.

UNIVERSITE SAAD DAHLEB DE BLIDA

Faculté des sciences de l'ingénieur

Département d'Electronique

MEMOIRE DE MAGISTER

Spécialité : SIGNAUX & SYSTEMES

IMPLEMENTATION SUR FPGA D'ALGORITHME EN LIGNE D'ANALYSE SPECTRALE ET DE FILTRAGE NUMERIQUE

Par

FERHAT TALEB née ALIM FATIHA

Devant le jury composé de :

A. Guessoum	Professeur, USDB	Président
K. Achour	Directeur de recherche, CDTA	Examinateur
B. kazed	Chargé de cours, USDB	Examinateur
H. Bessalah	Maître de recherche, CDTA	Rapporteur
H. Salhi	Maître de conférences, USDB	Co Rapporteur

Blida, Septembre 2005

RESUME

Dans ce travail, nous avons développé une architecture multidimensionnelle de la transformée en ondelettes en utilisant l'algorithme de décomposition et de reconstruction de Stéphane Mallat. Le processus de calcul de cet algorithme est caractérisé par une structure purement séquentielle. Dans cette perspective, le mode de calcul en ligne s'avère plus approprié pour la réalisation de cette structure. Cette étude est consacrée à l'introduction de l'arithmétique en ligne de la transformée en ondelettes et au développement de l'architecture correspondante en utilisant un calcul pipeline au lieu d'un calcul conventionnel et implémentation sur FPGA.

ABSTRACT

In this work, we have developed multidimensionnal architecture for the wavelet transform bay using the Stéphane Mallat decomposition reconstruction Algorithm. The computation procedure of this Algorithm is characterized by a purely sequential structure. With this propriety, the on line mode seems to be more appropriate to the realisation of this structure. In our study, fist we introduce the on line arithmetic to the wavelet transform. Second, the corresponding architecture is developed and implemented on FPGA by using a pipelined architecture instead of a conventional computation.

ملخص

قد قمنا في هذا العمل بتطوير بنية متعددة الأبعاد لتحويله ذات موجات صغيرة باستخدام خوارزم Stéphane Mallat للتفكيك و إعادة التركيب. تتميز عملية حساب هذا الخوار زم ببنية تتابعيه بحتة. و في هذا الإطار يتبين أن طريقة الحساب التسلسلي أكثر ملائمة لإنجاز هذه البنية. و تخصص هذه الدراسة لإدخال الحساب التسلسلي للتحويلية ذات الموجات الصغيرة و كذا تطوير البنية الموافقة باستخدام حساب متتابع بدلا من الحساب التقليدي و التركيب على FPGA .

DEDICACES

Je dédie ce travail à mon fils NABIL ABD EL AZIZ, à qui j'ai pris de son temps pour accomplir ce travail, qu'il trouve tout l'amour du monde d'une maman pour son fils.

Je n'oublierais jamais les moments où il me disait « mama reste avec moi je n'aime pas ton magister »

Surtout, je voudrais remercier mes parents, qui m'ont toujours soutenu (et supporté!); sans trop savoir jusqu'où tout cela pourrait mener.

Je remercie infiniment mon mari pour m'avoir épaulé et encouragé au cours de ces années.

A mes frères et sœurs, en particulier Raouf et Adim.

Je vous dis tous : « Mille merci, mille merci! mille merci!, et que Dieu vous protége ».

REMERCIEMENTS

Je remercie Dieu le tout puissant qui m'a donné le courage et la patience afin de mener bien ce projet.

J'adresse mes sentiments respectueux et reconnaissants au directeur du Centre de Développement des Technologie Avancées (CDTA) Monsieur H.BESSALAH premièrement, de m'avoir lancer dans la voie de la recherche; et je tiens à le remercier comme promoteur pour m'avoir proposé ce sujet de recherche d'avoir été toujours disponible et d'avoir effectuer ce travail dans de bonne conditions.

Mes très sincères remerciements à mon co promoteur H.SALHI Maître de conférences à l'université de Blida et mon enseignant, pour sa contribution dans le projet et pour ses judicieux conseils durant mes études d'ingénieur et de Magister.

J'adresse également mes remerciements à Monsieur A.GUESSOUM, professeur à l'université de Blida pour l'honneur qu'il m'a fait en présidant ce jury, ainsi que Monsieur K.ACHOUR directeur de recherche au CDTA aussi Monsieur B.KAZED, membres du jury qui ont bien voulu prendre le temps de s'intéresser au sujet du présent mémoire et ont contribué à en améliorer la qualité.

Je tiens à exprimer ma gratitude à mes amis : S.Seddiki, W.Benzaba, M.Bencherif, F.Abddat pour leurs bonne compagnie et pour l'ambiance de travail agréable qu'ils ont su créer et grâce à leur amitié et leur serviabilité; la difficulté du travail de recherche était surmontable.

Je tiens à exprimer ma gratitude à Monsieur M. Anane pour ses encouragements, ses conseils pertinents sans cesse renouvelés qui ont contribué à l'avancement de ce travail.

Je remercie vivement Monsieur O.Djekoune invité à la pré soutenance ainsi que H.Hamou pour le temps et le travail qu'a nécessité leurs lecture attentive.

Un grand merci à mes collègues de l'équipe de recherche A3SP et mes collègues du CDTA

J'adresse mes très sincères remerciements au responsable de la post-graduation Professeur O.Najmi et mon enseignent pour sa modestie et ces compétences.

A tous les esprits scientifiques qui croient que le savoir est universel en nous guidant et en nous simplifiant la recherche scientifique.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

Le domaine du traitement de l'image connaît une progression importante liée à l'évolution des technologies de l'information et de la communication. Il fait appel à un ensemble de théories et de méthodes, permettant d'analyser avec soin, de coder efficacement, de transmettre rapidement et enfin de reconstruire soigneusement à la réception les signaux transmis à l'origine. Souvent, ces images brutes ne peuvent être étudiées telles quelles se présentent et doivent subir une étape de transformation nécessaire à l'augmentation de l'efficacité du traitement. Cette étape de transformation consistera, notamment à extraire l'information nécessaire pour les applications de reconnaissance de motifs, de détection de contours ou de segmentation, ou à représenter l'information de manière plus compacte pour la compression.

Les outils d'analyse spectrale et de filtrage numérique constituent des composantes fondamentales du traitement du signal ; de nombreuses méthodes et transformées pour l'analyse existent, citons les transformées orthogonales, la transformée de Hough, …etc. Une technique qui a acquis une attention particulière et une reconnaissance grandissante dans la communauté scientifique, la transformée en ondelettes, la nouvelle norme JPEG2000 de compression; qui est considérée comme une évolution de la transformée de Fourier. De par ses qualités, associées à l'analyse multi résolution, elle permet l'observation et l'étude des objets présents dans l'image à différentes échelles. Elle est concrétisée par un grand nombre d'applications industrielles.

Les algorithmes d'analyse spectrale et de filtrage numérique ont un caractère paralléllo séquentiel. Les opérateurs parallèles de ces algorithmes autorisent des implémentations sur des machines parallèle du type SIMD (Single Instruction Multiple Data)ou MIMD (Multiple Instruction Multiple Data …) et permettent d'atteindre ainsi les performances requises en matière de vitesse et de précision .Quant aux opérations sériels, ils ne peuvent être exécutés que séquentiellement en utilisant des algorithmes séquentiels, cela réduit considérablement les performances globales.

Ainsi, le problème consiste à accélérer le processus de calcul de ces opérateurs sériels et à améliorer leurs performances en utilisant des architectures et des arithmétiques adéquates.

Par ailleurs, les algorithmes de traitement de signal sont basés sur des calculs répétitifs et utilisent un nombre réduit d'opérations simples d'un microprocesseur complet, donc ils n'ont pas besoin de toutes les capacités disponibles sur une machine universelle ; Alors, un circuit qui effectue les taches simples et répétitives d'un algorithme de traitement de signal sera moins complexe et plus rapide qu'un microprocesseur.

Le développement d'algorithmes fondés sur l'arithmétique en ligne s'avère très efficace pour le calcul d'une chaîne d'opérations arithmétiques sérielles , et permet de résoudre le problème de rapidité et d'engendrer des architectures pour une implémentation hardware, vue les nombreux avantages qu'elle procure du fait de la circulation des opérandes de manière sérielle bit plus fort en tête-MSB- (Most Significant Bit), ce qui favorise un traitement massif de l'information en utilisant le pipeline au niveau du bit et une meilleure précision, en particulier pour les opérations non parallélisables.

Les travaux de ce mémoire s'inscrivent dans le cadre des activités de l'équipe A3Sp du Centre de Développement des Technologie Avancées (CDTA). Cette étude consiste en l'utilisation de l'arithmétique en ligne pour l'execution de l'algorithme de la transformée en ondelettes de Stéphane Mallat en deux dimensions et au développement d'une nouvelle architecture multidimensionnelle, dans le cas de décomposition et de reconstruction pour le traitement des images et en particulier les images médicales.

Ce mémoire s'articule autour de quatre chapitres :

Dans le premier chapitre, nous ferons une introduction sommaire sur le traitement du signal et l'apparition de la transformée en ondelettes et nous insisterons particulièrement sur l'emploi de bancs de filtres ainsi que de l'algorithme de Stéphane Mallat, qui constitue l'une des techniques de la transformée en ondelettes la plus représentée dans le domaine de l'analyse et la synthèse des signaux. Dans le deuxième chapitre, nous aborderons l'arithmétique en ligne et ces particularités ainsi que les opérations de base (addition, multiplication et division). Le troisième chapitre sera

réservé à la présentation des différents types d'architectures relatives à l'implémentation de la transformée en ondelettes.

La méthodologie de conception sur FPGA de Xilinx et la description en VHDL ainsi que les résultats des simulations obtenues de notre architecture feront l'objet du quatrième chapitre .

Enfin, dans le dernier chapitre, nous donnerons une conclusion et quelques perspectives sur l'amélioration des travaux que nous avons abordé tout au long de ce mémoire.

CHAPITRE 1
DE LA TRANSFORMÉE DE FOURIER
A LA TRANSFORMÉE EN ONDELETTES

1.1. Introduction

Un problème très général en traitement du signal est celui posé par l'analyse, c'est l'extraction à partir des données d'un ensemble d'informations pertinentes qui rendent compte tant du contenu spectral du signal que de son organisation temporelle.

Il existe de nombreuses méthodes utilisées en traitement du signal. Les critères de sélection de l'une ou l'autre de ces méthodes peut dépendre de l'application envisagée; traitement de la parole, détection de contours,…etc.

Dans ce chapitre, nous présenterons tout d'abord la transformée de Fourier (TF), référence inévitable pour toute application ayant trait au domaine du traitement du signal, ainsi que ses limitations. Nous verrons également comment la transformée de Fourier à fenêtre glissante permet de réduire les inconvénients de la TF, sans pour autant posséder des caractéristiques optimales, notamment en terme d'analyse temps fréquence. Enfin, nous verrons pourquoi la transformée en ondelettes peut s'avérer supérieure aux deux méthodes précédentes lorsque l'analyse que l'on veut effectuer sur le signal doit prendre en compte à la fois le temps et la fréquence.

Par la suite, nous étudierons plus en détails la transformée en ondelettes dans le contexte de l'analyse multi résolution et nous présenterons les notions d'espaces d'approximation et de détail.

1.2. Transformée de Fourier

L'analyse par la transformée de Fourier reste l'outil fondamental pour le traitement du signal, c'est l'une des bases majeures de la physique et des mathématiques. Elle est indissociable du traitement de signal, et ce pour deux raisons principales : La première est l'universalité du concept de fréquence ou performance d'un point de vue distribution fréquentielle. La seconde tient à la structure même de l'analyse de Fourier qui se prête aisément à des transformations communes comme le filtrage linéaire en les traduisant de manière particulièrement simple [1].

Figure 1.1: Schéma de la transformée de Fourier [2]

Le principe de la transformée de Fourier repose sur le fait que toute fonction périodique peut être représentée comme la somme d'une série de sinus et de cosinus Figure 1.2 dont on fait varier d'une part les amplitudes, en les multipliant par des coefficients, et d'autre part les phases en les décalant de manière à ce qu'elles s'additionnent ou se compensent.

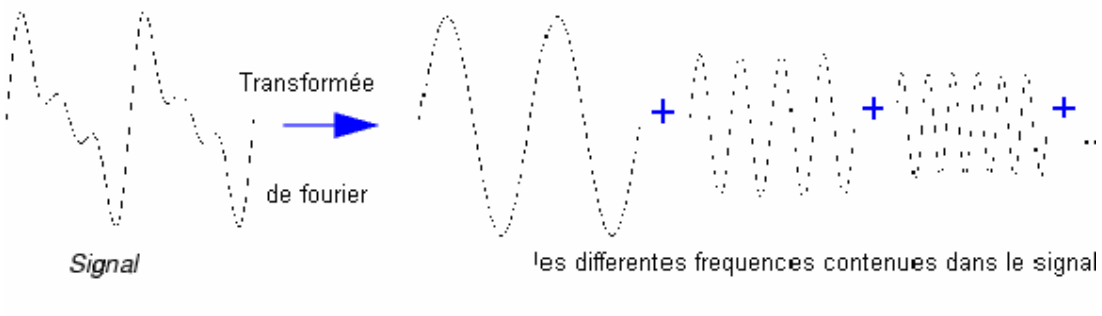


Figure 1.2: Principe de la transformée de Fourier

Cependant, l'analyse de Fourier, montre assez vite ses limitations. Son calcul nécessite la connaissance de toute l'histoire temporelle du signal, équation (1.1), transformée de Fourier, et équation (1.2), transformée de Fourier inverse [1].

Donc l'approche tant recherchée est là où les variations, ou fluctuations à différents moments et avec quelles amplitudes, se trouve inefficace pour l'étude des signaux transitoires ou évolutifs. Alors l'étude du signal est réduit soit en fonction du temps, soit en fonction des fréquences qu'il contient, sans possibilité de conjuguer les deux analyses.

$$X(w) = \int_{-\infty}^{+\infty} x(t) e^{-iwt}\, dt \qquad (1.1)$$

$$X(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} x(w) e^{iwt}\, dw \qquad (1.2)$$

Malgré le succès de la transformée de Fourier, elle reste peu adaptée à l'étude des signaux non stationnaires. Ces signaux se rencontrent beaucoup plus souvent que les signaux stationnaires. Pour leur étude, il faudra donc employer une autre méthode d'analyse qui tient compte de l'information temporelle et fréquentielle du signal.

1.3. La transformée de Fourier à fenêtre glissante (Gaborette)

En 1940, Denis GABOR introduit la transformée de Fourier à fenêtre glissante. L'approche était de fixer des fenêtres temporelles, qui sont translatées sur tout le domaine spectral du signal« fenêtre glissante », pour pallier le manque d'information sur le temps dans la transformée de Fourier.

Figure 1.3. Schéma de la transformée de Fourier à fenêtre glissante [2]

On définit une fenêtre qui sera utilisée comme masque sur le signal, et dans laquelle on considère que le signal est localement stationnaire, puis on décale cette fenêtre le long du signal afin de l'analyser entièrement.

La fenêtre est représentée par une fonction gaussienne g(x) [1] :

$$g_{a,b}(t) = e^{iat} g(t-b) \qquad (1.3)$$

Où *a* représente le facteur d'échelle, et *b* le facteur de translation.

On constate que le membre *g(t-b)* de l'équation (1.3) est indépendant de *a*, ce qui signifie que l'enveloppe de la fenêtre glissante sera constante ; on aura donc une résolution fixe sur toute la durée du signal.

Ainsi, l'étude d'un signal avec la transformée de Gabor permet d'obtenir à la fois une information sur le temps et sur la fréquence, mais la résolution d'analyse est fixée par le choix de la taille de l'enveloppe [1], si la fenêtre est trop petite, les basses fréquences n'y seront pas

contenues et si la fenêtre est trop grande, l'information sur les hautes fréquences sera noyée dans l'information concernant la totalité de l'intervalle contenu dans la fenêtre.

Cette méthode s'avéra donc figée car les variations peuvent se concentrer dans des régions temporelles alors que dans d'autres rien ne se passe, émergea alors le concept de variation d'échelle temporelle et fréquentielle.

1.4. Introduction à la transformée en ondelettes

La transformée de Fourier ainsi que ses dérivées ne sont donc pas toujours adaptées, du fait de leurs limitations, au traitement du signal et de l'image dans de nombreux cas de figures. En 1909, Alfred Haar fut le premier qui introduisit les ondelettes, c'est l'analyse «temps–échelle ».

En 1983, J.MORLET a proposé un procédé révolutionnaire, l'analyse et la synthèse des signaux par les ondelettes « analyse multi résolution » qui équivaut à une décomposition atomique où les atomes sont les ondelettes.

Figure 1.4 : Schéma de la transformée en ondelettes [2]

La méthode de Gabor permet donc de décomposer un signal donné en une combinaison linéaire temps - fréquence judicieusement choisie. Cependant on aimerait pouvoir faire varier la résolution d'analyse en fonction du signal afin de l'adapter à celui-ci. Un très haut niveau de résolution n'est pas nécessaire lorsque le signal est constitué uniquement de basses fréquences et il serait dommage d'étudier un signal comportant beaucoup de hautes fréquences avec une résolution trop basse. Ainsi, plutôt que de choisir à l'avance une résolution adaptée à un type de signal donné, il serait préférable de disposer d'une méthode d'analyse dont la résolution, aussi bien en temps qu'en fréquence, s'adapte au signal en fonction de ses caractéristiques. Pour ce faire, plutôt que de conserver une enveloppe fixe dans laquelle le nombre d'oscillations varie, on conserve un nombre

d'oscillations constant dans une enveloppe que l'on peut contracter et dilater à volonté : c'est l'analyse multi résolution, dont la figure 1.5 décrit le principe.

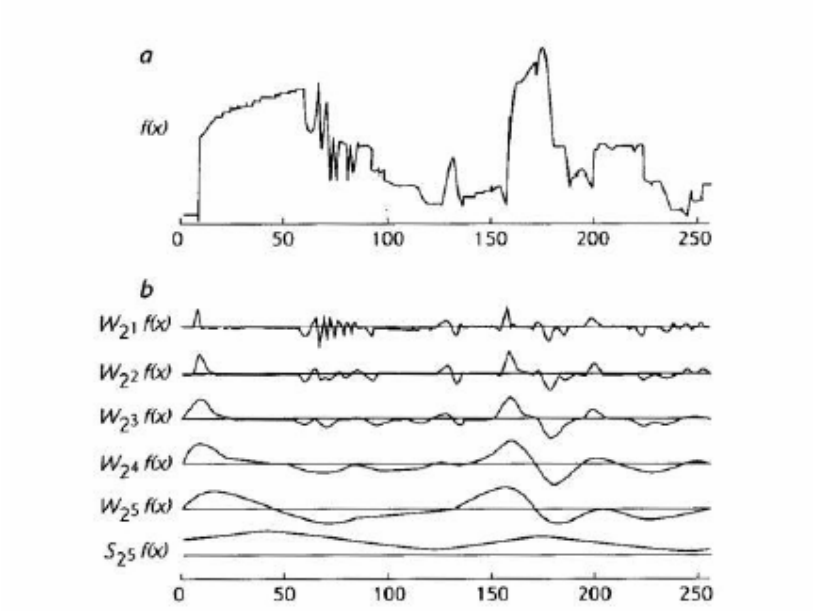


Figure 1.5 : Principe de l'analyse multi résolution [3]

Le signal d'origine (*a*) est décomposé en un ensemble de sous signaux (*b*) selon la procédure suivante :

La fonction f(x) est décomposée en un signal de détail W_{12} f(x) et un signal d'approximation S_{12}f(x) (qui n'est pas représenté ici). Ce signal S_{12}f(x) est à son tour décomposé en un détail W_{22} f(x) et une approximation S_{22}f(x), et ainsi de suite jusqu'à obtenir les cinq signaux de détails W_{12} f(x)à W_{25} f(x) correspondant aux détails des cinq premières résolutions, et le signal d'approximation à la résolution 5 S_{25}f(x).

Ainsi, l'analyse multi résolution agit comme un zoom mathématique sur le signal à analyser, on peut, en faisant varier l'échelle d'analyse du signal, extraire les détails présents à telle ou telle résolution.

1.5. La transformée en ondelettes continue

La transformée en ondelettes utilise des translations et des dilatations d'une fonction fixe [4], l'ondelette mère ψ .

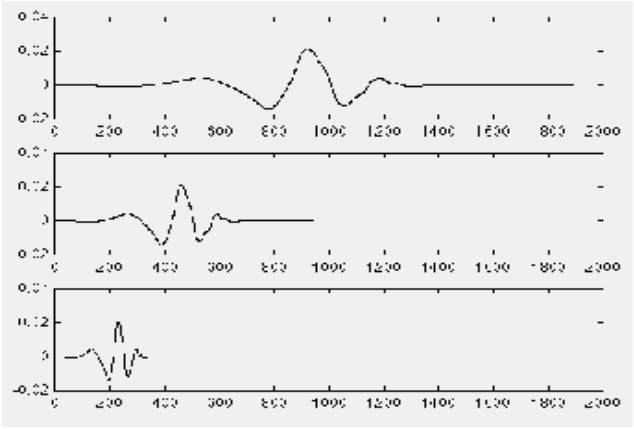
$$\psi_{a,b}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right) \qquad ; \text{ Avec } a>0 \qquad (1.4)$$

Avec *a*, b ∈ R ; $a \neq 0$ ou *a* sert à dilater (comprimer ou étendre) la fonction alors que *b* sert à la translater (la déplacer selon l'axe des temps).

Quand on analyse un signal f(x) avec les ondelettes, on le transforme en une fonction de deux variables (le temps et l'échelle d'analyse du signal) qu'on peut appeler W (*a*,b) :
Noté :

$$W(a,b) = \frac{1}{\sqrt{a}} \int f(x)\psi_{a,b}(x)dx \qquad (1.5)$$

a représente le facteur d'échelle. L'ondelette est plus comprimée lorsque le facteur d'échelle diminue.



$$f(t) = \psi(t) \quad ; \quad a = 1$$

$$f(t) = \psi(2t) \; ; \quad a = \frac{1}{2}$$

$$f(t) = \psi(4t) \; ; \quad a = \frac{1}{4}$$

1.6. Les étapes de calcul des coefficients de l'ondelette

Il existe cinq étapes pour le calcul des coefficients de la transformée en ondelettes [2]:
Étape 1 : Segmentation du signal et comparaison avec l'ondelette à échelle fixe .
Étape 2 : Calcul du degré de corrélation (figure 1.7).
Etape 3 : Décalage temporel (en position) et calcul des nouveaux coefficients (Figure 1.8).
Etape 4 : Variation du facteur d'échelle et calcul des nouveaux coefficients (figure 1.9).
Etape 5 : Refaire les étapes de 1 à 4 pour les différentes échelles.

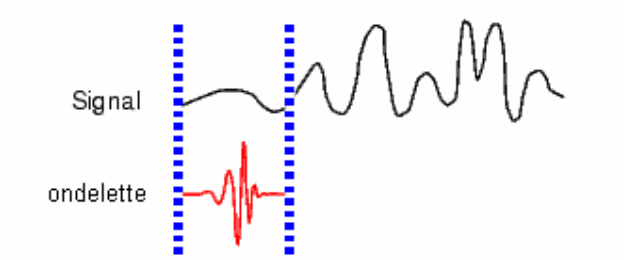


Figure 1.7. : Calcul du degré de corrélation.

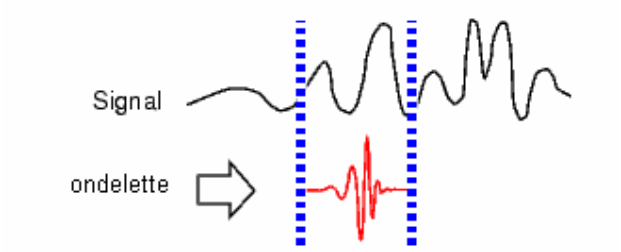


Figure 1.8 : Décalage temporel (en position)
et calcul des nouveaux coefficients.

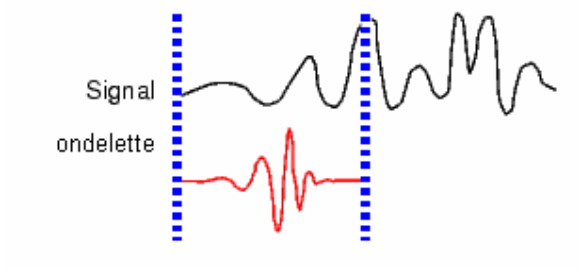


Figure 1.9: Variation du facteur d'échelle
et calcul des nouveaux coefficients.

1.7. Analyse multi résolution de Stéphane Mallat. (S. Mallat).

L'algorithme qui donna une avancée aux ondelettes ne fut établi qu'en 1988 par S.Mallat. Dès ses premiers travaux, avec son algorithme de décomposition et de reconstruction pour l'analyse multi résolution, les ondelettes se sont imposées comme une technique digne d'intérêt pour de nombreux problèmes de traitement de signal. Il a montré que les coefficients d'ondelettes peuvent être calculés à partir d'une transformée pyramidale mise en oeuvre à l'aide de filtres numériques, récursifs ou non [4]. Le principe de la transformée pyramidale consiste en la décomposition du signal à analyser à l'aide d'une paire de filtres. L'un de ces filtres fournira les coefficients d'ondelettes (ou détails), le second, les coefficients d'approximation est à son tour décomposée par une seconde paire de filtres, l'ensemble constituant une pyramide de filtres.

Ce signal est décomposé en entités indépendantes qui appartiennent chacune à différentes échelles successives et défini comme un emboîtement des différentes descriptions à différent temps/fréquence.

Chaque décomposition donne lieu à des signaux contenant une information discriminante [6].

D'un point de vue mathématique, l'analyse multi résolution consiste à approcher une fonction *f* par des approximations successives, représentées à différentes résolutions, (d'où le

nom de multi résolution) et contenant de plus en plus d'informations dans des espaces emboîtés, en enregistrant un degré de corrélation avec l'ondelette. L'idée de base est, donc, de mesurer les changements entre les approximations f_j et f_{j+1} de f aux échelles respectives 2_j et $2_{(j+1)}$. (D'où l'appellation Dyadique).

La résolution en sortie de chaque paire (ou banc) de filtres étant deux fois inférieure à la résolution d'entrée, on parle d'analyse multi résolution dyadique.

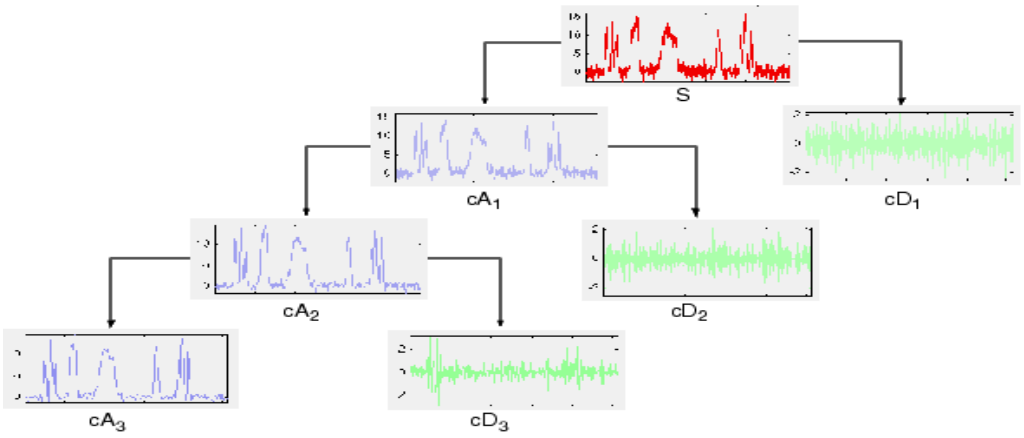


Figure 1.10 : Procédé de décomposition pour l'analyse multi résolution [2]

1.8. Algorithme de décomposition –reconstruction de S. Mallat

Le principe de cet algorithme pour les ondelettes est présenté dans la figure 1.10

Soit *x(n)* un signal échantillonné correspondant au signal d'origine. Ce signal est décomposé sur plusieurs niveaux de résolutions en deux bandes de fréquences (passe-haut et passe bas) à la manière approximation et détails.

1.8.1. Décomposition/reconstruction.

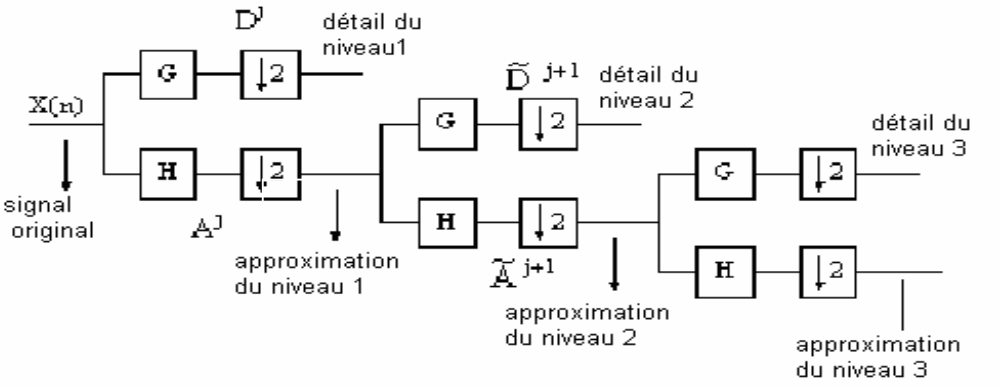


Figure 1.11 : Schéma de L'algorithme de Décomposition de S.Mallat [8]

L'opération de décomposition passe par plusieurs itérations. Une itération est représentée par le schéma suivant :

A^j : signal d'approximation de niveau j

$\tilde{A}^{j+1}$: signal issu de la convolution de A^j avec le filtre H

D^j : signal de détail de niveau j.

$\tilde{D}^{j+1}$: signal issu de la convolution de A^j avec le filtre G

Avec : $\tilde{A}^{j+1}(n) = \sum_k h(k-n)A^j$ (1.6)

$$\tilde{D}^{j+1}(n) = \sum_k g(k-n)A^j \quad (1.7)$$

1.8.1.1 Décimation **:** [9]

L'opération de décimation consiste à prendre un échantillon sur deux d'un signal [9]. d'où :

$$A^{j+1}(n) = \tilde{A}^{j+1}(2n) = \sum_k h(k-2n)A^j(k) \quad (1.8)$$

$$D^{j+1}(n) = \tilde{D}^{j+1}(2n) = \sum_k g(k-2n)A^j(k) \quad (1.9)$$

Le signal A^{j+1} représente une version lissée du signal A^j

D^{j+1} représente la différence d'information entre A^j et A^{j+1}.

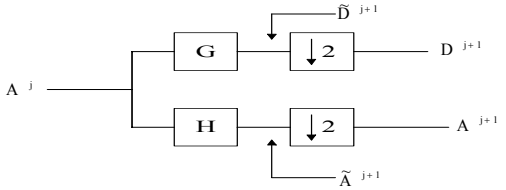


Figure 1.12 : Schéma de la décimation [8]

1.8.1.2. Reconstruction.

La reconstruction est l'étape inverse de la décomposition avec une interpolation

Le schéma de reconstruction du signal original est le suivant :

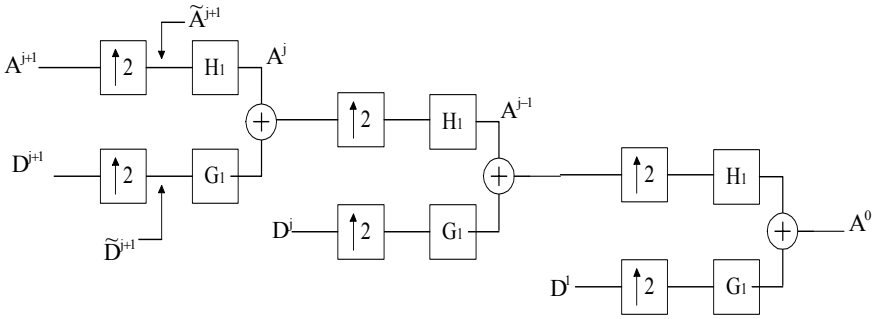


Figure 1.13 : Processus de reconstruction du signal

1.8.1.3. Interpolation par des zéros :

L'interpolation par des zéros consiste à insérer un zéro tous les deux échantillons successifs.

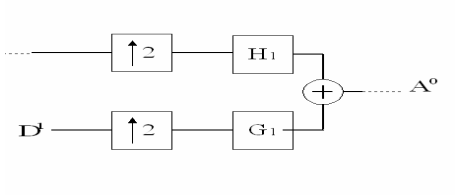


Figure 1.14 : Processus d'interpolation du signal

1-9. Calcul de complexité .

Le tableau 1.1 résume les différences entre les transformées présentées [3]. Le tableau montre que le nombre d'opérations nécessaires pour effectuer la transformée en ondelettes sur n échantillons est directement proportionnelle à n.

Tableau 1.1 : Nombre d'opérations nécessaires aux différents types de transformées sur n échantillons

Type de transformée	Opération
transformée de Fourier	$O(n^2)$
transformée de Fourier rapide	$O(n \log_2 n)$
transformée en ondelettes	$O(n)$

Conclusion

Dans ce chapitre, nous avons présenté l'évolution de la transformée de Fourier à la transformée en ondelettes. Aussi nous avons présentés l'algorithme multi résolution de S.Mallat qui présente un certains nombres d'avantages selon l'application envisagée.

CHAPITRE 2
EVALUATION DE L'ALGORITHME DE S.MALLAT PAR L'ARITHMETIQUE EN LIGNE

2.1. Introduction

Il est inutile de souligner que les opérations arithmétiques (addition, soustraction, multiplication, division) sont importantes, et que tout programmeur les utilise souvent sans savoir comment elles ont été calculées. Il suffit de parcourir la littérature scientifique consacrée aux algorithmes d'addition et de multiplication, pour constater que des centaines de méthodes différentes existent.

Calculer rapidement et avec une bonne précision est un but essentiel dans le domaine scientifique. Les exigences de rapidité et de précision ont souvent pu paraître antagonistes.

Les concepteurs choisissent souvent des circuits arithmétiques série quand la taille du circuit est une donnée critique de l'implémentation.

Afin de remédier aux différents problèmes dus à la circulation parallèle des opérandes et à la propagation de la retenue, l'utilisation du mode de calcul série se présente comme une bonne alternative; le développement du calcul MSB (Most Significant Bit) en tête s'avère très intéressant, vu que les systèmes redondants de représentation des nombres donnent naissance à plusieurs algorithmes dû à la non propagation de la retenue; citons parmi eux le mode de calcul en ligne. [10], [11],[12],[13].

2.2. L'arithmétique en ligne

Avizienis [14] a introduit l'écriture des nombres dans un système redondant de représentation de nombres (voir appendice A). Plus tard furent élaborés des algorithmes de calcul pour des opérations élémentaires et des fonctions plus complexes. Ces algorithmes sont basés sur la circulation des opérandes dans l'opérateur, bit par bit, le bit le plus significatif (MSB) en tête. [15], [16], [17], [18], [19], [20].

2.2.1. Propriétés de l'arithmétique en ligne

Les propriétés de l'arithmétique en ligne sont définies de la manière suivante [11] :

- Les opérandes sont introduits, à chaque cycle, bit par bit, le bit de poids fort en tête,
- Les résultats sont obtenus de la même manière, bit par bit, poids fort en tête, mais avec un retard *p*, tel qu'au pas *j*, alors que le $j^{(ième)}$ digit des opérandes sont introduits, le $(j-p)^{ième}$ digit du résultat est généré.
- Le retard *p* est très petit devant la taille des opérandes d'entrée.
- Les opérandes ainsi que les résultats sont écrits dans un système de notation redondant

2.3. Les systèmes redondants de représentation des nombres

L'arithmétique en ligne utilise un système de notation et de calcul non conventionnel appelé système de notation d'Avizienis ou notation redondante.

Ce système introduit la multitude d'écriture, pour certains nombres appelée redondance.

Cette écriture offre deux aspects fondamentaux à cette arithmétique :

1- La possibilité de représenter un nombre sous différentes représentations d'où le concept de correction du nombre résultat au fur et à mesure de sa génération.
2- La possibilité d'effectuer des opérations sans propagation de retenue, soit chiffre de poids fort en tête.

2.3.1. Formulation mathématique :

Les nombres sont représentés en base β avec des chiffres pris non plus dans l'ensemble {0,1... β-1} mais dans l'ensemble {-*a*...-1, 0 ,1...*a*}. , ou *a* est un entier inférieur ou égal à β-1. Ces chiffres négatifs sont à l'origine de la désignation de ce système comme système de représentation des nombres à chiffres signés (Signed Digit Number System).

- Si $2a+1 \geq \beta \Rightarrow$ tous les nombres sont représentés dans ce système.
- Si $2a+1 > \beta \Rightarrow$ il existe un algorithme d'addition sans propagation de retenue

L'intérêt du système de représentation redondante est qu'il existe dans ces systèmes des algorithmes permettant d'effectuer des additions de façon totalement parallèle, c'est à dire sans propagation de retenue. En particulier l'algorithme proposé par Avizienis.

Dans la suite et afin d'éviter toute confusion entre le signe des chiffres et l'opérateur de soustraction, nous noterons les chiffres négatifs avec une barre.

Par exemple, le chiffre -1 est noté $\bar{1}$.

Le nombre 2345 peut s'écrire en base 10 avec l'ensemble de chiffre

{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5} selon les codages suivants :

$235(-5) = 2*1000 +3*100+5*10 -5 = 2345$

$24(-5)(-5) = 2*1000+4*100-5*10-5= 2345$

On dit alors que le système est redondant.

Autre exemple :

La représentation binaire du nombre $x=3/4$ est : 0,11 ; ce même nombre peut avoir les représentations suivantes :

X= 1, $\bar{1}$ 1 ou X= 1,0 $\bar{1}$ ou X= 0,11

Nous pouvons vérifier que :

- $X= 1.2^0 - 1.2^{-1} +1.2^{-2} = 3/4$
- $X= 1.2^0 + 0.2^{-1} - 1.2^{-2} = 3/4$
- $X=0.\ 2^0 + 1.2^{-1} +1.2^{-2} = 3/4$

L'intérêt que cette multitude d'écriture pour un même nombre représenté, pour le mode de calcul en ligne réside dans le fait que les digits résultats, générés sont graduellement corrigés, de telle manière à ce que les digits générés convergent vers le résultat exact de l'opération.

2.3.1.1. Facteur de Redondance K:

Le nombre *g* est le plus grand nombre représenté dans une base ß avec une précision n :

$$K=\sum_{i=1}^{n} x_i .\beta^{-i} = g / (ß -1) \qquad (2.1)$$

Avec n tendant vers l'infini et les conditions sur *g*, [19] le système est défini comme suit:

Tableau 2.1 type de facteur de redondance

g	K	Système
ß /2	ß /(2.(ß -1))	Redondance minimale
ß -1	1	Redondance Maximale

Les notations mathématiques utilisées dans ce travail sont présentées dans le tableau suivant :

Tableau 2.2 : Notations utilisées

Notation	**Signification**
K	Facteur de Redondance
ß	Base de notation
g	Plus grand digit de la base
p	Retard en ligne
ε_j=1/2	Erreur résiduelle
R[j]	Reste partiel
H[j]	Résidu total
S(H[j])	Fonction de sélection

2.4. Application à une fonction Z=f(X, Y)

La méthodologie d'application du traitement en ligne à une fonction [12]

- Définition de l'opération à effectuer.
- Définition des plages d'opérandes.
- Calcul du reste partiel.
- Décomposition de la relation de récurrence.
- Calcul du reste complet.
- Calcul de l'intervalle de choix.
- Détermination de la fonction de choix

2.5. Algorithme de l'addition [21]

Addition de deux opérandes : X+Y

Les opérandes seront définis dans l'intervalle :

$1/k^{norm} \leq |\mathbf{X}| , |Y| < 1$

Avec **:** norm entier>0 : coefficient de normalisation

Les nombres sont normalisés afin d'assurer une convergence dans des intervalles bien définis.

Soit $Z_1 = X+Y$ (2.2)

Avec $2 / k^{norm} \leq |Z_1| < 2$ (2.3)

Pour normaliser Z_1 dans l'intervalle de convergence, il y'a lieu de le diviser par 2, d'où

$1/k^{norm} \leq |Z_1|/2 < 1$ (2.4)

Donc il faut calculer la somme de (X + Y)/2, puis multiplier le résultat par 2,

Posons alors, $Z=Z_1/2$ (2.5)

Comme énoncé précédemment dans 2.2.1, chaque opération en mode enligne possède un retard p spécifique, nous calculons ainsi la somme :

$$Z=k^{-p}.\ (X+Y)/2 \qquad (2.6)$$

Avec k^{-p} représentant un décalage .

Afin d'assurer la convergence de l'algorithme, une condition d'arrondi est définie comme suit : [21]

$$Z[j]-\ (k^{-j}).\varepsilon_j \le k^{-p}.(X[j]+Y[j])/2 < Z[j]+(k^{-j}).\varepsilon_j \qquad (2.7)$$

Afin d'assurer un résultat avec une précision de la moitié du dernier bit donc $\varepsilon_j=1/2$.

On défini un reste partiel .

$$R[j]= [(1/2).k^{-p}\ (X[j]+Y[j])-Z[j]].(k^{-j}) \quad \Leftrightarrow \quad -1/2 \le\ R[j] <1/2 \qquad (2.8)$$

En remplaçant tous les opérandes, le reste partiel sera de la forme suivante :

$$R[j] = k.R[j-1]+ (1/2).k^{-p}\ .(x_j + y_j)- z_j \qquad (2.9)$$

Définissons une nouvelle valeur appelée résidu complet H[j] tel que :

$$H[j]= k.R[j-1]+ (1/2).k^{-p}\ .(x_j + y_j) \qquad (2.10)$$

Alors $R[j]=H[j]- z_j$ (2.11)

Les valeurs de z_j appartiennent à {-g … + g }.

La fonction de sélection sera alors définie de la manière suivante,

$$z_j = \begin{cases} -g & \text{si } -g-1/2 \le H[j] < -g+1/2 \\ .. & .... \\ -1 & \text{si } -3/2 \le H[j] < -1/2 \\ 0 & \text{si } -1/2 \le H[j] < +1/2 \\ 1 & \text{si } +1/2 \le H[j] < 3/2 \\ \dots & \dots \\ g & \text{si } a-1/2 \le H[j] < g+1/2 \end{cases} \qquad (2.12)$$

Les intervalles sont alors définis par les valeurs limites de H[j] :

$$\begin{cases} g+1/2 > H[j]_{max} \\ -g-1/2 \le H[j]_{min} \end{cases} \qquad (2.13)$$

Cette inégalité permet de déterminer les valeurs limites $H[j]_{max}$, $H[j]_{min}$ de la manière suivante, conformément à (2.10) :

$$H[j] = k.\ R[j-1]+ (1/2).K^{-p}\ .(x_j + y_j) \ \text{ avec } -1/2 \le R[j]<1/2 \quad / \forall\ j \qquad (2.14)$$

Donc

$$H[j]_{max} = k.(1/2)+ (1/2).k^{-p} .(g+ g) \quad et \quad x_j = y_j = g \tag{2.15}$$

$$H[j]_{max} = k./2+ k^{-p} .g \tag{2.16}$$

Par ailleurs,

$$H[j]_{min} = k.(-1/2)+ (1/2).k^{-p} .(-g-g) \quad et \quad x_j = y_j = -g \tag{2.17}$$

$$H[j]_{min} = -k./2 - k^{-p} .g \tag{2.18}$$

D'ou $g+1/2 \geq k./2+ k^{-p} .g$ (2.19)

et $-g-1/2 \leq -k./2- k^{-p} .g$ (2.20)

On peut avoir : $k^{-p} \leq (2g+1-k) / 2g \Rightarrow p \geq \log_k((2g)/(2g+1-k))$ (2.21)

Le tableau suivant montre les différents délais en fonction de la base [21]:

$$p \geq \log_k((2g) / (2g+1-k)) \tag{2.22}$$

Tableau 2.3. Délai en ligne de l'addition

Base ß / g	2	4	6	8
1	1			
2		2		
3		1	2.585	
4			1.415	3
5			1	1.737
6				1.263
7				1

Nous remarquons que pour une base k donnée, la valeur de la fonction $\log_k((2g)/(2g+1-k))$ (et par conséquent celle de p) diminue avec les valeurs de g croissantes.

2.5.1. Algorithme de l'addition / soustraction

Initialisation :

R=0 ; Y=0 ; Z=0 ; p=1
Pour i=0 à p+N faire
Début
$R_j=2R_i+2^{-(p+1)}.(x_{.j} + y_j)$;

Si I>=p alors

Début

$H_i=R_i$

$$z_j=\begin{cases} -1 & \text{si } -3/2 \le H_j<-1/2 \\ 0 & \text{si } -1/2 \le H_j<1/2 \\ 1 & \text{si } 1/2 \le H_j< 3/2 \end{cases}$$

$R_i=H_i-z_i$;
$Z_j=Z_{j-1}+z_j2^{-j}$;
Fin ; fin si
fait
Resultat=$Z*2^p$
Fin programme

2.6. Algorithme de la multiplication : X*Y

Les opérandes sont définis dans l'intervalle $1/k^{norm} \le |X|$, $|Y|<1$.

Avec norm représentant le coefficient de normalisation.

Définissons la fonction (X*Y) en incluant le retard d'entrée des opérandes :

$$Z=k^{-p} *(X*Y) \qquad (2.23)$$

Le résidu complet est défini par :

$$H[j]= k.(H[j-1]-z_{j-1})+ [x_{j-1}.Y(j-1) + y_{j-1}.X(j)]. k^{-p+1}+ k^{-p+j}.x_j.y_j \qquad (2.24)$$

La fonction de sélection est alors :

$$z_j = \begin{cases} -g & \text{si} \quad -g-1/2 \le H[j]< -g+1/2 \\ -1 & \text{si} \quad -3/2 \le H[j]< -1/2 \\ 0 & \text{si} \quad -1/2 \le H[j]< +1/2 \\ 1 & \text{si} \quad +1/2 \le H[j]< 3/2 \\ g & \text{si} \quad g-1/2 \le H[j]< g+1/2 \end{cases} \qquad (2.25)$$

Le retard s'exprime par : $k^{-p} \le (2g+1-k)/(4g. k^{-norm})$ d'ou :

$$p \ge \log_k[(4.g)/(2g+1-k)]-norm \qquad (2.26)$$

Le tableau suivant montre les différents retards en fonction de la base de travail et du coefficient de normalisation

Tableau 2.4. Délai en ligne de la multiplication avec norm =1 (nombres normalisés)

Base ß / g	2	4	6	8
1	1			
2		2		
3		1	2.5850	
4			1.4150	3
5			1.0	1.7370
6				1.2630
7				1

Tableau 2.5 Délai en ligne de la multiplication avec norm = 2 (Nombres Pseudo Normalisés).

Base ß / g	2	4	6	8
1	0			
2		1		
3		0	1.580	
4			0.4150	2
5			0	0.7370
6				0.2630
7				0

2.6.1. Algorithme de la multiplication

Initialisation
R=0 ; Y=0 ;Z=0 ;p=1
Pour i=0 à p+N faire
Début
$R_i=2xR_i+2^{-p}.(x.Y+y.X)+2^{-p+i}.x.y$
$Y_i=Y_{i-1}+y_i.2^{-i}$; $X_i=X_{i-1}+x_i.2^{-i}$;
Si I>=P alors
Début
$H_i=R_i$
$z_i = -1$ si $-3/2 \le H_i < -1/2$
$z_i = 0$ si $-1/2 \le H_i < -1/2$
$z_i = 1$ si $1/2 \le H_i < 3/2$
$R_i=H_i-z_i$
$Z_i=Z_{i-1}+z_i2^{-i}$;
fin
fin
fait
Résultat=$Z \cdot 2^p$
Fin programme

2.7 Modélisation de l'algorithme de S.Mallat par l'arithmétique en ligne

2.7.1 Description Mathématique

Les modèles des filtre à base d'ondelettes sont des filtres à réponse impulsionnelle finie d'ordre N, tel que :

$$Y^*_n = \sum_{i=0}^{N} h_i X_{n-i} \tag{2.27}$$

Où : Y^*_n : signal de sortie; X_n : signal d'entrée;

h_i: Coefficients du filtre N : l'ordre du filtre.

A l'étape j, $X_n[j]$ et $Y^*_n[j]$ se présentent comme suit :

$$X_n[j] = X_n[j-1] + x_n[j]k^{-j} \tag{2.28}$$

$$Y^*_n[j] = Y^*_n[j-1] + y^*_n[j]k^{-j} \tag{2.29}$$

Tel que : $b^{-r} \leq |X_{n-i}| < 1$ (2.30)

Où b représente la base et r le coefficient de normalisation.

$$b^{-r}\sum_{i=0}^{N}|h_i| \leq \sum_{i=0}^{N}|h_i||X_{n-i}| < \sum_{i=0}^{N}|h_i| \tag{2.31}$$

De l'équation (2.27), il en résulte :

$$b^{-r}\sum_{i=0}^{N}|h_i| \leq Y^*_n < \sum_{i=0}^{N}|h_i| \tag{2.32}$$

Où, $Y^*_n \in [\sum_{i=0}^{N}|h_i|/b^r, \sum_{i=0}^{N}|h_i|[$ $\Leftrightarrow$ $Y^*_n/\sum_{i=0}^{N}|h_i| \in [1/b^r, 1[$ (2.33)

L'algorithme en ligne est caractérisée par le retard p qui est défini par [21]:

$$Y_n = b^{-p}.Y^*_n/\sum_{i=0}^{N}|h_i| \tag{2.34}$$

Afin d'utiliser une architecture en pipeline, nous calculons le nouveau intervalle de convergence pour un délai nul p=0.

Donc, au lieu de calculer Y^*_n directement, on calcule :

$$Y_n = Y^*_n / \sum_{i=0}^{N}|h_i| \quad \Leftrightarrow \quad Y_n = \sum_{i=0}^{N} h_i X_{n-i} / \sum_{i=0}^{N}|h_i| \tag{2.35}$$

Posons $\sum_{i=0}^{N}|h_i| = D$

L'expression du nouveau filtre devient :

$$Y_n = \sum_{i=0}^{N} h_i X_{n-i} /D \tag{2.36}$$

2.7.2 Calcul de l'intervalle de convergence :

Les opérandes appartiennent à l'intervalle, $]-b^{-r}..,.. b^{-r}[$, donc :

$$|X_{n-i}| \in [0 , b^{-r}[\quad (2.38)$$

Et $|h_i|.|X_{n-i}| \in [0 , |h_i|.[$ pour les (N+1) opérandes (2.39)

Donc :

$$\sum_{i=0}^{N} |h_i||X_{n-i}| \in [0 , \sum_{i=0}^{N} |h_i|. b^{-r}[\quad (2.40)$$

Pour assurer la convergence de l'algorithme :

$$\sum_{i=0}^{N} |h_i|. b^{-r} < 1 \quad \text{selon [21]} \quad (2.41)$$

Il en résulte : $r > \log_b(\sum_{i=0}^{N} |h_i|)$ (2.42)

2.7.3. Calcul en ligne

D'après l'équation(2.37) du nouveau filtre normalisé est :

$$Y_n = \sum_{i=0}^{N} h_i X_{n-i} /D \quad (2.43)$$

Prenons b=2.

La condition d'arrondissement symétrique est définie par :

$$Y_n[j] - \frac{2^{-j}}{2} \le \sum_{i=0}^{N} h_i X_{n-i}[j]/D < Y_n[j] + \frac{2^{-j}}{2} \quad (2.44)$$

$$\Leftrightarrow -1/2 \le 2^j.[(\sum_{i=0}^{N} h_i X_{n-i}[j]/D) - Y_n[j]] < 1/2 \quad (2.45)$$

2.7.4. Le résidu partiel

Le résidu partiel R[j] est défini comme suit :

$$R[j]=2^j.[(\sum_{i=0}^{N} h_i X_{n-i}[j]/D) - Y_n[j]] \quad (2.46)$$

L'algorithme converge si et seulement si.

$$-½ < R[j] \le ½ \quad (2.47)$$

En remplaçant les équations (2.28) et une version décalée de (2.43) on obtient :

$$R[j]=2^j[(\sum_{i=0}^{N} h_i X_{n-i}[j-1]/D) - Y_n[j-1]] + (\sum_{i=0}^{n} h_i x_{n-i}[j]/D) - y_n[j] \quad (2.48)$$

Après simplification de l'équation (2.48) on aura :

$$R[j]=2R[j-1]+L[j]-y_n[j] \quad (2.49)$$

Posons $L[J] = (\sum_{i=0}^{N} h_i X_{n-i}[j]/D)$ (2.50)

Comme étant des valeurs pré calculées et stockées en mémoire.

2.7.5. Le résidu complet

Le résidu complet est décrit par :

$$H[j]=R[j]+y_n[j] \quad (2.51)$$

Des équations (2.49) et (2.51):

$$H[j]= 2.H[j-1]+L[j]-2y_n[j-1] \quad (2.52)$$

Sachant que de l'équation (2.47):

$$-1/2\leq R[j-1]\leq 1/2 \quad (2.53)$$

Utilisant (2.51) et (2.52)

$$-1/2+y_n[j] \leq H[j-1]\leq 1/2+y_n[j] \quad (2.54)$$

Avec $y_n[j]\in\{-1,0,+1\}$: Redondance maximale utilisée

Finalement, le résidu complet définit la valeur du bit résultat par l'inégalité :

$$-3/2\leq H[j-1] < 3/2 \quad (2.55)$$

Des équations (2.52) et (2.54) la fonction de sélection est donnée par S(H[j]), définit par :

$$\begin{cases} H[j] = 2H[j-1] + L[j] - 2y_n[j-1] \\ y_n[j] = S(H[j]) \end{cases} \quad (2.56)$$

$$\text{Avec } y_n[j]= \begin{cases} -1 & \text{si } -3/2\leq H[j] < -1/2 \\ 0 & \text{si } -1/2\leq H[j] < 1/2 \\ 0 & \text{si } 1/2\leq H[j] < 3/2 \end{cases} \quad (2.57)$$

Notre travail consiste alors à implémenter les équations suivantes :

$$H[j] = 2.H[j-1]+L[j]-2y_n[j-1] \quad (2.58)$$

$$\text{Avec:} \quad L[j]= \sum_{i=0}^{N} h_i x_{n-i}[j]/D \quad (2.59)$$

Conclusion

Dans ce chapitre nous avons présenté le concepte de l'arithmétique en ligne et ses propriétés ainsi que le calcul des opérations de base ; l'addition et la multiplication. Une modélisation mathématique de l'algorithme multi résolution de Stéphane-Mallat a été présentée en vue de son implémentation sur un circuit FPGA.

CHAPITRE 3
ARCHITECTURE DE LA TRANSFORMEE EN ONDELETTES

3.1. Introduction

Depuis les travaux de S.Mallat sur les algorithmes multi résolutions, de nombreuses architectures de transformée en ondelettes sont apparues. La plupart d'entre elles reposent sur la technique des bancs de filtres [3].

Dans ce chapitre, nous verrons différents types d'architectures pour la transformée en ondelettes à deux dimensions (2-D) multi résolution de S.Mallat, nous présenterons aussi quelques systèmes issus de la littérature qui ont fait l'objet d'une implémentation sur circuit FPGA.

3.2. Contraintes pour la conception d'architectures [3]

Il existe plusieurs contraintes, dont les plus importantes sont :

3.2.1 Contrainte de qualité

Très importante dans le domaine du traitement du signal, elle nous indique si les données obtenues représentent d'une manière acceptable l'information traitée. C'est un critère majeur dans les domaines de la compression d'image par exemple, où l'image après décompression doit être la plus proche possible de l'originale. Lorsque les applications ne nécessitent pas la reconstruction de l'image, cette contrainte est beaucoup moins importante (détection de contours). Ce facteur est, dans le domaine du traitement du signal, le seul à être subjectif. Il se traduit en terme d'implantation numérique par une contrainte de précision sur les données.

3.2.2. Contrainte de précision

Lorsque les algorithmes sont implantés de manière logicielle, les valeurs traitées sont le plus souvent des valeurs réelles (virgule flottante) et représentent donc de manière quasi-parfaite le signal d'origine. D'autre part, lors des calculs, aucune approximation n'est nécessaire sur les valeurs traitées; il n'y a donc aucune perte d'information due aux calculs.

Par contre, lors de l'implantation matérielle, nous sommes souvent obligé de faire un compromis entre la précision souhaitée (largeur du bus de données, valeurs codées en entiers, virgule fixe ou flottante) et la surface ou la complexité du système. En diminuant la précision sur les valeurs, on peut ainsi diminuer la complexité du système final et donc la surface totale du circuit. On répond ainsi en partie à la troisième contrainte.

3.2.3 Contrainte de surface

Lors de la conception d'un système sur puce ou sur circuit programmable la contrainte de surface est primordiale. Dans le cas d'un circuit programmable (FPGA), il faudra tenir compte du nombre de blocs logiques disponibles sur le circuit. Plus le système sera complexe, plus le routage sera difficile à effectuer, et le nombre de circuits nécessaires à son implantation augmentera. De plus, en minimisant la taille totale du système on minimise également la consommation totale du circuit. Ces critères de surface et de consommation sont des facteurs d'une importance capitale notamment dans les systèmes embarqués.

Un élément essentiel de la surface totale est la quantité de mémoire nécessaire au traitement des données. Plus la mémoire implantée sera importante, plus la surface totale sera importante [22] [23].

3.3. Architectures pour la transformée en ondelettes 2-D par filtrage numérique

La transformée en ondelettes 2-D multi résolution est une opération gourmande en calculs à effectuer et introduit des délais considérables dans le traitement des images.

3.3.1. Architecture directe

Ce système consiste en un ou deux blocs de transformée en ondelettes1-D, et une unité de stockage N×N où N représente la taille de l'image. Cette architecture doit achever le calcul de chaque dimension de chaque niveau de résolution avant d'entamer la dimension ou la résolution suivante. Son unique avantage est d'avoir le nombre d'unité fonctionnelles la plus petite, mais impose une quantité de mémoire importante. [24],[25]

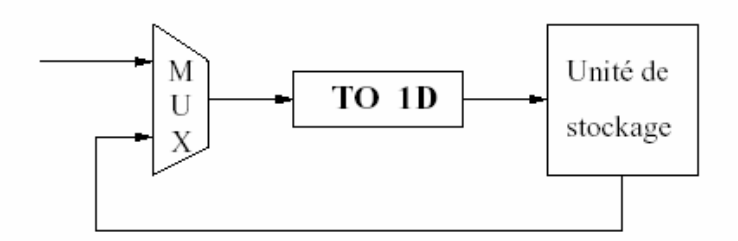


Figure 3.1 : Implantation 2.D directe [3]

Une variante consiste à remplacer le bloc de transformée d'une dimension (1-D) par un bloc de transformée 2-D ou à utiliser deux filtres 1-D simultanément. On augmente ainsi la surface nécessaire au bloc de calcul et le nombre d'itérations sera divisé par deux, ce qui permet de réduire la latence globale du système. Cependant, cela implique également l'ajout de la logique nécessaire à la transposition ligne, colonne et accroît considérablement le nombre d'accès à la mémoire.

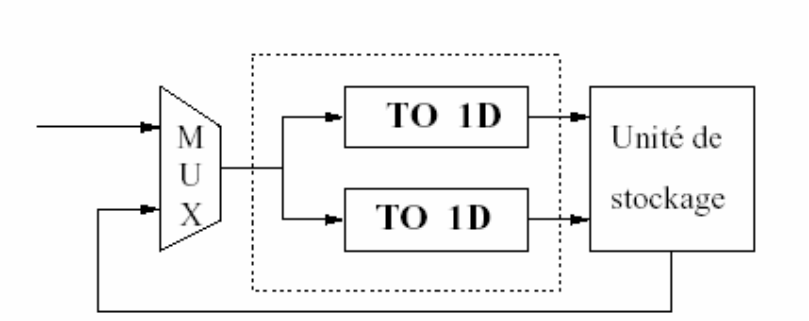


Figure 3.2 : Implantation 2.D directe – variante [3]

3.3.2. Architecture série –parallèle

Cette architecture est constituée en deux filtres de transformée en ondelettes 1D série pour le calcul horizontal et deux filtres 1-D parallèles pour le calcul vertical. le nombre d'unités fonctionnelles peut être trois fois plus important que pour le cas précédent, mais la quantité de mémoire nécessaire à son implémentation est considérablement réduite [27].

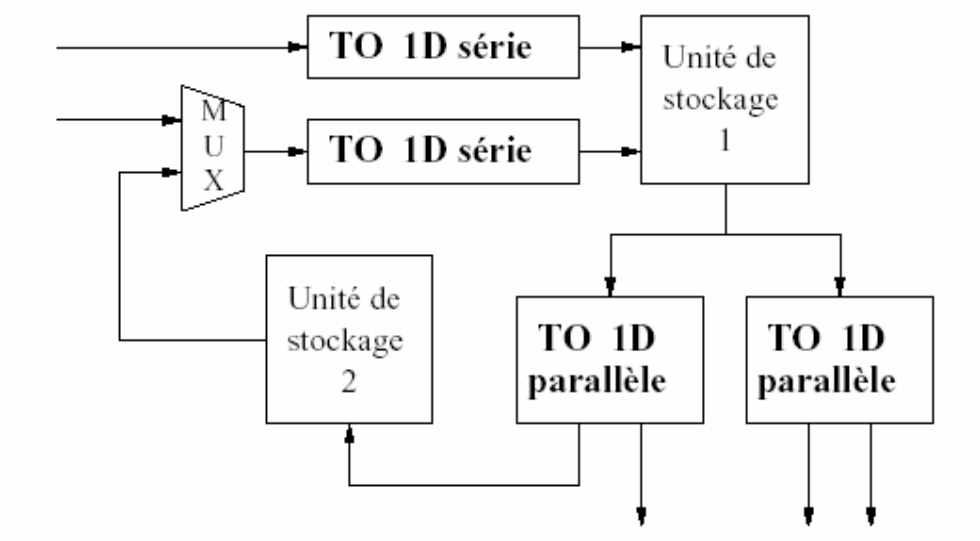


Figure 3.3 : Architecture série – parallèle

Chacun des blocs est composé d'un passe –bas et d'un passe haut. On a donc quatres filtres pour la décomposition horizontale (filtre série), ainsi que quatre filtres pour la décomposition verticale (filtre parallèle). Les filtres séries reçoivent les données selon l'axe horizontal et produisent les sorties dans l'ordre des lignes. Le premier filtre série effectue les calculs sur le premier niveau de résolution uniquement, alors que le second effectue les calculs sur toutes les résolutions. Les sorties de ces filtres sont stockées en mémoire dans l'ordre des lignes [3].

Les filtres en parallèles effectuent les calculs selon l'ordre des colonnes, les résultats sont également stockés selon l'ordre des lignes.

3.3.3. Architecture parallèle – parallèle

Cette architecture est une version modifiée de l'architecture série-paralléle, utilisée pour la décomposition horizontale. Celle ci est remplacée par un filtre parallèle. La bande passante nécessaire est plus importante que dans le cas précédent, mais la taille mémoire reste équivalente. [24,25, 28].

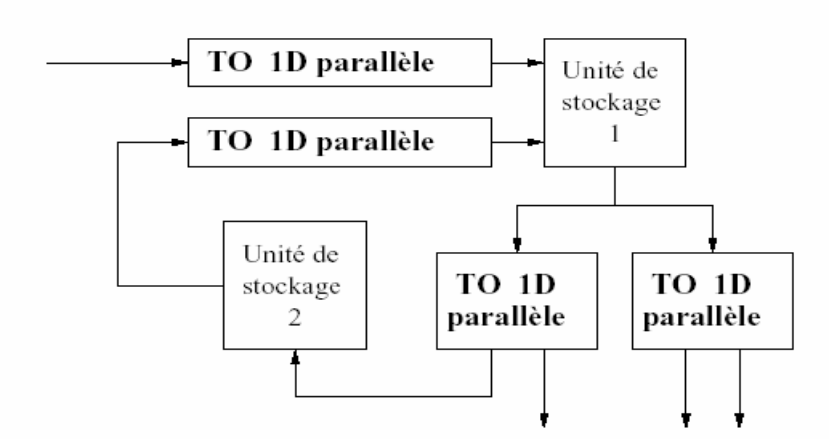


Figure 3.4 : Architecture parallèle–parallèle

Dans ce type d'implémentation, contrairement à l'architecture précédente, les filtres horizontaux sont remplacés par des filtres parallèles. De plus, une seule ligne est envoyée au système contrairement à l'architecture précédente qui effectue les calcules sur deux lignes en parallèle. Dans cette architecture les filtres horizontaux effectuent les calculs dans l'ordre des lignes, alors que les filtres verticaux travaillent selon l'ordre des colonnes. Les sorties de tous les filtres sont stockés dans l'ordre des lignes.

3.4 Comparaison des différentes architectures

Le tableau 3.1 montre les ressources nécessaires des différents types d'architectures pour la transformée en ondelettes 2-D. *K* représente la taille du filtre, les images étant de taille N×N [28].

Tableau 3.1. : Comparaison des différentes architectures

ARCHITECTURE	MULTIPLIEURS	TAILLE MEMOIRE	ROUTAGE	CONTROLE
directe	2*K*	N×N	FACILE	FACILE
série–parallèle	4*K*	~2*K* N+N	REGULIER	MODERE
parallèle	4*K*	~2*K* N+N	REGULIER	DIFFICILE

3.5 Implémentation matérielle de la transformée en ondelettes

Nous avons vu dans la section précédente les différentes catégories d'architectures existantes pour l'implémentation de la transformée en ondelettes. Nous allons présenter l'état de l'art des réalisations matérielles de la transformée en

ondelettes, notamment les améliorations apportées par les structures classiques afin d'économiser les ressources nécessaires et d'augmenter l'efficacité de ces architectures.

La littérature fournit plusieurs exemples d'implémentations de système pour la transformée en ondelettes 2-D. Les systèmes implémentés sur FPGA offrent l'avantage de pouvoir être facilement validés et offrent la possibilité d'être rapidement adaptés à des configurations particulières, changement des coefficients des filtres d'ondelettes, taille des mémoires, etc.

Nous pouvons distinguer deux méthodes d'implémentation de l'architecture directe. La première consiste à effectuer la transformée sur un niveau de résolution, à stocker la sous–image approximée en mémoire, puis à effectuer la transformée à la résolution suivante sur cette sous image. La seconde solution consiste à cascader les différentes blocs de décomposition 2-D, mais cette technique est très peu utilisée du fait de son coût important aussi bien en quantité de mémoire qu'en unités de calcul.

Trucheter [29] propose une architecture à base de FPGA qui repose sur la première technique. Le système nécessite deux FPGA de type Xilinx 4005 pour la compression, ainsi qu'un 4005 et un 4008 pour la décompression. Dans chacune des deux parties (compression et décompression) un FPGA est utilisé pour l'analyse ou la synthèse, le second étant employé pour le codage (ou le décodage).

Chaque FPGA dédié au calcul contient trois cellules de filtrage identiques constituées d'un filtre passe bas et d'un filtre passe haut chacune. Les filtres utilisés ne comportent que des coefficients dyadiques permettant l'implémentation de la transformée en ondelettes sans nécessiter l'emploi de multiplieurs dont le coût en surface est important figure 3.5.

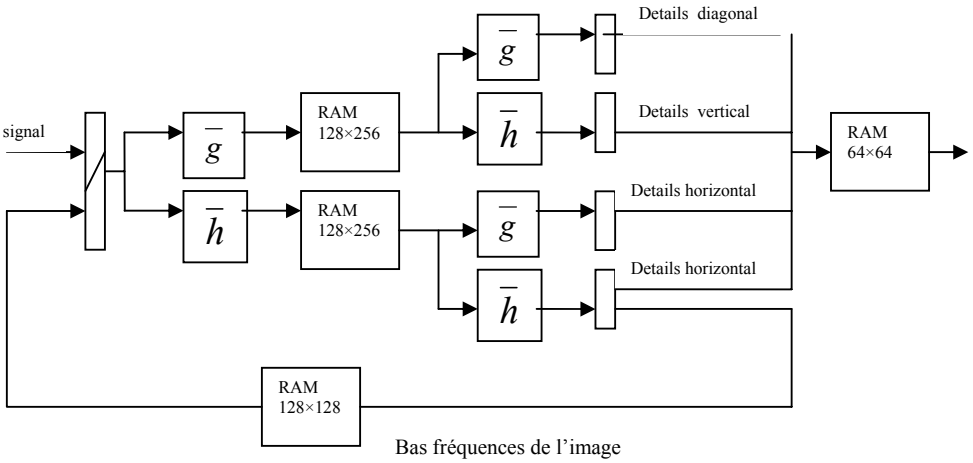


Figure 3.5 : Exemple d'implémentation d'une architecture directe sur FPGA

Le coût en mémoire de ce système reste important. Pour des images 256×256 pixels codés sur 8 bits les performances de ce système pour trois niveaux de résolution sont montrées dans le tableau 3.2 [3].

Tableau 3.2 : Performances d'une architecture directe sur FPGA.

Décomposition	Reconstruction
58 ms	16 ms

Afin de palier aux défauts de l'implantation directe de la transformée en ondelettes, certaines architectures permettant d'économiser les ressources nécessaires à la transformée ; aussi bien en termes de mémoire qu'en nombre de blocs logiques – ont vu le jour.

Guermeur [30] propose une architecture multi FPGA composée de deux circuits Xilinx 4013E (XA et XB) consacrés au calcul des algorithmes, et d'un Xilinx 4025E (XC) utilisé comme co-processeur d'adresse pour la mémoire partagée, et servant d'interface avec le monde extérieur. Une mémoire locale composée de 4 RAM est affectée aux deux circuits XA et XB. Une mémoire partagée composée de deux mémoires est destinée à l'enregistrement d'images.

Le filtre est partitionné dans les deux unités de calcul XA et XB lesquelles contiennent chacun deux multiplieurs, pour un taux d'occupation de 87 à 89 %. Les performances obtenues sont décrites dans le tableau suivant :

Tableau 3.3. Performances de l'architecture multi-FPGA

Images	Temps de traitement
581x763	88 ms → 1 image sur 3
512x512	51,6 ms → 1 image sur 2
256x256	12,9 ms → cadence vidéo

3.6. Les architectures de Alma technologies

Le circuit RC-2D- DWT figure 3.6 combine l'analyse et la synthèse de la transformée en ondelettes 2-D dans les caractéristiques. [31]

- Architecture basée sur une implémentation ligne colonne.
- Conception totalement synchrone.
- Combine l'analyse et la synthèse avec filtre 9/7 de Daubechies.
- Des images détaillées supérieure à 7 niveaux de résolution
- 16 bits inputs/outputs et 18 bits pour la précision interne.

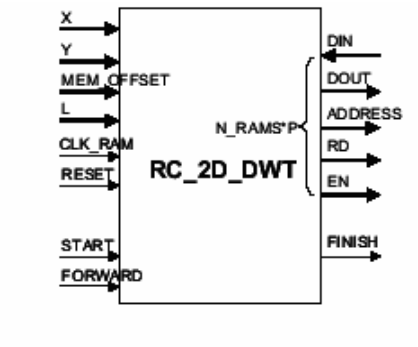


Figure 3.6.schéma global du RC-2D-DWT

Tableau 3.4 : Performances de RC-2D-DWT

Supported family	Device tested	CLB Slices	Clock IOB	IOB	PERFORMANCE (MHz)	XILINX TOOLS
Virtex	V300-6	1698	2	116	72	M4.1.01I
Virtex-E	V300-E-8	1698	2	116	90	M4.1.01I
Virtex-II	2V500-5	1695	2	116	95	M4.1.01I
Spartan-II	2S200-6	1698	2	116	77	M4.1.01I

3.7 . Implémentation Hardware de l'algorithme de S.MALLET.

Notre travail consiste à implémenter les équations suivantes :

$$H[j] = 2H[j-1]+L[j]-2y_n[j-1] \quad (3.1)$$

Avec: $$L[j]= \sum_{i=0}^{N} h_i x_{n-i}[j] / \sum_{i=0}^{N} h_i \quad (3.2)$$

Des équations (3.1) et (3.2), on définit une nouvelle quantité $L_p[j]$ telle que:

$$L_p[j] = \sum_{i=0}^{N} h_i x_{n-i}[j] / \sum_{i=0}^{N} |h_i| -2.y_n[j-1] \quad (3.3)$$

Le travail est divisé en deux parties, comme suit :

- Utilisation des blocs mémoires ou de lookup tables pour les valeurs pré calculées de $L_p[j]$, voir Annexe B.
- Le bloc en ligne pour la génération des bits résultats de la fonction de sélection.

3.7.1. Notation utilisée

L'utilisation d'une notation redondante, en complément à deux au niveau bit, est définie comme suit :

Tableau 3.5 : Notation en redondant

Digit redondant	**Notation complément à 2** (R_1R_2)
-1	11
0	00
1	01

Donc, chaque registre est divisé en deux sous registres R1 et R2, la valeur finale est obtenue de la façon suivante :

X= -2*R1+R2 (3.4)

3.7.2. Coefficients d'ondelettes en virgule fixe

Afin de réduire le nombre de coefficients d'ondelettes nuls, des filtres de petite taille sont utilisés [12], [13], et afin d'assurer une décomposition et une reconstruction sans pertes de l'image, des filtres bi orthogonaux à coefficients symétriques ont été implémentés.

Les coefficients du filtre, utilisé issus de la librairie de Matlab [2] sont :

Tableau 3.6 : Coefficients des filtres bi orthogonaux 3.1

Coefficients			
G0=[-0,3536	1,0607	1,0607	-0,3536]
H0=[-0,1768	0,5303	-0,5303	0,1768]
G1=[0,1768	0,5303	0,5303	0,1768]
H1=[-0,3536	-1,0607	1,0607	0,3536]

- G0, H0: filtres de décomposition
- G1, H1: filtres de reconstruction

Lors des implémentations à coefficients réels à précision infinie, Tableau 3.6, le dimensionnement de l'architecture (largeur des bus de données) est régi par la précision des coefficients. Afin de minimiser ce dépassement, des troncatures ou arrondies sont effectuées, ceci induit à des erreurs de calcul. Nous avons évité un tel procédé de la manière suivante :

Lors du filtrage de l'image, les filtres utilisés travaillent en paires :

- Décomposition : (G0, G0), (G0, H0), (H0, G0), (H0, H0)
- Reconstruction : (G1, H1), (G1, G1), (H1, G1), (H1, H1)

En factorisant $\sqrt{1/8}$, qui est une valeur à précision infinie qui ne peut être représentée dans un registre à taille fixe ; on obtient :

Tableau 3.7: Coefficients pré calculés

Coefficients
G0= [-1 3 3 -1]* $\sqrt{1/8}$
H0 =[-0.5 1.5 -1.5 0.5]* $\sqrt{1/8}$
G1 =[0.5 1.5 1.5 0.5]* $\sqrt{1/8}$
H1 =[-1 -3 3 1]* $\sqrt{1/8}$

Remarque :

Le produit croisé des filtres élimine la racine carrée, ceci amène le problème d'une précision infinie à une précision finie fixe sans erreur, donc le terme $\sqrt{1/8}$ à chaque décomposition / reconstruction est remplacé par 1/8 qui n'est qu'un simple shift de 3 bits.

Les nouveaux coefficients seront :

Tableau 3.8 : Normalisation des coefficients

Coefficients	|Somme du filtre|
G0= [-1 3 3 -1]	8
H0 = [-0,5 1,5 -1,5 0,5]	4
G1 =[0,5 1,5 1,5 0,5]	4
H1 =[-1 -3 3 1]	8

3.7.3. Stockage des valeurs $L_p[j]$

En développant l'équation (3.3) :

$$L_p[j]= [h_0.x_n+h_1x_{n-1}+h_2x_{n-2}+ h_3x_{n-3}]/\sum_{i=0}^{N}|h_i| -2.y_n[j-1] \quad (3.5)$$

Les valeurs de $L_p[j]$ sont stockées dans des mémoires et l'adresse est formée des bits de x_n à x_{n-3} et $y_n[j-1]$..

3.7.4. Taille des mémoires

Les mémoires correspondantes aux différentes architectures de décomposition et reconstruction sont mentionnées dans le tableau 5.7.

Tableau 3.9 : Taille des mémoires

	Décomposition		Reconstruction	
	Nombre de mémoire	Taille du filtre en bits	Nombre de mémoire	Taille du filtre en bits
Filtre Passe bas	4	8 x 243	16	8 x 27
Filtre Passe haut	4	8 x 243	16	8 x 27
Additionneur en ligne*			4	6 x 243

Cette architecture est utilisée pour chaque 16 échantillons de l'image correspondant à 4 filtres en parallèle, le pipeline est introduit dans le but de générer des résultats à chaque cycle, ces résultats sont réinjectés dans les étages suivants.

3.7.5. Taille des bus de données

Les mémoires calculées précédemment ont une taille maximale de 8 bits, toutefois la boucle concernant le H[j-1], tend à redimensionner l'architecture par l'utilisation de la récursivité, car les additions répétitives en boucle amplifient la

taille du bus d'une façon logarithmique. Afin de calculer la taille maximale de l'additionneur, l'approche mentionnée dans [64] été utilisée :

$$w=k+\log_2(m) \qquad (3.6)$$

Avec m : Taille initiale des mémoires suivant la valeur de L[j] sur 6, 7 ou 8 bits

Et k : Taille de l'opérande qui correspond à la profondeur de codage de l'image sur 8 bits.

Tableau 3.10 : Taille du bus de données

	Décomposition	**Reconstruction**
	Taille du CLA	**Taille du CLA**
Filtre passe bas	$w= 8+\log_2(8) = 11$	$w= 8+\log_2(8) = 11$
Filtre passe haut	$w= 8+\log_2(8) = 11$	$w= 8+\log_2(8) = 11$
Additionneur en ligne		$w= 8+\log_2(6) =10.5850 \approx 11$

3.7.6. Additionneur à retenue anticipée - CLA

Les additionneurs à retenues anticipées, voir Appendice C, sont des circuits digitaux dont le principe est le suivant : [65]

Le calcul des retenues est fait directement à partir des entrées.

- Avantages : Calcul en parallèle => Gain en rapidité.
- Inconvénients : Plus de portes logiques => Coût en complexité matérielle.

3.7.7. Bloc de sélection

Vu que H[j] est compris entre -3/2 et 3/2, trois bits seulement, peuvent nous renseigner sur l'appartenance de celui-ci, comme le montre le tableau suivant :

Tableau 3.11 :les Bits de sélection

Bits de H[J]	**Signification**
H_s	bit signe
H_{-1}	bit de poids « -1 »
H_0	bit de poids « 0 »

La sélection se fait de la manière suivante :

Tableau 3.12 : Fonction de sélection

H_s	H_0	H_{-1}	Y	y_nMsb	y_nLSB
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	-	-	-
1	0	0	-	-	-
1	0	1	-1	1	1
1	1	0	-1	1	1
1	1	1	0	0	0

Après simplification, on obtient le circuit suivant :

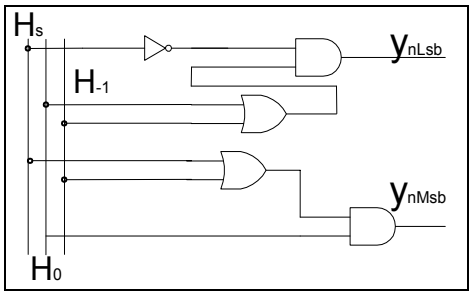


Figure 3.7 : Schéma logique du circuit de sélection

3.7.8. Architecture générale .

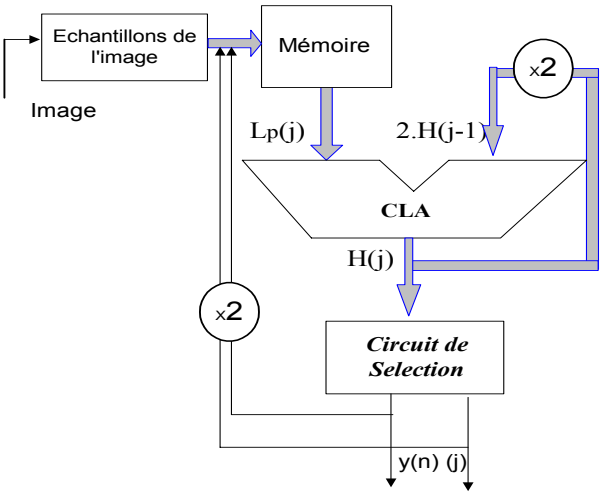


Figure 3.8 : Cellule de base de l'architecture

3.8. Procédé de lecture de l'image

Le calcul des convolutions lors de la transformée en ondelettes traite tous les pixels de l'image en deux dimensions. Ces différents calculs sont réalisés même pour les valeurs impaires qui vont être éliminés dans la phase de décimation.

Un autre procédé de calculs a été élaboré dans le but d'optimiser notre architecture, par l'utilisation d'une lecture matricielle. Celui-ci a été testé sous Matlab avec succès, il est défini comme suit :

Au lieu de calculer le pixel résultat de la première convolution intermédiaire avant décimation, nous avons fait le calcul en sélectionnant les pixels d'entrée nécessaire aux pixels de sortie, ceci nous a fait passer du calcul de n convolutions au calcul de n/4 convolutions.

Afin de définir la convolution matricielle, nous avons réalisé les opérations suivantes :

1. Extension de l'image du coté gauche et droit avec «(N-1)*dimension de l'image» de valeurs nulles, ou N est l'ordre du filtre.
2. Extension de l'image du coté supérieure avec «(N-1)*dimension de la nouvelle image» de valeurs nulles.
3. Accomplissement des convolutions en blocs avec un saut matriciel de « 2x2».

A chaque étape, un pixel est produit comme suit :

$$\text{Pixel} : [1x1] = \underbrace{[1 \quad . \quad . \quad . \quad N+1]}_{A} \underbrace{\begin{bmatrix} a+b+1 & . & . & . & a+b+N+1 \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & a+b+(N+1)+(N+1) \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} 1 \\ \\ \\ \\ N+1 \end{bmatrix}}_{C}$$

Le bloc A représente le filtre en forme vectorielle

Le bloc B représente le bloc en cours de lecture de l'image avec une dimension de (N+1)*(N+1)

1- La lecture suivante est effectuée en augmentant «*a*» de 2, *b* étant fixé jusqu'à la lecture horizontale complète.

2- Puis en augmentant «*b*» de 2 et en répétant la séquence de lecture horizontale jusqu'à compléter toute l'image

3- Le pixel obtenu est le pixel final de la première décomposition, pour un filtrage complet. Les quatre processus LL, LH, HL et HH sont effectués en même temps.

Cette méthode peut être très bien adaptée au calcul en ligne, car ceci permet la génération des bits du pixel final directement de l'image initiale en effectuant une convolution matricielle. Lorsque le premier bit est généré dans la première étape, il est injecté dans le second bloc pour commencer à générer les bits du pixel final, les opérations supplémentaires effectuées auparavant ainsi que la décimation sont éliminés par l'emploi d'un saut matriciel de (2 x 2) comme mentionné dans le schéma suivant :

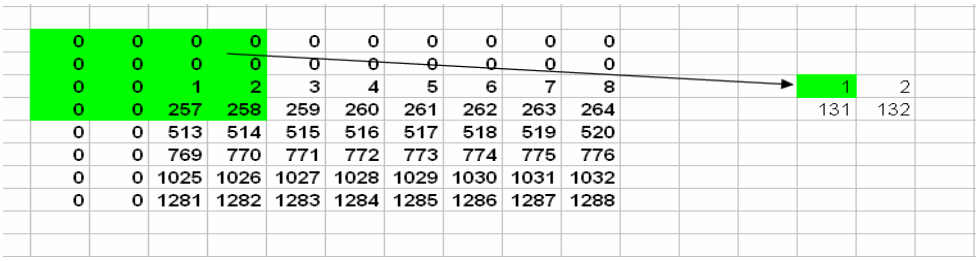


Figure 3.9 : Convolution matricielle du pixel résultat (1,1)

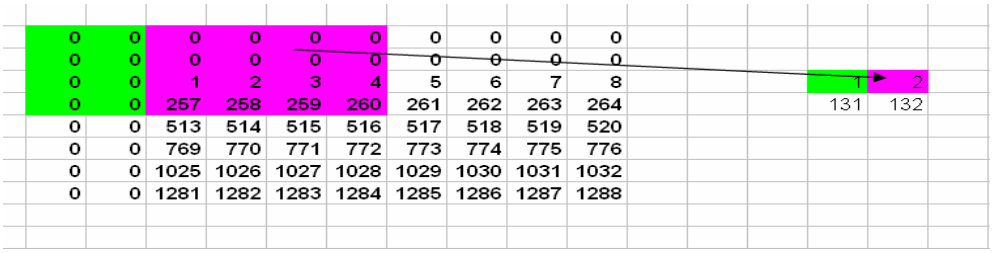


Figure 3.10 : Convolution matricielle du pixel résultat (1,2)

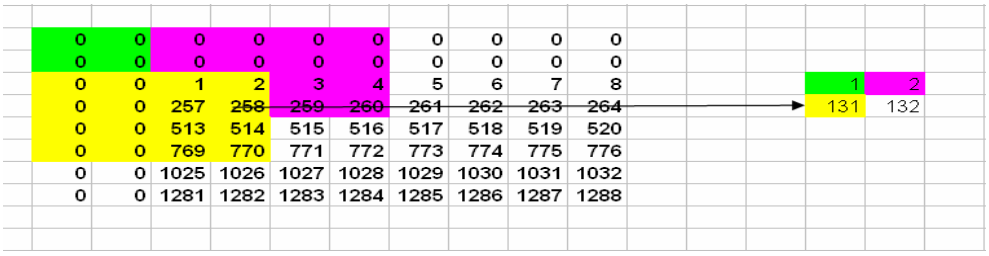


Figure 3.11 : Convolution matricielle du pixel résultat (2,1)

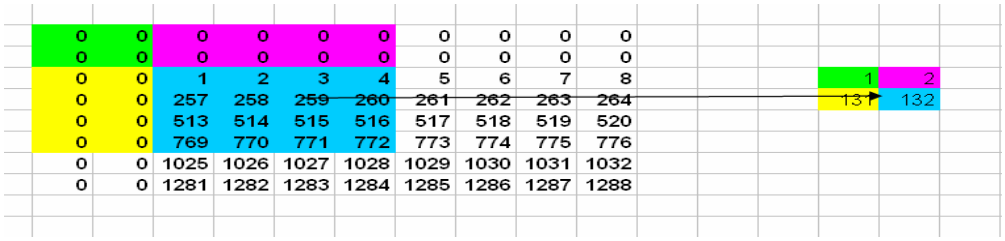


Figure 3.12 : Convolution matricielle du pixel résultat (2,2)

3.9. Ordre de complexité

Afin de comparer l'ordre de calcul avec la transformée en ondelettes réalisée sous Matlab, une image de 256 x 256 est utilisée avec un filtre bi orthogonal d'ordre 3, les calculs peuvent être généralisés à des images plus grandes.

NB : Une convolution linéaire est réalisée en 3 additions et 4 multiplications

En termes d'opérations élémentaires :

Tableau 3.13 : Complexité des calculs

	Décomposition	Reconstruction
Nombre de Convolutions	[260x260 +2 x(260 x 130)] x 2 = 270400	[4 x (130 x 260) + 2x 260x260)] = 270400
Multiplications	1081600	1081600
Additions	811200	811200
Additions hors convolutions	0	260x260+130x130x2=101400

Avec la méthode proposée :

Tableau 3.14 : Complexité de calculs de la méthode proposée

Nombre de Convolutions	Décomposition	Reconstruction
	[130 x130 x(4x2+4)] = 202800	(260x130+260x260)x4=405600
Multiplications	202800 x 4= 811200	405600x2=811200
Additions	202800 x 3= 608400	405600
Additions hors convolutions	0	260x260x3=202800

3.10. Architecture globale

L'architecture globale utilisant les blocs d'ondelettes en ligne sont présentés comme suit :

3.10.1. Bloc de décomposition

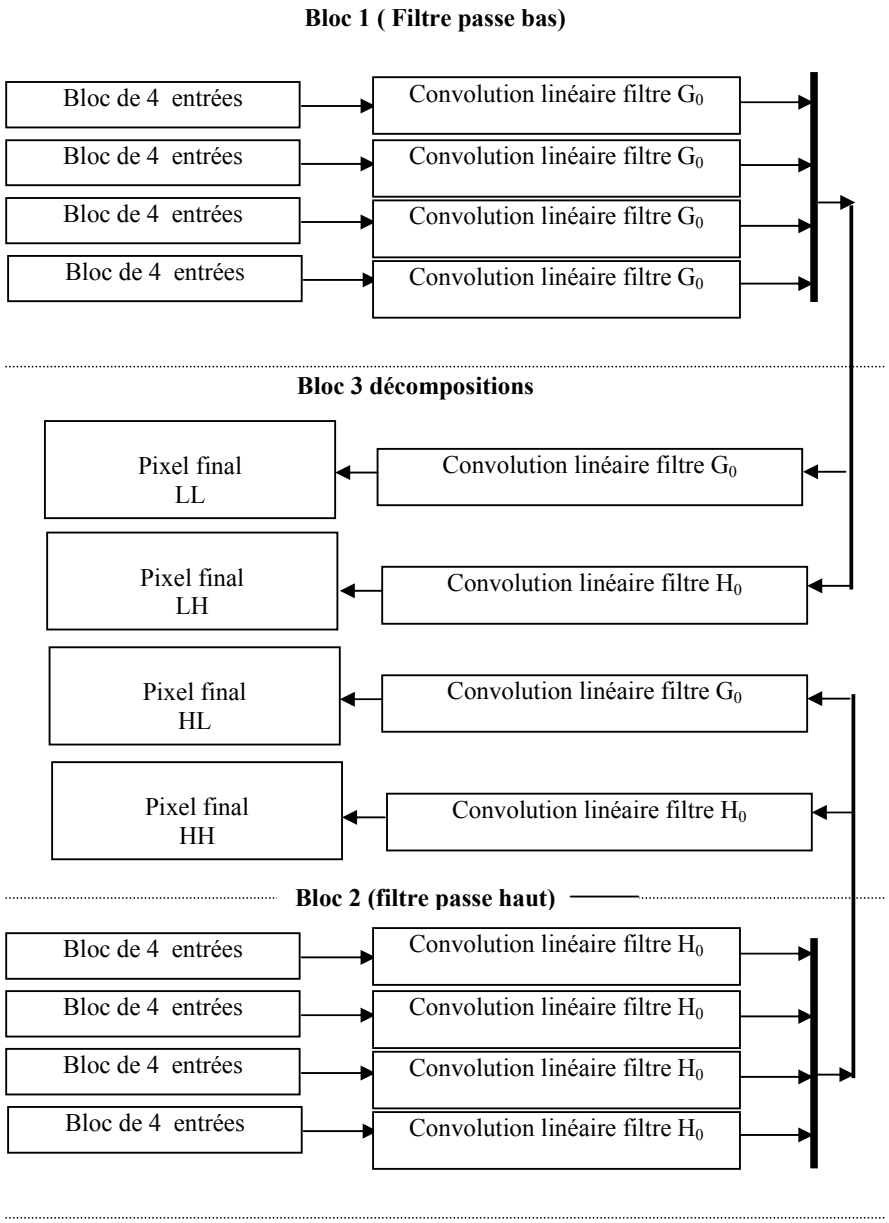


Figure3.13: Architecture de la décomposition

3.10.2. Bloc de Reconstruction

La reconstruction est le processus inverse de la décomposition, en se basant sur les bandes LL, LH, HL et HH, les tables de lecture ont été optimisées. Ceci est du essentiellement à l'insertion des zéros à la reconstruction, pour un filtre de 4

coefficients, 2 seulement sont utilisés, les deux autres sont des multiplications par des valeurs nulles, donc éliminés dès la phase de lecture.

De chaque sous bande LL, LH, HL,HH, 4 pixels sont lus et 16 convolutions sont effectuées en parallèle, le résultat de chaque 4 étages est additionné dans un additionneur en ligne pour aboutir à 4 points de l'image originale.

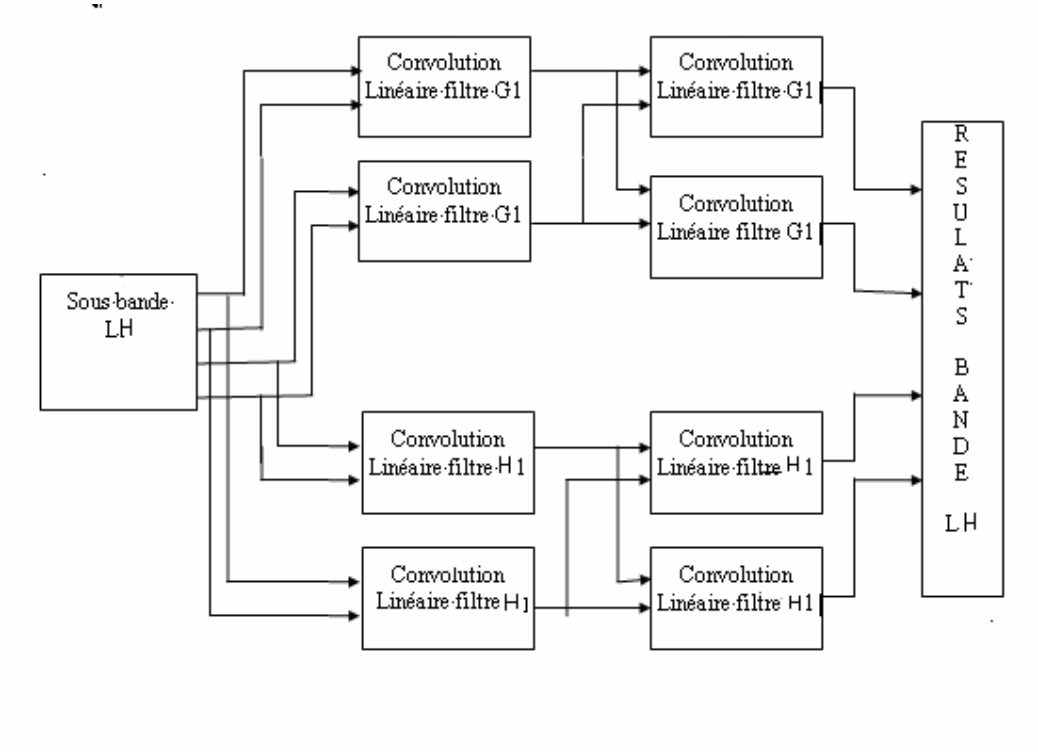


Figure3.14: Architecture de la reconstruction de la bande LH

La reconstruction de l'image originale est structurée de la manière suivante:

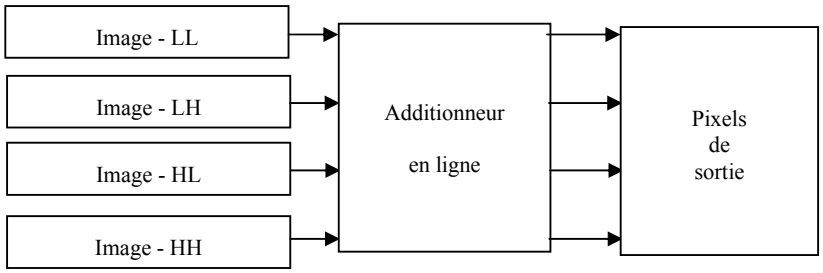


Figure3.15: **A**rchitecture complète des 4 bandes, phase de reconstruction

Tableau 3.15 : Nombre total de Cellules

	Nombre de cellules Décomposition	**Nombre de cellules Reconstruction**
Filtre passe bas	6	8
Filtre passe haut	6	8
Additionneur en ligne	0	4
Total des cellules	12	20

3.. Conclusion

Dans ce chapitre ; nous avons présenté plusieurs architectures qui ont fait l'objet d'implémentation sur circuit FPGA, ceci nous a permis de faire le point sur les différentes architectures existantes pour la transformée en ondelettes 2-D multi résolution (bancs de filtres) et de montrer que la mise en œuvre de ces implémentations est relativement complexe. Il faut trouver un bon compromis entre la rapidité de traitement et la surface nécessaire. Les nombres des points critiques présentés tout au long de ce chapitre sont liés à l'architecture employée pour la réalisation de la transformée en ondelettes. D'autre part, l'utilisation des bancs de filtres implique une gestion importante de la mémoire et nécessite des ressources de calcul importantes.

CHAPITRE 4
IMPLEMENTATION MATERIEL DE LA TRANSFORMEE EN ONDELETTES PAR L'ARITHMETIQUE EN LIGNE

4.1. Introduction

Les progrès technologiques continus dans le domaine des circuits intégrés, ont permis la réduction des coûts, de la consommation, et ont permis une réduction de la taille des systèmes numériques ainsi que la réalisation de circuits de plus en plus complexes, tout en améliorant leurs performances et leur fiabilité. Aujourd'hui, les techniques de traitement numérique occupent une place majeure dans tous les systèmes électroniques modernes grand public, professionnel ou de défense. De plus, les techniques de réalisation de circuits spécifiques, tant dans les aspects matériels que dans les aspects logiciels font de la microélectronique une des bases indispensables pour la réalisation de systèmes numériques performants. Elle impose néanmoins une méthodologie de développement en conception assistée très structurée.

Les concepteurs de circuit intégrés utilisent depuis longtemps la simulation, aucun ne peut se permettre de fabriquer une puce, de tester son fonctionnement, puis en cas de problème, d'effectuer la modification. Un simulateur permet de détecter des erreurs et les corriger avant la fabrication; La microélectronique est un des domaines où les erreurs coûtent le plus cher.

Lorsqu'un circuit est disponible l'expérimentation consiste à prendre des composantes, composer une maquette, la tester, l'adapter, la tester à nouveau. Les méthodes changent et de nouveau outils apparaissent, il fallait un langage descriptif de haut niveau pour la modélisation et la simulation. Avec l'apparition des outils de synthèse et des langages de description du matériel, la méthodologie de conception a beaucoup évalué. Cette nécessité s'imposa au département de la défense des Etats–Unis (DoD) au début des années 80 et donna naissance aux langages de description hardware comme le VHDL (**V**ery high Speed integrated circuits **H**ardware **D**escription **L**angage).

4.2. Les circuits à applications spécifiques ASIC

Les circuits ASIC regroupent tous les circuits, dont la fonction peut être personnalisée d'une manière ou d'une autre, en vue d'une application spécifique, par opposition aux circuits standards dont la fonction est définie et parfaitement décrite dans le catalogue de composants. Les ASIC peuvent être classés en plusieurs catégories selon leur niveau d'intégration [50], [51]. En fait un ASIC est défini par sa structure de base (réseau programmable, cellule de base, matrice, etc…). Sous le terme ASIC deux familles sont regroupées, les circuits semi-personnalisés et les personnalisés

Figure 4.1 : Famille des ASIC [52]

4.2.1 Les circuits semi-personnalisés

Les circuits semi-personnalisés sont des réseaux prédéfinis de transistors ou de fonctions logiques qui nécessitent une personnalisation de l'utilisateur pour réaliser la fonction désirée. Cette famille comprend :

- Les réseaux logiques programmables,
- Les réseaux prédiffusés.

4.2.1.1. Les réseaux logiques programmables

Ce composant ne nécessite aucune étape technologique supplémentaire [53] pour être personnalisé. Nous y trouvons les PAL/PLD, ce sont des circuits standard programmables par l'utilisateur grâce à différents outils de développement. La programmation consiste à établir des connexions en imposant un courant supérieur aux

courants de fonctionnement normaux (claquage de fusibles ou de jonctions). Les circuits logiques programmables incluent un grand nombre de solutions, toutes basées sur des variantes de l'architecture des portes ET et OU.
Nous y trouvons : ·

- PAL (Programmable Array Logic) matrice ET programmable, matrice OU figée
- PLA (Programmable Logic Array) matrice ET ou matrice OU programmable.

Ces composants très souples d'emploi sont limités à des fonctions numériques et adaptées à des productions de petites séries et ne présentent aucune garantie quant à la confidentialité.

4.2.1.2. Les circuits prédiffusés

Les réseaux prédiffusés sont des circuits partiellement préfabriqués. L'ensemble des éléments (transistors, diodes, résistances, capacités, etc.) est déjà implanté sur le circuit suivant une certaine topologie, mais les éléments ne sont pas connectés entre eux (sauf au niveau diffusion). La réalisation des connexions dans le but de définir la fonction souhaitée est à la tâche du concepteur. Pour cela, il dispose de bibliothèques de macro cellules et d'outils logiciels d'aide à la conception. A partir de cette liste d'interconnexions (netlist) le fondeur n'aura que quelques étapes technologiques à effectuer pour achever le circuit, c'est à dire le dépôt d'une ou plusieurs couches de métallisation.

Cette technique est intéressante sur le plan de la conception et de la fabrication, par contre elle présente l'inconvénient de ne pas permettre une optimisation en terme de densité de composants puisque les éléments de base sont pré implantés et pas forcément utilisés et que leur positionnement a priori n'est pas forcément optimal pour le but recherché.

4.2.2. Les circuits personnalisés

Ce sont des circuits non préfabriqués. Pour chaque application, on optimise le circuit intégré, ce qui conduit à la création de son propre composant. Cette famille comprend :

- les circuits à la demande,

- les circuits pré caractérisés.

4.2.2.1. Les circuits à la demande

Les solutions circuit à la demande (qu'on appelle full custom en anglais) présentent l'avantage d'autoriser une meilleure optimisation du placement, puisque celui-ci n'est pas prédéfini. On dispose d'une bibliothèque de modèles mathématiques de comportement et via notamment un compilateur de silicium, on peut concevoir toute l'architecture du circuit, en faire une validation logicielle (simulation logique) puis dans une avant dernière étape en déduire le dessin des divers masques de fabrication.

Toutes les opérations, de la conception à la fabrication, sont effectuées de façons spécifiques adaptées aux exigences de l'utilisation. L'ensemble des critères techniques est au choix du concepteur, que ce soit la taille du composant, le nombre de broches, le placement du moindre transistor. C'est l'ASIC le plus optimisé car aucune contrainte ne lui est imposé. Le placement des blocs fonctionnels et le routage des interconnexions, même si ces opérations sont assistées par ordinateur, sont effectués avec beaucoup plus d'interventions manuelles pour atteindre l'optimisation au niveau de chaque transistor.

Cependant, les phases de mise au point sont longues et onéreuses, il va de soi que la rentabilisation des investissements de développement nécessite un fort volume de production.

4.2.2.2. Les circuits pré caractérisés

Pour la réalisation de circuits pré caractérisés on dispose d'une bibliothèque de circuits élémentaires que le fondeur sait fabriquer et dont il peut garantir les caractéristiques et on les associe pour réaliser le circuit à la demande. Ici encore on utilisera en fabrication des masques personnalisés pour chacune des couches diffusées et des métallisations.

Le concept est très semblable à celui des circuits à la demande. La seule différence réside dans la réalisation du schéma puisque l'on accède à une bibliothèque de cellules prédéfinies générant de très nombreuses fonctions élémentaires ou élaborées.

Cette dernière constitue un véritable catalogue dans lequel le concepteur se sert pour constituer son schéma.

4.3. Avantages et inconvénients de l'utilisation d'ASIC

D'une manière générale, l'utilisation d'un ASIC conduit à de nombreux avantages provenant essentiellement de la réduction de la taille des systèmes. Parmi lesquels on citera:

- La réduction du nombre de composants sur le circuit imprimé. La consommation et l'encombrement s'en trouvent considérablement réduits.
- Le concept ASIC par définition assure une optimisation maximale du circuit à réaliser. Nous disposons alors d'un circuit intégré correspondant réellement à nos propres besoins.
- La personnalisation du circuit donne une confidentialité au concepteur et une protection industrielle.
- Enfin, ce type de composant augmente la complexité du circuit, sa vitesse de fonctionnement et sa fiabilité.
- L'inconvénient majeur réside dans le fait du passage obligatoire chez le fondeur ce qui implique des frais de développement élevés du circuit.

4.4. Description d'un FPGA (Field Programmable Gate Array)

Un FPGA consiste en une matrice carrée de cellules configurables CLB (Configurables Logique Blocs) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs d'entrées /sorties IOB (Input Output Blocs) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs. La figure 4.2 présente l'architecture générale de ces circuits.

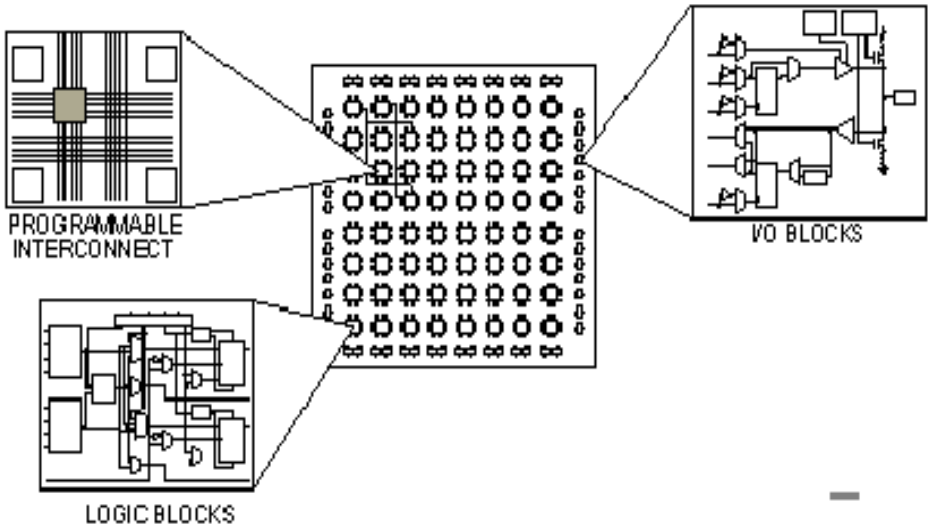


Figure .4.2 : Architecture générale d'un FPGA [54]

4.5. Les familles FPGA

Grâce aux évolutions de la technologie microélectronique les FPGA, deviennent de plus en plus performants avec des capacités sans cesse augmentées. Longtemps réalisés autour de blocs logique configurables à base de LUT (**L**ook **U**p **T**able), les FPGAs peuvent aujourd'hui comporter de larges mémoires RAM configurables des opérateurs arithmétiques complexes(comme les block select RAM et les block multiplieur du VIRTEX-II *de* XILINX) et des cœurs de microprocesseurs(tel que le VIRTEX-II pro) [55].

Les circuits FPGA de XILINIX sont caractérisés par une nomenclature spécifique qui définie les performances de chaque famille. Cette nomenclature est la suivante:

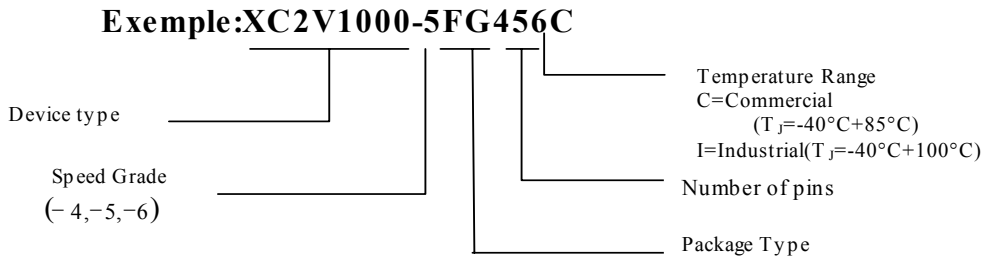


Figure 4.3. : Nomenclature d'un FPGA de XILINX

- **V**: "Device type" représente le type de famille, dans ce cas est : VIRTEX-II.
- **-5**: "Speed Grade" représente la vitesse du composant selon la technologie utilisée dans la fabrication du circuit.

- **456**: "Number of pins "représente le nombre de pins donnée un FPGA.
- **X**: représente la compagnie de fabrication XILIN***X***.
- **G**: "Package type" représente le type de boîtier.
- **C**: "Temperature Range" représente la gamme de température tolérée.

4.6. La famille VIRTEX-II

Cette famille a été conçue pour satisfaire les conditions demandées par les systèmes d'aujourd'hui. Elle à vu le jour grâce à la technologie d'intégration CMOS de 8 couches de métal ($0.15\mu m / 0.12\mu m$). Comparées aux familles FPGA, le VIRTEX-II est optimisé pour une haute densité et performance avec une grande capacité (jusqu'à 10 millions de portes). Actuellement, elle est utilisée dans plusieurs applications tels que: la télécommunication, les réseaux, la vidéo et les applications DSP (Digital Signal Processing).

Pour notre application, nous avons besoin d'une famille à très haute vitesse d'exécution. Dans ce but, nous avons choisi la famille VIRTEX-II.

4.6.1. Architecture interne de VIRTEX-II

Comme tout circuit FPGA, le VIRTEX-II est composé de blocs entrées /sorties IOB, de blocs logiques internes configurables voir appendice D, des ressources d'interconnexion et des lignes d'interconnexions.

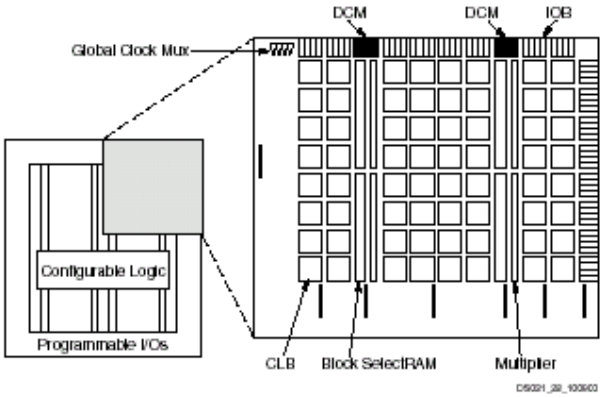


Figure 4.4: Architecture interne d'un VIRTEX-II [56]

4.6.2. Le bloc logique interne configurable

Le bloc logique interne configurable inclut quatre éléments majeurs. [57]

a) Multiplier bloc $(18bits\ x\ 18bits)$.

b) Bloc select RAM 18 bits.

c) DCM (**D**igital **C**lock **M**anager).

d) Bloc logique Configurable (CLB).

4.6.3. Les éléments logiques

Dans les Virtex l'élément logique de grain le plus fin est appelé un LC (Logic Cell) [58], il est composé d'une LUT (Look Up Table) un registre (pour synchroniser si nécessaire la sortie) est d'une chaîne de propagation rapide de la retenue (de ce fait on peut réaliser un full adder 1 bit par LC). Au grain logique supérieur l'élément est appelé un SLICE celui ci est composé de deux LC, voir appendice D, Puis les SLICE sont regroupés par deux pour former un CLB (Configurable Logic Blocks).

4.6.4. Bloc logique configurable (CLB)

Le CLB était l'élément déterminant des performances du FPGA. Mais pour le VIRTEX-II, ou on ne parle plus de CLB mais de slice, chaque CLB est composé de quatre slices (morceaux) et deux buffers à trois états.

Tous les slices sont identiques et contiennent :

- 2 générateurs de fonction (LUT F et LUT G).
- 2 éléments de stockage (bascule D).
- Des portes logiques arithmétique.
- Des multiplexeurs.
- Une chaîne rapide de logique de carry (retenue)
- Une chaîne cascadée des portes OR horizontale.

4.6.5. Bloc d'entrées/sorties (IOB)

Les blocs d'entrée/sortie fournissent l'interface entre les broches externes de module et la logique interne. Ils sont présents sur toute la périphérie du circuit FPGA.

4.7. Techniques de programmation des FPGA

Les circuits FPGA ne possèdent pas de programme résident. A chaque mise sous tension, il est nécessaire de les configurer. Leur configuration permet d'établir des interconnexions entre les CLB et les IOB. La configuration du circuit est mémorisée sur la couche réseau SRAM à partir d'une ROM externe. En effet, un dispositif interne permet à chaque mise sous tension de charger la SRAM interne à partir de la ROM.

Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive [59], [60].

Le format des données du fichier de configuration est produit automatiquement par le logiciel de développement sous forme d'un ensemble de bits organisés en champs de données.

4.8. Les étapes d'un développement en VHDL (Very High Speed Integrated Circuits Hardware Description Langage) [60], [61], [62].

4.8.1. Approche de conception

La conception, la description et la simulation des composants et systèmes en langage VHDL présentent deux aspects distincts pour un modèle.

- Une vue externe du circuit; appelée « Spécification »
- Une vue interne du circuit : appelée « Architecture »

Chacune de ces deux parties sera une unité de conception et ce sont ces unités qui constituent la bibliothèque VHDL. Une bibliothèque peut compter des centaines d'unités de conception séparées qui permettent de modifier facilement le fonctionnement interne.

Le VHDL est un langage très modulaire, on peut ne pas écrire de longues descriptions, mais des unités plus petites et hiérarchisées, certaines parties de ces descriptions peuvent être compilées séparément; Elles sont suffisantes en elles-mêmes pour être comprises. Ce sont les unités de conception. On peut dire que :

- Le circuit est considéré comme une entité (modèle)
- Cette entité peut être composée par d'autres entités.
- Chaque entité est composée de deux parties, interface et corps. Elle permet d'avoir plusieurs corps pour une même interface.

L'outil de simulation MODELSIM utilisé est l'un des logiciels de CAO les plus répandus, sur les plates formes de type PC pour la mise en œuvre des circuits programmables FPGA et qui réalise la simulation**.**

La conception se passe principalement en trois phases:

- La saisie du circuit.
- L'implémentation.
- La configuration du composant.

Auxquelles il faut ajouter les phases de vérification:

- La simulation fonctionnelle.
- La simulation temporelle.

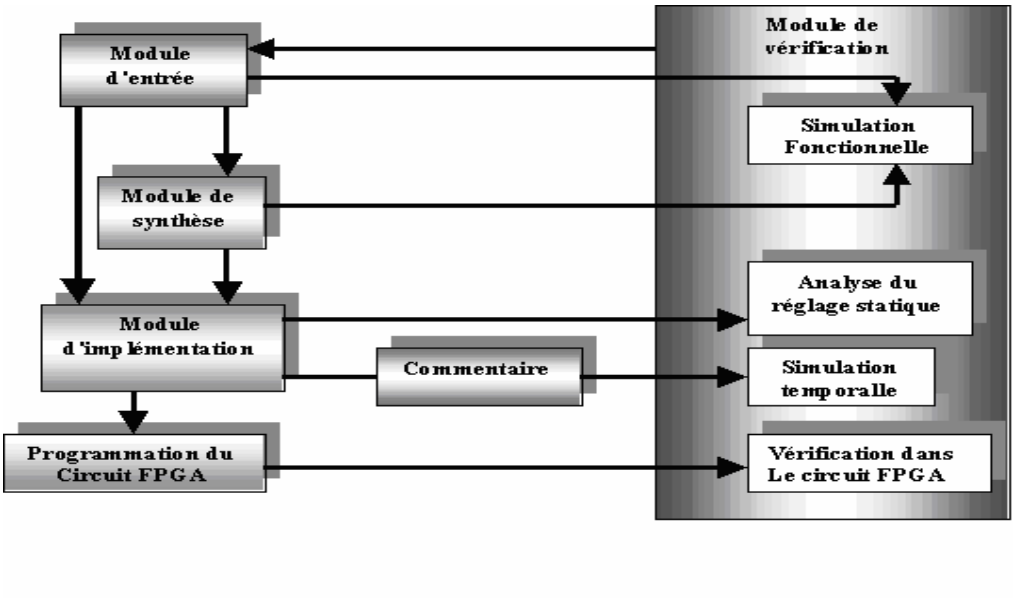


Figure 4.5 phases de conception sur FPGA

4.8.2. Module d'entrée "design entry"

Ce module permet la description d'une architecture ou d'un design soit en code VHDL, utilisation d'un éditeur de schéma ou bien par les diagrammes d'état.

4.8.3. Module de synthèse "Design Synthesis"

Ce module transforme la description VHDL en portes logiques (description proche des ressources matérielles) et procède à l'optimisation : les signaux inutilisés sont retirés, les expressions booléennes sont simplifiées, les signaux équivalents sont détectés.

4.8.4. Module d'implémentation de "Design Implementation"

Il réalise le mapping, c'est la projection des équations sur les différents blocs de circuit. Ainsi, il réalise le placement/ routage qui consiste à attribuer les blocs du circuit à chaque équation délivrée par la projection et à définir les connexions.

L'algorithme de placement place physiquement les différentes cellules et définit les chemins d'interconnexions dessinés entre les cellules afin de faciliter le routage.

4.8.5. Module de vérification "Design Verification"

Avec ce module, nous pouvons analyser le temps d'exécution (static timing analysis), et réaliser la simulation. Cette dernière déroule en parallèle chaque bloc pour vérifier que le design est opérationnel, et se fait en deux étapes:

4.9. Résultats des tests des images

Après la modélisation mathématique de l'algorithme de décomposition et de reconstruction de Stéphane Mallat par le mode de calcul en ligne et le développement des architectures correspondantes. Une description en VHDL et plusieurs simulations sous Modelsim et Matlab nous ont permis de juger de la faisabilité de cette architecture en mode en-ligne.

Cette architecture a été testée pour différentes images de différentes tailles, les temps de décomposition / reconstruction sont présentés dans le tableau suivant :

Tableau 4.1 : Timing de la simulation

Image	Modelsim simulation	
	Décomposition en ms	Reconstruction en ms
256 x 256	12.97	16.385
512 x 512	50.198	65.537
462 x 668	59.06	71.7
648 x 508	62.987	82.297
186 x 268	12.463	18.23
128 x 128	4.093	8.562

4.10. Timing de la simulation Hardware

Ces timings sont issus de l'implémentation sur circuits Xilinx Virtex II xcv2500-fg256-6 FPGA, sous Modelsim, ces temps seront ajustés lors des optimisations futures, les résultats des deux approches sont simulées avec un processeur Pentium IV 2.7Ghz, 512 Mo de Ram. Les programmes de simulation en VHDL et les résultats sont mentionnés dans les appendices E et F respectivement.

4.11. Les fréquences obtenues

Notre implémentation présente les fréquences suivantes :

Tableau 4.2 : Fréquences de la simulation

Décomposition	49.9 Mhz (Virtex II)
Reconstruction	34.829 Mhz (Virtex II)

4.12. Les résultats de la simulation

Lors de la phase de simulation, nous avons utilisé des fichiers de lecture issus de Matlab, les images en entrée ont été transformées de telle façon à faciliter la lecture en simulation, les fichiers images binaires sont réécrits sous une forme linéaire, ainsi que les fichiers en sortie issus de Modelsim, la lecture des résultats subit la même opération de réarrangement, les différentes bandes sont enregistrées chacune dans un fichier séparé.

- La reconstruction passe par la même procédure sous Matlab.
- Les résultats avant affichage sont normalisés dans les intervalles correspondants.
- Les programmes sous Matlab sont mentionnés dans l'appendice E.
- Les programmes en VHDL sont mentionnés dans l'appendice F.
- Les images testées sont mentionnées dans l'appendice G.
- Un exemple de chronogramme est mentionné dans l'appendice H.

.

4.13. Résultats d'implémentation

Tableau 4.3 : Ressources d'implémentation

	Décomposition	Reconstruction	Total	%	Disponible
Number of Slices	640	847	1487	48,40%	3072
Number of Slice Flip Flops	144	444	588	9,57%	6144
Number of 4 input LUT's	1180	1602	2782	45,28%	6144
Number of bonded IOBs	52	110	162	94,19%	172
Number of GCLKs	1	1	2	12,50%	16

4.14. Les schémas d'implémentation sur FPGA de Xilinx, Circuit Virtex2

Le schéma de l'implémentation sur circuit FPGA famille XC2V500 deVirtex-2 de Xilinx pour le processus de décomposition.

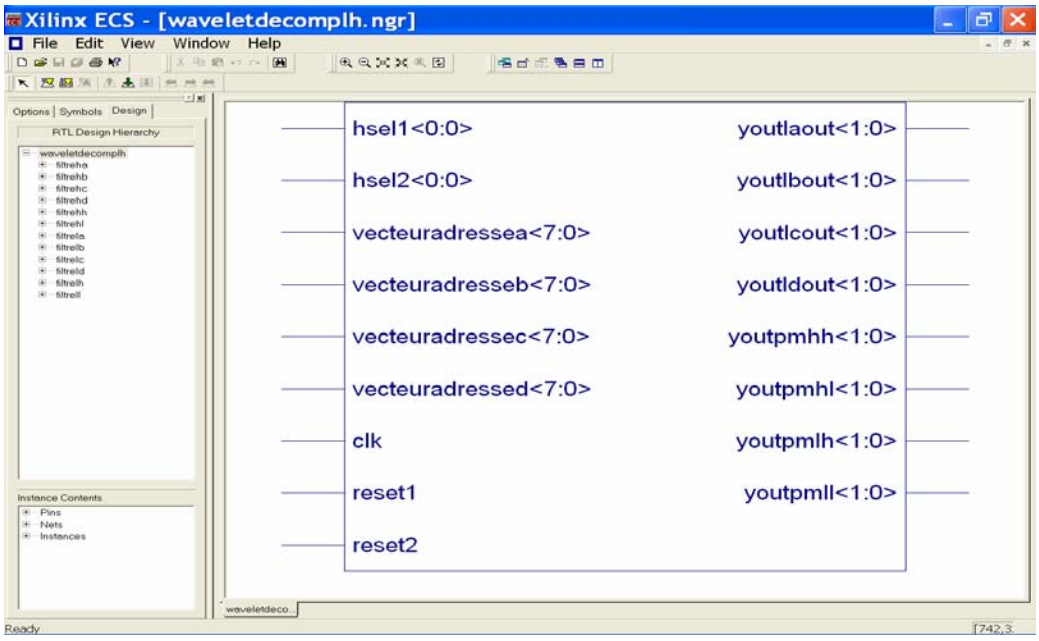


Figure 4.6. Schéma global de la décomposition

La figure 4.6 représente le schéma global de la décomposition en fonction des entrées sorties du bloc décomposition.

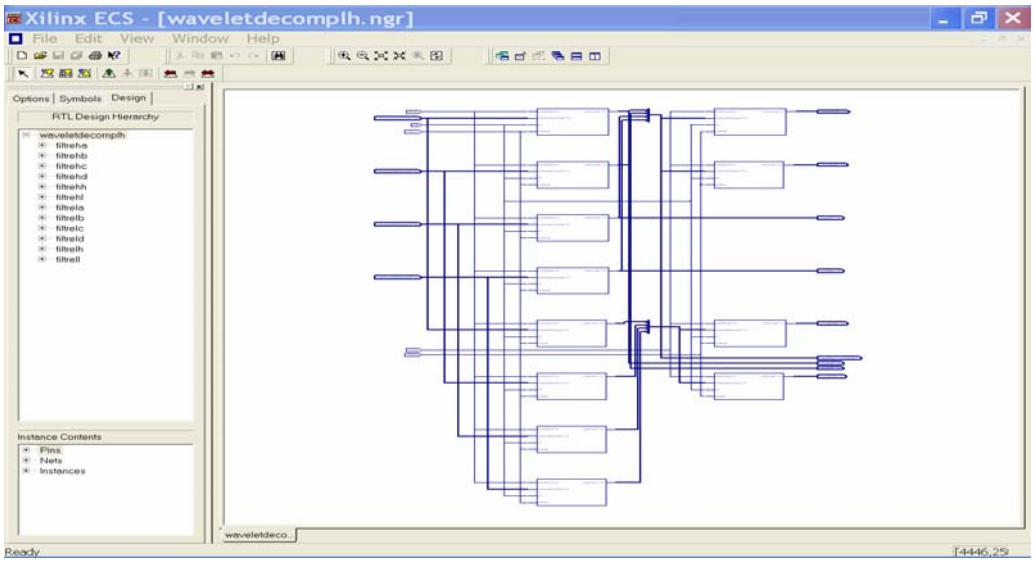


Figure 4.7. Schéma bloc de la décomposition.

Figure 4.7 présente la structure interne du bloc décomposition ou de l'architecture multidimensionnelle.

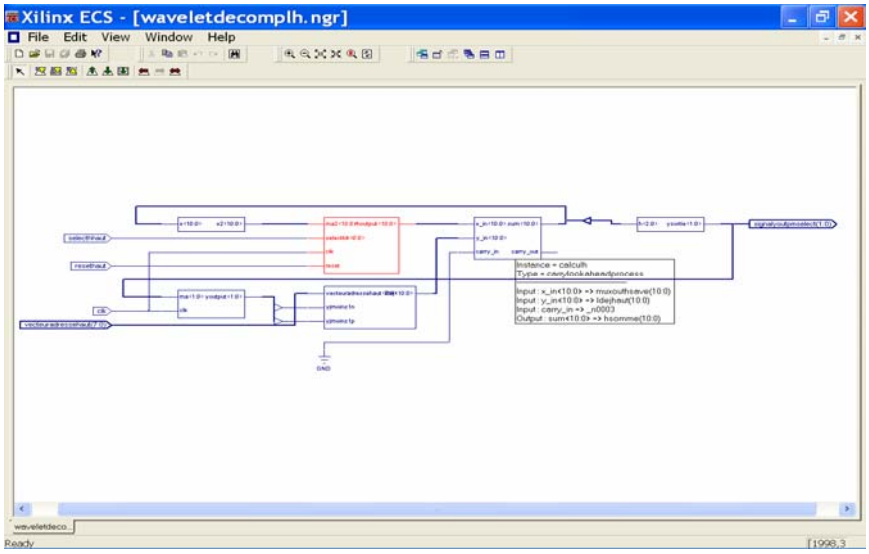


Figure 4.8. Schéma d'une cellule de base

Figure 4.8 présente la structure interne de l'architecture unidimensionnelle.

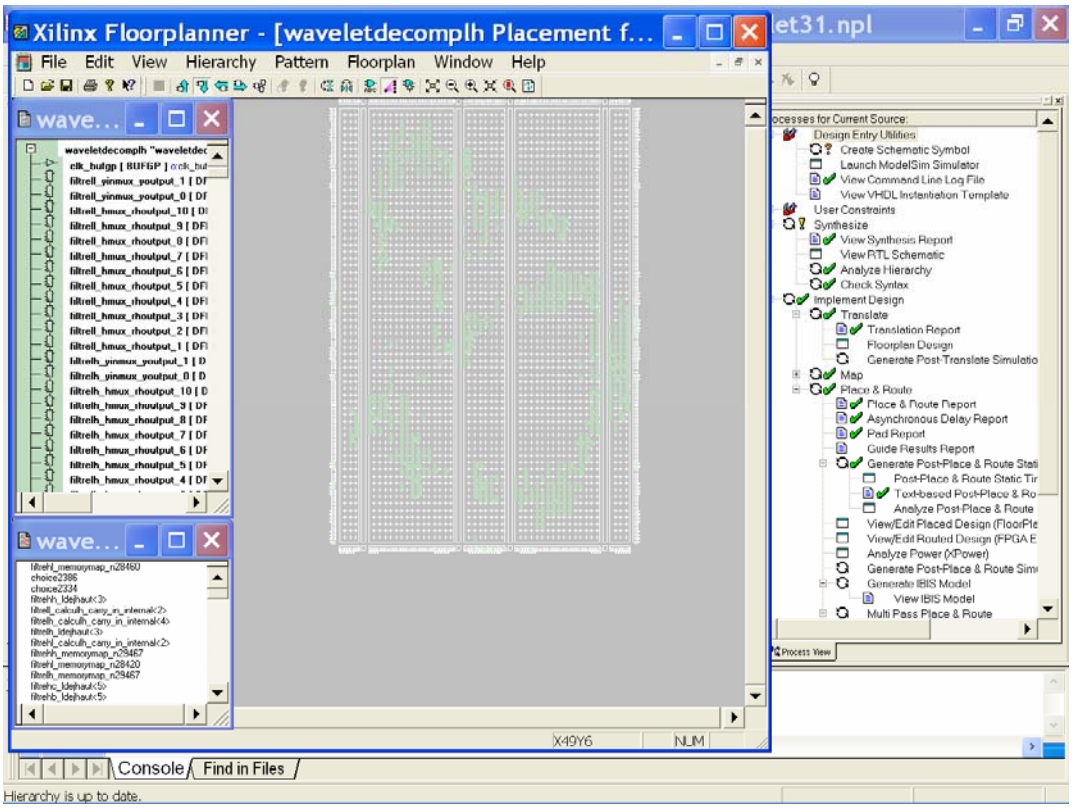


Figure 4.9. Layout du circuit de décomposition sur FPGA famille XC2V500 deVirtex-2 de Xilinx.

Le floorplanner nous délivre le Layout du circuit de décomposition sur FPGA famille XC2V500 deVirtex-2 de Xilinx ,figure 4.9.

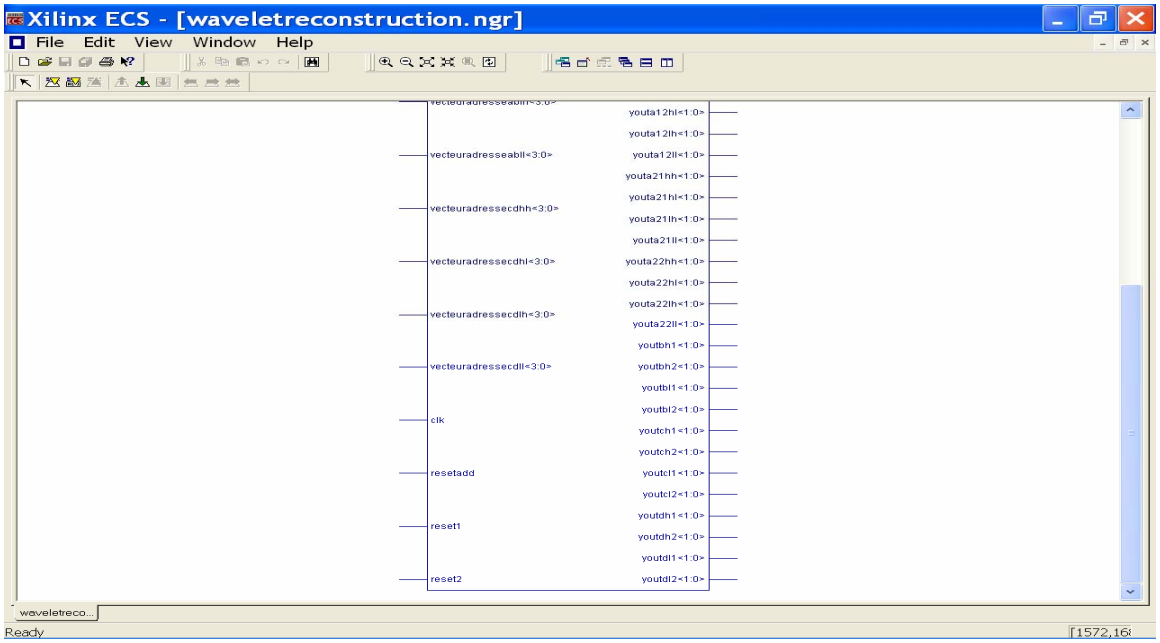


Figure 4.10. Schéma global Reconstruction

La figure 4.10 représente le schéma global de la décomposition en fonction des entrées sorties du bloc décomposition.

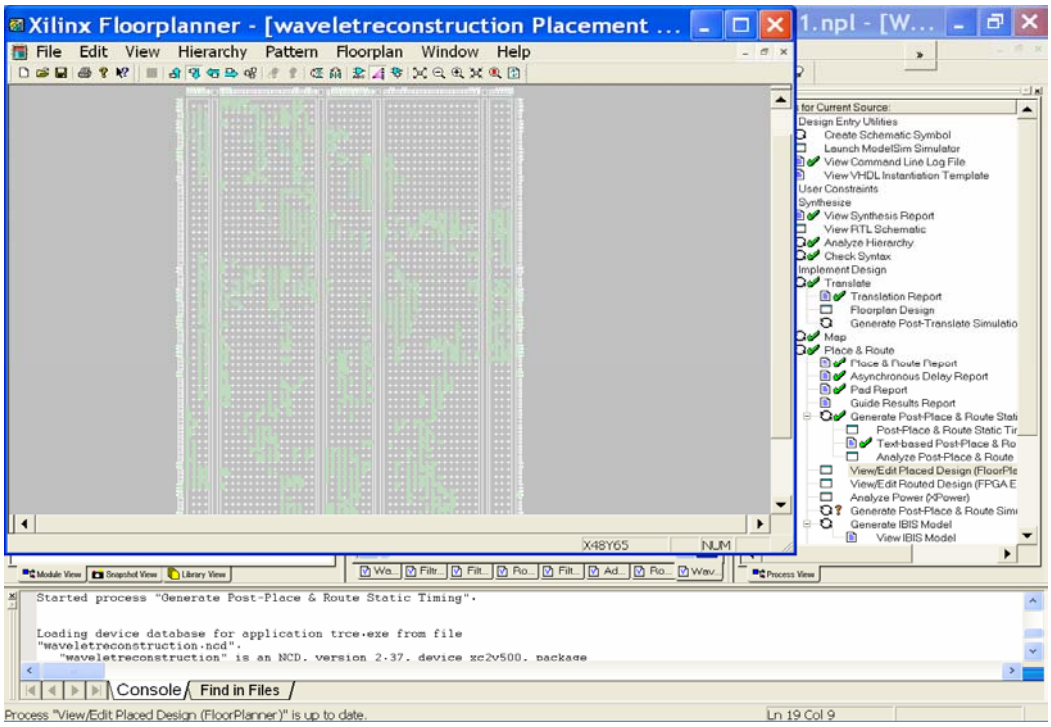


Figure 4.11. Layout du circuit de reconstruction sur FPGA famille XC2V500 deVirtex-2 de Xilinx

4.15. Les images testées

Plusieurs images sont testées permis ceux cette image médicale d'une coupe cervicale considéré comme référence.

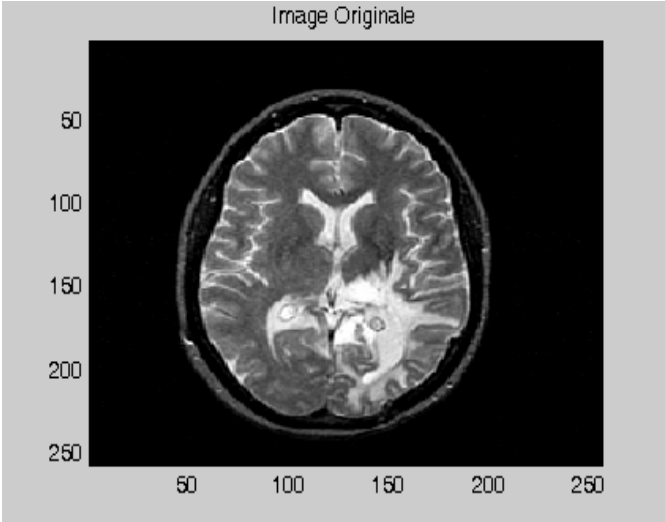


Figure 4.12. Image médicale d'une coupe cervicale (image originale)

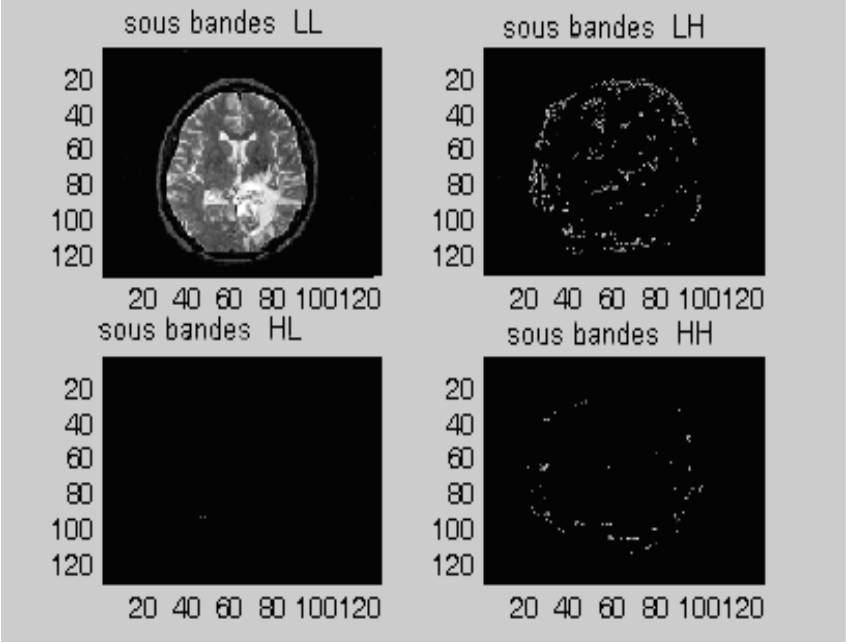


Figure 4.13. Résultats de décomposition sous Modelsim
Des 4 sous bandes .

Après décomposition de l'image originale en sous bandes LL(image approximatif), LH (image détail verticale), HL(image détail horizontal) et HH. (image détail diagonale) on obtient les résultats de la figure 4.13.

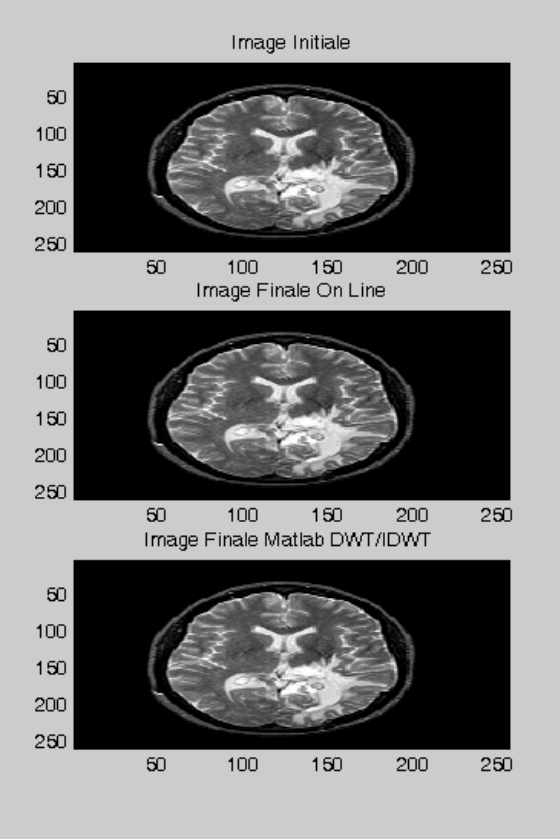


Figure 4.14. Résultats de la simulation de la reconstruction
de l'image originale sous Modelsim et Matlab

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de $8.5265.10^{-14}$.

Conclusion

L'évaluation des marchés pour des applications plus performantes en particulier dans le domaine des télécommunications, conduit à la réalisation de systèmes plus malléables et miniaturisés.les FPGA ont ouvert une énorme voie dans ce domaine.

Dans notre étude, nous avons implémenté l'algorithme de S.Mallat, cas du filtre bi orthogonale en faisant appel à une arithmétique spécifique telle que l'arithmétique en ligne.

Les résultats de la simulation nous ont montré une efficacité intéressante en terme de décomposition reconstruction des images. Plusieurs images sont testées exemple figure 4.14 avec un taux d'erreur de $8.5265.10^{-14}$. D'autres images sont mentionnées dans l'appendice G.

CONCLUSION

De nombreuses méthodes et transformées pour l'analyse du signal existent. L'une des techniques qui a acquis une attention particulière et une reconnaissance grandissante au sein de la communauté scientifique est la transformée en ondelettes. Celle ci est considérée comme une évolution de la transformée de Fourier.

Dans ce travail, nous avons développé la transformée en ondelettes en utilisant l'algorithme de décomposition et de reconstruction de Stéphane Mallat, par l'utilisation d'une lecture matricielle de l'image originale. Le processus de calcul de cet algorithme est caractérisé par une structure purement séquentielle. Dans cette perspective, le mode de calcul en ligne s'avère trés approprié pour la réalisation de cette structure. Ce travail est considéré comme une introduction de l'arithmétique en ligne pour le calcul de la transformée en ondelettes, et au développement de l'architecture et de l'algorithme correspondant, en utilisant un pipeline au niveau chiffre au lieu du calcul conventionnel.

Les étapes d'exécution de ce projet sont:

- La modélisation mathématique de l'algorithme de décomposition et de reconstruction Stéphane Mallat par le mode de calcul en ligne.
- Le développement de l'architecture correspondante.
- La validation des architectures par simulation en VHDL sous simulateur Modelsim 5.7.
- L'implémentation sur circuits FPGA de la famille Virtex II.

Nous sommes passée d'un niveau de représentation abstrait mathématique puis algorithmique de la transformée en ondelettes de l'algorithme de Stéphane-Mallat à une architecture multidimensionnelle à un niveau plus concret implémentation sur FPGA.

Ainsi, si la transformée en ondelettes est une technique efficace et relativement complexe à mettre en oeuvre par l'emploi de bancs de filtres, elle présente l'inconvénient d'être gourmande en ressources, aussi bien en mémoire qu'en surface.

Dans notre travail la quantité de mémoire nécessaire au calcul de la transformée est fortement réduite du fait du calcul « sur-place » des échantillons : un échantillon calculé vient

remplacer l'échantillon d'origine grâce au pipline au niveau du bit « propriété de l'arithmétique en ligne » .

La contrainte de qualité est très importante dans le domaine du traitement du signal, elle nous indique si les données obtenues représentent d'une manière acceptable ou exacte l'information traitée est un critère majeur dans les domaines de la compression d'image par exemple, où l'image après décompression doit être la plus proche possible de l'originale. Afin d'assurer une décomposition et une reconstruction sans pertes de l'image des filtres bi orthogonaux à coefficients symétriques ont été implémentés. Aussi lorsque les algorithmes sont implantés de manière logicielle, les valeurs traitées sont le plus souvent des valeurs réelles (virgule flottante) et représentent donc de manière quasi-parfaite le signal d'origine. D'autre part, lors des calculs, aucune approximation n'est nécessaire sur les valeurs traitées; il n'y a donc aucune perte d'information due au calcul.

Par contre, lors de l'implantation matérielle, nous sommes souvent obligés de faire un compromis entre la précision souhaitée (largeur du bus de données, valeurs codées en entiers, virgule fixe ou flottante) qui est régie par la précision des coefficients. Afin de minimiser ce dépassement, des troncatures ou arrondies sont effectuées, ceci induit à des erreurs de calcul.

Dans ce travail, nous avons évité un tel procédé de la manière suivante:

Lors du filtrage de l'image, les filtres utilisés travaillent en paires:

- Décomposition : (G0, G0), (G0, H0), (H0, G0), (H0, H0)
- Reconstruction : (G1, H1), (G1, G1), (H1, G1), (H1, H1)

Le produit croisé des filtres élimine la racine carrée, ceci amène le problème d'une précision infinie à une précision finie fixe sans erreur, d'où le terme $\sqrt{1/8}$ à chaque décomposition / reconstruction est remplacé par 1/8 qui n'est qu'un simple shift de 3 bits.

Le calcul des convolutions lors de la transformée en ondelettes traite tous les pixels de l'image en deux dimensions. Ces différents calculs sont réalisés même pour les valeurs impaires qui vont être éliminées dans la phase de décimation.

Dans notre travail, un autre procédé de calcul été élaboré dans le but d'optimiser notre architecture, par l'utilisation d'une lecture matricielle, elle est définie comme suit :

Au lieu de calculer le pixel résultat de la première convolution intermédiaire avant décimation, nous avons fait le calcul en sélectionnant les pixels d'entrée nécessaires au pixel de sortie, ceci nous a fait passer du calcul de *n* convolutions au calcul de *n/4* convolutions.

Les perspectives à envisager de ces travaux est de concevoir une architecture de telle manière qu'elle puisse être réemployée dans un système de traitement d'images complet; dans

un circuit pour la compression d'image par exemple, où on pourra insérer notre système de transformée en ondelettes entre la source et les modules de quantification/codage, de manière totalement transparente pour le système complet.

Ce travail fut l'objet de deux communications internationales.

- Conférence Internationale sur le Génie Électrique, CIGE04, SETIF, Novembre 2004.
- Third IEEE International Conference on Systems, Signals & Devices (SSD'05), Tunisia, Mars 2005.

APPENDICE A
ARITHMETIQUE EN LIGNE

Introduction

Lors de la conception d'opérateurs arithmétiques, deux modes de transmission des nombres s'affrontent: Le mode parallèle et le mode sériel. La majeure partie des circuits de calcul utilise le mode parallèle (figure 1a) : processeurs généraux, DSP, cartes graphiques. . . Toutefois, dans certaines applications; le nombre de broches d'un circuit intégré est limité, il peut constituer une contrainte sérieuse à l'intégration. Par exemple, une addition sur 64 bits est environ deux fois plus lente qu'une addition sur 32 bits avec un processeur 32 bits.

Une arithmétique sérielle peut être utilisée avantageusement. En effet, les opérateurs sériels sont souvent plus petits que leurs équivalents parallèles. La faible vitesse de calcul due au caractère séquentiel de cette arithmétique peut être contre-balancée par le parallélisme introduit par le pipeline au niveau du chiffre lors d'opérations successives. On a alors une décomposition des opérations en tout petits blocs comme dans le pipeline des processeurs récents pour en augmenter le débit.

En arithmétique sérielle, les nombres peuvent être transmis soit avec les poids forts en tête, soit avec les poids faibles en tête. L'arithmétique sérielle classique (figure 1.b), ou *((Least Significant Digit First))*, abrégée LSDF, parfois appelée arithmétique bit-série, permet de réaliser facilement l'addition et la multiplication. Par contre, les opérations de comparaison et division ne sont pas possibles directement, c'est-à-dire avant d'avoir reçu la totalité des chiffres des opérandes.

L'arithmétique en-ligne ou *((Most Significant Bit First))*, abégrée MSBF , est une arithmétique sérielle où les données circulent les poids forts en tête dans les opérateurs(figure 1.c). Cette arithmétique, basée sur un système redondant d'écriture des nombres permet d'effectuer toutes les opérations habituelles avec les poids forts en tête. L'arithmétique en-ligne a été utilisée avec succès pour des circuits de traitement du signal , les réseaux de neurones artificiels , le contrôle numérique de micro-systèmes ou encore la recherche de séquences dans des chaînes en informatique.

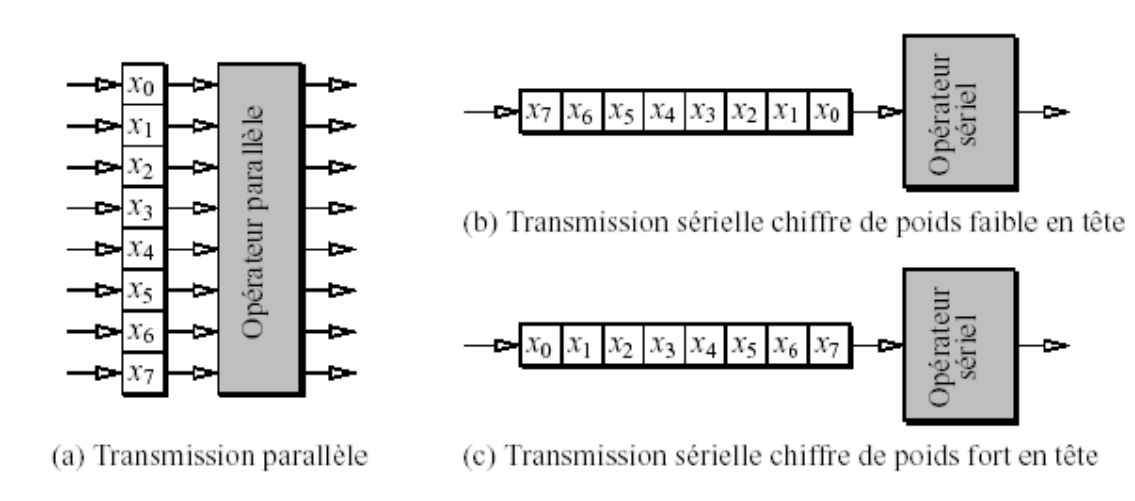


Figure 1. Divers modes de transmission d'un nombre entier de huit chiffres.

exemple si on veut calculer l'équation suivante $\sqrt{(a+b)+(c.d)}$:

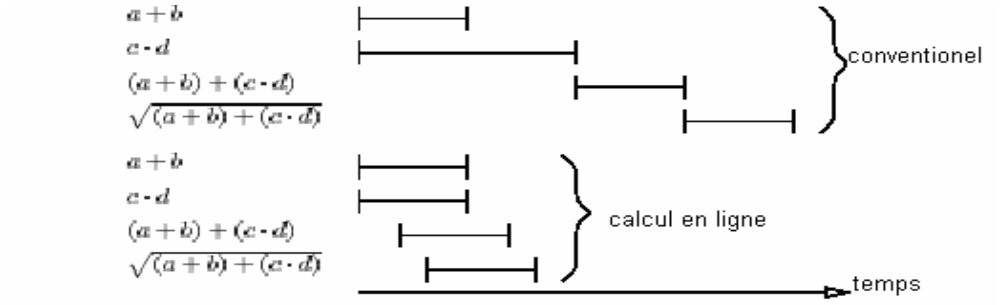
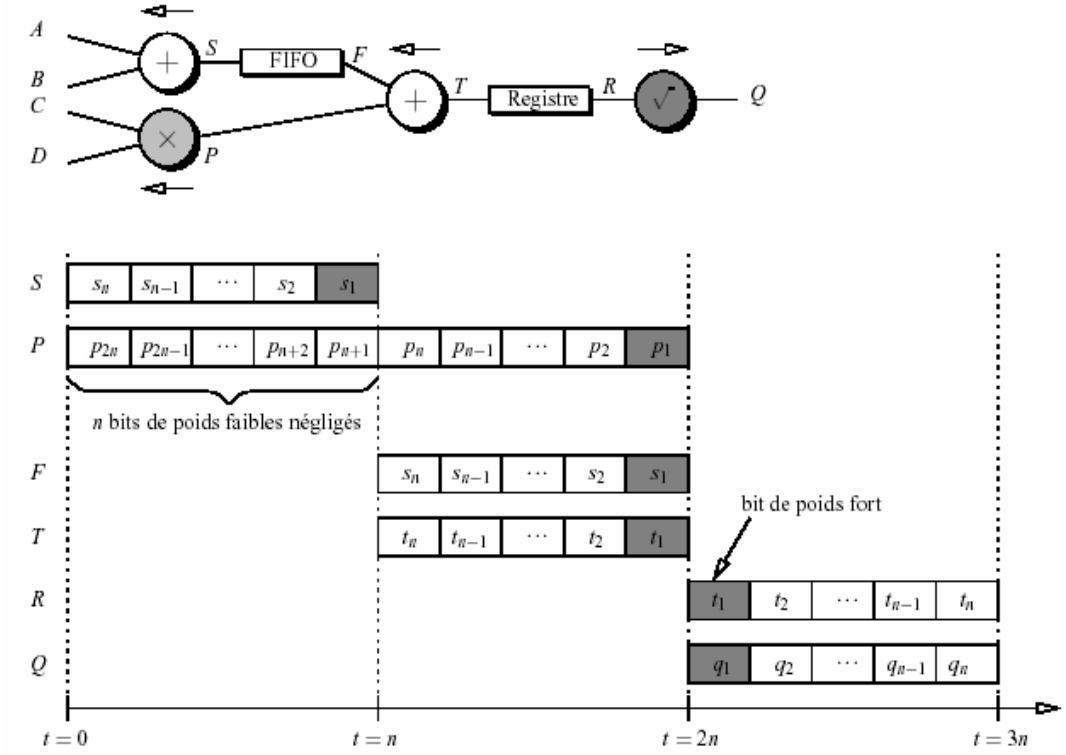


Figure 2 calcul de $\sqrt{(a+b)+(c\cdot d)}$.



a, *b*, *c* et *d* sont des nombres non signés purement fractionnaires de *n* bits transmis chiffre de poids faible en tête.Supposons encore le calcul effectué en virgule fixe. Au terme des *n* premiers cycles d'horloge, l'additionneur a établi la somme $s = a+b$. Le multiplicateur n'a toutefois déterminé que les *n* chiffres de poids faibles du produit $c \cdot d$. Un FIFO (First In First Out) retarde donc le résultat de l'addition, afin de le synchroniser avec les *n* chiffres significatifs du produit.

La racine carrée se calculant chiffre de poids fort en tête, la somme $(a+b)+(c \cdot d)$ est mémorisée dans un registre, puis transmise en mode MSDF au dernier opérateur. Nous constatons ainsi que 3*n*

cycles d'horloge se révèlent nécessaires à l'évaluation de la racine carrée de $(a+b)+(c \cdot d)$. De plus, un nouveau calcul ne peut débuter que lorsque la multiplication de *c* par *d* est achevée. Au moins *n* cycles d'horloge séparent par conséquent deux résultats successifs.

Autre exemple

$X = 8347$ et $Y = 1675$ et

Additionnons les en commençant par le chiffre de poids fort,figure 2. L'astuce consiste à exprimer $8 \cdot 10^3 + 1\ 10^3$ par $1\ 10^4 + \bar{1}\ 10^3$. Si une retenue provient de la colonne des centaines, elle sera ainsi annulée par $\bar{1} \cdot 10^3$ et ne pourra se propager plus loin. Le même principe régit le traitement des autres colonnes.

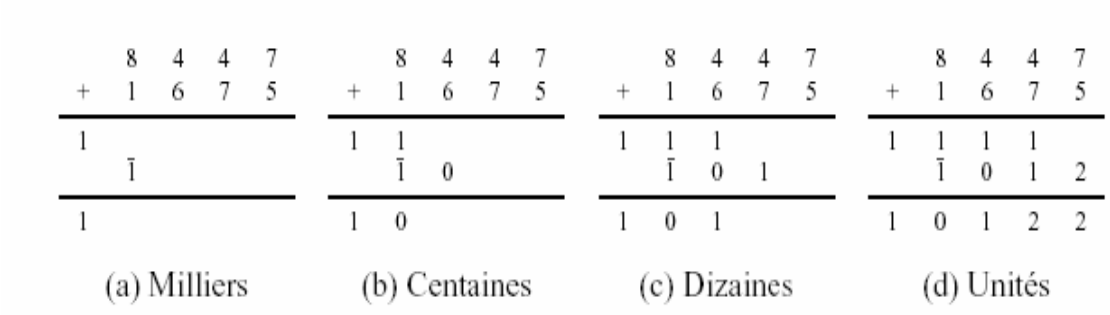


Figure 2.: Un exemple d'addition en temps constant
en base dix avec un ensemblede chiffres redondant.

Algorithme Avizienis

En 1961, A. Avizienis a proposé une méthode d'addition en temps constant représentations à chiffres signés. Si les chiffres des opérandes appartiennent à l'ensemble $D\rho = \{\bar{\rho}.....\rho\}$ tel que $2\rho \geq r+1$ et $\rho \leq r-1$, leur somme s'établit grâce à l'algorithme 3.1. Le coefficient *ti* intervenant dans le calcul est appelé chiffre de transfert (transfer digit). A. Avizienis a suggéré cette dénomination afin de distinguer *ti* d'une retenue apparaissant dans un algorithme d'addition traditionnel. L'algorithme utilise également des chiffres de sommes intermédiaires notés *wi*.

Algorithme 3.1 Algorithme d'addition en temps constant proposé par Avizienis.

Entrées : Deux nombres $X = 0.x1x2 \ldots xn$ et $Y = 0.y1y2 \ldots yn$ représentés en base *r* avec des chiffres appartenant à $D\rho = \{\rho,\rho+1, \ldots ,\rho-1,\rho\}$, où $2\rho \geq r+1$ et $\rho \leq r1$.

Résultat : Un nombre $S = s0.s1s2 \ldots sn = X + Y$.

- **Traitement :**

1- Initialisation : $w0 = tn = 0$

2- Pour i variant de 1 à n calculer **en parallèle**

Pour i variant de 0 à n calculer **en parallèle**

$$\begin{cases} t_{i-1} = \begin{cases} -1 & \text{si} \quad x_i + y_i < -\rho + 1 \\ 0 & \text{si} \quad -\rho + 1 \le x_i + y_i \le \rho - 1 \\ 1 & \text{si} \quad x_i + y_i > \rho - 1 \end{cases} \\ w_i = x_i + y_i - r \cdot t_{i-1} \end{cases}$$

3-Pour i variant de 0 à n calculer **en parallèle** $s_i = w_i + t_i$

Illustrons le déroulement de l'addition à l'aide d'un exemple. Considérons deux nombres $X = 0.952$ et $Y = 0.273$ codés en base dix avec $D10 = \{\bar{}9, \bar{}8, \ldots, 8, 9\}$. Un tel ensemble de chiffres satisfait évidemment les hypothèses requises par l'algorithme.

La figure 3, décrit le circuit effectuant le calcul. Il est facile de vérifier l'exactitude du résultat : $0.9\bar{}52 + 0.2\bar{}73 = 0.852 + 0.133 = 0.985 = 1.025$.

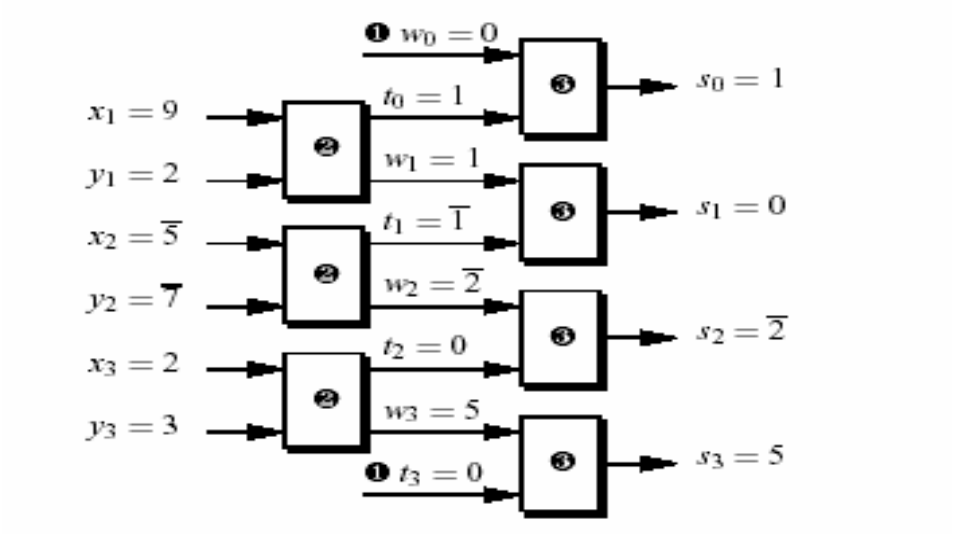
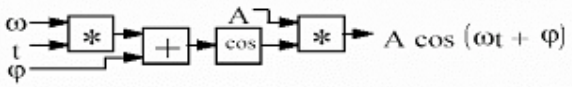


Figure 3: Exemple de fonctionnement de l'algorithme d'addition en temps constant d'Avizienis. Les numéros apparaissant sur le schéma correspondent aux différentes étapes de l'algorithme 3.1.

AVANTAGES DE L'ARITHMETIQUE EN LIGNE

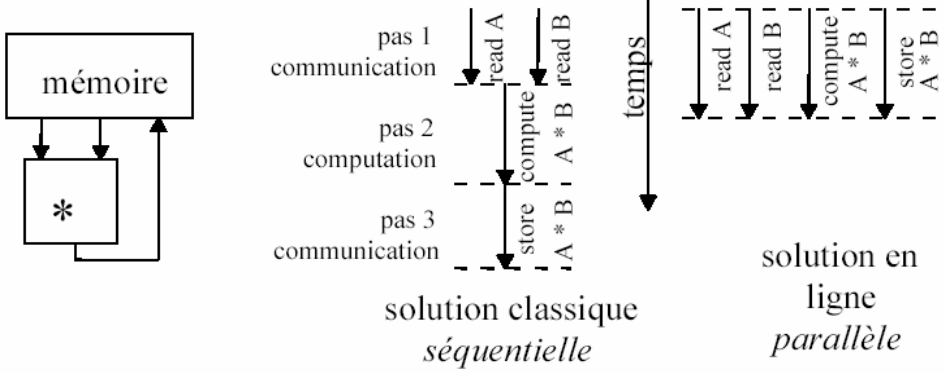
Avantage 1 : parallélisme à grain fin (niveau du chiffre)

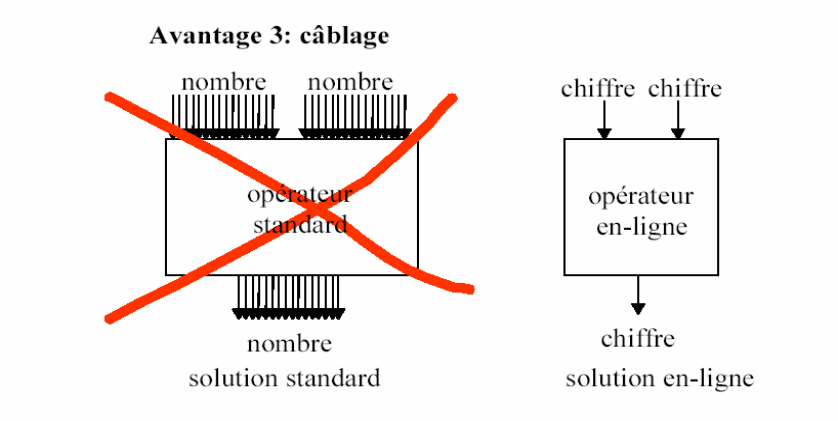


chaque opérateur travaille simultanément
(2 multiplieurs, 1 additionneur, 1 cosinus)

Avantage 2: la transmission recouvre l'exécution

exemple: multiplication de deux grands entiers A et B
(Les grands entiers doivent être transmis en série)




Avantage 3: câblage
nombre
nombre
opérateur
standard
nombre
solution standard
chiffre
chiffre
opérateur
en-ligne
chiffre
solution en-ligne

APPENDICE B

Tableau des valeurs
Pré- calculées de $L_p[j]$

```
LdejInt<="10111000"   when   "110101",  --  : valeur  =-72
          "10111010"   when   "110001",  --  : valeur  =-70
          "10111100"   when   "111101",  --  : valeur  =-68
          "10111110"   when   "000101",  --  : valeur  =-66
          "11000000"   when   "000001",  --  : valeur  =-64
          "11000010"   when   "001101",  --  : valeur  =-62
          "11000100"   when   "010101",  --  : valeur  =-60
          "11000110"   when   "010001",  --  : valeur  =-58
          "11001000"   when   "011101",  --  : valeur  =-56
          "11111000"   when   "110100",  --  : valeur  =-8
          "11111010"   when   "110000",  --  : valeur  =-6
          "11111100"   when   "111100",  --  : valeur  =-4
          "11111110"   when   "000100",  --  : valeur  =-2
          "00000000"   when   "000000",  --  : valeur  =0
          "00000010"   when   "001100",  --  : valeur  =2
          "00000100"   when   "010100",  --  : valeur  =4
          "00000110"   when   "010000",  --  : valeur  =6
          "00001000"   when   "011100",  --  : valeur  =8
          "00111000"   when   "110111",  --  : valeur  =56
          "00111010"   when   "110011",  --  : valeur  =58
          "00111100"   when   "111111",  --  : valeur  =60
          "00111110"   when   "000111",  --  : valeur  =62
          "01000000"   when   "000011",  --  : valeur  =64
          "01000010"   when   "001111",  --  : valeur  =66
          "01000100"   when   "010111",  --  : valeur  =68
          "01000110"   when   "010011",  --  : valeur  =70
          "01001000"   when   "011111",  --  : valeur  =72
```

APPENDICE C

ADDITIONNEUR A RETENUE ANTICIPEE

Additionneur à retenue anticipée (Carry Look Ahead CLA):

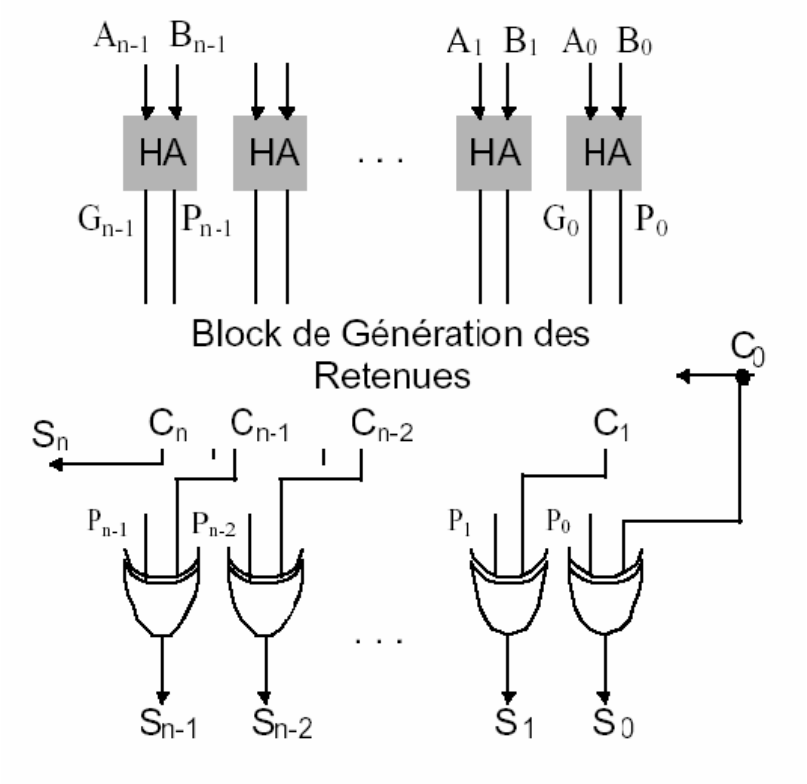


Schéma d'un additionneur à retenue anticipée « CLA » [66]

Avec

- $P_i = A_i \oplus B_i$ si $P_i=1$, la retenue précédente est propagée par l'étage i. [65],[66]

Et

- $G_i = A_i \cap B_i$ si $G_i=1$, l'étage i génère une retenue.

APPENDICE D
CONTROL LOGIC BLOCS « CLB »

Le « CLB » ou bloc logique interne configurable inclut quatre éléments majeurs. [60]

- Multiplier bloc $(18bit\,x\,18bit)$.
- Bloc select RAM 18 bits.
- DCM (**D**igital **C**lock **M**anager).
- Bloc logique Configurable (CLB).

Les éléments logiques

Dans les Virtex-2 l'élément logique de grain le plus fin est appelé un LC (*Logic Cell*) il est composé d'une LUT (*Loock Up Table*) un registre (pour synchroniser si nécessaire la sortie) est d'une chaîne de propagation rapide de la retenue (de ce fait on peut réaliser un full adder 1 bit par LC).

Au grain logique supérieur l'élément est appelé un SLICE celui ci est composé de deux LC, Puis les SLICE sont regroupé par deux pour former un CLB (*Configurable Logic Blocks*).

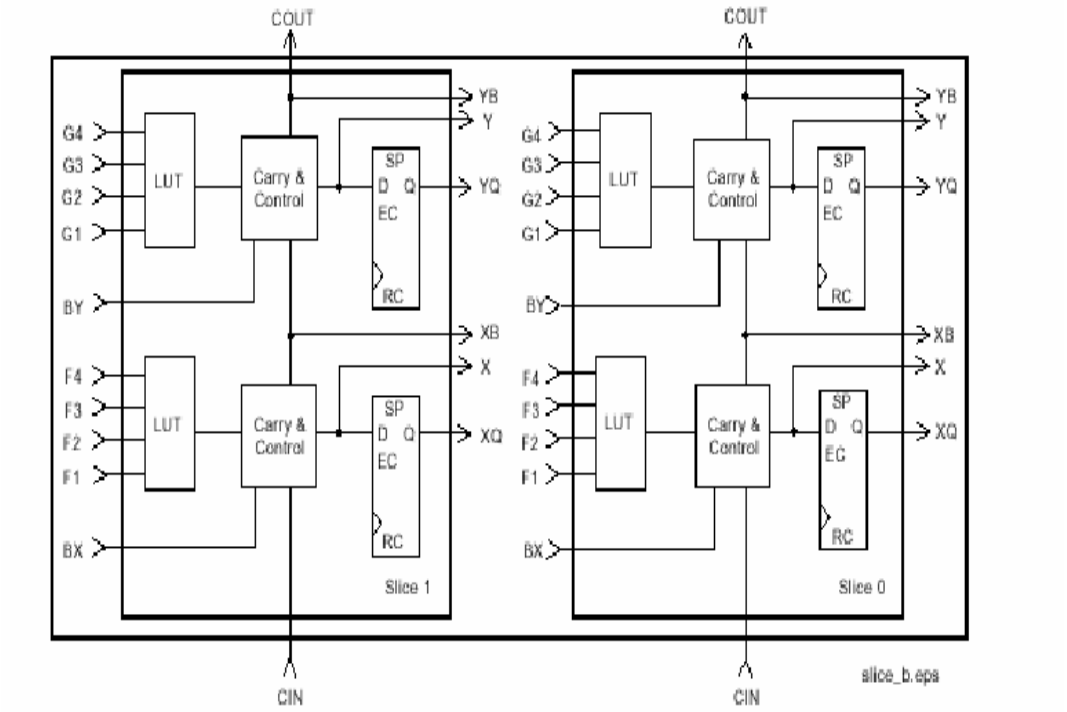


Schéma d'un CLB de VIRTEX-2

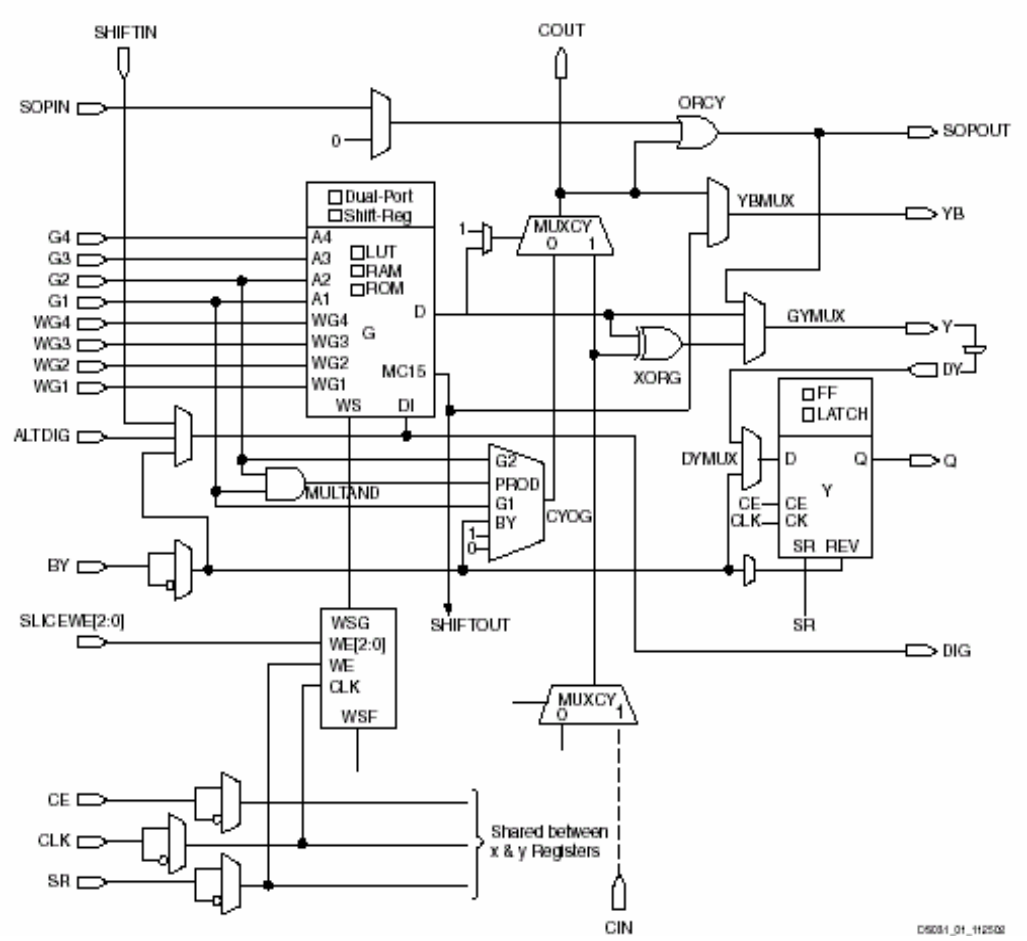


Schéma détaillé d'un slice d'un CLB de VIRTEX-2.

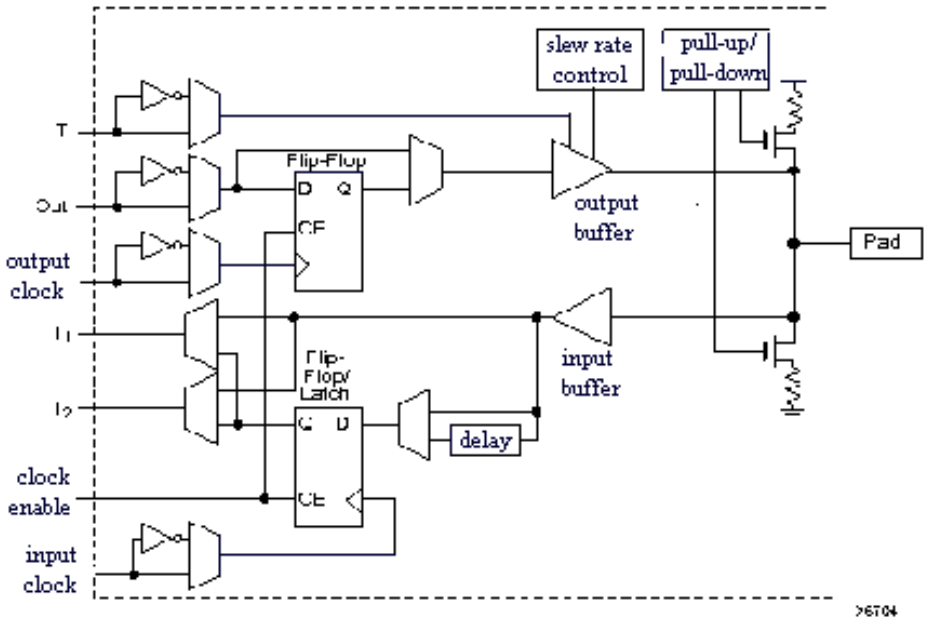


Schéma d'un IOB de VIRTEX-2

Chaque IOB contient

- Trois bascules D contrôlées par trois signaux d'entrés (nommés).
- Un set/reset (SR) et un signal d'horloge CLK.
- Un circuit de protection contre les dégâts électrostatiques, les décharges et les coupures ou surtensions, composé de résistances et de diodes. Il contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisée (haute impédance).

Le bloc IOB joue le rôle d'interface de communication, il permet de configurer les broches du circuit FPGA selon les options suivantes:

- Input/Output.
- TBUF (tapon à trois états logiques: On, Off, Haute impédance).
- Inverseur.
- Bascules flip flop.

APPENDICE E

PROGRAMMES SOUS MATLAB

TRANSFORMATION DES IMAGES ET AFFICHAGE

```
global NomWavelet Directory MyImage Mymap
global Tailleg0 Tailleh0 Tailleg1 Tailleh1
global g0 h0 g1 h1
global NomFichierLineaire
global a b a2 b2
global scale1
%L'image : : : : : Choix de l'image
%       Case 0 double(zeros(sizeImage,sizeImage));
%       Case 1 load clown
%       Case 2 load durer
%       Case 3 imread('029.gif');
%       Case 4 load woman
%MEDICALE Case 5 imread('medpix22544.jpg');
%MEDICALE Case 6 imread('021','gif');
%MEDICALE Case 7 imread('medpix225443','gif');
%MEDICALE Case 8 imread('angio1.gif');
clc;
clear;
dwtmode('zpd'); %% Insertion de zéros lors des ...
            %% extensions d'images
% Image à selectionner pour le traitement
   MonImage=3
% Image à selectionner pour le traitement
   [MyImage,Mymap]=GenererImage(MonImage,10);
   [a b]=size(MyImage)
%% Selection du filtre:
NomWavelet='bior3.1'
[g0,h0,g1,h1]=wfilters(NomWavelet);
scale1=1/sqrt(1/8);
%% Transformation du filtre
g0=g0*scale1;h0=h0*scale1;
g1=g1*scale1;h1=h1*scale1;
%% Transformation du filtre
Tailleg0=length(g0);Tailleh0=length(h0);
Tailleg1=length(g1);Tailleh1=length(h1);
ai=1:2:(a+(Tailleg0-2)*2)-(Tailleg0-1);
[si a2]=size(ai);
bi=1:2:(b+(Tailleg0-2)*2)-(Tailleg0-1);
[si b2]=size(bi);
%  Project Directory
Directory='C:\MyProject\BiorWavelet31\';
%% Fichier Initial de l'image
NomFichierLineaire='ImageLin.txt';
%% Fichier Initial de l'image
```

```
FichierSortie=[Directory NomFichierLineaire];
%% Fichiers issus de la Decomposition
FichierSortieLL=strcat(Directory,'ImageLL.txt');
FichierSortieLH=strcat(Directory,'ImageLH.txt');
FichierSortieHL=strcat(Directory,'ImageHL.txt');
FichierSortieHH=strcat(Directory,'ImageHH.txt');
%% Fichiers Retransformés par Matlab pour la Simulation
%% de la Reconstruction
%  NomFichierLLInput='ImageLL.txt';
%% LL
        NomFichierLL_Lin ='ImageLL_Lin.txt';
        FichierLL_Lin=[Directory NomFichierLL_Lin];
%% LH
        NomFichierLH_Lin ='ImageLH_Lin.txt';
        FichierLH_Lin=[Directory NomFichierLH_Lin];
%% HL
        NomFichierHL_Lin ='ImageHL_Lin.txt';
        FichierHL_Lin=[Directory NomFichierHL_Lin];
% HH
        NomFichierHH_Lin ='ImageHH_Lin.txt';
        FichierHH_Lin=[Directory NomFichierHH_Lin];
%% Fichiers issus de la Reconstruction
        FichierLL_AA=[Directory 'ImageLL_AA.txt'];
        FichierLH_AA=[Directory 'ImageLH_AA.txt'];
        FichierHL_AA=[Directory 'ImageHL_AA.txt'];
        FichierHH_AA=[Directory 'ImageHH_AA.txt'];
        ImageOA=[Directory 'imagefinale.txt']
%% Reconstruction
TailleConversion=24;
% les pixels sont transformés de 8 à 24 bits
% pour compenser les shifts de la normalisation
%% Reconstruction
Decomposition
display('Press a key for reconstruction');
pause;
Reconstruction
%%% Temps Réels
```

REARRANGEMENT DE L'IMAGE POUR SIMULATION

```
[Imlin,ImRed]=LinearIserImage(MyImage, FichierSortie, Tailleg0, Tailleh0,'d');
```

TRANSFORMEE EN ONDELETTES SOUS MATLAB : COMPARATIF

```
[ca ch cv cd]=dwt2(MyImage,g0,h0);
```

REARRANGEMENT DE L'IMAGE APRES SIMULATION

```
[ImageLLSim,ImageLHSim,ImageHLSim,ImageHHSim,LLLin,LHLin,HLLin,HHLin]=...
LectureResultat(FichierSortieLL,FichierSortieLH,FichierSortieHL,FichierSortieHH,a2,b2,'d');
```

FONCTION DE LECTURE DES RESULTATS

```
function [LLSim,LHSim,HLSim,HHSim,LLLin,LHLin,HLLin,HHLin,mode]=...
        LectureResultat(FichierEntreeLL,FichierEntreeLH,FichierEntreeHL,FichierEntreeHH,
        a2,b2,mode);
%%%%%%%%    LECTURE DU RESULTAT
switch mode
   case 'd'
      %% Sqrt(1/8)^2 pour le produit des deux filtres
echelleLL=(sqrt(1/8)^2)/(4);  echelleLH=(sqrt(1/8)^2)/(4);
echelleHL=(sqrt(1/8)^2)/(4);          echelleHH=(sqrt(1/8)^2)/(4);
Nombrelignes=a2;                      NombreColonnes=b2;
   case 'r'
     echelleLL=1;              echelleLH=1;        echelleHL=1;      echelleHH=1;
     Nombrelignes=2*a2;        NombreColonnes=2*b2;
   case 'e'
    echelleLL=1;               echelleLH=1;        echelleHL=1;      echelleHH=1;
    Nombrelignes=a2;           NombreColonnes=b2;
end

%% Decomposition /  Reconstruction
%% Bande LL
   fid=fopen(FichierEntreeLL,'r');    [LL_Lin,cnt]=fscanf(fid,'%d',inf);    fclose(fid);
%% Bande LH
   fid=fopen(FichierEntreeLH,'r');    [LH_Lin,cnt]=fscanf(fid,'%d',inf);    fclose(fid);
%% Bande HL
   fid=fopen(FichierEntreeHL,'r');    [HL_Lin,cnt]=fscanf(fid,'%d',inf);    fclose(fid);
% % %% Bande HH
   fid=fopen(FichierEntreeHH,'r');    [HH_Lin,cnt]=fscanf(fid,'%d',inf);    fclose(fid);
   %% Mise à L'échelle
   LLLin=LL_Lin*echelleLL;            LHLin=LH_Lin*echelleLH;
   HLLin=HL_Lin*echelleHL;            HHLin=HH_Lin*echelleHH;
switch mode
      case 'd'
         LLSim=reshape(LLLin,Nombrelignes,NombreColonnes);
         LHSim=reshape(LHLin,Nombrelignes,NombreColonnes);
         HLSim=reshape(HLLin,Nombrelignes,NombreColonnes);
         HHSim=reshape(HHLin,Nombrelignes,NombreColonnes);
      case 'e'
         LLSim=reshape(LLLin,Nombrelignes,NombreColonnes);
         LHSim=reshape(LHLin,Nombrelignes,NombreColonnes);
         HLSim=reshape(HLLin,Nombrelignes,NombreColonnes);
         HHSim=reshape(HHLin,Nombrelignes,NombreColonnes);
      case 'r'
         LLSim=reshape(LLLin,Nombrelignes,NombreColonnes)';
         LHSim=reshape(LHLin,Nombrelignes,NombreColonnes)';
         HLSim=reshape(HLLin,Nombrelignes,NombreColonnes)';
         HHSim=reshape(HHLin,Nombrelignes,NombreColonnes)';
   end;                    return
```

PROCESS DE RECONSTRUCTION

```
%%%%%%% Lecture des fichiers générés pas MODELSIM
%%%%%%% Sous forme de Matrices
[LLSim,LHSim,HLSim,HHSim,LLLin,LHLin,HLLin,HHLin]=...
LectureResultat(FichierSortieLL, FichierSortieLH, FichierSortieHL,FichierSortieHH,
a2,b2,'e');
```

REARRANGEMENT DE L'IMAGE POUR RECONSTRUCTION

```
[ImLL_Lin,ImLLRed]=LineariserImage(LLSim,FichierLL_Lin,Tailleg1,Tailleh1,'r');
 [ImLH_Lin,ImLHRed]=LineariserImage(LHSim,FichierLH_Lin,Tailleg1,Tailleh1,'r');
[ImHL_Lin,ImHLRed]=LineariserImage(HLSim,FichierHL_Lin,Tailleg1,Tailleh1,'r');
[ImHH_Lin,ImHHRed]=LineariserImage(HHSim,FichierHH_Lin,Tailleg1,Tailleh1,'r');
```

RECONSTRUCTION COMPLETE

```
   fid=fopen(ImageOA,'r');
   [Temp0,cnt]=fscanf(fid,'%d',inf);   fclose(fid);
   scale2=sym(scale1)%% = sqrt(8)
   FacteurEchelle=(2^3)*8;%% =sqrt(8)^2 pour eviter le sqrt(8)
   Temp1=Temp0/FacteurEchelle;
   k=1;
   clear A;
   for i=1:2:a;
      for j=1:2:b;
      C=Temp1(k:k+3)    ;
      B=reshape(C,[2,2]);       B=B';         A(i:i+1,j:j+1)=B;                  k=k+4;
      end;           end;
  ImageFinale=double(A);
```

APPENDICE F

PROGRAMMES EN VHDL

PROCESSUS DE DECOMPOSITION

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity WaveletDecompLH is
Generic (   DataPathBasD                    : Positive:=11;
                        DataPathHautD       : Positive:=11;
                        OrdreFiltreBas,
                        OrdreFiltreHaut     : Positive:=3);
            Port(       Clk                             : In bit;

-- Vecteurs Entrée
                        VecteurAdressea,
                        VecteurAdresseb,
                        VecteurAdressec,
                        VecteurAdressed     : in
            bit_vector(2*OrdreFiltreBas+2-1 downto 0);
-- Niveau 1 de decomposition     Image --->L
--et Image --->H
                        Reset1                          : In
                        bit;
                        HSel1                           : In
                        bit_vector (0 DOWNTO 0);
-------------------------------------------------------------------------------
-
-- Niveau 2 de decomposition     L --->LL
-           et L --->LH
-- Niveau 2 de decomposition     H --->HL
--          et H --->HH
            Reset2              : In        bit;
            HSel2: In  bit_vector (0 DOWNTO 0);
-- Pour la simulation avec Matlab
YoutLaOut: out      bit_vector  (1 downto 0);
YoutLbOut: out      bit_vector  (1 downto 0);
YoutLcOut: out      bit_vector  (1 downto 0);
YoutLdOut: out      bit_vector  (1 downto 0);
YoutpmLL: out       bit_vector  (1 downto 0);
YoutpmLH: out       bit_vector  (1 downto 0);
YoutpmHL: out       bit_vector  (1 downto 0);
YoutpmHH: out       bit_vector  (1 downto 0));
-- Pour la simulation avec Matlab
End WaveletDecompLH;

architecture Behavioral of WaveletDecompLH is
Component FiltrePasseBas is
 generic  ( DataPathBasD        :           Positive:=11;
            OrdreFiltreBas                  :  Positive:= 3);
   Port  (  Clk                             :  in bit;
              ResetBas                      :  in bit     ;
 SelectHBas             :  in bit_vector(0 downto 0)       ;
 VecteurAdresseBas  :  in bit_vector(2*(OrdreFiltreBas+1)-1
downto 0);
YoutpmBas           :  out bit_vector(1 downto
0):=(others=>'0')); end Component;

Component FiltrePasseHaut is
 generic  ( DataPathHautD       :Positive               :=12;
            OrdreFiltreHaut     :           Positive    :=3);
    Port (  Clk                 :  in bit               ;
            ResetHaut           :  in bit               ;
            SelectHHaut         :  in bit_vector(0 downto 0);
VecteurAdresseHaut  :  in bit_vector(2*OrdreFiltreHaut+2-1
downto 0);

YoutpmHaut  :  out bit_vector(1 downto 0):=(others=>'0'));
End Component;
```

```
Component InOut1 is
   Port (   La : in bit_vector(1 downto 0);
            Lb : in bit_vector(1 downto 0);
            Lc                  : in bit_vector(1 downto 0);
            Ld                  : in bit_vector(1 downto 0);
            Ha                  : in bit_vector(1 downto 0);
            Hb                  : in bit_vector(1 downto 0);
            Hc                  : in bit_vector(1 downto 0);
            Hd                  : in bit_vector(1 downto 0);

        sortieLabcd : out bit_vector(7 downto 0);
        sortieHabcd : out bit_vector(7 downto 0));
end Component;

signal      YoutLa,YoutLb,YoutLc,YoutLd,
            YoutHa,YoutHb,YoutHc,YoutHd
:bit_vector(1 downto 0);
signal      EntreeNiveauHLetHH,
            EntreeNiveauLLetLH:
bit_vector(2*OrdreFiltreBas+2-1 downto 0);

begin
-- DEBUT BLOC DE GENERATION DE LA BANDE L
-- les deux filtres travaillent en parallele
FiltreLa : FiltrePasseBas generic
map(DataPathBasD,OrdreFiltreBas)
  Port map(Clk,Reset1,HSel1,VecteurAdressea,YoutLa);

YoutLaOut<=YoutHa;
FiltreLb : FiltrePasseBas generic
map(DataPathBasD,OrdreFiltreBas)
Port map(Clk,Reset1,HSel1,VecteurAdresseb,YoutLb);
            YoutLbOut<=YoutHb;
FiltreLc : FiltrePasseBas generic
map(DataPathBasD,OrdreFiltreBas)
  Port map(Clk,Reset1,HSel1,VecteurAdressec,YoutLc);
            YoutLcOut<=YoutHc;                  FiltreLd :
FiltrePasseBas generic map(DataPathBasD,OrdreFiltreBas)
  Port map(Clk,Reset1,HSel1,VecteurAdressed,YoutLd);
            YoutLdOut<=YoutHd;                  -- Les
deux filtres travaillent en parallele
-- FIN BLOC DE GENERATION DE LA BANDE L

-- DEBUT DE LA GENERATION DE LA BANDE LL ET LH
FiltreLL : FiltrePasseBas generic
map(DataPathBasD,OrdreFiltreBas)
   Port  map
(Clk,Reset2,HSel2,EntreeNiveauLLetLH,YoutpmLL);

FiltreLH : FiltrePasseHaut generic
map(DataPathHautD,OrdreFiltreHaut)
   Port  map
(Clk,Reset2,HSel2,EntreeNiveauLLetLH,YoutpmLH);

-- FIN DE LA GENERATION DE LA BANDE LL ET LH

-- Bloc commun aux deux décompositions
DirectOutputL: InOut1 port
map(YoutLa,YoutLb,YoutLc,YoutLd,


YoutHa,YoutHb,YoutHc,YoutHd,


EntreeNiveauLLetLH,EntreeNiveauHLetHH);
-- Vecteur Adresse
-- Pour une synchronisation complète
-- Bloc commun aux deux décompositions
-- DEBUT BLOC DE GENERATION DE LA BANDE H
```

```
FiltreHa : FiltrePasseHaut generic
map(DataPathHautD,OrdreFiltreHaut)
  Port map(Clk,Reset1,HSel1,VecteurAdressea,YoutHa);

FiltreHb : FiltrePasseHaut generic
map(DataPathHautD,OrdreFiltreHaut)
  Port map(Clk,Reset1,HSel1,VecteurAdresseb,YoutHb);

FiltreHc : FiltrePasseHaut generic
map(DataPathHautD,OrdreFiltreHaut)
  Port map(Clk,Reset1,HSel1,VecteurAdressec,YoutHc);

FiltreHd : FiltrePasseHaut generic
map(DataPathHautD,OrdreFiltreHaut)
  Port map(Clk,Reset1,HSel1,VecteurAdressed,YoutHd);

-- Les deux filtres travaillent en parallele
-- FIN BLOC DE GENERATION DE LA BANDE L
-- REDIRECTION DES DONNEES

-- DEBUT DE LA GENERATION DE LA BANDE LL ET LH
FiltreHL : FiltrePasseBas
  generic map(DataPathBasD,OrdreFiltreBas)
    Port  map
(Clk,Reset2,HSel2,EntreeNiveauHLetHH,YoutpmHL);

FiltreHH : FiltrePasseHaut
  generic map(DataPathHautD,OrdreFiltreHaut)
    Port  map
(Clk,Reset2,HSel2,EntreeNiveauHLetHH,YoutpmHH);
-- FIN DE LA GENERATION DE LA BANDE LL ET LH

End Behavioral;
```

TEST BENCH DE LA DECOMPOSITION

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY testbench IS
END         testbench;

ARCHITECTURE behavior OF testbench IS

Component WaveletDecompLH is
Generic (DataPathBasD                   : Positive:=11;
            DataPathHautD               : Positive:=12;
            OrdreFiltreBas,
            OrdreFiltreHaut         : Positive:=3);
   Port(Clk
            : In bit;
-- Vecteurs Entrée
VecteurAdressea, VecteurAdresseb, VecteurAdressec,
VecteurAdressed       : in bit_vector(2*OrdreFiltreBas+2-1
downto 0);
-- Niveau 1 de decomposition     Image --->L             --et
Image --->H
            Reset1                      : In
            bit;
            HSel1                       : In
            bit_vector (0 DOWNTO 0);
-------------------------------------------------------------------------
--
-- Niveau 2 de decomposition              L --->LL
--                                                     et
            L --->LH
-- Niveau 2 de decomposition              H --->HL
--                                                     et
            H --->HH
                  Reset2                        : In
                  bit;
                  HSel2                         : In
                  bit_vector (0 DOWNTO 0);
-- Pour la simulation avec Matlab


ZoutLaOut: out bit_vector  (1 downto 0);
ZoutLbOutbit_vector  (1 downto 0);
ZoutLcOut                                       : out
            bit_vector  (1 downto 0);
            ZoutLdOut
            : out       bit_vector  (1 downto 0);
                        ZoutpmLL
            : out       bit_vector  (1 downto 0);
                        ZoutpmLH
                        : out       bit_vector  (1 downto 0);

            ZoutpmHL
            : out       bit_vector  (1 downto 0);
                        ZoutpmHH
                        : out       bit_vector  (1 downto 0));

-- Pour la simulation avec Matlab
end component;

SIGNAL clkLectureFichier                        :  bit;--
Important
            SIGNAL clk
                        :  bit;
            SIGNAL reset1
            :  bit;
            SIGNAL hsel1
            :  bit_vector(0 to 0);
            SIGNAL reset2
            :  bit;
            SIGNAL hsel2
            :  bit_vector(0 to 0);
            SIGNAL clkEcritureFichier
            :  bit;
-- Important -----------------------------------------------------------
--
            SIGNAL vecteuradressea
            :  bit_vector(7 downto 0);
            SIGNAL vecteuradresseb
            :  bit_vector(7 downto 0);
            SIGNAL vecteuradressec
            :  bit_vector(7 downto 0);
            SIGNAL vecteuradressed
            :  bit_vector(7 downto 0);
-------------------------------------------------------------
            SIGNAL ZoutLaOut
            : bit_vector  (1 downto 0);             SIGNAL
ZoutLbOut                                       :
bit_vector  (1 downto 0);
            SIGNAL ZoutLcOut
            : bit_vector  (1 downto 0);
            SIGNAL ZoutLdOut
            : bit_vector  (1 downto 0);
-------------------------------------------------------------
            SIGNAL zoutpmll
            :  bit_vector(1 downto 0);
            SIGNAL zoutpmlh
            :  bit_vector(1 downto 0);
            SIGNAL zoutpmhl
            :  bit_vector(1 downto 0);
            SIGNAL zoutpmhh
            :  bit_vector(1 downto 0);
---------------------- A D D I T I O N A L  ----------------------
Constant NOmbredeBitsAvecDelai: positive:=18;
Constant TailleConversion: positive:=10;
-- Bande LL
            SIGNAL ZResPlusLL    :
bit_vector(NOmbredeBitsAvecDelai downto 0);
            SIGNAL ZResMoinsLL :
bit_vector(NOmbredeBitsAvecDelai downto 0);
-- Bande LH
```

```
SIGNAL ZResPlusLH    : bit_vector(NOmbredeBitsAvecDelai
downto 0);
SIGNAL ZResMoinsLH : bit_vector(NOmbredeBitsAvecDelai
downto 0);
-----------------------------------------------------------------
-- Bande HL
SIGNAL ZResPlusHL  : bit_vector(NOmbredeBitsAvecDelai
downto 0);
SIGNAL ZResMoinsHL : bit_vector(NOmbredeBitsAvecDelai
downto 0);
-- Bande HH
SIGNAL ZResPlusHH  : bit_vector(NOmbredeBitsAvecDelai
downto 0);
SIGNAL ZResMoinsHH : bit_vector(NOmbredeBitsAvecDelai
downto 0);
-----------------------------------------------------------------
                Constant   DelaiL      : natural:=0;
                Constant   DelaiH      : natural:=0;
                Constant DataPathBas: Positive:=8;
                Constant DataPathHaut: Positive:=8;
-----------------------------------------------------------------
signal X1,X2,X3,X4  :  bit_vector(NOmbredeBitsAvecDelai-1
downto 0):=(others=>'0');
signal X5,X6,X7,X8  :  bit_vector(NOmbredeBitsAvecDelai-1
downto 0):=(others=>'0');
signal X9 ,X10,X11,X12:  bit_vector(NOmbredeBitsAvecDelai-
1 downto 0):=(others=>'0');
signal X13,X14,X15,X16:  bit_vector(NOmbredeBitsAvecDelai-
1 downto 0):=(others=>'0');
-----------------------------------------------------------------
SIGNAL compteurbit:  integer:=NOmbredeBitsAvecDelai-1; --
bits entrée premiers 5 blocs
Signal Compteurbitout:
                integer:=NOmbredeBitsAvecDelai;
---------------- Resultats Intermediaires ----------------------------
--------
---------Fichiers -------------------------------------------
file FichierLecture :   text open read_mode  Is
"C:\MyProject\BiorWavelet31\imageLin.txt";
file FichierconvRed:  text open write_mode Is
"C:\MyProject\BiorWavelet31\ImagBin.txt";
-----------------------------------------------------------------
file FichierEcritureLL :text open write_mode IS
"C:\MyProject\BiorWavelet31\ImageLL.txt";
file FichierEcritureLH:text open write_mode IS
"C:\MyProject\BiorWavelet31\ImageLH.txt";
file FichierEcritureHL:text open write_mode IS
"C:\MyProject\BiorWavelet31\ImageHL.txt";
file FichierEcritureHH:text open write_mode
IS"C:\MyProject\BiorWavelet31\ImageHH.txt";
-----------------------------------------------------------------
---------Constantes du process     -------------------------------
                constant DemiCycleHorloge  :time              :=20 ns; -
-Cycle minimal de Modelsim;
                constant CycleHorloge              :time
                :=2*DemiCycleHorloge;
---------- Délai : ------------------------------------------------------

                constant TempsAccesfichier
                :time:=DemiCycleHorloge;
constant DelaiUnitaire              :time        :=0*CycleHorloge;
---------- Process Reset
constant TempsdeReset               :time        :=CycleHorloge;
constant TempsduProcess             :time
                :=NOmbredeBitsAvecDelai*CycleHorloge;
---------- Process Lecture
constant TempsDeLecture             :time:=TempsAccesfichier;
constant
CycleLectureFichier:time:=TempsduProcess+DemiCycleHorlog
e;
---------- Process Ecriture
constant TempsEcriture:time       :=TempsAccesFichier;
constant AttenteCycleEcriture:time
                :=TempsduProcess+DemiCycleHorloge;
---------- Calcul de la boucle de H
                constant ValeurZero                 :integer:=0;
```

```
BEGIN
                UUT : WaveletDecompLH
                GENERIC MAP (      DataPathBasD           => 11,
                                                DataPathHautD
                => 12,
                                                OrdreFiltreBas => 3,
OrdreFiltreHaut=> 3  )
PORT MAP(               clk => clk,
                                vecteuradressea => vecteuradressea,
                                vecteuradresseb => vecteuradresseb,
                                vecteuradressec => vecteuradressec,
                                vecteuradressed => vecteuradressed,
                                reset1 => reset1                ,
                                hsel1 => hsel1
                ,
                                reset2 => reset2                ,
                                hsel2 => hsel2
                ,
                                ZoutLaOut=>ZoutLaOut            ,
                                ZoutLbOut=>ZoutLbOut            ,
                                ZoutLcOut=>ZoutLcOut            ,
                                ZoutLdOut=>ZoutLdOut            ,
                                zoutpmll => zoutpmll            ,
                                zoutpmlh => zoutpmlh            ,
                                zoutpmhl => zoutpmhl            ,
                                zoutpmhh => zoutpmhh            );
-- *** Test Bench - User Defined Section ***
                PROCESS -- clock process for Clk,
                BEGIN
                                CLOCK_LOOP : LOOP
                                Clk <= transport '1';
                                                WAIT FOR
DemiCycleHorloge;
                                Clk <= transport '0';
                                                WAIT FOR
DemiCycleHorloge;
                                END LOOP CLOCK_LOOP;
                END PROCESS;
-----------------------------------------------------------------
                PROCESS -- clock process Lecture FICHIER
                BEGIN
                                ClkLectureFichier <= transport '0';
                                                WAIT FOR
DemiCycleHorloge;
                                CLOCK_LOOP : LOOP
                                ClkLectureFichier <= transport '1';
                                                WAIT FOR TempsDeLecture;
                                ClkLectureFichier <= transport '0';
                                WAIT FOR CycleLectureFichier;
                                END LOOP CLOCK_LOOP;
                END PROCESS;
-----------------------------------------------------------------
-- Les Reset
--              SIGNAL reset1 :  std_logic;
--              SIGNAL reset2 :  std_logic;
-----------------------------------------------------------------
                PROCESS
                BEGIN
                                Reset1_LOOP : LOOP
                                Reset1 <= transport '1';
                                WAIT FOR CycleHorloge;
                                Reset1 <= transport '0';
                                WAIT FOR TempsDuProcess;
                                END LOOP Reset1_LOOP;
                END PROCESS;

-- Les selections de la boucle du H et Y Niveau 1
-----------------------------------------------------------------
                PROCESS
                BEGIN
                                Hsel1_LOOP : LOOP
                                Hsel1 <= transport "1";
                                                WAIT FOR 2*CycleHorloge;
                                Hsel1 <= transport "0";
                                                WAIT FOR TempsDuProcess-
CycleHorloge;
```

```
                END LOOP Hsel1_LOOP;
        END PROCESS;
-- Les selections de la boucle du H et Z Niveau  2
-------------------------------------------------------------------
        PROCESS
        BEGIN
                Reset2 <= transport '1';
                WAIT FOR CycleHorloge;
                Reset2_LOOP : LOOP
                        Reset2 <= transport '1';
                        WAIT FOR CycleHorloge;
                        Reset2 <= transport '0';
                        WAIT FOR TempsDuProcess;
                END LOOP Reset2_LOOP;
        END PROCESS;
-------------------------------------------------------------------
        PROCESS
        BEGIN
                Hsel2 <= transport "1";
                WAIT FOR CycleHorloge;
                Hsel2_LOOP : LOOP
                        Hsel2 <= transport "1";
                        WAIT FOR 2*CycleHorloge;
                        Hsel2 <= transport "0";
                        WAIT FOR TempsDuProcess-
CycleHorloge;
                END LOOP Hsel2_LOOP;
end process;

--      SIGNAL clkEcritureFichierLLetLH         :  bit;--
Important
-------------------------------------------------------------------
        PROCESS -- clock process Lecture FICHIER
        BEGIN
                clkEcritureFichier <= transport '0';
                WAIT FOR
CycleHorloge+DemiCycleHorloge;
                Ecriture_LOOP : LOOP
                        clkEcritureFichier <= transport
'0';
                        WAIT FOR TempsDuProcess;
                        clkEcritureFichier <= transport
'1';
                        WAIT FOR
TempsAccesFichier;
                        clkEcritureFichier <= transport
'0';
                        WAIT FOR
DemiCycleHorloge;
                END LOOP Ecriture_LOOP;
        END PROCESS;
-------------------------------------------------------------------
  ProcessLectureVecteur : PROCESS(ClkLectureFichier)
        Variable
        Ligne1,Ligne2,Ligne3,Ligne4     :       Line;
        Variable
        Pixel1,Pixel2,Pixel3,Pixel4     :
        Integer:=0;
        Variable        Zeros8          :
        bit_vector(7 downtO 0):=(others=>'0');
  BEGIN
        if not endfile(FichierLecture) then
                If ClkLectureFichier'event and
ClkLectureFichier='1'then
        ----------ligne 1  du fichier   ----------

        readline(FichierLecture,Ligne1);

        readline(FichierLecture,Ligne2);

        readline(FichierLecture,Ligne3);

        readline(FichierLecture,Ligne4);
                                read(Ligne1,Pixel1);
                                read(Ligne2,Pixel2);
                                read(Ligne3,Pixel3);
                                read(Ligne4,Pixel4);

        X1<=to_bitvector(conv_std_logic_vector(Pixel1,Taill
eConversion))&Zeros8;
        X2<=to_bitvector(conv_std_logic_vector(Pixel2,Taill
eConversion))&Zeros8;
        X3<=to_bitvector(conv_std_logic_vector(Pixel3,Taill
eConversion))&Zeros8;
        X4<=to_bitvector(conv_std_logic_vector(Pixel4,Taill
eConversion))&Zeros8;
-------------------------------------------------------------------

        readline(FichierLecture,Ligne1);

        readline(FichierLecture,Ligne2);

        readline(FichierLecture,Ligne3);

        readline(FichierLecture,Ligne4);
                                read(Ligne1,Pixel1);
                                read(Ligne2,Pixel2);
                                read(Ligne3,Pixel3);
                                read(Ligne4,Pixel4);
        X5<=to_bitvector(conv_std_logic_vector(Pixel1,Taill
eConversion))&Zeros8;
        X6<=to_bitvector(conv_std_logic_vector(Pixel2,Taill
eConversion))&Zeros8;
        X7<=to_bitvector(conv_std_logic_vector(Pixel3,Taill
eConversion))&Zeros8;
        X8<=to_bitvector(conv_std_logic_vector(Pixel4,Taill
eConversion))&Zeros8;
-------------------------------------------------------------------

        readline(FichierLecture,Ligne1);

        readline(FichierLecture,Ligne2);

        readline(FichierLecture,Ligne3);

        readline(FichierLecture,Ligne4);
                                read(Ligne1,Pixel1);
                                read(Ligne2,Pixel2);
                                read(Ligne3,Pixel3);
                                read(Ligne4,Pixel4);
        X9
<=to_bitvector(conv_std_logic_vector(Pixel1,TailleConversion)
)&Zeros8;
        X10<=to_bitvector(conv_std_logic_vector(Pixel2,Tai
lleConversion))&Zeros8;
        X11<=to_bitvector(conv_std_logic_vector(Pixel3,Tai
lleConversion))&Zeros8;
        X12<=to_bitvector(conv_std_logic_vector(Pixel4,Tai
lleConversion))&Zeros8;
-------------------------------------------------------------------

        readline(FichierLecture,Ligne1);

        readline(FichierLecture,Ligne2);

        readline(FichierLecture,Ligne3);

        readline(FichierLecture,Ligne4);
                                read(Ligne1,Pixel1);
                                read(Ligne2,Pixel2);
                                read(Ligne3,Pixel3);
                                read(Ligne4,Pixel4);

        X13<=to_bitvector(conv_std_logic_vector(Pixel1,Tai
lleConversion))&Zeros8;
        X14<=to_bitvector(conv_std_logic_vector(Pixel2,Tai
lleConversion))&Zeros8;
        X15<=to_bitvector(conv_std_logic_vector(Pixel3,Tai
lleConversion))&Zeros8;
        X16<=to_bitvector(conv_std_logic_vector(Pixel4,Tai
lleConversion))&Zeros8;
-------------------------------------------------------------------
```

```
                        end if;
--- Ecriture vers fichier de conversion ----------------
  else
            assert False report "fin de fichier " severity failure;
            file_close(FichierLecture)                    ;
end if;
END process;

-- Process pour lire du vecteur fichier vers le VHDL map port
bibit par bibit:
  Processinjectionbit : PROCESS(Clk,ClkLectureFichier)
  BEGIN
            If ClkLectureFichier'event and ClkLectureFichier='1'
then
                        Compteurbit<=NOmbredeBitsAvecDelai-
1;
            end if;
            If clk'event and clk='1'              then
If ClkLectureFichier='0' then
VecteurAdressea(7)<=          '0';
VecteurAdressea(6)<=          X1(compteurbit);
VecteurAdressea(5)<=          '0';
VecteurAdressea(4)<=          X2(compteurbit);
VecteurAdressea(3)<=          '0';
VecteurAdressea(2)<=          X3(compteurbit);
VecteurAdressea(1)<=          '0';
VecteurAdressea(0)<=          X4(compteurbit);
VecteurAdresseb(7)<=          '0';
VecteurAdresseb(6)<=          X5(compteurbit);
VecteurAdresseb(5)<=          '0';
VecteurAdresseb(4)<=          X6(compteurbit);
VecteurAdresseb(3)<=          '0';
VecteurAdresseb(2)<=          X7(compteurbit);
VecteurAdresseb(1)<=          '0';
VecteurAdresseb(0)<=          X8(compteurbit);
VecteurAdressec(7)<=          '0';
VecteurAdressec(6)<=          X9(compteurbit);

            VecteurAdressec(5)<=          '0';

            VecteurAdressec(4)<=          X10(compteurbit);

            VecteurAdressec(3)<=          '0';

            VecteurAdressec(2)<=          X11(compteurbit);

            VecteurAdressec(1)<=          '0';

            VecteurAdressec(0)<=          X12(compteurbit);
            VecteurAdressed(7)<=          '0';
VecteurAdressed(6)<=          X13(compteurbit);
VecteurAdressed(5)<=          '0';
VecteurAdressed(4)<=          X14(compteurbit);
VecteurAdressed(3)<=          '0';
VecteurAdressed(2)<=          X15(compteurbit);
VecteurAdressed(1)<=          '0';
VecteurAdressed(0)<=          X16(compteurbit);
Compteurbit<=(compteurbit-1) mod NombredeBitsAvecDelai;
End if;
End if;
  END PROCESS ; -- injection par bit;
 ProcessEcritureFichier : PROCESS(Reset2,Clk)
            Variable    ligneLL,ligneLH       : Line;
            Variable    ligneHL,ligneHH       : Line;
--------------------------------------------------------------------------
            Variable    ligneLLRed,ligneLHRed          : Line;
            Variable    ligneHLRed,ligneHHRed          : Line;
            Variable    lignezero
                        : Line;
            variable    zplusLL,zMoinsLL
            : STD_logic_vector(NOmbredeBitsAvecDelai
downto 0);
            variable    zplusLH,zMoinsLH
            : STD_logic_vector(NOmbredeBitsAvecDelai
downto 0);
            variable    zLL,zLH
                        : integer;
variable DataAvailable            : boolean:=False;
variable zplusHL,zMoinsHL     :
STD_logic_vector(NOmbredeBitsAvecDelai downto 0);
variable zplusHH,zMoinsHH     :
STD_logic_vector(NOmbredeBitsAvecDelai downto 0);
variable zHL,zHH                  : integer;

BEGIN
                        If Reset2'event and Reset2='0' then
                                                If
DataAvailable then
--BANDE LL
            zplusLL :=To_X01(ZResPlusLL); --
(NOmbredeBitsAvecDelai-1 downto 0));
            zmoinsLL:=To_X01(ZResMoinsLL);--
(NOmbredeBitsAvecDelai-1 downto 0));
            zLL:=-
2*CONV_INTEGER(zplusLL)+CONV_INTEGER(zMoinsLL);
                                                ------------------------
------------------------
                                    Write(ligneLL,zLL);

            WriteLine(FichierEcritureLL,ligneLL);
                                                --BANDE LH
            zplusLH :=To_X01(ZResPlusLH); --
(NOmbredeBitsAvecDelai-1 downto 0));
            zmoinsLH:=To_X01(ZResMoinsLH); --
(NOmbredeBitsAvecDelai-1 downto 0));
            zLH:=-
2*CONV_INTEGER(zplusLH)+CONV_INTEGER(zMoinsLH)
;
                                                ------------------------
------------------------
                                    Write(ligneLH,zLH);

            WriteLine(FichierEcritureLH,ligneLH);
--BANDE HL
            zplusHL :=To_X01(ZResPlusHL); --
(NOmbredeBitsAvecDelai-1 downto 0));
            zmoinsHL:=To_X01(ZResMoinsHL);--
(NOmbredeBitsAvecDelai-1 downto 0));
            zHL:=-
2*CONV_INTEGER(zplusHL)+CONV_INTEGER(zMoinsHL)
;
                                                ------------------------
------------------------
                                    Write(ligneHL,zHL);

            WriteLine(FichierEcritureHL,ligneHL);
                                                ------------------------
------------------------
--BANDE HH
            zplusHH :=To_X01(ZResPlusHH); --
(NOmbredeBitsAvecDelai-1 downto 0));
            zmoinsHH:=To_X01(ZResMoinsHH);--
(NOmbredeBitsAvecDelai-1 downto 0));
            zHH:=-
2*CONV_INTEGER(zplusHH)+CONV_INTEGER(zMoinsHH)
;
                                    Write(ligneHH,zHH);

            WriteLine(FichierEcritureHH,ligneHH);
            else
                                    DataAvailable:=True;
            end if;
Compteurbitout<=NOmbredeBitsAvecDelai;-- =9 downto 0 : 10
bits dont bit de convergence
-- bande LL
            ZResPlusLL  <=(others=>'0');-- Initialisation Vecteur
Sortie
            ZResMoinsLL <=(others=>'0');-- Initialisation
Vecteur Sortie
-- bande LH
END;
```

PROCESS DE LA RECONSTRUCTION

```
library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_ARITH.ALL;
Use IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity WaveletReconstruction is
Generic (   DataPathRL
            : Positive:=12;
                        DataPathRH
                        : Positive:=12;
                        OrdreFiltreR
            : Positive:=3);
   Port(    Clk
                        : In bit;
                        Reset1
                        : In bit;
                        HSel1
                        : In bit_vector (0 DOWNTO 0);
                        Reset2
                        : In bit;
                        HSel2
                        : In bit_vector (0 DOWNTO 0);
                        Resetadd
            : In bit;
                        SelectH
            : In bit_vector (0 DOWNTO 0);
-- BANDE LL
            VecteuradresseabLL,
            VecteurAdressecdLL : in         bit_vector
(2*OrdreFiltreR+1-4 downto 0);
-------------------------------------------------------------------------------
-------
            YoutaL1,YoutbL1,YoutcL1,YoutdL1,

            Youta11LL,Youta21LL,
            Youta12LL,Youta22LL: out bit_vector(1 downto 0);
-- BANDE LH
            VecteuradresseabLH,
            VecteurAdressecdLH : in         bit_vector
(2*OrdreFiltreR+1-4 downto 0);
-------------------------------------------------------------------------------
-------
            YoutaL2,YoutbL2,
            YoutcL2,YoutdL2,

            Youta11LH,Youta21LH,
            Youta12LH,Youta22LH: out bit_vector(1 downto 0);
-- BANDE HL
            VecteuradresseabHL,
            VecteurAdressecdHL : in         bit_vector
(2*OrdreFiltreR+1-4 downto 0);
-------------------------------------------------------------------------------
-------
            YoutaH1,YoutbH1,YoutcH1,YoutdH1,

            Youta11HL,Youta21HL,
            Youta12HL,Youta22HL: out bit_vector(1 downto 0);
-- BANDE HH
            VecteuradresseabHH,
            VecteurAdressecdHH: in          bit_vector
(2*OrdreFiltreR+1-4 downto 0);
-------------------------------------------------------------------------------
-------
            YoutaH2,YoutbH2,YoutcH2,YoutdH2,
Youta11HH,Youta21HH,
            Youta12HH,Youta22HH         : out bit_vector(1
downto 0);
            a11,a21, a12,a22                : out bit_vector(1
downto 0));
End WaveletReconstruction;



Architecture Behavioral of WaveletReconstruction is

Component FilterRecLL is
Generic (   DataPathRL
            : Positive:=12;
                        DataPathRH
                        : Positive:=12;
                        OrdreFiltreR
            : Positive:=3);
   Port(    Clk
                        : In bit;
                        Reset1
                        : In bit;
                        HSel1
                        : In bit_vector (0 DOWNTO 0);
-- Niveau 2 de decomposition     L --->LL
                        Reset2
                        : In bit;
                        HSel2
                        : In bit_vector (0 DOWNTO 0);
-- BANDE LL
                        Vecteuradresseab,
                        VecteurAdressecd    : in
            bit_vector (2*OrdreFiltreR+1-4 downto 0);
                        MemoryAda           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldeja               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHa                      : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdb           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejb               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHb                      : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdc           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejc               : inout
bit_vector(DataPathRL-1 DownTo 0) ;
                        ResultatHc                      : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdd           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejd               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHd                      : buffer
bit_vector(DataPathRL-1 DownTo 0);
-------------------------------------------------------------------------------
-------
                        Youta,Youtb, Youtc,Youtd,

                        Yfinala11,Yfinala12,
                        Yfinala21,Yfinala22             : out
bit_vector(1 downto 0));
end Component;


Component FilterRecHL is
Generic (   DataPathRL                  : Positive:=12;
                        DataPathRH                      :
Positive:=12;
                        OrdreFiltreR                    :
Positive:=3);
   Port(    Clk                         : In bit;

                        Reset1                          : In bit;

                        HSel1                           : In
bit_vector (0 DOWNTO 0);
-- Niveau 2 de decomposition     L --->LL
                        Reset2                          : In bit;
                        HSel2                           : In
bit_vector (0 DOWNTO 0);
-- BANDE LL
                        Vecteuradresseab,
                        VecteurAdressecd    : in
            bit_vector (2*OrdreFiltreR+1-4 downto 0);
```

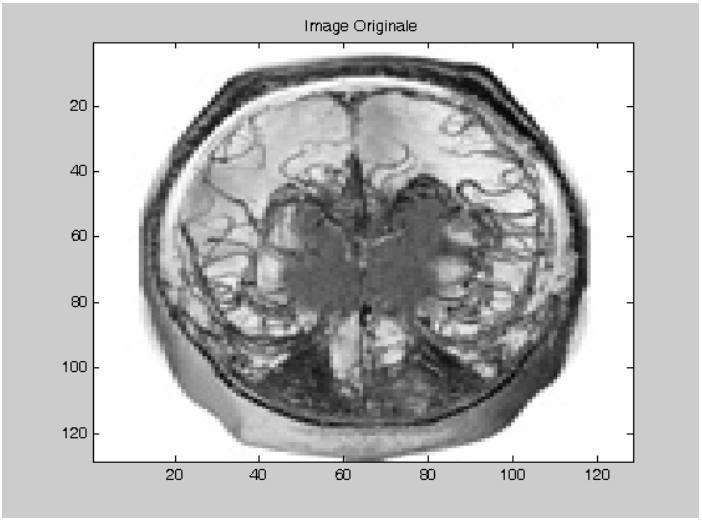
```
                        MemoryAda           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldeja               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHa                  : buffer
bit_vector(DataPathRL-1 DownTo 0);

                        MemoryAdb           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejb               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHb                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdc           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejc               : inout
bit_vector(DataPathRL-1 DownTo 0) ;
                        ResultatHc                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdd           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejd               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHd                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
--------------------------------------------------------------------------------
-------
                        Youta,Youtb, Youtc,Youtd,

                        Yfinala11,Yfinala12, Yfinala21,Yfinala22
                : out bit_vector(1 downto 0));
end Component;

Component FilterRecHH is
Generic (   DataPathRL              : Positive:=12;
                        DataPathRH              :
Positive:=12;
                        OrdreFiltreR            :
Positive:=3);
   Port(     Clk                       : In bit;

                        Reset1                  : In bit;

                        HSel1                   : In
bit_vector (0 DOWNTO 0);
-- Niveau 2 de decomposition     L --->LL
                        Reset2                  : In bit;
                        HSel2                   : In
bit_vector (0 DOWNTO 0);
-- BANDE LL
                        Vecteuradresseab,
                        VecteurAdressecd    : in
                bit_vector (2*OrdreFiltreR+1-4 downto 0);
                        MemoryAda           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldeja               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHa                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdb           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejb               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHb                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdc           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejc               : inout
bit_vector(DataPathRL-1 DownTo 0) ;
                        ResultatHc                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdd           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejd               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHd                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
--------------------------------------------------------------------------------
-------
                        Youta,Youtb, Youtc,Youtd,

                        Yfinala11,Yfinala12,
                        Yfinala21,Yfinala22  : out bit_vector(1
downto 0));
end Component;

Component FilterRecLH is
Generic (   DataPathRL              : Positive:=12;
                        DataPathRH              :
Positive:=12;
                        OrdreFiltreR            :
Positive:=3);
                Port(Clk                        : In bit;

                        Reset1                  : In bit;

                        HSel1                   : In
bit_vector (0 DOWNTO 0);
-- Niveau 2 de decomposition     L --->LL
                        Reset2                  : In bit;
                        HSel2                   : In
bit_vector (0 DOWNTO 0);
-- BANDE LL
                        Vecteuradresseab,
                        VecteurAdressecd    : in
                bit_vector (2*OrdreFiltreR+1-4 downto 0);
                        MemoryAda           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldeja               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHa                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdb           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejb               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHb                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdc           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejc               : inout
bit_vector(DataPathRL-1 DownTo 0) ;
                        ResultatHc                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
                        MemoryAdd           : out
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
                        Ldejd               : inout
bit_vector(DataPathRL-1 DownTo 0);
                        ResultatHd                  : buffer
bit_vector(DataPathRL-1 DownTo 0);
--------------------------------------------------------------------------------
-------
                        Youta,Youtb,
                        Youtc,Youtd,

                        Yfinala11,Yfinala12,
                        Yfinala21,Yfinala22  : out bit_vector(1
downto 0));
end Component;


Component AdditionneurEnLigne4 is
 generic  ( DataPath            :        Positive:=10;
                        OrdreFiltre             :
Positive:=3);
   Port  (    Clk                 : in bit                    ;
                        ResetAdd            : in bit
;
                        SelectH             : in bit_vector(0
downto 0) ;
```

```
                    VecteurAdresse                  : in
bit_vector(2*(OrdreFiltre+1)-1 downto 0);
---------- Resultats Intermediaires ----------
--------
                    Yout                            : out
bit_vector(1 downto 0):=(others=>'0'));
end Component;


signal      YfinalOuta11LLInt: bit_vector(1 downto 0);

signal      VecAdder11                  :
            bit_vector (7 downto 0);-- 4 éléments
signal      VecAdder12                  :
            bit_vector (7 downto 0);-- 4 éléments
signal      VecAdder21                  :
            bit_vector (7 downto 0);-- 4 éléments
signal      VecAdder22                  :
            bit_vector (7 downto 0);-- 4 éléments


signal                  MemoryAdaLL             :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal      LdejaLL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal      ResultatHaLL                :
bit_vector(DataPathRL-1 DownTo 0);

signal      MemoryAdbLL                 :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal      LdejbLL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal      ResultatHbLL                :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdcLL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejcLL                          :
bit_vector(DataPathRL-1 DownTo 0) ;
signal ResultatHcLL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAddLL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejdLL                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHdLL                     :
bit_vector(DataPathRL-1 DownTo 0);

---------- BANDE LH
signal MemoryAdaLH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejaLH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHaLH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdbLH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejbLH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHbLH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdcLH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejcLH                          :
bit_vector(DataPathRL-1 DownTo 0) ;
signal ResultatHcLH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAddLH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejdLH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHdLH                     :
bit_vector(DataPathRL-1 DownTo 0);
---------- BANDE HL
signal MemoryAdaHL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejaHL                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHaHL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdbHL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejbHL                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHbHL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdcHL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejcHL                          :
bit_vector(DataPathRL-1 DownTo 0) ;
signal ResultatHcHL                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAddHL                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejdHL                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHdHL                     :
bit_vector(DataPathRL-1 DownTo 0);
---------- BANDE HH
signal MemoryAdaHH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejaHH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHaHH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdbHH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejbHH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHbHH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAdcHH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejcHH                          :
bit_vector(DataPathRL-1 DownTo 0) ;
signal ResultatHcHH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal MemoryAddHH                      :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejdHH                          :
bit_vector(DataPathRL-1 DownTo 0);
signal ResultatHdHH                     :
bit_vector(DataPathRL-1 DownTo 0);
signal
            Youta11LLInt,Youta21LLInt,Youta12LLInt,Youta22
LLInt,
  Youta11LHInt,Youta21LHInt,Youta12LHInt,Youta22LHInt,
  Youta11HLInt,Youta21HLInt,Youta12HLInt,Youta22HLInt,

Youta11HHInt,Youta21HHInt,Youta12HHInt,Youta22HHInt:bit
_vector(1 DownTo 0);
Begin
-- BANDE LL
FiltreLL:  FilterRecLL Generic map
(DataPathRL,DataPathRL,OrdreFiltreR)
   Port map
(Clk,Reset1,HSel1,Reset2,HSel2,VecteuradresseabLL,
VecteurAdressecdLL,MemoryAdaLL,LdejaLL,
ResultatHaLL,MemoryAdbLL,LdejbLL, ResultatHbLL,
MemoryAdcLL, LdejcLL, ResultatHcLL, MemoryAddLL ,
LdejdLL, ResultatHdLL,YoutaL1,YoutbL1,YoutcL1,YoutdL1,
Youta11LLInt,Youta21LLInt,Youta12LLInt,Youta22LLInt);
Youta11LL<=Youta11LLInt;     Youta21LL<=Youta21LLInt;
Youta12LL<=Youta12LLInt; Youta22LL<=Youta22LLInt;
-- BANDE LH
FiltreLH:  FilterRecLH Generic map
(DataPathRL,DataPathRL,OrdreFiltreR)
   Port map (Clk, Reset1, HSel1, Reset2, HSel2,
VecteuradresseabLH, VecteurAdressecdLH,
MemoryAdaLH, LdejaLH, ResultatHaLH,
MemoryAdbLH,LdejbLH,ResultatHbLH,MemoryAdcLH ,
LdejcLH, ResultatHcLH, MemoryAddLH, LdejdLH ,
```

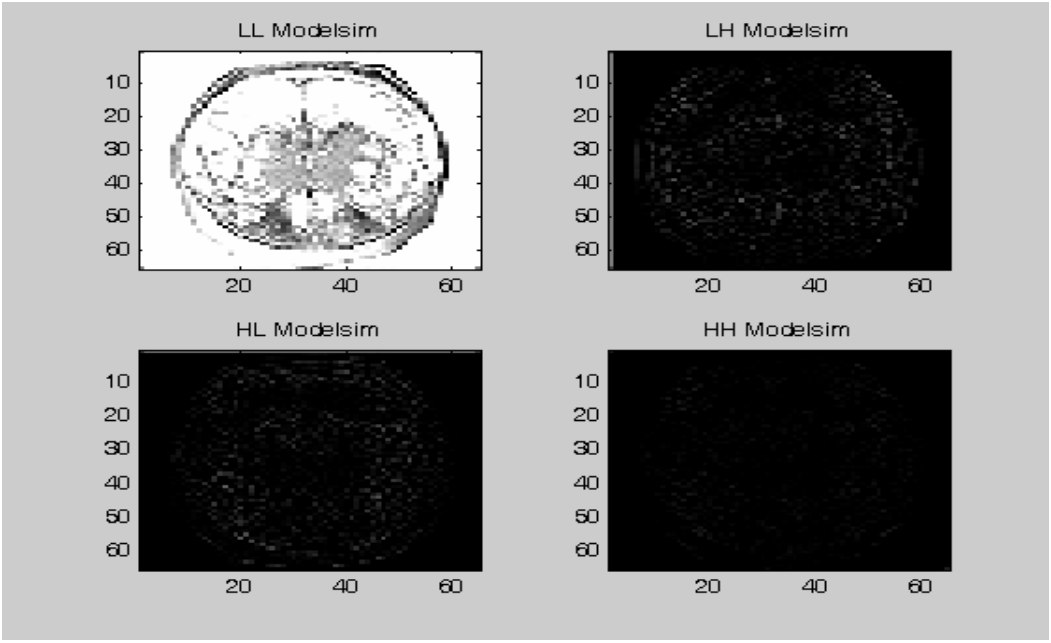
```
ResultatHdLH, YoutaL2,YoutbL2,
YoutcL2,YoutdL2,Youta11LHInt,Youta21LHInt,
            Youta12LHInt,Youta22LHInt);
Youta11LH<=Youta11LHInt;Youta21LH<=Youta21LHInt;
Youta12LH<=Youta12LHInt; Youta22LH<=Youta22LHInt;

-- BANDE HL

FiltreHL:  FilterRecHL Generic map
(DataPathRL,DataPathRL,OrdreFiltreR)
   Port map (Clk, Reset1, HSel1, Reset2, HSel2,
VecteuradresseabHL, VecteurAdressecdHL, MemoryAdaHL,
LdejaHL, ResultatHaHL, MemoryAdbHL , LdejbHL,
ResultatHbHL, MemoryAdcHL, LdejcHL, ResultatHcHL,
MemoryAddHL, LdejdHL, ResultatHdHL,  YoutaH1,YoutbH1,
YoutcH1,YoutdH1, Youta11HLInt,Youta21HLInt,
Youta12HLInt,Youta22HLInt);
Youta11HL<=Youta11HLInt; Youta21HL<=Youta21HLInt;
Youta12HL<=Youta12HLInt; Youta22HL<=Youta22HLInt;


FiltreHH:  FilterRecHH Generic map
(DataPathRL,DataPathRL,OrdreFiltreR)
   Port map (Clk, Reset1, HSel1, Reset2,HSel2,
VecteuradresseabHH,VecteurAdressecdHH,MemoryAdaHH    ,
LdejaHH ,ResultatHaHH,MemoryAdbHH,LdejbHH      ,
ResultatHbHH, MemoryAdcHH,LdejcHH,
ResultatHcHH,MemoryAddHH,LdejdHH,
ResultatHdHH,
YoutaH2,YoutbH2,YoutcH2,YoutdH2,Youta11HHInt,Youta21
HHInt,Youta12HHInt,Youta22HHInt);
            Youta11HH<=Youta11HHInt;
            Youta21HH<=Youta21HHInt;
            Youta12HH<=Youta12HHInt;
            Youta22HH<=Youta22HHInt;
            VecAdder11<=Youta11LLInt&Youta11LHInt&Yout
a11HLInt&Youta11HHInt;
            VecAdder21<=Youta21LLInt&Youta21LHInt&Yout
a21HLInt&Youta21HHInt;
            VecAdder12<=Youta12LLInt&Youta12LHInt&Yout
a12HLInt&Youta12HHInt;
            VecAdder22<=Youta22LLInt&Youta22LHInt&Yout
a22HLInt&Youta22HHInt;
Addera11:AdditionneurEnLigne4 Port
map(Clk,ResetAdd,SelectH,VecAdder11,a11);
Addera21:AdditionneurEnLigne4 Port
map(Clk,ResetAdd,SelectH,VecAdder21,a21);
Addera12:AdditionneurEnLigne4 Port
map(Clk,ResetAdd,SelectH,VecAdder12,a12);
Addera22:AdditionneurEnLigne4 Port
map(Clk,ResetAdd,SelectH,VecAdder22,a22);
End Behavioral;

signal MemoryAdaL1                  :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejaL1                      :
bit_vector(DataPathRH-1 DownTo 0);
signal ResultatHaL1                 :
bit_vector(DataPathRH-1 DownTo 0);
signal MemoryAdbL1                  :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejbL1                      :
bit_vector(DataPathRH-1 DownTo 0);
signal ResultatHbL1                 :
bit_vector(DataPathRH-1 DownTo 0);
signal MemoryAdcL1                  :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejcL1                      :
bit_vector(DataPathRH-1 DownTo 0) ;
signal ResultatHcL1                 :
bit_vector(DataPathRH-1 DownTo 0);
signal MemoryAddL1                  :
bit_vector(2*(OrdreFiltreR+1)-1-2 DownTo 0);
signal LdejdL1                      :
bit_vector(DataPathRH-1 DownTo 0);
signal ResultatHdL1                 :
bit_vector(DataPathRH-1 DownTo 0);
signal Youtfinala11,Youtfinala12,
            Youtfinalb21,Youtfinalb22       :
            bit_vector(1 downto 0);
signal      YoutInta,YoutIntb,YoutIntc,YoutIntd     :
            bit_vector(1 downto 0);
signal      EntreeNiveau2a,EntreeNiveau2b :
            bit_vector (2*(OrdreFiltreR+1)-1-4 downto 0);
begin
-- FiltreL1_1 : Convolution de 4 Yeros : Valeur Nulle
FiltreL1_2 : FilterHRec0V0V generic
map(DataPathRH,OrdreFiltreR)
  Port map(                 Clk,
                            Reset1,
                            HSel1,
                            Vecteuradresseab,
                            MemoryAda   ,
                            LdejaL1     ,
                            ResultatHa          ,
                            YoutInta); -- 0/V/0/V
-- FiltreL1_3 : Convolution de 4 Yeros : Valeur Nulle

FiltreL1_4 : FilterHRec0V0V generic
map(DataPathRH,OrdreFiltreR)
Port map(Clk, Reset1, HSel1, Vecteuradressecd, MemoryAdb   ,
LdejbL1, ResultatHb  , YoutIntb); -- 0/V/0/V
--------------------------------------------------------------------------------
FiltreL2_1 : FilterHRecV0V0 generic
map(DataPathRH,OrdreFiltreR)
  Port map(Clk, Reset1, HSel1, Vecteuradresseab, MemoryAdc  ,
LdejcL1, ResultatHc  , YoutIntc);
-- V/0/V/0
FiltreL2_3 : FilterHRecV0V0 generic
map(DataPathRH,OrdreFiltreR)
  Port map (Clk,Reset1, HSel1, Vecteuradressecd, MemoryAdd ,
LdejdL1, ResultatHd  , YoutIntd);
Youta<=YoutInta;     Youtb<=YoutIntb;     Youtc<=YoutIntc;
Youtd<=YoutIntd;

            EntreeNiveau2a<=YoutInta&YoutIntb;
            EntreeNiveau2b<=YoutIntc&YoutIntd;
            FiltreLa11 : FilterHRec0V0V generic
map(DataPathRH,OrdreFiltreR)
Port map(  Clk,Reset2, HSel2, --
EntreeNiveau2a, MemoryAdaL1, Ldeja      ,
ResultatHaL1, Yfinala11);
FiltreLa21 : FilterHRecV0V0 generic
map(DataPathRH,OrdreFiltreR)
Port map(  Clk, Reset2, HSel2,    EntreeNiveau2a  ,
MemoryAdbL1, Ldejb, ResultatHbL1, Yfinala21);

FiltreLa12 : FilterHRec0V0V generic
map(DataPathRH,OrdreFiltreR)
Port map(  Clk, Reset2, HSel2,    EntreeNiveau2b  ,
MemoryAdcL1, Ldejc , ResultatHcL1 , Yfinala12)      ;


FiltreLa22 : FilterHRecV0V0 generic
map(DataPathRH,OrdreFiltreR)
  Port map(Clk, Reset2, HSel2EntreeNiveau2b , MemoryAddL1,
Ldejd       , ResultatHdL1           , Yfinala22);

end Behavioral;
```

APPENDICE G

RESULTATS DES SIMULATIONS
DES DIFFERENTES IMAGES TESTEES

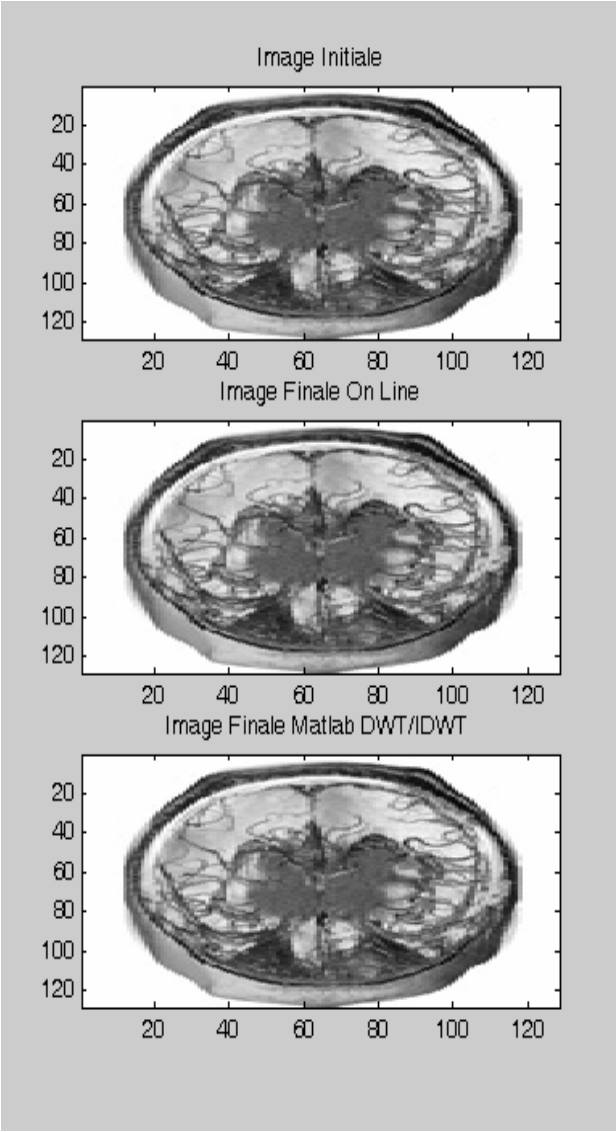


originale d'une coupe d'engiographe.
(Taille de l'image 462.668)



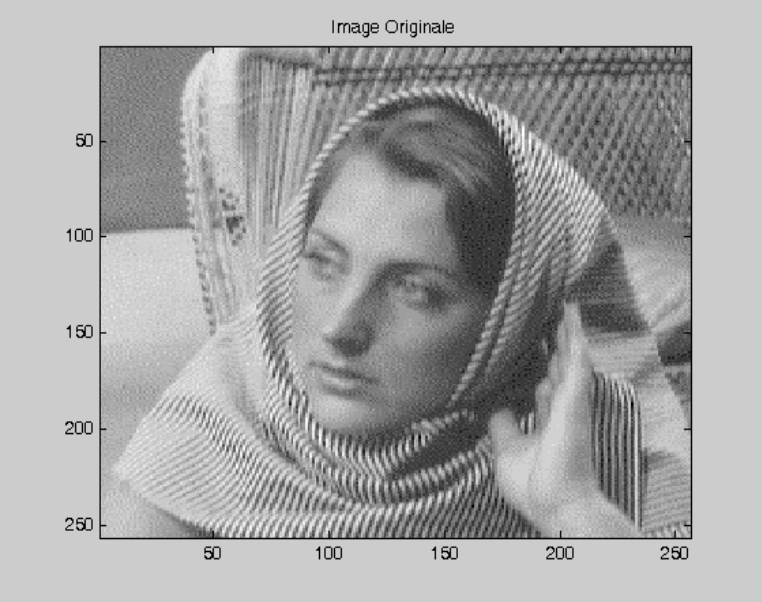
Résultats de décomposition sous Modelsim
des sous bandes LL, LH, HL et HH

Après décomposition de l'image originale en sous bandes LL(image approximatif), LH (image détail verticale), HL(image détail horizontal) et HH. (image détail diagonale) on obtient les résultats au dessus.

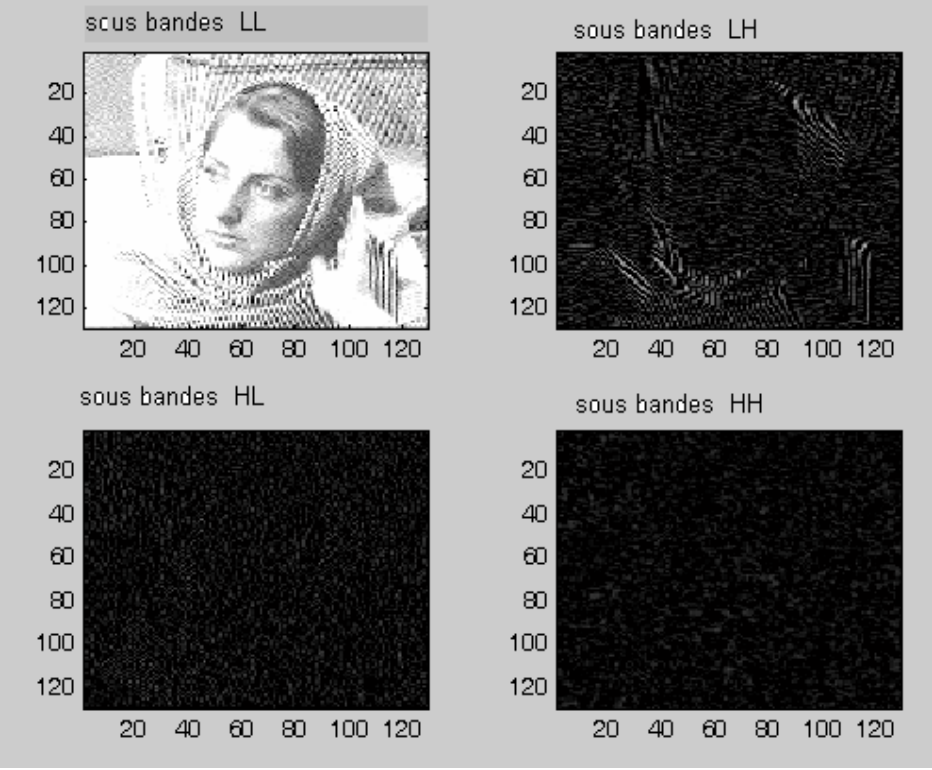


Résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlab

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de $6.27\ 10^{-14}$.

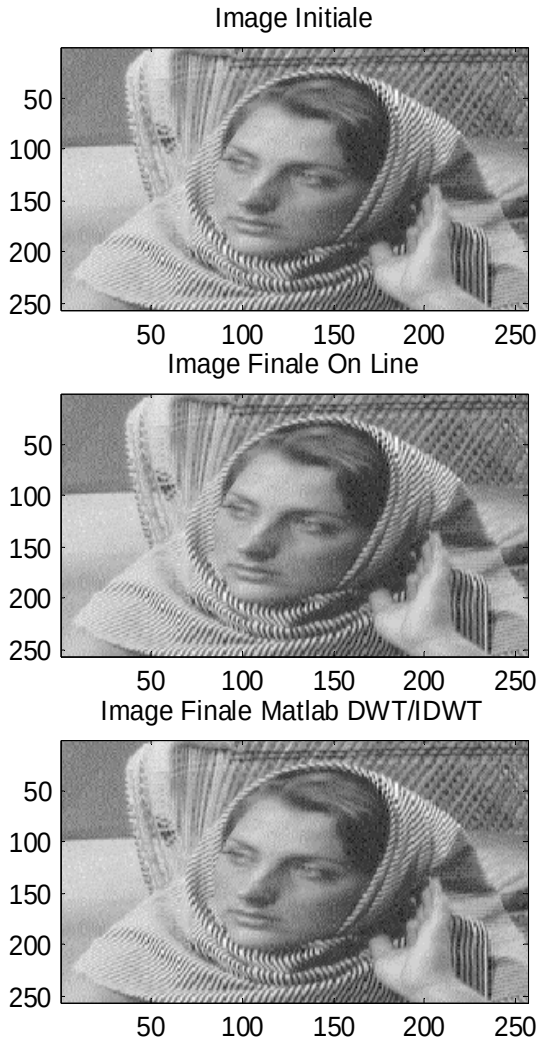


L'image originale de BARBARA [2]
(Taille de l'image 256.256)



Résultats de décomposition sous Modelsim
des sous bandes LL, LH, HL et HH.

Après décomposition de l'image originale en sous bandes LL(image approximatif), LH (image détail verticale), HL(image détail horizontal) et HH. (image détail diagonale) on obtient les résultats au dessus.



Résultats de simulations de la reconstruction de

l'image originale sous Modelsim et Matlab

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de$9.59\ 10^{-14}$ L'image originale de BARBARA.

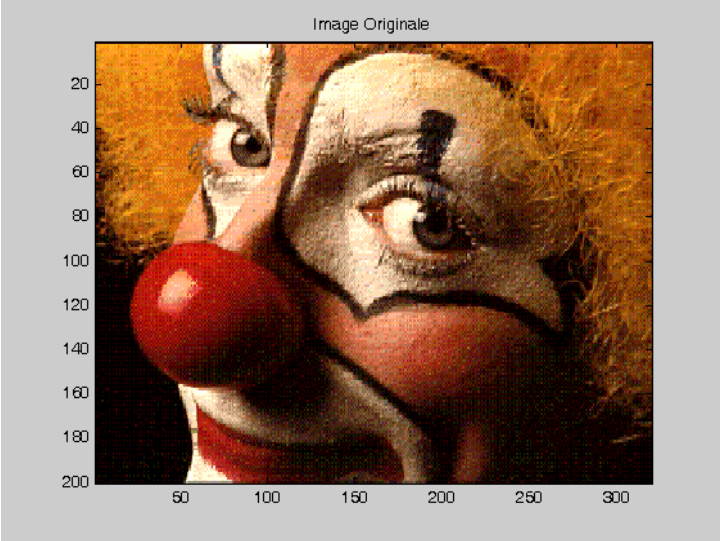
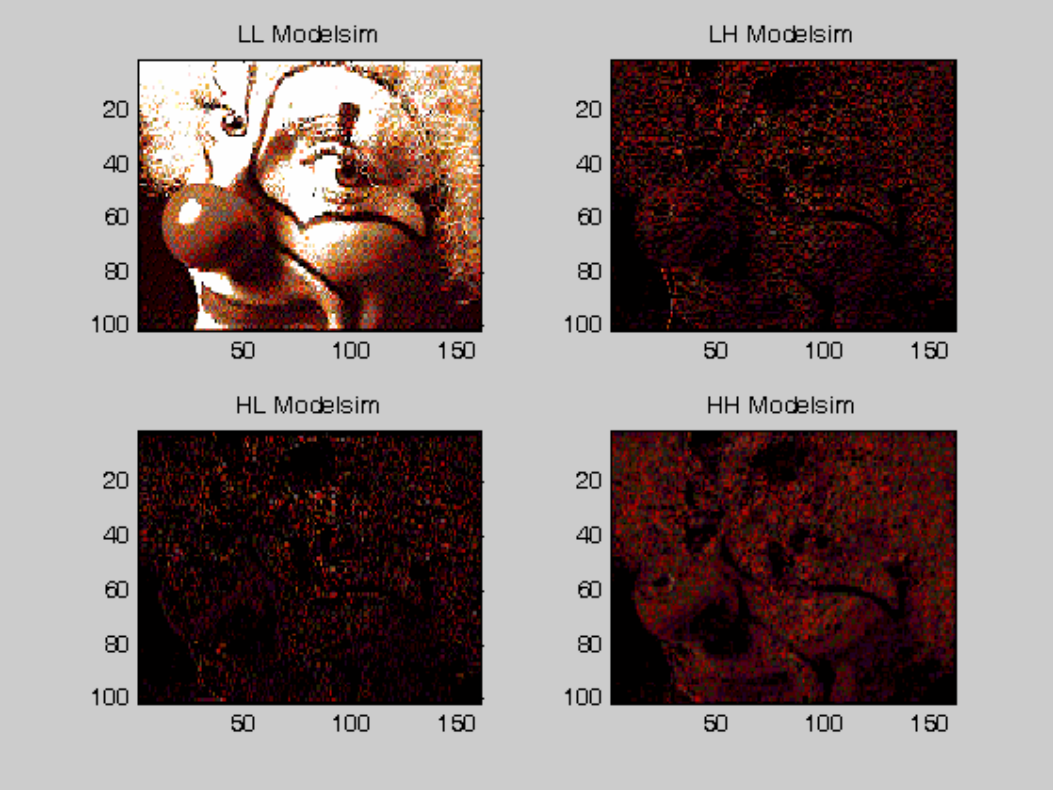
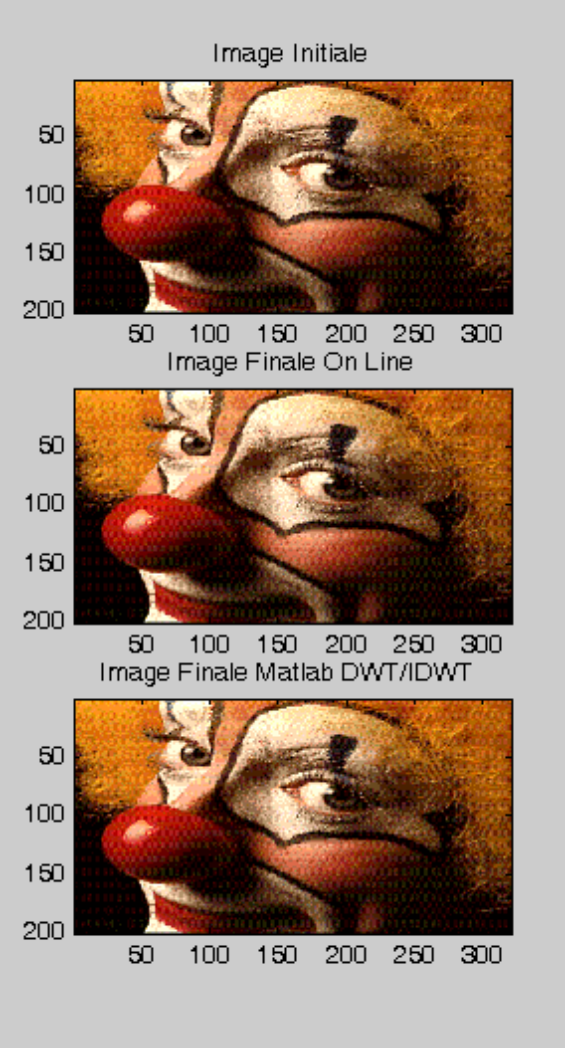


Image originale du clown [2]
(Taille de l'image 256.256)



Résultats de décomposition sous Modelsim
des sous bandes LL, LH, HL et HH

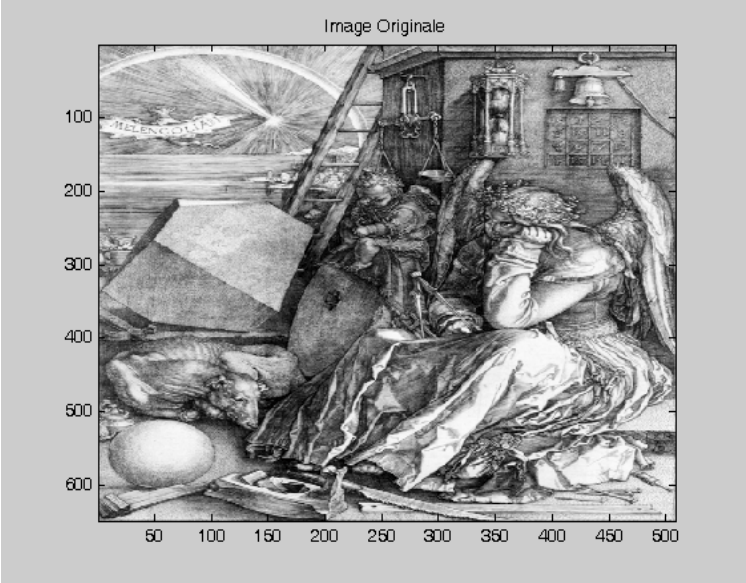
Après décomposition de l'image originale en sous bandes LL(image approximatif), LH (image détail verticale), HL(image détail horizontal) et HH. (image détail diagonale) on obtient les résultats au dessus.



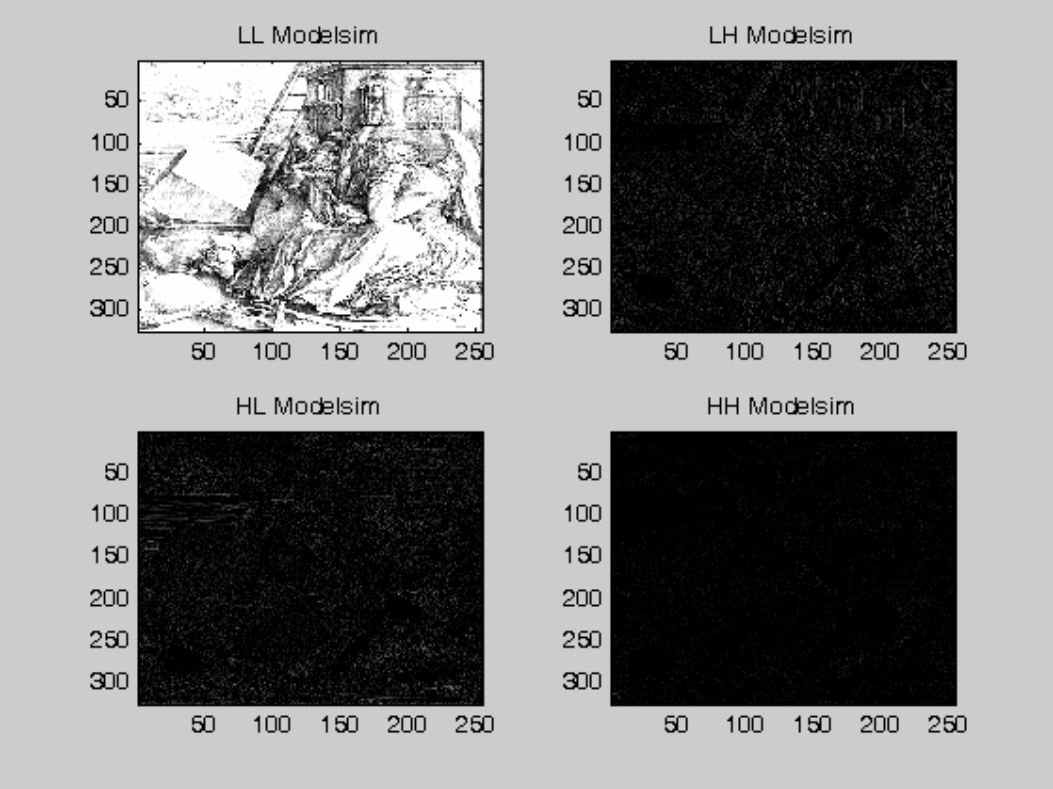
Résultats de simulations de la reconstruction

de l'image originale sous Modelsim et Matlab

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de7.9936 10^{-14} L'image originale du clown.

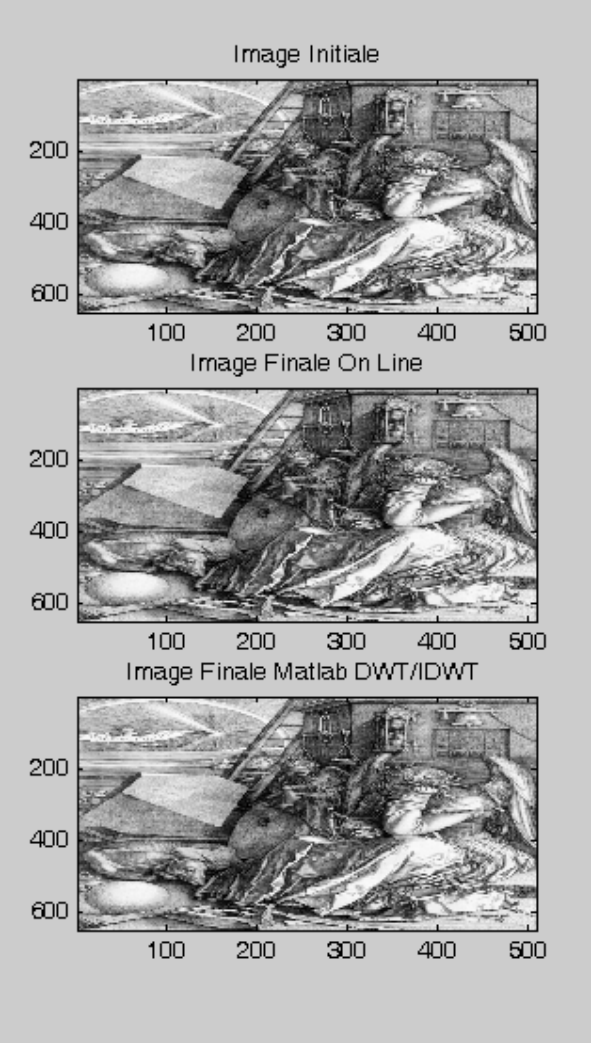


L'image originale de DURER [2]
(Taille de l'image 648.508)



Résultats de décomposition sous Modelsim des
sous bandes LL, LH, HL et HH

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de$7.9936\ 10^{-14}$ L'image originale de DURER.



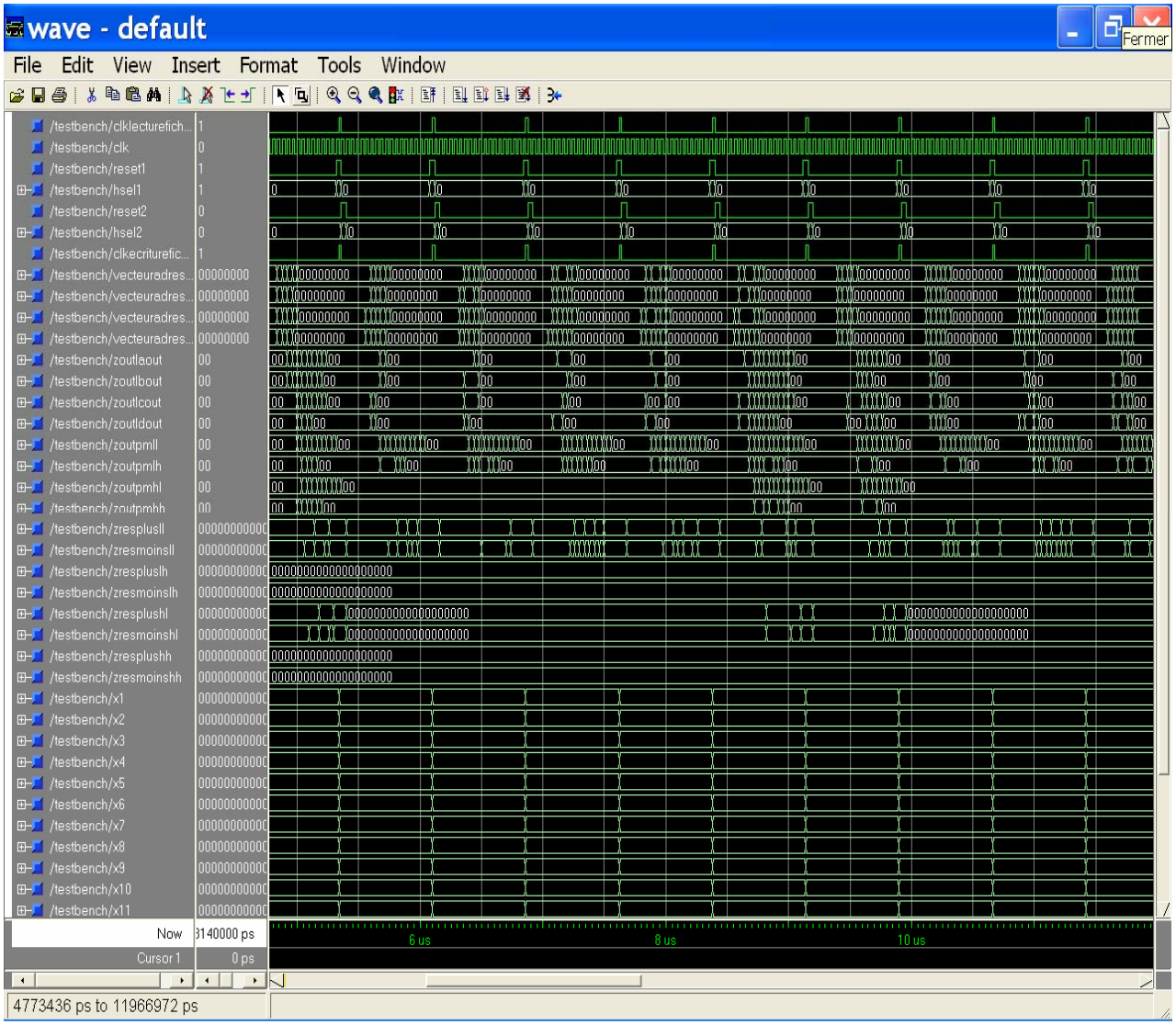
Résultats de simulations de la reconstruction de

l'image originale sous Modelsim et Matlab

A partir des résultats de simulations de la reconstruction de l'image originale sous Modelsim et Matlabdes des 4 sous bandes le taux d'erreur est de$3.312\ 10^{-14}$ L'image originale de DURER.

APPENDICE H

CHRONOGRAMME DE SIMULATION



APPENDICE I

JPEG2000

JPEG2000 est le nouveau système de codage d'image, utilisant l'état de l'art des techniques de compression basé sur la transformée en ondelettes. Elle est appropriée à un grand nombre d'applications depuis les appareils photos digitaux jusqu'a l'imagerie médicale et d'autre secteurs clé [63].

Les domaines d'application de JPEG 2000 englobent :

- L'Internet
- La photographie Digitale
- L'Imagerie médicale
- L'Imagerie sans fil
- L'Imagerie de documents
- La Pré-Presse
- Le Scientifique et l'Industriel
- Les archives d'images et les bases de données
- La Surveillance
- L'Impression et le Scannage

L'imagerie médicale

JPEG 2000 possède de nombreuses caractéristiques qui sont utiles à l'imagerie médicale. De l'information de fond concernant ce domaine est couvert par un document du comité JPEG (N2782) qui donne également des informations utiles à la compréhension du fonctionnement de JPEG 2000. Un aspect clé qui concerne souvent la profession médicale est la nécessité d'assurer que l'image sera communiquée sans pertes, et sans l'introduction de distorsions lors du processus de compression qui puisse mener à un diagnostique intermédiaire. Ceci entraîne souvent la manipulation d'images de grande taille, difficiles à stocker et transmettre. JPEG 2000 peut être utilisé pour encoder des images complètement (ou partiellement) sans perte, et fournit de bonnes capacités de compression (semblable, par exemple, à la méthode optimisée de compression du standard original JPEG, JPEG-LS (IS 14495)). Ce dernier ne possède cependant pas de fonctionnalités supplémentaires, rendant JPEG 2000 particulièrement attractif pour l'imagerie médicale :

l'Internet

De nombreuses applications clé du nouveau standard JPEG 2000 utiliseront l'Internet et les technologies de l'Internet pour distribuer des images. Les images JPEG 2000 ont un certain nombre de propriétés qui les rendront très appropriées à l'usage d'Internet. Typiquement, les utilisateurs de l'Internet sont contraints de télécharger des images larges, de très grande qualité à cause de la taille physique de leur fichier. Des fournisseurs d'images doivent souvent créer trois versions ou plus d'une image, allant d'une icône à la taille d'une page.
Les caméras digitales ont augmenté en qualité et en résolution jusqu'à un niveau où elles sont entrées effectivement en concurrence avec les films traditionnels. Les images qu'ils génèrent ne sont souvent plus directement appropriées au déploiement de l'Internet la qualité et la place

sont gaspillées sur les moniteurs traditionnels d'ordinateurs. Ceci est en partie dû au fait que les moniteurs ne peuvent souvent montrer qu'une partie de l'image capturée sans devoir la faire défiler, et en partie parce que le rendu des couleurs du moniteur ne correspond pas à celui de la caméra.
Ces deux problèmes sont traités par les standards JPEG 2000. Les images sauvegardées sous le format JPEG 2000 peuvent être codées de telle sorte que lors du transport et de la visualisation de données, la résolution augmente graduellement, en commençant par une icône, ou augmentant progressivement en qualité et l'utilisateur peut arrêter la transmission d'image un fois qu'il dispose d'assez de détails pour faire leur prochain choix, vu que les données sont ordonnées dans le fichier de telle manière que la transmission aux serveurs d'images est facilitée.
Par exemple, www.mapquest.com utilise déjà une architecture JPEG 2000 pour transmettre une grande partie des photographies liées à son application cartographique.

La photographie digital

La photographie a changé la manière avec laquelle les personnes enregistrent et se souviennent d'images, d'évènements et d'informations scientifiques. La photographie digitale a commencé par la venue de la première caméra digitale commercialisée pour les consommateurs et les professionnels au début des années '90, ainsi que le premier système de digitalisation des images de films. Avec l'évolution des technologies, le coût des caméras digitales et des services de digitalisation a fondu, et la qualité des images a augmenté. La taille des images des caméras digitales professionnelles continue à augmenter, allant d'environ 1 Megapixel en 1993 à plus de 10 Megapixels et plus en 2003.
La majorité des caméras digitales vendues ces 10 dernières années utiliser JPEG. La standard actuel continuera à avoir une grande place dans le marché des caméras digitales pour les consommateurs. Il est prévu que JPEG 2000 s'insère dans ce marché, avec l'évolution des caméras digitales.

L'imagerie de documents

Les applications d'imagerie de documents sont souvent soumises au compromis qualité/compression. Avec l'évolution de la technologie, et l'arrivée de la couleur dans de nombreuses normes de formats de publication, les attentes en qualité des utilisateurs ont également augmenté. Vu les exigences requises pour une visualisation fidèle sur écran, de même que l'aptitude à imprimer des fac-similés de haute qualité d'un document original, les besoins en compression présentent souvent des problèmes. De nombreux documents possèdent des zones qui peuvent être codées plus efficacement avec un format texte (permettant une compression et un indexage optimal), accompagnées de photographies, de graphiques et d'autres types d'images.

Une développement spécifique au sein de l'ensemble de standards JPEG 2000 est le Format de Fichier d'Image Composé (Compound Image File Format) 'JPM' défini dans la Partie 6 du standard JPEG 2000. Cette extension au format de fichier JPEG 2000 basique (défini dans les Parties 1 et 2 du Standard) propose des concepts et des syntaxes nécessaires pour gérer les documents composés codés sous le format MRC.

Le Scientifique et l'Industriel

De nombreuses applications du secteur scientifique et industriel se tournent actuellement vers l'usage de matériels de traitement d'images pour remplacer ou pour accroître des données déjà existantes. On peut citer la liaison de l'imagerie photographique satellite ou aérienne à des systèmes de cartographie, et la facilité d'utilisation de caméras digitales pour mettre en évidence le travail satisfaisant qui a été réalisé. Ces projets dépendent de l'ubiquité et de la disponibilité du standard, et utilisent souvent des solutions bien testées. La disponibilité généralisée des caméras digitales, et des software PC utilisés pour visualiser et imprimer les résultats continueront à permettre aux prix d'être assez bas - argument qui est spécialement important dans des secteurs où l'utilisation de ces dispositifs est fortement lié à la rentabilité de l'entreprise.

Cependant, il est assez commun d'avoir différentes versions des images du même article. Par exemple, un fabricant de voitures peut avoir des photographies du moteur d'un véhicule utilisées dans des applications de service, marketing, assurance qualité, entraînement et test. Alors que la gestion du capital digital est reconnue comme étant une fonction important qu'une entreprise devrait utiliser, plus d'attention devrait être portée sur la manière de réutiliser et de re-destiner ce capital digital, tel que les images. JPEG 2000 offre des nombreuses fonctionnalités intéressantes dans ce contexte une gestion personnalisée des couleurs, un compression permettant d'obtenir une version avec et sans pertes dans le même fichier, et des options étendues permettant d'ajouter des métadata standard et définies par l'utilisateur à un fichier d'image.

De plus, de nombreux aspects de l'utilisation scientifique et industrielle impliquent un traitement conséquent de l'image digitale, par exemple pour accroître les fonctionnalités ou compter des articles. En utilisant n'importe quelle forme de compression avec perte pour des images dans ce contexte, des problèmes pourraient surgir - après tout, l'information abandonnée durant la compression avec perte est généralement l'information imperceptible à l'oeil humain - parce que les même caractéristiques que le software de traitement d'image ne seraient pas spécialement montrées. Il devient donc plus important d'assurer que le matériel archivé soit stocké avec la plus grande fidélité possible - mais qu'il soit toujours facilement exploré et visible durant une étape de pré-traitement par exemple. De nouveau, JPEG 2000 peut offrir des avantages importants dans cet environnement.

Des boites à outils software sont disponible chez un certain nombre de vendeurs qui supportent le nouveau standard JPEG 2000. Celles-ci vont de la version software gratuite Jasper (C) et JJ2000 (Java) qui est lié à la Partie 5 du standard JPEG 2000 (Software de Référence), à des alternatives commerciales de KakaduSoft, Aware, Algovision Luratech, Leadtools, Pegasus et d'autres. Ceux-ci permettent l'intégration de fonctionnalités globales de JPEG 2000 dans une vaste gamme de produits et systèmes.

L'Imagerie sans fil

Les communications sans fils possèdent leurs problèmes particuliers. Plus précisément, les réseaux sans fil sont caractérisés par de fréquentes erreurs de transmissions de même qu'une largeur de bande étroite. C'est pourquoi, elles imposent d'importantes contraintes à la transmission d'images digitales. Puisque JPEG2000 propose une grande efficacité de

compression, il est un bon candidat pour les applications multimédia sans fil. JPEG2000 permet un grand nombre de stratégie de qualité de service (QoS) pour les opérateurs de réseaux. Pour être adopté largement pour les applications multimédia, JPEG2000 doit être robuste aux erreurs de transmission. Afin de résoudre ce problème, le comité JPEG a établi une nouvelle tâche de travaili, JPEG 2000 Sans Fil (JPWL - JPEG2000 Wireless), en tant que Partie 11 du strandard. Son but est de standardiser des outils et des méthodes permettant une transmission efficace d'imagerie JPEG2000 sur des réseaux sujets aux erreurs.
En utilisant les techniques standardisées pas JPWL, JPEG2000 devient très résistant aux erreurs de transmission. C'est pourquoi, JPEG2000 est le candidat idéal pour la transmission efficace d'images digitales et de vidéo pour des applications sans fil. Bien que les solutions proposées ne soient pas ajustées à un protocole réseau spécifique, une attention particulière a été dédiée à trois cas d'utilisation important : la 3eme génération de réseau de téléphonie sans fil (3GPP/3GPP2), WLAN (IEEE 802.11 family of standards) et Digital Radio Mondiale (DRM).

L'Impression et le Scannage

Un problème clé (davantage pour les utilisateurs PC que les utilisateurs Apple Mac) a été la difficulté de faire concorder les données sortant des scanners, des caméras digitales et des autres dispositifs de saisie aux moniteurs et imprimantes. Alors qu'il y a d'énormes progrès technologiques dans ce domaine, et qu'il est certainement possible de travailler dans un environnement possédant un calibrage couleur pour un petite gamme d'équipement d'utilisateurs, il est quasiment impossible de gérer la plupart des fichiers qu'un utilisateur reçoit efficacement et fidèlement. L'arrivée de JPEG 2000 changera cette situation, puisqu'une partie obligatoire du standard impose que les équipements générant des fichiers JPEG 2000 définissent comment la couleur doit être traitée au sein du fichier d'image.

JPEG 2000 possède de nombreux bénéfices intéressants. Les utilisateurs pourraient sélectivement couper, réduire la profondeur en bit ou la résolution, et choisir de compresser ces images de telle sorte que des centaines de ces images pourraient être stockées sur un média adéquat tout en gardant la possibilité de garder autant de qualité que nécessaire pour certaines images clé.

La Surveillance

La technologie traditionnelle de surveillance a mis un certain temps avant d'adopter le traitement d'image digitale. Cela est du d'une part aux méthodes de stockage de grande ampleur que les vastes volumes de données ont requis et, d'autre part, au coût excessif de passage à l'imagerie digitale. Cependant, durant ces dernières années, ces coûts ont sérieusement diminué alors que la puissance de calcul et les capacités se sont améliorées tout autant. Cela permet de se pencher sur la plupart des applications traditionnelles de surveillance, tout en considérant les soucis actuels de notre société en matière de vie privée et d'intrusion.

Beaucoup de ces aspects démontrent l'utilité potentielle de JPEG 2000 dans ce domaine:

- L'utilisation de Motion JPEG 2000 a des avantages évidents lorsqu'il s'agit de capturer des séquences d'actions, pour lesquelles l'aperçu initial peu être à basse résolution et

de plus grande résolution lorque l'on change de moniteur de contrôle, augmentant le débit de frames et en ajoutant plus de métadonnées et des régions d'intérêt**.**

- Les formats de fichiers définis pour JPEG 2000 permettent de stocker avec l'image des données standardisées mais aussi de l'utilisateur.
- Les nouvelles parties de JPEG 2000 augmentent sont utilité en considérant la sécurité, la communication efficace entre serveur et client, et la possibilités de mettre en oeuvre ces proporiétés dans un environnement sans fil sensible aux erreurs.
- En tant que standard, le coût d'implémentation de la technologie devrait être considérablement plus attrayant que dans le cas d'une technologie propriétaire avec moins de risques.

Le pré-presse

Le pré-presse est le processus utilisé lorsque des fichiers digitaux sont préparés pour l'impression. Deux conditions clés pour ce processus sont la fidélité et la régularité. Dans le passé, l'industrie de la pré-presse dépendait de la compression d'images lossless (par exemple en utilisant les formats de fichiers EPS ou TIFF) et du calibrage des couleurs de tous les composantes de ce processus, utilisant des conditions pré-définies d'éclairage et de visualisation afin d'obtenir des résultats optimaux.

JPEG 2000 offre des opportunités pour l'industrie de la pré-presse afin de substituer les formats traditionnels par les aspects plus avancés de JPEG 2000, et de réutiliser son contenu pour lui permettre d'être utilisé dans des publications dans Internet ou d'autres contextes. La même image JPEG 2000 peut générer des icônes, des images d'écran et imprimer du matériel simplement en tronquant le codestream à différentes endroits. De plus, la gestion et l'association puissante des métadata dans les fichiers JPEG 2000 signifie que les processus de « Digital Asset Management » peuvent être facilement liés à des transmissions de pré-presse, fournissant de la sécurité au photographe, créateur d'images et imprimeur.
Une facette clé est la possibilité qu'à JPEG 2000 de fournir une vrai compression sans pertes d’images.

REFERENCES

1. Jaideva C. Goswami et Andrew K. Chan. « Fundamentals of Wavelets : Theory Algorithms, and Applications». Wiley Series in Microwave and Optical Engineering. Wiley Interscience.
2. Matlab, "Wavelet Toolbox: Wavelets: A New Tool for Signal Analysis: Five Easy Steps to a Continuous Wavelet Transform", Mathworks Inc.
3. C.Diou, «Contribution à l'intégration sur silicium de la transformée en ondelettes application au traitement d'images », Thèse de doctorat université de Montpellier II Décembre. 2002.
4. Stéphane.G.Mallat, « A theory for mutiresolution signal decomposition: The wavelet representation", IEEE transactions on pattern Analysis and Machine Intelligence, Vol II , N°07 July 1989.
5. J.Auvray, « Cours de Mr Jean Auvray- Traitement du Signal 2001-2002 », Université Pierre et Marie Curie.
6. D. Sripathi, « Efficient implementations of discrete wavelet transforms », Master thesis of science, 2003.
7. Strang, G., T. Nguyen," Wavelets and Filter Banks", Wellesley-Cambridge Press, 1996.
8. A. Lewis, G. Knowles "Image Compression Using the 2-D Wavelet", IEEE Transactions on Image Processing, Vol. 1 No. 2 . pp. 244-250. 1992
9. H.Bessalah, F.Alim, S.Seddiki " Implementation du filtre de S.Mallat en mode de calcul en ligne » 5^{eme} colloque africain sur la recherche en informatique Antananarivo Madagascar octobre 2000.
10. M. D. Ercegovac, "On-line Arithmetic for Recurrence Problems", SPIE vol. 1556 . Signal. Proccessing (1991).
11. H.Bessalah, «Multidimensional pipeline and array architectures for the fast orthogonal transforms", Processing. Of Parallel Computing Technologies PaCT-93, Obninsk, Russia, 1993.
12. P. Kang-Guo Tu «On-Line Arithmetic Algorithms for Efficient Implementation», Thèse de doctorat de l'université de Californie, Los Angeles, 1990.
13. J.C. Bajard « Evaluation de Fonctions dans des Systèmes Redondants d'Ecriture des Nombres », Thèse de doctorat de l'ENS Lyon, Février 1993.

14. A.Avizienis, «Signed-Digit number representations for fast parallel arithmetic»,IRE Transactions on Electronic Computers, p:389–400, September 1961.
15. Jean-Luc Beuchat,« Etude et conception d'opérateurs arithmétiques optimisés pour circuits programmables », thèse, Ecole polytechnique fédérale de Lausanne, 2001.
16. .H.Bessalah, F.Alim"On line algorithm and architecture of null delay multiplier" Engineering and industrial application ACIDCA 2000 Sfax Tunisia.
17. A.Skaf, « Conception de Processeurs Arithmétiques Redondants et En Ligne », Thèse de doctorat de l'institut national polytechnique de Grenoble, Septembre 1995.
18. A.Guyot, Y.Kusumaputri, J.M.Muller «Ocapi: Opérateur de Calcul à Précision Illimitée », rapport technique, Toulouse, Juin 1989.
19. H.Bessalah, « Application of the half line mode of computation to digital signal processing ».the proceedings of the 7 th international conference on signal processing application et technology. Boston,Massachusetts,USA October 1996
20. H.Bessalah, F.Alim, S.Seddiki « Implementation of the hough transform by the on line mode »Vipromcom 02; 4th EURASIP IEEE region 8 international symposium on video/image processing and multimedia communication Zadar ,Croatia.
21. H. Bessalah, D.Belhandouz, A.Guessoum, "Modélisation d'algorithmes machines de calcul en modes on-line et Half-line", Institut de Mathématiques, USTHB, 1984.
22. H. Bessalah, «High halh line VLSI computation for images compression »the sixth international conference on microelectronics .marmara research centre Istanbul Turkey 1994.
23. L.Bossuet, « Modélisation d'architectures reconfigurables embarquées» mémoire de DEA université bretagne sud, juin 2001.
24 Chaitali Chakrabarti, Mohan Vishwanath, et Robert Michael Owens. « Architectures for the Wavelet Transforms : A Survey ». Journal of VLSI Signal Processing, p:171–192,1996.
25 Mohan Vishwanath, Robert Michael Owens, et Mary Jane Irwin. « VLSI Architectures for the Discrete Wavelet Transform ». IEEE Transactions on Circuits and Systems – II :Analog and Digital Signal Processing, p :305–316, Mai 1995.
26 N. M. Nasrabadi and R. A. King, Image coding using vector quantization : review, IEEE Transactions on Communications, pp. 957-971, 1988:
27 http ://www.alma-tech .com
28 . Elkefi, M. Antonini compression de maillage 3D multi résolution transformée en ondelettes 2ème génération "Rapport de recherche ISRN, novembre 2003.

29 Frédéric Truchetet et André Forys. « Implementation of still-image compression decompression scheme on fpga circuits ». *SPIE*, Janvier 1996.

30 Philippe Guermeur et Stéphane Guermeur. « Une implantation de la transformée en ondelettes discrètes à base de circuits FPGA ». In Actes des Journées Adquation Algorithme Architecture en Traitement du Signal et Images, pages 201–206, Saclay, France, 28-30janvier 1998.

31 Frantz Lohier « Méthodologies de programmation et évaluation des processeurs de Traitement de signal parallèles pour letraitement d'images en temps réel thèse de L'université pierre et marie curie (paris 6)- spécialité : robotique Février 2000.

32 R. Ziani, N.Abbes, « Etude et implémentation de la multi résolution par transformée en ondelette appliquée à la segmentation d'images par extraction de contours », Blida décembre 1997.

33 B. Torresani, « Introduction à l'analyse continue par ondelettes », Ecole d'été d'ondelette 1993, INICA, Toulouse France.

34 Y.Meer, « Les ondelettes, algorithme et application », édition Armand Collin, 1992.

35 A.Cohen, « Ondelettes et traitement numérique du signal », Collection RMA (Recherche en Mathématiques Appliqués)

36 www.wavelet.org

37 M.Lapointe, « Architecture concurrente et applications à des réalisations rapides de filtres numériques invariants et adaptatifs », Thèse de Doctorat, Université Laval, Août 1990.

38 http://www.lpmi.uhp-nancy.fr/realisation/filtre/index.html

39 http://www.di.unipi.it/~boerger/ASMTutorialEtaps.html

40 http://www.bridgeport.edu/~matanya/vlsi/ictutor.html

41 http ://www.univ-reunion.fr

42 http ://www.lirmm.fr

43 http ://www.amphion.com

44 http://www.enseignement.polytechnique.fr

46 Laure Portal et Marie Lozano. « Implémentation d'un algorithme de compression d'images basée sur la théorie des ondelettes ». Projet industriel de fin d'études, Institutdes Sciences de l'Ingénieur de Montpellier, Université Montpellier II, 1997.

47 .Sripathi, «Efficient implementation of discrete wavelet transforms using FPGA's» These de Doctorat, The Florida state university, 2003.

48 R.Grisel, «Les circuit FPGA, Field programmable gate array », Cours de l'université Picardie jules,

49 F.N.Abdesselam, « Conception, Synthese et Test d'IP pour la transformée en ondelettes sur FPGA », DEA en électronique, ENSTA de Paris.

50 http ://www.xilinx .com

51 G. Sassatelli. Architectures Reconfigurables Dynamiquement pour les Systèmes surPuce. Ph.D. Thesis, Université de Montpellier, France, April 2002.

52 Lucazeau, B. Trézéguet, « ASIC : Circuits intégrés pour application spécifique » Technique de l'Ingénieur, traité électronique, vol E, pp 19-4.

53 Xilinx, « Synthesis and simulation Design Guide », Xilinx INC, USA.1998.

54 Xilinx,« Virtex ^{TM}-II Platform FPGAs: Introduction: Introduction and Overview», advance product specification, DS031-1(v1.9) September26, 2002.

55 [XIL98] Xilinx. Synthesis and simulation design guide. Xilinx Inc, USA. (1998).

56 Logic synthesis and verfication algorithms "Gary D Hachtel-Fabio Somenzi" edition KAP.

57 L. Minter, "The role of distributed arithmetic in FPGAs ", Xilinx corporation.

58 D.Wilde, L.De.Brummer, "Multi Operand Addition", spring, 2002.

59 M. Jézéquel, « Circuits combinatoires », ISP304, Département Électronique

60 Issam. A.n, "Outils de CAO pour la génération d'opérateurs arithmétiques auto contrôlables », Thèse de Doctorat de l'INPG, 2001.

61 N. Tredennick, B. Shimamoto. The Rise of Reconfigurable Systems. In proceeding of Engineering of Reconfigurable Systems and Application, ERSA 03, Las Vegas, Nevada, USA. June 23-26, 2003

62 L. Dutrieux, D. Demigny, Logique programmable, Architecture des FPGA et CPLD, Méthodes de conception, le langage VHDL, Eyrolles, Paris, 1997.

63 http://www.jpeg.org/apps

.