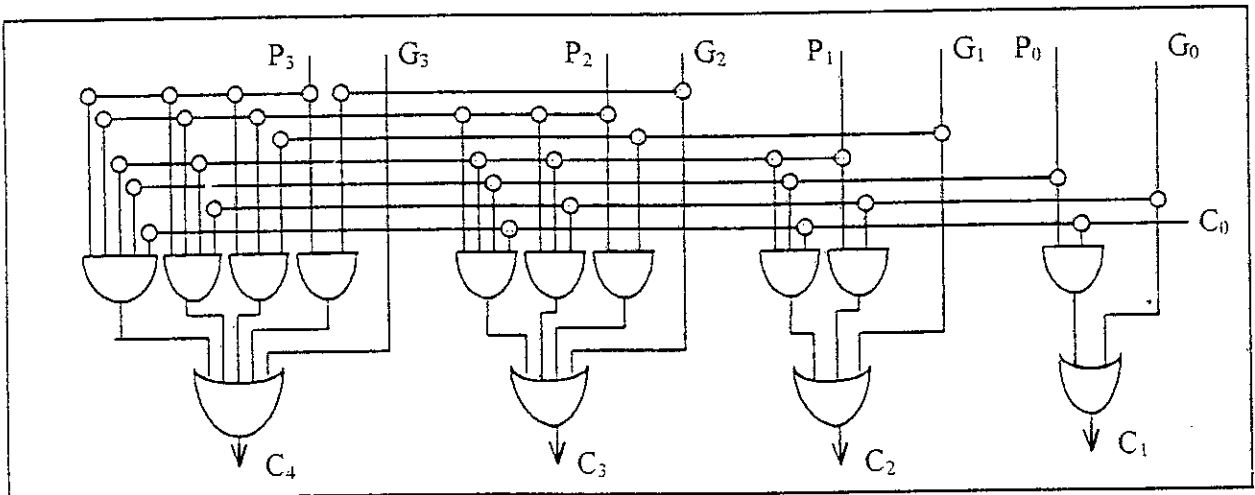


Figure A.6 - Schéma logique de l'étage CLU 4-bits.

# ANNEXE B



## PROGRAMMATION EN LANGAGE C<sup>++</sup>

```

/*-----!
!          CALCUL DE LA TCD-radix_8 PAR LE MODE HALF-LINE          !
!-----*/

#include<stdio.h>
#include<stdio.h>
#include<math.h>
#include<conio.h>
unsigned char X[8][8];
int y[8][8][16],x[8][8][16],i,j,k,l,m,n,g,nb,nbp,p = 3;
float Z,s,Yr[8][8],yc[8][8],pi = 3.141592654,

/*-----Bloc de transformation radix 8 -----*/

                                □
A[8][8]={{ 0.176776695, 0.176776695, 0.176776695, 0.176776695,
           0.176776695, 0.176776695, 0.176776695, 0.176776695},
 { 0.245196320, 0.207867403, 0.138892558, 0.048772581,
   -0.048772581,-0.138892558,-0.207867403,-0.245196320},
 { 0.230969883, 0.095670858,-0.095670858,-0.230969883,
   -0.230969883,-0.095670858, 0.095670858, 0.230969883},
 { 0.207867403,-0.048772581,-0.245196320,-0.138892558,
   0.138892558, 0.245196320, 0.048772581,-0.207867403},
 { 0.176776695,-0.176776695,-0.176776695,-0.176776695,
   0.176776695,-0.176776695,-0.176776695, 0.176776695},
 { 0.138892558,-0.245196320, 0.048772581, 0.207867403,
   -0.207867403,-0.048772581, 0.245196320,-0.138892558},
 { 0.095670858,-0.230969883, 0.230969883,-0.095670858,
   -0.095670858, 0.230969883,-0.230969883, 0.095670858},
 { 0.048772581,-0.138892558, 0.207867403,-0.245196320,
   0.245196320,-0.207867403, 0.138892558,-0.048772581}};

/*----- Fonctions prototypes -----*/
void generation(void);
void conversion(int nb);
void Halfline(int nb);
float C(int n);
float TCD(int l,int g);

/*----- Fonction de generation des entrées -----*/

                                □
void generation(void)
{
    for(g=0;g<8;g++) X[j][g] = rand();
}

/*----- Fonction de conversion (b=2)-----*/
void conversion(int nb)
{
    int m;
    for(i=0;i<nb;i++) x[j][g][i] = 0;
    m = X[j][g];
    i = nb-1;
    while(m != 0)
    {
        x[j][g][i] = m % 2;
        m = m / 2;
        i--;
    }
}

/*----- Procedure Half-Line -----*/
void Halfline(int nb)
{

```

```

int m;
float L,H,R,Hl;
i = 0 ;
Z = 0.0;
R = 0.0;
while( i <= nb-1 )
{
    L = 0.0;
    for(j=0;j<8;j++) L += A[l][j] * x[j][g][i] * pow(2,-p);
    H = 2 * R + L;
    m = H - 2;
    Hl = m / 2;
    if( Hl >= 0.5 ) y[l][g][i] = 1;
    else
    if( Hl < - 0.5 ) y[l][g][i] = -1;
    else
    if( ( Hl < 0.5 ) && ( Hl >= -0.5 ) ) y[l][g][i] = 0;
    R = H - y[l][g][i];
    Z += y[l][g][i] * pow(2,-i-1);
    i++;
}
}

```

----- TCD 2D réelle -----\*/

```

float C(int n)
{
    float w;
    if ( n == 0 ) w = 1 / sqrt(2);
    else w = 1.0;
    return(w);
}

float TCD(int l,int g)
{
    float s;
    s = 0.0;
    for(i=0;i<8;i++)
        for(j=0;j<8;j++)
            s += 0.0625 * C(l) * C(g) * X[i][j] / 256 *
                cos((2*i+1)*pi*g/16) * cos((2*j+1)*pi*l/16);
    return(s);
}

```

----- Programme principal -----\*/

```

void main()
{
    char rep;
    int t,T,nba;
    float Af[8][8],Er[8][8],Yc[8][8],Ermoy,
        Emoyt,Epmoy,Erqua,Eqmoyt,Epqua;
    clrscr();
    printf("\n");
    printf("          Nombre de blocs : T=");
    scanf("%d",&T);
    printf("          Nombre de bits des pixels : nbp=");
    scanf("%d",&nbp);
    printf("\n");
    printf("          Taille des noyaux Cosinus : nba= ");
    scanf("%d",&nba);
    for (i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
        {
            A[i][j]=A[i][j]*pow(2.0,nba);
            Af[i][j]=A[i][j];
        }
    }
}

```



```

        A[i][j]=Af[i][j]*pow(2.0,0-nba);
    }
    printf("\n");
for(t=0;t<T;t++)
    {
        /* début-boucle(Blocs)*/
        printf("Bloc n°: %d \n",t);
/*----- Initialisation des erreurs -----*/
        Emoyt=0.0;
        Epmoy=0.0;
        Eqmoyt=0.0;
        Epqua=0.0;
/*----- g,n,ratlon des pixels -----*/
        for(j=0;j<8;j++) generation();
/*----- Half-line aux Lignes -----*/
        for(l=0;l<8;l++)
            for(g=0;g<8;g++)
                {
                    for(i=0;i<nb;i++) y[l][g][i] = 0;
                    for(j=0;j<8;j++) conversion(nbp);
                    Halfline(nbp);
                }
/*----- Transposition -----*/
        for(l=0;l<8;l++)
            for(g=0;g<8;g++)
                for(i=0;i<nbp;i++) x[l][g][i] = y[g][l][i];
/*----- Half-line aux colonnes -----*/
        Ermoy = 0.0;
        for(l=0;l<8;l++)
            {
                for(g=0;g<8;g++)
                    {
                        Halfline(nbp);
                        Yc[l][g] = Z * pow(2, p + 3);
                        Er[l][g] = fabs( TCD(l,g) - Yc[l][g] ) ;
                    }
            }
/*----- Calcul des erreurs -----*/
        Ermoy=0.0;
        for(m=0;m<8;m++)
            {
                for(n=0;n<8;n++) Ermoy +=Er[m][n];
            }
        Ermoy =Ermoy/64;
        if ( Epmoy <= Ermoy) Epmoy=Ermoy;
        Emoyt += Ermoy;
        Erqua=0.0;
        for(m=0;m<8;m++)
            {
                for(n=0;n<8;n++)
                    Erqua +=pow((Ermoy-Er[m][n]),2.0);
            }
        Erqua=Erqua/64;
        if ( Epqua <= Erqua) Epqua=Erqua;
        Eqmoyt += Erqua;
            /*----- Fin boucle (blocs) -----*/
/*-----Erreurs totales-----*/
        Emoyt =Emoyt/T;
        Eqmoyt =Eqmoyt/T;
    }
/*----- Fin -----*/

```

```

/*-----!
!           CALCUL DE LA TCD PAR L'ARITHMETIQUE DISTRIBUEE           !
!-----*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#define PI 3.141592654
#define NM 256
typedef float Vect[8][8];
signed char X[8][8];
int x[8][8][20],i,j,k,L,L1,L2,l,m,n;
Vect Xf,Yr;

/*-----Fonctions utilisées-----*/
int conv_C2(int i,int j,int L);
float C(int l);
float TCD_2D(int m,int n);

/*-----Fonction de conversion -----*/
int conv_C2(int i,int j,int L)
{
    int M,K;
    for(k=0;k<L;k++) x[i][j][k]=0;
    M=fabs(Xf[i][j]);
    k=L-1;
    while(M!=0)
    {
        x[i][j][k]=M%2;
        M=M/2;
        k--;
    }
    if(Xf[i][j]<0)
    for(k=L-1;k>=0;k--)
    {
        if(x[i][j][k]==1)
        {
            for(K=k-1;K>=0;K--)
                x[i][j][K]=!x[i][j][K];
            goto suite;
        }
        else continue;
    }
    suite: return(x[i][j][L]);
}

/*-----Fonction TCD 2-D réelle-----*/
float C(int l)
{
    float w;
    if(l==0) w = 1.0/sqrt(2.0);
    else w = 1.0;
    return(w);
}
float TCD_2D(int m, int n)
{
    float s=0.0;
    for(i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
            s +=(X[i][j] *cos((2*j+1)*PI*n/16)) *cos((2*i+1)*PI*m/16);
    }
}

```

```

    Yr[m][n] = s *0.0625 *C(m)*C(n);
    return(Yr[m][n]);
}
/*-----Programme Principal-----*/
void main()
{
    char rep;
    int Af[8][8], LA, t, T, max;
    Vect Ylc, Yc, Er;
    float Ylcmax, R[20], som, Ermoy, Emoyt, Epmoy, Erqua, Eqmoyt, Epqua,

/*----- Matrice TCD 1-D radix 8 (normalisée) -----*/

    A[8][8]={ { 0.176776695, 0.176776695, 0.176776695, 0.176776695,
                0.176776695, 0.176776695, 0.176776695, 0.176776695 },
              { 0.245196320, 0.207867403, 0.138892558, 0.048772580,
                -0.048772580, -0.138892558, -0.207867403, -0.245196320 },
              { 0.230969883, 0.095670858, -0.095670858, -0.230969883,
                -0.230969883, -0.095670858, 0.095670858, 0.230969883 },
              { 0.207867403, -0.048772580, -0.245196320, -0.138892558,
                0.138892558, 0.245196320, 0.048772580, -0.207867403 },
              { 0.176776695, -0.176776695, -0.176776695, 0.176776695,
                0.176776695, -0.176776695, -0.176776695, 0.176776695 },
              { 0.138892558, -0.245196320, 0.048772580, 0.207867403,
                -0.207867403, -0.048772580, 0.245196320, -0.138892558 },
              { 0.095670858, -0.230969883, 0.230969883, -0.095670858,
                -0.095670858, 0.230969883, -0.230969883, 0.095670858 },
              { 0.048772580, -0.138892558, 0.207867403, -0.245196320,
                0.245196320, -0.207867403, 0.138892558, -0.048772580 } };

    clrscr();
    printf("\n");
    printf("          Nombre de blocs : T=");
    scanf("%d",&T);
    printf("          Taille des noyaux Cosinus : ");
    scanf("%d",&LA);
    for (i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
        {
            A[i][j]=A[i][j]*pow(2.0, LA);
            Af[i][j]=A[i][j];
            A[i][j]=Af[i][j]*pow(2.0, 0-LA);
        }
    }

/*----- Initialisation des erreurs -----*/
    Emoyt=0.0;
    Epmoy=0.0;
    Eqmoyt=0.0;
    Epqua=0.0;

    for(t=0;t<T;t++)          /* début-boucle(Blocs)*/
    {

/*----- Génération + Seuillage + Conversion -----*/
        for (i=0;i<8;i++)
        {
            for(j=0;j<8;j++)
            {
                X[i][j]=random(NM)-128;
                Ll=9;
                Xf[i][j]= (X[i][j]/128.000001)*pow(2.0,Ll-1.0);
                conv_C2(i,j,Ll);
            }
        }
    }
}

```

```

        A[i][j]=Af[i][j]*pow(2.0,0-nba);
    }
    printf("\n");          */
for(t=0;t<T;t++)
    {
        printf("Bloc n°:%d \n",t);
/*----- Initialisation des erreurs -----*/
        Emoyt=0.0;
        Epmoy=0.0;
        Eqmoyt=0.0;
        Epqua=0.0;
/*----- g,n,ratation des pixels -----*/
        for(j=0;j<8;j++) generation();

/*----- Half-line aux Lignes -----*/
        for(l=0;l<8;l++)
            for(g=0;g<8;g++)
                {
                    for(i=0;i<nb;i++) y[l][g][i] = 0;
                    for(j=0;j<8;j++) conversion(nbp);
                    Halfline(nbp);
                }
/*----- Transposition -----*/
        for(l=0;l<8;l++)
            for(g=0;g<8;g++)
                for(i=0;i<nbp;i++) x[l][g][i] = y[g][l][i];
/*----- Half-line aux colonnes -----*/
        Ermoy = 0.0;
        for(l=0;l<8;l++)
            {
                for(g=0;g<8;g++)
                    {
                        Halfline(nbp);
                        Yc[l][g] = Z * pow(2, p + 3);
                        Er[l][g] = fabs( TCD(l,g) - Yc[l][g] ) ;
                    }
            }
/*----- Calcul des erreurs -----*/
        Ermoy=0.0;
        for(m=0;m<8;m++)
            {
                for(n=0;n<8;n++) Ermoy +=Er[m][n];
            }
        Ermoy =Ermoy/64;
        if ( Epmoy <= Ermoy) Epmoy=Ermoy;
        Emoyt += Ermoy;
        Erqua=0.0;
        for(m=0;m<8;m++)
            {
                for(n=0;n<8;n++)
                    Erqua +=pow((Ermoy-Er[m][n]),2.0);
            }
        Erqua=Erqua/64;
        if ( Epqua <= Erqua) Epqua=Erqua;
        Eqmoyt += Erqua;
            }
/*----- Fin boucle (blocs) -----*/
/*-----Erreurs totales-----*/
        Emoyt =Emoyt/T;
        Eqmoyt =Eqmoyt/T;
    }
/*----- Fin -----*/

```

```

/*-----*
!          CALCUL DE LA TCD-radix_8 PAR LE MODE HALF-LINE          !
!-----*/

#include<stdio.h>
#include<stdio.h>
#include<math.h>
#include<conio.h>
unsigned char X[8][8];
int y[8][8][16],x[8][8][16],i,j,k,l,m,n,g,nb,nbp,p = 3;
float Z,s,Yr[8][8],yc[8][8],pi = 3.141592654,

/*-----Bloc de transformation radix 8 -----*/

      □
A[8][8]={{ 0.176776695, 0.176776695, 0.176776695, 0.176776695,
           0.176776695, 0.176776695, 0.176776695, 0.176776695},
 { 0.245196320, 0.207867403, 0.138892558, 0.048772581,
   -0.048772581,-0.138892558,-0.207867403,-0.245196320},
 { 0.230969883, 0.095670858,-0.095670858,-0.230969883,
   -0.230969883,-0.095670858, 0.095670858, 0.230969883},
 { 0.207867403,-0.048772581,-0.245196320,-0.138892558,
   0.138892558, 0.245196320, 0.048772581,-0.207867403},
 { 0.176776695,-0.176776695,-0.176776695, 0.176776695,
   0.176776695,-0.176776695,-0.176776695, 0.176776695},
 { 0.138892558,-0.245196320, 0.048772581, 0.207867403,
   -0.207867403,-0.048772581, 0.245196320,-0.138892558},
 { 0.095670858,-0.230969883, 0.230969883,-0.095670858,
   -0.095670858, 0.230969883,-0.230969883, 0.095670858},
 { 0.048772581,-0.138892558, 0.207867403,-0.245196320,
   0.245196320,-0.207867403, 0.138892558,-0.048772581}};

/*----- Fonctions prototypes -----*/
void generation(void);
void conversion(int nb);
void Halfline(int nb);
float C(int n);
float TCD(int l,int g);

/*----- Fonction de generation des entrées -----*/
      □
void generation(void)
{
  for(g=0;g<8;g++) X[j][g] = rand();
}

/*----- Fonction de conversion (b=2)-----*/
void conversion(int nb)
{
  int m;
  for(i=0;i<nb;i++) x[j][g][i] = 0;
  m = X[j][g];
  i = nb-1;
  while(m != 0)
  {
    x[j][g][i] = m % 2;
    m = m / 2;
    i--;
  }
}

/*----- Procedure Half-Line -----*/
void Halfline(int nb)
{

```

```

Yr[m][n] = s * 0.0625 * C(m) * C(n);
return(Yr[m][n]);
}
/*-----Programme Principal-----*/
void main()
{
char rep;
int Af[8][8], LA, t, T, max;
Vect Ylc, Yc, Er;
float Ylcmax, R[20], som, Ermoy, Emoyt, Epmoy, Erqua, Eqmoyt, Epqua,

/*----- Matrice TCD 1-D radix 8 (normalisée) -----*/
A[8][8]={{ 0.176776695, 0.176776695, 0.176776695, 0.176776695,
0.176776695, 0.176776695, 0.176776695, 0.176776695 },
{ 0.245196320, 0.207867403, 0.138892558, 0.048772580,
-0.048772580, -0.138892558, -0.207867403, -0.245196320 },
{ 0.230969883, 0.095670858, -0.095670858, -0.230969883,
-0.230969883, -0.095670858, 0.095670858, 0.230969883 },
{ 0.207867403, -0.048772580, -0.245196320, -0.138892558,
0.138892558, 0.245196320, 0.048772580, -0.207867403 },
{ 0.176776695, -0.176776695, -0.176776695, 0.176776695,
0.176776695, -0.176776695, -0.176776695, 0.176776695 },
{ 0.138892558, -0.245196320, 0.048772580, 0.207867403,
-0.207867403, -0.048772580, 0.245196320, -0.138892558 },
{ 0.095670858, -0.230969883, 0.230969883, -0.095670858,
-0.095670858, 0.230969883, -0.230969883, 0.095670858 },
{ 0.048772580, -0.138892558, 0.207867403, -0.245196320,
0.245196320, -0.207867403, 0.138892558, -0.048772580 }};

clrscr();
printf("\n");
printf("                                Nombre de blocs : T=");
scanf("%d", &T);
printf("                                Taille des noyaux Cosinus : ");
scanf("%d", &LA);
for (i=0; i<8; i++)
{
for (j=0; j<8; j++)
{
A[i][j]=A[i][j]*pow(2.0, LA);
Af[i][j]=A[i][j];
A[i][j]=Af[i][j]*pow(2.0, 0-LA);
}
}

/*----- Initialisation des erreurs -----*/
Emoyt=0.0;
Epmoy=0.0;
Eqmoyt=0.0;
Epqua=0.0;

for(t=0; t<T; t++) /* début-boucle(Blocs)*/
{
/*----- Génération + Seuillage + Conversion -----*/
for (i=0; i<8; i++)
{
for (j=0; j<8; j++)
{
X[i][j]=random(NM)-128;
L1=9;
Xf[i][j]= (X[i][j]/128.000001)*pow(2.0, L1-1.0);
conv_C2(i, j, L1);
}
}
}

```

```

/*----- Appel de la fonction TCD_2D-----*/
for(m=0;m<8;m++)
{
for (n=0;n<8;n++)
{
Yr[m][n]=0.0;
TCD_2D(m,n);
}
}
/*----- Initialisation des calculs -----*/
for (n=0;n<8;n++)
{
for (m=0;m<8;m++)
{
Ylc[m][n]=0.0;
Yc[m][n]=0.0;
Er[m][n]=0.0;
}
}
/*-----Calcul sur les lignes -----*/
Ylcmax=0.0;
max=0;
for (n=0;n<8;n++)
{
for (m=0;m<8;m++)
{
for (k=0;k<L1;k++) R[k]=0.0;
for(j=0;j<8;j++)
R[0] +=-(x[n][j][0]*A[m][j]);
for(k=1;k<L1;k++)
{
som=0.0;
for(j=0;j<8;j++)
som +=(x[n][j][k]*A[m][j]);
R[k] = 2*R[k-1]+som;
}
Ylc[m][n]=(R[L1-1]*128.000001)*pow(2.0,1-L1);
if (Ylcmax <= (fabs(Ylc[m][n])) ) Ylcmax=fabs(Ylc[m][n]);
}
}
max=floor(Ylcmax)+1;
/*----- Transposition des Ylc[m][n] + Conversion -----*/
for(i=0;i<8;i++)
{
for(j=0;j<8;j++)
{
L2=12;
Xf[i][j] = (Ylc[j][i]/max)*pow(2.0,L2-1);
conv_C2(i,j,L2);
}
}
/*-----Calcul sur les colonnes-----*/
for (n=0;n<8;n++)
{
for (m=0;m<8;m++)
{
for (k=0;k<L2;k++) R[k]=0.0;
for(j=0;j<8;j++)
R[0] +=-(x[j][n][0]*A[m][j]);
for(k=1;k<L2;k++)
{
som=0.0;

```

```

        for(j=0;j<8;j++)
        som +=(x[j][n][k]*A[m][j]);
        R[k] = 2*R[k-1]+som;
    }
    Yc[m][n]=(R[L2-1]*max)*pow(2.0,1-L2);
}
}

/*-----Calcul des erreurs-----*/
Ermoy=0.0;
for(m=0;m<8;m++)
{
    for(n=0;n<8;n++)
    {
        Er[m][n]= fabs(Yc[m][n]-Yr[m][n]);
        Ermoy +=Er[m][n];
    }
}
Ermoy =Ermoy/64;
if ( Epmoy <= Ermoy) Epmoy=Ermoy;
Emoyt += Ermoy;

Erqua=0.0;
for(m=0;m<8;m++)
{
    for(n=0;n<8;n++)
        Erqua +=pow((Ermoy-Er[m][n]),2.0);
}
Erqua=Erqua/64;
if ( Epqua <= Erqua) Epqua=Erqua;
Eqmoyt += Erqua;

}          /* Fin boucle (blocs) */

/*-----Erreurs totales-----*/
Emoyt = Emoyt/T;
Eqmoyt = Eqmoyt/T;

/* ----- Fin -----*/
}

```





# BIBLIOGRAPHIE

- [1] : C.Piquet et A. Stauffer, « Synthèse de circuits ASIC : Des choix méthodologiques aux applications industrielles », Dunod, 1990.
- [2] : D.D. Gajski, « Silicon Compilation », Addison-Wesley Publishing Company Inc., 1988.
- [3] : N. Ahmed, T. Natarajan and K. R. Rao, « Discrete Cosine Transform », IEEE Trans. Comm., Vol. COM 23, pp. 90- 93, Jan 1974.
- [4] : A. N. Netravali et al., « Picture Coding : A review », Proceeding of the IEEE, Vol.68, N°3, pp. 366- 407, March 1980.
- [5] : N. Gamaz, « Etude et développement d'algorithmes de compression d'images fixes par TCD 2-D », Thèse de Magister C.D.T.A , 1992.
- [6] : J. W. Modestino and al., « Combined Source-Channel Coding of images », IEEE Trans. on Commun., Vol. 27, N°11, pp. 1644-1659 , November 1979.
- [7] : A. Spataru, « Fondements de la théorie de la transmission de l'information », Presses Polytechniques Romandes, Lausanne, Suisse, 1987.
- [8] : P. Wintz, « Transform Picture Coding », Proceeding of the IEEE, Vol.60, N°7, pp. 809-820, July 1972.
- [9] : M. Abdat et M. Bellanger, « Standardisation Multimedia », Proceeding du 1<sup>ier</sup> séminaire national SSA'92 , Blida, Algérie, Vol.2, pp.352-372, 13-15 Décembre 1992.
- [10] : M. Kunt, « Traitement numérique des images », Collection électricité : Traitement de l'information - Vol.2, Presses Polytechniques Romandes, Lausanne, Suisse, 1993.
- [11] : D.F.Elliot & K.Ramamohan Rao, « Fast Transforms, Analysis, Applications », Academic Press, Inc 82.
- [12] : A. Habibi, « Hybrid Coding of Pictorial Data », IEEE Trans. on Commun., Vol. 22, N°5, pp. 614-624, May 1974.
- [13] : J. A. Roese, « Interframe Cosine Transform Image Coding », IEEE Trans. on Commun., Vol. 25, N°11, pp. 1329-1339, November 1977.
- [14] : N. M. Nasrabadi and R. A. King, « Image Coding Using Vector Quantization : A review », IEEE Trans Comm., Vol. 36, N°8, pp. 957- 971, August 1988.
- [15] : B. Ramamurthi and A. Gersho, « Classified Vector Quantization Of Images », IEEE Trans Comm., Vol. 34, N°11, pp. 1105-1115 , November 1986.

- [16] : Y. Linde, A. Buzo and R. Gray, « An Algorithm for Vector Quantizer Design », IEEE Trans Commun., Vol. 28, N°1, pp. 84-95 , January 1980.
- [17] : E. J. Delp and al., « Image Compression Using Truncation Coding », IEEE Trans Commun., Vol. 27, N°9, pp. 1335-1342 , September 1979.
- [18] : M. Kunt, « Traitement numerique des signaux », Edition Dunod, Lausanne, Suisse, 1987.
- [19] : T. S. Hung, « Coding of two Tone Images », IEEE Trans Commun., Vol. 25, N°11, pp. 1406-1424, November 1977.
- [20] : B. Revillet, « Télécopie », Techniques de l'ingénieur, E7830, 1992.
- [21] : C. B. Jones, « An Efficient Coding System for Long Sources Sequences », IEEE Trans. on Information Theory, Vol. 27, N°3, pp. 297-305, March 1981.
- [22] : G. K. Wallace, «The J.P.E.G Still Picture Compression Standard», Communications of the ACM, Vol.34, N°4, pp. 30-44, April 1991
- [23] : CCITT, « Specialing Group on Coding for Visual Telephony »
- [24] : J. C. Jolivet, « Visiophone », Techniques de l'ingénieur, E7820, 1992.
- [25] : D. Le Gall, « M.P.E.G : A Video Compression Standard for Multimedia Applications », Communications of the ACM, Vol.34, N°4, pp. 46-58, April 1991.
- [26] : J. Makhoul, « A Fast cosine transform in one and two dimensions », IEEE Trans ASSP, Vol. 28, pp. 27-34, Feb. 1980.
- [27] : F.A.Kamangar & K.R. Rao, « Fast algorithms for the 2-D discrete cosine transform », IEEE trans. Comput., Vol. C-31, pp.899-906, Sep 1982.
- [28] : S. Uramoto, Y. Inoue and Co « A 100 MHZ 2-D DCT Core Processor », IEEE journal of solid State Circuits, Vol 27, N°4, April 92.
- [29] : H. Bessalah, « Evaluation des transformées Orthogonales par l'arithmétique On-line et Half-line », in *Proc. CARI'95*, 15- 18 oct, Ouagadougou 1994.
- [30] : M.A. Haque, « A two dimentional fast cosine transform », IEEE trans. ASSP, Vol.33, pp. 1532-1539, Dec. 1985.
- [31] : C. Ma , « A Fast recursive two dimentional cosine transform », Intelligent Robots and Computer Vision : Seventh in series, David P. Casasent, Ed., in *Proc. SPIE 1002*, pp. 541-548, 1988.
- [32] : M. Vetterli, « Fast 2-D Discrete Cosine Transform », in *Proc. ICASSP ' 85*, Mar. 1985.

- [33] : P. Duhamel et C. Guillemot, «Polynomial Transform Computation of the 2-D DCT», Proc. ICASSP ' 90, pp.1515-1518, Apr. 1990.
- [34] : N.I. Cho & S.U. Lee, « Fast Algorithm and implementation of 2-D DCT », IEEE Trans. on Circuits and Systems, vol.38, N°3, Mars 91.
- [35] : N.I. Cho, I.D. Yun & S.U. Lee, « On the regular structure for the fast 2-D DCT algorithm», IEEE Trans. On circuits and systems-II : Analog and Digital signal Processing, Vol.40, N°4, pp. 259-266, Apr. 1993.
- [36] : H. Bessalah, « Multidimensional pipeline implementation for fast orthogonal transform On-line computation », Thèse PHD, Institut Polytechnique de Kiev, 1981.
- [37] : H. S. Hou « A Fast Recursive Algorithm for Computing the DCT », IEEE Trans. ASSP., Vol.35, pp 1455-1461,-Oct 87.
- [38] : W. H. Chen, H. Shith & S. C. Fralick, « A Fast Computational Algorithm for the DCT » , IEEE Trans on Comm, Vol 25, N°4, Sept 77.
- [39] : M. Vetterli, H. Nussbaumer « Simple FFT and DCT algorithms with reduced number of operation », Signal Process, Vol.6, pp. 267-278, Aug. 1984.
- [40] : J.I. Guo, C.M. Liu and C.W. Jen, « The efficient memory-based VLSI array designs for DFT and DCT », IEEE Trans. On circuits and systems-II : Analog and Digital signal Processing, Vol.39, N°10, pp. 723-733, Oct. 1992.
- [41] : M. J. Narasimha and A.M. Peterson , « On the computation of the discrete cosine transform », IEEE Trans. On commun., Vol.26, N°6, pp. 934-936, June 1978.
- [42] : E.M. Ait Nouri et H. Bessalah « Conception et contribution à la réalisation d'un système de codage d'images par transformées », Thèse de PFE, Institut d'Electronique, USTHB, Bab Ezzouar, Juillet 92.
- [43] : P. Duhamel, « Algorithmes rapides et architectures de traitement du signal », GDR 134, Traitement du signal et images, Lannion 14-15 Sept. 1992.
- [44] : Jean Michel Muller, «Arithmétique des ordinateurs, Opérateurs et fonctions élémentaires », édition Masson, 1989.
- [45] : Y. Kusumaputri, «Opérateurs Arithmétiques Standards En-ligne à Très Grande Précision », Thèse de Doctorat, INPG, Mai 93.
- [46] : E. M. Aït Nouri, « Modularité Systématique par Concaténation des Algorithmes On-line pour le Calcul en Grande Précision », Proc MCEA'95, Grenoble, Septembre 95.

- [47] : R. Bouraoui, « Calcul sur les Grands Nombres et VLSI : Application au PGCD Etendu à la Distance Euclidienne », Thèse de Doctorat, INPG, janvier 1993.
- [48] : A. Avezienis, « Signed-Digit Number Representations for Fast Parallel Arithmetic », IRE trans. Electron. Compt, vol. EC-10, pp.389-400, sept 1961.
- [49] : M.D Ercegovac et K.S. Trivedi, « On line algorithms for division and multiplication », IEEE Transactions on Computers, Vol.C-26, pages 681-687, Juillet 1977.
- [50]: M.D Ercegovac, « On line Arithmetic : An Overview », Proc. SPIE Conference on real time signal processing, 1984.
- [51] : Y. B. Yu Chan et A. G. Constantinides, « Variable size block matching motion compensation with application to video coding », IEE PROC., Vol. 137. Pt. I, N° 4, pp. 205-211, 1990.
- [52] : H. Bessalah, « Multidimensional Pipeline and Array Architectures for On-line Computation of fast Orthogonal Transforms », LYON: Lip, ENS, 1993
- [53] : N.Zerkane, M. Morceli, « Conception d'une Bibliothèque de Cellules de l'Algorithme On-line et Half-line Appliqués aux Opérations de Traitement du Signal », PFE, Inst. d'Elect., USTHB, Bab Ezzouar, sept 95.
- [54] : M.Tounsi et A. Farah, « Conception d'un ASIC TCD/TCDI 2-D en vue de la compression d'images par l'application de l'arithmétique distribuée et des modes de calcul On-line et Half-line », Proceeding of the IEEE International Annual Conference, University of Batna, Vol.2, pp.64-69, 7-9 Dec. 97.
- [55] : Kai Hwang, «Computer Arithmetic : Principles, Architectures and Design », 1979, édition John Wiley and Sons.
- [56] : Stanley A.White, « Application of distributed Arithmetic to Digital Signal Processing : A Tutorial Review », IEEE ASSP Magazine, Juillet 1989.
- [57] : A.Peled, B.Liu, « A New Hardware Realisation of Digital Filters », IEEE Trans. on ASSP-22, pp456-462, Decembre 1974.
- [58] : A. Croisier, D.J.Esteban, M.E.Levilion et V.Rizo, « Digital Filters for PCM encoded signals », U.S Patent, Décembre 1973.
- [59] : S.Zohar, « New Hardware Realisation of Nonrecursive Digital Filters », IEEE Trans.on Computers, Vol. C-22, pp.328-338, Avril 1973.
- [60] : G De Micheli, « Synthesis and optimization of digital circuits », Mc Graw Hill, 1994

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

DÉPARTEMENT : ELECTRONIQUE

OPTION : TELECOMMUNICATIONS

RESUME DU MEMOIRE DE MAGISTER

**THEME**

CONCEPTION D'ARCHITECTURES DIGITALES UNIFIÉES TCD/TCDI 2-D :  
ETUDE COMPARATIVE DU MODE DE CALCUL HALF-LINE  
ET DE L'ARITHMÉTIQUE DISTRIBUÉE

Présenté par :

Mr TOUNSI Mohamed

Ingénieur d'état, ENP.

Sous la direction de :

Mr FARAH Ahcène

Professeur, ENP.

**ملخص**

إن استعمال محاولة تحب المنفصلة (TCD) بشفرة الصور الرقمية في وقتنا الحاضر، مقرر في مختلف القواعد الدولية . اهدف هو إيجاد تطبيقات وخدمات مثل المحاضرة-البصرية و HDTV بحيث لا يمكن تحقيق ذلك إلا بترجع TCD و تحويلها العكسية في الدارات المتكاملة (ASICs) نظرا لاحتلالها الطبقات الأكثر عملا. بالرغم من أن معظم حساباتها تستنتج بالدمج التسلسلي، فإن دمجها الحسي أصبح هو العنصر المفتاح في VLSI لضغط الصور.

سنتين في هذا البحث أن هناك خفة عالية في التوافق بين الحسابات و الأشكال بحيث تبقى دائمة باستعمال درامة مقارنة و أدوات حسابية فعالة كالحساب الموزع و طريقة الحساب بالخط "ON LINE" التي تسمح باشتقاق الحساب المهجين من أحل تكوين شكل موحد

**RÉSUMÉ**

L'utilisation de la transformée cosinus discrète (T.C.D) en codage d'images numériques, est de nos jours adoptée par les diverses normes internationales. L'objectif d'offrir des applications et des services tels que la visioconférence et la HDTV, reporté sur le codec, ne pourrait être atteint que par l'implantation de la TCD et sa transformée inverse sur des circuits intégrés dédiés (ASICs), vu qu'elles occupent les blocs les plus opérationnels. Quoique la majorité de leurs algorithmes, sont dédiés à une implémentation software, leur implémentation hardware est devenue l'élément-clé en VLSI pour la compression d'image. Dans notre travail de thèse, nous montrons avec une étude comparative, qu'une grande souplesse d'adaptation algorithmes-architectures existe avec l'utilisation d'outils arithmétiques efficaces tels l'arithmétique distribuée et le mode de calcul half-line qui permettent de dériver un algorithme hybride TCD-TCDI 2-D pour la conception d'une architecture unifiée.

**ABSTRACT**

Transform coding utilising the discrete cosine transform (DCT) has been adopted in various standards of image compression. As it is the most operative bloc of the encoder's range, the modern digital image systems require fast DCT computational algorithms, especially dedicated to VLSI implementation. In this work, we show that an adequation algorithms-architectures exists and we present the derivation and the choice of an hybrid algorithm DCT-IDCT 2-D, based on the distributed arithmetic and the half-line mode of computation, in order to obtain a regular unified architecture TCD-TCDI 2-D.