Contribution à la définition, à l'optimisation et à l'implantation d'IP de traitement du signal et des données en temps réel sur des cibles programmables

Yousri Ouerhani

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Sciences et Technologies de l'Information et de la Communication - Spécialité Traitement du Signal et des Images
École Doctorale SICMA

Présentée par
Yousri OUERHANI
Préparée dans l'équipe VISION
ISEN Brest

Contribution à la définition, à l'optimisation et à l'implantation d'IP de traitement du signal et des données en temps réel sur des cibles programmables

Thèse soutenue le 16 novembre 2012
devant le jury composé de :

Badr-Eddine BENKELFAT
Professeur, Telecom & Management SudParis / *Rapporteur*

Denis HAMAD
Professeur, Université du Littoral Côte d'Opale / *Rapporteur*

Ayman ALFALOU
Professeur, ISEN Brest / *Directeur de la thèse*

Maher JRIDI
Enseignant Chercheur, ISEN Brest / *Encadrant de la thèse*

Christian BROSSEAU
Professeur, Université de Bretagne Occidentale / *Président*

Jihad ZALLAT
Professeur, Université de Strasbourg / *Examinateur*

Bruno ROLLAND
Président d'Interface Concept / Invité

Laurent UHEL
Ingénieur Interface Concept / Invité

université de bretagne occidentale

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Sciences et Technologies de l'Information et de la Communication - Spécialité Traitement du Signal et des Images
École Doctorale SICMA

Présentée par
Yousri OUERHANI
Préparée dans l'équipe VISION
ISEN Brest

Contribution à la définition, à l'optimisation et à l'implantation d'IP de traitement du signal et des données en temps réel sur des cibles programmables

Thèse soutenue le 16 novembre 2012
devant le jury composé de :

Badr-Eddine BENKELFAT
Professeur, Telecom & Management SudParis / *Rapporteur*

Denis HAMAD
Professeur, Université du Littoral Côte d'Opale / *Rapporteur*

Ayman ALFALOU
Professeur, ISEN Brest / *Directeur de la thèse*

Maher JRIDI
Enseignant Chercheur, ISEN Brest / *Encadrant de la thèse*

Christian BROSSEAU
Professeur, Université de Bretagne Occidentale / *Président*

Jihad ZALLAT
Professeur, Université de Strasbourg / *Examinateur*

Bruno ROLLAND
Président d'Interface Concept / Invité

Laurent UHEL
Ingénieur Interface Concept / Invité

Remerciements

Je ne saurais présenter cette étude sans remercier tous ceux qui ont contribué à son aboutissement.

Je tiens tout d'abord à remercier le professeur Christian BROSSEAU de m'avoir faire l'honneur de présider le jury de ma thèse. Je remercie également les professeurs BENKELFAT Badr-Eddine et HAMAD Denis pour l'attention qu'ils ont accordée à la lecture de ce manuscrit ainsi que pour les remarques pertinentes qu'ils ont soulevées.

Toute ma gratitude et mes remerciements vont à mes encadrants Messieurs ALFALOU Ayman, Professeur à l'ISEN Brest, et JRIDI Maher Jridi, Enseignant Chercheur à l'ISEN Brest . Travailler avec eux a été à la fois très enrichissant scientifiquement et facile humainement grâce à leurs conseils, leur disponibilité, leur confiance et leur assistance.

Des remerciements particuliers vont à Monsieur Roland Bruno, président d'Interface concept, pour l'accueil chaleureux qu'il m'a adressé au sein de l'entreprise, grâce à qui j'ai pu réaliser des tests expérimentaux, mais aussi pour avoir fait le déplacement pour participer à mon jury de thèse. Les compétences scientifiques de Monsieur Laurent Uhel et les discussions que nous avons eu ensemble tant au plan professionnel et personnel m'ont été très précieuses.

es trois années de thèse n'auraient pas été particulièrement réussies sans l'ensemble des permanents et des doctorants que j'ai eu l'occasion de rencontrer à l'ISEN et qui ont contribué à créer une ambiance de travail très agréable.

Un énorme merci va à mes amis qui m'ont soutenu, ils sauront se reconnaitre.

Pour finir, je tiens à exprimer ma très profonde gratitude et mes plus grands remerciements envers mes parents et mes sœurs, pour leur amour, leur confiance, leur soutien, leurs encouragements,.., je ne saurais l'exprimer à sa juste valeur au travers de ces quelques lignes mais : merci à vous.

Table des matières

Table des figures

Liste des tableaux

Acronymes

AC	Alternative Component
ASIC	Application-Specific Integrated Circuit
AUC	Area Under Curve
BPOF	Binary POF
CAN	Convertisseur Analogique-Numerique
CCD	Charge-Coupled Device
CLB	Configurable Logic Bloc
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DC	Direct Component
DCT	Discret Cosine Transform
DFT	Discret Fourier Transform
DIF	Decimation in Frequency
DIT	Decimation in Time
DSP	Digital Signal Processor
DWT	Discret Wavelet Transform
ELINT	ELectronic INTelligence
EQM	Erreur Quadratique Moyenne
ESM	Electronic Support Mesures
FFT	Fast Fourier Transform
FIFO	First in First Out
FN	False Negative
FP	False Positive
FPGA	Field Programmable Gate Array
FPR	False Positive Rate
GE	Guerre Electromagnétique
GPGPU	General Purpose GPU
GPP	General Purpose Processor
GPU	Graphical Processing Unit
ICON	Integrated CONtroller
IFFT	Inverse Fast Fourier Transform
ILA	Integrated Logic Analyzer
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
JPEG	Joint Photographic Experts Groups
JTAG	Joint Test Action Group
JTC	Joint Transform Correlator
LSB	Least Significant Bit
LUT	Look Up Tables
MEMS	Micro-Electro-Mechanical Systems
MIPS	Million Instructions per Second
MOF	Maximum Operating Frequency

MPEG	Moving Picture Experts Group
MSB	Most Significant Bit
MSE	Mean Squared Error
PAL	Programmable Array Logic
PCE	Peak to Correlation Energy
PE	Processing Element
POF	Phase only Filter
PSNR	Peak Signal to Noise Ratio
RAM	Random Access Memory
RF	Radio Frequency
ROC	Receiver Operating Characteristic
ROM	Read Only Memory
RSBQ	Rapport Signal à Bruit de Quantification
SFDR	Spurious Free Dynamic Range
SIP	System in Package
SNR	Signal to Noise Ratio
SPOF	Segmented POF
SVM	Support Vector Machines
TCD	Transformée en Cosinus Discrète
TF	Transformation de Fourier
TN	True Negative
TP	True Positive
TPR	True Positive Rate
VHDL	(Very High Speed Integrated Circuit) Hardware Description Language
VLC	Vander Lugt Correlator
WB	Wide Band

Introduction

Les événements qui ont submergé le monde ces dernières années ont montré encore une fois l'importance de l'image numérique. En effet, l'image porteuse d'informations objectives, facilement transmise sur des réseaux sociaux, a contribué au soulèvement des peuples contre les dictatures les plus coriaces. De plus, cette image a permis de découvrir les atrocités commises et a contribué à une meilleure compréhension de la situation de ces pays.

D'un autre côté, nous pouvons aussi constater l'intérêt grandissant de l'utilisation de l'image dans la littérature grâce à l'implantation optique des algorithmes de traitement d'images. En effet, depuis les années 60, les implantations optiques des applications de traitement d'images comme la reconnaissance de formes (et plus récemment la compression et le cryptage) ont connu un succès croissant. Plusieurs travaux se sont intéressés à ce sujet. Nous pouvons citer par exemple le montage de base du corrélateur optique proposé par "Vanderlugt" pour la reconnaissance de formes [2] et les schémas implantant optiquement la compression JPEG proposés dans [1]. Nous pouvons aussi citer les travaux développés dans notre laboratoire portant sur la corrélation [3].

Ce succès est réalisé grâce à trois principaux facteurs. D'abord, l'émergence de ces applications dans plusieurs domaines tels que la sécurité, le suivi, la transmission, etc. Ensuite, la réduction des temps de réponse de ces applications grâce à l'utilisation des composants optiques. Enfin, les percées technologiques aux niveaux de la fabrication des modulateurs spatiaux de lumière et des interfaces opto-électroniques ont facilité le lien avec des systèmes électroniques.

En dépit de ce succès, le traitement optique de l'information suscite aujourd'hui moins d'intérêt que dans les années 80-90. D'un côté, ceci est dû à l'encombrement des réalisations optiques (frein pour des systèmes embarqués), la qualité des images traitées, les pertes de performances entre les différents étages pour un système tout optique, sans oublier le coût des composants optiques. D'un autre côté, les réalisations optiques ont eu du mal à résister à la concurrence des circuits électroniques.

En effet, ces dernières années, nous assistons à une révolution technologique qui a permis à l'électronique numérique d'enrichir notre quotidien avec des systèmes aussi multiples qu'utiles. Ces systèmes sont omniprésents dans plusieurs domaines allant de la santé à la sécurité en passant par les télécommunications et les divertissements. Une telle diversité des applications est due à la possibilité d'embarquer une grande capacité de calcul de moins en moins chère.

La rencontre entre l'électronique numérique et le traitement d'images a donné naissance à une nouvelle thématique connue sous le nom de traitement d'images embarqué. Cette thématique est en plein essor et le développement des circuits reconfigurables tels que les FPGA "Field Programmable Gate Array" a contribué à ce succès. En effet, les industriels fabricants de composants FPGA ont réussi à faire de ces derniers les circuits incontournables pour l'implantation d'algorithmes de traitement d'images. Ceci a été facilité par le nombre important de portes logiques intégrées dans les FPGA (des centaines de millions en 2012) ainsi que par la possibilité de paralléliser plusieurs traitements et d'utiliser des fonctions personnalisables. Néanmoins, pour être plus compétitifs, les contraintes de faibles coûts et de hautes performances ne suffisent plus. En effet, aujourd'hui, les industriels doivent aussi gérer le temps de mise sur le marché de leurs produits.

Afin de réduire le " Time To Market ", deux nouveaux concepts ont vu le jour. Le premier traite l'élévation du niveau d'abstraction pour la conception de systèmes complexes au niveau algorithmique ou au niveau système. Il devient alors préférable lors de la conception de systèmes numériques d'étudier simultanément les aspects algorithmiques et architecturaux. Une attention

particulière est portée sur leurs interactions en vue d'une implantation optimisée en un temps de développement réduit. Le deuxième concept vise aussi à réduire le temps de conception en utilisant des circuits conçus préalablement. Il s'agit de la réutilisation des IP ou "Intellectual Properties". Ces derniers sont des blocs (matériels ou logiciels) génériques et flexibles. Ils sont conçus sous forme de bibliothèques mises à la disposition du concepteur afin de les intégrer dans son design sans recours à les concevoir. Cependant, malgré que les IP commerciales soient optimisées, elles ne sont pas portables. De plus, le plus souvent ces IP sont payantes ce qui entraîne une dépendance entre le concepteur de l'IP et le fabricant de la carte.

C'est dans ce cadre que s'inscrivent les travaux de cette thèse développée en collaboration avec le leader européen de fabrication de cartes : Interface Concept. Dans cette dissertation, nous visons à proposer des implantations numériques des algorithmes optiques de traitement d'images. Pour cette fin, nous nous sommes focalisés, dans une première phase, à une implantation optimisée des IP de traitement d'images et de signal tels que l'IP de la transformée de Fourier (IP FFT) et celle de la transformée en Cosinus discret (IP DCT). En effet, le choix de ces deux transformations est justifié par l'utilisation massive de ces deux transformées (FFT et DCT), dans les algorithmes de reconnaissance de formes et de compression, respectivement. Dans la deuxième phase, nous nous sommes intéressés à des implantations numériques de ces applications de traitement d'images. Ces implantations ont été réalisées en utilisant d'abord des GPU "Graphical Processing Unit" et ensuite des FPGA en instanciant les IP proposées. Ainsi, le but est double : d'une part nous proposons des IP génériques et optimisées qui peuvent être utilisées dans de nombreuses applications. D'autre part, nous visons à accélérer le temps d'exécution des applications de traitement d'images quand elles sont implantées sur les nouvelles générations de circuits programmables afin de se rapprocher de la rapidité des composants optiques.

Ce mémoire est organisé en trois parties.

La première partie expose le contexte de ce travail. En effet, nous étudions dans une première phase les différents algorithmes optiques de traitement d'images employés dans cette thèse. Nous nous intéressons plus particulièrement aux algorithmes de reconnaissance de formes et de compression optique. Ensuite, dans une deuxième phase, nous nous focalisons sur les progrès de la microélectronique (capacité des circuits à intégrer un grand nombre de transistors, évolution des fréquences de fonctionnement). Nous soulignons le fait que les cibles programmables ont profité de ces avancés technologiques. Plus particulièrement, nous nous intéressons à l'architecture et au fonctionnement de deux cibles : les FPGA et les GPU. Nous terminons ce chapitre par une étude portant sur les possibilités d'une implantation numérique des algorithmes de traitement d'images utilisés dans cette thèse.

La deuxième partie de ce mémoire est dédiée à la proposition des IP optimisées pour le traitement de signal et de l'image. Dans le premier chapitre de cette partie, nous nous intéressons à l'IP FFT. D'abord, nous détaillons le principe et l'intérêt de la transformée de Fourier. Ensuite, nous présentons notre conception conjointe algorithme-architecture de la FFT basée sur le partage d'éléments de mémorisation. Enfin, nous finissons ce chapitre par présenter les résultats d'implantation en les comparant avec ceux des IP commerciales et des architectures proposées dans la littérature. Le second chapitre de cette partie est consacré à la conception de l'IP optimisée pour la DCT. Après avoir présenté le principe de la transformation en cosinus discrète en mettant l'accent sur son utilité, nous décrivons l'architecture évolutive proposée pour le calcul des coefficients de la DCT 2D. Nous finissons ce chapitre par présenter la possibilité d'exploiter l'IP proposée par plusieurs standards de compressions d'images et de vidéo.

La troisième et dernière partie de ce cette thèse décrit les résultats des mesures réalisées dans les locaux de l'entreprise Interface Concept lors de l'implantation des applications de traitement d'images et de signal en utilisant les IP FFT et DCT proposées dans la partie précédente. Dans le premier chapitre, nous validons, le fonctionnement de l'IP FFT proposée avec un banc décrivant une chaine de numérisation. Ensuite, nous présentons l'implantation d'une application de guerre électronique utilisant notre IP FFT. Nous mettons l'accent sur l'avantage de l'IP proposée par rapport aux IP commerciales en se basant sur des critères technologiques tels que les rapports surface-temps et consommation de puissance mesurée. Dans le chapitre suivant, nous réalisons

une implantation numérique des algorithmes optiques de la reconnaissance de visage. Dans un premier temps, nous utilisons l'IP FFT pour une implantation sur FPGA de la technique de corrélation. Ensuite, nous proposons un nouvel algorithme optimisé pour la reconnaissance de visage. Le principe de cet algorithme consiste à effectuer une reconnaissance à deux niveaux afin de réduire le temps de calcul. Cet algorithme a été implanté sur GPU en utilisant d'abord des images fixes et ensuite, un flux vidéo dans le cadre d'une application de reconnaissance en temps réel. Nous terminons cette partie avec un dernier chapitre présentant une implantation sur FPGA de la norme de compression JPEG en proposant une architecture basée sur la quantification sélective afin de réduire le nombre d'opérateurs réalisés.

Finalement, une conclusion vient clôturer ce document dans le but de résumer les résultats obtenus pendant cette thèse et de présenter nos perspectives dans le court, moyen et long termes.

Première partie

Contexte de la Thèse

Chapitre 1

Du traitement optique au traitement numérique de l'information

Sommaire

1.1 Introduction

Une partie de la communauté scientifique s'intéresse activement au traitement optique de l'information. Ceci s'explique d'une part par le parallélisme de la lumière et la possibilité de réaliser des transformations quasi instantanément en optique cohérente, et d'autre part par les percés technologiques aux niveaux des interfaces électro-optiques ont retenu l'attention de la communauté scientifique comme en témoigne le nombre grandissant d'articles publiés sur ce sujet [2–8].

En dépit du progrès que connait l'optique, son utilisation pose toujours de nombreux problèmes : portabilité, encombrement, faible résolution, coût élevé des appareils optiques, etc. Pour faire face à ces problèmes et pour profiter des progrès réalisés dans les domaines de la conception et la fabrication des circuits électroniques, il y a eu une migration remarquable vers l'implantation numérique des applications de traitement de signal et de l'image.

Dans ce chapitre nous présentons, dans une première phase, les différentes applications et algorithmes optiques de traitement d'images qui sont traités dans cette thèse. Nous nous intéressons plus particulièrement aux algorithmes de reconnaissances de formes ainsi qu'aux algorithmes de compression optique. Ensuite, dans une deuxième phase, nous nous focalisons sur le progrès technologique et sur la capacité des circuits à intégrer un grand nombre de transistors. En effet, les cibles programmables ont profité de ces avancés technologiques et ont retenu notre attention. Plus particulièrement, nous nous intéressons à l'architecture et au fonctionnement de deux cibles programmables qui sont les FPGA et le GPU. Nous terminons ce chapitre par une étude portant sur les possibilités d'une implantation numérique de ces algorithmes.

1.2 Traitement Optique de l'information

1.2.1 Introduction

Le traitement optique de l'image a connu un grand développement grâce à la capacité d'implanter des algorithmes classiques de traitement du signal et de l'image (reconnaissances de formes, compression, cryptage, etc). En effet, depuis les années soixante nous avons constaté l'apparition de plusieurs travaux implantant optiquement différentes applications en traitement du signal comme par exemple le montage proposé par "Vander Lugt" pour implanter optiquement un système de reconnaissance de formes au travers de la corrélation optique [2]. Récemment, d'autres travaux sont paru et s'intéressent à la reconnaissance de formes et à la compression des images [3, 4]. Cet intérêt est due à plusieurs facteurs.

- Croissance de domaines d'applications de ces travaux. En effet, parmi les domaines d'applications basés sur la reconnaissance de formes on peut citer la sécurité notamment dans les aéroports, le suivi d'objets,etc.
- Le besoin de la compression des images ne cessent de s'amplifier continuellement.
- Le progrès technologique des outils de traitement optique de l'information (grâce à l'apparition d'interfaces comme les modulateurs spatiaux de lumière) [5].

Dans cette partie nous nous intéressons, uniquement, aux algorithmes de reconnaissances de formes et de compression basés sur une transformation de Fourier (TF). Ces deux algorithmes ont été largement étudiés et implantés optiquement par l'équipe Vision du laboratoire L@bISEN. Nous rappelons le principe de ces méthodes et nous discutons les résultats obtenus. Par ailleurs, la technique de base utilisée dans les applications de reconnaissance de formes est la corrélation. Nous étudions ainsi cette méthode et ses différentes architectures.

1.2.2 La Corrélation Optique

1.2.2.1 Principe de la corrélation

La corrélation optique est une des techniques de reconnaissance des formes la plus performante [3]. En effet, elle possède un très bon niveau de discrimination (prise de décisions) et permet de détecter un objet cible. Son principe est très simple et consiste à comparer l'image cible (image à reconnaitre) avec une ou plusieurs images références issues d'une base d'apprentissage. Elle est très utilisée pour implanter optiquement différents applications de reconnaissance de formes depuis l'apparition des algorithmes de Vander Lugt [2], Weaver [6] et Goodman [7] et Horner [8]. Aussi, il faut noter qu'il existe deux principales architectures de corrélateurs, le corrélateur de Vander Lugt VLC [2] et le corrélateur à transformée de Fourier Conjointe JTC [7].

1.2.2.2 Corrélateur de Vander Lugt

C'est le premier corrélateur qui a été publié par Vander Lugt [2] en 1964 et basé sur l'analyse de Fourier. Le principe de ce corrélateur consiste à effectuer deux transformations de Fourier successives réalisées optiquement par le moyen de deux lentilles convergentes et un filtrage dans le domaine spectral. Le synoptique du corrélateur optique Vander Lugt est illustré sur la Figure 1.1.

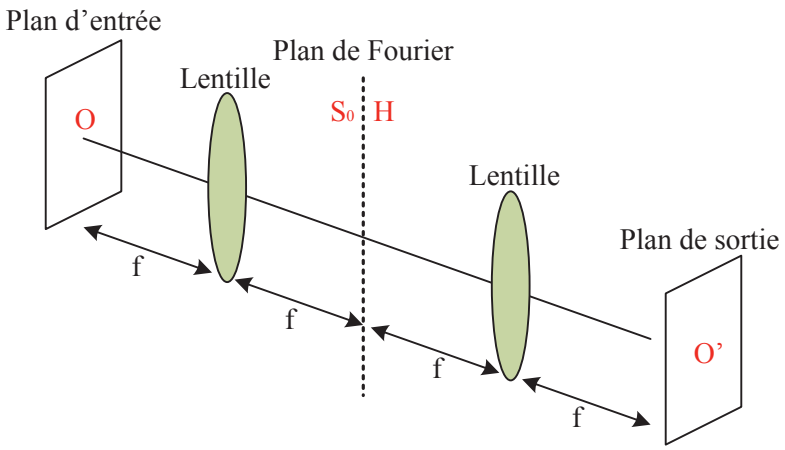


FIGURE 1.1 – Corrélation Optique

L'objet O, supposé plan, est éclairé par une onde plane monochromatique [5]. Une lentille convergente forme S_0 la TF "Transformée de Fourier" de l'objet dans son plan focal image (plan de Fourier ou plan spectral) où est placé un filtre H. Ce dernier est calculé en fonction du traitement désiré. Enfin, une deuxième lentille effectue une seconde TF dans le plan de sortie (plan de corrélation).

Le principe de filtrage consiste à agir sur le spectre de l'objet, donc dans le plan de Fourier, de façon à modifier sa répartition spectrale. Le résultat obtenu est affiché au niveau du plan de sortie aussi appelé plan de corrélation. La prise de décision est basée sur un critère de performance appelé PCE (Peak to Correlation Energy) [9]. Le PCE représente le rapport de l'énergie du pic de corrélation par rapport à l'énergie totale du plan de corrélation).

$$PCE = \frac{Energie\ du\ pic\ de\ corrlation}{Energie\ du\ plan\ de\ corrlation} \tag{1.1}$$

Par ailleurs, il est possible de corréler une image avec toutes sortes de filtres. Ainsi plusieurs formes de filtres on été proposées par les chercheurs afin d'améliorer la robustesse ainsi que la discrimination du corrélateur VLC. Parmi ces filtres, nous citons le filtre adapté [2], le filtre de phase pure (POF "Phase only Filter") [8,10], le filtre composite [11,12] et le filtre segmenté [13].

1.2.2.3 Corrélateur à transformée de Fourier *con*jointe (JTC)

En 1966, Weaver et Goodman [6] proposent une alternative à l'architecture de VLC pour convoluer deux fonctions. Ils mettent en évidence la principale difficulté de mise en œuvre du corrélateur de VLC, à savoir l'alignement du filtre avec le spectre du signal d'entrée. Ils estiment cette précision à quelques microns. Leur solution consiste à placer les deux signaux à convoluer (ou à corréler) dans un même plan. Une transformée de Fourier appliquée à ce plan permet d'obtenir le spectre joint de ces deux images. Ensuite, une TF de l'intensité de ce plan permet d'obtenir la corrélation entre les deux signaux. L'avantage de cette méthode est qu'il n'y a pas de problème d'alignement, puisqu'il n'y a pas de filtre à introduire dans le plan de Fourier.
En revanche, il y a une étape d'enregistrement photographique à chaque opération, alors que, dans le système de Vander Lugt, le filtre est enregistré une seule fois. D'autre part, il est nécessaire d'utiliser un faisceau laser de référence pour pouvoir enregistrer le film dans la bonne gamme d'exposition.

Suite à cette publication, différentes architectures du corrélateur optique sont parues, plus connue sous le nom de corrélateur à transformée de Fourier conjointe ou JTC "Joint Transform Correlator". Une première réalisation optique a été rapportée dans [14]. Ce type de corrélateur est

largement étudié et utilisé dans la littérature notamment en raison de sa simplicité d'implantation [3].

Le principe de corrélateur JTC est donnée par la Figure 1.2.

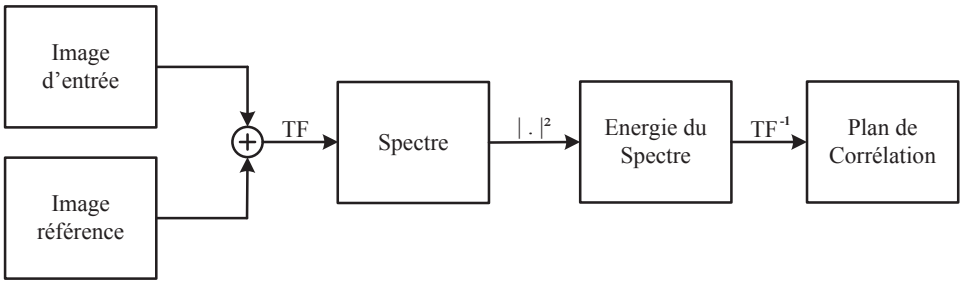


FIGURE 1.2 – Principe du corrélateur JTC

Considérons t le plan d'entrée du corrélateur JTC, ce plan s'écrit sous la forme :

$$t(i,j) = o(i+d,j) + r(i-d,j) \tag{1.2}$$

Où $o(i+d,j)$ représente l'image d'entrée et $r(i-d,j)$ représente l'image référence, d est une distance de décalage entrainant le décalage de l'image cible et de l'image référence dans le plan d'entrée. Le spectre de ce plan est donné par l'équation :

$$t(u,v) = |O(u,v)|e^{(\varphi_o(u,v))}e^{(jud)} + |R(u,v)|e^{(\varphi_r(u,v))}e^{(jud)} \tag{1.3}$$

Après avoir effectué le module au carré, le résultat obtenu s'écrit de la manière suivante :

$$|t(u,v)|^2 = |O(u,v)|^2 + |R(u,v)|^2 + 2|O(u,v)||R(u,v)|cos([\varphi_o(u,v) - \varphi_r(u,v) + 2ud]) \tag{1.4}$$

Dans cette expression et après la transformée de Fourier inverse, les deux termes $|O|^2$ et $|R|^2$ représentent l'auto-corrélation des deux images placées dans le plan d'entrée. Le reste de l'équation représente l'inter-corrélation entre les deux images. La Figure 1.3 représente deux différents plans d'entrées pour un corrélateur JTC. Les résultats des corrélations JTC de ces deux plans sont illustrés sur la Figure 1.4. En effet, la Figure 1.4(a) contient un pic au centre et deux pics symétriques qui indiquent une similitude entre l'image cible et celle de référence. Cependant, la Figure 1.4(b) contient un pic au centre et deux autres pics de faibles intensités ce qui montre qu'il y a très peu de ressemblance entre les deux images. La présence d'un faible pic de corrélation s'explique par la ressemblance dans le fond de l'image cible et le fond de l'image référence.

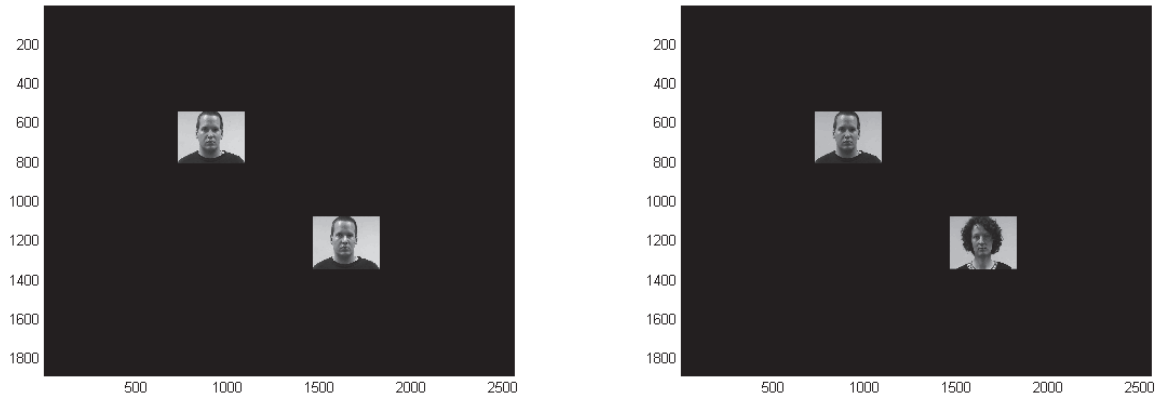
1.2.3 La Compression Optique

1.2.3.1 Principe de la compression

La compression est une opération qui consiste à réduire la quantité d'information utile à la représentation d'une donnée sans dégrader la qualité de ces informations (compression sans perte) ou pour représenter approximativement ces données (compression avec perte). L'utilisation de la compression est devenue un facteur économique pour l'archivage ou la transmission d'images.

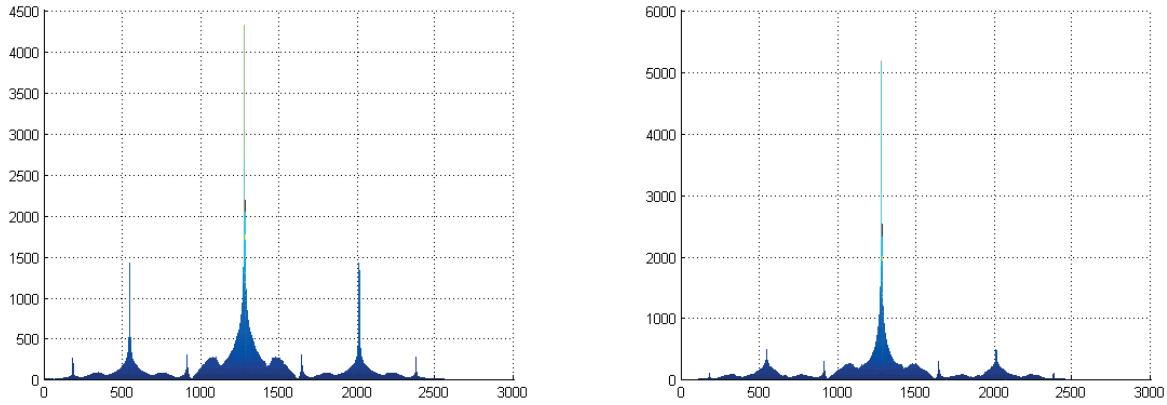
Cette opération est souvent réalisée, en aval, uniquement après une conversion de l'image analogique en une image numérique. Cependant, à l'origine l'image est optique, ce qui a amené une partie des optoélectroniciens à s'intéresser aux techniques de compression numérique les plus performantes et aux possibilités de les implanter par des moyens optiques.

Dans notre équipe vision nous avons développé une première approche implantant optiquement la compression JPEG suivie d'une autre approche pour implanter JPEG 2000 [1]. C'est pour cette raison que nous nous intéressons dans ce chapitre à ces implantations optiques.



(a) (b)

FIGURE 1.3 – Plans d'entrée



(a) (b)

FIGURE 1.4 – Plans de sorties résultants d'un corrélateur JTC

1.2.3.2 Compression optique JPEG

La JPEG est l'une des normes de compression la plus utilisée pour comprimer une image fixe. Le principe de l'algorithme JPEG pour une image à niveau de gris est présenté sur la Figure 1.5.

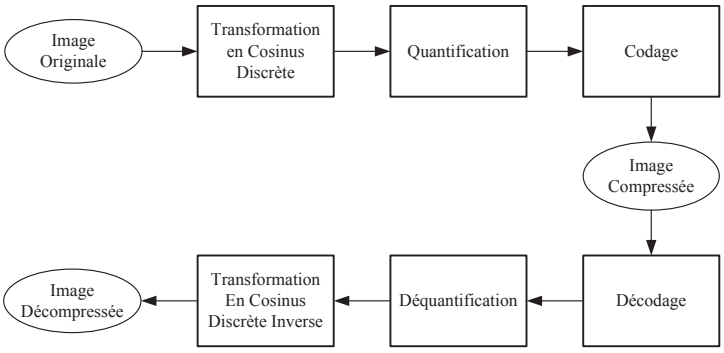


FIGURE 1.5 – Principe de la compression / décompression JPEG

La première étape de la compression JPEG consiste à réaliser une Transformation en Cosinus Discrète (DCT "Discrete Cosine Transform"). Par conséquent, l'implantation optique de l'archi-

tecture de compression JPEG doit passer d'abord par une implantation optique de la DCT. En effet, vue la simplicité avec la quelle les opticiens réalisent une TF en utilisant une simple lentille convergente, et les similitudes entre la TF et la DCT, l'implantation optique de la DCT peut être réalisée en se basant sur les propriétés de la TF optique.

Après avoir montré la possibilité d'une implantation optique de la DCT en utilisant une TF, la réalisation optique de l'algorithme JPEG est désormais possible (sauf pour l'étape de codage entropique). En effet, le synoptique de l'implantation optique de la compression JPEG proposé par [1] est présenté dans la Figure 1.6.

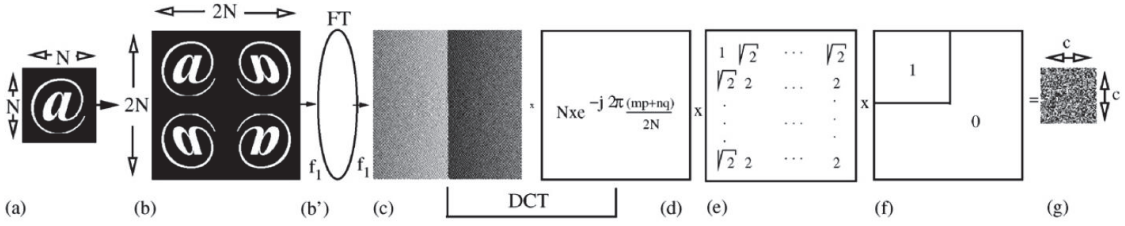


FIGURE 1.6 – Synoptique de la compression optique JPEG

La première étape de cette architecture consiste à dupliquer l'image d'entrée composée de $(N \times N)$ pixels d'une manière spécifique afin d'obtenir une image $\widetilde{I}(2N \times 2N)$ comme cela est montré sur la Figure 1.6 (b). La phase suivante consiste à réaliser la TF de l'image $\tilde{I}(2N \times 2N)$ en utilisant une lentille convergente L_1 (Figure 1.6(c)). Par la suite, le spectre obtenu sera multiplié par plusieurs hologrammes de compression. Quatrièmement, et afin de normaliser les coefficients, le résultat obtenu sera multiplié par un autre hologramme d'amplitude (Figure 1.6(f)). Le résultat de cette multiplication représente exactement la DCT de l'image répliquée (Figure 1.6(b)). Finalement, le résultat obtenu est multiplié par un filtre passe-bas (masque Figure 1.6(e)) pour sélectionner les informations pertinentes et ainsi filtrer les informations redondantes. Cela nous permet de réaliser optiquement les deux étapes les plus importantes dans la méthode JPEG à savoir : la DCT et le filtrage optique de l'image d'entrée (Figure 1.6(g)). Cette dernière, de taille $(c \times c)$ pixels, contient les données à transmettre dans le canal de transmission ou à stocker sur un support de stockage. Ainsi en partant d'une image d'entrée $(N \times N)$ pixels, nous obtenons un spectre $(c \times c)$ pixels réels.

1.2.3.3 Décompression optique JPEG

Du côté du récepteur, les données transmises ($(c \times c)$ pixels) doivent être décompressées optiquement pour reconstituer l'image comprimée. Cette décompression consiste à faire dans l'ordre inverse les différentes étapes réalisées lors la compression. Le diagramme illustré sur la Figure 1.7 et proposé par [1].

La première étape consiste à reconstituer le spectre compressé sous la forme présentée (Figure 1.7(b)), c'est-à-dire, une phase de remplissage par des zéros. Ensuite, le spectre reconstruit sera multiplié point par point par les hologrammes inverses de l'étape de compression (Figure 1.7(c) et Figure 1.7(d)). Le résultat de cette multiplication donne le spectre de l'image compressée $(I(2N \times 2N))$ pixels mais filtré (Figure 1.7(e)). Finalement une transformée de Fourier inverse optique, obtenue en utilisant une simple lentille convergente "L_2" réalise l'image dupliquée "$\widetilde{I'}$" comme présenté dans la Figure 1.7(f). Un simple masque est utilisé pour obtenir un des quatre quadrants donnant l'image originale décomprimée (Figure 1.7(c)).

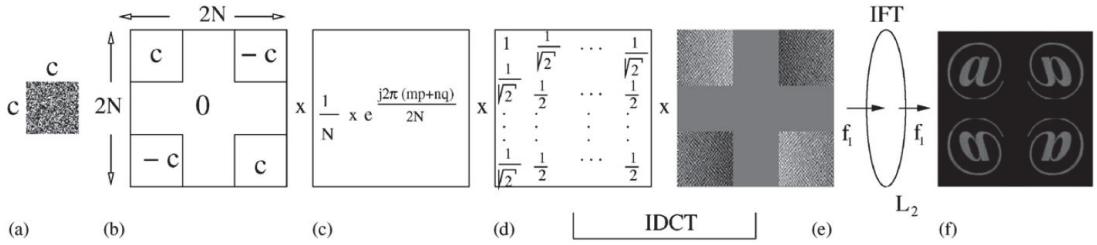


FIGURE 1.7 – Synoptique de la décompression optique JPEG

1.2.3.4 Compression JPEG2000

Bien qu'il soit très utilisé dans le domaine de la compression des images fixes, le standard JPEG présente plusieurs limites essentiellement causées par la Transformation en Cosinus Discrète. Ces limites peuvent être résumées en :

- une mauvaise qualité visuelle à fort taux de compression (effet mosaïque).
- une mauvaise compression des images avec du texte car le standard compresse mal les hautes fréquences spatiales, c'est à dire les contours.

Pour cela il était nécessaire de trouver d'autres algorithmes de compression basés sur une transformation plus appropriée. Parmi les diverses possibilités, la transformée en ondelettes attire, en particulier, l'attention de la communauté scientifique. En effet, la transformation en ondelettes a trouvé beaucoup d'applications pratiques dans le domaine de traitement de signal et du traitement d'image. Par conséquent, en se basant sur les avantages de la transformée en ondelettes et pour améliorer la compression d'images, le JTC (Joint Technical Committee) a fait appel à contribution en mars 1992, pour un nouveau standard, le JPEG2000 est né afin de corriger les problèmes du JPEG. En décembre 2000, le JPEG2000 devient un standard international. Ce standard présente de nombreux intérêts [15,16] :

- Il corrige les problèmes initiaux du JPEG.
- Il offre des performances à fort taux de compression supérieures aux standards existants.
- Il permet un affichage progressif, c'est à dire que l'on peut afficher une image soit en augmentant la qualité visuelle progressivement, soit en augmentant la définition de l'image progressivement.

Cependant, dans certaines applications, l'image dont on dispose est sous forme optique et il est souhaitable de la compresser directement sous cette forme. C'est pour cette raison que l'équipe vision a proposé une implantation optique de la norme JPEG2000 [1]. La Figure 1.8 illustre le principe de la compression JPEG2000.

L'étape centrale du processus de compression JPEG2000 est la transformée en ondelettes de l'image que l'on veut compresser. C'est la raison pour la quelle une implantation optique de cette transformée est cruciale. En effet, le rôle de la transformation en ondelette est de concentrer l'énergie sur un ensemble de coefficients aussi restreint que possible, afin d'avoir une meilleure efficacité de compression par rapport à un codage direct de l'image. C'est lors de l'étape de quantification, ayant pour rôle de remplacer les valeurs réelles de coefficients par des valeurs discrètes que les principales pertes de codage sont introduites.

Après avoir proposé une architecture pour implanter optiquement la DCT, une implantation optique de la norme JPEG 2000 a été proposée et validée par [1]. Cette architecture est présentée sur la Figure 1.9.

Le plan d'entrée, éclairé par une onde plane issue d'un laser contient l'image à compresser de taille $(N \times N)$. Cette image d'entrée est multipliée par un hologramme (élément diffractif périodique), ce qui permet de répliquer le spectre de l'image d'entrée 4 fois comme indiqué

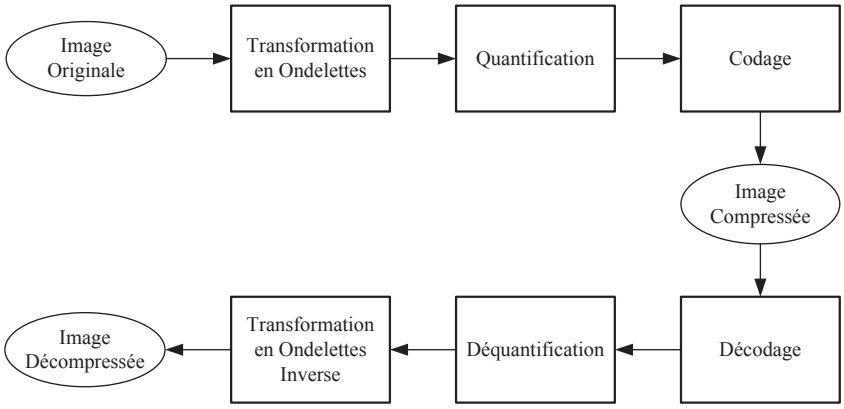


FIGURE 1.8 – Principe de la compression / décompression JPEG2000

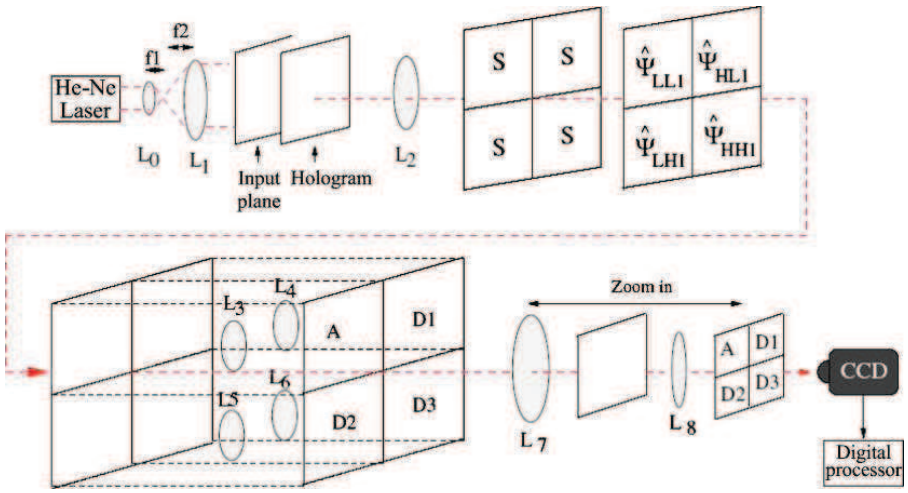


FIGURE 1.9 – Synoptique de la compression JPEG2000

sur la Figure 1.9. Par la suite, une TF est appliquée via une lentille L_2. Après avoir dupliqué le spectre de l'image d'entrée, les 4 parties obtenues sont multipliées point par point par un deuxième hologramme de phase implémentant les filtres $(\hat{\Psi}_{LL}, \hat{\Psi}_{LH})$, $\hat{\Psi}_{HL}$, $\hat{\Psi}_{HH}$, ou $\hat{\Psi}$ *désigne la transformée de Fourier de la transformée en ondelettes de Haar*. Le résultat de cette multiplication donne les spectres des images d'approximation et de détails. Enfin, les 4 lentilles L_3, L_4, L_5 et L_6 réalisent séparément et en parallèle la TF inverse de chacun de ces spectres filtrés. Un sous échantillonnage optique du résultat est réalisé via deux lentilles convergentes L_7 et L_8 et permet de revenir à la taille de l'image originale. Une caméra CCD est utilisée pour numériser l'image comprimée afin de continuer numériquement les étapes restantes du codage JPEG2000.

1.2.3.5 Décompression JPEG2000

A la réception, les données transmises devraient être décompressées pour reconstruire l'image source. La décompression JPEG2000, consiste à faire les opérations inverses de chaque étape

effectuée de côté de la compression. Le montage de l'implantation optique de la méthode de décompression JPEG2000 optique est présenté sur la Figure 1.10.

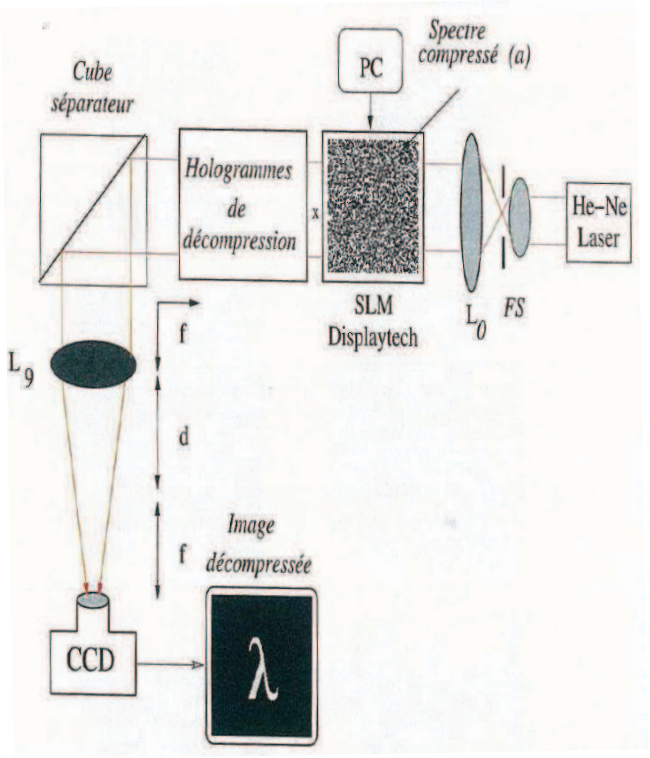


FIGURE 1.10 – Synoptique de la décompression JPEG2000 [1]

Le plan d'entrée présente l'hologramme contenant les 4 spectres comprimés de l'image en utilisant la technique JPEG2000. Cet hologramme est multiplié par un hologramme de déquantification ayant une amplitude uniforme. Finalement une lentille L_9 est utilisée pour implanter la TF. Le plan de sortie est affiché sur une caméra CCD permettant d'observer l'image décompressée.

1.2.3.6 Résultats

Les architectures de compression optique présentées précédemment ont été validées sur plusieurs types d'images. La Figure 1.11 présente les résultats obtenus pour la compression / décompression en implantant la norme JPEG optiquement. Ces résultats montrent que les architectures proposées permettent de réaliser une compression suivie par une décompression et ainsi reconstruire l'image comprimée. Cependant, une dégradation importante de l'image reconstruite est visible. D'autres tests ont été effectués avec plusieurs taux de compression C_r afin de vérifier la robustesse de l'architecture. En effet, le taux de compression est un critère utilisé pour évaluer les algorithmes de compression. Il consiste à mesurer la quantité d'information dans une image avant et après la compression. Ici, nous l'avons calculé avant l'étape du codage, il est ainsi calculé en utilisant l'équation suivante :

$$C_r = \frac{Taille\ de\ l'image\ d'origine}{Taille\ de\ l'image\ comprimee} \tag{1.5}$$

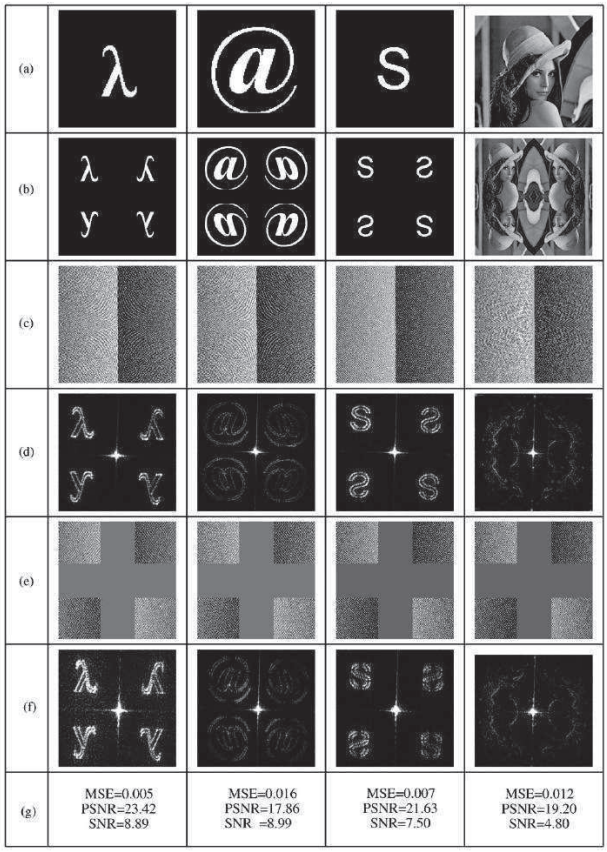


FIGURE 1.11 – Résultats de l'implantation optique de la compression decompression JPEG

Les résultats de ces tests sont illustrés sur la Figure 1.12. Ces résultats montrent que quand C_r augmente, la quantité d'informations est plus petite et par conséquent, l'image reconstruite est de plus mauvaise qualité.

1.2.4 Conclusion

Les algorithmes présentés dans cette partie peuvent être des solutions efficaces dans plusieurs domaines notamment la reconnaissance de visages (application liée à la sécurité) et la compression (applications de télécommunications). Cependant, les problèmes rencontrés lors de l'implantation optique restent des soucis majeurs pour que ces algorithmes soient bien exploités. En effet, l'espace encombrant d'une telle implantation rend l'embarquement du système très difficile. En outre, les résultats obtenus après une compression/décompression optique ont montré une dégradation importante de la qualité des images reconstruites. Aussi, la réalisation toute optique cause incontestablement des pertes des performances limitant la diffusion de ces méthodes à la base optique.

D'autre part, nous assistons ces dernières années, à une révolution technologique qui a permis à l'électronique numérique d'enrichir notre quotidien avec une utilisation et une utilité de plus en plus grande dans plusieurs domaines comme la santé, la sécurité, les communications, l'éducation

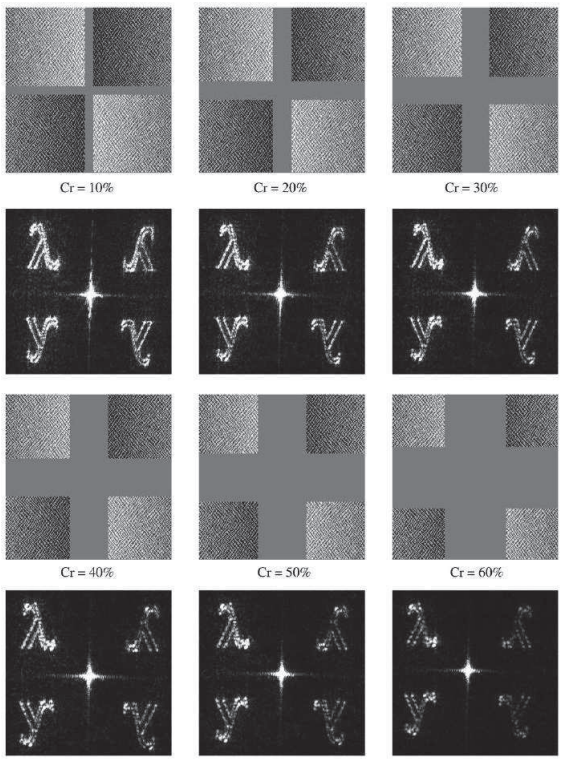


FIGURE 1.12 – Performances de l'implantation optique de la décompression JPEG

et les loisirs. Fort de ce constat, nous optons pour une implantation numérique des algorithmes de compression et de reconnaissances.

1.3 Evolution de l'Electronique Numérique

1.3.1 Introduction

La communauté du traitement du signal et des images s'est toujours intéressée à l'implantation des algorithmes qu'elle concevait. Cependant, ces algorithmes sont généralement connus par une complexité très importante ce qui ne facilite par leur implantation. D'autre part, ces dernières années nous assistons à un envahissement de notre vie quotidienne et professionnelle par des systèmes électroniques rapides et puissants. Ces systèmes connaissent une évolution importante due, entre autres, aux progrès dans l'industrie des semi-conducteurs. Cette évolution aussi constante que prévisible a permis de changer les méthodes et les outils d'implantation d'algorithmes de traitement du signal sur des circuits électroniques.

1.3.2 Evolution de l'industrie des semi-conducteurs

En 1965, Gordon Moore, cofondateur d'Intel, avait prédit que les capacités d'intégration sur les technologies silicium augmenteraient de manière exponentielle [17]. Il a introduit une loi qui porte son nom (loi de Moore) avec laquelle il prédit que le nombre de transistors double tous les 18 mois. De plus, étant donné que les délais de propagation et la puissance consommée sont liés à la densité d'intégration (taille de grille pour les transistors MOS par unité de surface), la loi de Moore peut être étendue pour suivre la rapidité, la consommation voire même le prix. Par exemple, en 1954 (5 années avant l'invention du circuit intégré) le prix du transistor coûtait 5.52 $. En 2004, ce prix a atteint 10^{-12} $ [18]. Néanmoins, cette évolution doit respecter les limites pratiques (par exemple la taille d'une mémoire ne doit pas dépasser 145 mm^2 et celle

d'un processeur ne doit pas dépasser 310 mm^2) [18]. Suite à cette évolution, la réduction de la taille des transistors est devenue un facteur important dans l'évolution de la technologie des semi-conducteurs pour plus de 40 ans. En effet, grâce à la diminution du coût des fonctions élémentaires (coût du bit par mémoire ou coût du MIPS " Million Instructions per Second " par processeur), l'industrie du semi-conducteur a atteint la barre de 250 billion $ du chiffre d'affaires ce qui représente 16% du total de l'économie mondiale. La Figure 1.13 résume le cycle de d'évolution de l'industrie des semi-conducteurs [19].

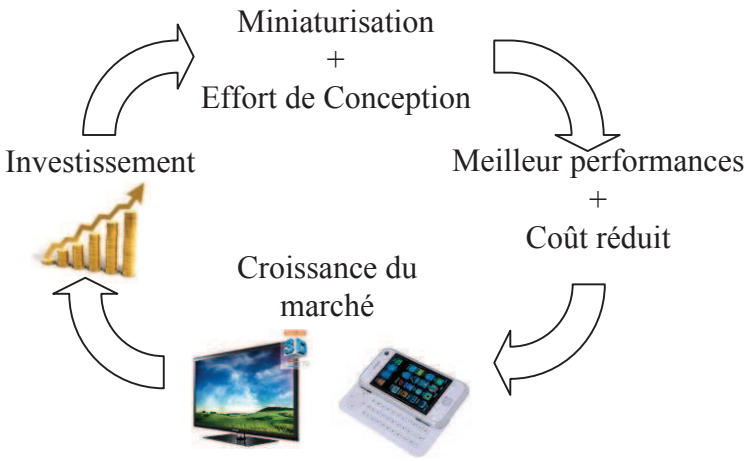


FIGURE 1.13 – Cycle de l'industrie des semi-conducteurs [19]

Dans la Figure 1.13, il est montré que la réduction des tailles des transistors combinée avec des effort de conception permettent l'amélioration des performances des circuits ainsi que la réduction des coûts de fabrication de ces circuits. La réduction des coûts de fabrication des circuits électroniques engendrent ainsi une croissance dans le marché des produits électronique avec l'apparition de nouveaux produits de surfaces réduites assurant de bonnes performances avec un coût réduit. Ce qui va amener les entrepreneurs à investir plus pour réduire d'avantage la taille des transistors.

Plus concrètement, ce cercle vicieux s'est matérialisé à travers l'apparition de 2 voies appelée "More Moore" et "More than Moore". La Figure 1.14 souligne cette évolution.

La première voie est appelée "More Moore" consiste à la miniaturisation des transistors basés sur la technologie CMOS (Complementary Metal Oxide Semiconductor) en particulier la réduction de la largeur de la grille qui a pu atteindre les 22 nm en 2012. Cette miniaturisation permet principalement d'augmenter les performances des circuits.

La deuxième voie est venue pour faire coexister des systèmes hétérogènes dans la même puce. A travers cette voie, la course à la miniaturisation n'est pas l'objectif principal. En effet, il s'agit ici de diversifier les fonctions dans un même système en utilisant des composants passifs, des MEMS "Micro-Electro-Mechanical Systems" et des circuits radiofréquences. Ceci a rendu possible l'intégration de plusieurs composants sur une seule puce et a participé à la naissance des SIP "System in Package".

D'autre part, la complexité croissante des systèmes sur puce rend leur conception de plus en plus difficile. Ceci provoque un écart de productivité (design productivity gap). En effet, dans [20] il a été annoncé que la productivité ne suit plus la complexité comme le montre la Figure 1.15. Cet écart de productivité représente l'écart existant entre le nombre de transistors que l'on peut mettre sur une puce et le nombre moyen de transistors d'un circuit qu'un ingénieur peut concevoir en une durée de travail bien définie.

En effet, l'ITRS [20] considère que l'écart de productivité est dû en grande partie aux méthodes de vérification actuelles, basées essentiellement sur la simulation, qui nécessitent un temps et des moyens considérables afin de pouvoir produire un système de qualité acceptable. Afin de

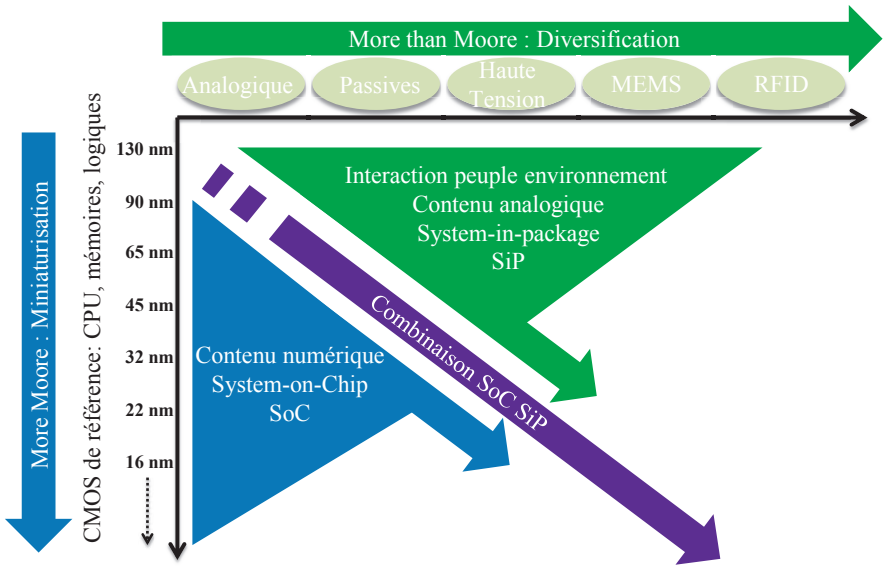


FIGURE 1.14 – More than moore ou la poursuite de la miniaturisation

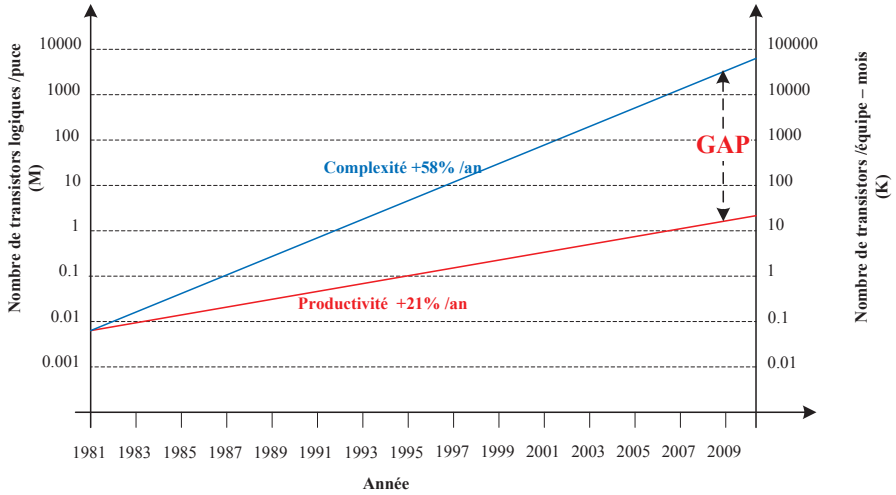


FIGURE 1.15 – Evolution de la productivité et de la complexité des circuits intégrés

réduire cet écart, plusieurs alternatives ont été proposées. Parmi ces alternatives, la technique de conception conjointe logiciel/ matériel ou Codesign et la réutilisation des composants virtuels IP "Intellectual Property" sont des solutions prometteuses.

1.3.3 Moyens d'augmentation de la productivité

1.3.3.1 Conception conjointe : Codesign

Dans une approche classique la conception des parties logiciels et celle des parties matériels sont séparées et ce n'est qu'à la fin du processus de conception que les différentes parties sont testées ensemble. Le mariage de la conception logicielle et matérielle, appelé codesign, permet

de maîtriser la conception des systèmes complexes afin d'augmenter la productivité. Souvent, le codesign logiciel/matériel revient à bénéficier de la souplesse d'une implantation logicielle tout en utilisant le matériel comme accélérateur. Le processus de codesign inclut plusieurs étapes de conception. Partant d'une spécification au niveau système, l'architecture sera raffinée jusqu'à la définition de l'architecture logicielle/matérielle de réalisation [21,22]. Les étapes de conception sont :

- La spécification et la décomposition fonctionnelle : l'exploration concerne la recherche d'une conception, qui satisfait aux besoins fonctionnels, mais aussi à des objectifs de performances. Souvent les performances considérées à ce niveau sont plutôt liées à des critères fonctionnels.
- Le partitionnement logiciel/matériel : Cette étape consiste à rechercher pour chaque fonctionnalité une réalisation soit logicielle soit matérielle. L'objectif du partitionnement est de trouver le meilleur compromis entre les parties logicielles et matérielles en fonction de leurs interactions et des contraintes dictées notamment par des critères de performances et de réutilisation.
- La co-synthèse : Il s'agit d'assurer la synthèse des partitions logicielles et matérielles ainsi que la communication.
- Le prototypage virtuel et physique : au cours de cette phase, les solutions résultant des étapes de partitionnement et de co-synthèse sont validées. Il s'agit de vérifier le bon fonctionnement du système et la satisfaction des contraintes liées au contexte de l'application du système.

1.3.3.2 Réutilisation des IP

Cette solution a été adoptée par les concepteurs afin de faire face à la complexité des applications pour des systèmes sur puce et d'assurer leurs flexibilité. En effet, les IP "Intellectual Property" sont conçues sous forme de bibliothèque dont le concepteur dispose et puisse les intégrer dans son design sans recours à les concevoir. Par conséquent la description des circuits complexes sera simplifiée en assemblant un certain nombre d'IP entre eux. Ainsi, le temps de conception sera consacré à concevoir des parties innovantes en s'appuyant sur le principe de la réutilisation des IP.

Les IP sont classées en fonction de leur flexibilité en trois groupes [23,24] :

- les Soft-cores : ces blocs sont connus par leurs flexibilité. Ils sont des blocs décrits dans un langage de description matérielle synthétisable. Ils sont indépendants de toute technologie et sont donc très souples d'utilisation.
- les Hard-cores : sont optimisés et placés-routés pour une technologie donnée. Le bloc est donc implicitement caractérisé en termes de performances et de surface. L'inconvénient est qu'ils sont moins flexibles et portables car le processus est dépendant de la technologie (plate-forme).
- les Firm-cores : sont des blocs ayant subis une optimisation en surface et en vitesse par des techniques de placement relatif. Décrits en HDL structurel, ils font appel à des composants élémentaires d'une librairie générique. Cette implantation autorise des évaluations plus fines de ressources utilisées et de performances. Le Firm-Core n'est pas routé. Ce type de macro bloc représente donc un compromis entre les IP qui sont complètement logiciels Soft-cores et les IP complètement matériels Hard-cores.

1.3.4 Cibles d'implantation

1.3.4.1 Exploration d'architectures

Les cibles d'implantation peuvent être classées en fonction de leur flexibilité. La première famille rassemble les circuits à architectures généralistes. La deuxième famille est basée sur les architectures complètement dédiées. La troisième est une solution intermédiaire entre les deux premières familles et consiste à utiliser des circuits programmables.

Nous évaluerons dans cette section les avantages et inconvénients de ces cibles en insistant sur les performances des systèmes en termes de coût et de la facilité de développement des applications pour chacune d'entre elles.

1. **Les circuits à architectures généralistes**
   Ces architectures ont été conçues pour une exécution efficace des programmes ou logiciels. La principale caractéristique de ces architectures est leur niveau de flexibilité très élevé. Cette grande flexibilité vient du fait que ces architectures utilisent les processeurs généralistes (CPU "Central Processing Unit", GPP "General Purpose Processor"). Par ailleurs, la capacité de calcul de ces processeurs ne cesse de s'accroître à travers l'augmentation de la fréquence d'horloge et l'exploitation du parallélisme. Cependant, des limites physiques ont contribué à limiter l'accroissement de la fréquence de fonctionnement de ces circuits. Pour palier à ces problèmes, des architectures utilisant plusieurs cœurs ont permis d'améliorer les performances des processeurs. D'autres processeurs moins généralistes dédiés à des applications spécifiques tels que les jeux vidéo ou le calcul scientifique offrent une puissance de calcul plus importante. Ces processeurs sont caractérisés par l'intégration d'un grand nombre d'unités de traitement. Parmi ces architectures nous citons les processeurs Cell conçus conjointement par IBM, Sony et Toshiba et révélé en février 2005. Ils équipent notamment la console de jeu vidéo PlayStation 3 [25]. Plus récemment sont parus d'autres processeurs plus performants tels que les processeurs graphiques GPU " Graphical Processing Unit " et les GPGPU " General Purpose GPU " [26].
2. **Les circuits à architectures dédiées**
   Contrairement aux processeurs à architectures généralistes, les circuits à architectures dédiées consistent à concevoir des architectures spécifiques pour des applications bien identifiées en utilisant les circuits intégrés ASIC. Ces circuits permettent de faire une implantation matérielle des applications par opposition à l'implantation logicielle que les processeurs généralistes réalisent. En outre, étant donné que les ASIC sont conçus pour réaliser une seule fonction, les traitements à implanter dans ces circuits sont les plus optimisés en termes de rapidité et consommation de puissance. De même, la surface utilisée dans l'ASIC est la plus réduite.
   Malgré leurs avantages en termes de rapidité, taille réduite et faible consommation, les ASIC demeurent toujours une solution difficile à réaliser. En effet, la conception et la fabrication des circuits ASIC nécessitent différentes phases de développement à un coût très important. De plus, les longues phases de conception augmentent considérablement le temps de mise sur le marché de ces produits, ce qui est parfois incompatible avec les exigences du marché. Enfin, un autre inconvénient majeur de ces circuits vient de leur non reconfigurabilité. En effet, le degré de flexibilité des ASIC est très faible puisqu'ils ne sont adaptés qu'à une seule application. Ceci pose le problème au niveau de productivité comme nous l'avons précisé dans la section 1.3.2 (Figure 1.15).
   En définitif, l'utilisation de cette architecture peut constituer une solution intéressante pour produire des circuits dédiés à des applications bien précises, nottament quand la quantité à produire est importante pour amortir le coût de conception.
3. **Les circuits à architectures programmables**
   Une solution intermédiaire entre les implantations logicielles avec les processeurs généralistes et les circuits entièrement dédiés consiste à utiliser des circuits programmables tels que les FPGA "Field Programmable Gate Array". Ces derniers sont des architectures génériques se composant d'un réseau de portes logiques, de blocs mémoire et d'un réseau d'interconnexion. L'architecture de base d'un FPGA est donnée par la Figure 1.16.
   Par ailleurs, les derniers FPGA intègrent des fonctions précablées comme des multiplieurs et des processeurs implantés au sein du réseau de portes logiques. Ces circuits permettent donc un partitionnement adéquat entre les aspects matériels et logiciels, tout en offrant une flexibilité et une reconfigurabilité très importante [27]. En effet, les FPGA permettent la conception matérielle des applications sans passer par les longues phases coûteuses de développement comme dans le cas des ASIC. Les calculs sont effectués de manière câblés

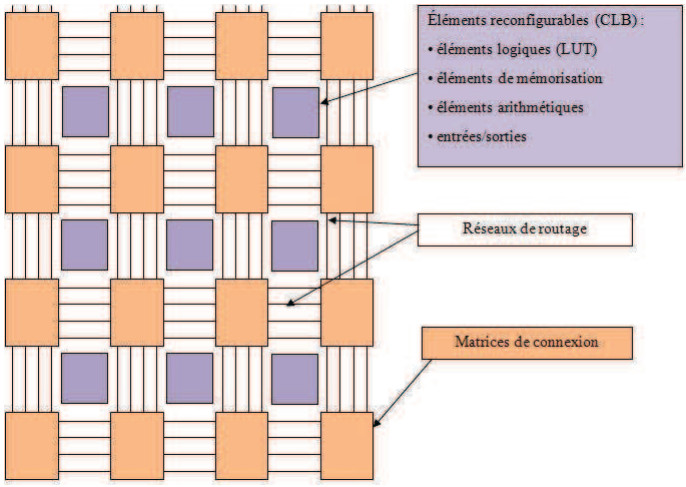


FIGURE 1.16 – Architecture de base d'un FPGA

et spécialisés, des gains importants en performance peuvent être obtenus par rapport à une exécution sur processeur générique. De plus ces architectures peuvent être utilisées temporairement pour réaliser une application donnée et reconfigurées par la suite pour effectuer une autre application. Cette reconfiguration peut être à la fin ou en cours de l'exécution de l'application et peut être totale ou partielle, c'est-à-dire modifier le fonctionnement de l'application au cours de l'exécution.

Malheureusement, l'utilisation des FPGA présente l'inconvénient majeur de la surconsommation d'énergie. De plus les performances en termes de rapidité des circuits FPGA sont moins importantes que celle des circuits ASIC.

1.3.4.2 Compromis performance/flexibilité/consommation

Dans la Figure 1.17, les différents cibles programmables sont classées par rapport à la rapidité d'exécution et à la flexibilité pour une technologie équivalente. Bien que les cibles spécifiques (ASIC) offrent de meilleures performances pour une application donnée, elles restent des solutions dédiées et limitées en flexibilité. Les GPP sont les cibles les plus flexibles, cependant, leurs performances en termes de rapidité sont très faibles en comparaison aux ASIC. Les circuits à architectures reconfigurables représentent ainsi le meilleur compromis en termes de performance/flexibilité. Ces architectures représentent aussi le meilleur compromis en termes de coût de développement et de consommation. En effet, comme il est montré dans la Figure 1.17, le coût de développement est important quand l'architecture n'est pas flexible. Cependant, la reconfigurabilité et par conséquent la flexibilité des circuits est coûteuse en terme de puissance consommée [28].

Dans ce travail, nous nous intéressons à deux architectures programmables qui sont le GPU et le FPGA. Dans les sections suivantes nous détaillons les architectures internes et le principe de fonctionnement de ces deux cibles.

1.3.5 Les GPU

1.3.5.1 Motivations

A partir des années 2000, l'évolution de la fréquence des processeurs CPU ne suit plus la loi de Moore. Cela est dû à l'importante consommation d'énergie [29]. De ce fait, plusieurs chercheurs ont tenté de trouver une solution permettant d'augmenter la fréquence de calcul des CPU tout en tenant compte de leur consommation d'énergie. Une des solutions consiste à utiliser des architectures multi-cœurs pour augmenter la fréquence de fonctionnement des circuits.

Parmi ces modèles multi-cœurs, nous pouvons citer les GPU, processeurs de la carte graphique. Ces processeurs, initialement conçus pour le rendu graphique, commencent à trouver

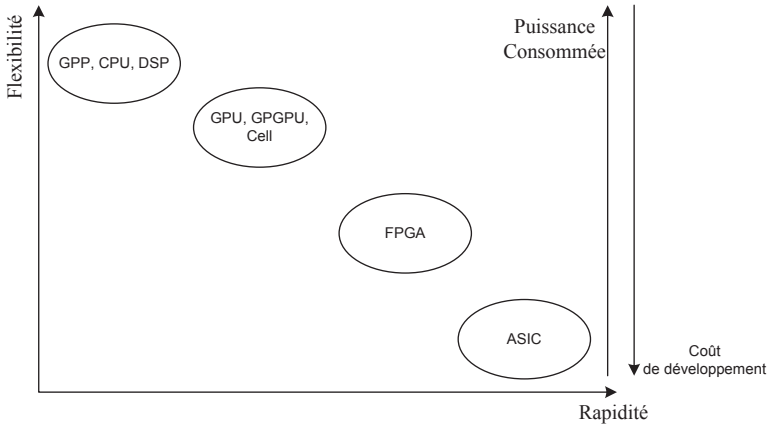


FIGURE 1.17 – Compromis Performance Flexibilité des SoC

leur place dans les applications scientifiques depuis 2003 [30]. Les processeurs GPU d'aujourd'hui peuvent être vus comme des processeurs parallèles dotés d'une couche logicielle qui permet de les programmer et d'exploiter toute leur puissance de calcul [31]. C'est à partir de ce moment que la puissance de calcul des GPU devance celle des CPU.

Une étude comparative sur l'évolution de performance des GPU de la famille NVIDIA(le premier fabricant des GPU) et celles des CPU a été réalisée en 2008 [30]. Nous avons étendu cette étude jusu'à 2012. Nous avons mentionné le maximum de performance des cibles par famille en terme de rapidité. Le résultat de cette étude est illustré dans la Figure 1.18. Il est montré que le rapport entre le maximum des performances en termes de rapidité des GPU et celui des CPU continue à augmenter chaque année. En 2011, ce rapport à atteint la valeur de 15.

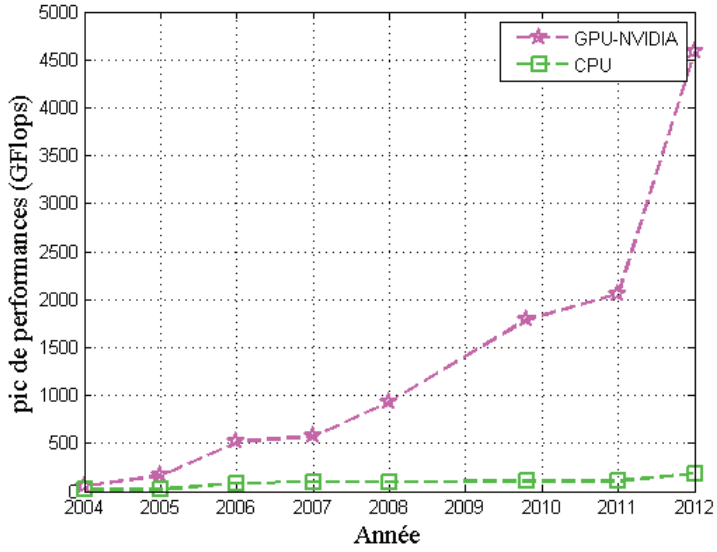


FIGURE 1.18 – Comparaison performances CPU-GPU [32, 33]

1.3.5.2 Historique

A l'origine, les GPU ont été exploités pour réaliser un ensemble de traitement permettant de projeter une scène 3D composée de polygones sur une grille 2D de pixels correspondant à

l'écran. Ces traitements sont regroupés sous l'appellation de " pipeline graphique ". Le principe de fonctionnement de ce pipeline consiste à prendre en entrée une représentation 3D, effectuer des traitements et afficher en sortie une image plane 2D. Les trois étages principaux du pipeline graphique sont :

- La transformation des sommets du polygone positionné dans l'espace 3D vers le plan de l'écran 2D à l'aide d'une matrice de projection.
- Le polygone projeté sur l'écran est transformé en une série de pixels ou fragments.
- Le traitement de chaque pixel de l'image.

Ce pipeline graphique a permis une évolution rapide des performances jusqu'à la fin des années 90. Par contre sa conception figée ne laissait pas de place pour la programmation de nouveaux effets non-pré-câblés .

Vers le début des années 2000, les unités de traitement de sommets et de fragments sont devenues programmables ce qui a permis à ce pipeline d'être ouvert à l'exécution de code utilisateur. Ces unités sont remplacées par des processeurs de sommets et de fragments (Vertex Shader unit et Pixels Shader unit). Cette évolution a provoqué la naissance de langages spécifiques de bas niveau (assembleur) et de haut niveau (OpenGL Shading Language, le langage d'NVIDIA Cg) [34].

En 2006, et dans le but d'augmenter la puissance de calcul, des unités de calcul parallèle génériques ont été développées afin de remplacer simultanément les processeurs de sommets et de fragments. Cette architecture permet de bénéficier de toute la puissance de calcul parallèle des GPU en toute circonstance ce qui a permis l'utilisation de ces processeurs graphiques à des fins de calculs génériques. On parle ainsi du GPGPU pour General-Purpose GPU.

Il est à signaler que l'évolution de l'architecture interne des GPU a été suivie par une évolution au niveau des moyens de programmation. En 2007 le fabricant des GPU NVIDIA a proposé la plateforme CUDA (Compute Unified Device Architecture), afin d'exploiter les capacités de calcul des GPU en exécutant en parallèle un certain nombre de tâches d'une application appelées "threads".

1.3.5.3 Architecture et principe de fonctionnement

Les processeurs graphiques sont de nature vectorielle. Leur mode de fonctionnement est désigné en anglais sous le terme de "stream processing", ce que l'on pourrait traduire par " traitement de données par flux ". Ces systèmes sont caractérisés par de multiples sous-unités de calculs (modules) reliées entre elles (par des canaux) et sont destinés à traiter un volume de données important éventuellement en temps réel. Embarquant plusieurs microprocesseurs et une très importante bande passante mémoire, les GPU actuels offrent une incroyable ressource pour des calculs aussi bien graphiques que non graphiques.

Dans cette thèse, nous avons utilisé exclusivement des GPU fabriqués par le constructeur taïwanais NVIDIA, et ceci pour plusieurs raisons :

- NVIDIA est le premier constructeur à avoir fourni un véritable langage (avec compilateur et bibliothèques) dédié à son matériel.
- NVIDIA est le constructeur qui a le plus communiqué sur son architecture matérielle.
- Même si aujourd'hui d'autres fabricants commencent à exploiter la puissance de calcul des GPU pour les applications scientifiques, NVIDIA est encore le seul qui fournit de véritables unités de calcul ainsi que les outils logiciels dédiés.

Une carte graphique de type NVIDIA (GeForce, Quadro, ou Tesla) supportant CUDA est un ensemble de multiprocesseurs (un multiprocesseur étant un ensemble de processeurs graphiques). Chaque multiprocesseur possède 4 catégories de mémoire différentes. Par exemple, la carte GeForce 8800GTX compte 16 multiprocesseurs, chaque multiprocesseur contenant 16 processeurs graphiques ce qui fait au total 128 processeurs graphiques. Chaque multiprocesseur a une architecture SIMD (Single Instruction on Multiple Data) : à chaque cycle d'horloge, l'ensemble des processeurs traitent un processus élémentaire pour des données différentes. Ainsi, le temps d'exécution des taches sur un GPU en nombre de cycles d'horloge qui est exprimé par l'équation 1.6 :

$$Temps\ d'execution = \frac{Nombre\ de\ thread}{nombre\ de\ processeurs\ graphiques} \tag{1.6}$$

Les threads sont organisés sous forme de warp (32 thread par warp). Ces warp sont structurés dans des blocs avec un maximum de 512 threads par blocs (16 warp par bloc). Ces blocs sont eux même organisés dans une grille comme indiqué dans la Figure 1.19.

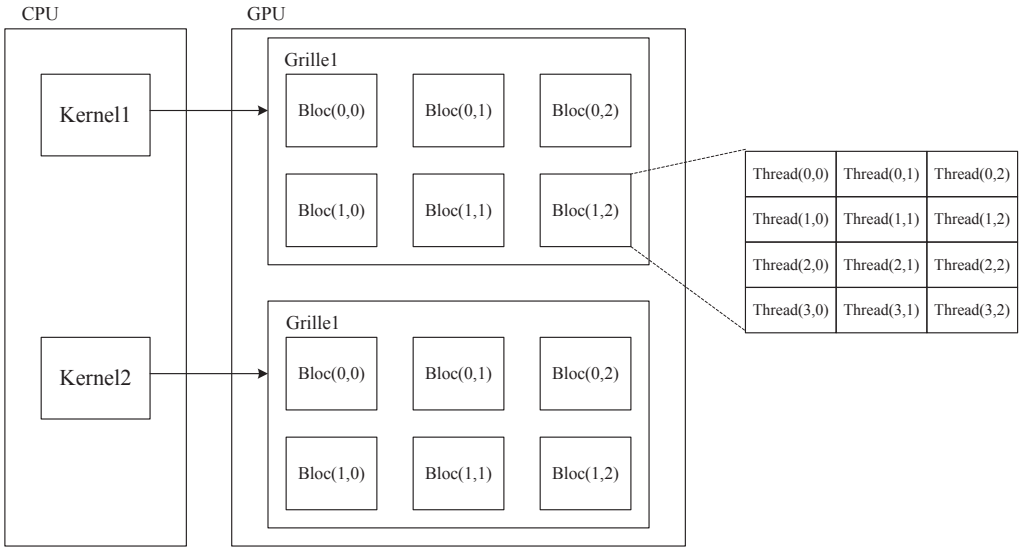


FIGURE 1.19 – Architecture CUDA

La grille est obligatoirement constituée par des blocs bidimensionnelles et a pour dimension maximale 65535×65535 blocs. Chaque élément de la grille est un bloc de thread pouvant être 1D, 2D, ou 3D. Chaque bloc est distingué par son identifiant (bloc ID). Pour une grille de taille (Gx, Gy), l'identifiant du bloc (bx, by) est égal à $bx + by \times Gx$.
Un bloc de thread a pour dimensions maximales $512 \times 512 \times 64$ threads tout en ayant pour contrainte au maximum 512 threads. De la même manière que les blocs dans une grille, chaque thread d'un bloc est identifié par son thread ID. Cet identifiant s'exprime par un ensemble de coordonnées 2D ou 3D selon la dimension du bloc employé. Par exemple, pour un bloc bidimensionnel de taille (Bx, By), l'identifiant du thread (tx, ty) est égal à tx + tyBx. Les threads d'un même bloc sont exécutés par un seul multiprocesseur et partagent les données via l'utilisation d'une mémoire particulière et extrêmement rapide appelée mémoire partagée [30].

1.3.6 FPGA

Un FPGA est un circuit électronique dont l'architecture correspond à une matrice de portes logiques séparées par des réseaux d'interconnexion. Ils existent deux types de FPGA : les non reprogrammables (technologie de type anti-fusible) et les reprogrammables (technologies de type SRAM ou Flash).

1.3.6.1 Evolution de l'architecture des FPGA

Les FPGA sont une évolution des circuits logiques programmables appelés PAL "Programmable Array Logic". Les PAL sont un assemblage de portes logiques offrant au concepteur la possibilité d'activer ou non les connexions entre ces portes. Les FPGA sont nés à base de ce principe.

L'architecture des FPGA correspond à une matrice de blocs logiques séparés par des réseaux d'interconnexion comme il est illustré dans la Figure 1.16. On distinguera les FPGA non reprogrammables (technologies de type antifusible, offrant les meilleures performances en termes de densité d'intégration et de vitesse), et les FPGA reprogammables (technologies de type SRAM par exemple). Ces derniers sont les plus utilisés grâce à leur flexibilité. Dans ce qui va suivre, nous nous intéressons à l'architecture interne des FPGA de la famille Xilinx ainsi que l'évolution de ses composants de base ces dernières années.

Comme il est montré dans la Figure 1.16, l'architecture de ces FPGA est composée d'une matrice de blocs logiques, d'un réseau d'interconnexions et d'un bloc d'entrée/sortie. Ces blocs sont appelés CLB "Configurable Logic Bloc" [35]. Chaque bloc CLB est composé d'un nombre de sous blocs, définis en fonction de type du FPGA, nommé Slices. Ces derniers contiennent un ensemble de tables de vérité (LUT "Look Up Tables") qui représentent les éléments de base du FPGA.

A partir des années 2000 et dans le but d'augmenter la productivité, l'architecture des FPGA a évolué pour intégrer plus de blocs tels que des mémoires BRAM, des blocs multiplieurs, des processeurs ainsi que des blocs IP et un nombre très important de cellules logiques. La Figure 1.20 illustre l'architecture interne d'un FPGA de la famille Xilinx Virtex-5. Nous avons choisi de montrer l'architecture de la carte FPGA Virtex-5 parcequ'elle sera utilisée dans la suite de la thèse.

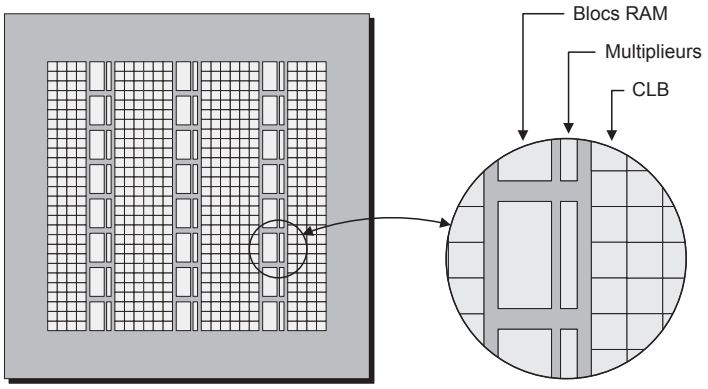


FIGURE 1.20 – Architecture interne d'un FPGA Virtex-5

1.3.6.2 Evolution de l'architecture des CLB

Les blocs logiques configurables (CLB) sont utilisés pour implanter les fonctions séquentielles et combinatoires. Tout élément du CLB est relié à une matrice de connexion lui permettant d'accéder aux réseaux de routage. La structure des CLB varient en fonction de la génération des FPGA. En effet, les CLB des FPGA antérieures aux FPGA virtex-4 sont composés de 4 Slices rassemblés en pair comme indiqué dans la Figure 1.21.

L'architecture des CLB des FPGA Virtex-5, Virtex-6 et Virtex-7 diffère légèrement de celle des FPGA Virtex-4. En effet, ils contiennent 2 Slices répertoriés en deux types : $Slice_M$ et $Slice_L$.

1.3.6.3 Evolution des Slices

Le changement de la structure des CLB a été accompagné par un changement de structure des Slices. En effet, les Slices des FPGA des familles antérieures à Virtex-4 contiennent deux LUT à 4 entrées et 1 sortie, des multiplexeurs et des registres. Pour les nouvelles générations des FPGA, le nombre de LUT par Slice a augmenté pour passer de 2 à 4. De plus, le nombre des entrées et des sorties par LUT a augmentée pour passer des LUT à 4 entrées et une sortie à des LUT à 5 entrées et 2 sorties ou bien des LUT à 6 entrées et 1 sortie. Le changement de

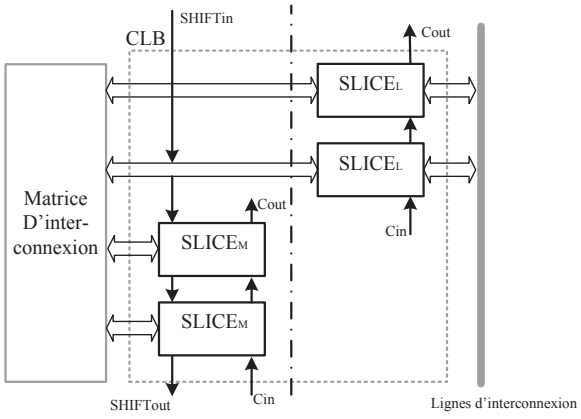


FIGURE 1.21 – Architecture interne d'un CLB d'un FPGA Virtex-4

l'architecture interne est venu suite à une étude effectuée dans [36] et qui a montré que le nombre optimal des entrées d'une LUT doit être compris entre 4 et 6 et que le nombre de LUT par Slice doit être compris entre 4 à 10 en fonction de l'application.

Il est à noter qu'il existe deux type de Slices, les $Slice_M$ et les $Slice_L$. La principale différence entre ces deux types réside dans le fait que les $Slice_M$ permettent de réaliser plus de fonctions que celles effectuées par les $Slice_L$. En effet, en plus des fonctions logiques réalisées par les deux types de Slices, les $Slice_M$ peuvent être configurés en tant que mémoires ou des registres à décalage.

Les circuits logiques qui exploitent la configuration des LUT à 5 entrées et 2 sorties, exigent que ces dernières doivent être synchrones. Ainsi ces sorties doivent passer ou pas par des registres intermédiaires. Dans le cas ou le passage par des registres est nécessaires, la seconde sortie sera sauvegardée dans un registre d'une autre LUT libre ce qui introduit des délais de routage. La solution de ce problème a été introduite dans les FPGA Virtex-6. En effet, un élément de sauvegarde supplémentaire a été ajouté dans les Slices de cet FPGA. Par conséquent, les deux sorties auront des registres de sauvegarde indépendants. En cas d'inutilisation d'un des deux registres, il sera accessible via le réseau d'interconnexion pour une utilisation efficace de la surface.

Il est a noter qu'entre les FPGA Virtex-6 et Virtex-7, il n'y a pas eu de changement dans l'architecture interne des CLB et des Slices. La seule différence entre ces deux famille réside dans le nombre de cellules logiques intégrées. En effet, une cellule logique est composée d'une LUT, d'un registre et d'un multiplexeur.

Parallèlement à l'évolution de l'architecture, le nombre de Slice par FPGA a aussi évolué. La Figure 1.22 (a) illustre cette évolution [35]. Il est montré que le nombre de Slices augmente d'une génération à l'autre à l'exception de la famille Virtex-5. En effet, les FPGA de la famille Virtex-5 contiennent moins de Slices à cause du changement de l'architecture interne des Slices détaillé précedemment. Afin de mettre l'accent sur cette évolution, nous représentons l'évolution en fonction du nombre de cellules logiques. La Figure 1.22 (b) illustre cette évolution [35].

1.3.6.4 Evolution de la fréquence

L'évolution de l'architecture interne des FPGA a aussi été suivie par une évolution dans la fréquence d'horloge. L'ensemble des données que nous avons collectés du site de Xilinx [35] montre que la fréquence d'horloge des FPGA ne cessent d'augmenter (Figure 1.23).

Malgré cette augmentation la fréquence d'horloge des FPGA reste faible comparée à celle des processeurs. En effet, cette fréquence ne dépasse pas quelques centaines de MHz par contre celle d'un processeur peut atteindre plusieurs GHz. Malgré que la fréquence d'horloge des FPGA reste faible par rapport à celle des processeurs, le temps de traitement d'une application implantée sur FPGA peut être plus court comparé à une implantation sur processeur. En effet, lors d'une

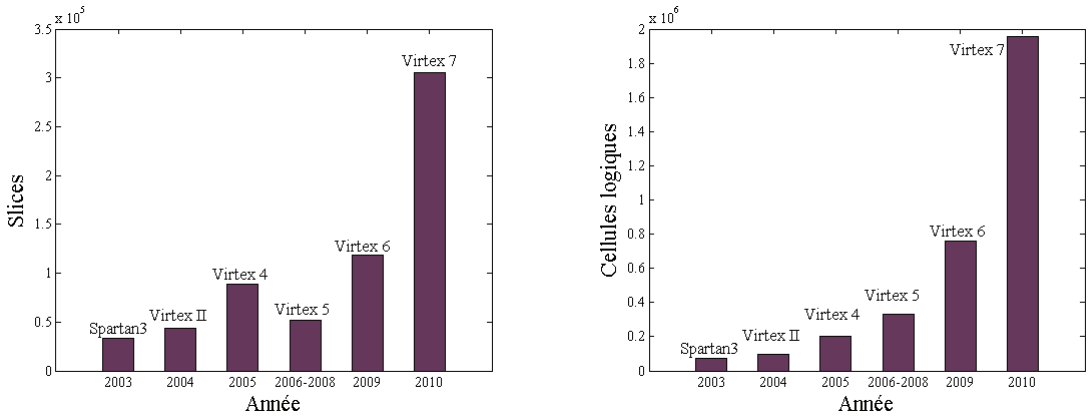


FIGURE 1.22 – Evolution des éléments logique des FPGA Xilinx (a) Nombre de Slices ; (b) Nombre de cellules logiques

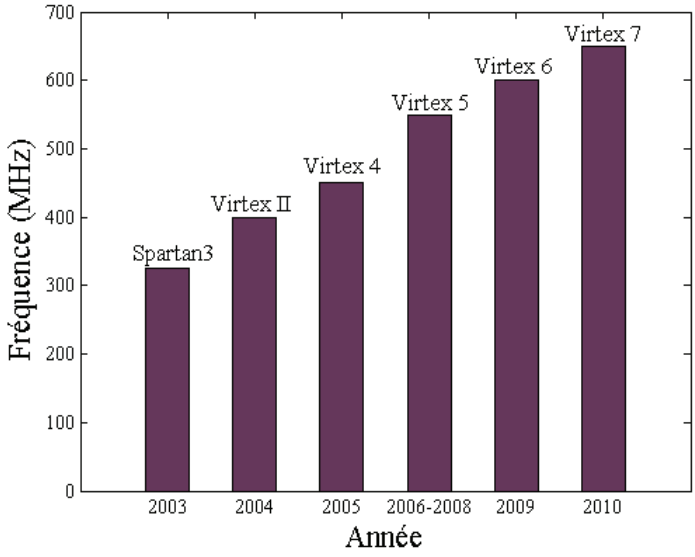


FIGURE 1.23 – Evolution de la fréquence d'horloge des FPGA de la famille Xilinx

implantation sur FPGA, un seul signal cycle d'horloge peut servir à exécuter plusieurs traitement.

1.3.6.5 Evolution des IP

Les FPGA de nouvelles générations contiennent des blocs permettant de réaliser des opérations arithmétiques bien définies au sein des réseaux de portes logiques tels que les DSP, les processeurs, les blocs mémoires, etc.

Parmi ces IP nous citons les DSP Blocs. Ces blocs étaient introduits par Xilinx dans les années 2000 dans les FPGA Virtex-II sous forme de multiplieurs embarqués 18 bits x18 bits. En 2003, Altera a proposé son premier DSP bloc dans les FPGA Stratix. Ces blocs ont été intégrés par la suite dans les FPGA Xilinx Virtex-4 et sont composés par un multiplieur 18×18, suivie par un additionneur/soustracteur 48 bits ou une unité accumulateur.

Les DSP48 proposés dans les FPGA Virtex-5/6 (DSP48E pour les FPGA Virtex-5 et DSP48E1 pour les FPGA Virtex-6) contiennent des multiplieurs de plus larges taille 18×25 bits. L'additionneur/accumulateur peut réaliser d'autres opérations telles que les opérations logiques ET, OU, XOR. De plus, les DSP48E1 contiennent des additionneurs qui précèdent la multiplication et qui peuvent être utiles dans plusieurs algorithmes de traitement de signal.

Dans les plus récents FPGA, nous trouvons aussi des blocs de mémorisation afin de permettre

la lecture et la sauvegarde des données. Ces blocs sont configurables, en fonction de l'application, en plusieurs façons RAM "Random Access Memory", ROM "Read Only Memory", FIFO "First in First Out" [28].

Les mémoires embarquées des FPGA Virtex-4 de la famille Xilinx ont une capacité maximale de 18 kbits. Chaque bloc mémoire BRAM (Block RAM) possède deux ports complètement indépendants partageant les données enregistrées. Le contenu des BRAM peut être configuré par un fichier de configuration. Cette option est utile dans le cas où nous avons besoin d'enregistrer des données d'initialisation.Dans les FPGA Virtex-5/6, la taille des blocs mémoires a augmenté pour atteindre une capacité de 36 kbits. Dans les FPGA Virtex-7, il y a eu une augmentation dans le nombre de blocs mémoires.

Nous pouvons trouver aussi dans les FPGA récents d'autres blocs IP implantés en logiciel ou en matériel ou en mixte. Parmi ces IP nous trouvons les processeurs matériels tels que Power PC de Xilinx ou Nios d'Altera, et des processeurs software tel que le processeur Microblaze proposé par Xilinx. Enfin, il est important de noter, que dans le cadre d'implantation de circuits et de systèmes numériques, nous avons la possibilité d'instancier des IP conçus pour réaliser des traitements spécifiques. Nous pouvons citer les algorithmes de Reed-Solomon et 3GPP Turbo Encoder pour les applications de télécommunications. Dans cette thèse nous allons nous référer à l'IP FFT et DCT pour des applications de traitement de signal et de traitement d'image.

1.3.7 Conclusion

Dans ce travail, nous allons opter pour l'utilisation de l'électronique pour l'implantation numérique des différents algorithmes et applications de traitements du signal et d'images. Ce choix a été effectué pour plusieurs raisons :

1. L'apparition de plusieurs architectures parallèles permettant d'avoir de très bonnes performances.
2. Les architectures numériques offrent une résolution nettement meilleure que celle fournie par l'optique ce qui permet d'avoir des meilleurs résultats.
3. Le progrès technologique permet aux architectures numériques d'être facilement embarquable contrairement aux appareils optiques.
4. L'implantation des applications sur des architectures numériques est beaucoup plus facile que de les réaliser sur des appareils optiques qui nécessitent un coût et un temps de développement très important.

1.4 Exemple de migration du traitement optique vers un traitement numérique de l'information

1.4.1 Introduction

La corrélation optique est une des techniques les plus performantes pour la reconnaissance des formes [3]. En effet, elle a un très bon niveau de discrimination (prise de décisions) et permet à la fois de détecter et d'estimer un objet cible. Son principe est très simple et consiste à comparer l'image cible (image à reconnaitre) avec une ou plusieurs images références issues d'une base d'apprentissage. La corrélation optique s'est vraiment développée dans les années 80 et 90 selon deux axes : algorithmique et architectural. Malgré ces avancées et les bonnes performances de la corrélation optique, la diffusion de cette méthode s'est essoufflée dans les années 2000. Cette perte d'intérêt, vis-à-vis des méthodes purement numériques, est dû entre autres, à la contrainte qui consiste à imposer une réalisation physique (très rapide) de cette méthode i.e. un montage optique. Une contrainte de rapidité non justifiée dans de nombreux cas d'applications.

En effet, la cadence vidéo est largement suffisante dans bien des cas, comme par exemple le système de détection de chutes de personnes âgées proposé et validé par Elbouz et al [37]. De plus, les méthodes purement numériques bénéficient de l'avantage d'être plus facilement

reconfigurable que les méthodes purement optiques. Un dernier point susceptible de limiter la diffusion des corrélateurs tout optique est la nécessité de post-traiter le plan de corrélation pour prendre une décision. Généralement cela est réalisé numériquement en calculant un critère de décision comme par exemple le PCE : Peak-to-correlation energy [3].

Dans cette partie nous présentons une approche permettant de rendre plus performant ce genre d'applications en adaptant l'algorithme de la corrélation à une implantation toute numérique. Pour cela, nous allons étudier la transformation de Fourier qui est à la base de la méthode de corrélation. En effet, d'un point de vue purement algorithmique, la corrélation de VLC est basée sur la réalisation de deux transformations de Fourier et une multiplication spectrale entre le spectre de l'image cible et un filtre de corrélation bien défini [3]. Nous commençons par proposer une réalisation toute numérique de la transformation optique d'une image. Pour cela, nous nous sommes basés sur la diffraction de Fraunhofer [38]. Ensuite nous proposons deux possibilités pour implanter numériquement le corrélateur de Vanderlugt en utilisant : (1) l'algorithme Fast Fourier Transform et (2) en utilisant une implémentation numérique de la transformation de Fourier optique (NO_{FT} : Numerical Optical Fourier Transform). Ensuite nous présenterons différents tests montrant les avantages de l'utilisation de la NO_{FT}.

1.4.2 Implantation numérique de la transformée de Fourier à base de la diffraction de Fraunhofer

Rappelons que la méthode de corrélation est basée sur le montage $4f$. Cependant, d'un point de vue purement algorithmique un corrélateur peut être représenté par la Figure 1.24.

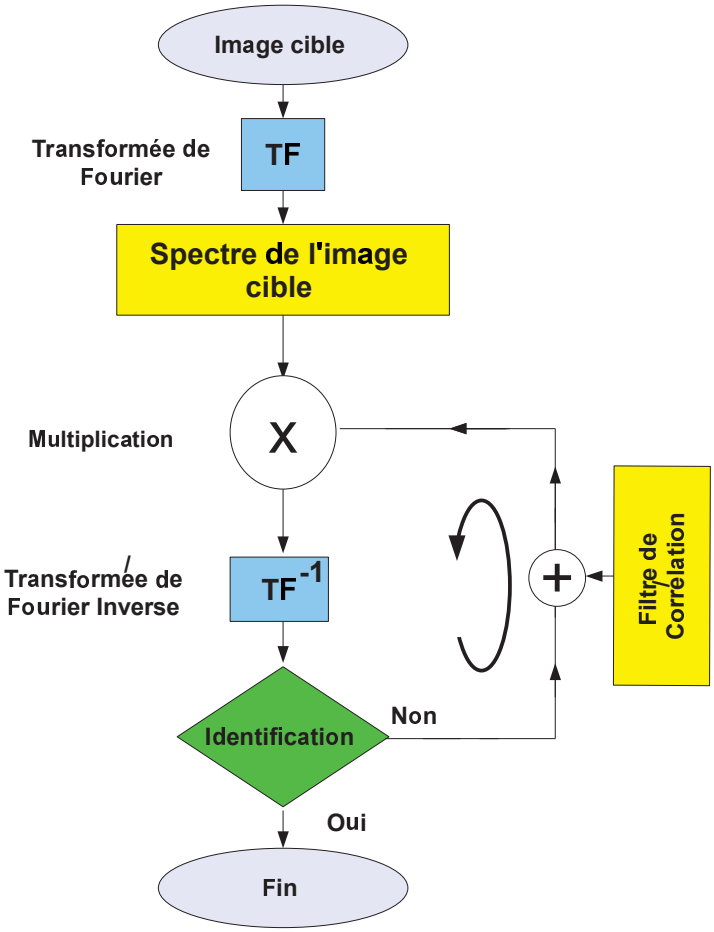


FIGURE 1.24 – Principe de l'algorithme de corrélation

En effet, après avoir pris en compte la scène à étudier (qui contient l'image cible ; image à reconnaitre), nous réalisons une première transformation de Fourier. Cette première " TF

", nous donne le spectre de l'image cible. Ensuite, nous multiplions ce dernier par un filtre de corrélation : multiplication point-par-point. Le filtre de corrélation est fabriqué auparavant à partir d'une image référence selon une méthode bien définie [3]. Dans ce travail nous avons utilisé un filtre de corrélation POF "Phase Only Filter". Par la suite, nous réalisons une transformation de Fourier inverse (TF^{-1}) de l'ensemble. Ainsi, en sortie de l'algorithme, nous obtenons un plan de corrélation. Ce dernier contient un pic de corrélation de hauteur variable, selon le degré de ressemblance entre l'image cible et l'image référence qui a servit à fabriquer le filtre. Finalement une étude de ce pic permet de décider de l'identification ou pas de l'image cible. Pour cela, différents critères ont été proposé dans la littérature pour réaliser objectivement cette étude i.e. dans ce travail nous avons utilisé le critère PCE Peak-to-correlation-Energy [3].

Ce petit rappel montre bien que pour avoir une corrélation numérique, nous avons besoin de réaliser deux transformations de Fourier : une directe (image/spectre) et une inverse (spectre/image). Ainsi il est nécessaire d'étudier la réalisation numérique d'une TF. Dans la littérature, différents algorithmes de calcul numérique de la TF ont été proposés et validés. Cependant, pour avoir un corrélateur rapide, nous nous sommes intéressés aux algorithmes rapides de calcul de la TF comme les algorithmes FFT (Fast Fourier Transform). Ces algorithmes calculent la transformée de Fourier Discrète (DFT) et ne permette pas, en leur forme originaire, de calculer une transformation de Fourier optique . Ce dernier a des caractéristiques différentes d'une simple DFT. Parmi, les différents travaux proposés, nous avons choisi la technique basée sur la diffraction de Fraunhofer pour implémenter numériquement une TF optique. Pour cela on s'est basé sur les travaux publiés [38]. La propagation de Fraunhofer d'une onde objet d'amplitude U_0 avec une lentille peut s'écrite :

$$U_i(u,v) = \frac{e^{jkf}}{j\lambda f} e^{j\frac{k}{2f}(u^2+v^2)} \int\int U_0(x,y).P(x,y).e^{-j\frac{2\pi}{\lambda f}(ux+vy)}dxdy \tag{1.7}$$

où k est le nombre d'onde $k = \frac{2\pi}{\lambda}$, f est la distance focale de la lentille, λ est la longueur d'onde de la source utilisée pour éclairer l'image d'entrée, (u,v) sont les coordonnées fréquentielles et $P(x,y)$ est la fonction pupillaire de la lentille. Dans notre cas nous avons considéré $P(x,y) = 1$ si à l'intérieure de l'ouverture de la lentille et $P(x,y) = 0$ à l'extérieur.

1.4.3 Exemple de corrélation numérique basée sur la FFT

Après avoir choisi et validé un modèle numérique pour calculer une transformation de Fourier optique, nous allons détailler et montrer les résultats d'une implémentation numérique d'un corrélateur de type VanderLugt. L'exemple que nous avons choisi de développer dans cette section est la corrélation d'une image (Figure 1.25 (a)) avec un filtre classique de corrélation de type POF. Ce dernier est fabriqué à partir d'une image référence identique à l'image cible (Figure 1.25 (a)). Une représentation de ce filtre est illustré sur la Figure 1.25 (b). Après avoir calculé numériquement la TF optique de l'image (Figure 1.25 (a)), nous multiplions son spectre point-par-point avec le filtre POF (Figure 1.25 (b)). En calculant une TF inverse de l'ensemble on obtient le plan de corrélation présenté dans la Figure 1.25 (c). Dans la Figure 1.25 (d) nous avons présenté le plan 3D de la corrélation obtenue.

1.4.4 Comparaison des performances de corrélateur numérique

L'implantation numérique en utilisant la FFT optique basée sur la diffraction de Fraunhofer nous a paru intéressante à développer et à étudier. D'autant plus que si l'implémentation optique a été largement étudiée dans la littérature, l'implémentation numérique l'est beaucoup moins. Pour cela nous allons comparer deux implémentations numériques d'un corrélateur VLC utilisant un filtre composite classique POF 5-Références [3] : (1) une réalisation avec l'algorithme FFT et (2) une deuxième en calculant numériquement la TF optique comme cela est montré dans l'article [39]. En effet, un filtre composite classique est une combinaison linéaire d'un ensemble d'images [11, 12]. La robustesse et la discrimination de ces deux réalisations numériques ont été testées vis-à-vis des rotations horizontales et verticales de l'image cible par rapport aux images

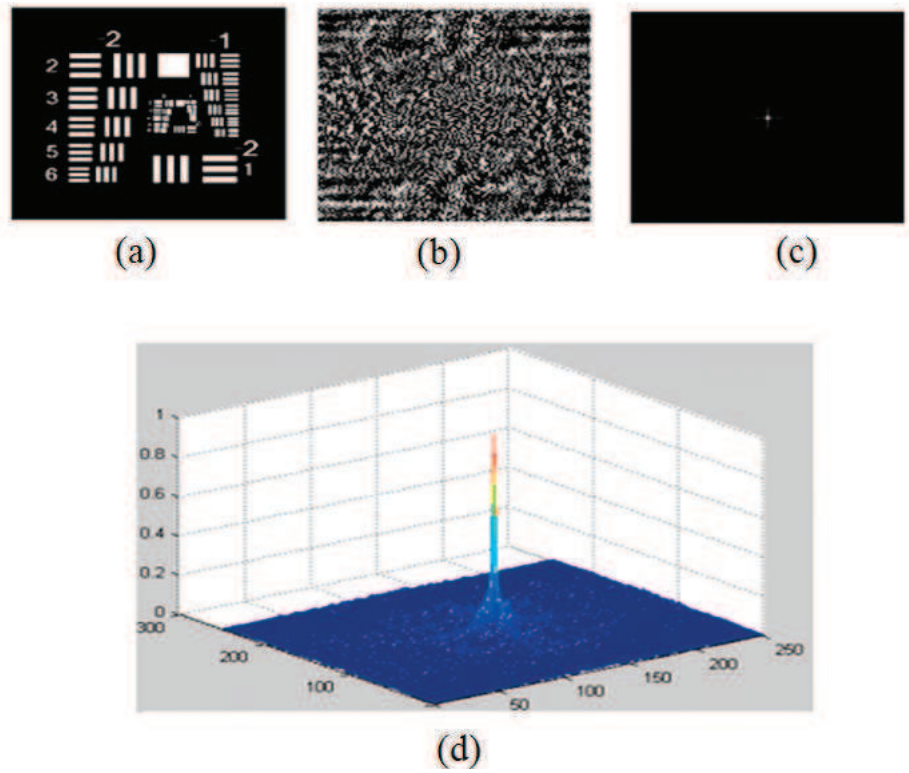


FIGURE 1.25 – Exemple de la corrélation numérique

références. Pour réaliser ces tests nous avons utilisé 4 visages différents de la base PHPID [40]. La Figure 1.26 montre un exemple d'une base utilisée pour la personne 4 (P4) : nous avons un visage avec une rotation horizontale allant de [-90, +90] et une rotation verticale de [-15, +15]. Les images références qui ont servi à fabriquer le filtre composite POF 5_{Refs} sont 5 images de la personne 4 avec une rotation verticale et horizontalement nous avons pris cinq angles (-45, -15, +15, +30 et +45) : $P4_{0(-45)}$; $P4_{0(-15)}$; $P4_{0(+15)}$; $P4_{0(+30)}$; $P4_{0(+45)}$.

Les résultats de corrélation (plan de corrélation en 3D) en utilisant ces deux réalisations numériques sont présentés sur la Figure 1.27. Sur cette figure, nous avons en a, b et c trois exemples de plans de corrélation obtenus en corrélant les images de la personne 4 (P4) (angle : (-15,+30), (0,-30) et (+15,0)) avec un filtre composite 5-références. Sur la Figure 1.26 (d, e et f) nous avons les trois plans de corrélation de la même personne avec le même filtre en calculant numériquement la TF optique (NO_{FT} : Numerical Optical Fourier Transform). Ces plans de corrélation montrent très clairement qu'avec une implémentation numérique de la TF optique (NO_{FT}) nous obtenons des meilleurs résultats qu'avec la FFT. En effet les plans des corrélations

FIGURE 1.26 – Examples de rotation faciale

(d, e et f) sont moins bruités et présentent des pics de corrélation plus fins. Cela prouve que la TF optique permet d'avoir une corrélation plus robuste. En effet pour l'angle (+15,0) (c) le pic de corrélation est absent avec l'algorithme FFT, cependant on le voit très bien avec un NO_{FT} (f).

Pour confirmer cela nous avons corrélé toute la base (Figure 1.26) avec le filtre composite 5_{Refs}. De plus, nous avons corrélé le même filtre avec trois autres personnes i.e. pour tester la discrimination. Finalement, nous avons présenté les résultats sous la forme des courbes ROC "Receiver Operating Characteristic" [41]. Chaque point de la courbe ROC est obtenu en fonction de la bonne détection (en Y) par rapport aux fausses alarmes (en x). Le principe des courbes ROC est détaillé dans l'annexe A. La Figure 1.28 (a) présente la courbe ROC obtenue avec un corrélateur numérique utilisant l'algorithme FFT de MATLAB. La courbe ROC obtenue en calculant numériquement la TF optique (NO_{FT}) est donnée sur la Figure 1.28 (b). La comparaison entre ces deux courbes montrent bien les très bonnes performances, en terme de robustesse et de discrimination, du corrélateur implantant la TF optique. En effet avec la TF optique nous avons 71% de bonnes détections pour 0% de fausse alarme. Cela prouve bien que pour des simulations numériques ou pour une implémentation toute numérique des corrélateurs, il vaut mieux utiliser le calcul numérique de la TF optique (NO_{FT}) plutôt que la FFT.

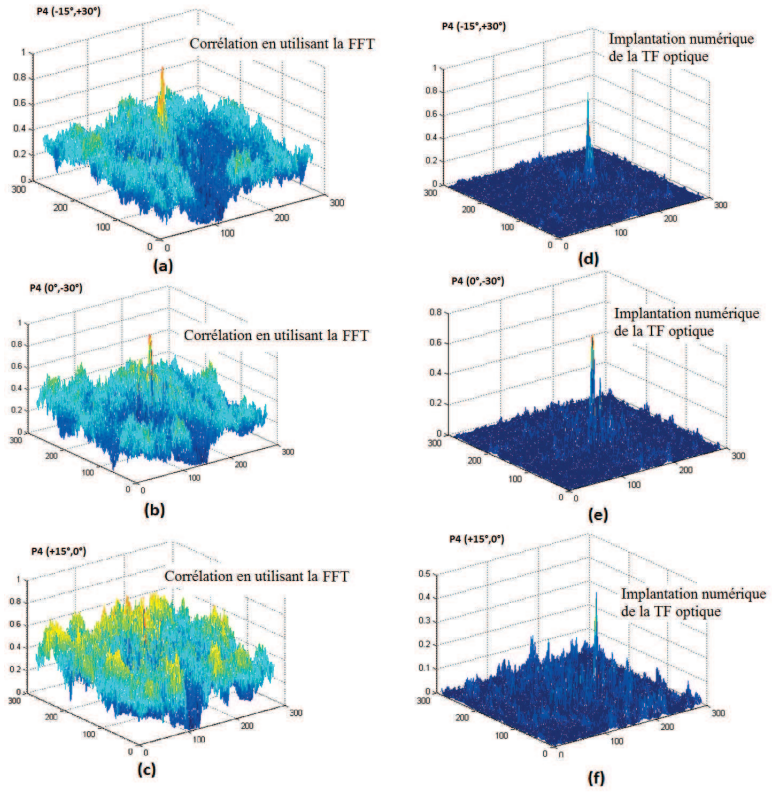


FIGURE 1.27 – Plans de Corrélations

1.4.5 Conclusion

Dans ce travail, nous avons montré une technique pour calculer numériquement la Transformation optique (NO_{FT}). Ensuite, nous avons utilisé cette réalisation numérique pour implémenter numériquement un corrélateur optique. Cette implémentation toute numérique s'est avérée très robuste et très discriminante. Cela est prouvé par une comparaison avec un corrélateur numérique implémentant une transformation de Fourier avec l'algorithme de base " Fast Fourier Transform " (FFT).

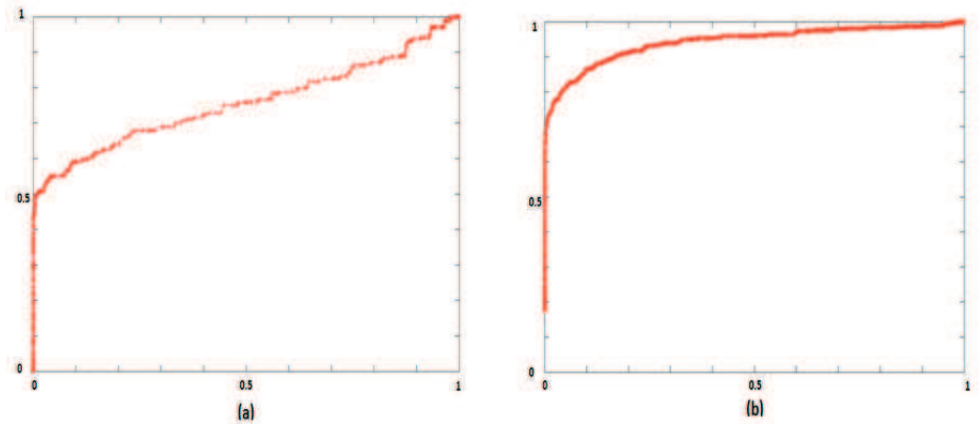


FIGURE 1.28 – Courbes ROC

1.5 Conclusion

Ce chapitre a été consacré à la présentation des différents algorithmes et applications optiques en expliquant leurs principe de fonctionnement. Afin de faire face aux inconvénients de l'implémentation optique qui souffre de plusieurs problèmes tels que la mauvaise qualité des résultats, le coût, la complexité de réalisation, nous nous sommes intéressés aux architectures numériques qui offrent une facilité d'implémentation et une très bonne résolution du résultat. Cependant ces architectures sont moins rapides en les comparant aux appareils optiques. Pour résoudre ce problème nous nous intéressons aux architectures offrant un bon compromis en termes de performances/flexibilité et nous essayons d'optimiser l'algorithme et l'architecture de l'implantation numérique des algorithmes de reconnaissances de formes et de compression. Par conséquent nous commençons par proposer une IP optimisée pour le calcul de la transformée de Fourier l'un des algorithmes les plus utilisés dans les applications de reconnaissances de formes.

Deuxième partie

IP pour le traitement de l'Image et du Signal

Chapitre 2

Conception d'architectures optimisées pour la Transformée de Fourier Rapide

Sommaire

2.1 Introduction

La Transformée de Fourier Discrète (TFD plus connue sous le DFT "Discrete Fourier Transform") est l'opération qui permet de calculer la transformée de Fourier d'un signal discret. La DFT est un des algorithmes les plus utilisés dans les applications de traitement du signal et de l'image comme dans les applications de reconnaissance des formes [3], de détection de signaux [42], et de télécommunications [43]. Toutes ces applications, basées sur la DFT, exigent un temps de traitement très rapide afin de se rapprocher des contraintes temps réel. C'est dans ce cadre que s'inscrivent nos travaux : nous visons à proposer une IP optimisée pour le calcul de la FFT.

Nous commençons ce chapitre par une brève définition de la Transformée de Fourier discrète et rapide. Par la suite, nous présentons les différents algorithmes de calcul des coefficients de la FFT. Dans la section 2.4, nous listons les différentes architectures permettant une implantation FPGA de la FFT en présentant les avantages et les inconvénients de chaque architecture. Finalement, nous décrivons les architectures que nous avons proposées et nous présentons les résultats de l'implantation matérielle.

2.2 Définitions

2.2.1 Transformée de Fourier

Une fonction, par définition, est l'opération permettant de faire correspondre à un intervalle de départ un intervalle d'arrivée. Cependant, la transformation fait correspondre un ensemble de fonctions vers un autre ensemble de fonctions. Parmi les transformations les plus connues dans les applications de traitement d'images et de signal nous trouvons la Transformée de Fourier.

Découverte par le mathématicien et physicien français Joseph Fourier (1768-1830), la transformée de Fourier consiste à décomposer tout signal continu en une somme de sinusoïdes. La Transformée de Fourier permet ainsi de convertir la représentation temporelle d'un signal d'entrée en une représentation fréquentielle. En effet, le signal est représenté sous forme d'une combinaison linéaire infinie des fonctions trigonométriques de toutes les fréquences qui forment son spectre. Ainsi, la transformée de Fourier permet de donner une autre vue du signal (vue fréquentielle) différente de la vue temporelle pour permettre une analyse spectrale du signal.

Etant donné que les signaux qui se propagent dans notre environnement sont de nature analogique (température, pression, onde sonore, etc.), la transformée de Fourier qui s'applique à ces signaux est une TF continue. La TF continue de Fourier s'applique aux fonctions intégrables et les représente par une intégrale pondérée d'exponentielles complexes. Ainsi si une fonction f est intégrable sur $\mathbb{R}$, sa Transformée de Fourier est la fonction F donnée par la formule suivante :

$$F(v) = \int_{-\infty}^{+\infty} f(t) e^{-2j\pi vt} dt \tag{2.1}$$

La fonction $f(t)$ peut être obtenue à partir de $F(v)$ en utilisant la Transformée de Fourier inverse donnée par :

$$f(t) = \int_{-\infty}^{+\infty} F(v) e^{+2j\pi vt} dv \tag{2.2}$$

avec $t, v \in \mathbb{R}$.

2.2.2 Transformée de Fourier Discrète

Le calcul de la transformée de Fourier par des moyens informatique ne peut pas être réalisé sur l'espace infini des réels $\mathbb{R}$, ce qui nécessiterait un temps de calcul infini. En revanche, la théorie du traitement numérique du signal a permis d'adapter la transformée de Fourier sur un nombre fini d'échantillons donnant lieu à l'algorithme de Transformée de Fourier Discrète (DFT). Pour un signal discret de N échantillons, la DFT $X(k)$ est définie par l'équation 2.3 :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-2j\pi nk}{N}} \tag{2.3}$$

avec $n, k \in [0, N-1]$.

Notons que l'équation 2.3 peut être représentée sous forme d'une multiplication matricielle donnée par :

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & W_N^1 & W_N^2 & \dots & W_N^{N-2} & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{N-2} & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-2)(N-1)} & W_N^{(N-1)(N-1)} \end{pmatrix} \times \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{pmatrix} \tag{2.4}$$

W_N^n sont appelés facteurs de rotation et sont définis par $W_N^n = e^{\frac{-2j\pi n}{N}}$.

Etant donné que la transformation de Fourier est inversible, le signal $x(n)$ peut être restitué en utilisant la Transformée de Fourier Discrète inverse donnée par :

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) e^{\frac{+2j\pi nk}{N}} \tag{2.5}$$

2.2.3 Transformée de Fourier Rapide

Le calcul de la TFD pour un signal complexe à N échantillons nécessite N^2 multiplications complexes et $N(N-1)$ additions complexes. Par conséquent, l'implantation de cet algorithme sur des architectures programmables nécessite un nombre d'opérateurs arithmétiques très élevé. Ceci rend complexe cette réalisation voire impossible si N est grand. Pour faire face à ce problème, Cooley-Tukey [44] ont proposé et validé un algorithme basé sur l'utilisation des propriétés de périodicité et de symétrie des facteurs de rotation comme indiqué dans l'équation 2.6. Ainsi, ils ont introduit un premier algorithme de calcul de la TFD d'une manière rapide d'où son nom Transformée de Fourier Rapide TFR ou FFT "Fast Fourier Transform".

$$W_N^k = W_N^{k+\alpha N} = W_N^{k+\frac{N}{2}} \tag{2.6}$$

avec α est un entier naturel.

L'algorithme proposé par Cooley Tukey, connu aussi sous le nom de Radix-2, a permis de réduire la complexité algorithmique de calcul de la TFD de $O(N^2)$ à $O(NlogN)$. Suite à cette proposition, plusieurs algorithmes ont été proposés dans la littérature afin d'optimiser le temps de calcul et réduire davantage la complexité d'implantation de la DFT. Parmi ces algorithmes, nous citons radix-4 [45], Split Radix [46], radix-8 [47], prime factor [48], etc.

Ce qui change d'un algorithme à l'autre c'est le nombre d'entrées et de sorties utilisées. Ainsi, la FFT à base de Radix-4 se calcule en utilisant des éléments de calcul à 4 entrées et 4 sorties alors que celle basée sur le Radix-8 utilise 8 entrées et 8 sorties. De plus, l'organisation des calculs au sein d'un bloc Radix-2, Radix-4 ou Radix-8 peut se faire de deux façons. Nous pouvons subdiviser les échantillons en pairs et impairs. Il s'agit là d'une décimation dans le temps (DIT "Decimation in Time"). Ou alors, nous classons les échantillons d'entrée à la FFT selon leur ordre d'arrivée et calculons les sorties paires et impaires, dans ce cas c'est une décimation en fréquence (DIF "Decimation in Frequency").

2.3 Algorithmes pour la FFT

2.3.1 Algorithme à base de Radix

2.3.1.1 Radix 2

C'est le premier algorithme qui a été introduit afin de réduire le nombre d'opérateurs utilisé pour le calcul de la transformée de Fourier discrète. Le principe de cet algorithme consiste à décomposer le calcul de la DFT en deux parties selon deux méthodes : Radix-2 DIT et Radix-2 DIF.

Pour un signal d'entrée $x(n)$, $n \in [0, N-1]$, sa transformée de Fourier $X(k)$, $k \in [0, N-1]$ définie dans l'équation 2.3 peut s'écrire par l'équation 2.7 :

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{2.7}$$

Radix-2 DIT

L'algorithme Radix-2 DIT décompose les entrées de la transformée de Fourier Discrète en deux parties. Une première partie contient les entrées à indices paires et une deuxième contient

les entrées à indices impaires. Une DFT de taille $\frac{N}{2}$ est appliquée à chacune des deux parties et les résultats obtenus sont combinés pour donner une DFT de taille N points. Cette procédure est répétitive jusqu'à l'obtention des transformées simples de taille 2 points. Par conséquent la valeur de N doit être une puissance de 2. Le principe de cet algorithme est donné par l'équation 2.8 :

$$\begin{aligned} X(k) &= \overbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk}}^{Partie\ paire} + \overbrace{\sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{(2n+1)k}}^{Partie\ impaire} \\ X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{2nk}W_N^k \\ X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(2n)W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)W_N^{(2n)k} \end{aligned} \tag{2.8}$$

Nous considérons $x_1(r) = x(2n)$ et $x_2(r) = x(2n+1)$ et comme $W_N^{2k} = W_{\frac{N}{2}}^k$, nous pouvons ainsi écrire :

$$\begin{aligned} X(k) &= \sum_{r=0}^{\frac{N}{2}-1} x_1(r)W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r)W_{\frac{N}{2}}^{rk} \\ X(k) &= X_1(k) + W_N^k X_2(k) \end{aligned} \tag{2.9}$$

avec r, $k \in [0, \frac{N}{2} - 1]$, X_1 est la DFT de taille $\frac{N}{2}$ des entrées à indices pairs et X_2 celle des entrées à indices impaires. D'autre part, les sorties de la DFT peuvent être divisées en deux parties. La première partie peut s'écrire conformément à 2.10 :

$$X(k) = X_1(k) + W_N^k X_2(k) \tag{2.10}$$

La deuxième partie est décrite par 2.11 :

$$\begin{aligned} X(k + \frac{N}{2}) &= X_1(k + \frac{N}{2}) + W_N^{k+\frac{N}{2}} X_2(k + \frac{N}{2}) \\ X(k + \frac{N}{2}) &= X_1(k + \frac{N}{2}) - W_N^k X_2(k + \frac{N}{2}) \end{aligned} \tag{2.11}$$

pour toute valeur de $k \in [0, \frac{N}{2} - 1]$.
Nous obtenons ainsi :

$$\begin{aligned} X(k) &= X_1(k) + W_N^k X_2(k) \\ X(k + \frac{N}{2}) &= X_1(k + \frac{N}{2}) - W_N^k X_2(k + \frac{N}{2}) \end{aligned} \tag{2.12}$$

Afin de donner un exemple pour le calcul de la FFT utilisant l'algorithme Radix-2 DIT, nous avons présenté graphiquement les différentes étapes de calcul dans le cas $N = 8$. Le schéma bloc de la Figure 2.1 illustre un exemple de calcul pour un vecteur à 8 points $(x(0), .., x(7))$. Nous remarquons que ce schéma est basé sur l'utilisation d'un élément de calcul (PE pour "Processing Element") appelé aussi papillon ou Butterfly. De plus, notons que la FFT 8 points à base de l'algorithme Radix-2 DIT est réalisé en trois étages de calcul.

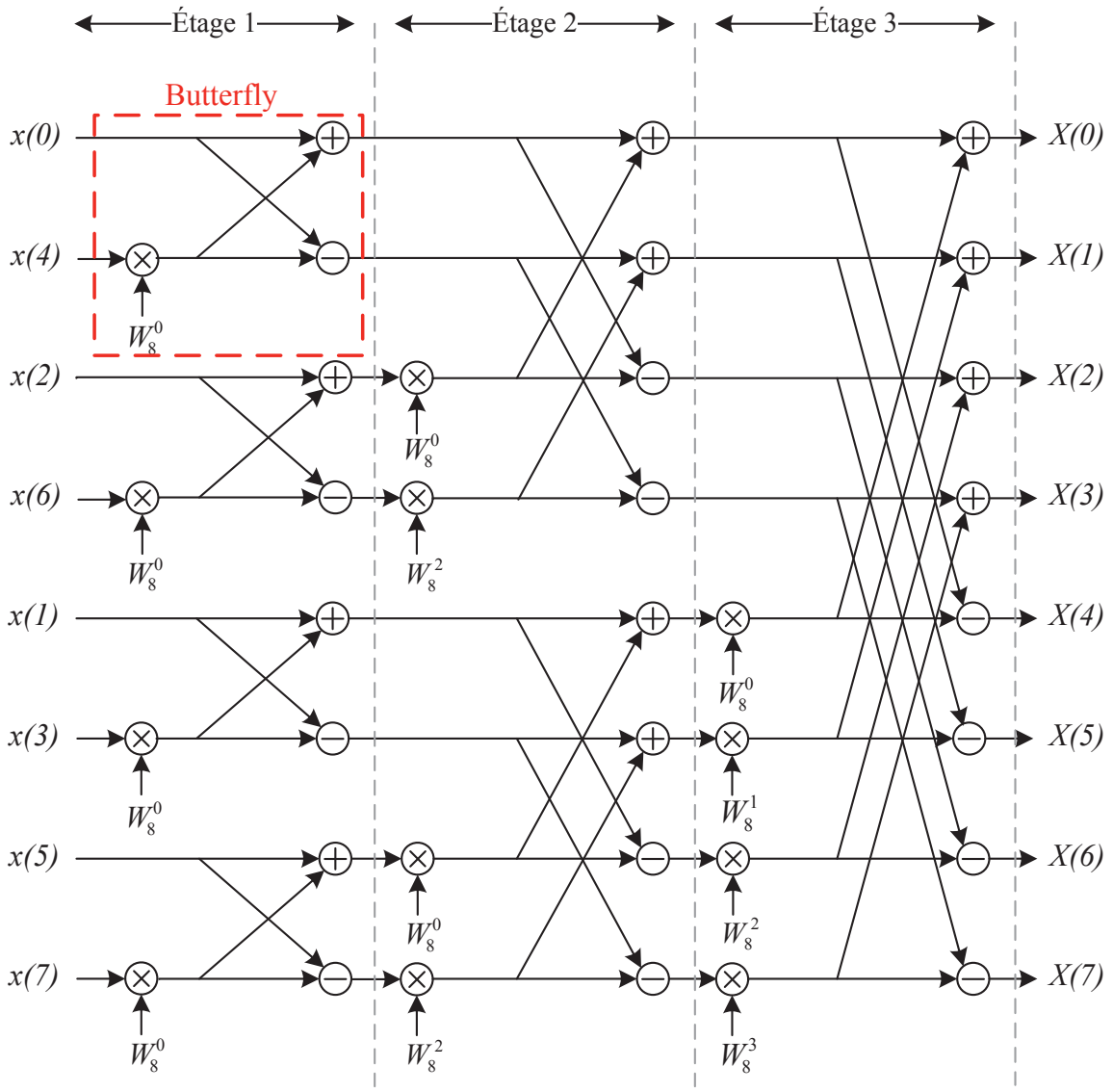


FIGURE 2.1 – FFT 8 points à base de l'algorithme Radix-2 DIT

2.3.1.2 Radix-2 DIF

L'algorithme Radix-2 DIF décompose aussi les entrées de la Transformée de Fourier Discrète en deux parties. Une première rassemblant les $\frac{N}{2}$ premières entrées, tandis que la deuxième partie rassemble les entrées à indices supérieurs ou égaux à $\frac{N}{2}$. Comme dans le cas de la DIT, une DFT de taille $\frac{N}{2}$ est appliquée séparément à chacun des deux groupes d'échantillons. Par la suite, ces deux DFT de taille $\frac{N}{2}$ points sont combinées pour former une DFT de taille N points. La procédure est aussi répétitive jusqu'à l'obtention des transformées de taille 2 points. Le principe de cet algorithme est donné par les équations 2.13 et 2.14

$$\begin{aligned} X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n)W_N^{nk} \\ X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\frac{N}{2})W_N^{(n+\frac{N}{2})k} \end{aligned} \tag{2.13}$$

$$\begin{aligned} X(k) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n)W_N^{nk} + W_N^{\frac{N}{2}k} x(n+\frac{N}{2})]W_N^{nk} \\ X(k) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n) + (-1)^k x(n+\frac{N}{2})]W_N^{nk} \end{aligned} \tag{2.14}$$

Les sorties de la FFT Radix-2 DIF peuvent être séparées en deux parties. Les sorties paires sont présentées par l'équation 2.15 :

$$\begin{aligned} X(2k) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+\frac{N}{2})]W_N^{2nk} \\ X(2k) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+\frac{N}{2})]W_{\frac{N}{2}}^{nk} \end{aligned} \tag{2.15}$$

Quant à la partie impaire, elle est représentée par l'équation 2.16

$$\begin{aligned} X(2k+1) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n) - x(n+\frac{N}{2})]W_N^{n(2k+1)} \\ X(2k+1) &= \sum_{n=0}^{\frac{N}{2}-1} [x(n) - x(n+\frac{N}{2})]W_{\frac{N}{2}}^{nk} W_N^n \end{aligned} \tag{2.16}$$

avec k $\in [0, \frac{N}{2} - 1]$.
Le " butterfly " correspondant à l'algorithme Radix-2 DIF pour $N = 2$ est illustré par la Figure 2.2. A noter que le Butterfly utilisé dans cet algorithme Radix-2 DIF est différent de celui utilisé dans l'algorithme Radix-2 DIT présenté sur la Figure 2.1. En effet, le positionnement de la multiplication est différent d'un butterfly à l'autre.

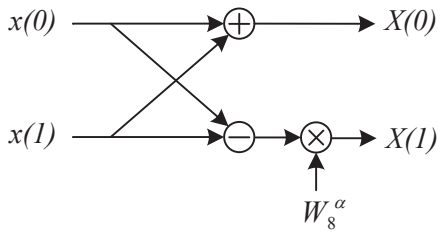


FIGURE 2.2 – Butterfly de l'algorithme Radix-2 DIF

Sur la Figure 2.2, α est un nombre entier positif qui dépend de la position de l'étage du Butterfly ainsi que de la position de ce dernier dans l'étage.

De manière plus générale, si N est une puissance de 2, on peut réitérer l'algorithme Radix-2 $log_2(N)$ fois et calculer la TFD d'ordre N à l'aide de $log_2(N)$ étages de $\frac{N}{2}$ papillons chacun. Le nombre d'opérations ainsi requis est de $log_2(N)\frac{N}{2}$ multiplications complexes et $log_2(N)N$ additions complexes. Soit une complexité de calcul en $O(Nlog_2(N))$.

2.3.1.3 Radix 4

L'algorithme radix-4 de la FFT décompose l'algorithme de transformée de Fourier discrète en 4 transformées indépendantes de taille $\frac{N}{4}$ [45]. De la même manière que l'algorithme Radix-2, deux représentations sont possibles : Radix-4 DIF et Radix-4 DIT. A titre d'exemple nous présentons, dans ce paragraphe, le principe de l'algorithme Radix-4 DIF.

Si N est une puissance de 4, les équations permettant de calculer les sorties d'une FFT de taille N points en utilisant l'algorithme Radix-4 DIF sont données par :

$$\begin{aligned}
X(4k) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) + x(n+\frac{N}{4}) + x(n+\frac{N}{2}) + x(n+\frac{3N}{4})] W_N^0 W_{\frac{N}{4}}^{kn} \\
X(4k+1) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) - jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})] W_N^n W_{\frac{N}{4}}^{kn} \\
X(4k+2) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) - x(n+\frac{N}{4}) + x(n+\frac{N}{2}) - x(n+\frac{3N}{4})] W_N^{2n} W_{\frac{N}{4}}^{kn} \\
X(4k+3) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) + jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})] W_N^{3n} W_{\frac{N}{4}}^{kn}
\end{aligned} \tag{2.17}$$

Avec $k \in [0, \frac{N}{4} - 1]$.

Le plus petit élément de calcul (PE) utilisé dans cet algorithme est présenté dans la Figure 2.3 pour N=4.

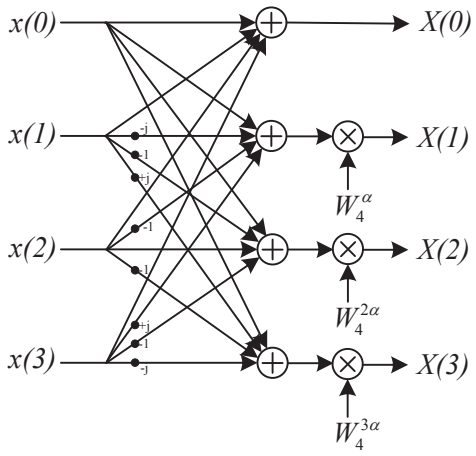


FIGURE 2.3 – PE de l'algorithme Radix-4 DIF

De la même manière que dans la Figure 2.2, α est un nombre entier positif qui dépend de la position de l'étage du Butterfly ainsi que de la position de ce dernier dans l'étage.

En comparaison avec l'algorithme Radix-2, l'algorithme Radix-4 est une succession de $log_4(N)$ étages de $\frac{N}{4}$ PE où N est une puissance de 4 représentant le nombre d'échantillons constituant le signal considéré. Le nombre d'opérations requis par cet algorithme est de $3log_4(N)$ multiplications complexes et $8log_4(N)$ additions complexes. Ce nombre est inférieur au nombre d'opérations demandé dans l'algorithme Radix-2.

2.3.2 Algorithme mixte : Split Radix

Introduit en 1984 par P. Duhamel et H. Hollmann [46], l'algorithme Split Radix de la FFT est une combinaison des algorithmes Radix-2 et Radix-4 afin de bénéficier des avantages des deux algorithmes. En effet, avec l'algorithme Radix-4 les nombres d'opérateurs et d'étages (latence) sont inférieurs à ceux requis par l'algorithme Radix-2, cependant, avec ce dernier nous avons 2

fois plus de choix au niveau de la taille de la FFT. En effet, l'algorithme Radix-2 ne peut être utilisé que pour des valeurs de N puissance de 2 alors que l'algorithme Radix-4 ne peut être utilisé que pour des valeurs de N puissance de 4.

L'algorithme Split Radix divise récursivement le calcul de la FFT en une FFT de taille $\frac{N}{2}$ (utilisation de l'algorithme Radix-2) et deux FFT de taille $\frac{N}{4}$ (utilisation de l'algorithme Radix-4). La première phase pour le calcul des coefficients de la FFT en utilisant l'algorithme Splix Radix consiste à calculer les sorties paires de la FFT en utilisant l'algorithme Radix-2 DIF. Cette phase se fait conformément à l'équation 2.18 :

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n + \frac{N}{2})] W_{\frac{N}{2}}^{n} \tag{2.18}$$

avec $k \in [0, \frac{N}{2} - 1]$.

Ensuite, les sorties impaires sont calculées en utilisant l'algorithme Radix-4. Le calcul de ces sorties en utilisant l'algorithme Radix-4 DIF se fait conformément à l'équation 2.19.

$$\begin{aligned} X(4k+1) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) - jx(n + \frac{N}{4}) - x(n + \frac{N}{2}) + jx(n + \frac{3N}{4})] W_N^n W_{\frac{N}{4}}^{kn} \\ X(4k+3) &= \sum_{n=0}^{\frac{N}{4}-1} [x(n) + jx(n + \frac{N}{4}) - x(n + \frac{N}{2}) - jx(n + \frac{3N}{4})] W_N^{3n} W_{\frac{N}{4}}^{kn} \end{aligned} \tag{2.19}$$

avec $k \in [0, \frac{N}{4} - 1]$.

Le PE correspondant à l'algorithme Split Radix DIF est ainsi donné par la Figure 2.4.

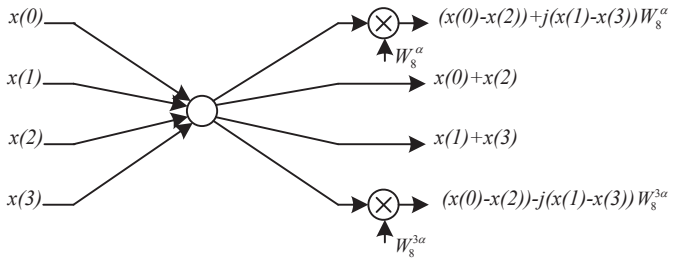


FIGURE 2.4 – PE de l'algorithme Split Radix DIF

Afin d'expliquer davantage le principe de cet algorithme nous considérons le cas où $N = 8$. Dans ce cas, le schéma bloc de calcul de la FFT de taille 8 points à base de l'algorithme Split Radix est présenté sur la Figure 2.5. Comme il est montré dans cette figure, nous utilisons 3 PE Split Radix et 3 PE Radix-2. Pour une FFT de taille N points, le nombre d'opérateurs requis par cet algorithme est de $N(\frac{log_2(N)}{3} - \frac{2}{9}) + \frac{2}{9}(-1)^{log_2(N)}$ multiplications complexes et $N(log_2(N) + \frac{1}{3}) + \frac{2}{3}(-1)^{log_2(N)}$ [49]. Le nombre d'opérateurs est ainsi inférieur à celui utilisé dans l'algorithme Radix-2 et aussi à celui utilisé dans l'algorithme Radix-4.

2.3.3 Autres algorithmes

2.3.3.1 L'algorithme FFT Prime Factor

L'algorithme Prime Factor aussi connu sous le nom PFA (Prime Factor Algorithhm) est un des algorithmes de calcul rapide de la transformée de Fourier proposé dans la littérature. Ce dernier calcule la DFT de taille $N = N_1 \times N_2$, en se basant sur une DFT à deux dimensions $N_1 \times N_2$; où N_1 et N_2 sont des nombres premiers entre eux [48]. Pour illustrer le principe de cet algorithme, considérons l'exemple de calcul d'une DFT $X(k)$ de N points pour une séquence

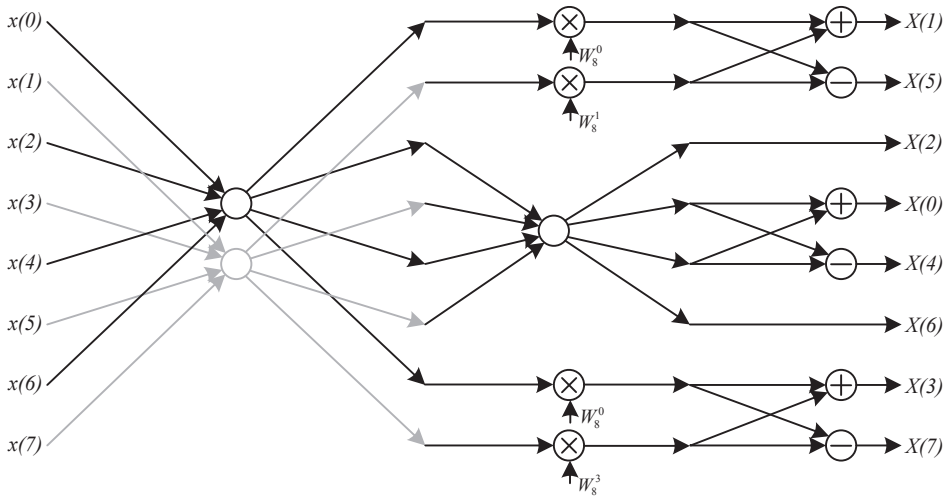


FIGURE 2.5 – Schéma bloc d'une FFT 8 points à base de Split Radix DIF

d'entrée $x(n)$ avec $n \in [0, N-1]$. Les deux vecteurs $x(n)$ et $X(k)$ ont les propriétés suivantes : $x(n_1, n_2) = x(N_2 n_1 + n_2)$ et $X(k_1, k_2) = X(k_1 + N_1 k_2)$, avec n_1, n_2 et k_1, k_2 sont respectivement les indices temporels et fréquentiels : n_1, $k_1 \in [0,\ N_1 - 1]$ et n_2, $k_2 \in [0,\ N_2 - 1]$ [50]. La DFT peut être calculée par :

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_{N_1}^{(n_1+\gamma n_2)k} W_{N_1}^{n_2(k_2+\beta k_1)} \tag{2.20}$$

avec γ et β des entiers tel que $\alpha N_2 + \beta N_1 = 1$ modulo$N_1 N_2$.

Les inconvénients de cet algorithme résident dans le fait que son architecture a une structure irrégulière. Cela rend son implantation matérielle très difficilement réalisable. De plus, il y a une restriction liée à la taille de la FFT calculée avec cet algorithme puisque N doit être exprimé en produit de 2 nombres premiers entre eux.

2.3.3.2 L'algorithme FFT de Stockham

L'algorithme FFT de Stockham est développé pour concevoir un algorithme qui s'affranchit des problèmes liés à l'ordre des entrées et des sorties des algorithmes à base de Radix [51]. Le cœur de calcul de l'algorithme s'appui sur les algorithmes Radix-2 ou Radix-4. La structure de cet algorithme est ainsi conforme aux algorithmes radix-2 et Radix-4. Par conséquent la complexité de calcul de cet algorithme est strictement identique à celle de Radix-2 ou Radix-4. Tout comme l'algorithme à base de Radix, l'algorithme FFT de Stockham est une succession de $log_2(N)$ étages de $\frac{N}{2}$ papillons dans le cas d'une utilisation de l'algorithme Radix-2. Cependant, l'algorithme de Stockham diffère de celui à base de Radix par son intégration de l'ordonnancement des entrées et des sorties dans le calcul de la FFT. Cet ordonnancement est réalisé en utilisant une mémoire supplémentaire dans l'implantation de l'algorithme. Cette mémoire permet de changer l'emplacement des résultats après chaque étage de calcul de la FFT. Les mémoires initiales et supplémentaires alternent leur rôle après chaque étape de mémorisation. L'avantage principal de cet algorithme est la production des résultats dans l'ordre naturel sans avoir besoin d'une étape d'ordonnancement. Il permet ainsi d'améliorer la latence de calcul des algorithmes à base de Radix. Cependant, son inconvénient majeur réside dans l'utilisation de mémoires supplémentaires à chaque étage [52].

Pour la suite de ce travail nous avons choisi d'utiliser l'algorithme Radix-4 pour plusieurs raisons. En effet :

– Il est à base d'un PE qui permet de fournir quatre sorties en parallèles, ce qui engendre une augmentation du débit.

- Une architecture à base de l'algorithme Radix-4 contient moins d'étages comparée à celle à base des algorithmes Radix-2 ou Split Radix. Par conséquent, elle est plus rapide tout en utilisant moins de registres intermédiaires.
- Comparé avec les Radix de taille supérieures (8, 16, etc.), le Radix-4 présente l'avantage d'avoir une surface occupée réduite par rapport à celle occupée par ces autres Radix. De plus la taille de la FFT à implanter à base de ces Radix est très restreinte.

Cependant, la seule contrainte à respecter pour utiliser cet algorithme est que le nombre de points N doit être une puissance de 4.

2.4 Architectures matérielles pour l'implantation des algorithmes FFT

Grâce à son utilisation massive dans les applications de traitement d'image et du signal, l'implantation de la FFT sur FPGA est devenue une thématique courtisée. En effet, le nombre important d'opérations et de mémoires pour calculer les sorties de la FFT en un temps d'exécution réduit constituent les problèmes majeurs de cette implantation. Afin d'alléger les contraintes matérielles, plusieurs architectures ont été proposées. Les architectures étudiées dans cette thèse, peuvent être classées en trois catégories. La première catégorie concerne les architectures à base de réalisation directe. La deuxième réunit les architectures à base de mémoires. La dernière concerne les architectures "pipelines".

2.4.1 Architecture à réalisation directe

La première catégorie des architectures implantant la FFT sur FPGA est la réalisation directe. Cette architecture consiste à réaliser une implantation matérielle de la FFT en se basant sur la multiplication matricielle de l'équation 2.4. L'idée générale consiste à dupliquer dans l'espace plusieurs blocs de calcul élémentaire PE. Afin d'illustrer le principe de cette architecture, nous nous basons sur la division de l'équation de la FFT introduite par Shousheng et Torkelson dans [53]. L'équation 2.7 peut ainsi être écrite sous la forme :

$$X(s+Mq+MEp) = \sum_{l=0}^{L-1}\sum_{m=0}^{M-1}\sum_{e=0}^{E-1} x(l,m,e) W_N^{(M.E.l+M.m+e)(M.E.p+M.q+s)} \tag{2.21}$$

Avec $N = M \times E \times L$; $k = s + M.q + M.E.p$; $n = M.E.l + M.m + e$; M, E et L sont des entiers ; $l \in [0, L-1]$, $m \in [0, M-1]$ et $e \in [0, N-1]$. Ainsi une FFT de taille N points peut être divisée en une FFT à trois dimensions. Nous Considérons le cas $N = 64$ et $M = E = L = 4$. Par conséquent, nous pouvons calculer les sorties de la FFT de taille 64 points en la divisant en des FFT de taille 4 points.

Afin de simplifier la représentation du la FFT 64 points à base du PE Radix-4, nous simplifions la représentation de ce dernier comme montré dans la Figure 2.6. Le schéma bloc d'une FFT de taille 64 points à base de la réalisation directe en utilisant le PE Radix-4 est illustré par la Figure B.1 (Annexe B).

L'architecture d'une FFT en utilisant l'algorithme Radix-4 est composée de plusieurs étages. Pour une FFT de taille N points, nous avons besoin de $\alpha = log_4(N)$ étages et $\beta = \frac{N}{4}$ butterfly par étage. La réalisation de FFT 64 points, en utilisant le PE de l'algorithme Radix-4, donné par la Figure B.1 contient 3 étages ($\alpha = 3$) et 16 Butterfly ($\beta = 16$). Les sorties de chaque étage dépendent de celles de l'étage qui le précède. Plus généralement, il y a une dépendance entre les données et les différents étages de calcul de la FFT.

Pour une FFT de taille N points nous avons besoin de $\frac{N}{4}log_4(N)$ FFT 4 points ce qui est équivalent à $\frac{3N}{4}log(N)$ multiplieurs complexes et $\frac{8N}{4}log(N)$ additionneurs complexes. L'avantage de cette architecture réside dans sa rapidité. Par contre ses inconvénients majeurs sont : (1) la surface occupée et (2) le nombre d'opérateurs utilisés.

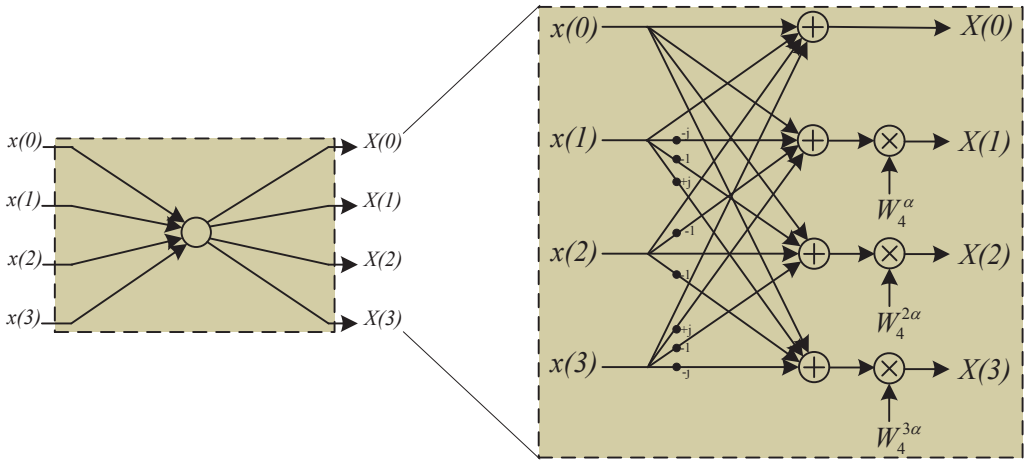


FIGURE 2.6 – PE simplifié de l'algorithme Radix 4

D'un autre côté, il faut signaler le fait que cette architecture fait appel à des traitements rapides. En effet, le temps d'exécution du PE de l'algorithme Radix-4 est de $2T_A + T_M$, où T_A désigne le temps d'une addition et T_M est le temps d'exécution d'une multiplication. D'autre part, le PE utilisé dans le dernier étage de cette architecture ne contient pas de multiplieurs. Ainsi, la latence, définie par le temps écoulé entre l'arrivée de la première donnée et la présence de la première sortie, requise pour cette architecture est donnée par l'équation 2.22 :

$$L = 2log_4(N)T_A + (log_4(N) - 1)T_M \tag{2.22}$$

2.4.2 Architecture récursive

La deuxième méthode consiste à réaliser la transformée avec un seul bloc FFT dont sa fonctionnalité est exécutée $\frac{N}{i}log_i N$ fois, avec N la longueur de la transformée et i correspond à la taille du PE utilisé. Un exemple de réalisation de cette méthode à base de l'algorithme Radix-4 est donné par la Figure 2.7 [54].

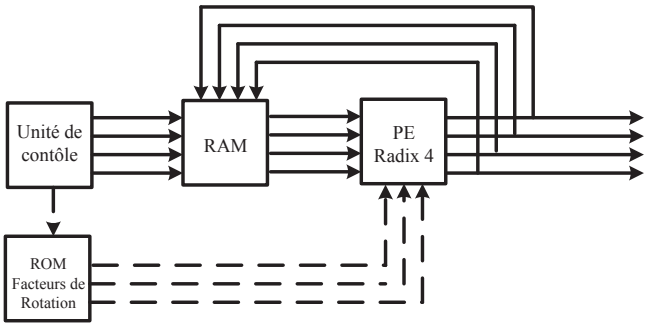


FIGURE 2.7 – Implantation récursive de la FFT

Cette architecture est composée de deux mémoires, une unité de contrôle et un bloc PE de l'algorithme Radix-4. La mémoire RAM est utilisée pour sauvegarder les entrées ainsi que les données intermédiaires des différents étages. La mémoire ROM est utilisée pour sauvegarder les valeurs des facteurs de rotation. L'unité de contrôle sert à contrôler le fonctionnement du PE que des phases de lecture et d'écriture des deux mémoires et elle sert aussi à la génération des adresses de la mémoire des données ainsi que de la mémoire des facteurs de rotation.

D'autres configurations de la Figure 2.7 sont possibles. En effet, l'architecture à base de mémoire peut être obtenue en utilisant un Radix-2 à une sortie ou à 2 sorties à la place du PE Radix-4. Une étude parue dans [54] a détaillé les différentes possibilités. Les deux architectures

les plus rapides que nous pouvons utiliser sont RX2-B4 et MXRX-B4[1]. L'architecture RX2-B4 est basée sur l'algorithme Radix-2 et fournit 4 sorties en parallèle. Cependant, l'architecture MXRX-B4 est basée sur l'algorithme Split Radix et fournit 4 sorties en parallèle.

Cette architecture est intéressante en termes de surface occupée. En effet, le nombre d'opérateurs ainsi requis par cette architecture est celui du bloc FFT utilisé, ie, 3 multiplieurs et 8 additionneurs complexes dans le cas d'un PE Radix-4. Cependant, elle souffre d'une latence très importante. En effet, si le PE utilisé est celui de l'algorithme Radix-4, la latence L est donnée par :

$$L = \frac{N}{4}[(2T_A + T_M)(log_4(N) - 1) + 2T_A] \tag{2.23}$$

2.4.3 Architecture pipeline

L'architecture "pipeline" représente un bon compromis entre le temps d'exécution et la complexité matérielle. La Figure 2.8 illustre le principe de fonctionnement des architectures basées sur cette méthode.

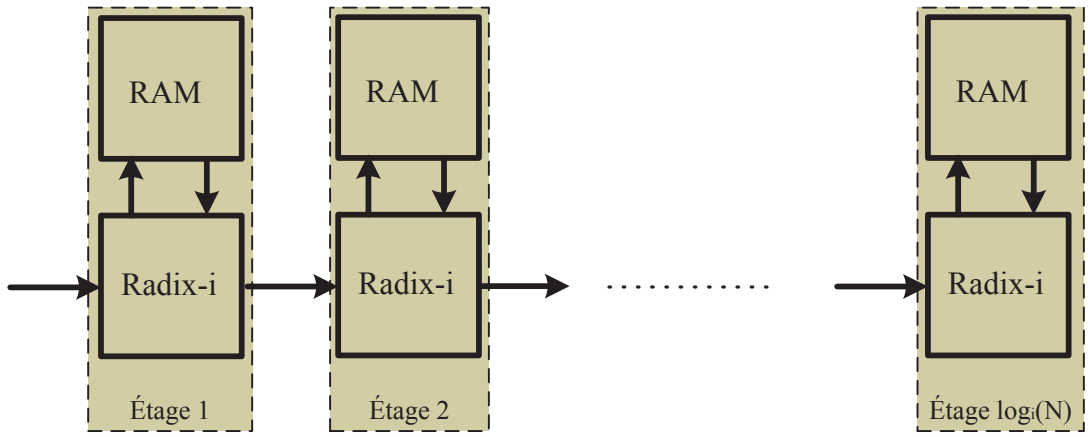


FIGURE 2.8 – Architecture pipeline de la FFT

Cette architecture consiste à implanter $s = log_i(N)$ étages. Chaque étage est constitué d'un bloc Radix-i points (i est une puissance de 2) et des blocs mémoires. Les mémoires sont utilisées pour sauvegarder les résultats de chaque étage afin de les exploiter par l'étage suivant. Cette architecture représente une solution intéressante pour l'implantation des FFT sur FGPA. Cependant, le nombre important de mémoires utilisées (une par étage) constitue l'inconvénient majeur de cette architecture.

Concernant les architectures "pipelines", plusieurs travaux ont présenté des optimisations afin d'avoir de meilleurs performances en termes de rapidité et surface occupée. Les architectures les plus connus qui ont été proposé sont R2MDC "Radix-2 Multi-path Delay Commutator" [55], R2SDF "Radix-2 Single-Path Delay Feedback" [56], R4SDF "Radix-4 Single-Path Delay Feedback" [57], R2^2SDF "Radix-2^2 Single-Path Delay Feedback" [58]. La différence entre ces architectures réside dans le nombre des entrées/sorties et dans l'algorithme utilisé. Dans [59], une implantation matérielle de deux architectures pipelines (R2^2SDF et R4SDF) a été proposée. Nous utiliserons par la suite les résultats de ces implantations pour des comparaisons.

2.4.4 Etude de cas : IP FFT de Xilinx

Dans ce paragraphe, nous proposons d'étudier brièvement l'architecture de l'IP Xilinx [60]. Cette IP permet de calculer des FFT, directes ou inverses, de taille N pouvant aller de 2^3à 2^{16} points. Pour effectuer une FFT directe ou une FFT inverse, le seul paramètre qui différencie l'utilisation de cette IP est respectivement la mise à 1 ou à 0 du signal fwd_inv (Figure 2.9).

1. Le préfixe RX représente le Radix utilisé et Bi indique le nombre de sorties en parallèle.

Les données d'entrée peuvent être fournies sous la forme d'un vecteur de valeurs complexes représenté par des nombres binaires comprise entre 8 et 24 bits. De même, les facteurs de rotation, peuvent être représentés sur 8 à 24 bits. Plusieurs paramètres sont configurables dont la taille du bloc FFT. En effet une interface graphique permet de paramétrer le bloc FFT/IFFT afin de choisir le nombre de points, le nombre de bits du signal d'entrée, le type de calcul avec arrondi ou troncature en gardant la pleine échelle ou pas. Enfin, en fonction de la valeur d'entrée *order*, les sorties peuvent être fournies dans un ordre naturel et inversé. Dans le cas du mode sorties en ordre naturel, une mémoire supplémentaire est requise. La Figure 2.9 [60] illustre un schéma simplifié des différentes entrées et sorties de l'IP Xilinx. Il y a bien entendu des entrées sorties additionnelles qui offrent plus d'options pour l'utilisation de cet IP.

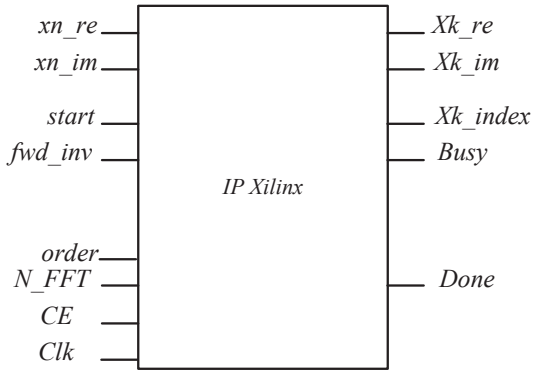


FIGURE 2.9 – Schéma simplifié de l'IP Xilinx

Les entrées xn_re et xn_im sont les données réelles et complexes du signal d'entrée. L'entrée N_FFT permet de préciser la taille de la FFT à réaliser, l'entrée CE est utilisé pour activer ou désactiver le fonctionnement de l'IP et l'entrée *clk* correspond à l'horloge du système. D'autre part, les sorties Xk_re et Xk_im représente les sorties de la FFT, Xk_index représente l'indice correspondant à chaque sortie. La sortie $Busy$ est utilisée pour indiquer si l'IP est en cours d'exécution ou pas. Finalement la sortie $Done$ permet d'indiquer la fin du traitement.

D'autre part, l'IP Xilinx propose 4 types d'architectures dont 3 architectures à base de mémoire et une architecture pipeline. Les architectures à base de mémoires proposées par Xilinx sont connues sous les noms Radix-4 Burst I/O, Radix-2 Burst I/O et Radix-2 Lite Burst I/O. Par contre l'architecture pipeline est appelée Pipelined Streaming I/O. Ces architectures utilisent les algorithme Radix-2 et Radix-4 avec une décimation dans le temps pour les architectures à base de mémoire et une décimation en fréquence pour l'architecture pipeline.

L'architectue pipeline permet un traitement continu des données en offrant ainsi un haut débit et une rapidité de calcul meilleure que celle obtenue avec des architectures à base de mémoires. En effet, ces dernières chargent et traitent les données séparément en utilisant une approche itérative. De plus, la surface occupée de l'architecture à base de mémoires est réduite mais le temps d'exécution est important. Une comparaison en termes de ressources consommées ainsi que de débit de sortie a été effectuée et illustrée dans la Figure 2.10. Il est montré que pour avoir plus de débit il faut utiliser plus de ressources matérielles. Les performances quantifiées de cette IP seront détaillées dans la section 2.6.

D'un point de vue architecturale, le schéma bloc de la réalisation pipeline proposée par Xilinx est identique à celui de la Figure 2.8 en remplaçant Radix-i par Radix-2 et $log_i(N)$ par $log_2(N)$. Cette architecture utilise donc plusieurs Butterfly Radix-2 afin d'assurer un traitement continu des données. Chaque bloc Radix-2 est lié à une mémoire pour sauvegarder les entrées et les données intermédiaires. Cette IP a la possibilité de calculer un flux de données tout en sauvegardant les entrées du flux suivant et en fournissant les sorties du flux précédent. Il est ainsi possible de charger des données de façon continue et après une latence de traitement obtenir des résultats en continu. Il est aussi possible de traiter un seul flux de données ou bien des flux de données séparés dans le temps.

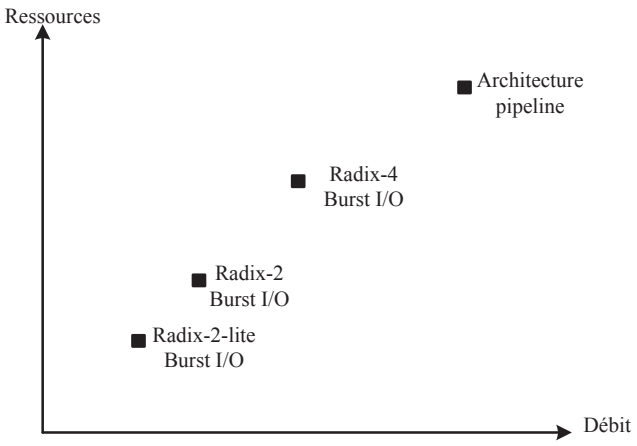


FIGURE 2.10 – Comparaison entres les architectures de Xilinx

2.4.5 Discussion

Après avoir détaillé le principe de fonctionnement des différentes architectures d'implantation matérielle de la FFT, nous proposons ici de comparer les différentes architectures en termes de nombre d'opérateurs et de latence afin de sélectionner l'architecture adéquate à une implantation optimisée de la FFT. Le tableau 2.1 illustre cette comparaison.

	Multiplieurs	Additionneurs	Latence
Réalisation directe	$\frac{3N}{4}log(N)$	$\frac{8N}{4}log(N)$	$logN(2T_A + T_M)$
Architecture récursive	3	8	$NlogN(2T_A + T_M)$
Architecture pipeline	$3log(N)$	$8log(N)$	$N(2T_A + T_M)$

Tableau 2.1 – Comparaison entre les différentes architectures

La latence présentée dans le tableau 2.1 représente une valeur approximative du temps écoulé entre l'arrivée de la première donnée et l'obtention de la première sortie. Il est montré que l'architecture récursive présente une latence N fois plus importante que celle de la réalisation directe. Ceci est inefficace pour les applications temps réel. Quant à l'architecture pipeline, il semble qu'elle présente le meilleur compromis en termes de surface/rapidité (la surface est exprimé en fonction du nombre d'additions et de multiplication et le temps est exprimé en fonction de la latence). La confirmation de ce choix sera donnée dans la section 2.6.

2.5 Proposition d'architectures optimisées de la FFT

2.5.1 Architecture à entrée série / sortie série

2.5.1.1 Principe

Comme nous l'avons détaillé précédemment, l'inconvénient majeur des architectures pipelines est le nombre de mémoires utilisées. Ce nombre peut être très contraignant dans le cas où N est grand. Pour remédier à ce problème nous proposons une architecture pipeline qui consiste à utiliser un PE par étage et une seule mémoire partagée par tous les blocs.

L'architecture proposée est présentée sur la Figure 2.11. Le calcul de la FFT est divisé en 4 étages pour $N = 256$. Chaque étage est composé d'un bloc Radix-4. La dépendance des données entre les différents étages exige la sauvegarde des sorties après chaque bloc multiplieur. Pour cette raison, une mémoire est nécessaire à chaque étage pour sauvegarder les sorties de chaque étage. De plus, afin de réduire le nombre de mémoires requises nous proposons d'utiliser une seule mémoire qui sera partagée par les différents étages. Enfin, pour réduire le nombre d'accès simultané à cette mémoire, nous avons modifié l'architecture de l'algorithme Radix-4 afin d'avoir

une entrée et une sortie. Par conséquent un seul multiplieur complexe sera utilisé par étage au lieu de 3.

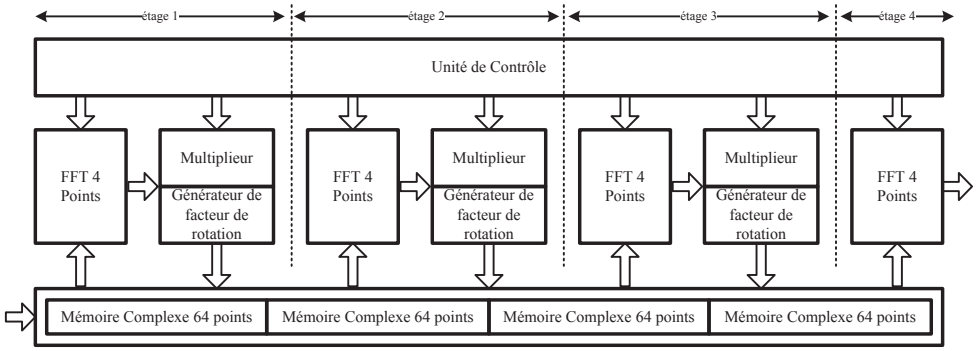


FIGURE 2.11 – Schéma bloc de l'architecture proposée à entrée série/sortie série

2.5.1.2 Modification du PE du Radix-4

L'architecture du PE de l'algorithme Radix-4 nécessite un accès multiple aux mémoires et a besoin de 3 multiplieurs complexes par étage. En effet, cette architecture utilise 4 entrées et fournit 4 sorties par cycle d'horloge. Afin d'éviter cet accès multiple et pour réduire le nombre de multiplieurs complexes par étage, nous proposons une nouvelle architecture de l'algorithme Radix-4. Comme il est montré sur la Figure 2.12, l'architecture proposée, a la particularité d'avoir une entrée et de fournir une sortie par étage. Ainsi nous éliminons l'accès multiple aux mémoires et nous réduisons le nombre de multiplieurs complexes en utilisant 1 seul multiplieur. Un diagramme temporel présenté sur la Figure 2.13 explique le fonctionnement de l'architecture modifiée. Des signaux intermédiaires A, B, C et D sont utilisés pour garder l'aspect pipeline de l'architecture.

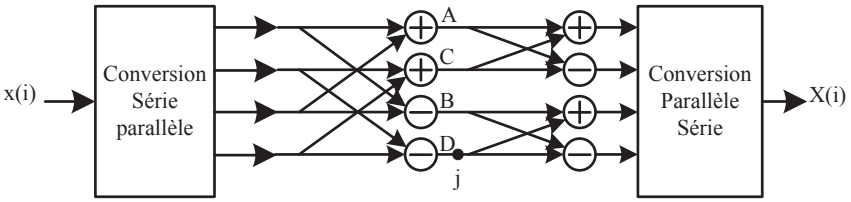


FIGURE 2.12 – Schéma bloc de l'architecture modifiée du PE Radix-4

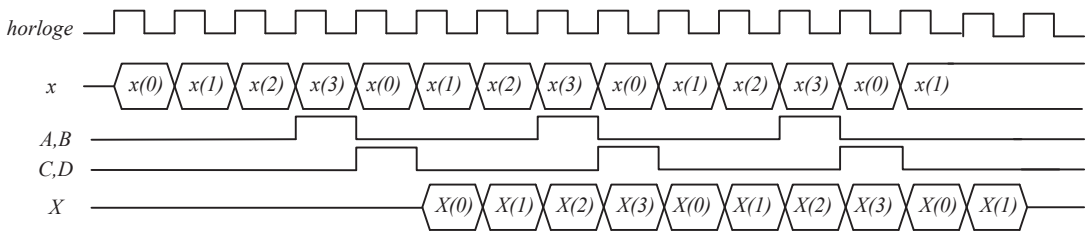


FIGURE 2.13 – Diagramme temporel de l'architecture modifiée du PE Radix-4

Comme il est montré sur la Figure 2.13, les entrées $x(i)$ sont présentées en série et une conversion série/parallèle est nécessaire. Après 2 cycles d'horloge, les signaux intermédiaires $A = x(0) + x(2)$ et $B = x(0) - x(2)$ peuvent être calculés et le résultat sera prêt un cycle d'horloge plus tard. De la même manière $C = x(1) + x(3)$ et $D = x(1) - x(3)$ sont prêts après 4 cycles d'horloge. Ainsi, la première sortie $X(0) = A + C$ est prête après 5 coups d'horloge et de la même manière les autres sorties sont calculées et les sorties sont fournies d'une manière séquentielle et continue.

2.5.1.3 Partage de la mémoire

Comme il est indiqué dans la Figure 2.11 à chaque sortie du PE Radix-4, un générateur de facteurs de rotation permet de générer la phase de rotation correspondante ($W_N^n = e^{\frac{-2j\pi n}{N}}$). Un multiplieur complexe permet de calculer la multiplication complexe. Le résultat obtenu sera sauvegardé dans une mémoire de taille N points. Cette dernière sera réutilisée et partagée entre les différents étages. Comme nous l'avons cité précédemment, l'implantation de la FFT de taille N points nécessite $log_4(N)$ mémoires de taille N points. Dans l'architecture que nous proposons, nous utiliserons une seule mémoire de taille N points. De plus, afin d'améliorer la latence, la mémoire utilisée sera divisée en 4 mémoires de taille $\frac{N}{4}$ points chacune. Cette division se fait conformément aux indices des entrées/sorties de chaque bloc. L'intérêt de cette division consiste à améliorer la latence en permettant l'accès simultané aux données pour l'écriture des sorties et la lecture des entrées par deux étages successifs. Cette architecture a été implantée sur un FPGA de la famille Xilinx. Elle présente l'avantage d'une utilisation efficace des ressources. Malgré cette amélioration, cette architecture souffre d'une latence assez élevée et un débit réduit vue qu'elle fournit une seule sortie par étage. Afin d'améliorer davantage la rapidité et le débit de cette architecture, nous proposons une nouvelle architecture de calcul de FFT.

2.5.2 Architecture à entrée parallèle / sortie parallèle

Cette architecture utilise l'algorithme Radix-4 à 4 entrées et 4 sorties afin d'avoir un meilleur débit. L'utilisation de cet algorithme peut causer des problèmes au niveau de l'accès mémoire. Pour remédier à ce problème nous proposons une nouvelle technique de division et de partage de mémoire.

2.5.2.1 Calcul à base d'éléments de mémorisation

Il s'agit d'une architecture pipeline à 4 entrées et 4 sorties. Le schéma bloc est illustré dans la Figure 2.14. Il est à noter que pour simplifier l'explication du principe de partage de mémoires, nous utilisons plusieurs PE par étage. Cependant, ceci ne reflète pas l'architecture proposée qui contient uniquement un PE par étage qui sera utilisé $\frac{N}{4}$ fois.

Pour une FFT de taille N points, les entrées sont sauvegardées dans une mémoire d'entrée de taille N points appelée Mem_1. Chaque étage contient $\frac{N}{4}$ PE Radix-4. Considérons bi_1 avec $i \in [0, \frac{N}{4}\text{-}1]$, l'indice du PE de l'étage. Les adresses des entrées à bi_1 sont i_1, $i_1 + \frac{N}{4}$, $i_1 + \frac{2N}{4}$, $i_1 + \frac{3N}{4}$. Les sorties de ce PE seront sauvegardées avec les mêmes adresses dans une deuxième mémoire de taille N points complexes Mem_2.

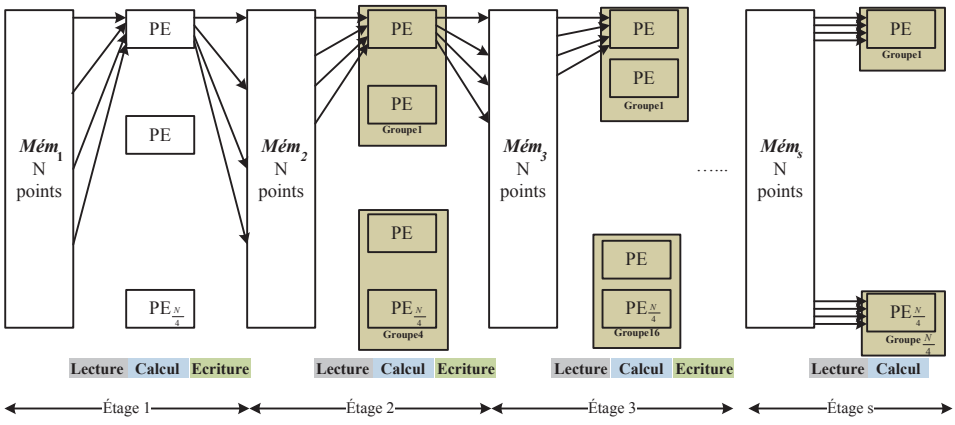


FIGURE 2.14 – Schéma bloc de la FFT N points

Le deuxième étage est aussi composé de $\frac{N}{4}$ PE. Les entrées à cet étage sont sauvegardées dans Mem_2. Les PE de cet étage sont divisés en 4 groupes de $\frac{N}{16}$ PE chacun en fonction des indices de leurs entrées. En effet, les $\frac{N}{4}$ premières données de la Mem_2 seront utilisées par le

premier groupe de PE. Si nous considérons bi_2 l'indice d'un PE du premier groupe avec $i_2 \in [0, \frac{N}{16}\text{-}1]$. Les indices d'entrée à ce PE sont i_2, $i_2 + \frac{N}{16}$, $i_2 + \frac{2N}{16}$ et $i_2 + \frac{3N}{16}$. Quant aux indices d'un PE du deuxième groupe sont $i_2 + \frac{N}{4}$, $i_2 + \frac{N}{16} + \frac{N}{4}$, $i_2 + \frac{2N}{16} + \frac{N}{4}$ et $i_2 + \frac{3N}{16} + \frac{N}{4}$. De la même manière, les indices des PE du troisième et quatrième groupe sont décalés d'un facteur de $\frac{2N}{4}$ et $\frac{3N}{4}$ respectivement par rapport à ceux du premier groupe. Les sorties de chaque PE sont sauvegardées dans une mémoire de taille N points complexes avec les mêmes adresses utilisées pour la lecture.

Le troisième étage est composé aussi de $\frac{N}{4}$ PE. Les PE de cet étage quant à eux sont divisés en 16 groupes en fonction des indices de leurs entrées. Cette division se poursuit jusqu'à avoir $\frac{N}{4}$ groupes. Ce principe est répété jusqu'au dernier étage. Cependant cet étage est traité différemment. En effet, le PE utilisé dans cet étage ne contient pas de multiplieurs et les mémoires utilisées sont toutes de taille 1 point (nous utilisons ainsi des registres à la place des mémoires).

2.5.2.2 Technique de division et de partage des mémoires

Afin de réduire la taille et le nombre de mémoires, nous proposons d'utiliser conjointement deux techniques :

Calcul "in place" du premier étage

Une première optimisation que nous proposons consiste à utiliser une seule mémoire de taille N points complexes qui sera partagée entre le premier et le second étage, ie, Mem_1 et Mem_2 sont la même mémoire. En effet, les données arrivent dans un ordre naturel et afin de commencer le traitement du premier PE, il est nécessaire que les $\frac{3N}{4}$ premières données soient présentes. Les adresses de lecture de la mémoire Mem_1 et d'écriture dans la mémoire Mem_2 sont les mêmes. De plus, pour une case mémoire donnée, l'instant d'écriture et celui de lecture sont différents. Nous proposons ainsi d'utiliser une seule mémoire Mem_{1-2} de taille N points qui sera partagée entre ces deux étages. La Figure 2.15 illustre le principe de fonctionnement de ce partage.

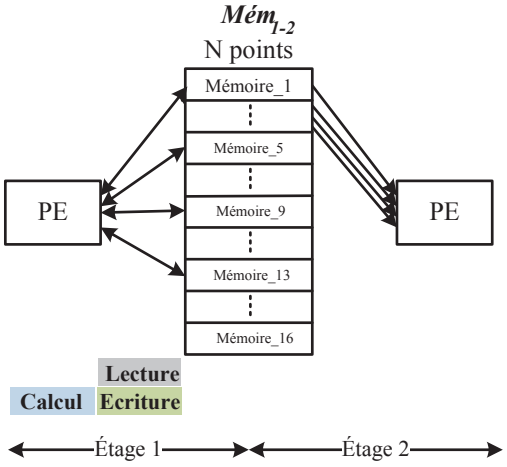


FIGURE 2.15 – Principe de partage de mémoire

Division et partage des mémoires

Les mémoires embarquées sont des mémoires à un seul ou deux ports d'écritures au maximum. Cependant, comme nous utilisons l'algorithme Radix-4, il est nécessaire d'effectuer 4 accès simultanés à la mémoire. Par conséquent, les mémoires utilisées doivent être divisées en 4 mémoires. De plus, dans la partie précédente, nous avons montré qu'entre deux étages successifs, il existe un facteur de 4 entre les adresses d'écriture et celle de lecture. Par exemple pour la mémoire Mem_1, les 4 premières sorties du premier PE doivent être sauvegardées avec les mêmes adresses de lecture de Mem_2, ie, 0, $\frac{N}{4}$, $\frac{2N}{4}$, $\frac{3N}{4}$. Ainsi cette mémoire doit être divisée en 4 parties

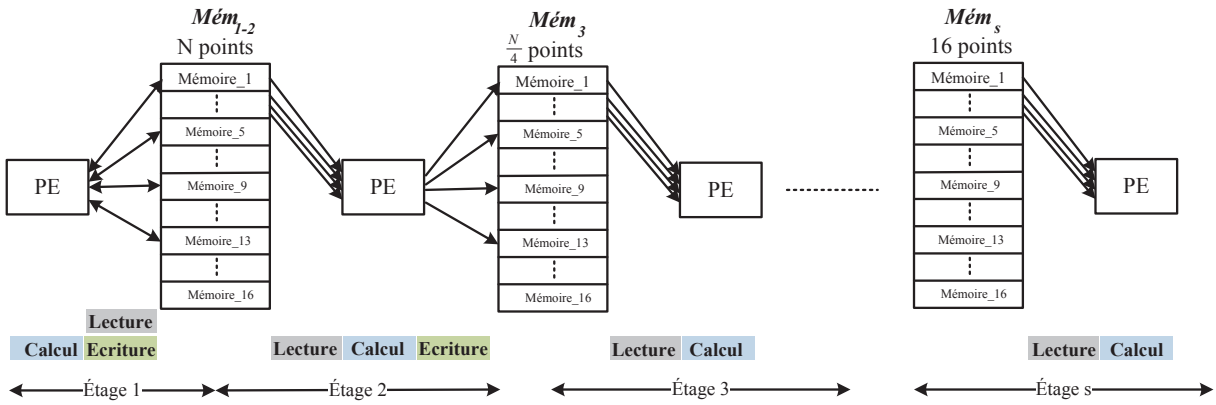


FIGURE 2.16 – Schéma bloc de l'architecture proposée

de taille $\frac{N}{4}$ afin d'effectuer 4 accès simultanés. D'autre part, les 4 premiers indices des entrées du premier PE du deuxième étage sont 0, $\frac{N}{16}$, $\frac{2N}{16}$, $\frac{3N}{16}$. Ces données sont toutes sauvegardées dans la première mémoire de taille $\frac{N}{4}$. Il est ainsi nécessaire de diviser cette mémoire en 4 mémoires de taille $\frac{N}{16}$. De la même manière, nous divisons les mémoires de chaque étage en 16 mémoires.

D'un autre côte et comme nous l'avons indiqué précédemment, les entrées à chaque groupe de PE sont indépendantes. Il est ainsi possible d'utiliser une seule mémoire permettant de sauvegarder uniquement les données pour chaque groupe de PE. Cette mémoire sera partagée entre les différents groupes grâce à l'indépendance des données entre les groupes. Par conséquent, nous pouvons utiliser une mémoire de taille $\frac{N}{4}$ divisée en 16 mémoires de taille $\frac{N}{64}$ chacune pour la mémoire Mem_3. De la même manière, la taille de la mémoire du quatrième étage est de $\frac{N}{16}$ points et celle du dernier étage est de 16 points.

Gestion des calculs par une unité de contrôle

En se basant sur les optimisations décrites ci-dessus, nous présentons ainsi une architecture optimisée pour une implantation matérielle de la FFT illustrée sur la Figure 2.16. En effet, l'architecture proposée contient un PE Radix-4 et une mémoire par étage. Contrairement aux architectures pipelines présentées dans la Figure 2.8 qui utilisent une mémoire de taille N points par étage, la taille des mémoires utilisées dans cette architecture est divisée par 4 après chaque étage. Le schéma bloc du PE utilisé, basé sur l'algorithme Radix-4, est illustré dans la Figure 2.3.

L'architecture du PE du dernier étage est différente à celle des autres étages. En effet, le PE utilisé dans le dernier étage est une FFT de taille 4 points. Cette FFT est composée uniquement d'additionneurs et de soustracteurs.

La Figure 2.17 représente le schéma bloc de l'architecture proposée pour N=256 points. Cette architecture est composée de 4 étages. Le premier et le deuxième étage partagent une mémoire de taille de 256 points. Chaque étage est composé d'un PE composé de 8 additionneurs/soustracteurs et 3 multiplieurs complexes. Le dernier étage est composé uniquement d'une FFT 4 points composée de 8 additionneurs/soustracteurs. Le fonctionnement de cette architecture est contrôlé par une unité de contrôle globale permettant la gestion des entrées et des sorties à chaque PE et d'une unité de contrôle locale qui gère les constantes de multiplication.

2.5.2.3 Analyse temporelle

Dans cette section, nous présentons une analyse de la latence introduite par l'architecture parallèle proposée dans la Figure 2.17. Il s'agit de la même architecture présentée sur la Figure 2.16 mais le niveau de détail est différent. Rappelons que, le temps d'exécution d'un PE Radix-4 est égal à $2T_A+T_M$. Cependant le temps d'exécution du PE Radix-4 de dernier étage est de $2T_A$. Par conséquent le temps d'exécution T_{Trait} de tous les blocs pour une FFT de taille N points

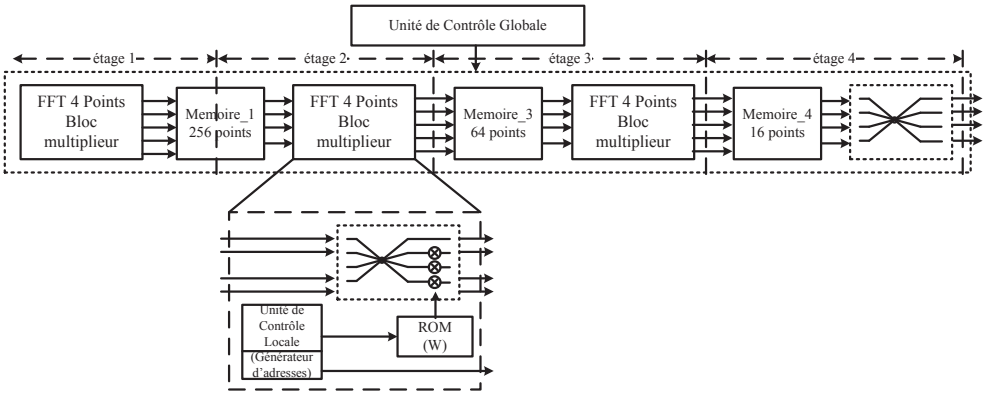


FIGURE 2.17 – Architecture FFT 256 points

est de

$$T_{Trait} = (log_4(N) - 1)(2T_A + T_M) + 2T_A \tag{2.24}$$

Si nous considérons T_{Mem} le temps d'un accès mémoire. Le nombre d'accès mémoire peut être exprimé par l'équation 2.25.

$$T_{Mem} = 3\frac{N}{16} + 3\frac{N}{64} + .. + 3\frac{N}{N} = N - 3\frac{N}{4} = \frac{N}{4} \tag{2.25}$$

Par conséquent la latence relative L qui correspond au temps écoulé entre le début de calcul et la présence de la première sortie est donnée par :

$$L = \frac{N}{4}T_{Mem} + (log_4(N) - 1)(2T_A + T_M) + 2T_A. \tag{2.26}$$

2.5.3 Complexité algorithmique

Avant de comparer les performances des architectures existantes avec celle que nous avons proposées, nous commençons par évaluer l'utilisation du nombre d'opérateurs arithmétiques et les tailles des mémoires utilisées. Le nombre de multiplieurs complexes pour les architectures à entrées parallèles et sorties parallèles est trois fois plus important que celui des architectures à entrées séries-sorties séries.

Les tableaux 2.2 et 2.3 montrent le compromis réalisé par plusieurs architectures en termes de rapidité/surface. En effet, la surface est mesurée en fonction du nombre des multiplieurs complexes, des additionneurs complexes ainsi que la taille totale des mémoires utilisées. Quant à la rapidité, elle est exprimée en fonction du débit et de la latence. Les architectures présentées ont différentes structures. En effet, les architectures [56–58,61,62], ont une structure entrée série, sortie série et les architectures [55,63] ont une architectures entrées parallèles, sorties parallèles.

Réferences	Structure	Multiplieurs Complexes	Aadditionneurs Complexes	Taille mémoire	Latence Absolue
R2SDF [56]	Radix-2	log_2N-2	$2log_2$N	N-1	N
R2^2SDF [58]	Radix-2	log_4N-1	$4log_4$N	N-1	N
R2^2SDF [61]	Radix-2	log_4N-1	log_2N	2(N-1)	N
FB_Radix-2 [64]	Radix-2	log_4N-1	$2log_4$N	$\frac{4}{3}$N	N
R4SDF [62]	Radix-4	log_4N-1	$8log_4$N	N-1	N
R4SDF [57]	Radix-4	log_4N-1	$3log_4$N	2(N-1)	N
Entrée série Sortie série	Radix-4	log_4N-1	$8log_4$N	N	N

Tableau 2.2 – Complexité algorithmique des architectures pipeline à sortie séries

Réferences	Structure	Multiplieurs Complexes	Aadditionneurs Complexes	Taille mémoire	débit	Latence Absolue
R2MDC [55]	Radix-2	log_2N-2	$2log_2$N	$\frac{3}{2}$ N-2	2	N
FF_Radix-2 [63]	Radix-2	2(log_4N-2)	$2log_2$N	4 N	2	N
R4MDC [55]	Radix-4	3(log_4N-1)	$8log_4$N	$\frac{5}{2}$N-2	4	N
FF_Radix-4 [63]	Radix-4	3(log_4N-1)	$8log_4$N	$\frac{8}{3}$N	4	$\frac{N}{3}$
Entrée parallèle/ Sortie parallèle	Radix-4	3(log_4N-1)	$8log_4$N	$\frac{4}{3}$N	4	$\frac{N}{4}$

Tableau 2.3 – Complexité algorithmique des architectures pipeline à sortie parallèle

Comparée aux architectures parallèles proposées dans la littérature, l'architecture proposée sur la Figure 2.16 présente le même nombre d'opérateurs ($3(log_N - 1)$ de multiplieurs et $8(log_N)$ d'additionneurs). Cependant, l'architecture proposée permet de calculer les sorties plus rapidement. En effet, la latence est de $\frac{N}{4}$ alors qu'elle vaut $\frac{N}{3}$ et N respectivement pour [63] et [55]. Cette latence absolue est exprimée en négligeant le terme additif en *log* pour toutes les architectures. De plus, la taille mémoire de l'architecture proposée est presque la moitié de celle des architectures parallèles avec le même nombre de multiplieurs et additionneurs complexes.

2.6 Résultats d'implantation matérielle

2.6.1 Complexité matérielle des différentes familles d'architectures

Dans cette section nous décrivons la complexité matérielle des différentes architectures de la FFT détaillées dans la section 2.4. Le critère de base utilisé pour la comparaison est le produit surface-temps. Les complexités matérielle et temporelle des architectures proposées ainsi que celle des autres architectures sont illustrées sur le tableau 2.4. Toutes les architectures présentées dans le tableau sont développées en utilisant le langage de description matérielle VHDL et synthétisées en utilisant l'outil ISE de Xilinx et une carte de développement contenant un FPGA de la famille Spartan-3E.

Nous utilisons le délai de propagation et le temps d'exécution comme critères pour quantifier la complexité temporelle. Le temps d'exécution est obtenu en multipliant le délai de propagation par le nombre de cycles d'horloge nécessaire pour avoir toutes les sorties . Le délai de propagation est défini par la durée d'un cycle d'horloge.

L'architecture présentée sur la Figure B.1 représente la réalisation directe d'une FFT de taille 64 points en utilisant l'algorithme Radix-4. Cette structure présente le temps d'exécution le plus réduit mais une surface occupée très importante. D'autre part, pour l'architecture récursive, le temps d'exécution est le plus important et la surface consommée est la plus réduite. Concernant les deux structures proposées, elles ont le meilleur délai de propagation. Notons aussi que l'architecture à entrées parallèles et sorties parallèles est 2 fois plus rapide que celle à entrées séries et sorties séries. Enfin, en termes de produit surface, il est montré que l'architecture à sorties parallèles est la plus performante. En effet, le produite surface temps est un paramètre de mesure de performance connu aussi sous le nom "area delay product". Il permet de donner une idée combinée de la rapidité et de l'occupation de surface.

2.6.2 Complexité matérielle des architectures pipelines

Afin de montrer l'efficacité de nos architectures proposées, une comparaison des résultats de synthèse obtenus en utilisant une carte de développement embarquant un FPGA de la famille Spartan 3 avec plusieurs architectures a été effectuée. Les tableaux 2.5 et 2.6 illustrent une comparaison avec les architectures pipelines et à base de mémoire présentées dans la littérature ainsi que l'IP de Xilinx. Les performances des différentes architectures sont exprimées en termes de surface (nombre de Slices Luts), de fréquence maximale de fonctionnement (MOF "Maximum Operating Frequency"), de débit (Méga sorties/sec) qui évalue le nombre de sorties par seconde,

Architecture	Slices Luts	Délai de propagation(ns)	Temps d'éxecution(μs)	Produit Surface.temps (Slice.μs)
Réalisation directe Figure B.1	43623	15.3	**0.15**	6543.4
Architecture récursive Figure 2.7 [54]	2281	30.3	8.3	18932.3
Architecture proposée entrée série/ sortie série	**1155**	9.4	2.03	2344.6
Architecture proposée entrée parallèle/ sortie parallèle	2345	**8.4**	0.8	**1876**

Tableau 2.4 – Complexité matérielle des différentes familles d'architectures

du temps d'exécution et le produit surface-temps. Dans certaines réalisations le circuit généré peut contenir des multiplieurs embarqués, des BRAM ou des DSP Slices. Afin de comparer plusieurs architectures nous sommes amenés à déterminer la surface occupée exclusivement en fonction du nombre de Slice. Par conséquent, nous avons le choix entre 2 méthodes. Nous pouvons synthétiser le circuit en utilisant uniquement des Slices Luts. Ou bien, nous calculons le nombre de Slices Luts nécessaire pour une BRAM ou un multiplieur ou un DSP Slice et nous l'additionnons au nombre de Slices Luts requis. Pour des raisons de simplicité nous avons choisi la dernière méthode.

Architecture	Structure	Slices Luts	MOF (MHz)	Débit (MS/s)	Temps d'exécution(s)	Produit Surface Temps (Slice.μs)
R4SDC [59]	Radix-4	2662	107	107	1.2	3194
R2^{2}SDF [59]	Radix-2	2520	98	98	1.4	3528
Xilinx IP [65]	Radix-2	1477	**152**	152	1.68	2481
Entrée série sortie série	Radix-4	**1155**	106	106	2.03	2344
Entrée parallèle Sortie parallèle	Radix-4	2345	118	**472**	**0.8**	**1876**

Tableau 2.5 – Résultats de synthèse pour une FFT de taille 64 points

Architecture	Structure	Slices Luts	MOF (MHz)	Débit (MS/s)	Temps d'exécution(s)	Produit Surface Temps (Slice.μs)
R4SDC [59]	Radix-4	4555	111	111	4.6	20953
R2^{2}SDF [59]	Radix-2	3868	98	98	5.36	20732
Xilinx IP [65]	Radix-2	2455	**148**	148	7.6	18854
Entrée série Sortie série	Radix-4	**1924**	91	91	10.1	19432
Entrée parallèle Sortie parallèle	Radix-4	4325	103	**412**	**3.4**	**18785**

Tableau 2.6 – Résultats de synthèse pour une FFT de taille 256 points

En se référant aux tableaux 2.5 et 2.6. Il est montré que le temps d'exécution de l'architecture proposée à entrée parallèle /sortie parallèle est 56%, 36%, et 26% inférieur à celui de l'IP de Xilinx, l'architecture R2^{2}SDC et R4SDC respectivement pour une FFT de taille 256 points. En ce qui concerne le produit surface-temps, l'architecture proposée permet de le réduire à hauteur

de 11% et 10% par rapport à celui de l'architecture R4SDC et R2^{2}SDF pour une FFT de taille 256 points.

D'autre part une comparaison en fonction du rapport débit par surface entre les architectures proposées et celles présentées dans les tableaux 2.5 et 2.6 pour des FFT de taille 64 et 256 points est illustrée sur la Figure 2.18. Le rapport débit par surface utilisé représente le débit produit par chaque Slice Lut consommée. La Figure montré l'efficacité des architectures proposées. En effet, l'architecture proposée à entrée / sortie parallèles (Figure 2.17) présente une augmentation de 26% et 19% en termes du facteur débit par surface comparé respectivement à l'architecture série proposée 2.11 et par rapport à celle de l'IP de Xilinx pour une FFT de taille 256 points.

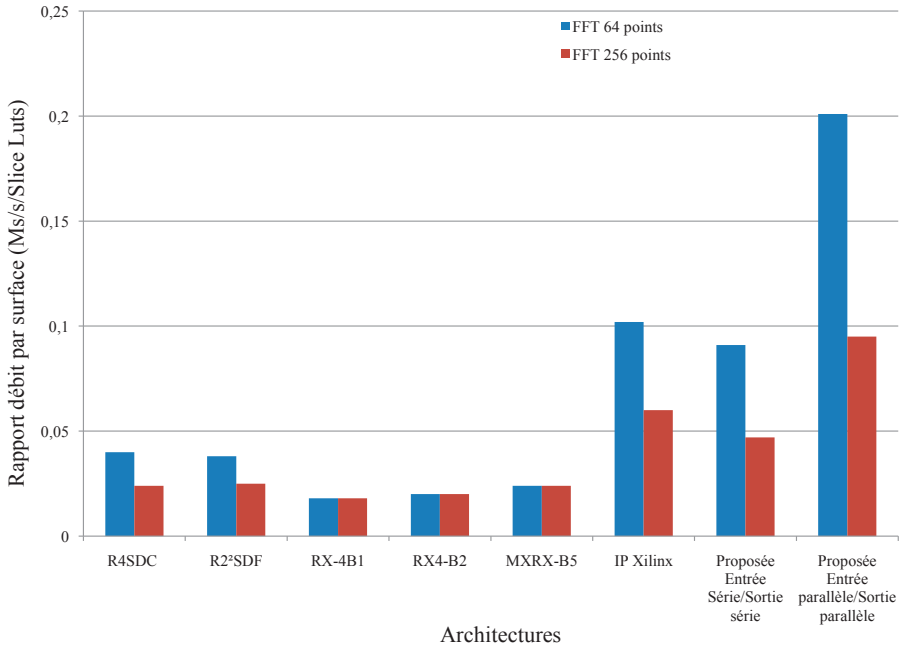


FIGURE 2.18 – Comparaison en termes de débit surface

2.6.3 Comparaison avec l'IP Xilinx

Après avoir comparé les architectures proposées avec les architectures existantes, nous avons montré que deux architectures se dégagent : IP Xilinx et l'architecture parallèle proposée. Nous proposons ici une comparaison plus approfondie entre l'architecture pipeline de l'IP de Xilinx et celle proposée. Deux paramètres sont évalués : le produit surface temps et le rapport débit par surface. Ces résultats sont obtenus après synthèse sur un FPGA de type Virtex-5. La Figure 2.19 montre la comparaison en termes de rapport débit par surface entre l'architecture proposée et celle de Xilinx pour de tailles de FFT allant de 16 points 4 à 16k points.

Il est ainsi montré que l'architecture proposée représente un rapport de débit par surface meilleur que celui de l'IP de Xilinx pour toutes les tailles de la FFT. Ce rapport peut atteindre 5 pour N=256 points.

D'autre part, une comparaison en termes de produit surface temps avec l'IP de Xilinx pour les mêmes tailles de FFT a été effectué. Les résultats de cette comparaison sont illustrés sur la Figure 2.20.

Les résultats ainsi obtenus montrent l'avantage de l'architecture proposée par rapport à celle de l'IP de Xilinx en termes de produit surface temps. En effet, malgré que l'architecture de Xilinx a une surface plus réduite et une fréquence de fonctionnement meilleure que l'architecture proposée, cette dernière est deux fois plus rapide grâce la faible latence assurée par la technique

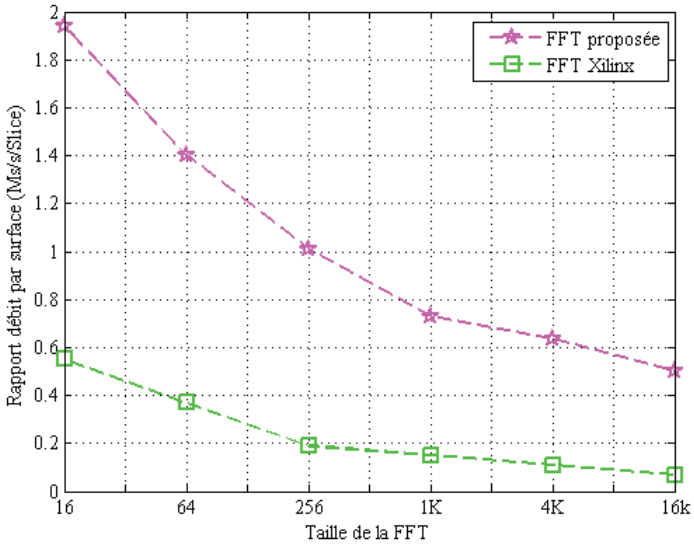
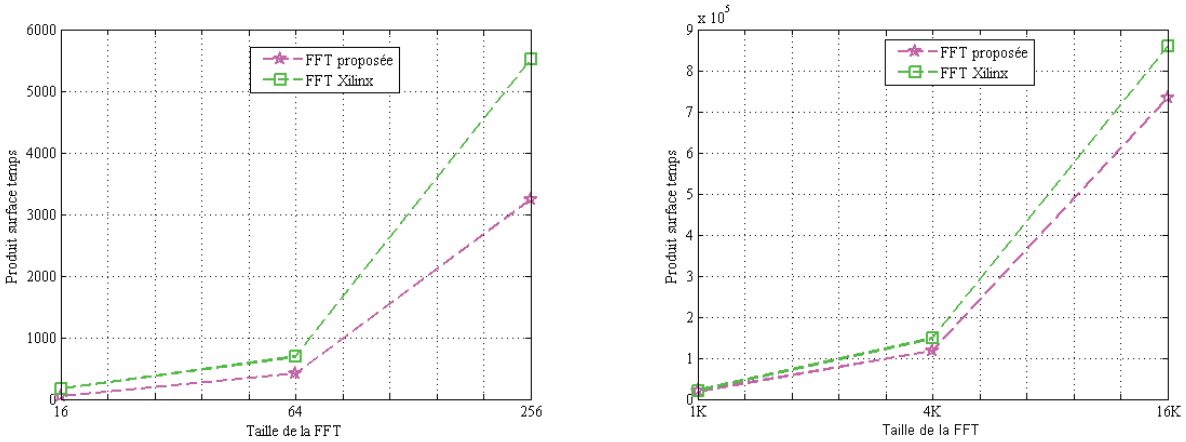


FIGURE 2.19 – Comparaison avec l'IP de Xilinx en terme de rapport débit par surface



(a) (b)

FIGURE 2.20 – Comparaison avec l'IP Xilinx en terme de produit surface temps

de division des mémoires. De plus, la technique de partage de mémoire a permis de réduire la surface occupée ce qui a contribué à avoir un produit surface rapidité meilleur. Enfin, le nombre de sorties par cycle d'horloge de l'architecture proposée a permis d'avoir un débit 4 fois meilleur que celui de Xilinx. Ce débit important a permis ainsi d'avoir un rapport débit par surface meilleur.

2.6.4 Rapport Signal à Bruit de Quantification

Les entrées des architectures proposées sont codées en virgule fixe afin de rendre l'implantation de la FFT possible. Cependant, ce choix exige d'utiliser la troncature afin de maintenir la dynamique des signaux intermédiaires et de sorties. Ces troncatures peuvent affecter la précision des sorties de la FFT en introduisant du bruit de quantification. Il est ainsi nécessaire d'évaluer ce bruit de quantification.

Plusieurs travaux ont traité le bruit causé par la troncature des opérateurs pour des valeurs à virgule fixe utilisées dans les transformations orthogonales tels que FFT [66], DCT [67] et filtres FIR [68]. Notre objectif dans cette section est d'évaluer les effets de troncatures pour notre architecture. Les troncatures ont été réalisées uniquement pour les multiplications. En

effet, les entrées sont codées sur n bits. Le calcul de chaque élément de calcul PE engendre une augmentation de 2 bits. Cette augmentation est dûe au fait que le PE à base de Radix-4 contient une cascade de 2 additionneurs, chacun introduit un dépassement de 1 bit. Concernant l'étape de multiplication, les valeurs de facteurs de rotation sont comprises entre 0 et 1. Par conséquent la multiplication par ces facteurs n'augmente pas la dynamique de la sortie du multiplieur. Pour cette raison, nous avons supposé que les entrées et sorties à chaque bloc multiplieur ont la même taille. Si nous supposons que le facteur de rotation est codé sur c bits, le résultat de la multiplication s'écrit naturellement sur $c+n$ bits. La trancature se fait en gardant que les n bits en MSB. Par conséquent, les sorties d'une FFT de taille N points sont codées sur $n + 2log_4N$ bits. Le terme log_4N représente le nombre d'étages.

Afin d'évaluer le RSBQ (Rapport Signal à Bruit de Quantification), nous appliquons un signal d'entrée sinusoïdale. Nous choisissons par exemple une fréquence d'entrée de 32 kHz et une fréquence d'échantillonnage de 100 kHz. Deux FFT ont été calculées. La première notée X_{fl} représente la FFT obtenue avec des données flottantes codées sur 64 bits en utilisant Matlab. La seconde FFT notée X_{fx} est celle obtenue en utilisant l'architecture proposée à entrée parallèle/sortie parallèle codée en VHDL. Le RSBQ est défini par :

$$RSBQ = 10log_{10}(\frac{\sum_k |X_{fx}(k)|^2}{\sum_k (|X_{fl}(k)| - |X_{fx}(k)|)^2}) \tag{2.27}$$

Ainsi, le dénominateur représente le bruit dû à la quantification. Nous présentons sur la Figure 2.21 l'évaluation du RSBQ pour différentes tailles de la FFT et différentes tailles des entrée. La taille des facteurs de rotation est fixée à 12 bits. Le RSBQ dépend du nombre d'étage de calcul parcouru et par conséquent de la taille de la FFT. En effet, en augmentant la taille de la FFT le nombre de troncature augmente, le bruit dû à la quantification augmente et par conséquent le RSBQ diminue. De plus, en augmentant la taille des entrées, la précision est meilleure et par conséquent le RSBQ augmente.

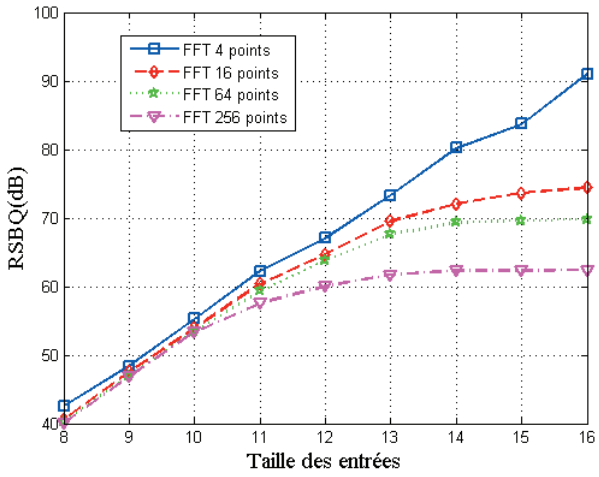


FIGURE 2.21 – Rapport Signal à Bruit de Quantification

2.7 Conclusion

Dans ce chapitre nous avons étudié dans un premier temps les algorithmes de calcul de la FFT. Par la suite après avoir exploré les architectures permettant l'implantation de FFT sur FPGA, nous avons proposé deux architectures optimisées afin d'améliorer les performances d'une implantation FPGA de la FFT. Les résultats obtenus ont montré l'efficacité des IP proposées. Dans la suite de la thèse, cette IP sera utilisée pour implanter des applications de traitements de signal et d'images.

Chapitre 3

Conception d'un algorithme optimisé pour la Transformée en Cosinus Discrète

Sommaire

3.1 Introduction

La DCT "Discrete Cosine Transform" (TCD "Transformée en Cosinus Discrète") est une transformation orthogonale discrète. Elle est utilisée dans sa forme unidimensionnelle (1D) dans de nombreuses applications telles que la réduction d'erreurs temporelles entre deux signaux [69], etc. Mais, le plus souvent, la DCT est utilisée sous sa forme bidimensionnelle (2D) en traitement d'images pour des applications diverses telles que la détection de formes [70], le tatouage numérique "watermarking" [71], l'estimation de mouvement [72] et plus particulièrement la compression d'images [73].

En outre, elle est utilisée dans la majorité des standards de compressions d'images telles que le JPEG et le JPEG XR et de vidéos telles que MPEG, H264 et HEVC. En effet, la DCT contribue à la réduction de la redondance spatiale en transformant le domaine spatial en domaine spectral. Cependant, la forme et la taille des blocs de la DCT 2D changent en fonction de la norme à implanter. Par conséquent, les concepteurs sont amenés à développer une architecture adaptée à chaque norme. Afin de réduire, cette diversification des architectures, nous proposons dans ce chapitre une architecture DCT 2D générique et adaptée pour tous les multistandards.

Nous commençons par définir la DCT dans sa forme 1D et 2D. Par la suite, nous étudions différents algorithmes de calcul de la DCT 1D en mettant l'accent sur les deux plus utilisés.

Ensuite, nous présenterons différentes architectures matérielles pour l'implantation de la DCT 2D. Finalement, nous proposons une architecture optimisée adaptée aux multistandards afin de réduire le nombre d'opérateurs utilisés. Les résultats obtenus montre l'efficacité de l'architecture proposée.

3.2 Définitions

La DCT est un des outils mathématiques qui permettent de convertir un signal temporel en un signal fréquentiel. Contrairement à la DFT qui effectue une analyse fréquentielle avec des nombres complexes, la DCT réalise cette analyse avec des nombres réels.

3.2.1 DCT 1D

Pour une séquence d'entrée $x(n)$, n un nombre entier naturel $\in [0, N-1]$, les coefficients de la DCT $X(u)$, u nombre entier naturel $\in [0, N-1]$ sont donnés par :

$$X(u) = C(u) \sum_{n=0}^{N-1} x(n) \cos \frac{(2n+1)u\pi}{2N} \tag{3.1}$$

Avec N est un nombre entier naturel qui représente la taille de la DCT. Cette transformée est reversible permettant de restituer le signal d'origine (temporel). Pour cela il suffit d'appliquer la Transformée en Cosinus Discrète Inverse : TCDI (IDCT "Inverse Discrete Cosine Transform") au signal fréquentiel X. Cette transformation inverse est donnée par :

$$x(n) = \sum_{u=0}^{N-1} C(u)X(u) \cos \frac{(2n+1)u\pi}{2N} \tag{3.2}$$

Les constantes de pondération $C(u)$ utilisées dans les équations 3.1 et 3.2 sont données par :

$$C(u) = \begin{cases} \frac{1}{\sqrt{N}} & \text{si } u = 0 \\ \frac{2}{\sqrt{N}} & \text{sinon} \end{cases} \tag{3.3}$$

En utilisant cette définition (équation 3.3) et en remplaçant $C(0)$ par sa valeur dans l'équation 3.1, le coefficient DCT à la fréquence 0 est donné par $X(0) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n)$. Ainsi, la première valeur de la transformée représente la somme de toutes les entrées pondérée par $\frac{1}{\sqrt{N}}$. Cette valeur est appelée DC (Direct Component) et représente l'offset de la transformée. Toutes les autres valeurs sont nommées Coefficients AC (Alternative Component) [74].

Comme cela est montré dans [75], le calcul des coefficients de la DCT peut se faire avec une combinaison linéaire d'un ensemble de fonctions définies par :

$$F_n(u) = C(u) \cos \frac{(2n+1)u\pi}{2N} \tag{3.4}$$

avec $u \in [0, N-1]$.

Ces fonctions sont appelées les fonctions de base.

L'équation 3.1 peut donc s'écrire sous la forme :

$$X(u) = \sum_{n=0}^{N-1} x(n)F_n(u) \tag{3.5}$$

L'avantage d'une telle décomposition réside dans le fait que ces fonctions, indépendantes de $x(n)$, peuvent être pré-calculées à l'avance. Ainsi, le calcul des coefficients la DCT s'effectue en réalisant une combinaison linéaire entre le signal d'entrée $x(n)$ et ces fonctions. Pour faciliter la compréhension de ces fonctions une représentation graphique de ces fonctions de base pour $N = 8$ est illustrée dans la Figure 3.1. Le premier schéma en haut à gauche de la Figure 3.1,

correspondant à $u = 0$, représente la composante DC des fonctions de base. Les autres courbes pour les valeurs de $u \in [1, 7]$ ont des fréquences qui augmentent progressivement. Notons que toutes ces fonctions sont orthogonales. Par conséquent, le produit scalaire point par point de deux fonctions différentes de la Figure 3.1 donne 0. Cependant, le résultat du produit scalaire point par point d'une fonction par elle-même donne une constante. Les fonctions orthogonales sont indépendantes, ainsi aucune de ses fonctions ne peut être obtenue par une combinaison linéaire des autres.

Enfin, notons que si la taille de la séquence d'entrée est supérieure à N, elle peut être divisée en plusieurs sous-séquences de taille N et par la suite la DCT peut être obtenue en utilisant les sous-DCT calculées avec les sous-séquences.

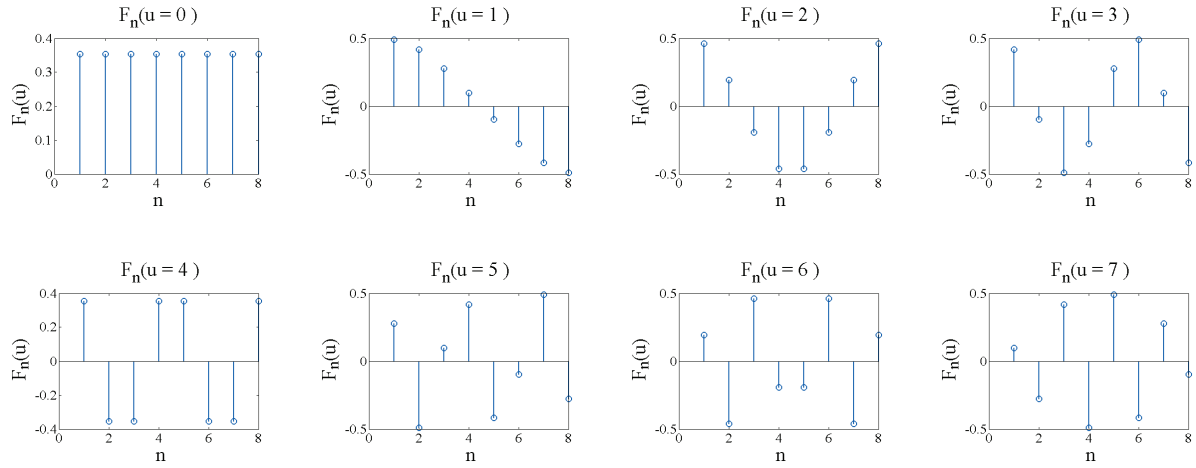


FIGURE 3.1 – Fonctions de base pour une DCT de taille 8 points [74]

3.2.2 DCT 2D

La DCT 2D est une transformation très utilisée dans les applications de traitement d'images et de vidéo comme la compression et le codage. Le calcul des coefficients $Y(u, v)$ d'une matrice d'entrée $y(i, j)$ peut se faire par l'équation 3.6.

$$Y(v,u) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \frac{C(v)}{2} \frac{C(u)}{2} y(i,j) \cos \frac{(2i+1)u\pi}{2N} \cos \frac{(2j+1)v\pi}{2N} \tag{3.6}$$

$C(u)$ et $C(v)$ sont définis par l'équation 3.3 et u, v, i, j sont des entiers naturels $\in [0, N-1]$. La matrice d'entrée $y(i, j)$ peut être recalculée à partir de $Y(u, v)$ en utilisant la DCT 2D inverse IDCT 2D définie par :

$$y(i,j) = \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} \frac{C(v)}{2} \frac{C(u)}{2} Y(v,u) \cos \frac{(2i+1)u\pi}{2N} \cos \frac{(2j+1)v\pi}{2N} \tag{3.7}$$

De la même manière que pour la DCT 1D, la DCT 2D peut être écrite sous la forme d'une combinaison linéaire d'un ensemble de fonctions appelées fonctions de bases. Le calcul de ces fonctions est réalisé par l'équation 3.8.

$$F_{i,j}(u,v) = \frac{C(v)}{2} \frac{C(u)}{2} \cos \frac{(2i+1)u\pi}{2N} \cos \frac{(2j+1)v\pi}{2N} \tag{3.8}$$

Par conséquent, l'équation permettant de calculer les coefficients de la DCT 2D est donnée par :

$$Y(v,u) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} F_{i,j}(u,v) y(i,j) \tag{3.9}$$

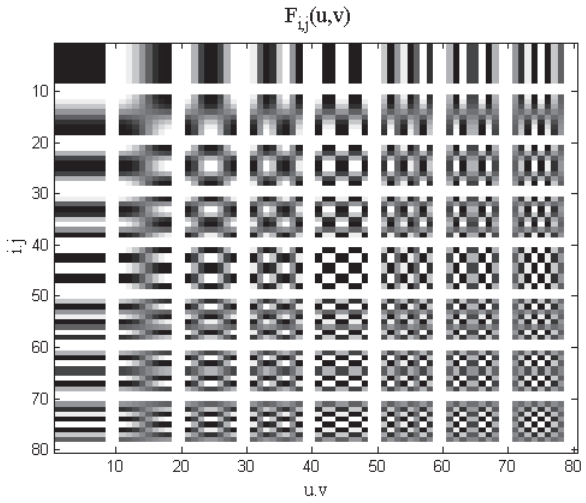


FIGURE 3.2 – Les fonctions de base de la DCT 2D de taille 8x8 [74]

Les fonctions de base de la DCT 2D peuvent être générées à partir des fonctions de base 1D (Figure 3.1). La répartition de ces fonctions pour $N = 8$ est illustrée dans la Figure 3.2.

Il est à noter que ces fonctions présentent une augmentation progressive de la fréquence à la fois dans le sens vertical et horizontal comme cela est montré sur la Figure 3.3. La fonction de base en haut à gauche de la Figure 3.2 représente la multiplication de la composante DC des fonctions de base de la DCT 1D par sa transposée. Ainsi, cette fonction est constante et représente la composante DC des fonctions de base de la DCT 2D. Les autres coefficients représentent les composantes AC.

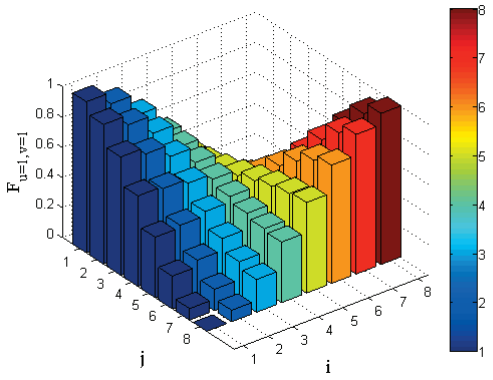


FIGURE 3.3 – Exemple de fonction de base de la DCT 2D

3.2.3 Intérêt de l'utilisation de la DCT

La DCT comme la DFT permet de transformer un signal temporel ou plan spatial en un signal fréquentiel ou plan spectral respectivement. Evidement, il s'agit de deux algorithmes différents et par conséquent le calcul des coefficients fréquentiel diffère selon l'algorithme utilisé : DFT ou DCT. Une des principales différences entre la DFT et la DCT au niveau du nombre d'opérateurs nécessaires pour réaliser l'une ou l'autre transformation. Cependant, dans les deux cas, toutes les multiplications utilisées sont des multiplications par des constantes. Dans le premier cas, pour $N = 8$, le calcul des coefficients de la DFT nécessite entre autres deux multiplications par la constante $\cos\frac{pi}{4}$ et deux multiplications par la constante $-\cos\frac{pi}{4}$ [44]. Cependant, comme nous le montrerons dans la section 3.3, les architectures optimisées de la DCT requièrent entre 11 et 16 multiplications. Par conséquent, le calcul des coefficients de la DCT est plus complexe que

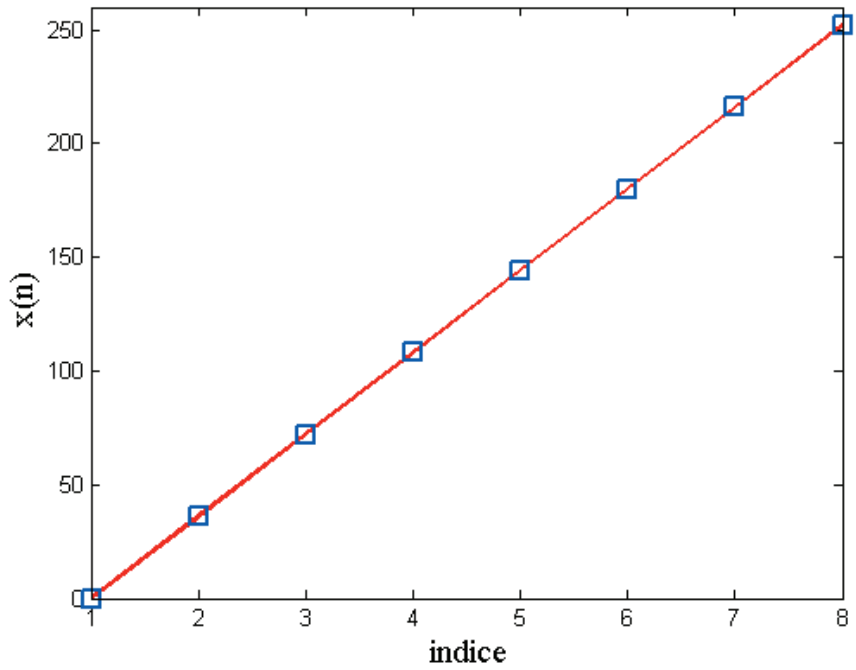


FIGURE 3.4 – Signal d'entrée

celui de la DFT dans le sens où il nécessite plus d'opérateurs. Malgré cela, pour des applications de compression : les standards JPEG, MPEG et H.26x ont privilégié l'utilisation de la DCT plutôt que la DFT. Afin de comprendre les raisons de ce choix une comparaison succincte entre l'efficacité de restitution d'un signal avec les deux transformations DCT/IDCT "Inverse Discrete Cosine Transform" et DFT/IDFT "Inverse Discrete Fourier Transform" a été présentée dans [76] et reproduite ici. Soit un signal sous forme d'une rampe défini sur 8 points de 8 bits chacun (Figure 3.2.3).

Les valeurs de sortie de la DCT et de la DFT appliquées à ce signal conformément aux équations 3.1 et 2.7 sont listées dans le tableau 3.1.

indice	$x(n)$	$DCT(x(n))$	$DFT(x(n))$
1	0	356.38	1008
2	36	-231.92	$-144+347j$
3	72	0	$-144+144j$
4	108	-24.24	$-144+59j$
5	144	0	-144
6	180	-7.23	$-144-59j$
7	216	0	$-144-144j$
8	252	-1.82	$-144-347j$

Tableau 3.1 – Valeurs des transformées d'une rampe

Avec j un nombre complexe vérifiant $j^2 = -1$.

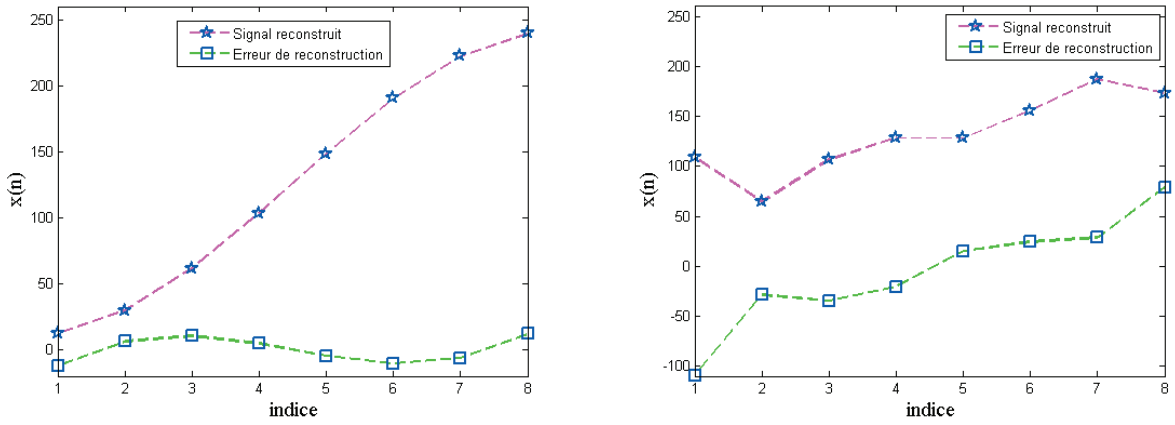
Pour pouvoir comparer l'efficacité de restitution de la DCT et de la DFT, nous n'utilisons que quelques valeurs des coefficients (dans le domaine spectral) de ces deux transformations, les autres valeurs sont fixées à zéro. Ceci constitue une modélisation très grossière de l'opération de quantification utilisée dans les applications de compression. Ainsi, dans cet exemple nous n'avons considéré que 3 valeurs dans le domaine spectral (les valeurs les plus importantes en intensité) comme indiqué dans le tableau 3.2.

Nous pouvons, facilement, remarquer que la DCT/IDCT donne une très bonne approxima-

indice	$x(n)$	$D\tilde{C}T(x(n))$	$ID\tilde{C}T(x(n))$	$D\tilde{F}T(x(n))$	$ID\tilde{F}T(x(n))$
1	0	356.38	2.28	1008	$90+61j$
2	36	-231.92	31.92	$-144+347j$	64
3	72	0	73.34	$-144+59j$	$100-36j$
4	108	-24.24	110.04	0	$126-25j$
5	144	0	141.95	0	$126-25j$
6	180	0	178.65	0	$151-36j$
7	216	0	220.07	0	187
8	252	0	249.71	0	$162+61j$

Tableau 3.2 – Transformées inverse après troncature

tion du signal original comme il est montré dans la Figure 3.5 (a). En effet, avec la DCT/IDCT et seulement 3 coefficients spectraux, nous avons réussi à restituer le signal d'origine avec une erreur de reconstruction (différence entre le signal $x(n)$ et signal le reconstruit après transformation)comprise entre [-12, 12] (Figure 3.5 (a)). Cependant, la restitution du signal d'origine avec DFT/IDFT cause une erreur qui varie de [-108, 78](Figure 3.5 (b)).



(a) (b)

FIGURE 3.5 – Evolution du signal : (a) Signal reconstruit avec DCT tronquée, (b) Signal reconstruit à partir d'une DFT tronquée

En effet, la DCT permet de localiser l'information utile dans les basses fréquences. Par conséquent, il est possible d'utiliser seulement cette information utile pour retrouver le signal d'origine. Cette propriété est très utile dans les applications de compression où nous essayons de minimiser la quantité de l'information à traiter. Ainsi, l'intérêt de la DCT comparé à celui de la DFT réside dans la capacité à restituer un signal avec peu de coefficients spectraux. Ceci peut expliquer son utilisation massive dans les applications de compression d'images.

3.3 Algorithmes pour le calcul de la DCT 1D

La réalisation directe de la DCT nécessite un nombre important d'opérateurs arithmétiques ($N \times (N-1)$) additions et N^2 multiplications [77]. Afin de proposer une architecture efficace de la DCT qui se prête bien dans les systèmes numériques embarqués, plusieurs algorithmes ont été proposés pour traiter la dynamique des coefficients de sorties de la DCT, l'architecture des multiplieurs à constantes [78–83], la réduction du nombre des opérateurs arithmétiques [84–91], etc.

Nous nous sommes intéressés plus particulièrement à des architectures permettant la réduction du nombre d'opérateurs afin d'optimiser la taille du circuit, sa consommation, la rapidité des

traitements et par conséquent le coût du circuit. Dans le but d'atteindre cet objectif, nous étudions les différents algorithmes proposés et validés dans la littérature. Ces algorithmes peuvent être classés en plusieurs catégories selon les méthodes de calcul employées.

3.3.1 Méthodes de calcul de la DCT

3.3.1.1 Calcul récursif : [89, 92]

La récursivité peut être utilisée pour calculer les coefficients d'une DCT de grande taille à partir de plusieurs blocs de DCT de petites tailles. Plusieurs algorithmes basés sur cette méthode ont été proposés. On peut citer par exemple l'algorithme proposé par Hou [89]. Cet algorithme consiste à utiliser une méthode récursive permettant le calcul des coefficients d'une DCT de taille N points à partir de deux blocs de DCT de taille $\frac{N}{2}$. En effet, pour un nombre d'entrées qui s'écrit sous la forme d'une puissance de deux, les coefficients de la DCT se calculent sur 2 parties. Une première utilise les éléments d'indices pairs tandis que la seconde partie utilise les éléments d'indices impairs. Ce principe peut se reproduire jusqu'à $N = 2$. Afin de mieux comprendre cette méthode, nous proposons de décrire l'équation de la DCT avec une représentation matricielle :

$$X = C_N \times x \tag{3.10}$$

Avec x est le signal d'entrée, C_N est une matrice de taille $N \times N$ définie par $C_N(i,j) = \cos(\frac{(2i+1)i\pi}{2N})$ et $i, j \in [0, N-1]$. Soient : (1) P_N la permutation qui organise les lignes d'une matrice de taille $N \times N$ afin d'avoir l'ordre suivant : $0, 2, 4, 6, .., N-2, N-1, N-3, .., 7, 5, 3, 1$ et (2) S_N la permutation qui organise les colonnes dans l'ordre suivant : $0, 2, 4, 6, ..., N-2, 1, 3, 5, 7, .., N-1$. Nous définissons la matrice $\hat{C}_N$ et $\bar{C}_N$ par :

$$\begin{aligned} \hat{C}_N &= C_N P_N \\ \bar{C}_N &= S_N^T \hat{C}_N \end{aligned} \tag{3.11}$$

La matrice $\bar{C}_N$ peut s'écrire sous la forme :

$$\bar{C}_N = \begin{bmatrix} \hat{C}_{\frac{N}{2}} & \hat{C}_{\frac{N}{2}} \\ D_{\frac{N}{2}} & -D_{\frac{N}{2}} \end{bmatrix} \tag{3.12}$$

Avec $D_{\frac{N}{2}}$ est une matrice de taille $\frac{N}{2} \times \frac{N}{2}$ qui s'écrit sous la forme [93]

$$D_{\frac{N}{2}} = L_{\frac{N}{2}} \hat{C}_{\frac{N}{2}} Q_{\frac{N}{2}} \tag{3.13}$$

Où $L_{\frac{N}{2}} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 2 & 0 & \cdots & 0 \\ -1 & 2 & -2 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & 2 & -2 & 2 & \cdots & 2 \end{pmatrix}$.

$Q_{\frac{N}{2}} = digonal[cos(\theta_n)]$, $\theta_n = cos(\frac{(4n+1)\pi}{2N})$, pour $n \in [0, \frac{N}{2} - 1]$ et la fonction *diagonal* est une fonction qui permet de créer une matrice diagonale.
Les différentes étapes réalisées par cette méthode pour le calcul des coefficients de la DCT 1D de taille N points sont illustrées dans la Figure 3.6.

Avec $M = \frac{N}{2}$, x_p et x_i représentent la partie paire et impaire du signal x, X_p est la DFT du signal x_p et X_i est la DFT du signal x_i.

Pour $N = 8$, cet algorithme permet de calculer les coefficients de la DCT en utilisant 12 multiplieurs et 29 additionneurs [89]. Cependant, comme le montre la Figure 3.6 cette méthode nécessite plusieurs éléments de stockage ainsi que des éléments de contrôle. En effet, le résultat obtenu à la fin de chaque étage doit être sauvegardé dans des mémoires ou des registres supplémentaires. De plus les permutations S_N et P_N nécessitent l'utilisation de plusieurs éléments de contrôle.

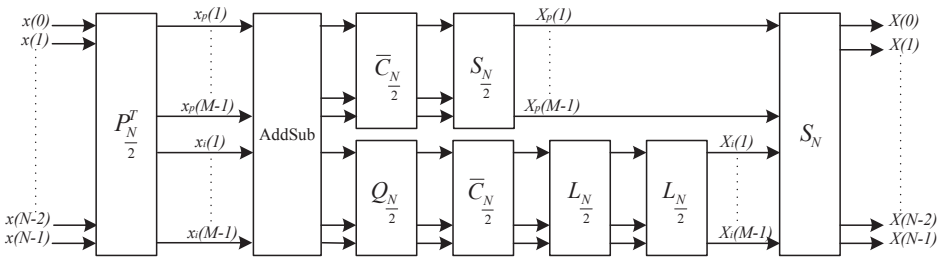


FIGURE 3.6 – Principe de la méthode récursive

3.3.1.2 Calcul indirect : [87, 94–98]

Cette méthode utilise la transformée de Fourier rapide (FFT) et la transformée WHT "Walsh-Hadamard transform" pour le calcul des coefficients de la DCT. En effet, les auteurs de [94] ont montré que le calcul de la DCT peut être effectué en utilisant deux blocs de calcul de FFT N points. Dans [95], les auteurs ont proposé un algorithme permettant d'obtenir la DCT en calculant uniquement la partie réelle des premiers N coefficients de la DFT de $2N$ points. Pour bien illustrer cet algorithme, considérons un signal $x(m)$, avec $m \in [0, N-1]$. La première phase de calcul des coefficients de la DCT du signal $x(m)$ consiste à dupliquer le signal comme suit :

$$x'(m) = \begin{cases} x(m_1) & \text{pour } 0 \leq m_1 \leq N-1 \\ x(2N - m_1 - 1) & \text{pour } N \leq m_1 \leq 2N-1 \end{cases} \tag{3.14}$$

La DFT de taille $2N$ appliquée au signal $x'(m)$ est donnée par :

$$X'(k) = \sum_{m=0}^{2N-1} x'(m) e^{-j\frac{2\pi}{2N}km} \tag{3.15}$$

L'équation 3.15 peut s'écrire sous la forme d'une somme de deux fonctions de N termes comme indiqué par l'équation 3.16.

$$\begin{aligned} X'(k) &= \sum_{m=0}^{N-1} x'(m) e^{-j\frac{2\pi}{2N}km} + \sum_{m=N}^{2N-1} x'(m) e^{-j\frac{2\pi}{2N}km} \\ X'(k) &= \sum_{m=0}^{N-1} x'(m) e^{-j\frac{2\pi}{2N}km} + \sum_{m=0}^{N-1} x'(m) e^{-j\frac{2\pi}{2N}k(2N-1-m)} \end{aligned} \tag{3.16}$$

Etant donné que $e^{-j2\pi k} = 1$, nous pouvons alors écrire :

$$\begin{aligned} X'(k) &= \sum_{m=0}^{N-1} x'(m)[e^{-j\frac{2\pi}{2N}km} + e^{j\frac{2\pi}{2N}km} e^{j\frac{2\pi}{2N}k}] \\ X'(k) &= \sum_{m=0}^{N-1} x'(m) e^{j\frac{\pi}{2N}k}[e^{-j\frac{2\pi}{2N}km} e^{-j\frac{\pi}{2N}k} + e^{j\frac{2\pi}{2N}km} e^{j\frac{\pi}{2N}k}] \\ X'(k) &= \sum_{m=0}^{N-1} x'(m) e^{j\frac{\pi}{2N}k}[e^{-j\frac{kn\pi}{2N}(2m+1)} + e^{j\frac{kn\pi}{2N}(2m+1)}] \end{aligned} \tag{3.17}$$

En se basant sur les propriétés de symétrie des fonctions trigonométriques, l'équation 3.17 devient :

$$\begin{aligned} X'(k) &= \sum_{m=0}^{N-1} 2x'(m)e^{j\frac{\pi}{2N}k}\cos(k(2m+1)\frac{\pi}{2N}) \\ X'(k) &= 2e^{j\frac{\pi}{2N}k}\sum_{m=0}^{N-1} x'(m)\cos(k(2m+1)\frac{\pi}{2N}) \end{aligned} \tag{3.18}$$

avec $k \in [0, N-1]$. La combinaison des équations 3.1 et 3.18 donne :

$$X(u) = \frac{1}{2}e^{-j\frac{\pi}{2N}u}C(u)X'(u) \tag{3.19}$$

Notons $C'(u) = \frac{1}{2}e^{-j\frac{\pi}{2N}u}C(u)$, ainsi l'équation 3.19 s'écrit :

$$X(u) = C'(u)X'(u) \tag{3.20}$$

Par conséquent l'équation 3.20 montre bien la possibilité de calculer les coefficients de la DCT 1D de taille N à partir d'une DFT de taille $2N$ points. Le schéma bloc présenté dans la Figure 3.7 illustre le synoptique de cette méthode.

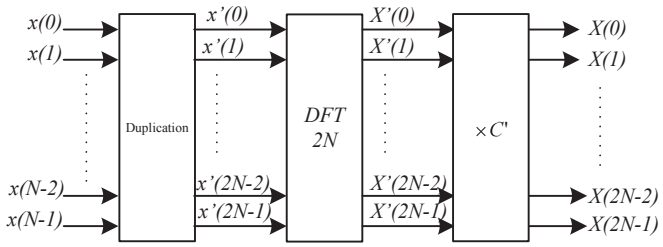


FIGURE 3.7 – Principe de la méthode de calcul indirecte

En utilisant cette méthode les coefficients d'une DCT de taille 8 points peuvent être obtenus en utilisant 12 multiplieurs et 29 additionneurs [87]. Cependant, l'inconvénient majeur de cette méthode est l'utilisation aussi des mémoires supplémentaires. Ces derniers sont nécessaires pour sauvegarder le signal en entrée de la DFT de taille $2N$.

3.3.1.3 DCT à base de convolution : [99, 100]

Cette méthode est basée sur le principe de reformulation des nombres premiers en un certain nombre de convolutions cycliques. Elle ne s'applique que pour un nombre N impair. Dans cette optique, les auteurs de [99] ont proposé une architecture de calcul de la DCT à base de deux convolutions cycliques similaires. Une deuxième architecture à été proposée dans [100]. Elle permet de calculer une DCT de N points à base de 4 convolutions cycliques de taille $\frac{N}{4}$. En effet, l'équation 3.1 peut s'écrire sous la forme :

$$X(k) = A'(k)\cos(\frac{k\pi}{2N}) - B'(k)\sin(\frac{k\pi}{2N}) \tag{3.21}$$

Avec

$$\begin{aligned} A'(k) &= \sum_{n=0}^{N-1} z(n)\cos(\frac{\pi kn}{M}) \\ B'(k) &= \sum_{n=0}^{N-1} z(n)\sin(\frac{\pi kn}{M}) \end{aligned} \tag{3.22}$$

$M = \frac{N}{2}$ et

$$z(n) = \begin{cases} x(2n) & \text{pour } 0 \leq n \leq M-1 \\ x(2N-2n-1) & \text{pour } M \leq n \leq N-1 \end{cases} \tag{3.23}$$

Considérons $a'(n) = z(n) + z(N-n)$ et $b'(n) = z(n) - z(N-n)$ avec $n \in [0, M]$, l'équation 3.22 peut donc s'écrire sous la forme :

$$\begin{aligned} A'(k) &= \sum_{n=0}^{M} a'(n) \cos(\frac{\pi kn}{M}) \\ B'(k) &= \sum_{n=0}^{M} b'(n) \sin(\frac{\pi kn}{M}) \end{aligned} \tag{3.24}$$

De la même manière que $X(k)$, $A'(k)$ et $B'(k)$ seront divisés en deux parties avec :

$$\begin{aligned} A'(2k) &= s'(0) + S'(k) \\ A'(2k-1) &= d'(0) + D'(k) \\ A'(2k) &= E'(k) \\ A'(2k-1) &= G'(k) \end{aligned} \tag{3.25}$$

avec :

$$S'(k) = \sum_{n=1}^{L} s'(n) \cos(\frac{\pi kn}{M}) \tag{3.26}$$

et

$$s'(n) = a'(n) + a'(M-n) \tag{3.27}$$

Dans le cas où $M = 7$, les coefficients de la DCT peuvent être obtenus en divisant le calcul sous la forme de 4 convolutions cycliques. L'équation 3.28 représente à titre d'exemple, une de ces convolutions :

$$\begin{pmatrix} X(0) \\ X(2) \\ X(4) \end{pmatrix} = \begin{pmatrix} c(2) & -c(3) & -c(1) \\ -c(1) & c(2) & -c(3) \\ -c(3) & -c(1) & c(2) \end{pmatrix} \times \begin{pmatrix} s(1) \\ s(2) \\ s(3) \end{pmatrix} \tag{3.28}$$

Le nombre d'opérateurs utilisés par cette méthode est de $\frac{N}{2}+3$ multiplieurs et $\frac{N}{2}+6$ additionneurs [100]. En effet, les auteurs de la référence [100] ont utilisé le principe des multiplieurs à base de mémoires afin de réduire d'avantage le nombre d'opérateurs requis. Ce nombre est ainsi inférieur à celui utilisé par les autres méthodes. Cependant, cette méthode n'est applicable que pour un nombre de points N impair. Ceci peut constituer un inconvénient pour des applications de compression où N est généralement égal à 4, 8, 16 ou 32. Les standards de compression fixent un nombre pair pour (taille de blocs $=4 \times 4$, 8×8, 16×16, etc.)

3.3.1.4 Factorisation directe : [85, 86, 88, 90, 101–103]

Le calcul de la DCT en utilisant la factorisation directe se base sur la propriété de symétrie de la matrice des coefficients de la DCT. L'avantage principal de cette méthode réside dans l'utilisation d'un nombre réduit d'additionneurs ou de multiplieurs. Cette méthode permet de calculer la DCT de tout vecteur de taille N qui s'écrit sous la forme de puissance de 2. Parmi les algorithmes basés sur cette méthode, on peut citer l'algorithme de Lee [86], de Wang [101], de Chen [85] et de Loeffler [90]. En effet, Wang [101] a proposé un algorithme optimisé pour

le calcul des coefficients de la DCT 1D. Cet algorithme permet aussi d'obtenir la transformée en sinus discrète (DST), la transformée en ondelettes DWT "Discrete Wavelet Transform" et la transformée de fourrier discrète (DFT) à partir de son algorithme pour la DCT. Une version optimisée de cet algorithme a été proposée par Suehiro [88]. Cette optimisation a permis de réduire le nombre de multiplieurs utilisés pour le calcul de la DCT ainsi que pour les autres transformées déduites.

Par ailleurs, dans cette famille d'algorithmes à base de factorisation directe, l'algorithme de Chen [85] est l'un des algorithmes les plus utilisés dans la littérature. Cet algorithme se base sur le principe de la factorisation matricielle afin de réduire le nombre d'opérateurs arithmétiques tout en gardant un nombre réduits d'éléments de contrôle et de mémorisation. En effet, l'algorithme de Chen permet d'obtenir les coefficients d'une DCT de taille 8 points en utilisant 16 multiplieurs et 26 additionneurs et un nombre réduit d'éléments de contrôle. Il présente ainsi le nombre le plus réduit d'additionneurs.

Un autre algorithme, très utilisé dans la littérature, basé sur la méthode de factorisation directe est l'algorithme de Loeffler [90]. La principale raison du succès de cet algorithme est le nombre réduit de multiplieurs requis. En effet, l'algorithme de Loeffler permet de calculer les coefficients la DCT de taille 8 points en utilisant uniquement 11 multiplieurs et 29 additionneurs. Le nombre de multiplieurs ainsi proposé représente la limite théorique pour le calcul des coefficients de la DCT introduite par Duhamel [91]. Un autre avantage de cet algorithme vient du fait que le nombre d'éléments de contrôle utilisé par cet algorithme est réduit.

Les auteurs dans [104] ont présenté une comparaison entre ces différents algorithmes. Le tableau 3.3 résume cette comparaison.

Auteur	Chen [85]	Wang [101]	Lee [86]	Suehiro [88]	Hou [89]	Vetterli [87]	Loeffler [90]
Mult	16	13	12	12	12	12	**11**
Add	**26**	29	29	29	29	29	29

Tableau 3.3 – Comparaison entre les différents algorithmes en termes de nombre d'opérateurs

Cette comparaison montre que la méthode basée sur la factorisation directe et celle basée sur le calcul indirecte nécessite moins d'opérateurs que celle basée sur le calcul récursif. De plus, bien que le nombre d'opérateurs utilisés par la méthode basée sur le calcul indirect soit équivalent à celui utilisé par la méthode basée sur le factorisation directe, cette dernière présente l'avantage de ne pas utiliser d'éléments de mémorisations et de logique séquentielle. En effet, la première étape de la méthode basée sur le calcul indirect consiste à dupliquer le signal d'entrée afin d'appliquer la DFT. Cette duplication nécessite l'utilisation de mémoires supplémentaires. En outre, la division du signal d'entrée en deux parties et les différents post-traitements illustrés dans la Figure 3.6 nécessitent plusieurs éléments séquentiels. Malgré le fait que la méthode à base de convolution utilise un nombre réduit d'opérateurs, l'utilisation de cette méthode est limitée vue qu'elle n'est appliquable que pour un nombre N impair. Cette contrainte rend son utilisation ambigüe pour les nouveaux standards de compression.

De plus, comme nous l'avons cité précédemment, les algorithmes de Loeffler [90] et de Chen [85] sont des algorithmes basés sur la méthode de calcul direct. Ces algorithmes sont les plus utilisés dans la littérature grâce au nombre d'opérateurs réduit utilisé dans le calcul des coefficients de la DCT ainsi qu'un nombre réduits d'éléments de contrôle et de mémorisations. Par conséquent, toutes ces raisons nous ont amené à étudier les spécificités de ces deux algorithmes.

3.3.2 Algorithme de Loeffler

Comme nous l'avons indiqué précédemment, Duhamel dans [91] a montré que la limite minimale théorique pour le calcul des coefficients de la DCT de taille 8 point en termes de nombre de multiplieurs est égale à 11 multiplieurs. Etant donné que l'algorithme proposé par Loeffler [90] a pu atteindre cette limite, c'est tout naturellement qu'il devient le plus utilisé. Le schéma bloc de la DCT de taille 8 points basée sur l'algorithme de Loeffler est illustré dans la Figure 3.8.

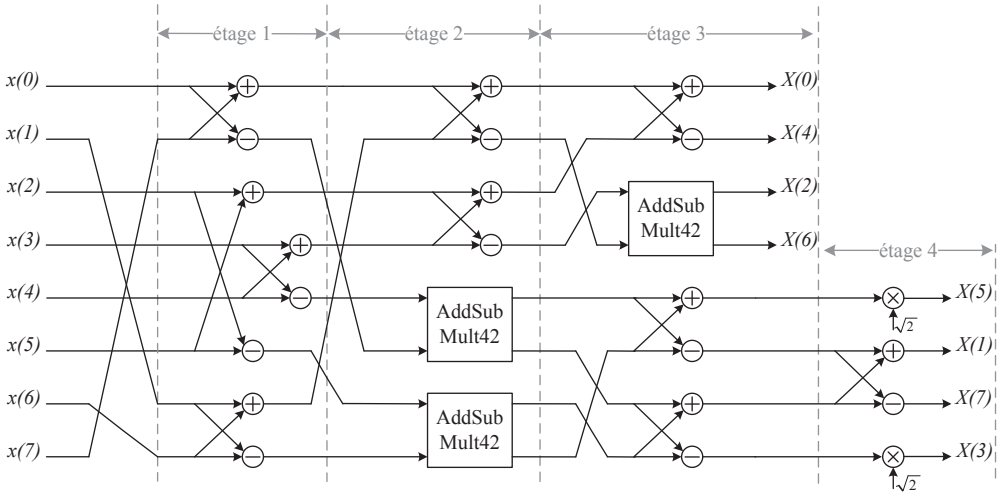


FIGURE 3.8 – Schéma bloc de la DCT Loeffler 8 points

Cette structure est composée de 4 étages. Le premier contient 4 additionneurs et 4 soustracteurs. Les entrées à cet étage sont placées dans un ordre naturel. Le second étage est composé de deux additionneurs, deux soustracteurs et deux blocs appelés papillons ou Butterfly. Nous donnons le nom de AddSubMult42 à ces butterfly vue qu'ils utilisent 4 multiplieurs et 2 additionneurs/soustracteurs. Cette architecture fournit des sorties dans un ordre inversé.

Une optimisation en termes de nombre d'opérateurs consommés par le bloc AddSubMult42 a été proposée par Wang dans [101] et consiste à utiliser 3 multiplieurs et 3 additionneurs au lieu de 4 multiplieurs et 2 additionneurs. Ce changement vient du fait que les sorties de ce bloc sont sous la forme :

$$\begin{aligned} out_0 &= \alpha \times in_0 + \beta \times in_1 \\ out_1 &= -\beta \times in_0 + \alpha \times in_1 \end{aligned} \tag{3.29}$$

avec in_0 et in_1 sont les entrées du Butterfly, out_0 et out_1 sont les sorties du Butterfly et α et β sont des constantes. Les sorties du bloc AddSubMult42 peuvent être schématisées comme indiqué dans la Figure 3.9.

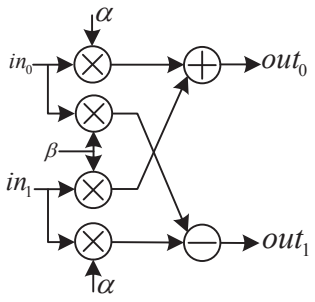


FIGURE 3.9 – Butterfly à base de 4 multiplications et 2 addititions

Les sorties de l'équation 3.29 peuvent être simplifiées :

$$\begin{aligned} out_0 &= \alpha \times in_0 + \beta \times in_1 = (\beta - \alpha) \times in_1 + \alpha \times (in_1 + in_0) \\ out_1 &= -\beta \times in_0 + \alpha \times in1 = -(\beta + \alpha) \times in_0 + \alpha \times (in_1 + in_0) \end{aligned} \tag{3.30}$$

Ainsi, nous pouvons économiser des opérateurs arithmétiques en structurant les sorties conformément à la Figure 3.10.

Au total, l'algorithme de Loeffler consomme 10 blocs composé chacun d'un additionneur et un soustracteur, 3 blocs de AddSubMult42 et 2 blocs de multiplications par constante. Par

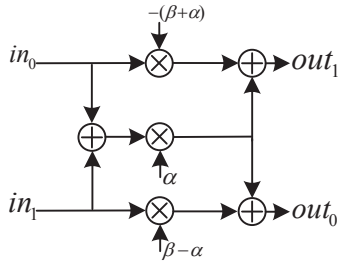


FIGURE 3.10 – Butterfly à base de 3 multiplications et 3 addititions

conséquent, le nombre total d'opérateurs arithmétiques utilisés par l'algorithme de Loeffler est de 11 multiplieurs et 29 additionneurs.

Dans l'objectif d'effectuer une analyse temporelle, nous étudions la latence absolue et relative de cet algorithme. En effet, la latence absolue correspond au nombre de cycle d'horloges qui sépare l'arrivée de la première entrée et la présence de la première sortie. La latence relative correspond aux temps écoulé entre l'arrivée de la première entrée et la présence de la première sortie. Pour effectuer cette analyse temporelle nous considérons les constantes T_A et T_M qui indiquent respectivement le temps d'une addition et le temps d'une multiplication.

La dépendance des données pour le calcul de la DCT exige la présence de toutes les données à traiter. Cette constatation est illustrée dans la Figure 3.8. Cependant, les sorties ne sont pas toutes prêtes au même instant. En effet, les sorties $X(0)$ et $X(4)$ sont prêtes après que les données ont parcourus 3 étages alors que les autres sorties sont disponibles après le parcous de 4 étages. De plus, les blocs utilisés pour calculer les sorties $X(0)$ et $X(4)$ contiennent uniquement des additions et des soustractions. La sortie de chaque bloc est donnée alors après un temps égal à T_A ou après 1 seul cycle d'horloge si nous supposons qu'une opération d'addition ou de soustraction s'exécute en un cycle d'horloge. La latence relative de l'algorithme de Loeffler est égale à $3T_A$ et celle absolue est de 3 cycles d'horloge.

3.3.3 Algorithme de Chen

L'algorithme de Chen [85] est parmi les premiers algorithmes qui ont été proposés pour le calcul de la DCT. Tout comme l'algorithme de Loeffler, l'algorithme de Chen permet la réalisation de la DCT de façon simple et économique en termes de ressources. Cet algorithme possède une architecture structurée et utilise 26 additionneurs. Ce dernier représente le nombre minimum d'additionneurs pour une DCT de taille 8 points. Les équations permettant le calcul des coefficients de la DCT peuvent s'écrire sous une forme matricielle donnée par :

$$\begin{aligned}
\begin{pmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{pmatrix} &= \begin{pmatrix} f_4 & f_4 & f_4 & f_4 \\ f_2 & f_6 & -f_6 & f_2 \\ f_4 & -f_4 & -f_4 & f_4 \\ f_6 & -f_2 & f_2 & -f_6 \end{pmatrix} \times \begin{pmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{pmatrix} \\
\begin{pmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{pmatrix} &= \begin{pmatrix} f_1 & f_3 & f_5 & f_7 \\ f_3 & -f_7 & -f_1 & -f_5 \\ f_5 & -f_1 & f_7 & f_3 \\ f_7 & -f_5 & f_3 & -f_1 \end{pmatrix} \times \begin{pmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{pmatrix}
\end{aligned} \tag{3.31}$$

Avec $f_i = \cos(\frac{2i\pi}{16})$. Ces équations peuvent être présentées par le schéma bloc comme dans la Figure 3.11.

L'architecture proposée par Chen nécessite un ordre naturel pour les entrées et fournit des sortie dans un ordre inversé. Cette architecture est composée de 4 étages et 13 blocs butterfly. Chacun de ces blocs représente une multiplication par une matrice 2×2. Les blocs utilisés dans cette architecture sont divisés en trois groupes. Un premier groupe réalise uniquement des additions et des soustractions. Un deuxième groupe AddSubMult42 utilise 4 multiplieurs et

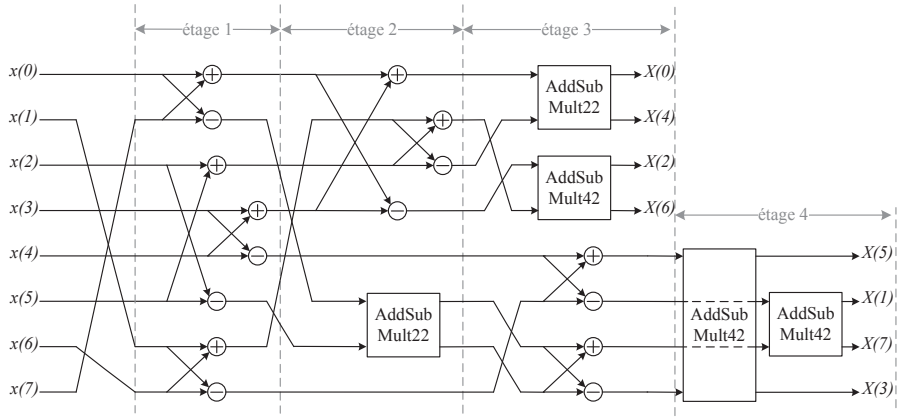


FIGURE 3.11 – Schéma bloc de la DCT Chen 8 points

2 additionneurs. Ce nombre peut être modifié à 3 additionneurs et 3 multiplieurs en utilisant la simplification de Wang comme expliqué dans la Figure 3.10. Le troisième et dernier groupe AddSubMult22 utilise uniquement 2 multiplieurs et deux additionneurs. Le schéma bloc du AddSubMult22 est illustré dans la Figure 3.12. Au total, le nombre d'opérateurs utilisés par cette architecture est de 16 multiplieurs et 26 additionneurs. Ce nombres peut passer à 13 multiplieurs et 29 additionneurs en exploitant la factorisation de Wang [101].

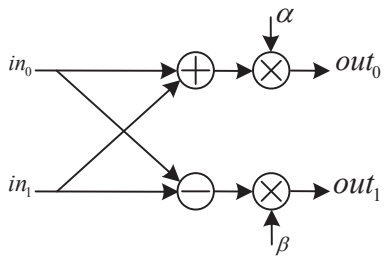


FIGURE 3.12 – Butterfly à base de 2 multiplications et 4 addititions

Afin d'effectuer une analyse temporelle de l'algorithme de Chen, nous considérons les mêmes constantes T_A et T_M utilisées précédemment. En effet, comme montré dans la Figure 3.11, les premières sorties disponibles sont les sorties paires $X(0), X(2), X(4), X(6)$. Contrairement à l'algorithme de Loeffler où tous les blocs utilisés pour calculer ces deux sorties contiennent uniquement des additions, le bloc utilisé dans le troisième étage du schéma bloc de l'algorithme de Chen (Figure 3.11) contient aussi des multiplications (Figure 3.12). Le temps d'exécution de ce bloc est égal à $T_M + T_A$. Ainsi, la latence relative de l'algorithme de Chen est égale à $3T_A + T_M$. Comme dans l'algorithme de Loeffler, si nous supposons que toute opération arithmétique s'exécute en un seul cycle d'horloge, la latence absolue est égale à 4.

3.3.4 Discussion

Dans cette section, une étude comparative entre l'algorithme de Loeffler et celui de Chen a été effectuée. Il s'agit d'une comparaison qui est basée sur le nombre d'opérateurs arithmétiques utilisés et sur les latences absolue et relative. Le résultat de cette comparaison est illustré dans le tableau suivant :

Il est montré que l'algorithme de Loeffler présente des latences plus faibles et un nombre d'opérateurs utilisés plus réduit. Afin de confirmer ces résultats, nous effectuons une synthèse logique de ces deux algorithmes et nous présenterons les résultats de synthèse obtenus dans le chapitre 6. Dans la section suivante nous étudions les architectures pour une implantation matérielle de la DCT 2D.

	Chen	Chen et Wang	Loeffler	Loeffler et Wang
Multiplieurs	13	16	11	13
Additionneurs	29	26	29	27
Latence relative	$3T_A + T_M$	$3T_A + T_M$	$3T_A$	$3T_A$
Latence absolue	4	4	3	3

Tableau 3.4 – Comparaison en termes de nombres d'opérateurs et de latence entre l'algorithme de Chen et l'algorithme de Loeffler

3.4 Architectures pour la DCT 2D

Les transformées DCT 2D directes et inverses permettent de passer du domaine bidimentionnel spatial au domaine bidimentionnel fréquentiel et vice versa. Pour une matrice d'entrée y de taille $N \times N$ la DCT 2D est donnée par l'équation 3.6. Cette équation peut être écrite sous une forme matricielle en utilisant des produits de matrices entre la matrice y et une matrice A.

$$Y = AyA^T \tag{3.32}$$

avec $A_{ij} = C(i)\cos\frac{(2j+1)i\pi}{2N}$

La réalisation de la DCT 2D se base sur l'algorithme de la DCT 1D. Comme nous l'avons vu précédemment, les algorithmes de la DCT 1D ont été largement étudiés dans les années 1970 par Loeffler [90] et dans les années 1980 par Chen [85]. A l'époque, l'enjeu était de réaliser la transformée sur une surface très réduite puisque le silicium était très cher. Aujourd'hui, la technologie a évolué et les enjeux se sont déplacés vers la rapidité, réduction de la puissance et la flexibilité tout en gardant une surface acceptable. Pour cette raison, nous avons étudié les algorithmes de Chen et de Loeffler qui sont les plus économiques en surface et nous nous efforçons à trouver les moyens algorithmiques et architecturaux pour une réalisation efficace de la DCT 2D.

3.4.1 Réalisation à base de séparation ligne colonne

Le nombre important d'opérations requis pour des grandes valeurs de N a amené les chercheurs à diviser la matrice d'entrée en des blocs de tailles plus petites (2×2, 4×4, 8×8 ou 16×16). Pour une image d'entrée de taille $N \times N$, et après division en plusieurs blocs de petites tailles, le principe de la DCT 2D basée sur la séparation ligne colonne consiste à : (1) appliquer l'algorithme de la DCT sur chaque ligne et (2) appliquer la DCT par la suite sur chaque colonne [86]. Une illustration de ce principe est donnée sur la Figure 3.13.

En effet, la transformation en colonne peut être exprimée par :

$$Y(v,u) = \sum_{j=0}^{N-1} X(v,j)\cos\frac{(2j+1)u\pi}{2N} \tag{3.33}$$

où $X(v,j)$ est la transformation en ligne . Cette transformation est exprimée par :

$$X(v,j) = \sum_{i=0}^{N-1} y(v,i)\cos\frac{(2i+1)v\pi}{2N} \tag{3.34}$$

où $X(v,j)$ représente la transformée ligne de la matrice d'entrée.

La réalisation de la DCT à base de la séparation ligne colonne est la plus adaptée pour une implantation matérielle de la DCT 2D. La réalisation matérielle de cette architecture présentée sur la Figure 3.14 est composée de trois étages. Le premier étage consiste à utiliser une première DCT qui sera appliquée à chaque vecteur ligne de la matrice d'entrée. Dans le deuxième étage, le résultat obtenu sera sauvegardé dans une mémoire intermédiaire afin d'effectuer la transposé de la matrice résultante. Le dernier étage est composé d'une DCT 1D à appliquer sur chaque ligne de la mémoire.

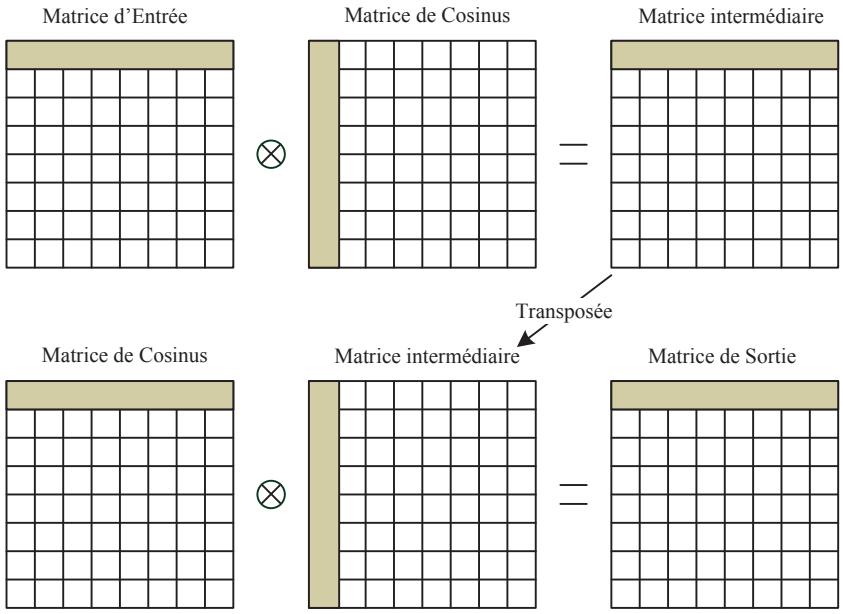


FIGURE 3.13 – Principe de la réalisation ligne colonne de la DCT 2D

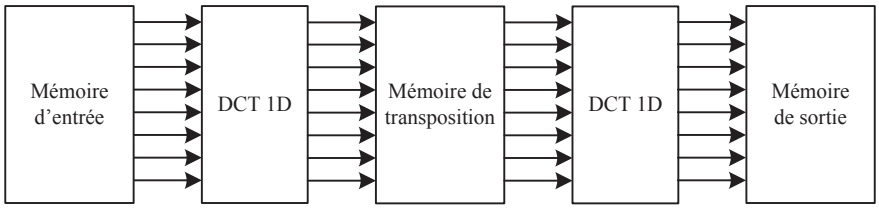


FIGURE 3.14 – Schéma bloc de la réalisation matérielle de la séparation ligne colonne

3.4.2 Réalisation directe

La deuxième approche permettant une implantation matérielle de la DCT 2D consiste à effectuer une réalisation directe en se basant sur la multiplication matricielle présente dans l'équation 3.32 [105]. Pour simplifier la représentation matricielle, prenons l'exemple où $N = 4$, l'équation 3.32 combinée avec l'équation 3.6 donnent :

$$Y = AyA^T = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix} \begin{pmatrix} y(0,0) & y(0,1) & y(0,2) & y(0,3) \\ y(1,0) & y(1,1) & y(1,2) & y(1,3) \\ y(2,0) & y(2,1) & y(2,2) & y(2,3) \\ y(3,0) & y(3,1) & y(3,2) & y(3,3) \end{pmatrix} \begin{pmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{pmatrix} \tag{3.35}$$

La matrice A est représentée par 3 éléments qui sont $a = \frac{1}{3}$, $b = \frac{1}{\sqrt{2}}\cos(\frac{\pi}{8})$ et $c = \frac{1}{\sqrt{2}}\cos(\frac{3\pi}{8})$. La réalisation directe de cette multiplication nécessite un nombre élevé d'opérateurs arithmétiques (additionneurs et multiplieurs). En effet, pour une matrice de taille N, le nombre d'opérations nécessaires pour réaliser l'équation 3.35 est de $2N^3$ multiplications et $2N^2(N-1)$ additions. Par conséquent, pour $N = 4$, cette équation nécessite 128 multiplications et 96 additions.

3.4.3 Proposition d'une architecture optimisée de la DCT 2D

Dans cette partie nous proposons une architecture optimisée pour la réalisation directe de la DCT 2D, à titre d'exemple, nous avons considéré le cas où $N = 4$. Il s'agit d'une optimisation conjointe entre le calcul des coefficients de la DCT et le processus de quantification dans le cadre d'une application de compression d'images. Cette architecture peut être étendue pour des valeurs $N > 4$; un exemple où $N = 8$ est traité dans l'annexe C.

3.4.3.1 Algorithme

La multiplication matricielle présentée dans l'équation 3.35 peut être simplifiée. En effet, la matrice A dans l'équation 3.35 peut s'écrire sous la forme

$$A = B \times C \tag{3.36}$$

B et C sont 2 matrices de taille 4×4 représentées par :

$$B = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & b \end{pmatrix} \text{ et } C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{pmatrix}$$

Avec $d = \frac{c}{b}$.

Ainsi, les équations 3.35 et 3.36 donnent

$$Y = BCyC^T B \tag{3.37}$$

Etant donné que la matrice B est une matrice diagonale, Y peut s'écrire :

$$Y = CyC^T \otimes E \tag{3.38}$$

Avec $E = \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix}$

L'opérateur $\otimes$ correspond à la multiplication point par point entre la matrice E et la matrice CyC^T.

Il est évident que la matrice E influe sur la répartition des coefficients de la DCT 2D. Mais puisqu'il s'agit de constantes de multiplication, et dans le but d'avoir une architecture optimisée, la multiplication par la matrice E sera déplacée dans le processus de quantification pour des applications de compression d'images. Elle sera ainsi considérée comme un facteur d'échelle pour la transformée directe et inverse.

La matrice CyC^T sera utilisée pour le calcul de la DCT 2D de taille 4×4. Ainsi le calcul de la DCT 2D se fait à partir de l'équation suivante :

$$Y = CyC^T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{pmatrix} \begin{pmatrix} y(0,0) & y(0,1) & y(0,2) & y(0,3) \\ y(1,0) & y(1,1) & y(1,2) & y(1,3) \\ y(2,0) & y(2,1) & y(2,2) & y(2,3) \\ y(3,0) & y(3,1) & y(3,2) & y(3,3) \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{pmatrix} \tag{3.39}$$

Cette représentation est plus simple à réaliser quand elle est comparée avec celle de l'équation 3.35. En effet, la matrice C contient peu de constantes et présente plusieurs symétries ce qui facilite la détermination d'une structure systolique à base de butterfly. En effet, la Figure 3.15 présente le schéma bloc de la transformation DCT 2D de taille 4×4 résultant de l'équation 3.35. Il est montré que le schéma bloc de cette transformation est divisé en 4 étages. Le premier et le troisième étage sont composés uniquement d'additionneurs et de soustracteurs (Figure 3.16) alors que le second et le dernier étage sont composés d'additionneurs, soustracteurs et aussi de multiplieurs (Figure 3.17) par la constante d. Le nombre d'opérateurs utilisés est égal à 16 multiplieurs et 64 additionneurs/soustracteurs.

3.4.3.2 Analyse des performances

L'architecture proposée est une architecture systolique basée sur deux éléments de base PE et PE_d "Processing Element" présentés dans la Figure 3.16 et 3.17.

Cette nouvelle architecture permet ainsi de réduire le nombre de multiplieurs. Cette réduction vient du fait que nous avons diminué le nombre de coefficients de la matrice A. En effet, le tableau

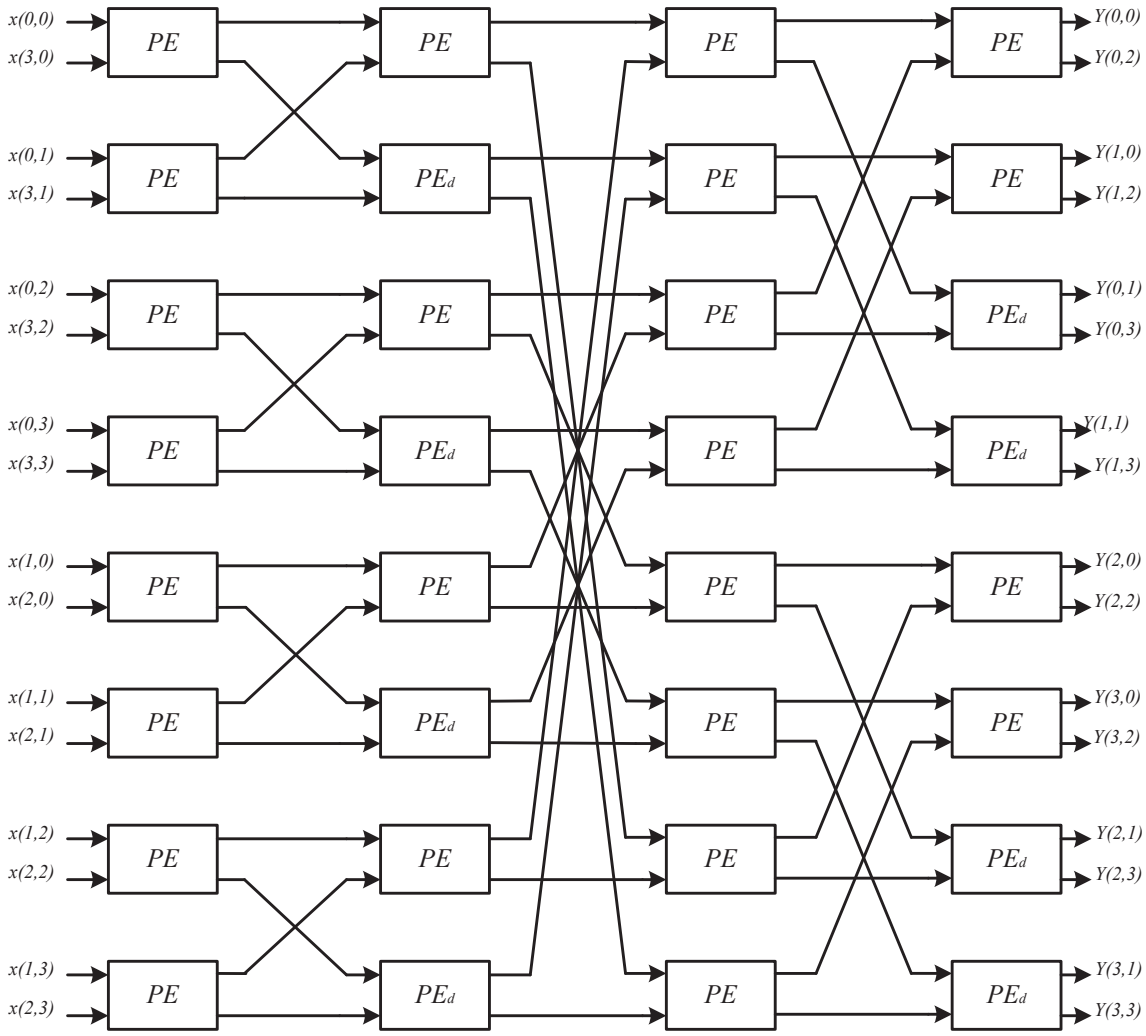


FIGURE 3.15 – Schéma bloc de l'architecture proposée pour une DCT 2D 4×4

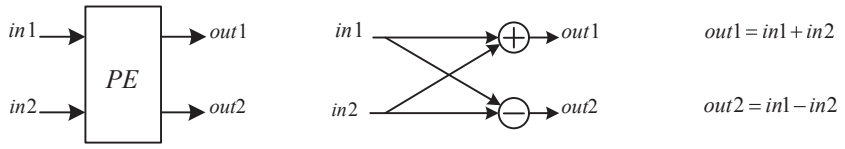


FIGURE 3.16 – Elément de base (PE)

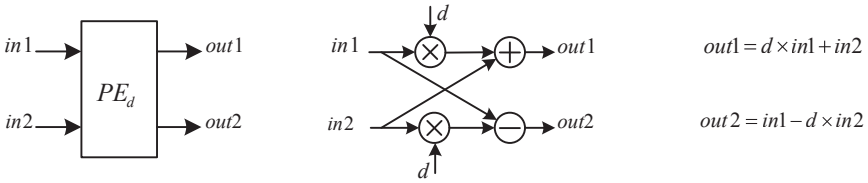


FIGURE 3.17 – Elément de base (PE_d)

3.5 illustre une comparaison en termes de nombre d'opérations arithmétiques et de latence entre plusieurs réalisations possibles d'une DCT 2D de taille 4×4.

Il est montré d'une part que l'architecture proposée effectue moins d'opérations qu'une réalisation à base de la séparation ligne colonne avec les algorithmes de Chen ou de Loeffler [1]. D'autre

1. Le nombre entre parenthèses représente la valeur correspondante sans exploiter la factorisation de Wang présentée dans la Figure 3.10

	Séparation ligne colonne		Réalisation directe	
	Chen	Loeffler	Théorique	Proposée
Multiplications	40 (48)[1]	24 (32)	128	16
Additions	72 (64)	72 (64)	96	72
Latence relative	$5[2T_A + T_M]$ ($5[3T_A + T_M]$)	$5[2T_A + T_M]$ ($5[3T_A + T_M]$)	$4T_A + 2T_M$	$4T_A + 2T_M$
Latence absolue	15 (20)	15 (20)	6	6
Temps d'exécution	$8[2T_A + T_M]$ ($8[3T_A + T_M]$)	$8[3T_A + T_M]$ ($8[3T_A + T_M]$)	$4T_A + 2T_M$	$4T_A + 2T_M$

Tableau 3.5 – Comparaison entre plusieurs architectures de DCT 2D

part, la simplification de la représentation matricielle de l'équation 3.35 et la description de l'architecture de calcul sous forme de Butterfly comme montré sur la Figure 3.15 ont permis de réduire le nombre d'opérations. De plus nous avons gardé la même latence et le même temps d'exécution. Par ailleurs, le nombre de multiplications réalisées par l'architecture proposée est 8 fois inférieur à celui utilisé par l'équation 3.35 et 3 fois inférieur à celui utilisé par la réalisation ligne colonne basée sur l'algorithme de Chen.

De plus l'architecture proposée à base de réalisation directe permet de calculer les coefficients en 4 étages. Ceci induit une latence faible comparée aux architectures existantes. En effet, si nous considérons les mêmes constantes T_A et T_M introduites dans la section 3.3, le temps d'exécution d'un PE est de T_A et celui du PE_d est de $T_M + T_A$. Par conséquent, les coefficients de DCT 2D sont disponibles après $4T_A + 2T_M$. Ce temps est inférieur à celui de l'architecture à base de la séparation ligne colonne qui est de $5(2T_A + T_M)$. Un autre avantage de l'architecture proposée réside dans le temps d'exécution faible. En effet, toutes les sorties sont disponibles en même instant ce qui rend le temps d'exécution égale à la latence, contrairement à l'architecture ligne colonne où la latence est inférieure au temps d'exécution.

3.4.3.3 Extension : Applications multistandards

Les nouveaux standards tels que MPEG-4 et H.264 [106] introduits dans les années 2000 utilisent la DCT entière à la place de la DCT réelle. Cette forme de DCT est une approximation de la DCT réelle. Elle s'applique à des blocs de taille 4×4 et elle est donnée par la multiplication matricielle suivante :

$$Y = HyH^T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & -2 & -1 \\ 1 & -1 & -1 & 1 \\ 2 & -1 & 1 & -2 \end{pmatrix} \begin{pmatrix} y(0,0) & y(0,1) & y(0,2) & y(0,3) \\ y(1,0) & y(1,1) & y(1,2) & y(1,3) \\ y(2,0) & y(2,1) & y(2,2) & y(2,3) \\ y(3,0) & y(3,1) & y(3,2) & y(3,3) \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 2 \\ 1 & 2 & -1 & -1 \\ 1 & -2 & -1 & 1 \\ 1 & -1 & 1 & -2 \end{pmatrix} \quad (3.40)$$

En effet, avec cette forme de DCT, la matrice cosinus de l'équation 3.39 est remplacée par une matrice à valeurs entière 1, -1, 2, -2, d'où le nom de DCT entière donné à cette DCT. Le but de cette DCT entière consiste à alléger les contraintes matérielles (notamment au niveau des multiplieurs nécessaires dans le cas de la DCT réelle).

Nous pouvons remarquer qu'il y a une forte similitude entre les équations 3.40 et 3.39. En effet, en remplaçant d par 2, nous passons d'une DCT réelle avec un facteur d'échelle inclus dans la matrice de quantification à une DCT entière. De ce fait, le schéma bloc pour la réalisation d'une DCT entière peut être déduit de celui de la Figure 3.15. De plus, pour une architecture de DCT utilisée dans la compression d'images, nous proposons d'utiliser le PE_{int} de la Figure 3.18. Cela nous permet, à l'aide d'une entrée de sélection, placée au niveau du multiplieur, nommée *sel*, de passer d'une architecture adaptée à une structure de type JPEG ou H.263 à un standard de type H.264 ou HEVC.

Par conséquent une seule architecture peut être utilisée pour calculer à la fois la DCT 2D 4×4 réelle et entière.

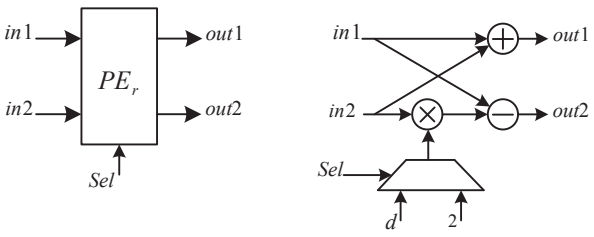


FIGURE 3.18 – Elément de base (PE_{int})

3.5 Conclusion

Dans ce chapitre nous avons présenté dans un premier temps les différents algorithmes de calcul des coefficients de la DCT 1D ainsi qu'une comparaison algorithmique entre les deux algorithmes les plus utilisés. Ensuite, après avoir parlé des algorithmes de la réalisation de la DCT 2D nous avons présenté une nouvelle architecture afin d'améliorer la rapidité de calcul tout en optimisant le nombre d'opérateurs arithmétiques nécessaires (par conséquent nous avons optimisé la surface occupée).

Les résultats obtenus ont montré que le nombre de multiplications réalisées par l'architecture proposée est 8 fois inférieur à celui utilisé pour une DCT à base de multiplication matérielle et 3 fois inférieur à celui utilisé par la réalisation ligne colonne basée sur l'algorithme de Chen. De plus, nous avons montré que l'architecture proposée se prête bien à une utilisation en compression d'images et de la vidéo quelque soit le standard employé. Une implantation matérielle à été réalisée et les résultats obtenus sont détaillés dans le chapitre 6.

Troisième partie

Validation et Applications

Chapitre 4

Validation et Mesures

Sommaire

4.1 Introduction

Dans ce chapitre nous nous concentrons sur l'utilisation des IP pour l'implantation des applications de traitements de signal sur FPGA. Plus particulièrement, nous nous sommes intéressés à des applications de réception d'un signal radiofréquence (RF). En effet, afin de tester les limites de l'IP FFT proposée, nous avons choisi de la tester dans un environnement contraignant (le temps de réponse et l'efficacité des traitements) comme celui de la guerre électronique. Pour cette application la FFT est utilisée pour réaliser la conversion du domaine temporel au domaine fréquentiel et dans le but de détecter la fréquence d'un signal ce qui facilite sa restitution.

Nous commençons ce chapitre par une validation du fonctionnement de la FFT en l'intégrant dans une chaine de numérisation. Par la suite, nous utilisons un banc de test afin de mesurer la puissance consommée par l'IP FFT proposée. Ensuite, nous présentons un des domaine d'utilisation de la FFT dans les applications de traitement de signal : La détection des signaux dans la guerre électronique. Nous finissons ce chapitre par présenter les résultats d'implantation obtenus et pour montrer l'avantage de l'IP proposée par rapport à l'IP de Xilinx.

4.2 Chaine de numérisation

Afin de valider le fonctionnement de l'IP FFT, nous l'avons utilisé dans un contexte d'évaluation d'une chaine de numérisation.

4.2.1 Banc de test

Le banc de test que nous avons mis en place pour implanter et valider cette chaine de numérisation est montré sur la Figure 4.1.

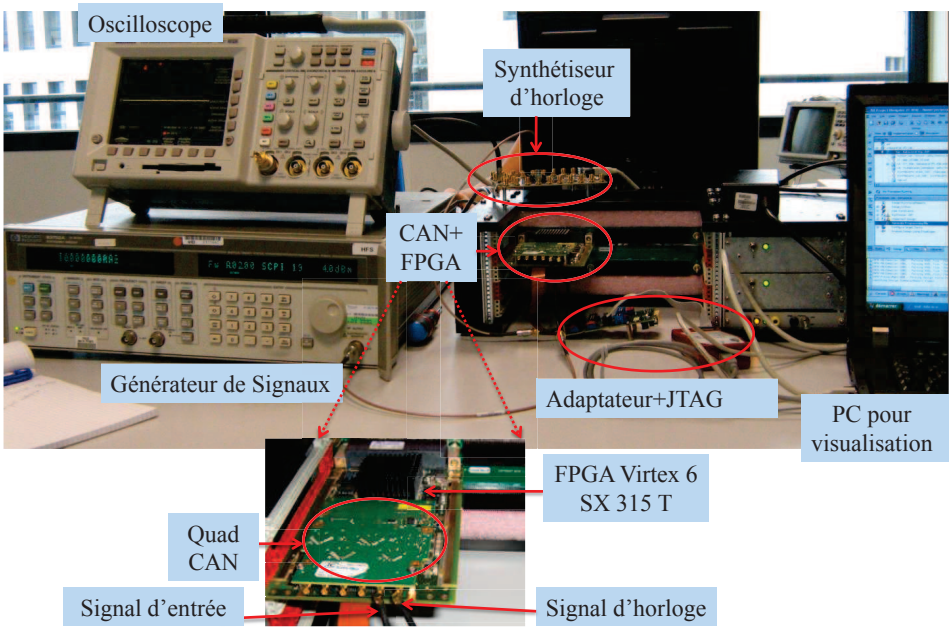


FIGURE 4.1 – Banc de test

Ce banc de test est composé de :

- Un générateur de signaux analogiques (HP 83752AlB).
- Un oscilloscope pour visualiser le signal analogique.
- Un convertisseur analogique numérique CAN (Quad CAN, F_e=122.88 MHz, res=16 bits).
- L'IP FFT proposée implantée sur une carte FPGA (Virtex-6 SX 315 T).
- Synthétiseur d'horloge (Texas Instruments CDCE72010)
- PC et outil de visualisation des résultats (ChipScope, ILA, VIO, etc.).
- Un adaptateur et une liaison JTAG.

La Figure 4.2 illustre un schéma simplifié de l'interconnexion des différents composants de ce banc de test.

Le générateur de signaux analogiques (HP 83752AlB) permet de fournir des signaux analogiques sous une forme sinusoïdale avec une fréquence F_0. Nous fixons cette fréquence $F_0 = 32,72$ MHz. Ce générateur est lié à un convertisseur analogique numérique CAN afin de numériser le signal d'entrée. Le CAN utilisé échantillonne le signal analogique avec une fréquence d'échantillonnage F_e égale à 122.88 MHz et fournit des échantillons codés sur 16 bits. La fréquence F_e est considérée comme fréquence globale de tout le système.

Ce convertisseur est un Quad CAN ce qui signifie qu'il permet d'échantillonner 4 signaux analogiques et fournir 4 sorties numériques codées sur 16 bits chacune. Cependant, faute de matériel nous utilisons uniquement une seule entrée et une seule sortie de ce CAN.

Nous appliquons par la suite à ces échantillons une FFT de taille 256 points en utilisant l'IP FFT proposée et implantée sur une carte FPGA Virtex-6. Cette carte est liée à un PC via un adaptateur et une liaison JTAG afin de visualiser les sorties en utilisant l'outil ChipScope Pro.

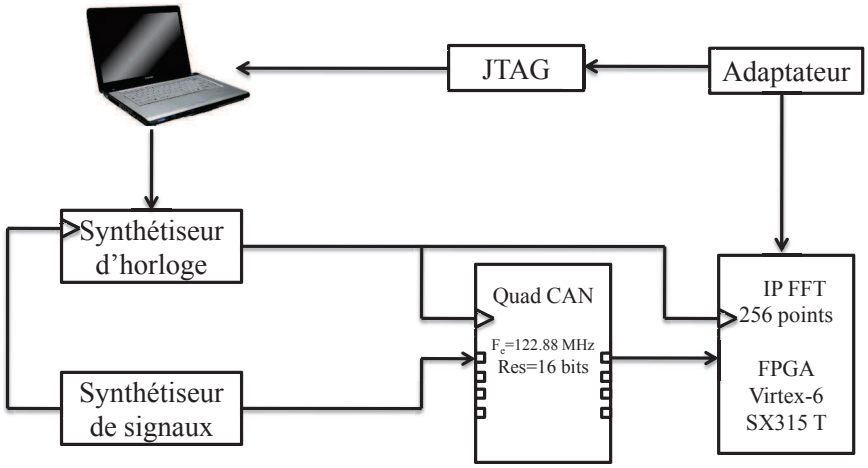


FIGURE 4.2 – Synoptique du banc de test utilisé

4.2.2 Prototypage rapide sur FPGA Xilinx

ChipScope Pro est un outil de déboguage temps-réel pour les FPGA de Xilinx. Il permet de visualiser des données internes au FPGA en liant le FPGA au PC à travers un câble JTAG (Joint Test Action Group). Aussi ChipScope permet de contrôler des signaux du design en y insérant deux modules internes : ICON "Integrated CONtroller" et un ILA "Integrated Logic Analyzer". Le module ICON est utilisé pour le contrôle et le module ILA pour la visualisation et l'affichage des signaux. Il est constitué de deux parties : ChipScope Core Generator et ChipScope. ChipScope Core Generator génère un composant VHDL que l'on insère dans le code. Il permet de connecter jusqu'à 255 signaux pour les visualiser. Une fois le code inséré dans le FPGA, on peut utiliser ChipScope pour télécharger les données du FPGA. Comme l'ordinateur et le FPGA n'ont pas même échelle de temps, on règle un trigger qui va se déclencher selon certains paramètres des signaux observés. Une fois le trigger déclenché, les données sont stockées dans les blocks RAM du FPGA qui jouent le rôle de buffers. Lorsque le buffer est plein, les données sont téléchargées sur l'ordinateur. On peut alors utiliser l'interface de CHIPSCOPE pour afficher les données en série (bit après bit) ou en parallèle (sous forme de bus) voire même tracer l'évolution temporelle des signaux [107].

4.2.3 Validation fonctionnelle

La Figure 4.3 représente le signal d'entrée après échantillonnage, nous avons utilisé l'outil ChipScope afin de visualiser le signal d'entrée numérisé auquel nous appliquons la FFT 256 points.

La fréquence maximale de fonctionnement de l'IP FFT proposée est supérieure à celle de l'échantillonnage, il est ainsi possible d'appliquer la FFT en temps réel. En effet, la cadence d'arrivée des entrées est inférieure à la fréquence maximale de fonctionnement de la FFT. Le résultat obtenu est visualisé avec ChipScope et illustré dans la Figure 4.4.

La Figure 4.4 représente les signaux de sortie de la FFT 256 points appliquée au signal d'entrée de la Figure 4.3. Nous utilisons un signal *trigger* afin de déclencher le début et la fin du fonctionnement de la FFT. Les signaux *sortie_re* et *sortie_im* représentent la partie réelle et imaginaire de la sortie FFT, respectivement. Nous calculons aussi l'amplitude en chaque point du spectre obtenu que nous le sauvegardons dans la variable *Amplitude* et nous le représentons dans la Figure 4.5. Cette amplitude revient à multiplier le vecteur de la sortie de la FFT par son conjugué. De plus, étant donné que la fonction log_{10} n'est pas synthétisable, nous avons exporté le vecteur d'amplitude afin de représenter le spectre du signal d'entrée. Ce dernier est représenté dans la Figure 4.6.

Nous remarquons qu'après une latence, qui correspond au temps de la sauvegarde des entrées dans une mémoire et du traitement nécessaire pour le calcul de la FFT, nous obtenons deux pics

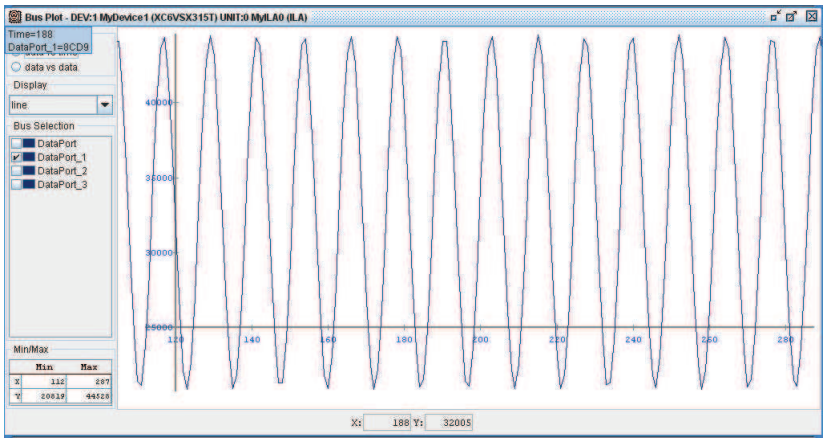


FIGURE 4.3 – Signal d'entrée

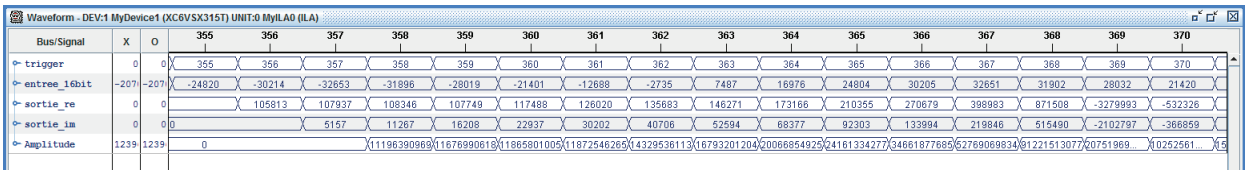


FIGURE 4.4 – Sortie de la FFT 256 points

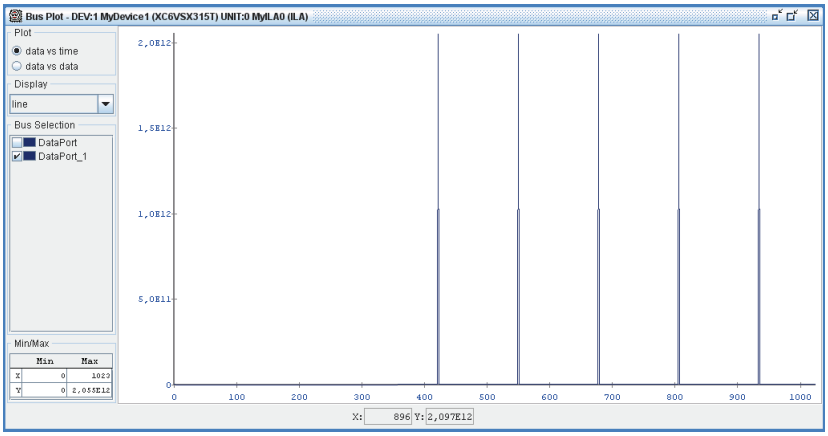


FIGURE 4.5 – Module du spectre de sortie de la FFT

pour chaque FFT de taille 256 points. En effet, il est nécessaire de mémoriser les échantillons fournis par le CAN. Une fois que les données sont sauvegardées dans une mémoire nous appliquons une FFT de taille 256 points et nous obtenons toutes les sorties après 356 cycles d'horloges.

Pendant que le calcul de la FFT se fait, nous réalisons l'acquisition des nouveaux 256 points pour les stocker dans la mémoire. Ainsi, la FFT conçue est capable de transformer les échantillons en temps réel. Ceci explique l'apparition des 5 pics symétriques sur le spectre de la Figure 4.5 (pour s'assurer de la clarté de l'illustration nous nous sommes arrêtés à 5 pics). Signalons ici que la latence de 356 cycles d'horloges n'affecte que les sorties de la 1^{ere} FFT.

Enfin, le calcul de la FFT en temps réel appliqué aux données numérisées par le CAN permet d'évaluer (en offline) ce convertisseur. En effet, les paramètres tels que le SNR "Signal to Noise Ration" ou le SFDR "Spurious Free Dynamic Range" [108] peuvent être déterminés, une fois que la conversion s'achève, en se basant sur le spectre du signal converti.

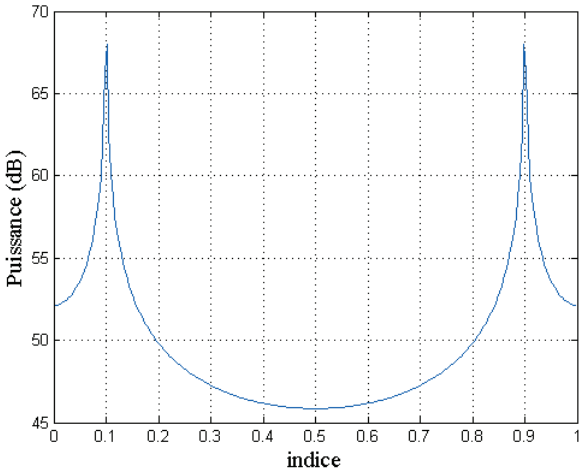


FIGURE 4.6 – Spectre de sortie de la FFT

4.3 Mesure de puissance

Dans cette section, nous nous intéressons à la puissance consommée par les IP FFT proposées. Afin d'effectuer cette mesure de puissance, nous utilisons un banc de test composé d'une carte FPGA Virtex-6 SX 315 T liée à un appareil de mesure de puissance (CPX 400A DUAL 60V 20A). Le banc de test utilisé est présenté sur la Figure 4.7.

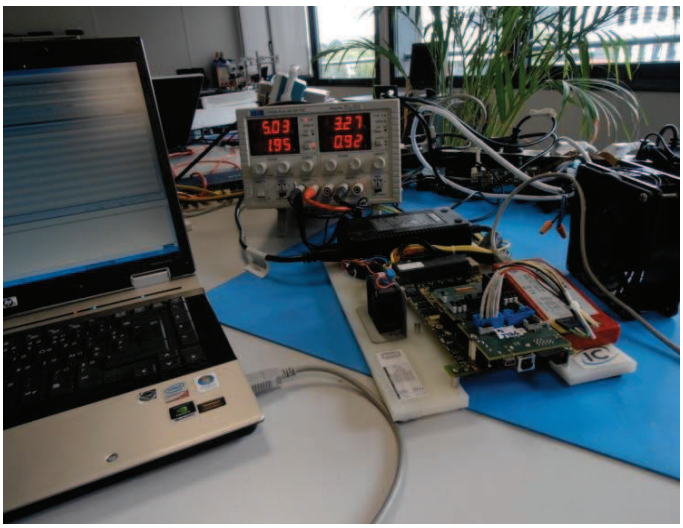


FIGURE 4.7 – Banc de test pour la mesure de puissance

L'appareil de mesure est lié à la carte FPGA via la sortie d'alimentation en fournissant une tension d'entrée égale à 5 V. En mesurant le courant de sortie de la carte nous pouvons ainsi déduire la puissance consomée P_{cons} en utilisant l'équation 4.1.

$$P = U \times I \tag{4.1}$$

Où P est la puissance consommée, U est la tension et I représente le courant.

Par ailleurs, afin de mesurer de la puissance consommée, il est nécessaire de mesurer la puissance à vide et par la suite mesurer la puissance après chargement du FPGA et déduire la puissance consommée en se basant sur l'équation 4.2.

$$P_{cons} = P_{tot} - P_{vide} \tag{4.2}$$

où P_{tot} est la puissance mesurée après un traitement, P_{vide} est la puissance mesurée à vide obtenu en mesurant la puissance consommée par le FPGA qui contient un algorithme de comptage.

D'autre part, sachant que la puissance dynamique mesurée est sensiblement inférieure à la puissance statique et afin d'avoir une mesure précise, nous avons opter pour une valeur moyenne de cette puissance. En effet, nous avons procédé à une duplication du traitement considéré dans la carte FPGA pour augmenter la valeur de la puissance dynamique. Enfin, pour retrouver la valeur de la puissance correspondant à un seul traitement, nous divisons la puissance consommée par le nombre de duplications.

Nous avons mesuré la puissance pour des tailles différentes de FFT en utilisant l'IP FFT proposée. Nous avons dupliqué cette IP afin d'avoir une mesure précise de la puissance consommée. Nous utilisons une IP DDS afin de générer les entrées pour chacune des IP. De plus, nous connectons une sortie de chacune des IP FFT dupliquées à ChipScope afin de s'assurer de son implantation.

Le courant à vide est de 1,95 A. La puissance à vide est ainsi égale à $5,03 \times 1,95 = 9,8085$ W. L'IP FFT 16 points a été dupliqué 100 fois. Le courant ainsi fournit est égal à 2.42 A. La puissance consommée par l'ensemble est ainsi égale à 12.1726 W. Si nous retirons la puissance à vide et nous divisons la puissance consommée par le nombre de duplications réalisées nous déduisons ainsi la puissance consommée par FFT 16 points qui égale à 23.4 mW. Nous avons réalisé la même opération pour différentes tailles de FFT de l'IP proposée et nous représentons les résultats obtenus sur la Figure 4.8. Aussi, nous avons réalisé les mêmes opérations précédemment citées sur l'IP FFT Xilinx afin de comparer les puissances consommées des différents IP.

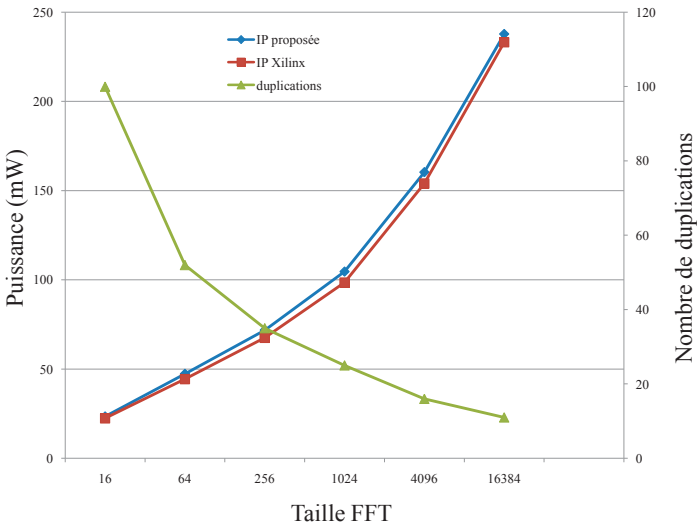


FIGURE 4.8 – Puissances mesurées de l'IP FFT proposée et celle de Xilinx

Il est montré que les consommations de puissance de l'IP proposée et celle de Xilinx sont très proches. Cependant, rappelons qu'en terme de rapidité, l'IP que nous proposons est trois fois

plus rapide que celle de Xilinx. De plus le débit fournit est 4 fois meilleur que celui de l'IP de Xilinx.

4.4 Applications de guerre électronique

4.4.1 La guerre électronique

Dans les conflits modernes, les maîtrises de l'information, de sa transmission et de son traitement sont devenues décisives. Avec l'essor prodigieux des télécommunications numériques et l'usage intensif des émissions radioélectriques dans les applications civiles et militaires, la guerre électronique (GE) tient maintenant une place majeure dans le dispositif de défense.

La guerre électronique consiste en l'exploitation des émissions radioélectriques d'un adversaire et, inversement, elle consiste à l'empêcher d'en faire autant. Il s'agit donc de toutes les opérations qui visent à acquérir la maitrise du spectre électromagnétique. On distingue trois principales composantes de la guerre électronique :

- Le renseignement d'origine électromagnétique.
- L'attaque électromagnétique
- La protection électronique

Le renseignement d'origine électromagnétique regroupe tous les systèmes passifs chargés de détecter, de localiser et d'identifier les émetteurs présents dans l'environnement par l'interception des ondes électromagnétiques. on parle aussi des ESM (Electronic Support Mesures). Les ESM qui sont passifs, permettent de voir sans être vu. Il existe de tels systèmes aussi bien pour les radars, les systèmes de communications que les systèmes optroniques. Dans le domaine radariste, les ESM sont souvent nommés détecteurs de radars.

L'attaque électromagnétique est la composante active de la guerre électronique. Elle consiste à empêcher un adversaire d'utiliser le spectre électromagnétique. Aussi appelée contre-mesure électroniques (Electronic Counter Measures), il s'agit pour l'essentiel de mesures de brouillage afin de rendre inexploitable les émissions, ou de leurrage dans le but de donner de fausses indications. Les missiles antiradars, les brouilleurs et les leurres constituent aujourd'hui les principaux outils de l'attaque électromagnétique. Elle inclut également l'usage des armes à énergie dirigée et particulièrement l'arme hyperfréquence afin de neutraliser voir de détruire les systèmes électroniques adverses.

Enfin, la protection est une composante très diversifiée de la GE. Elles incluent tous les dispositifs et toutes les procédures permettant de contrer les attaques électroniques de l'adversaire. Nous parlons aussi du contre contre-mesures électroniques (Electronic Counter Counter Measures). Dans ce travail nous nous intéressons à la composante passive de la GE, l'ESM.

4.4.2 Les ESM

L'objectif principal des ESM est la description de l'environnement électromagnétique de la cible qui le transporte. Pour cela, il assure trois fonctions : détecter, analyser et identifier.

- Détecter : tout signal capté provenant d'une source électromagnétique volontaire, doit être extrait du bruit et mesuré pour pouvoir faire l'objet de traitements.
- Analyser : les signaux provenant de radars différents doivent d'abord être séparés les uns des autres, puis caractérisés par un certain nombre de variables descriptives.
- Identifier : les caractéristiques des signaux radars interceptés sont comparées à une base de données. Il s'agit en particulier de connaître les porteurs de ces radars (avions, missiles, bateaux), et leur degré d'hostilité (ami, ennemi, neutre).

Suivant les missions dévolues aux ESM, leur conception est un compromis entre trois fonctions :

- Réactivité : le temps de réaction à l'apparition d'une nouvelle émission.
- Couverture : l'exhaustivité de la description de l'environnement.
- Précision : la qualité de cette description.

En fonction de l'importance accordée à chacune de ces fonctions, on distingue schématiquement trois familles de systèmes, les ESM d'alertes, les ESM de tenue de situation et les ESM de

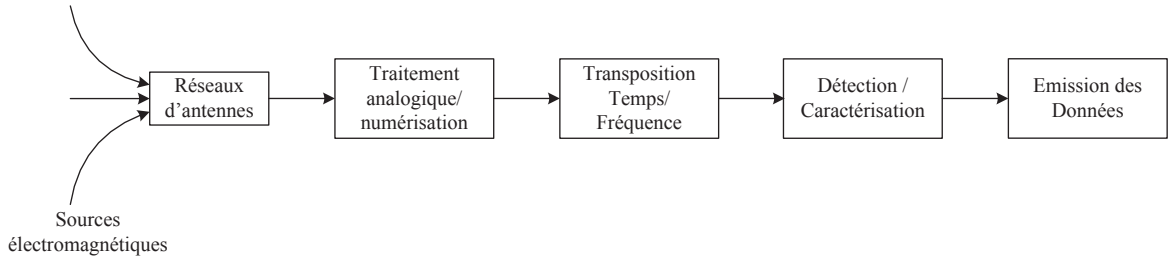


FIGURE 4.9 – Récepteur de guerre électronique

renseignement électronique (ELINT pour ELectronic INTelligence).

Les ESM d'alerte accordent une grande importance à la réactivité. Néanmoins, la couverture ne concerne que des types de radars répertoriés comme menaçants. La précision de tels systèmes reste relativement faible en comparaison des autres types ESM qui suivent.

Les ESM de tenue de situation n'acceptent qu'une réactivité moyenne. La principale contrainte est imposée à la couverture de l'environnement qui doit être totale. La précision réclamée à ces systèmes est en générale assez bonne par rapport à celle des ESM d'alertes.

Les ESM d'ELINT ont une réactivité faible. La couverture ne concerne que quelques cibles bien définies. La principale exigence porte sur la précision de la description de ces cibles.

4.4.3 La détection en guerre électronique

4.4.3.1 Principe de la détection

L'architecture de fonctionnement d'un ESM est donnée sur la Figure 4.9.

Les antennes transforment les signaux électromagnétiques en signaux électriques hyperfréquences. Suivant le type de traitement désiré, on peut trouver des antennes capables de couvrir de larges bandes de fréquence comme pour la détection en guerre électronique pour les récepteurs WB "Wide Band" ou des bandes plus étroites comme pour les récepteurs superhétérodynes (SH). Quelque soit le type de réception, une transformation du domaine temporel au domaine fréquentiel est réalisée. Cette transformation est effectuée en utilisant des FFT. A ce moment, l'ESM met en œuvre les moyens d'analyse et d'identification du signal reçu [109]. Dans la guerre électronique,les radars transmettent dans une bande allant de 2 à 18 GHz. Pour s'assurer de la qualité de détection, les deux récepteurs WB et SH sont utilisés. Le principe de fonctionnement de cette application est donnée sur la Figure 4.10.

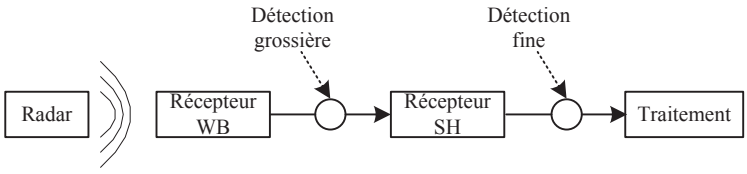


FIGURE 4.10 – Principe de la détection radar

Afin de réaliser une détection et identification des signaux radar reçus, ces derniers doivent passer par deux récepteurs comme il est montré sur la Figure 4.10. Le premier est un récepteur large bande qui permet de concentrer la bande de fréquence en une bande plus étroite autour du signal utile. on parle ainsi d'une détection grossière. Par la suite, un deuxième récepteur de type superhétérodyne est utilisé afin de permettre une meilleure sélectivité et par conséquent une meilleure détection. on parle ainsi de détection fine qui a pour but de confirmer la détection grossière.

Nous détaillons dans les sections suivantes le principe de fonctionnement de ces deux récepteurs.

4.4.3.2 Le récepteur large bande

Les récepteurs large bande sont des récepteur qui permettent de recevoir des signaux à large bande de fréquence. Ces récepteurs permettent de couvrir la bande de 2 à 18 GHz pour les applications de guerre électronique. Le synoptique expliquant le fonctionnement de ces récepteurs est donné par la Figure 4.11.

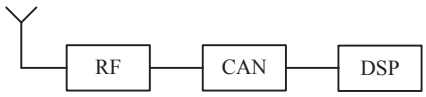


FIGURE 4.11 – Architecture du récepteur large bande

Ces récepteurs sont composés d'un bloc radiofréquence analogique RF, d'un convertisseur analogique numérique et d'un bloc de traitement numérique (DSP). Le bloc RF assure le filtrage, l'amplification et la transposition de la fréquence du signal vers la bande de base. Cette transposition fréquentielle permet au CAN de numériser des signaux de dynamique plus faible que les signaux RF reçu ce qui facilite cette tâche de numérisation. Finalement, un DSP est utilisé afin de traiter les signaux reçus afin de les identifier et par la suite les restituer.

4.4.3.3 Le récepteur superhétérodyne

Inventé par L. Lévy en 1917, le récepteur superhétérodyne est connu pour ses bonnes performances en termes de sélectivité et de sensibilité. Ces récepteurs sont utilisés dans les systèmes radar afin de convertir les signaux radiofréquences reçus par l'antenne en signal vidéofréquences pour en extraire les données sur les échos de retour des cibles. La Figure 4.12 montre le schéma bloc d'un récepteur superhétérodyne.

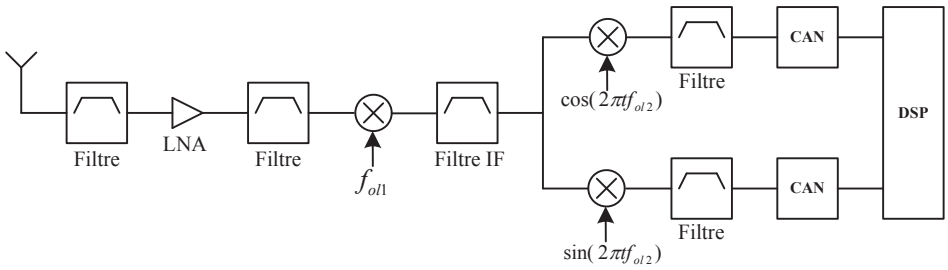


FIGURE 4.12 – Architecture du récepteur superhétérodyne

Le signal de radiofréquence (RF) venant de l'antenne passe par un filtre qui ne laisse passer que la gamme de fréquences désirée suivi d'un amplificateur à faible bruit LNA. Le signal passe ensuite dans un mélangeur qui l'ajoute à une onde produite par un oscillateur local stable. Le principe de ce mélangeur est donné par l'équation 4.3.

$$\begin{aligned} f_{FI} &= f_{ol_1} - f_0 \\ cos(2\pi f_{ol_1} t) \times cos(2\pi f_0 t) &= \frac{1}{2}[cos(2\pi(f_{ol_1} + f_0)t + cos(2\pi f_{FI} t)] \end{aligned} \tag{4.3}$$

où f_{ol_1} représente la fréquence produite par l'oscillateur local, f_0 est la fréquence d'entrée à l'oscillateur dite fréquence image et f_{FI} est la fréquence produite par le mélangeur dite fréquence intermédiaire. Le filtre de fréquence intermédiaire permet de ne garder que la fréquence désirée du signal de sortie du mélangeur. En suite nous utilisons un second oscillateur cette fois à fréquence variable f_{ol_2} pour transposer la fréquence centrale du canal souhaité envers la bande de base. Par suite un second filtre IF est utilisé pour réaliser un pré-filtrage avant le second mélangeur qui permet la transposition du signal en bande de base. Mais il est préférable de décomposer le signal sur deux voies I et Q pour ne pas perdre d'informations.

Cette transposition est réalisée à l'aide de deux mélangeurs en quadrature de phase. Et enfin la sélection de canal se fait d'une façon analogique sur les deux canaux avant la numérisation du signal.

Bien que la technique superhétérodyne présente l'inconvénient majeur d'être encombrante réduisant ainsi l'intégration du récepteur. Elle présente plusieurs avantages tel que une bonne sélectivité des canaux et bonne sensibilité de réception [110].

4.4.4 Implantation de la détection

Après avoir reçu et numériser le signal radar en utilisant un récepteur large bande en premier lieu et un récepteur superhétérodyne. La deuxième phase des applications radar consiste à restituer ce signal. Cette étape consiste à appliquer une transformée de Fourier au signal numérique envoyé par le récepteur. En suite, l'amplitude du module de la FFT doit être calculé afin de déterminer les caractéristiques du signal et par la suite le restituer. Nous proposons ainsi d'utiliser l'IP FFT proposée afin d'implanter l'application de restitution sur FPGA.

Dans cette section, nous proposons d'abord une validation expérimentale de l'application de restitution et nous montrons par la suite l'avantage de l'IP proposée par rapport à celui de Xilinx.

4.4.4.1 Validation expérimentale

La restitution d'un signal consiste à appliquer dans un premier temps une FFT sur le signal numérique. La taille de la FFT dépend de la donnée d'observation et des fréquences mises en jeu. Par la suite, nous calculons l'amplitude en chaque point du spectre obtenu. A partir de l'indice de l'amplitude maximale du spectre, nous pouvons déduire la fréquence du signal d'entrée. L'architecture que nous avons implantée pour réaliser cette application est donnée sur la Figure 4.13.

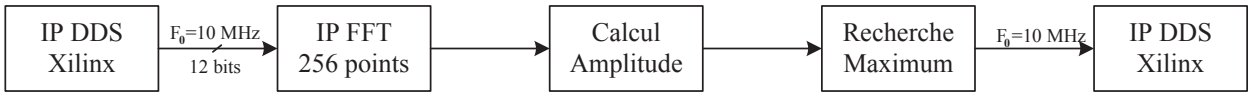


FIGURE 4.13 – Architecture de l'application de restitution

Le signal d'entrée que nous utilisons pour la réalisation de cette application est généré en utilisant l'IP DDS de Xilinx [111]. En effet, à partir d'une fréquence d'entrée choisie, cette IP permet générer un signal sinusoïdal. Nous choisissons ainsi une fréquence d'entrée égale à $10MHz$. En effet, l'application de restitution exposée dans la Figure 4.13 permet de déterminer la valeur de cette fréquence. Nous appliquons une FFT de taille 256 points au signal d'entrée. Par la suite, nous calculons l'amplitude du module de la FFT à chaque point du spectre obtenu. Le calcul de l'amplitude consiste à multiplier chaque valeur du spectre par son conjugué. Ensuite, Nous cherchons le maximum de ces amplitudes ainsi que son indice. Pour cela, nous supposons que le maximum est d'abord fixé à la première valeur et que son indice est fixé à 1 (ceci est considéré comme référence). Ensuite, si la valeur suivante est supérieur à la référence, alors cette valeur est considérée comme nouvelle référence, nous incrémentons l'indice et nous passons à la valeur suivante et ce jusqu'à la fin du parcours de tout le spectre. A la fin de ce parcours nous obtenons la valeur maximale ainsi que son indice. A partir de l'indice obtenu, nous pouvons déduire la fréquence du signal d'entrée. En effet, cette fréquence peut être trouvée en utilisant l'équation 4.4 :

$$Indice = F_0 \times \frac{N}{Fe} \tag{4.4}$$

avec N est le nombre de points de la FFT.

Nous avons visualisé le signal d'entrée généré par l'IP DDS en utilisant ChipScope et nous représentons le résultat sur la Figure 4.14.

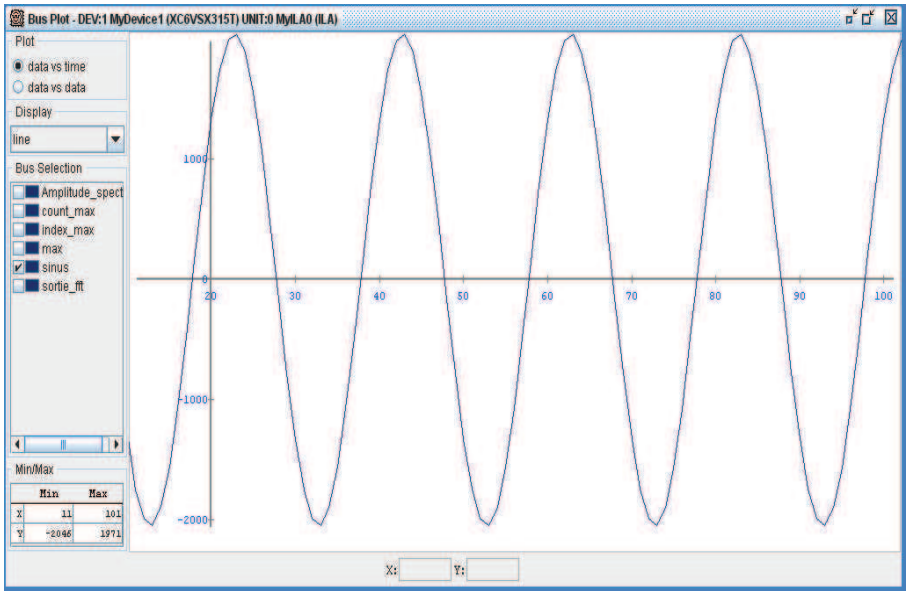


FIGURE 4.14 – Signal de sortie de l'IP DDS

Une FFT de taille 256 points a été appliquée à ce signal. Nous calculons l'amplitude à chaque point, nous cherchons la valeur maximale ainsi que l'indice de cette valeur et nous visualisons le résultat en utilisant ChipScope comme il est montré dans la Figure 4.15.

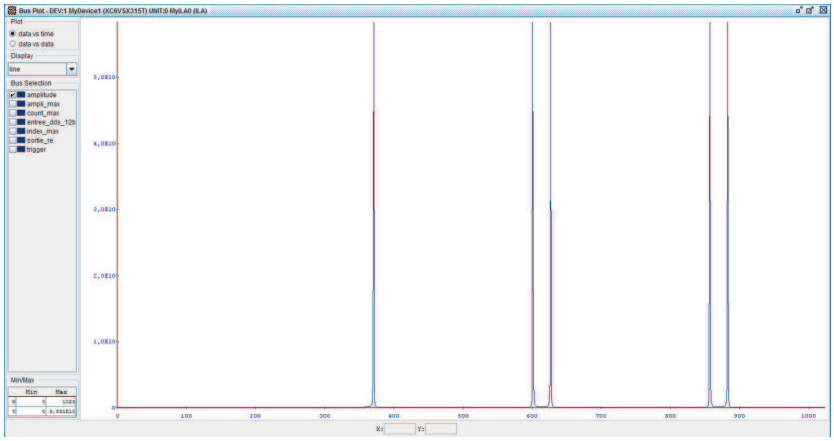


FIGURE 4.15 – Amplitude du spectre de sortie

La Figure 4.16 représente l'amplitude du spectre obtenu ainsi que des valeurs internes qui nous aiderons à déterminer la valeur de la fréquence F_0. En effet, *trigger* représente le trigger utilisé pour déclencher le début et la fin du traitement, *entree_dds_12bit* représente le signal généré par l'IP DDS de Xilinx et utilisé comme entrée à l'IP FFT proposée et implanté sur FPGA. Le signal *sortie_re* représente la partie réelle de la sortie de la FFT, le signal *amplitude* est l'amplitude calculé en chaque point du spectre. Nous utilisons un signal que nous le nommons *ready* pour indiquer l'instant auquel la première valeur des amplitudes est prête, *count_max* est le compteur que nous utilisons pour parcourir la mémoire contenant les amplitudes des spectre. Le signal *index_max* contient l'indice de la valeur maximale des amplitudes. Cette dernière est sauvegardée dans un signal que nous l'appelons *max*.

Rappelons que les résultats de la FFT sont disponibles après 356 cycles d'horloges. Ces résultats sont conformes à ceux obtenus sur FPGA et visualisé sous ChipScope dans la Figure

4.16.

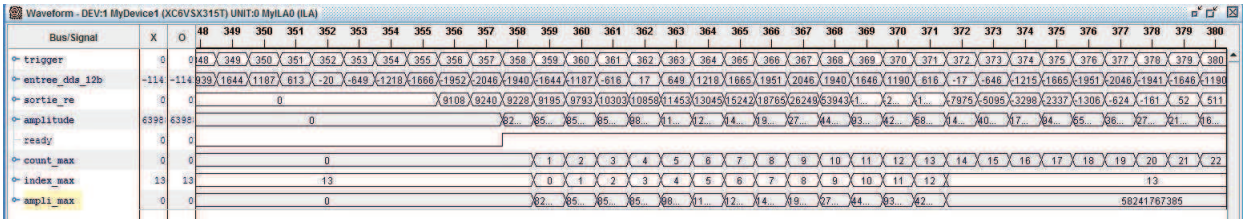


FIGURE 4.16 – Calcul du maximum des amplitudes du spectre

Nous pouvons remarquer que *index_max* affiche la valeur 13 et ce jusqu'à la fin de calcul de tous les énergies du spectre. L'indice obtenu est 13 et par conséquent nous pouvons utiliser cet indice pour restituer le signal d'entrée en trouvant sa fréquence :

$$F_{out} = Fe \times \frac{indice}{N} = 200 \times \frac{13}{256} = 10,1MHz \tag{4.5}$$

Nous pouvons ainsi restituer le signal d'origine en utilisant une IP DDS Xilinx.

4.4.4.2 Limites de l'utilisation de l'IP de Xilinx

Après la validation expérimentale du principe de fonctionnement, nous souhaitons analyser dans cette section les limites de l'utilisation de l'IP FFT de Xilinx et l'apport de l'IP FFT proposée dans un contexte de détection radar.

Dans certaines applications de guerre électronique notamment l'autoprotection, la fréquence d'échantillonnage du signal reçu est très élevée. Pour modéliser une application de GE nous utilisons un CAN de type ADC12D1000 ou bien ADC12D1600 de Texas Instrument. Ces convertisseurs échantillonnent un signal analogique avec une fréquence égale à 1,25 GHz. Chaque échantillon est codé sur 12 bits. Le convertisseur effectue une première phase de démultiplexage avant d'envoyer les sorties. En effet, les signaux de sorties de ce type de convertisseur est une concaténation de 4 échantillons successifs. Par conséquent, la fréquence de sortie est égale à $\frac{1,25GHz}{4} = 312,5$ MHz et les signaux de sortie sont ainsi codés sur 48 bits.

Dans le cas de l'utilisation de l'IP FFT de Xilinx dans une application de GE plusieurs problèmes majeurs se présentent. En effet, les données fournies par le CAN avec une fréquence de 312,5 MHz et codés sur 48 bits chacun nécessitent une première phase d'extraction afin de séparer les 4 échantillons du CAN. Cette phase d'extraction peut être réalisée de deux manière différentes. La première consiste à fournir 4 échantillons de 12 bits en parallèle par cycle d'horloge avec une fréquence de 312,5 MHz comme il est montré sur la Figure 4.17. Il est ainsi indispensable d'utiliser une architecture FFT avec 4 entrées en parallèle. Avec l'IP Xilinx, nous avons la possibilité de choisir une de ses architectures récursives tels que Radix-4 Burst I/O, Radix-2 Burst I/O et Radix-2 Lite Burst I/O. Malgré que ces architectures permettent d'avoir plusieurs entrées en parallèles et fournir plusieurs sorties en parallèles, l'inconvénient majeur de ces architectures réside dans le fait qu'elles ne permettent pas un traitement continue. En effet, comme nous l'avons expliqué dans la section 2.4.2, ce type d'architecture utilise une seule mémoire et un seul bloc de traitement, par conséquent, le traitement en temps réel n'est pas réalisable avec ce type d'architecture.

La deuxième manière d'extraction est donnée sur la Figure 4.18. Dans ce cas, nous sérialisons les sorties du CAN en les envoyant sur un seul bus de 12 bits avec une fréquence de 1,25 GHz. Dans ce cas nous pouvons utiliser l'architecture Pipelined Streaming I/O de Xilinx afin d'avoir un traitement en continu des données. Cependant, quelque soit le type du FPGA que nous souhaitons utiliser, la fréquence maximale de fonctionnement de la FFT ne peut pas atteindre la cadence d'arrivée des données. Par conséquent, le traitement des données en temps réel en utilisant cette architecture est aussi irréalisable.

Contrairement à ce qui a été proposé par Xilinx, l'IP FFT conçue dans cette thèse permet de traiter 4 voies en parallèle d'une manière continue. Ce qui constitue une solution intéressante

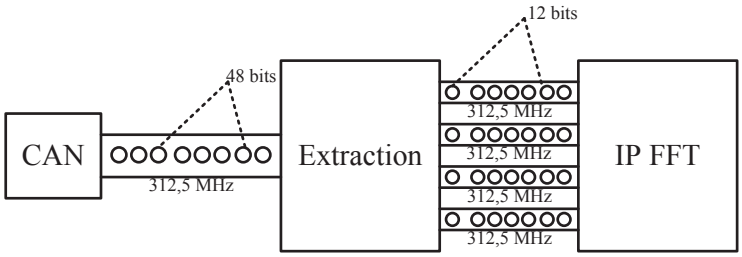


FIGURE 4.17 – Extraction parallèle des sortie du CAN

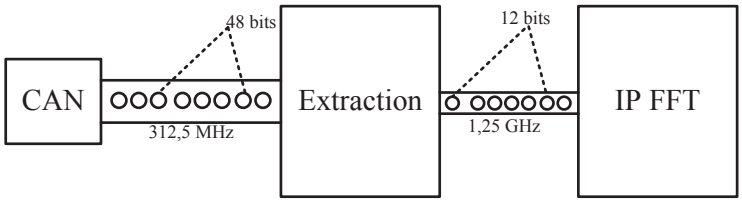


FIGURE 4.18 – Extraction série des sortie du CAN

pour réaliser le traitement en temps réel du flux de données reçu par le CAN. En effet, si nous utilisons l'architecture d'extraction présentée sur la Figure 4.17, nous pouvons appliquer à ces sorties notre IP FFT à entrées parallèles et sorties parallèles. Ceci est possible grâce à la fréquence de fonctionnement de la FFT qui est de l'ordre de 320 MHz. Cette fréquence est supérieure à la cadence avec laquelle les données arrivent.

Afin de valider le fonctionnement de notre IP pour des données échantillonnées à une fréquence très élevée et faute de disponibilité du matériel nécessaire (CAN), nous utilisons comme entrée une mémoire de 64 points à 48 bits chacun. Nous essayons d'appliquer notre IP FFT de taille 256 points. L'architecture proposée est ainsi donnée sur la Figure 4.19

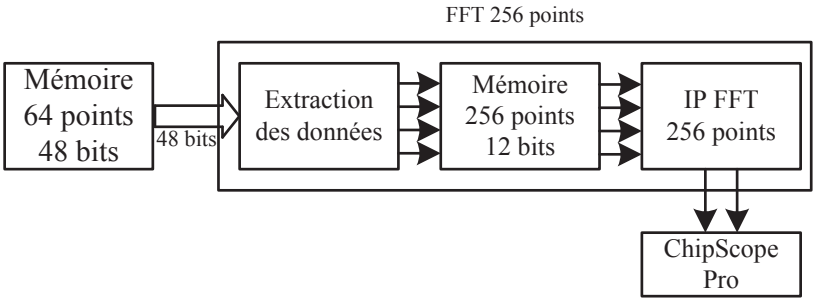


FIGURE 4.19 – Architecture de la FFT proposée

Chaque case de la mémoire utilisée contient une concaténation de 4 échantillons de 12 bits chacun. Ces échantillons forment un signal sinusoïdal que nous avons généré en utilisant l'outil Matlab. La première phase de ce système consiste donc à réaliser une extraction des données en envoyant 4 échantillons de 12 bits chacun. Tous ces signaux doivent être sauvegardés dans une mémoire de taille 256 points si nous souhaitons appliquer une FFT de taille 256 points. Par la suite, nous utilisons l'IP FFT à entrée parallèle et sortie parallèle que nous avons proposée afin de calculer la transformée de Fourier du signal d'entrée. Finalement nous calculons le module en chaque point du spectre obtenu et nous affichons le résultat en utilisant l'outil ChipScope Pro.

Cette phase a été validée avec succès, et les résultats obtenus sont semblables à ceux illustrées sur les Figures 4.15 et 4.16

4.5 Conclusion

Dans ce chapitre nous avons validé dans un premier temps l'architecture de l'IP FFT proposée en utilisant un banc de test. Par la suite nous avons réalisé une mesure de la puissance consommée de notre IP FFT ainsi que de l'IP FFT Xilinx. Cette mesure a montré que l'IP FFT proposée présente une consommation très similaire à celle obtenue par l'IP de Xilinx. Toutefois, notre IP FFT est plus performante en terme de rapidité. Dans un deuxième temps, nous avons utilisé notre IP FFT pour implanter des applications de guerre électronique. Les résultats obtenus après cette implantation ont montré le bon fonctionnement de cette IP ainsi que son avantage par rapport à l'IP FFT de Xilinx en traitant des données échantillonnées à une fréquence de 1,25 Gega samples/sec en temps réel ce que n'est pas réalisable avec les IP commerciaux.

Chapitre 5

Implantation numérique des algorithmes de reconnaissance des formes

Sommaire

5.1 Introduction

Dans ce chapitre, nous présentons une deuxième application pour l'utilisation de l'IP FFT proposée. Cette application consiste à implanter numériquement une méthode de reconnaissances de visages à base de la corrélation. En effet, la corrélation est une des techniques de reconnaissance de formes les plus performantes. Cependant, malgré qu'elle permet de fournir une décision fiable, le nombre de corrélations à réaliser peut être très grand et par conséquent le temps de prise de décision est très élevé. Pour faire face à ce problème, nous proposons de tirer profit de la richesse considérable des cibles programmables en proposant deux implantations différentes. La première consiste à proposer une nouvelle architecture implantant la corrélation en utilisant une carte FPGA. La deuxième consiste à proposer une implantation GPU d'un nouvel algorithme qui permet de limiter le nombre de corrélations à réaliser sans pour autant baisser la qualité décisionnelle de la corrélation.

Nous commençons ce chapitre par rappeler le principe d'un point de vue algorithmique de la technique de corrélation utilisée pour identifier un visage. Ce rappel algorithmique a pour but de décrire les différents blocs de la méthode de corrélation (opérations mathématiques, transformations, etc.) pour en proposer une implantation toute numérique du corrélateur. Cette implantation a été réalisée sur les deux cibles étudiées précédemment : le FPGA et le GPU. Pour l'implantation sur FPGA, nous avons proposé une architecture basée sur le calcul de la

FFT/IFFT 2D en utilisant l'IP FFT/IFFT 1D (section 2.5 du chapitre 2). Pour l'implantation GPU, nous avons utilisé les fonctions de la FFT existantes pour proposer un nouvel algorithme de reconnaissances de visages. L'un des avantages de cet algorithme consiste à réduire le nombre de corrélations à réaliser afin réduire le nombre de comparaisons à réaliser dans le cas d'une base de donnée très grande.

5.2 Algorithmes

5.2.1 Corrélation

L'algorithme de base de la corrélation, appelé corrélateur VLC, est présenté sur la Figure 5.1. Dans ce travail, nous évitons cette notation " VLC ". En effet, elle est souvent associée à une implantation optique et nous utiliserons plutôt le terme "algorithme de corrélation". Cet algorithme est constitué de sept blocs réalisant chacun une tâche bien spécifique. Ainsi, nous pouvons définir l'algorithme de corrélation comme l'exécution de tous ces blocs dans l'ordre (Figure 5.1).

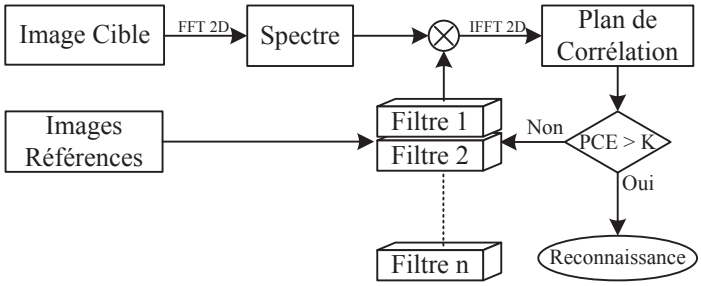


FIGURE 5.1 – Schéma algorithmique du corrélateur numérique

Le premier bloc consiste à charger la scène cible (qui contient le visage cible : visage à reconnaître pour une application de détection de visages) à partir d'un fichier ou prise directement avec une caméra. Ensuite, nous réalisons son spectre en utilisant la transformation de Fourier numérique. Après le chargement du filtre de corrélation (fabriqué à l'avance à partir d'une image référence), nous multiplions ce dernier avec le spectre de l'image cible. Le principe de fabrication des filtres de corrélation sera détaillé dans la section suivante. L'opération suivante consiste à réaliser une transformation de Fourier inverse de ce produit. Pour cela, nous utilisons l'algorithme de la transformée de Fourier inverse IFFT. Ensuite, nous calculons l'amplitude du plan de corrélation pour déterminer le PCE. Finalement, une décision est prise selon la valeur du PCE : il y a reconnaissance si seulement sa valeur est plus grande qu'un seuil défini à l'avance. Dans ce cas, l'algorithme est arrêté, et l'image cible est identifiée comme étant identique à l'image référence qui a servie à fabriquer le filtre. Dans le cas contraire (la valeur du PCE est plus petite que le seuil fixé), l'image n'est pas reconnue et un autre filtre de corrélation est ainsi chargé. Les opérations de multiplication, IFFT, calcul du PCE et prise de décision sont répétées jusqu'à la reconnaissance de l'objet cible ou bien jusqu'à l'épuisement de tous les filtres de la base considérée. Il est à noter que la valeur du seuil est déterminée expérimentalement en prenant en compte tous les résultats de corrélation obtenus avec toute la base de sorte à minimiser les fausses alarmes.

5.2.2 Filtres

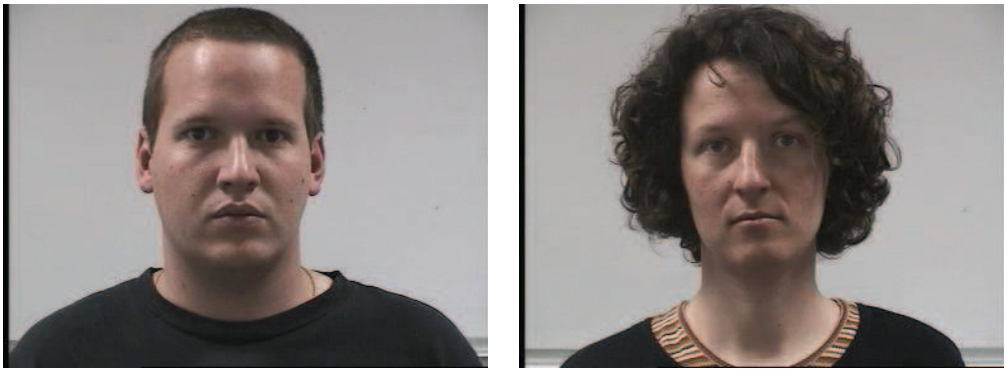
5.2.2.1 Filtre adapté

Le filtre adapté est le premier filtre de corrélation qui a été utilisé. Il correspond au conjugué du spectre de l'image référence issue d'une base d' images. Désignons par R la transformée de Fourier de la référence r, le filtre adapté de corrélation est décrit de la façon suivante :

$$H_{adapte}(u,v) = R^*(u,v) \tag{5.1}$$

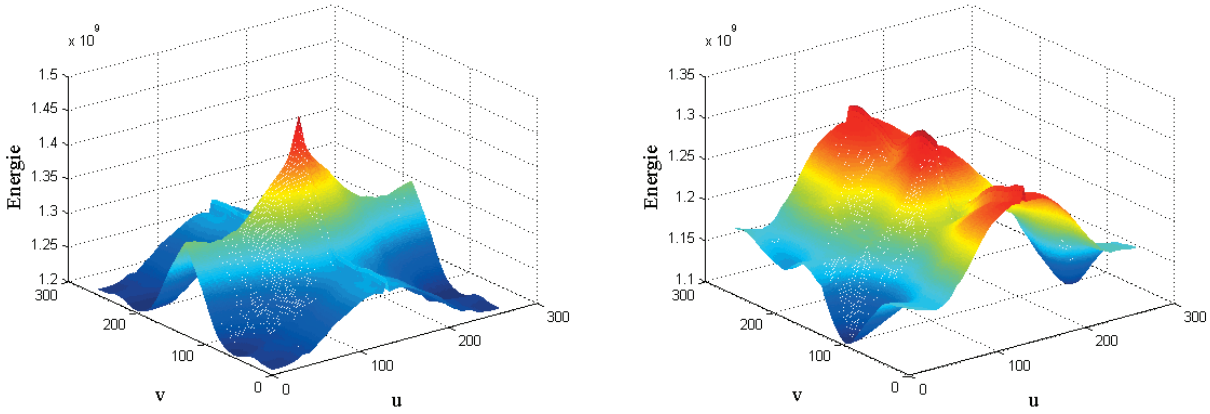
où H_{adapte} est le filtre adapté et u, v sont les coordonnées du filtre dans le plan de Fourier.

En considérant l'image présentée sur la Figure 5.2(a) comme l'image référence (utilisée pour fabriquer le filtre de corrélation adapté), nous avons effectué numériquement (en utilisant l'algorithme de la Figure 5.1) en premier lieu, une auto-corrélation, ensuite nous corrélons le filtre fabriqué avec l'image 5.2(a) avec l'image présentée sur la Figure 5.2(b). Les résultats obtenus sont illustrés dans la Figure 5.3. En effet, la Figure 5.3(a) contient un pic situé au centre du plan de corrélation qui indique la similitude entre l'image cible et l'image référence. Contrairement au deuxième plan de corrélation, présenté dans la Figure 5.3(b), ce dernier ne contient pas de pic étant donné qu'il n'y a pas de similitude entre les deux images.



(a) (b)

FIGURE 5.2 – Exemples d'images utilisées pour les simulations



(a) (b)

FIGURE 5.3 – Résultats de simulations de la corrélation en utilisant le filtre adapté

L'inconvénient majeur de ce filtre réside dans sa faible discrimination. De plus, la largeur du pic de corrélation rend très difficile la localisation du pic de corrélation [3].

5.2.2.2 Filtre POF

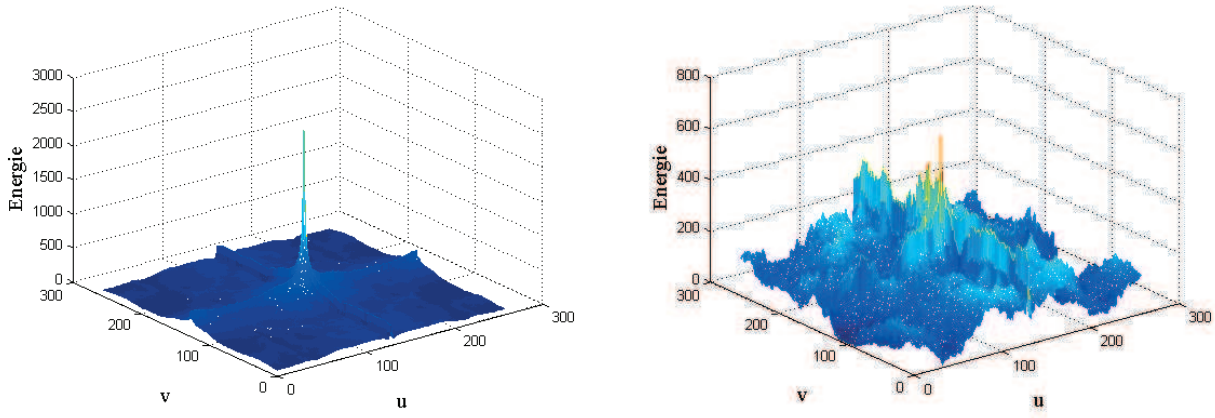
Optiquement et afin d'améliorer les performances du corrélateur utilisant le filtre adapté, plusieurs travaux ont été effectués en s'appuyant entre autres, sur les progrès dans le domaine des modulateurs spatiaux de lumière (SLM) utilisés pour afficher optiquement le filtre en temps réel. En effet, Horner [8] s'appuie sur le constat qui stipule que l'information contenue dans la phase de la transformée de Fourier d'un objet est beaucoup plus importante que celle contenue dans son amplitude [10]. Ainsi, il a suggéré l'utilisation un filtre uniquement de phase. Ce dernier permet ainsi une meilleure utilisation de l'efficacité optique : un paramètre important pour une implantation optique. Ainsi, il propose un filtre de phase pure POF "Phase-only Filter" qui

est un filtre adapté dont l'amplitude a été mise à 1 en tout point. Pour se faire, on divise le conjugué du spectre de l'image référence par son module à chaque point. Ce filtre est connu par son pouvoir de discrimination important par rapport au filtre adapté ainsi que pour sa simplicité d'implantation [3]. De la même manière que le filtre adapté, le filtre H fabriqué à partir de la référence r est donné par l'équation :

$$H_{POF}(u,v) = \frac{R^*(u,v)}{|R(u,v)|} \tag{5.2}$$

Où R est le spectre de l'image référence et R^* son conjugué.

Pour montrer l'efficacité de cette famille de filtres, nous réalisons les corrélations entre les images de la Figure 5.2 avec un filtre POF fabriqué a partir de l'image de la Figure 5.2(a). Dans ce cas nous obtenons les plans de corrélations présentés sur la Figure 5.4. En (a), nous avons une corrélation entre deux images identiques, tandis qu'en (b), une corrélation entre une image cible et une image référence différentes. Si nous comparons les plans de corrélation des deux filtres (adapté et POF : Figure 5.3 et Figure 5.4), nous constatons que l'utilisation du filtre POF permet d'éliminer la quasi totalité du bruit présent dans le plan de corrélation du filtre adapté. Ainsi, nous avons un pic avec une forte intensité lorsqu'il s'agit de deux images identiques. En effet, le pic d'énergie de la Figure 5.4(a) est 3 fois supérieur à celui de la Figure 5.4(b) alors que celui présenté dans la Figure 5.3(a) est 1,2 fois supérieur à celui de la Figure 5.3(b). Ceci nous offre une souplesse pour positionner le seuil utilisé pour confirmer ou infirmer la reconnaissance.



(a) (b)

FIGURE 5.4 – Résultats de simulations de la corrélation en utilisant le filtre POF

5.2.2.3 Filtre BPOF

Le filtre BPOF (Binary POF), introduit par Horner [112], est une version binaire du filtre de phase pure. Ce filtre représente une solution intéressante pour une implantation matérielle de la corrélation car nous avons besoin d'un nombre de bits réduit pour coder chacun de ses pixels.

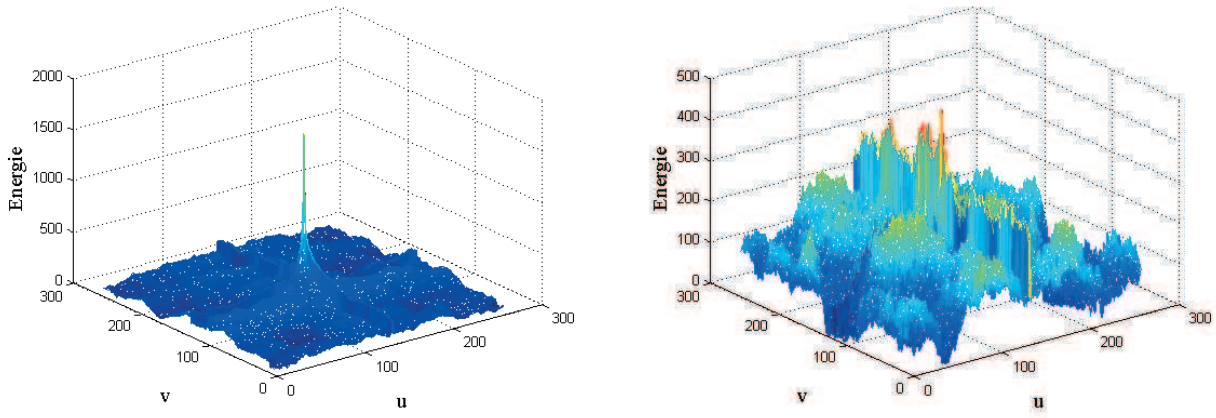
Considérons R la transformée de Fourier de la référence r. Le filtre BPOF est défini par :

$$H_{BPOF} = B[H_{POF}] \tag{5.3}$$

Cette binarisation est donnée par

$$H_{BPOF} = \begin{cases} 1 & \text{si } Real(H_{POF}) > 0 \\ -1 & \text{sinon} \end{cases} \tag{5.4}$$

L'opérateur *Real* désigne la partie réelle d'un nombre complexe. La Figure 5.5 illustre les résultats des corrélations obtenues entre les images présentées sur la Figure 5.2 en considérant la Figure 5.2(a) comme référence. Ces plans de corrélations ressemblent à ceux obtenus en utilisant



(a) (b)

FIGURE 5.5 – Résultats de simulations de la corrélation en utilisant le filtre BPOF

le filtre POF. Ainsi, le filtre BPOF permet la même souplesse et efficacité de reconnaissance que le filtre POF. De plus, il présente une grande simplicité de programmation, vu que la réponse du filtre ne contient que deux valeurs : 1 et -1.

5.2.2.4 Filtre composite

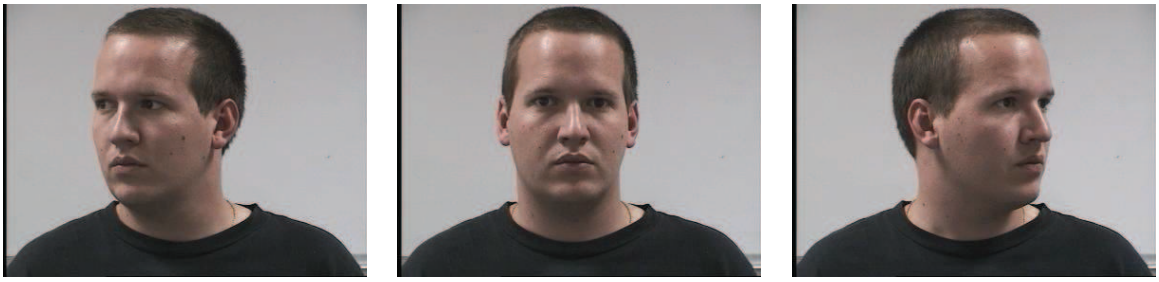
Dans la réalité, l'image à reconnaitre peut être déformée (échelle, rotation, déformation de l'objet, etc). Pour faire face à ces distorsions possibles, la corrélation doit être effectuée avec plusieurs références de la base d'apprentissage [5]. Par conséquent, le nombre de corrélations à réaliser augmente et ainsi le temps d'exécution sera très important.

Afin de réduire la base de référence utilisée dans la corrélation plusieurs travaux se sont focalisés sur ce sujet. Parmi les solutions proposées nous pouvons citer le filtre composite proposé par [11, 12].

Ce filtre consiste à effectuer une combinaison linéaire d'un ensemble de filtres H_i (pondérés ou pas) définis à partir d'une phase préliminaire qui consiste à choisir la base d'apprentissage.

$$H_{composite} = \sum H_i \tag{5.5}$$

Dans cette partie, nous fabriquons un filtre composite en utilisant trois filtres POF construits à partir des images de la Figure 5.6, et nous réalisons une corrélation entre ce filtre et chacune des images utilisées pour sa fabrication. Les résultats obtenus, présentés sur la Figure 5.7, montrent l'existence d'un pic de corrélation pour chacune de ces images. En dépit de son utilité, puisqu'il permet de réduire le nombre d'images références utilisées, le filtre composite présente plusieurs problèmes liés à la saturation causée par l'addition d'un certain nombre de filtres pour un nombre de bits fixe.



(a) (b) (c)

FIGURE 5.6 – Images utilisées pour la fabrication du filtre composite et segmenté

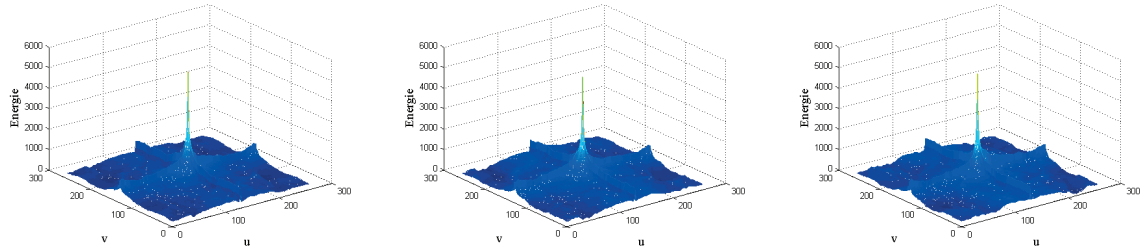


FIGURE 5.7 – Résultats de simulations de la corrélation en utilisant le filtre composite

5.2.2.5 Filtre segmenté

Le filtre segmenté, proposé par Alfalou et al [13], est une version optimisée du filtre composite afin d'éviter les problèmes de saturation causés par la combinaison linéaire d'un nombre important de références.

Le principe du filtre segmenté consiste à diviser le plan de Fourier du filtre en plusieurs zones. Chaque zone sera allouée à un élément d'une des images utilisées pour la construction du filtre.

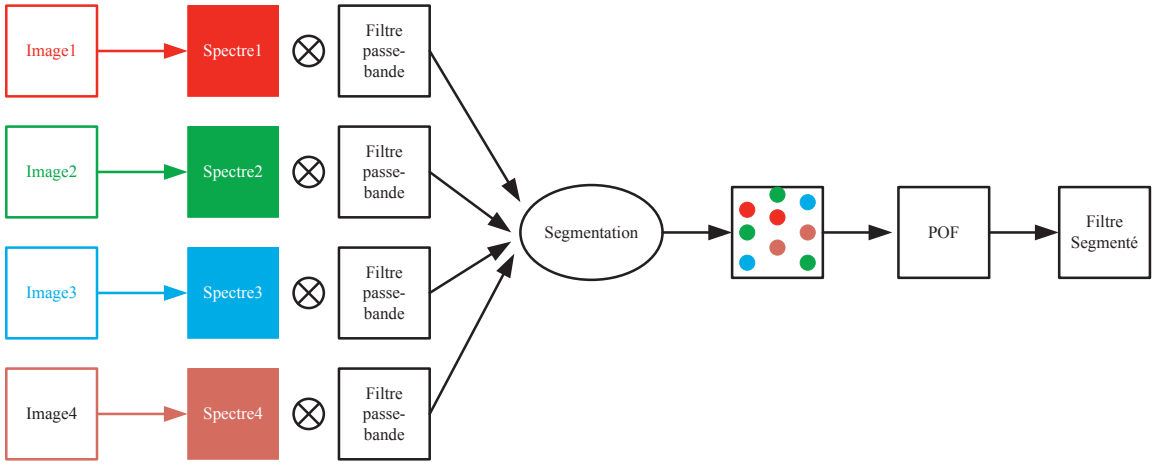


FIGURE 5.8 – Principe de fabrication du filtre segmenté

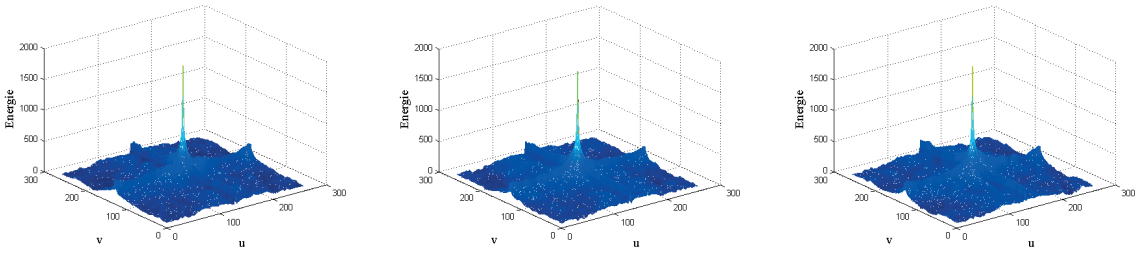
Dans la Figure 5.8, il est indiqué que la construction du filtre segmenté s'effectue en fusionnant des éléments des spectres des images références en se basant sur un critère précis. Un des critères de segmentation utilisé consiste à prendre la valeur maximale de l'énergie normalisée en chaque point. Nous considérons les mêmes images utilisées dans la fabrication du filtre segmenté pour fabriquer le filtre composite (Figure 5.6). Les résultats obtenus, présentés sur la Figure 5.9, confirment l'existence d'un pic dans le plan de corrélation pour chacune des images utilisées pour la fabrication du filtre. Une étude détaillée des avantages et inconvénients de ce filtre segmenté est présentée dans [3]. Dans la suite de ce travail nous optons pour l'utilisation des filtres segmentés grâce à ses bonnes performances en termes de robustesse et de descrimintation comparé aux filtres composites.

5.3 Implantations FPGA

5.3.1 Implantation sur FPGA de la FFT 2D

5.3.1.1 Introduction

La transformée de Fourier FFT 2D est l'un des algorithmes les plus utilisés dans les applications de traitements d'image. En effet, comme nous l'avons montré dans le chapitre 1, les applications de reconnaissances de formes ainsi que certaines application de compression sont



(a) (b) (c)

FIGURE 5.9 – Résultats de simulations de la corrélation en utilisant le filtre segmenté

basées sur l'utilisation des transformations de Fourier en deux dimensions 2D directe et inverse. Ainsi, l'implantation matérielle sur FPGA de ces applications doit passer d'abord par une implantation FPGA de la FFT 2D.

Conformément à [113], il existe 4 réalisations pour le calcul de la FFT 2D : ligne colonne, vecteur-Radix, transformation polynomiale et imbriquée. Dans ce travail, nous nous intéressons à la réalisation ligne colonne la plus utilisée dans les implantations matérielles des FFT 2D.

5.3.1.2 Principe de la décomposition

Le calcul des sorties de la FFT 2D $Y(u,v)$ d'une matrice d'entrée $y(l,k)$ peut se faire par l'équation 5.6.

$$Y(v,u) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} y(l,k)e^{-j\frac{2lu\pi}{N}}e^{-j\frac{2kv\pi}{N}} \tag{5.6}$$

La matrice d'entrée peut être déterminée à partir de $Y(u,v)$ en utilisant la FFT 2D inverse IFFT 2D définie par :

$$y(l,k) = \frac{1}{N^2}\sum_{v=0}^{N-1}\sum_{u=0}^{N-1} Y(v,u)e^{j\frac{2lu\pi}{N}}e^{j\frac{2kv\pi}{N}} \tag{5.7}$$

avec u,v,l,k sont des entiers naturels $\in [0, N-1]$.

Architecture

L'équation 5.6 peut être divisée en deux FFT 1D comme illustré dans l'équation 5.8

$$Y(v,u) = \sum_{l=0}^{N-1} Z(u,l)e^{-j\frac{2lv\pi}{N}} \tag{5.8}$$

avec

$$Z(u,l) = \sum_{i=0}^{N-1} y(i,l)e^{-j\frac{2iu\pi}{N}} \tag{5.9}$$

La matrice Z est ainsi la FFT 1D appliquée à chaque ligne de la matrice d'entrée, et u,v,l,k sont des entiers naturels $\in [0, N-1]$.

D'après les équations 5.8 et 5.9, le calcul de la FFT 2D consiste à calculer une FFT 1D à chaque ligne suivie par une FFT 1D sur chaque colonne. Le principe de cette architecture est ainsi donné par la Figure 5.10.

5.3.1.3 Architecture proposée pour la FFT 2D

Comme il est montré par l'équation 5.8 une FFT 2D d'une matrice de taille $N \times N$, est obtenue en appliquant N fois une FFT de taille N points à chaque ligne et N FFT 1D de taille N points à chaque colonne de la matrice résultante. Les résultats intermédiaires sont sauvegardées dans une mémoire appelée mémoire de transposition comme le montre la Figure 5.10.

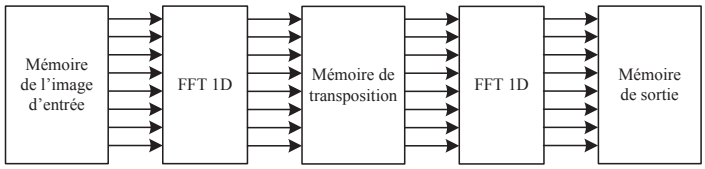


FIGURE 5.10 – Schéma bloc de la séparation ligne colonne de la FFT 2D

Notons que l'application de la FFT 1D sur chaque colonne de la matrice résultante ne peut commencer qu'après l'obtention de toutes les sorties de la FFT 1D appliquée à chaque ligne. Il est ainsi possible d'utiliser le même bloc FFT 1D pour réaliser les transformées pour les lignes et les colonnes. Evidement, l'utilisation d'une unité de contrôle s'impose pour synchroniser les différents traitements. La Figure 5.11 montre un synoptique général de l'architecture proposée pour la FFT 2D.

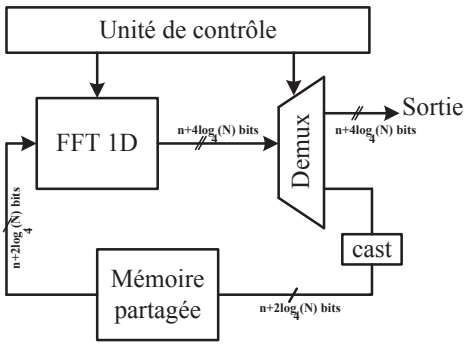


FIGURE 5.11 – Schéma bloc de l'architecture proposée pour l'implantation de la FFT 2D

Dans un premier temps les entrées de la FFT 2D sont sauvegardées dans la mémoire (mémoire partagée). Une première FFT est appliquée à ces entrées en utilisant le bloc FFT 1D. Les sorties de ce blocs vont passer par un bloc démultiplexeur qui permet de gérer ces sorties. En effet, dans le cas d'une première utilisation du bloc IP FFT 1D, nous sauvegardons dans la même mémoire partagée et en écrasant les données d'entrées. Nous sauvegardons que les $n + 2log_4(N)$ bits du poids le plus faible (LSB) en utilisant le bloc *cast* de la Figure 5.11. Par la suite, nous utilisons une deuxième fois le bloc FFT 1D pour l'appliquer aux contenus de la mémoire partagée et le résultat obtenu est renvoyé en sortie. Une unité de contrôle est utilisée afin de gérer les données ainsi que les signaux de contrôle de chaque bloc.

Comme il est montré dans la section 2.6, l'IP FFT 1D proposée à entrées parallèles et sorties parallèles a été favorablement comparée avec les architectures existantes (y compris l'IP de Xilinx). Nous utilisons cette IP pour la réalisation de la FFT 2D. L'IP ainsi proposée possède 4 entrées et 4 sorties, il devient nécessaire de diviser la mémoire utilisée afin d'éviter l'accès multiple et simultanée à la même mémoire.

Pour une FFT 2D de taille $N \times N$, les sorties de la première FFT 1D doivent être sauvegardées dans une mémoire de taille $N \times N$ points. Les indices de sortie de chaque FFT sont symétriques par rapport à $\frac{N}{4}$. La mémoire doit être ainsi divisée en 4 mémoires de tailles $N \times \frac{N}{4}$ points chacune comme il est montré dans la Figure 5.12. Les entrées de la FFT 1D sont organisées en colonnes. Il est ainsi nécessaire de diviser chacune des mémoires de taille $N \times \frac{N}{4}$ points en 4 mémoires de taille $\frac{N}{4} \times \frac{N}{4}$ points chacune comme il est indiqué dans la Figure 5.12.

Au final, l'architecture proposée pour le calcul de la FFT 2D sur FPGA est composée de 4 blocs. Un bloc FFT 1D, un multiplexeur, une mémoire et une unité de contrôle. Le bloc FFT 1D est utilisé pour le calcul des FFT de chaque ligne et de chaque colonne. La mémoire de taille $N \times N$ points est divisée en 16 mémoires de taille $\frac{N}{4} \times \frac{N}{4}$ points chacune. Cette mémoire est utilisée pour sauvegarder les résultats de la FFT 1D appliquée à chaque ligne de la matrice d'entrée. Il est aussi à noter que les données sont sauvegardées dans la mémoire sont organisées sous forme d'un vecteur colonne. Le multiplexeur est utilisé pour la sélection des entrées au bloc

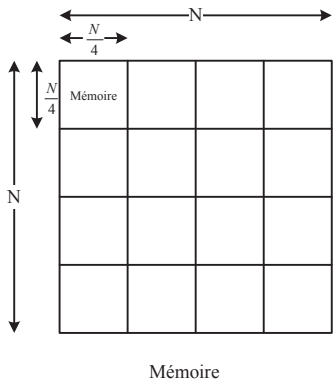


FIGURE 5.12 – Division de la mémoire de transposition

FFT 1D. Finalement, l'unité de contrôle est utilisée pour gérer les adresses de lecture et d'écriture de chaque mémoire ainsi que l'entrée de sélection pour le multiplexeur.

5.3.1.4 Evolution de la dynamique

Comme nous utilisons un seul bloc FFT 1D pour calculer les sorties de la FFT de chaque ligne et de chaque colonne, il devient alors crucial de spécifier la dynamique de ce bloc afin de respecter la taille des entrées et des sorties de ce bloc. En effet, comme il est montré dans la section 2.5, la dynamique de l'IP FFT 1D proposée augmente de $2log_4(N)$ bits pour une FFT de taille N points. Si les pixels de l'image sont codés sur n bits, la FFT 1D doit traiter 2 cas différents. Le premier est lié à l'utilisation de la FFT 1D sur les lignes. Dans ce cas, nous avons un dépassement de $2log_4(N)$ bits. Le deuxième cas d'utilisation vient du fait que la FFT s'applique sur les sorties de la 1^{re} dont les données sont codées sur $n + 2log_4(N)$ bits. En respectant le même dépassement nous obtenons une sortie de FFT 2D codée sur $n + 4log_4(N)$ bits. Ainsi, dans le but d'utiliser un seul bloc FFT 1D conformément à la Figure 5.11, les entrées de la FFT sont codées sur $n + 2log_4(N)$ bits. Pour que ceci soit conforme à l'utilisation de la FFT 1D, nous concaténons $2log_4(N)$ zéros au niveau des n bits de l'entrée que nous sauvegardons dans la mémoire partagée. Aussi, nous troncons $2log_4(N)$ au niveau de la sortie que nous sauvegardons dans la même mémoire partagée.

Plus concrètement, prenons l'exemple d'une image de taille 256×256 en niveau de gris à la quelle nous appliquons une FFT 2D. Les données d'entrée sont codées sur 8 bits. Nous utilisons un bloc FFT 1D qui traite des données d'entrée sur $8 + 2 \times 4 = 16$ bits pour avoir des sorties sur $8 + 4 \times 4 = 24$ bits. Dans le cas de la FFT 1D sur les lignes, nous concaténons 8 zéros aux 8 bits de l'entrée et nous troncons 8 bits sur les 24 bits des sorties afin d'obtenir 16 bits. Dans le cas de la FFT 1D sur les colonnes, nous utilisons 16 bits à l'entrée pour avoir 24 bits en sortie.

5.3.1.5 Analyse temporelle

Dans cette section nous étudions le temps d'exécution de l'architecture FFT 2D proposée en termes de nombre de cycles d'horloge. Les entrées de la FFT 2D sont généralement sauvegardées en mémoire interne ou externe. Rappelons que dans la section 2.5.2.3, nous avons montré que la latence de la FFT est de $\frac{N}{4} + 3(log_4(N) - 1) + 2$ cycles d'horloge (si nous supposons qu'une opération arithmétique (addition ou multiplication) s'exécute en 1 cycle d'horloge). Sachant que toutes les sorties d'une FFT de taille N points sont présentes après $\frac{N}{4}$ cycles d'horloge.

D'autre part, l'architecture de la FFT 1D proposée permet un traitement pipeline des données. En effet, après une latence égale à $\frac{N}{4} + 3log_4(N) - 1$, tous les $\frac{N}{4}$ cycles d'horloge nous obtenons les sorties de la FFT 1D de chaque ligne. Ainsi, le temps d'exécution nécessaire pour avoir toutes les sorties des FFT 1D appliquées à chaque ligne exprimé en nombre de cycles

d'horloge est T_{ligne} donné par :

$$T_{ligne} = \frac{N}{4} + 3log_4(N) - 1 + N \times \frac{N}{4} \tag{5.10}$$

A cet instant, les traitements des FFT 1D appliquées à chaque colonne doit commencer de la même manière que celles réalisées pour les lignes. Par conséquent, le temps d'exécution T_{total} de la FFT 2D appliquée à une une image de taille $N \times N$ est donnée par :

$$T_{tot\_FFT2D} = 2(\frac{N}{4} + 3log_4(N) - 1 + N \times \frac{N}{4}) \tag{5.11}$$

5.3.1.6 Résultats de synthèse

Une implantation FPGA de l'architecture proposée de la FFT 2D pour $N = 256$ a été réalisée en utilisant un FPGA Xilinx Virtex 5 SX95T. Les résultats obtenus après une synthèse logique sont illustrés dans le Tableau 5.1. De plus, nous avons remplacé dans la Figure 5.11 l'IP proposée par celle de Xilinx et nous avons reporté les résultats dans le tableau 5.1.

	Architecture proposée	Architecture à base de l'IP Xilinx
Slices Luts	5131	2587
BRAM	64	64
Fréquence (MHz)	312	374
T_{tot_FFT2D}	32910	132842
Temps d'exécution(ms)	0.105	0.357
Produit surface temps	538	932

Tableau 5.1 – Comparaison entre les résultats de synthèse des différentes architectures de la FFT 2D

Les blocs BRAM sont utilisés pour sauvegarder les données de la mémoire de transposition présentée dans la Figure 5.12. Le temps d'exécution présenté dans le Tableau 5.1 est calculé en multipliant le nombre de cycles d'horloge nécessaire pour l'obtention de toutes les sorties (T_{tot_FFT2D}) par le délai de propagation de chaque architecture. Il est ainsi montré que l'architecture proposée est 3 fois plus rapide que celle utilisant l'IP de Xilinx. Malgré que la surface occupée de l'architecture proposée est plus importante que celle à base d'IP de Xilinx, elle présente un produit surface temps meilleur et par conséquent un meilleur compromis en termes de surface temps.

5.3.1.7 Mesure de la puissance de la FFT 2D

Dans cette section, nous avons utilisé le même banc de test présenté dans la Figure 4.7 afin d'effectuer une mesure de puissance de la FFT 2D que nous avons proposée.

Nous avons dupliqué l'architecture de la FFT 2D afin d'avoir une mesure précise de la puissance consommée. Nous utilisons une IP DDS afin de générer les entrées pour chacune des IP. De plus, nous connectons une sortie de chaque IP FFT utilisé à l'IP Core ChipScope afin de s'assurer de son implantation.

Le tableau 5.2 illustre une comparaison en terme de puissance consommée entre l'architecture de la FFT 2D en utilisant l'IP FFT proposée et celle de Xilinx pour une taille de 256×256.

	FFT 2D à base de l'IP FFT proposée	FFT 2D à base de l'IP Xilinx
Puissance mesurée (mW)	84.3	81.1
Temps d'exécution (ms)	0.105	0.357
Produit temps-puissance (mW.ms)	8.8515	28.9527

Tableau 5.2 – Puissance mesurée des architectures de la FFT 2D

Comme nous l'avons mentionné pour les IP FFT, la consommation de l'architecture FFT 2D proposée à base de l'IP est un peu plus élevée que celle à base de l'IP de Xilinx. Cette surconsommation est due à l'aspect multi-canaux de l'IP proposée. Cependant, malgrè que la consommation de puissance des deux architectures est quasiment identique, l'architecture FFT 2D à base de l'IP FFT proposée est trois fois plus rapide que celle à base de l'IP de Xilinx.

5.3.2 Implantation sur FPGA de la corrélation

5.3.2.1 Introduction

Pour une implantation FPGA de la corrélation, nous nous sommes basés sur l'implantation FPGA de la FFT 2D présentée précédemment. En effet, comme il est montré dans la section 5.2.1, l'architecture de base de l'algorithme de corrélation est basée sur l'utilisation de la FFT 2D et la IFFT 2D et d'une multiplication avec un filtre. Afin de réaliser cette corrélation, il est ainsi nécessaire de choisir le filtre à utiliser pour une implantation FPGA.

Comme nous l'avons cité dans la section 5.2.2, le filtre BPOF est le filtre le plus adapté pour une implantation matérielle. En effet, les valeurs de ce filtre se limitent à 1 et -1 ce qui permet ainsi de réduire la surface occupée. De plus, il possède de très bonnes capacités de discrimination. Ce type de filtre présente ainsi un grand intérêt pour une implantation FPGA de la corrélation.

5.3.2.2 Architecture proposée

Le calcul de la IFFT 2D du spectre obtenu après multiplication par le filtre BPOF ne peut être réalisé qu'après la fin de calcul de la FFT 2D de l'image d'entrée. En outre, comme le montre les équations 5.6 et 5.7, le calcul de l'IFFT 2D et celui de la FFT 2D se font exactement de la même manière à un facteur de normalisation près et égale à $\frac{1}{N^2}$. Pour ces raisons et dans le but d'avoir une surface occupée réduite, nous proposons d'utiliser un seul bloc FFT 2D pour le calcul de la transformée de l'image et la transformée inverse du spectre.

Par ailleurs, comme nous optons pour l'utilisation du filtre BPOF, nous n'avons pas besoin d'utiliser des multiplieurs embarqués pour réaliser la multiplication entre le spectre de l'image et le filtre. En effet, nous mémorisons la valeur 0 quand le contenu du filtre est égal à -1 et la valeur 1 quand le contenu du filtre est égal à 1. Ainsi, en fonction de la valeur du filtre, nous sauvegardons la valeur du spectre ou bien son opposé (multiplication par -1). D'autre part, comme nous avons une architecture qui permet de fournir 4 sorties par cycle d'horloge, afin d'éviter l'accès simultané et multiple à la mémoire de chaque filtre, cette dernière sera divisée en 4 mémoires de taille $N \times \frac{N}{4}$ chacune. Le principe de cette multiplication est donné par la Figure 5.13.

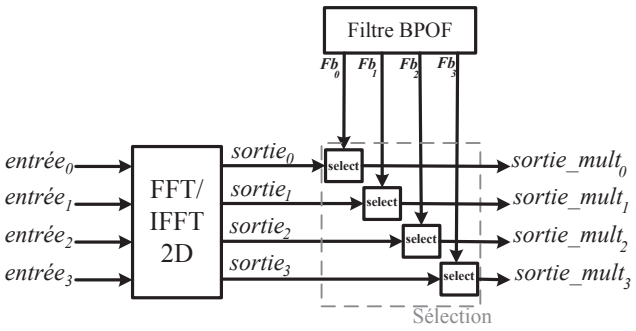


FIGURE 5.13 – Principe de la multiplication par le Filtre BPOF

Les sorties du bloc FFT /IFFT 1D passe par une phase de sélection (bloc *select* de la Figure 5.13). Ce bloc permet, en fonction d'une entrée de sélection, de renvoyer la sortie du bloc FFT/IFFT 1D ou bien son opposé. Le principe de fonctionnement de ce bloc est donné par l'équation 5.12.

$$sortie\_mult_i = \begin{cases} sortie_i & \text{si } Fb_i = 1 \\ -sortie_i & \text{si } Fb_i = 0 \end{cases} \tag{5.12}$$

avec i$\in [0,3]$.

Quant à la mémoire du spectre elle doit être aussi divisée en un premier lieu en 4 mémoires de taille $N \times \frac{N}{4}$ points chacune pour éviter l'accès multiple lors de l'écriture. Par la suite chaque mémoire doit aussi être divisée en 4 mémoires de tailles $\frac{N}{4} \times \frac{N}{4}$ points chacune pour éviter l'accès multiple lors de la lecture. La Figure 5.14 illustre le schéma bloc de l'architecture proposée pour une implantation FPGA de la méthode de la corrélation.

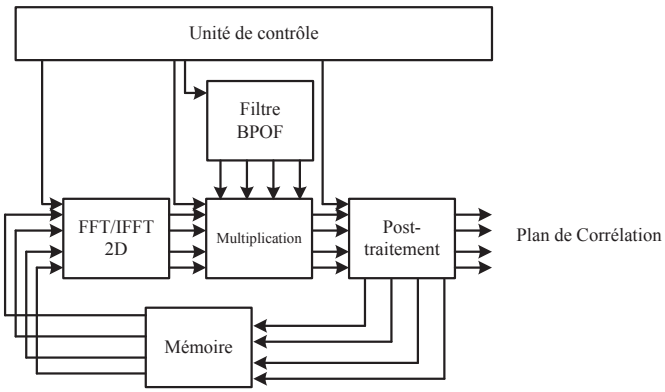


FIGURE 5.14 – Schéma bloc de l'architecture proposée pour la corrélation

L'architecture proposée est ainsi composée d'un bloc FFT/IFFT 2D, deux blocs mémoires de taille $N \times N$ chacune et un bloc de sélection. Le bloc FFT/IFFT 2D est utilisé pour le calcul de la transformée de Fourier 2D directe et inverse. Ce bloc est composé d'une FFT 1D et une mémoire de taille $N \times N$ conformément à la Figure 5.12. Le bloc FFT 1D est ainsi utilisé 2 fois pour chaque transformation en deux dimensions. La mémoire de taille $N \times N$ est utilisée pour sauvegarder l'image d'entrée ainsi que la matrice intermédiaire pour le calcul de la FFT 2D comme nous l'avons expliqué précédemment. Les blocs mémoires de taille $N \times N$ de cette architectures sont utilisées comme suit : la première mémoire est utilisée pour sauvegarder le filtre BPOF fabriqué à partir d'une image référence parmi les images de la Figure 5.2, cette mémoire est de taille $N \times N$ et chaque élément de cette mémoire est codé sur 1 bit. La deuxième mémoire (de taille $N \times N$) est utilisée pour sauvegarder le résultat de la multiplication du spectre de l'image d'entrée par le filtre BPOF fabriqué à partir d'une image de référence. Le dernier bloc sert à faire la multiplication qui se traduit dans ce contexte par une opération de sélection.

5.3.2.3 Evolution de la dynamique

L'architecture de la corrélation proposée est composée, entre autres, d'un bloc FFT/IFFT 2D utilisé pour le calcul des transformées de Fourier en deux dimensions. Cette FFT/IFFT 2D est composé d'un bloc FFT/IFFT 1D utilisé 4 fois : deux fois pour la transformée directe (section 5.3.1) et deux fois pour la transformée inverse. Il est ainsi nécessaire de spécifier la dynamique de ce bloc afin de respecter l'évolution de la dynamique après chaque transformée. En effet, comme nous l'avons cité précédemment, la dynamique d'une FFT 1D augmente de $2log_4(N)$ pour une FFT de taille N points. Par conséquent les valeurs du spectre obtenu sont codées sur $n + 4log_4(N)$. La multiplication par le filtre BPOF est une étape de sélection de la valeur du spectre à sauvegarder et par conséquent elle ne cause pas une augmentation de la dynamique pendant cette multiplication. D'autre part, comme il est montré dans l'équation la IFFT 2D (équation 5.7), les résultats obtenus après les deux sommes sont divisés par N^2. Il est ainsi possible de réduire le gain obtenu après chaque transformation. En effet, les transformées en ligne et en colonne provoque un gain de $4log_4(N)$ bits et la division par N^2 permet une réduction par $2log_2(N)$. Par conséquent, les entrées et les sorties du bloc FFT/IFFT 1D sont codées sur $n + 6log_4(N) - 2log_2(N)$ et $n + 8log_4(N) - 2log_2(N)$ respectivement.

5.3.2.4 Analyse temporelle

Dans cette section, nous présentons le temps d'exécution en termes de nombre de cycle d'horloge de l'architecture de corrélation proposée. Pour se faire nous réalisons une seule comparaison c'est-à-dire, notre base de donnée contient un seul filtre et nous généralisons pour une base de données de L filtres.

Effectuer une corrélation revient à calculer deux transformées de Fourier 2D. Comme nous avons étudié lors de l'implantation FPGA de la FFT/IFFT 2D (5.3.1), le temps d'exécution T_{tot_FFT2D} d'une FFT/IFFT 2D est donnée par :

$$T_{tot\_FFT2D} = 2(\frac{N}{4} + 3log_4(N) - 1 + N \times \frac{N}{4}) \tag{5.13}$$

La multiplication par le filtre BPOF est une étape de sélection et par conséquent, cette phase n'a besoin que d'un seul cycle d'horloge. Ainsi, le temps d'exécution d'une corrélation est, à un cycle près, celui de deux FFT/IFFT 2D ce qui vaut donc

$$T_{tot\_corr} = 4(\frac{N}{4} + 3log_4(N) - 1 + N \times \frac{N}{4}) + 1 \tag{5.14}$$

Dans le cas où la base de données contient plus qu'un filtre, uniquement les phases de multiplication et de la transformée inverse seront réalisées L fois. Le calcul du spectre de l'image résultante ne s'effectue qu'une seule fois. Ainsi le temps d'exécution de cette application devient donc

$$T_{tot\_app} = 2(L+1)(\frac{N}{4} + 3log_4(N) - 1 + N \times \frac{N}{4}) + L \tag{5.15}$$

Notons que nous n'avons calculé que le temps d'exécution des traitements. Dans le cas d'une application de reconnaissance, nous devons ajouter le temps de calcul de PCE et le temps nécessaire pour la prise de décision.

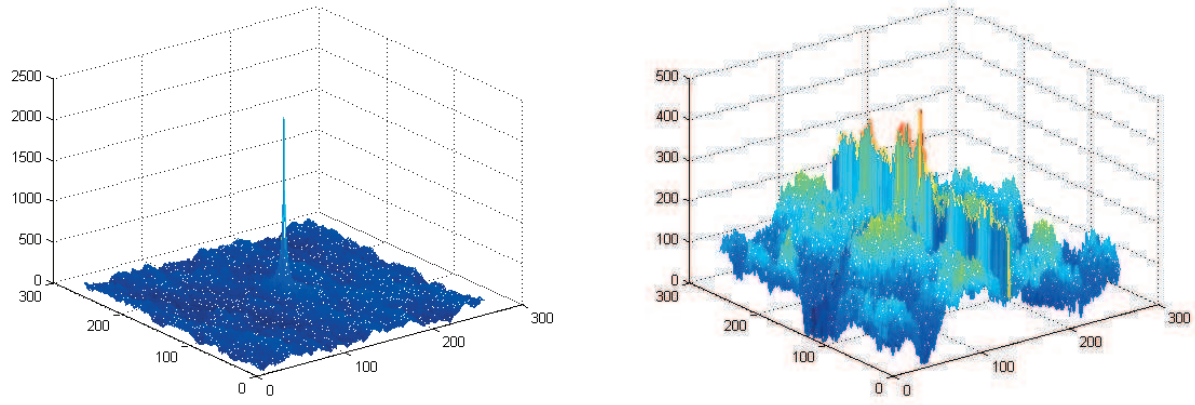
5.3.2.5 Résultats d'implantation

L'architecture de corrélation a été implantée sur un FPGA Xilinx Virtex 5 SX95T en utilisant les images de la Figure 5.15. Les plans de corrélation obtenus sont illustrés sur la Figure 5.16.

FIGURE 5.15 – Images de test

Les résultats obtenus montrent une forte similitude avec ceux obtenus en simulations présentés dans la Figure 5.5. En effet, pour deux images similaires (cible et référence), le plan de corrélation contient un pic intense au centre. Cela permet de mesurer le degré de ressemblance entre l'image cible et l'image référence utilisée pour la fabrication du filtre. Par contre, dans le cas où les deux images sont différentes, le plan de corrélation ne contient aucun pic.

Les résultats de synthèse logique de cette implantation sont illustrés dans le tableau 5.3.



(a) (b)

FIGURE 5.16 – Résultats d'une implantation FPGA de l'architecture de la corrélation proposée

	Architecture à base de l'IP FFT proposée	Architecture à base de l'IP Xilinx
Slices Luts	7462	3503
BRAM	192	192
Fréquence (MHz)	312	342
Temps d'exécution(ms)	0.210	0.776
Produit surface temps	1567	2718

Tableau 5.3 – Comparaison entre les résultats de synthèse des différentes architectures de la corrélation

De la même manière que pour la FFT/IFFT 2D, nous avons implanté l'architecture proposée en utilisant l'IP FFT 1D de Xilinx. Il est montré que l'architecture à base de l'IP FFT 1D proposée est 3 fois plus rapide que celle à base de l'IP FFT de Xilinx. Avec ce temps d'exécution, il est possible de réaliser plus de 4000 corrélations par seconde. Ainsi, nous avons pu atteindre une rapidité proche d'une implantation optique. D'autre part, l'architecture proposée présente un produit surface temps meilleur que celui obtenu avec l'IP Xilinx.

5.3.2.6 Mesure de la puissance consommée par la corrélation

Afin de mesurer la puissance consommée par l'architecture de corrélation de la Figure 5.14. Nous utilisons une IP DDS afin de générer les signaux d'entrée, nous dupliquons l'architecture 21 fois afin d'avoir une mesure précise. Nous utilisons l'IP FFT proposée ainsi que celle de Xilinx et nous mesurons la puissance consommée à base de ces deux IP pour une taille de 256×256. Le résultat de cette comparaison est illustré dans le tableau 5.4.

	Architecture à base de l'IP FFT proposée	Architecture à base de l'IP Xilinx
Puissance mesurée (mW)	126.9	117.36
Temps d'exécution (ms)	0.210	0.776
Produit temps-puissance (mW.ms)	26.649	91.07136

Tableau 5.4 – Puissance mesurée des architectures de la corrélation

Nous calculons aussi le produit temps-puissance. Les résultats obtenus montrent un rapport de 3 fois meilleur en faveur de l'architecture à base de l'IP FFT proposée en termes de produit temps-puissance par rapport à l'architecture à base de l'IP FFT de Xilinx.

5.4 Implantation GPU

5.4.1 Introduction

Dans cette section nous proposons une implantation GPU d'un nouvel algorithme de reconnaissance optimisé. Le choix du GPU est effectué pour sa simplicité de programmation comparée aux FPGA et aussi par ses bonnes performances en termes de rapidité comme nous l'avons expliqué dans la section 1.3.5. Pour la réalisation de cet algorithme nous nous basons sur plusieurs techniques. En effet :

- Nous nous intéressons, à la famille des filtres composites [3, 5, 12, 13, 114]. Ces derniers permettent de regrouper plusieurs images références, réduisant ainsi le nombre de filtres nécessaire à défiler pour avoir une décision fiable. Ce filtre composite de phase permet de réduire d'un côté la taille mémoire nécessaire pour les stockages des différents filtres et le temps nécessaire pour les manipuler de l'autre coté. Afin d'optimiser la fusion des images références, dans le plan de Fourier du filtre composite, nous utilisons une version optimisée de ce filtre. De plus, nous adaptons le critère de segmentation pour tenir compte de la probabilité d'apparition d'une référence en entrée du système. Comme exemple le visage référence avec une rotation 0 a la plus grande probabilité i.e. une personne qui veut entrer dans une salle doit regarder en face la caméra.
- Nous nous intéressons aussi au filtres POF. En effet, ce type de filtre donne des très bonnes performances en termes de discrimination [8] tout en éliminant l'amplitude spectrale du filtre, i.e. l'amplitude spectrale est caractérisée par une très grande dynamique et par conséquence elle nécessite un nombre très grand nombre de bits pour la coder.
- Nous nous limitions à la taille de l'objet à reconnaitre dans la scène. Pour se faire nous avons utilisé la méthode de SVM "Support Vector Machines" [115] pour détecter le visage : sélection de l'information utile.
- Finalement un prétraitement spécifique du plan d'entrée est appliqué. Cette optimisation est basée sur l'utilisation et l'adaptation de la méthode de reconstruction d'image à partir de la seule information de phase dans le domaine spectral [116]. Cela nous a permis de rendre le corrélateur numérique plus robuste vis-à-vis de différents types de bruit : blanc, structuré, additif, etc. Une attention toute particulière a été portée pour limiter le nombre d'itérations nécessaire pour converger notre algorithme de reconstruction à seulement 3 ou 4 itérations.

5.4.2 Algorithme optimisé

L'algorithme proposé est inspiré de la méthode dite "arbre de décision" et il est composé de deux niveaux (d'autres niveaux peuvent être ajoutés si besoin) :

- Niveau 1 : Ce niveau consiste à réaliser un premier classement du visage cible parmi les différentes classes existantes dans la base d'apprentissage considérée : chacune des classes est constituée des différents visages-références d'une seule personne i.e. une classe équivalent à une personne. Un filtre spécial de corrélation est utilisé pour réaliser ce classement qui sera détaillé par la suite.
- Niveau 2 : Une fois le classement est réalisé, nous nous intéressons à un nombre bien déterminé de classes i.e. les classes qui présentent le maximum de ressemblance avec l'objet cible. Ensuite, une comparaison avec chacune de ces classes gagnantes est réalisée pour déterminer l'objet cible. Dans cette partie nous avons utilisé le filtre composite segmenté.

Nous commençons par décrire l'application choisie pour valider notre approche. En effet, bien connaître l'application permet d'un coté de choisir le processeur le mieux adapté à cette application qui permet d'avoir un compromis entre la rapidité et la complexité physique tolérée. De l'autre coté, elle permet de mieux adapter l'algorithme pour une utilisation optimale des ressources disponibles sur la cible numérique choisie.

L'application, choisie dans ce travail, est une application biométrique qui consiste à reconnaître et à identifier un visage cible, présent en entrée du corrélateur. Cette identification du

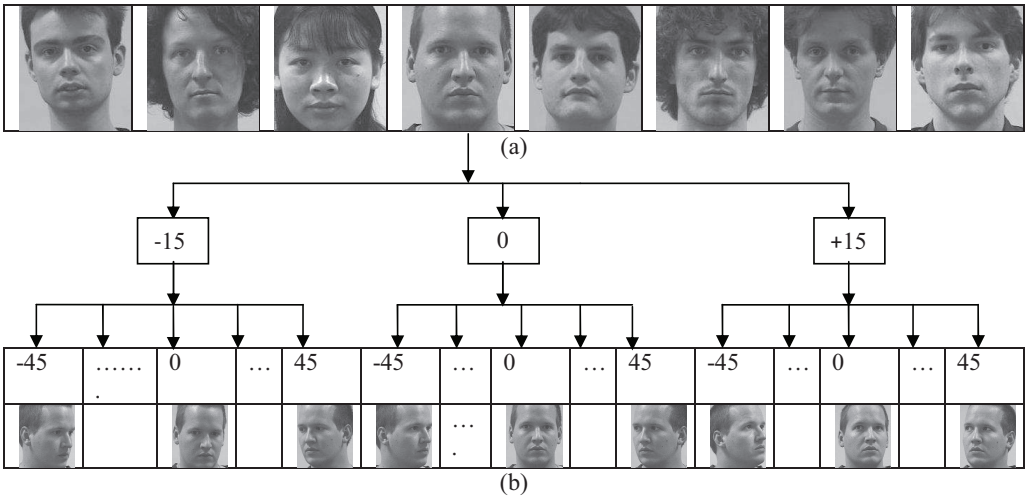


FIGURE 5.17 – La base de test utilisée

visage doit être très discriminante et en même temps robuste. La robustesse est nécessaire pour tenir en compte des différentes modifications possibles du visage cible par rapport aux visages références de la base d'apprentissage : rotation, échelle, bruit de fond, etc. Pour valider le principe de notre architecture implantant numériquement le corrélateur, nous étudions dans cette section, les effets de la rotation (pour les autres problèmes, il suffit de suivre le même principe). Les différents tests réalisés dans ce travail utilisent la base des visages PHPID [117]. Cette base est constituée de plusieurs visages (plusieurs personnes) avec différents angles de rotation, dans un environnement contrôlé et avec un bruit de fond fixe. Pour simplifier la compréhension et l'interprétation des résultats, nous allons seulement exposer les résultats obtenus avec 8 personnes différentes (8 classes). Les 8 personnes utilisées sont présentées 5.17(a). Chacune des personnes présente des rotations verticales comprises entre [-15, +15] et des rotations horizontales comprises entre [-45, +45] (Figure 5.17(b)).

5.4.3 Prétraitement du plan d'entrée

L'introduction de cette étape de prétraitement, facilement réalisable numériquement, a pour objectifs d'augmenter le pouvoir discriminant du corrélateur tout en augmentant sa robustesse aux modifications de l'image cible (rotation, bruit, etc.). Pour se faire, l'image cible reconstruite est un compromis entre deux cas : (1) un premier cas qui consiste à utiliser la seule information de la phase dans son plan de Fourier et (2) le deuxième cas où l'amplitude (amplitude spectrale d'origine) et la phase de l'image cible sont utilisées .

Rappelons que le filtre introduit dans le plan de Fourier du corrélateur est un filtre de phase pure. Dans le premier cas où l'image cible est représentée seulement par sa phase spectrale, la multiplication de ce spectre de phase avec le filtre POF donne la valeur 1 lorsque l'image cible est identique à l'image référence (Equation 5.16).

$$\begin{aligned} Corr &= TF^{-1}[e^{j(\Phi_{cible})}e^{-j(\Phi_{reference})}] \\ Corr &= TF^{-1}[1] \\ Corr &= \delta \end{aligned} \tag{5.16}$$

Dans ce cas, nous obtenons un Dirac dans le plan de corrélation i.e. un pic de corrélation très fin et très énergétique. Cependant, ce filtre est très peu robuste et très sensible aux changements de l'image cible par rapport à l'image référence.

Pour remédier au problème de sensibilité, nous proposons d'ajouter à ce spectre de phase une amplitude reconstruite afin de rendre le corrélateur plus robuste. Après avoir introduit le principe de cette étape de prétraitement, nous compléterons ce paragraphe par une étude sur les effets du nombre d'itérations sur la robustesse et la discrimination du corrélateur.

5.4.3.1 Algorithme de prétraitement

L'algorithme utilisé pour optimiser les performances décisionnelles du corrélateur est décrit sur la Figure 5.18. Cet algorithme consiste : (1) à sélectionner les contours de l'image cible en utilisant la seule information de la phase spectrale de cette dernière, (2) à minimiser les effets de l'amplitude continue de l'image cible (contenue) dans la prise de décision et (3) à reconstruire le plan d'entrée avec une méthode itérative avec un nombre très limité d'itérations (N_{it} = 4 ou 5 itérations) [118]. Pour se faire, nous commençons par introduire l'image cible " Ic(N,N) " pixels dans une image I(2N,2N) pixels i.e. Ic est entourée de zéros. Après, une première transformation de Fourier de la nouvelle image cible I(2N,2N) nous éliminons ses amplitudes spectrales (Amplitude = 1). Ensuite, nous réalisons une transformation de Fourier inverse du spectre modifié. Nous obtenons ainsi une image reconstruite ($2N \times 2N$). Cette image contour Ic est localisée sur (N,N) pixels et le reste n'est que du bruit. En sélectionnant seulement l'image Ic nous obtenons l'image cible modifiée. Si le nombre d'itérations N_{it} est inférieur à un nombre fixé par l'utilisateur, nous réalisons une transformation de Fourier de l'image ainsi reconstruite après l'avoir mise dans une image. De ce spectre nous récupérons seulement l'amplitude et nous la rajoutons au spectre de l'image cible. L'algorithme prend fin quand le nombre de d'itérations est supérieur à n_{it}. Avec n_{it} est le nombre maximal d'itération fixé à l'avance.

Le premier paramètre testé dans cette partie est celui qui affecte le plus les performances du corrélateur ainsi que le temps nécessaire pour prendre une décision à savoir le nombre d'itérations. Pour cela nous avons étudié le comportement du plan de corrélation vis-à-vis de l'augmentation du nombre d'itérations dans l'algorithme présenté dans la Figure 5.18.

5.4.3.2 Résultat d'implantation

La Figure 5.19 présente un aperçu des différents résultats. La Figure 5.19(a) présente l'image cible d'origine. L'image cible reconstruite avec notre algorithme et en utilisant seulement 4 itérations est affichée Figure 5.19(b). L'image cible reconstruite après 200 itérations est présentée sur la Figure 5.19(c). L'image référence utilisée pour fabriquer le filtre POF est illustrée dans la Figure 5.19(d). Les plans de corrélations obtenues sans itérations, avec 4 itérations et avec 200 itérations sont présentés respectivement sur les Figures 5.19(e), (f) et (g). Nous pouvons constater une nette amélioration du plan de corrélation en utilisant 4 itérations (Figure 5.18(f)) : un pic de corrélation plus fin et un bruit nettement inférieur à celui de l'mage cible d'origine (Figure 5.19(e)). Cependant nous retrouvons des similitudes entre les plans de corrélation (e) et (g) de la Figure 5.19. En effet, l'image reconstruite après 200 itérations (Figure 5.19(c)) ressemble à l'image d'origine (Figure 5.19(a)). Pour confirmer cela, nous avons calculé le MSE entre l'image cible d'origine (a) et l'image reconstruite en utilisant différents nombre d'itérations. En effet, le MSE " Mean Square Error" (erreur quadratique moyenne) permet de quantifier la différence entre les valeurs de deux signaux. Les résultats sont présentés sur la Figure 5.19 (h). Ainsi, nous pouvons confirmer que plus le nombre d'itérations est grand plus l'image reconstruite ressemble à l'image cible d'origine, idem pour le plan de corrélation. Finalement, la Figure 5.19 (i) présente

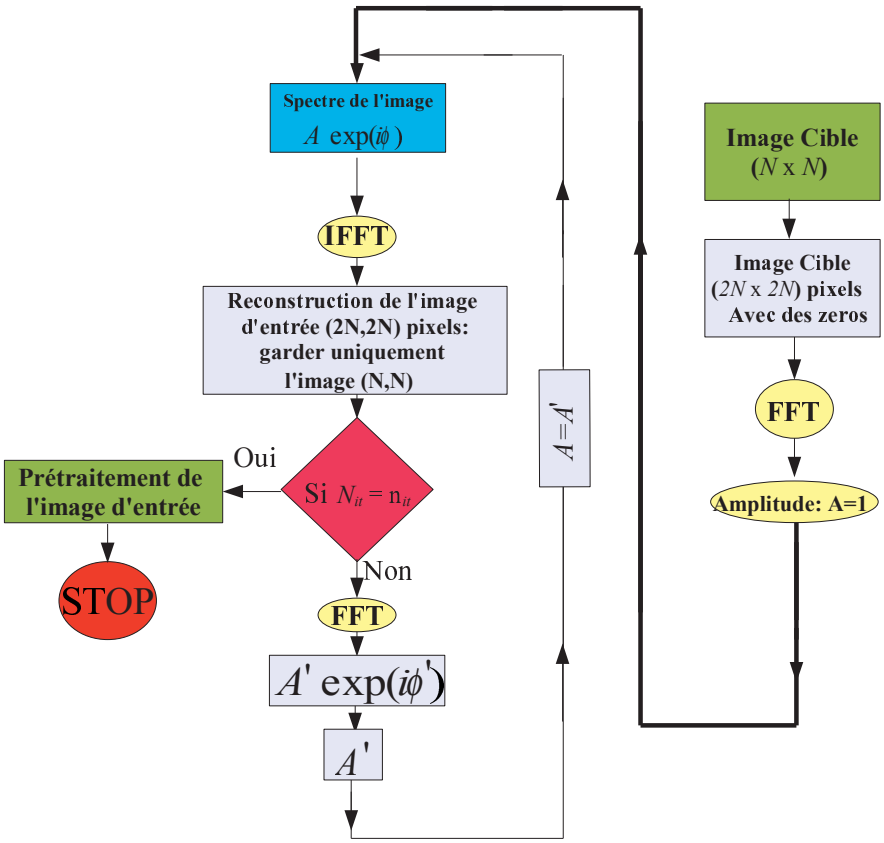


FIGURE 5.18 – L'algorithme utilisé pour traiter l'image cible : rehaussement des contours et lissage du contenue

les PCE obtenus avec les différents nombre d'itérations. Nous pouvons voir que nous avons un maximum pour un $n = 4$ (3 ou 5). Ainsi, il est inutile de réaliser plus d'itérations car d'un coté les performances du corrélateur baisseront et le temps de calcul nécessaire pour ce traitement augmente. Ceci peut ralentir la prise de décision du corrélateur.

Ensuite, nous avons étudié l'effet du prétraitement sur la robustesse du pic de corrélation vis-à-vis de la rotation. Pour cela nous avons comparé les PCE obtenus sans et avec prétraitement en utilisant un filtre POF. Dans la Figure 5.20 nous avons considéré seulement des angles allant de -30 à +30 degrés. Les résultats avec toute la base (Figure 5.17, différents angles de rotations) et en utilisant un filtre segmenté à 3 références sont présentées sur la Figure 5.21.

La Figure 5.20 montre bien que les PCE obtenus avec le prétraitement sont meilleurs que ceux obtenus sans prétraitement de l'image cible. Ainsi, avec ce prétraitement notre corrélateur est beaucoup moins sensible à l'effet de la rotation. La Figure 5.21 montre la courbe ROC obtenue avec toute la base (différents angles de rotations horizontal et vertical). Les résultats obtenus montrent une augmentation du taux de bonnes détections avec le prétraitement. En effet nous obtenons, pour un taux de fausses alarmes proche de 5%, un taux des bonnes détections égale à 58.8 % avec prétraitement contre un taux égale à 47,2% sans prétraitement.

La Figure 5.22 montre un exemple de corrélation après l'application du prétraitement de l'algorithme (Figure 5.18). L'image cible (256,256) pixels est constituée d'un visage avec bruit Figure 5.22(a). L'image référence utilisée pour fabriquer un simple filtre POF est présentée Figure 5.22(b). Nous avons utilisé ce filtre POF pour montrer le rehaussement décisionnelle du

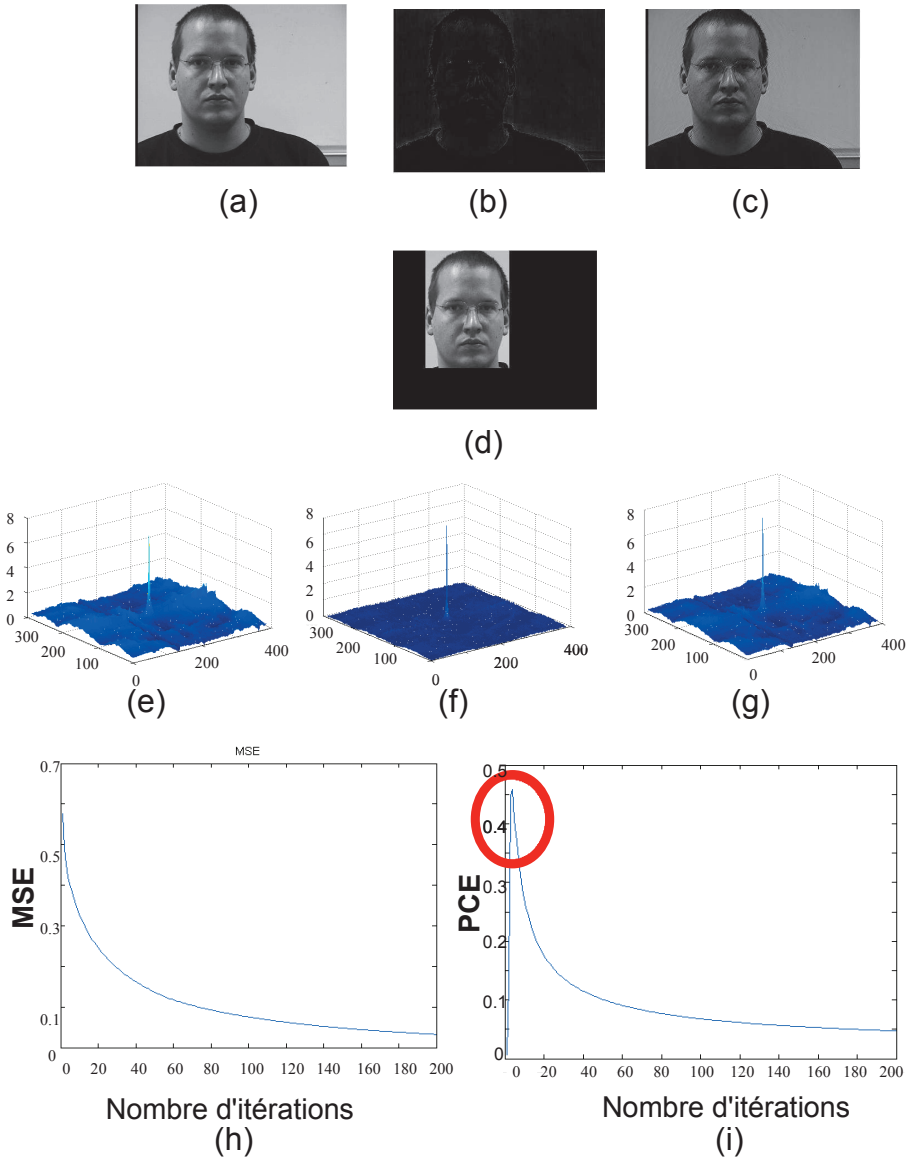


FIGURE 5.19 – Effet du nombre d'itérations sur les plans de corrélation

corrélateur avec ce type de d'algorithme, par la suite nous utilisons un filtre plus complexe et optimisé pour notre application.

Le plan d'autocorrélation obtenu entre l'image cible sans traitement (Figure 5.22(a)) et l'image référence (Figure 5.22 (b)) est présenté dans la Figure 5.22(c). La valeur du PCE de ce plan est égale à 0.09. Après l'application de notre algorithme de prétraitement sur l'image Figure 5.22(a) et après seulement 4 itérations nous obtenons le plan de corrélation présenté dans la Figure 5.22(d) avec un $PCE = 0.86$. Une simple comparaison entre les deux plans de corrélation montre bien le bon comportement du prétraitement proposé. En effet le plan de corrélation avec prétraitement présente beaucoup moins de bruit que celui sans prétraitement. Nous avons testé ce prétraitement sur différents types d'images (binaire et à plusieurs niveaux de gris) avec et sans bruit. Les différents résultats obtenus montrent le bon comportement de ce prétraitement.

Ensuite, nous proposons une deuxième optimisation, adaptée à une implantation sur GPU, qui consiste à réduire le nombre de filtres de corrélation dans la base d'apprentissage sans pour

				
PCE=0.063	PCE=0.117	PCE=0.13	PCE=0.11	PCE=0.061
PCE=0.13 Avec prétraitement	**PCE=0.2233** Avec prétraitement	**PCE=0.51** Avec prétraitement	**PCE=0.237** Avec prétraitement	**PCE=0.18** Avec prétraitement

FIGURE 5.20 – Effet du prétraitement sur la robustesse du corrélateur vis-à-vis de la rotation

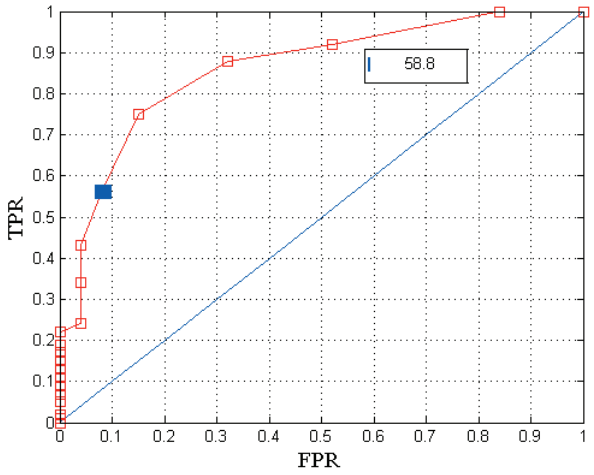


FIGURE 5.21 – Courbe ROC avec prétraitement

autant réduire le nombre des images références. Cela est rendu possible grâce à l'utilisation du filtre composite segmenté (SPOF) [13,119]. Pour valider le principe de notre travail, nous avons utilisé le filtre segmenté classique [13]. Cependant pour augmenter le nombre des références dans le filtre nous pouvons utiliser une version optimisée du filtre segmentée comme celle présenté dans [119].

Le principe du filtre composite segmenté comme nous l'avons présenté dans la section 5.2.2 consiste à fusionner ensemble différents spectres des images références. Pour y arriver, nous commençons par segmenter le plan du filtre en différentes zones. Ensuite, nous affectons chacune des ces zones à un seul spectre issue d'une seule image référence. Pour affecter une zone à un spectre plutôt qu'un autre nous avons utilisé un critère énergétique. Ce dernier compare l'énergie relative d'un pixel (dans le domaine de Fourier) d'un spectre par rapport aux mêmes pixels des autres spectres. Ce critère présente un certain nombre de limitations qui conduisent à une baisse de performances du corrélateur lorsque les images références sont très proches : exemple deux références de la même personne avec un angle de rotation de moins de 10 degrés. Dans ce cas, le plan du filtre est divisé en des multitudes de petites zones (voir même des pixels isolés). Chacune des zones est allouée à une seule référence ce qui ne permet pas d'avoir une corrélation en sortie. Pour remédier à ce problème, nous proposons d'optimiser la segmentation en rajoutant des coefficients de pondérations (a_i) à notre critère [5]. En effet le pixel est alloué à une des références si son énergie relative est a_i fois supérieur à son rivale. Dans le cas contraire ce pixel est alloué à la référence majoritaire autour de ce pixel. Ces coefficients sont obtenus itérativement en augmentant les bonnes détections et en réduisant les fausses alarmes comme

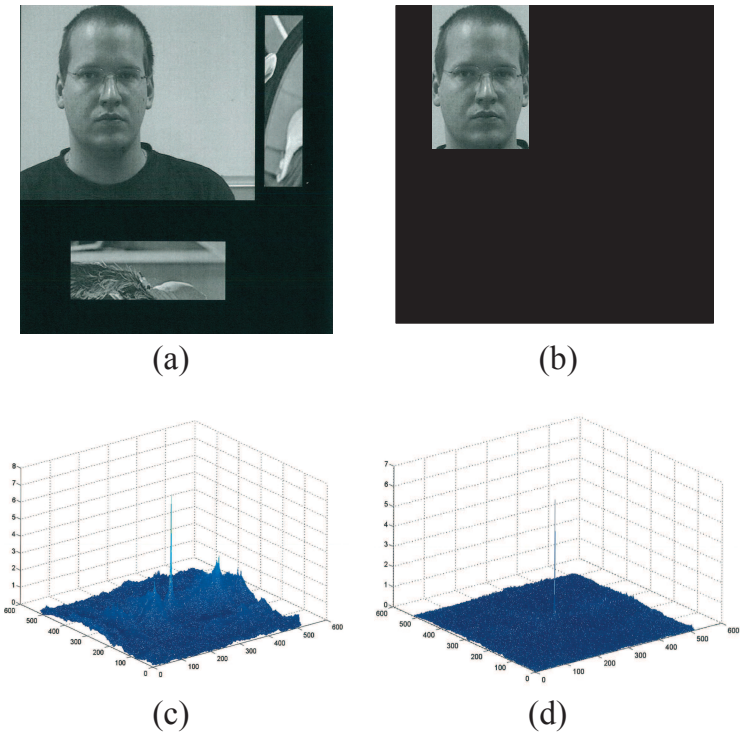


FIGURE 5.22 – Résultats de corrélation

cela est habituellement utilisée avec le filtre composite. Avec cette première optimisation, nous avons réussi à diviser le temps de décision par M (avec M est le nombre des images références considéré dans le filtre composite segmenté).

5.4.4 Architecture de l'algorithme proposée

5.4.4.1 Principe

Pour réduire d'avantage le temps de calcul nécessaire pour la prise de décision, nous proposons un algorithme de corrélation réalisé sur deux niveaux :

- Un premier niveau consiste à déterminer l'appartenance " probable " de l'image cible à une ou à plusieurs classes (un nombre très réduit des classes).
- Ensuite nous identifions l'image cible parmi les images références de la classe considérée.

En d'autres terme, cette optimisation consiste à éviter de comparer l'image cible avec toutes les images de la base d'apprentissage.

5.4.4.2 Classes

Après avoir défini la base d'apprentissage à utiliser comportant K images (Figure 5.23). Pour valider le principe de notre approche tout en simplifiant la présentation et l'interprétation des résultats, nous avons considère 8 personnes, 21 images références sont nécessaires pour reconnaître le personnage quelques soit sa rotation. Cela nous donne un nombre $K = 8 \times 21 = 168$ images. Nous commençons par regrouper les images références d'une même personne ensemble dans une seule classe (nous obtenons ainsi L classes correspondant au nombre des personnes considérées dans la base d'apprentissage (Figure 5.24)). Ensuite, nous regroupons les images d'une même classe selon leurs propriétés dans des " secteurs " comme par exemple la rotation (Figure 5.24).

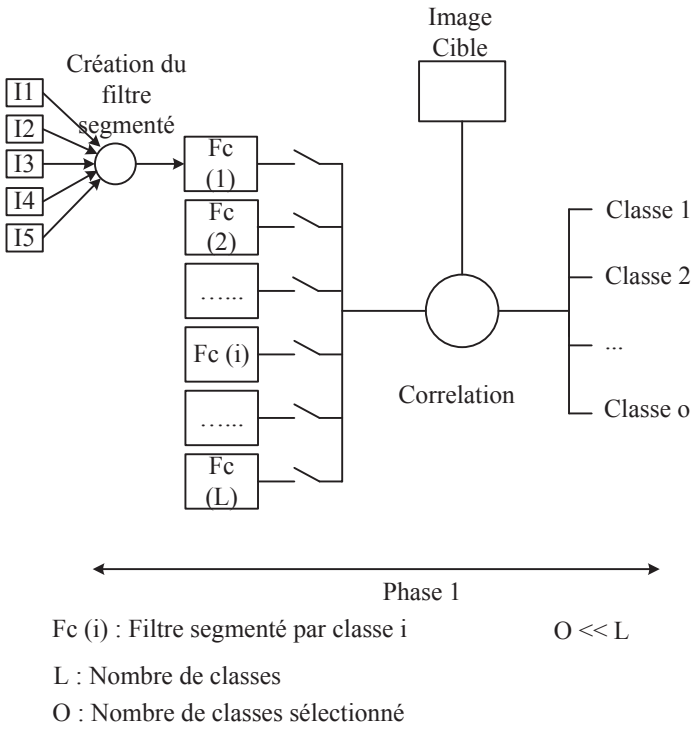


FIGURE 5.23 – La décomposition de la base d'apprentissage en classes

5.4.4.3 Algorithme Niveau 1

L'objectif de cet algorithme est de déterminer l'appartenance de l'image cible (visage à reconnaître) à une ou plusieurs classes de la base d'apprentissage. Pour se faire nous utilisons l'algorithme Niveau (1) présenté dans la Figure 5.24(a). L'image d'entrée prétraitée est introduite dans le corrélateur numérique. Après une première transformation de Fourier nous multiplions son spectre avec un filtre appelé " filtre-classe(i) ". Le filtre-classe(i) est un filtre de corrélation de la famille SPOF fabriqué en fusionnant différentes images références de la seule classe (i). Ainsi nous avons autant de filtres que de classes. Dans ce travail nous avons fabriqué ce $Filtre_{classe}$ en utilisant 5 images références pour chacune des classes considérées. Le choix des images références et leur nombre dépend directement de la classe et de la complexité de l'application. Pour notre cas, seule la rotation du visage est considérée, 5 images référence sont nécessaires pour fabriquer des filtres-classe robustes et discriminants. Selon les différents tests réalisés, le nombre des images références ne doit pas excéder 10 images références dans chacun des filtres pour garder une bonne discrimination. Au delà de ce nombre, il faut songer à avoir un deuxième filtre par classe. A noter que seulement la multiplication avec le filtre de corrélation, la transformation de Fourier inverse et le calcul du PCE sont répétés jusqu'au la corrélation de l'image cible avec toutes les classes.

Une fois l'image cible est corrélée avec tous les filtres classes, nous classons les résultats selon leurs valeurs de PCE. Ensuite, nous considérons seulement les classes qui ont donné les plus grandes valeurs des PCE (2, 3 ou 4 classes). Avec les classes choisies nous utilisons le deuxième niveau. Les résultats de corrélation, de ce premier niveau, obtenus en considérant toutes les images de la base en entrée du corrélateur sont présentées sur la Figure 5.25. En abscisse, nous avons la position de l'image cible dans la base. En ordonnée, nous présentons la valeur du PCE. Chacune des courbes représente toutes les valeurs des PCE obtenues pour une classe. En Haut, nous avons affiché le résultat du classement du personnage "personne 5", par exemple, en utilisant notre premier niveau. Nous pouvons constater qu'avec ce premier niveau nous classons le vraie personnage "personne 5" en première ou en deux positions. Sauf pour deux cas ou il est classé en 3^{eme} et en 4^{eme} ème position mais jamais au delà de 4^{eme} position. C'est la raison pour laquelle nous nous limitons à 4 classes au maximum.

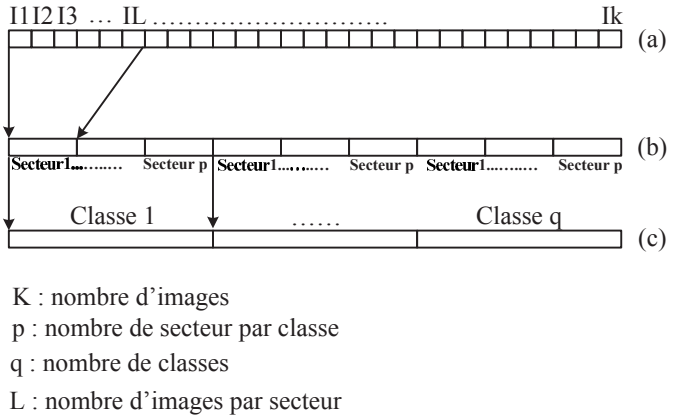


FIGURE 5.24 – Algorithme de corrélation à deux niveaux : (a) classification de l'image cible (b) identification de l'image cible

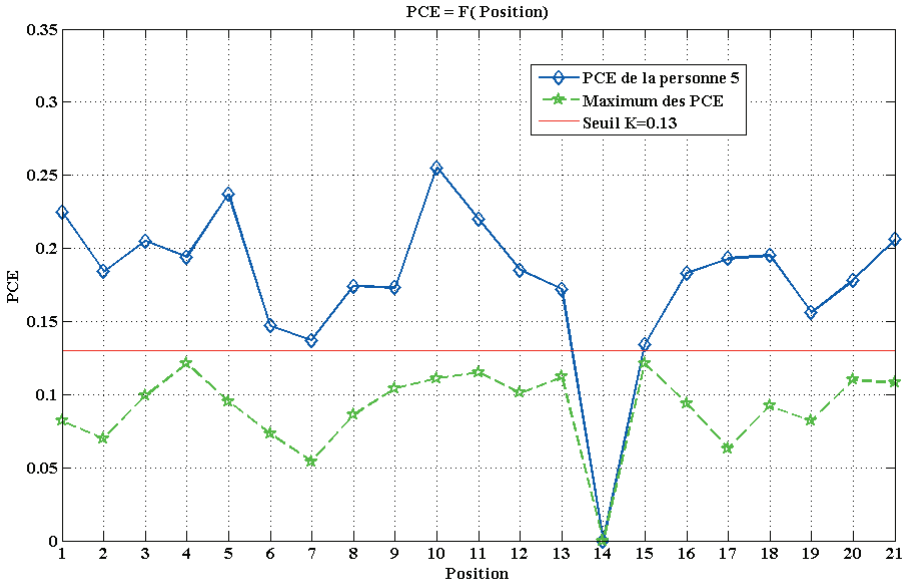


FIGURE 5.25 – Résultats de corrélation de toutes les images considérées avec les 8 filtres classe considérée

5.4.4.4 Algorithme Niveau 2

La première étape de ce niveau consiste donc à classer l'image cible parmi les différentes classes considérées dans la base d'apprentissage i.e. nous sélectionnons seulement les classes qui présentent le plus de ressemblance avec l'image cible. Ensuite, nous fabriquons pour chacune de ces classes différents filtres de secteurs. Pour se faire, nous avons utilisé 3 images références pour définir chacun des secteurs et nous avons utilisé aussi un filtre SPOF (Figure 5.24(b)). Pour réaliser cette étape nous avons besoin de fabriquer 7 filtre par classes. Ensuite, nous corrélons l'image cible avec les différents filtres des secteurs considérés.

La Figure 5.26 présente les résultats des corrélations obtenus avec ce deuxième niveau. Nous avons considéré seulement 3 classes en sortie du premier niveau. Sur cette figure, nous avons représenté deux courbes en introduisant en entrée différentes images du personnage "personne 5" : (1) en continue ceux sont les valeurs des PCE qui correspondent à une bonne identification, (2) en pointillée nous avons représenté les maximes des PCE obtenus avec les 2 autres classes considérées. On constate bien qu'avec seulement 3 classes en sortie du premier niveau nous obtenons un taux de bonne reconnaissance égale à 95 % et un taux de fausse alarme avoisinant

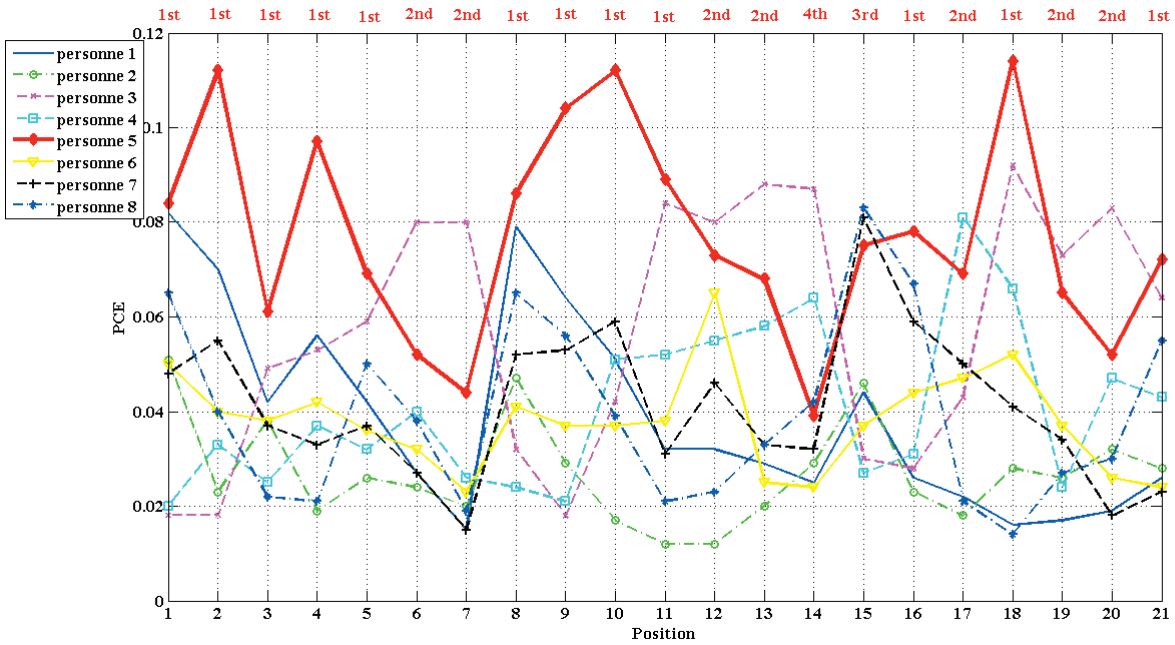


FIGURE 5.26 – Résultats d'identification du visage de la personne 5 en sortie du deuxième niveau

le 0% (aucune fausse identification : le visage est toujours identifié comme la personne 5 ou pas reconnue du tout).

Pour comprendre l'effet de la classification du premier niveau, nous avons considéré 3 cas : (1) seulement deux classes ont été considérées dans le premier niveau, (2) trois classes et (3) quatre classe sont considérées et nous avons considéré toute la base d'apprentissage avec toutes les positions.

La première constatation facile à détecter est que : plus on a de classes en sortie du premier niveau ; plus il y a de corrélations à faire et par conséquence plus le temps de calcul est grand. Ainsi nous allons voir si l'augmentation de nombre de classe en sortie du niveau-1 peut être justifiée par une augmentation des performances décisionnelle du corrélateur numérique. Pour cela nous avons synthétisé tous les résultats dans le tableau 5.5. La première ligne représente le nombre de références considéré dans chaque filtre secteur. Dans ce travail nous avons considéré seulement deux cas 3 et 5 images références (si on veut incorporer plus d'images il faut utiliser une version optimisée du filtre SPOF comme par exemple celle présentée dans [119]). La deuxième ligne représente le nombre de classes considérées en sortie du premier niveau. La troisième ligne représente le taux moyen de reconnaissance (calculé sur toute la base). Finalement, la dernière ligne présente le temps maximum nécessaire pour identifier le visage en entrée du corrélateur parmi toute la base.

Avec seulement 3 images références, nous avons un taux de reconnaisse supérieur à 74 %. Nous pouvons voir aussi que ce taux de reconnaissance augmente avec le nombre de classes considérées en sortie du premier niveau. Cependant, le temps d'exécution augmente aussi. Nous avons la même constatation avec 5 images références dans le filtre SPOF. Finalement, nous constatons que le passage de deux classes à trois en sortie du premier niveau augmente considérable le taux de reconnaissance. Cependant, au delà de 3 classes l'augmentation est beaucoup mois visible cependant le temps augmente considérablement.

5.4.5 Application temps réel

Afin de valider le fonctionnement de notre algorithme optimisé, nous l'avons utilisé pour l'implantation d'une application de reconnaissance de visages en temps réel sur GPU. Les données d'entrée à l'application sont fournies a partir d'un fichier vidéo ou via une webcam.

	Filtre SPOF secteur avec 3 images références			Filtre SPOF secteur avec 5 images références		
Nombre de classe en sortie du premier niveau	2	3	4	2	3	4
Moyenne du Taux de reconnaissance en (%) sur toute la base	74	83	85	80	85	87
Temps maximum pour l'identification (ms)	90	119	147	90	119	147

Tableau 5.5 – Résultats de corrélation de toute la base en utilisant un corrélateur à deux niveaux tout numérique sur une carte GPU

Dans une première phase nous devons construire la base de données des filtres pour l'application de reconnaissance. Pour se faire, nous considérons trois angles à chaque position de la personne à identifier par rapport à la caméra. Pour réduire le nombre de corrélations à réaliser, nous fabriquons un filtre segmenté par position. Pour chaque personne, trois filtres sont fabriqués et par conséquent trois position sont considérées.

Dans une deuxième phase, nous utilisons la bibliothèque OpenCV afin d'interfacer le fichier vidéo sauvegardé dans la mémoire ou bien venant de la webcam avec le GPU. Par la suite, ce fichier vidéo est divisé en plusieurs images de taille 288×352 chacune. Le premier traitement que subissent ces images consistent à sélectionner la zone contenant uniquement le visage de la personne à identifier. Nous obtenons ainsi des images de taille plus réduite(160×160). Cette réduction dans la taille des images nous permettra ainsi de minimiser le temps de calcul ainsi que le temps de transfert des images entre la mémoire de CPU et celle du GPU.

La validation de notre algorithme est effectuée en considérant une situation dans la quelle nous essayons d'identifier une personne parmi une base de données contenant trois personnes. Pour chacune de ces personnes nous considérons trois positions à trois angles différentes. Nous pouvons ainsi appliquer l'algorithme optimisé que nous avons proposé dans la section 5.4.4 à chacune des images de la vidéo. Ce dernier, contenant 140 images pour une durée de 6s [120]. Nous avons montré que nous sommes capables de traiter toutes les images de la vidéo avec un taux de reconnaissance de 77%. En effet, en se basant sur notre algorithme optimisé chaque image de la vidéo a besoin d'un temps de reconnaissance égale à $58,8$ ms en utilisant le GPU Nvidia GeForce 8400 GS. L'identification d'une personne dans la vidéo nécessite 3+9 corrélations au maximum. Au niveau du premier niveau 3 corrélations sont à réaliser, quant au deuxième niveau, nous effectuons au maximum 9 corrélations (3 classes avec 3 filtres par classe).

Le temps d'exécution d'une corrélation est ainsi de $4,9$ ms ($4,9 \times 12 = 58,8$ ms). Ainsi, pour une base données plus grande de p personnes, nous pouvons calculer le nombre d'images à traiter par seconde (Ips) en utilisant l'équation 5.17 :

$$Ips = \frac{1s}{T_{corr} \times (p+9)} \tag{5.17}$$

avec T_{corr} représente le temps nécessaire pour réaliser une corrélation et p est le nombre de personnes de la base.

Il est à noter qu'en utilisant le GPU Nvidia GeForce 8400 GS, le Ips passe de 10 à 2 si p passe de 10 à 100. L'utilisation de GPU plus récents comme le Quadro FX 770M qui contient 4 multiprocesseurs chacun d'entre eux formé par 8 processeurs graphiques permet d'augmenter le nombre d'images par seconde. En implantant le même algorithme sur ce GPU, les résultats obtenus ont montré que le Ips est égale à 29, 19 et 4 si p égale à 3, 10 et 100 respectivement. Ainsi, le nombre d'images traités par seconde est deux fois supérieur comparé à l'implantation avec le GPU GeForce 8400 GS.

5.5 Conclusion

Dans ce chapitre nous avons proposé et validé dans un premier temps, une architecture optimisée pour le calcul de la corrélation sur FPGA. Dans une première phase, nous avons proposé une architecture pour l'implantation de la FFT/IFFT 2D sur FPGA à base de notre IP FFT 1D. Les résultats obtenus avec cette implantation ont montré un rapport de rapidité de 3 par rapport à la même architecture à base de l'IP de Xilinx. Par la suite, nous avons proposé une architecture pour l'implantation de la corrélation sur FPGA. Nous nous sommes basés sur l'architecture de la FFT proposée pour l'implantation de la corrélation sur FPGA. De plus, l'utilisation du filtre BPOF permet d'un coté de réduire la surface occupée et d'un autre coté d'éviter l'utilisation de multiplieurs embarqués.

De plus, l'implantation de l'architecture de corrélation sur un FPGA de la famille Virtex-5 et un GPU Quadro de la même technologie de fabrication (65 nm) a montré un facteur de rapidité de 7 fois meilleur en faveur de l'implantation FPGA. Par contre, afin de tirer profit de la souplesse d'une implantation sur GPU, nous avons choisi d'implanter un algorithme optimisé de reconnaissance de visage. En effet, l'algorithme proposé permet une réduction en termes de corrélations à réaliser dans le cas où la base de filtres est importante. Le traitement de cet algorithme est réalisé à deux niveaux : un premier pour classer l'image cible et ensuite le deuxième consiste à l'identifier. Le taux de bonne reconnaissance est supérieur à 85 % et le temps de calcul est inferieur de 120 ms.

Finalement, nous avons validé l'algorithme proposé dans le cas d'une application de reconnaissance temps réel. Les résultats obtenus ont montré la possibilité de traiter un flux vidéo avec un taux de reconnaissance de 77% dans le cas d'une base contenant 3 personnes. Aussi, il a été conclu que le nombre d'images à traiter dans le flux vidéo dépend du GPU utilisé.

Chapitre 6

Implantation numérique des algorithmes de compression des images

Sommaire

6.1 Introduction

Plusieurs applications telles que la vidéoconférence, l'imagerie médicale, la télésurveillance, les services d'informations sur l'internet, etc. sont basées sur la fiabilité de sauvegarder et transmettre les images. Le stockage des images sur les disques durs des ordinateurs grand public ainsi que l'utilisation des technologies numériques pour le traitement et la retouche des images nécessitent d'acquérir les images sous format numérique (encore appelé format électronique) [121]. Bien qu'il soit le plus adapté aux applications citées plus haut, ce format est extrêmement coûteux en taille mémoire. Pour résoudre ce problème qui peut limiter la faisabilité de stockage et de transmission des images, des techniques de compression d'images ont été élaborées pour compacter leur représentation numérique.

Dans ce chapitre nous utilisons l'IP DCT développées dans le chapitre 3 pour proposer une nouvelle architecture pour l'implantation FPGA de la norme de compression JPEG. Pour réaliser cette implantation, nous commençons par proposer une nouvelle technique permettant de réduire le nombre de coefficients DCT à calculer et par conséquent réduire le nombre de multiplications par la matrice de quantification. Nous finissons ce chapitre par une conclusion dans la quelle nous résumons les implantations réalisées et les résultats obtenus.

6.2 Les techniques de compression

La compression est une opération qui consiste à réduire le nombre de bits utilisés pour représenter un ensemble d'informations sans dégrader la qualité de ces informations (compression sans perte) ou pour représenter approximativement ces données (compression avec perte).

Les techniques de compression se divisent ainsi en deux catégories principales " Compression sans perte " et " Compression avec perte ". La " Compression sans perte " consiste à enlever la redondance dans les données nécessaires pour représenter l'image. Cette redondance est directement liée à la prédictibilité des éléments constituants de l'image. Par exemple, une image de couleur unie est totalement redondante du fait que la couleur fournit suffisamment d'informations pour représenter toute l'image. La " Compression sans perte " identifie les éléments constituants de l'image et exploite leur structure pour réduire la quantité des données. Une élimination simple de la redondance par la " Compression sans perte " ne fournit pas une représentation suffisamment compacte pour plusieurs applications [121].

6.2.1 Compression avec perte

La compression avec perte consiste à enlever les informations indésirables pour reconstruire une image dégradée. Cette redondance est directement liée à la prédictibilité des éléments constituant l'image. Le choix d'une méthode de compression va dépendre de l'application voulue, mais surtout du type d'information non indispensable. Cette redondance peut être classée en trois catégories :

- La redondance spatiale entre pixels ou blocs voisins dans l'image
- La redondance spectrale entre plans de couleur ou bandes spectrales
- La redondance temporelle entre images successives dans une séquence vidéo

6.2.2 Compression sans perte

La compression est dite sans perte lorsqu'il n'y a aucune perte de données entre le message émis par la source et celui reçu et restitué par le récepteur. Ainsi, dans ce type de compression, il y a autant d'informations après la compression qu'avant. L'opération de compression est obtenue en réécrivant d'une manière plus concise les données.

La compression sans perte est parfaitement réversible, l'image décomprimée est identique à l'image originale. Il existe trois types d'algorithmes de compression sans perte :

- Codage statistique dont le but consiste à réduire le nombre de bits utilisés pour le codage des caractères fréquents et à augmenter ce nombre pour des caractères plus rares.
- Substitution de séquences qui sert à comprimer les séquences de caractères identiques.
- Utilisation d'un dictionnaire pour réduire le nombre de bits utilisés pour le codage des mots fréquents et augmenter ce nombre pour des mots plus rares.

6.2.3 Discussion

Les deux techniques de compression (sans et avec pertes) sont basées sur la transformation en ondelettes ou sur la transformation en cosinus discrète. Les techniques basées sur les ondelettes donnent des performances en termes de taux de compression et du rapport signal à bruit meilleurs que ceux des normes à base de la DCT. Cependant, la majorité des normes de compression telles que MPEG, JPEG et H26x sont basées sur la DCT. Le choix de cette transformation est justifié par la faible complexité de l'implantation de la DCT comparée à la transformation en ondelettes. Dans ce travail, nous nous intéressons à la norme JPEG qui est l'une des normes basée sur la transformation DCT.

6.3 Principe de la compression JPEG

JPEG sont les initiales de Joint Photographic Experts Groups, le groupe qui a créé ce format en 1987. Ce format a l'avantage de fournir des images de bonnes qualités avec un taux de

compression élevé. De plus, il permet de choisir la qualité d'image à restituer en faisant un compromis avec la taille de l'image. Cette compression est donc une compression avec pertes.

Le principe de l'algorithme JPEG pour une image à niveaux de gris (sachant qu'une image en couleur est considérée comme superposition des images de ce type) est illustré dans la Figure 6.1.

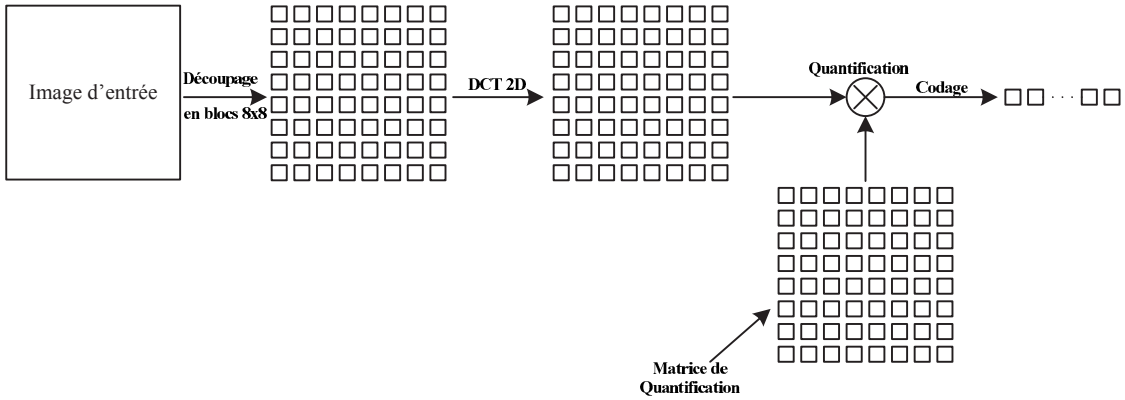


FIGURE 6.1 – Principe de la compression JPEG [73]

La compression d'une image en niveau de gris après sa décomposition en blocs de taille 8×8 passe par trois principales opérations : transformation, quantification et codage.

6.3.1 Transformation

Après une première phase de prétraitement, qui consiste à diviser l'image d'entrée en blocs, la première opération que chaque bloc subit est la Transformation en Cosinus Discrète DCT. Cette transformation permet de concentrer l'information sur l'image en haut et à gauche de la matrice afin de faciliter sa restitution. En effet, nous distinguons deux ensembles de coefficients : Pour un bloc de taille 8×8, la DCT donne un coefficient DC et 63 coefficients AC. Le coefficient DC est la moyenne pondérée des échantillons transformés et représente les détails les plus bruts du bloc d'image (plus basse fréquence spatiale). Les coefficients AC représentent les détails les plus fins de l'image (fréquences spatiales plus élevées). La transformée inverse est indispensable pour retrouver les échantillons dans leur domaine d'origine.

6.3.2 Quantification

Les coefficients de la transformée sont quantifiés à l'aide d'une matrice de quantification de 64 éléments. Cette matrice permet de fixer un élément de quantification important pour certaines composantes jugées peu significatives visuellement. La quantification (Q) n'est que le processus de réduction du nombre de bits nécessaires au stockage d'une valeur entière par la diminution de la précision de la qualité. Plus concrètement, pour chaque coefficient dans la matrice DCT, il faut calculer une valeur quantifiée correspondante en divisant chaque coefficient par un pas de quantification. La formule de quantification de chaque coefficient est la suivante [122] :

$$C^q(i,j) = \frac{Y(i,j)}{Q(i,j)} \tag{6.1}$$

Avec Q(i,j) est l'élément de quantification, $i, j \in [0, N-1]$ et $Y(i,j)$ est le coefficient de la transformée DCT. La matrice de quantification Q est souvent fixée par le standard en se basant sur des constatations empiriques. Par exemple, les matrices de quantification de JPEG prédéfinies dans le standard sont employées pour un facteur de qualité $F_q = 50$, pour la luminance et la chrominance, respectivement. Pour d'autres facteurs de qualité, les éléments de la matrice de quantification sont multipliés par un facteur de compression λ, défini par l'équation 6.2 :

$$C(u) = \begin{cases} \lambda = \frac{50}{F_q} & \text{si } 1 \leq F_q \leq 50 \\ \lambda = 2 - 2 \times \frac{50}{F_q} & \text{si } 51 \leq F_q \leq 99 \end{cases} \tag{6.2}$$

Le résultat obtenu est ensuite arrondi à l'entier le plus proche. La quantification est le principal moyen par lequel on peut contrôler le degré de compression et/ou de fidélité de restitution de l'information compressée. En effet, la quantification permet d'éliminer la redondance spatiale et d'augmenter le nombre de coefficients nuls, en éliminant les valeurs faibles qui portent une information peu énergétique. Des codages entropiques sont ensuite réalisés en utilisant les propriétés statistiques des images.

6.3.3 Codage entropique

Le concept du codage entropique est basé sur la théorie de l'information développée par Shannon [123]. La stratégie consiste à exploiter l'analogie entre la probabilité d'un événement et la quantité d'information à transmettre. Le codage entropique permet d'encoder différents symboles (luminance, chrominance) en leur attribuant un mot binaire. La longueur de ce mot dépend de la fréquence d'apparition du symbole considéré. La réalisation d'un code attribuant aux symboles les plus fréquents les mots les plus courts a pour conséquence de réduire le temps de transmission et de stockage.

Avant l'exécution du codage entropique, les coefficients DCT quantifiés sont arrangés en un ensemble unidimensionnel par balayage en zigzag comme il est montré dans la Figure 6.2. Cet arrangement place le coefficient DC en premier dans l'ensemble, et les coefficients AC restants sont classés de la basse fréquence à la haute fréquence. Cette méthode de balayage permet d'augmenter le nombre d'occurrence d'un même symbole afin de le coder de manière efficace.

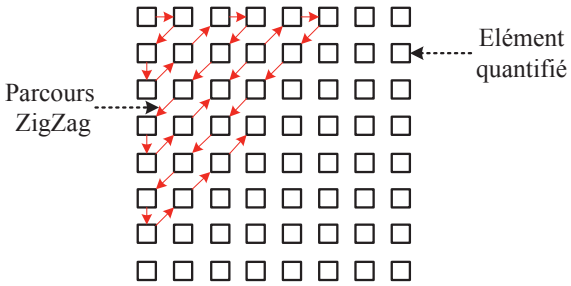


FIGURE 6.2 – Le parcours ZigZag

La norme JPEG spécifie deux méthodes de codage entropique : codage de Huffman [124] et le codage arithmétique [125]. Le codage de Huffman spécifie l'utilisation des tables de codage au cours de la compression et de la décompression. Ces tables sont indiquées dans le standard et doivent être intégrées dans le codec (codeur/décodeur) et peuvent être spécifiées pour une image donnée ou utilisées par défaut pour toutes les images, et ceci suivant l'implantation utilisée. De l'autre coté, les codeurs arithmétiques étaient moins utilisés pour cause de leur protection par des brevets industriels. Leur principe consiste à subdiviser successivement un intervalle de probabilité (initialisé à [0, 1]) suivant la valeur du symbole et de sa probabilité. Au final, la valeur décimale trouvée constitue le code de la séquence d'entrée. Malgré le fait que ce codeur nous permet un gain en compression, il souffre d'une grande complexité de calcul.

6.4 Proposition d'un nouvel algorithme de compression à base de quantification sélective

Dans cette section nous nous basons sur la conception conjointe de la transformation et de la quantification pour proposer une architecture optimisée d'un nouveau schéma de compression. Nous utilisons l'architecture de la transformation DCT 2D proposée dans la section 3.4.3 pour

proposer une architecture optimisée pour l'implantation FPGA de la norme de compression JPEG.

6.4.1 Transformation

L'architecture de la DCT 2D proposée consiste à calculer les sorties de la DCT en se basant sur la réalisation directe et en intégrant les coefficients de la matrice de multiplication dans la matrice de quantification. Cette optimisation, bien qu'elle représente l'avantage d'avoir un rapport surface temps meilleur que celui de l'architecture à base de la séparation ligne colonne, la surface occupée est toujours importante plus particulièrement pour les tailles 8×8 et 16×16. Ici, nous proposons une optimisation supplémentaire afin de réduire davantage la surface occupée pour l'implantation FPGA de l'architecture proposée pour le calcul de la DCT 2D.

Comme nous l'avons cité précédemment, les étapes de transformation et de quantification permettent d'annuler les énergies de hautes fréquences. Nous proposons ainsi de ne pas calculer toutes les sorties. En effet, il est inutile de calculer les sorties de hautes fréquences pour les fixer à zéros après quantification. Pour ces raisons nous proposons pour des blocs de taille 8×8, 4 configurations possibles en calculant des zones de taille 2×2, 3×3, 4×4 et 5×5 comme il est montré dans la Figure 6.3. Chaque zone correspond à un taux de compression spécifique. Pour des blocs de taille 8×8, la transformation permet de calculer 64 coefficients. Cependant, l'architecture proposée permet de réduire le nombre de coefficients à calculer et par conséquent réduire le temps de calcul et la surface occupée. Dans la zone 1, uniquement 4 coefficients sont calculés Y(0,0), Y(0,1), Y(1,0) et Y(1,1). Nous supprimons ainsi les opérations utilisées pour le calcul des autres coefficients. Dans les zones 2, 3 et 4 nous calculons 9, 16 et 25 coefficients respectivement. Tous les autres coefficients sont mis à zéros. Pour des DCT 2D de taille 4×4, nous considérons les mêmes zones 1, 2, 3 précédemment citées.

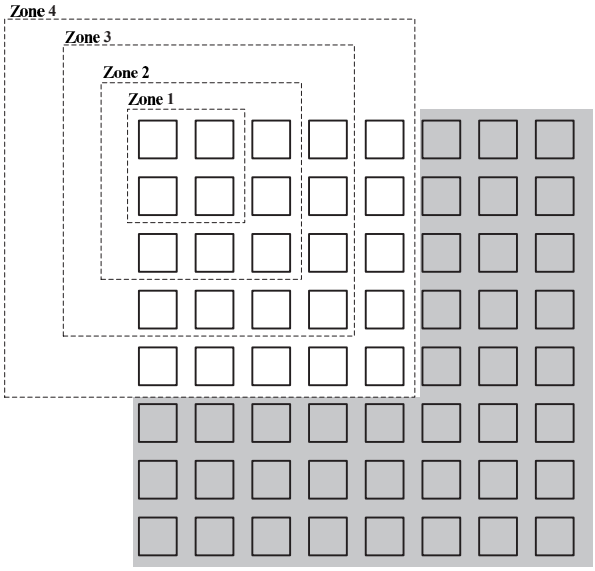


FIGURE 6.3 – Division de la DCT 2D en zones

6.4.2 Quantification

Dans cette étape, nous divisons les résultats de la transformation par la matrice de quantification modifiée dans la section 3.4.3. En effet, la nouvelle matrice de quantification contient le résultat de la multiplication de la matrice de quantification telle qu'elle est définie par les standards par les facteurs d'échelle de la section 3.4.3. De plus, comme nous calculons uniquement des coefficients par zone, ainsi, la matrice de quantification contiendra uniquement les valeurs de quantifications correspondant à chaque zone. Ainsi, la matrice de quantification contiendra uniquement 4, 9, 16 et 25 valeurs respectivement pour les zones 1, 2, 3 et 4.

6.4.3 Complexité algorithmique

La complexité algorithmique du schéma de compression dépend de la zone utilisée. En effet, en fonction de la zone choisie, le nombre de sorties à calculer et par conséquent le nombre d'opérations varie. Afin d'illustrer la réduction en termes de nombre d'opérations, nous présentons l'architecture proposée dans le cas d'une transformation de taille 4×4 et en calculant uniquement les sorties de la zone 1.

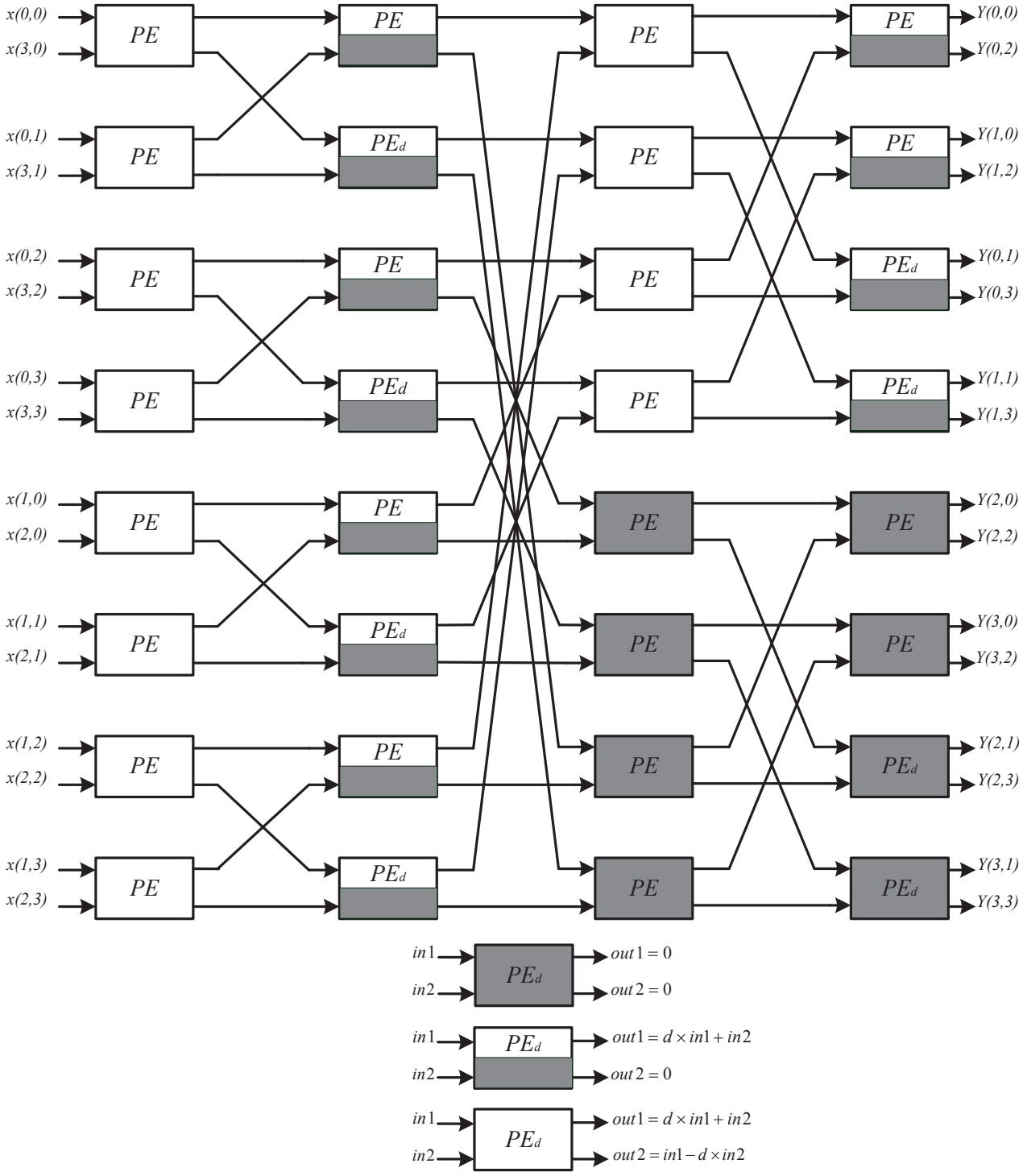


FIGURE 6.4 – Schéma bloc de l'architecture DCT 2D proposée pour 4×4

Les blocs de calcul utilisés dans le schéma bloc de la Figure 6.4 peuvent être divisés en trois groupes en fonctions de leurs couleurs. Les blocs en couleur grise dans l'architecture proposée

ne sont pas utilisés pour le calcul des coefficients de la zone 1. Les blocs avec la couleur grise et blanche utilisent la moitié de leurs nombre d'opérateurs. Par exemple, les blocs PE présentés dans la Figure 3.17 ont besoin uniquement d'une opération addition/soustraction au lieu de deux opérations. Le nombre total d'opérations requis par zone pour des transformations de taille 4×4 et 8×8 ainsi qu'une comparaison avec les architectures discutées précédemment(chapitre 3) est illustrée dans les tableaux 6.1 et 6.2.

	Séparation ligne colonne		Proposée			
	Loeffler	Chen	Zone 1	Zone 2	Zone 3	section 3.4.3
additions	64	64	33	45	72	72
Multiplications	32	32	6	7	16	16

Tableau 6.1 – Complexité algorithmique de l'architecture DCT DCT 4×4 proposée

	Séparation ligne colonne		Proposée				
	Loeffler	Chen	Zone 1	Zone 2	Zone 3	Zone 4	section 3.4.3
additions	416	464	84	160	236	327	496
Multiplications	176	128	30	40	73	78	208

Tableau 6.2 – Complexité algorithmique de l'architecture DCT 8×8 proposée

Les tableaux 6.1 et 6.2 montrent une comparaison entre le nombre d'opérations nécessaires pour le calcul de la DCT 2D de l'algorithme à base de la quantification sélective pour des tailles 4×4 et 8×8 en utilisant différentes zones avec l'architecture de la séparation ligne colonne en utilisant l'algorithme de Loeffler et Chen et aussi avec l'architecture proposée dans la section 3.4.3 du chapitre 3. Il est ainsi montré qu'en fonction de la zone choisie pour le calcul de la DCT 2D, le nombre d'opérations à réaliser diminue. Ce nombre est inférieur à celui nécessaire pour les autres architectures.

L'étape de quantification consiste à effectuer une multiplication point par point entre les coefficients de sortie de la DCT 2D et la matrice de quantification modifiée. Le nombre de multiplications à réaliser dépend aussi de la zone choisie. En effet, dans le cas où nous choisissons la zone 1, uniquement 4 multiplications sont requises. Ce nombre peut passer à 9, 16 et 25 si nous choisissons les zones 2, 3 et 4 respectivement.

Le tableau 6.3 illustre le nombre d'opérations requis par zone pour les étapes de transformation et de quantification.

	Zone 1	Zone 2	Zone 3	Zone 3	Proposée section 3.4.3
Additions	84	160	236	327	496
Multiplications	34	49	89	103	192

Tableau 6.3 – Complexité algorithmique en termes de nombre d'opérations de l'architecture JPEG proposée

Il est montré que le nombre d'opérations est plus faible quand la zone est plus réduite. En effet, en augmentant la taille de la zone, nous augmentons le nombre de sorties à calculer et par conséquent le nombre d'opérations. Il est ainsi crucial d'étudier l'efficacité de chaque zone en calculant le taux de compression ainsi que le rapport signal à bruit.

6.4.4 Taux de compression

La matrice de quantification fixée par le standard JPEG pour la composante de luminosité de l'image pour un facteur de qualité $Fq = 50$ est donnée par l'équation 6.3 :

$$Q = \begin{pmatrix} 16 & 11 & 12 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 92 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{6.3}$$

La première modification que nous apportons à cette matrice consiste à intégrer le facteur d'échelle que nous avons déterminé dans la section 3.4.3 du chapitre 3. La matrice de quantification modifié, Q_{modif}, est donnée par l'équation 6.4.

$$Q_{modif} = Q./E = \begin{pmatrix} 64 & 22 & 28 & 46 & 96 & 259 & 696 & 3205 \\ 24 & 12 & 17 & 28 & 54 & 195 & 425 & 1502 \\ 32 & 15 & 21 & 40 & 93 & 216 & 552 & 1723 \\ 40 & 25 & 37 & 60 & 147 & 407 & 790 & 2356 \\ 72 & 45 & 86 & 162 & 272 & 706 & 1406 & 4834 \\ 155 & 117 & 208 & 299 & 524 & 1091 & 2499 & 7831 \\ 669 & 454 & 624 & 859 & 1406 & 2676 & 5595 & 18120 \\ 3783 & 2512 & 2924 & 3724 & 5885 & 8512 & 18479 & 68342 \end{pmatrix} \tag{6.4}$$

Les matrice E et Q sont des définies pour des blocs de taille 8×8, la matrice E est présentée dans l'annexe C.1 et l'opération ./ représente la division point par point entre les deux matrices.

Pour le schéma de compression par quantification sélective, Q_{modif} dépend de la zone choisie. En effet, si nous choisissons la zone 1 la matrice de quantification modifiée contiendra uniquement 4 valeurs, 9 dans le cas d'un choix de la zone 2 et 16 dans le cas d'un choix de la zone 3. La matrice Q_{modif_1} représente la matrice de quantification modifiée dans le cas d'un choix de la zone 1 est donnée par l'équation 6.5.

$$Q_{modif_1} = \begin{pmatrix} 64 & 22 & 1 & 1 & 1 & 1 & 1 & 1 \\ 24 & 12 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{6.5}$$

Pour une image d'entrée au niveau de gris, les entrées sont codées sur 8 bits. Dans le cas d'une transformation de taille 8×8, l'architecture de la DCT 2D proposée est composée de 8 étages, chaque étage contient 1 additions /soustraction et éventuellement des multiplications. Chaque opération d'addition ou de soustraction introduit un dépassement de 1 bit. Concernant la multiplication, les coefficients de multiplications sont compris entre 0 et 1. Par conséquent la multiplication par ces coefficients n'augmente pas la dynamique de la sortie du multiplieur. Ainsi, chaque étage introduit un gain de 1 bits. Par conséquent, tous les coefficients AC de sortie de la DCT 2D sont codées 16 bits pour des entrées codées sur 8 bits. Pour le coefficient DC, il est le résultat de 6 étages d'additions, il sera codé sur 14 bits.

Dans le cas d'une sélection de la zone 1, la matrice résultante C^q de la division de la sortie de la DCT 2D par la matrice de quantification contient uniquement 4 valeurs $C^q(0,0)$, $C^q(0,1)$,

$C^q(1,0)$, $C^q(1,1)$. Avec $C^q = Y./Q_{modif_1}$ et la matrice Y représente les coefficients de la transformée DCT.

La valeur de Y(0,0) codée sur 14 bits est divisée par $Q_{modif_1}(0,0) = 64 = 2^6$, ainsi la valeur de $C^q(0,0)$ est codée sur 8 bits. D'autre part, $Y(1,0)$ est divisée par $Q_{modif_1}(1,0) = 24$. Ainsi la valeur de $C^q(1,1)$ est codée sur 11 bits. De la même manière nous calculons le nombre de bits utilisés pour coder chaque valeur de la matrice résultante. Ainsi, dans le cas où nous choisissons la zone 1, les 4 coefficients résultats seront codés sur 44 bits. Le taux de compression pour la zone 1 est donnée par :

$$C_r = \frac{8\ bits * 64}{42\ bits} = 12.19 \tag{6.6}$$

Afin d'évaluer d'avantage l'architecture de la compression proposée, nous calculons le rapport signal à bruit PSNR défini par l'équation 6.7 :

$$PSNR = 10log_{10}\frac{d^2}{EQM} \tag{6.7}$$

avec d représente la valeur maximale dans l'image. Nous utilisons des images en niveau de gris, les valeurs sont ainsi codées sur 8 bits, ainsi $d \leq 2^8 - 1$. EQM représente l'Erreur Quadratique Moyenne entre l'image d'origine et l'image reconstruite. Cette erreur EQM est définie par l'équation 6.8 :

$$EQM = \frac{1}{N^2}\sum_{i=0}^{N_0-1}\sum_{j=0}^{N-1}|I_1(i,j) - I_0(i,j)|^2 \tag{6.8}$$

Avec I_0 et I_1 représentent l'image d'origine et l'image reconstruite de taille $N \times N$. Pour les simulations, nous avons utilisé les images de Lena et Barbara présentées dans la Figure 5.15 comme images de test.

A ces images, nous appliquons l'algorithme de compression en utilisant différentes architectures. Dans un premier temps nous utilisons l'architecture de la DCT 2D à base de la séparation ligne colonne suivi d'une quantification telle qu'elle est spécifiée dans la norme JPEG. Par la suite, nous utilisons l'architecture à base de la quantification sélective que nous avons proposée en modifiant à chaque fois la zone sélectionnée parmi les 4 zones détaillées précédemment.

Les Figures 6.5 et 6.6 représentent les images de Lena et Barbara ainsi que leur reconstruction avec l'algorithme de décompression JPEG en utilisant la fonction DCT 2D inverse de Matlab. La valeur du PSNR correspondante à chaque architecture utilisée est mentionnée dans la même figure.

Il est ainsi montré que la qualité des images reconstruites en utilisant l'algorithme de compression à quantification sélective s'améliore en fonction de la zone de quantification choisie. En effet, nous remarquons qu'a partir de la zone 2 nous obtenons des images reconstruites avec une bonne qualité. Il est ainsi crucial de trouver un bon compromis entre la qualité de l'image reconstruite d'un coté et la surface occupée, la puissance consommée et la fréquence de fonctionnement de l'autre côté.

6.4.5 Résultats de synthèse

Une implantation FPGA a été effectuée pour valider le fonctionnement des différents algorithmes étudiés dans le chapitre 3 et dans ce chapitre. Dans un premier temps, nous avons réalisé une implantation sur un FPGA des algorithmes Loeffler et Chen pour le calcul de la DCT 1D. Pour cette implantation nous avons utilisé un FGPA de type Xilinx Virtex 5 SX 95T. Une comparaison en termes de rapidité exprimé en fonction de la fréquence de fonctionnement, et de surface exprimée en fonction du nombre de Slices Luts consommés est illustrée dans le tableau 6.4.

Il est montré que la surface consommée exprimée en nombre de Slices Luts de l'algorithme de Loeffler est inférieure à celle consommée par l'algorithme de Chen. La fréquence maximale de fonctionnement de l'implantation de l'algorithme de Loeffler est aussi meilleur que celle de

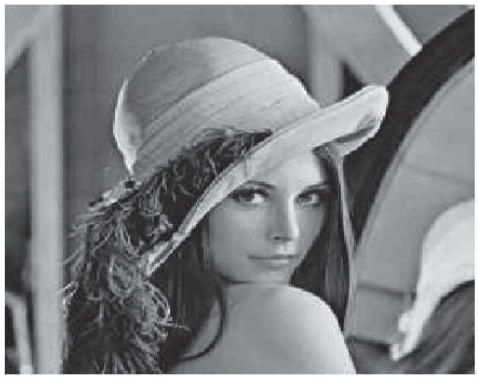
Barbara	Lena
	
Séparation ligne colonne PSNR=31.06 dB	Séparation ligne colonne PSNR= 33.69 dB
	
Zone 1 PSNR=24.61 dB	Zone 1 PSNR=25.50 dB
	
Zone 2 PSNR=26.44 dB	Zone 2 PSNR=28.53 dB

FIGURE 6.5 – Résultats d'implantation FPGA de la compression JPEG

l'algorithme de Chen. Ces résultats confirment ceux présentés dans le tableau 3.4. Par conséquent, nous utilisons l'architecture de l'algorithme de Loeffler pour notre IP DCT 1D.

Une deuxième implantation sur FPGA de la DCT 2D de taille 4×4 en utilisant l'architecture

Zone 3
PSNR=27.58 dB

Zone 3
PSNR=30.9 dB

Zone 4
PSNR=28.60 dB

Zone 4
PSNR=32.62 dB

FIGURE 6.6 – Résultats d'implantation FPGA de la compression JPEG

	Algorithme de Loeffler	Algorithme de Chen
Slices Luts	158	251
Fréquence(MHz)	239	207

Tableau 6.4 – Résultats de synthèse pour une implantation FPGA des algorithmes de Chen et Loeffler

à base de la réalisation directe proposée dans la section 3.4.3 et celle à base de la séparation ligne-colonne en utilisant l'algorithme de Loeffler a été effectuée. Une comparaison en termes de surface occupée ainsi que de la rapidité a été réalisée et présentée dans le tableau 6.5.

	Séparation ligne-colonne (Loeffler)	Architecture proposée section 3.4.3
Slices Luts	341	431
Fréquence maximale	119 MHz	203 MHz
Temps d'éxecution	0.10 μs	0.024 μs
Produit Surface-Temps	34.1	10.6

Tableau 6.5 – Résultats de synthèse de l'architecture DCT 4x4 proposée

En comparant l'architecture proposée avec celle à base de la séparation ligne-colonne, nous

pouvons remarquer que notre architecture consomme plus de surface en termes de Slices Luts mais elle est 4 fois plus rapide. Par conséquent, en comparant le produit surface-temps, qui est le résultat de la multiplication de la surface et le temps d'exécution et qui permet de comparer les architectures en termes du compromis surface-temps, nous remarquons que l'architecture proposée présente un produit 3 fois meilleur que celui à base de la séparation ligne-colonne en utilisant l'algorithme Loeffler. Notons que le temps d'exécution est calculé en multipliant le nombre de cycles d'horloge écoulé pour obtenir les coefficients de la DCT par la période minimale de fonctionnement.

Finalement, une implantation de l'architecture proposée à base de la quantification sélective pour le calcul des coefficients et de la DCT 2D par zones pour la compression JPEG a été réalisée. Le tableau 6.6 illustre une comparaison entre les résultats de synthèse d'une implantation de DCT 2D de taille 8×8 pour l'architecture à base de la séparation ligne colonne et l'architecture proposée dans le cas d'un choix de la zone 1.

	Séparation ligne-colonne (Loeffler)	Architecture proposée Zone 1	Architecture proposée Zone 2
Slices Luts	700	753	1680
Fréquence maximale	207 MHz	241 MHz	241 MHz
Temps d'exécution	0.125 μs	0.033 μs	0.033 μs
Produit Surface-Temps	87.5	24.849	55.44

Tableau 6.6 – Résultats de synthèse de l'architecture DCT 8x8 proposée

Il est ainsi montré que l'architecture proposée à base de la réalisation directe pour les sorties de la zone 1 représente un rapport surface temps plus que 3 fois meilleur que celui de l'architecture à base de la séparation ligne colonne en utilisant l'algorithme Loeffler. Nous remarquons aussi qu'en sélectionnant la zone 2, l'architecture utilisée présente une rapidité 4 fois meilleur que celle à base de la séparation ligne colonne et un produit Surface-Temps 38% meilleur. D'autre part, comme nous l'avons montré précedemment qu'en utilisant l'architecture de compression à base de la quantification sélective, la sélection de la zone 2 permet d'obtenir des images recontruites avec une bonne qualité. Malgré que la zone 2 nécessite plus de coefficients à calculer que la zone 1, la fréquence maximale reste inchangée. Ceci prouve bien que l'architecture est pipeline et que le calcul se fait de façon parallèle.

6.4.6 Mesure de la puissance de la DCT 2D

Finalement, nous avons mesuré la puissance consommée par l'architecture proposée dans la section 3.4.3 du chapitre 3 pour de calcul de la DCT 2D 4×4. Nous avons utilisé l'IP DDS pour générer des entrées codées sur 8 bits et nous avons dupliqué l'architecture de la DCT afin d'avoir une mesure précise. Nous avons comparé les résultats en utilisant les mêmes opérations de mesure pour l'architecture de calcul de la DCT 2D 4×4 à base de la séparation ligne colonne et nous avons comparé les résultats obtenus. Ces résultats sont illustrés dans le tableau 6.7.

	DCT 4×4 proposée à base de la réalisation directe	DCT 4×4 à base de la séparation ligne colonne
Puissance mesurée (mW)	37.2	17.29
Temps d'exécution (μs)	0.024	0.10
Produit puissance-temps(mW.μs)	0.8928	1.729

Tableau 6.7 – Puissance mesurée des architectures de la DCT 2D

En calculant le produit temps-puissance, nous avons trouvé que l'architecture proposée dans la section 3.4.3 est deux fois meilleur que celle à base de la séparation ligne colonne.

6.5 Conclusion

Dans ce chapitre, nous avons proposé une architecture optimisée pour une implantation FPGA de la norme de compression JPEG. L'architecture que nous proposons est basée sur l'optimisation réalisée pour le calcul des coefficients de la DCT 2D présentée dans la section 3.4.3. Afin de réduire d'avantage la surface occupée par cette architecture, nous avons proposé de réduire le nombre de coefficients de DCT à calculer et par conséquent le nombre de multiplications à réaliser dans le calcul de la DCT et de la quantification. Quatre zones ont été définies en fonction du nombre de sorties à calculer. Les résultats obtenus en termes de complexité algorithmique, taux de compression et rapport signal à bruit ont montré l'efficacité de l'architecture proposée.

Conclusion et Perspectives

Conclusion et Perspectives

Les travaux présentés dans ce document s'intéressent à l'implantation numérique des algorithmes de traitement d'images. Plus particulièrement, nous nous sommes focalisés sur les algorithmes de compression et de reconnaissance de formes. L'objectif est de tirer profit de l'efficacité des algorithmes optiques d'une part et de la haute résolution offerte par les cibles numériques. Cependant, comparés aux composants optiques, les circuits numériques sont moins rapides. C'est pour ces raisons que nous nous sommes efforcés de proposer une implantation numérique en essayant de se rapprocher de la rapidité des réalisations optiques. Afin d'atteindre cet objectif, nous avons commencé par détailler les différentes implantations optiques pour la reconnaissance de formes et de compression. En dépit des bonnes performances en termes de rapidité offerte par ces implantations, elles souffrent de plusieurs problèmes tels que la mauvaise qualité des résultats, le coût, la complexité de la réalisation. Pour y remédier, nous nous sommes focalisés sur les architectures numériques offrant un bon compromis en termes de performances/flexibilité.

Par la suite, nous avons proposé une architecture optimisée pour le calcul de la FFT qui est l'un des algorithmes les plus utilisés dans les algorithmes de traitement de l'image et du signal. Nous avons commencé par étudier les différents algorithmes de calcul de la FFT. Ensuite et après avoir exploré les architectures permettant l'implantation de la FFT sur FPGA, nous avons proposé une architecture de l'IP FFT optimisée. L'optimisation est réalisée conjointement en termes de rapidité et de surface. En effet, nous nous sommes basés sur l'algorithme Radix-4 afin de proposer une architecture à entrées parallèles et sorties parallèles ce qui permet de réduire la latence et par conséquence le temps d'exécution. D'autre part, et afin de réduire la surface occupée nous avons proposé une technique de partage de mémoire entre les différents étages de calcul. Les résultats obtenus ont montré l'efficacité de l'IP proposée.

En outre, nous nous sommes intéressés à la transformation en cosinus discrète DCT comme étant la fonction la plus utilisée dans la majorité des normes de compression d'images. Cependant, en fonction du standard de compression, la DCT peut avoir plusieurs formes et plusieurs tailles. En effet, la DCT utilisée dans la norme de compression JPEG est de taille 8×8 sous sa forme réelle. Par contre, dans la norme H264, la DCT utilisée est entière, de taille allant de 2×2 à 16×16. Afin de faire face à ces divergences, nous avons proposé une architecture optimisée et générique qui peut être utilisé par plusieurs standards.

Dans cette thèse, nous avons touché à plusieurs aspects de recherche dans le métier de l'électronique. Non seulement des IP optimisées ont été proposées, mais aussi nous avons établi un banc de test dans les locaux de l'entreprise Interface Concept. Le but étant d'aller de l'algorithme à la puce et d'évaluer expérimentalement les performances des IP conçues. En effet, à travers un prototypage rapide utilisant les outils de Xilinx, nous avons réussi à implanter correctement une application de détection de signal dans un contexte d'autoprotection en Guerre Electronique. Nous avons montré que l'IP FFT peut être utilisée derrière un CAN échantillonnant les signaux avec une fréquence allant jusqu'à 1,25 GHz. Ceci n'est pas le cas des IP commerciales.

De plus, l'IP FFT a été intégrée pour calculer le spectre des images dans une application de détection de visage à base de la corrélation. Il a été montré que pour une puissance mesurée proche de celle l'IP FFT de Xilinx, l'IP FFT proposée est 2 fois meilleure en terme de rapidité pour des images de taille 256×256. De plus, l'implantation de l'architecture de corrélation sur un FPGA de la famille Virtex-5 et un GPU Quadro de la même technologie de fabrication (65 nm) a montré un facteur de rapidité de 7 fois meilleur en faveur de l'implantation FPGA. Par contre, afin de tirer profit de la souplesse d'une implantation sur GPU, nous avons choisi d'implanter

une application de reconnaissance à deux niveaux en temps réel. Cette implantation a été réalisée avec succès et a montré que le taux de détection peut atteindre 77 % avec la possibilité de traiter un flux vidéo à une cadence de 19 images par seconde pour une base de données composée de 10 personnes avec 9 images par personne.

Quant à l'IP DCT, elle a été correctement implantée pour une application de compression inspiré du standard JPEG. De plus, afin d'améliorer le rapport surface-temps, nous avons proposé une méthode de compression basée sur la quantification sélective. Les résultats obtenus ont montré l'avantage de l'architecture proposée par rapport aux schémas classiques de compression.

Les perspectives de ce travail se divisent en trois parties : court, moyen et long termes.

D'abord les perspectives à court termes consistent à réaliser dans un premier temps une implantation numérique de la TF optique. En effet, les travaux réalisés dans l'équipe Vision de notre laboratoire ont montré que la TF optique permet d'avoir de meilleurs résultats par rapport à la FFT [39]. De plus, la TF optique peut être obtenue en utilisant la FFT. Ainsi, nous visons à utiliser l'IP FFT optimisée afin d'implanter la TF optique dans le but de se rapprocher de la rapidité et des performances de la TF optique.

Dans un deuxième temps, nous nous intéresserons à l'architecture de l'IP FFT proposée afin de l'optimiser davantage pour une utilisation dans les applications de reconnaissance de formes. En effet, nous avons remarqué que l'information utile est celle calculée sans utiliser des multiplications par les facteurs de rotation. Nous proposons ainsi de substituer les multiplications par des opérations de décalage estimant les valeurs de facteurs de rotations par des constantes en puissance de 2. Par conséquent, nous éliminons les multiplications dans le calcul des coefficients de la FFT. Ceci permet de réduire la surface occupée d'une part et d'améliorer la rapidité en augmentant la fréquence maximale de fonctionnement d'un autre côté.

Les perspectives à moyen termes, ont pour objectif d'utiliser les IP FFT proposée dans des applications de reconnaissance de visages basées sur un flux vidéo en temps réel. En effet, nous visons à implanter l'algorithme de reconnaissance de visage sur FPGA en utilisant le même algorithme validé par une implantation sur GPU. Nous devons ainsi tout d'abord concevoir une architecture permettant de recevoir un flux vidéo à partir d'une caméra ou un fichier vidéo et de le traiter afin d'extraire les différentes images et par la suite appliquer l'algorithme de reconnaissance de visage à deux niveaux.

Aussi, nous comptons implanter sur FPGA une application de compression vidéo dans le cadre du projet "Rapide" proposée par l'entreprise Interface Concept. En effet, le projet consiste à envoyer des avions équipés par des caméras haute définition afin d'enregistrer une scène vidéo pour une durée maximale. Cependant, les mémoires des FPGA sont de taille limitées. L'idée est ainsi de compresser le flux vidéo en temps réel ce qui permettra d'enregistrer une période plus longue. Nous proposons d'implanter la norme H264 pour la compression des vidéos en utilisant l'architecture de la DCT 2D et un codeur entropique performant.

A long terme, nous visons aussi à exploiter l'IP FFT optimisée pour l'implantation des projets en cours de réalisation dans l'équipe Vision de notre laboratoire. En effet, l'IP FFT proposée trouve bien sa place dans le projet de détection des mines élaboré dans le cadre d'une collaboration entre l'entreprise Thales et l'ISEN Brest [126]. L'objectif de ce projet est la détection, la classification et l'identification des mines sous marines dans des vidéos. Les résultats obtenus dans ce projet ont montré l'efficacité de l'algorithme proposé. Cependant, le temps de traitement est assez long. Afin d'améliorer ce dernier, nous proposons d'utiliser l'IP FFT proposée pour une implantation sur FPGA ou bien une solution à base de corrélation implantée sur GPU.

De plus, un autre projet est en cours de réalisation dans notre équipe consiste à la détection des chutes de personnes âgées. Ce projet est dans le cadre d'une collaboration entre l'entreprise Malakoff Médéric et l'ISEN Brest. Il consiste à identifier une personne dans une chambre, la suivre et en cas de chute la détecter. Afin d'améliorer le temps de calcul, nous proposons d'utiliser l'algorithme de reconnaissance à deux niveaux implanté sur GPU. En effet, les résultats obtenus ont montré l'efficacité de l'algorithme proposé pour des applications en temps réel.

Annexes

Annexe A

Les courbes ROC

Concrètement, la reconnaissance faciale consiste en la séparation des résultats en deux groupes distincts suivant un seuil (le sujet est soit reconnu comme étant celui utilisé pour la création du filtre, soit non reconnu). Il s'agit donc d'un classifieur binaire. On définit le vecteur w, appelé vecteur d'observation, composé de n observations w_1, ..,w_n. Le principe est de prendre la meilleure décision à partir d'un ensemble d'observations suivant un critère $\hat{\theta}(w)$ qui soit la meilleure estimation possible du paramètre décisionnel θ. On pose Y le vecteur formé par l'ensemble des valeurs prises par le critère d'évaluation lorsque le sujet cible correspond au sujet de référence, Z le vecteur formé dans le cas contraire. On définit l'espérance $E[Y]$ de Y, variable aléatoire de loi de probabilité discrète (p_i,y_i) , contenant n éléments équiprobables par :

$$E[Y] = \mu = \sum_{i=1}^{n} p_i y_i = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{A.1}$$

et l'écart-type σ par :

$$\sigma = E[Y]^2 - E[Y^2] \tag{A.2}$$

Le signal est dit gaussien lorsque son vecteur aléatoire est gaussien, c'est-à-dire lorsque les vecteurs formés par les valeurs mesurées forment une loi normale $N(\mu, \sigma^2)$. On a donc Y $N(\mu_Y, \sigma_Y^2)$ et Z $N(\mu_Z, \sigma_Z^2)$. Les valeurs prises par la densité de probabilité de chacun des deux vecteurs Y et Z forment donc deux gaussiennes distinctes. On obtient une courbe de la forme :

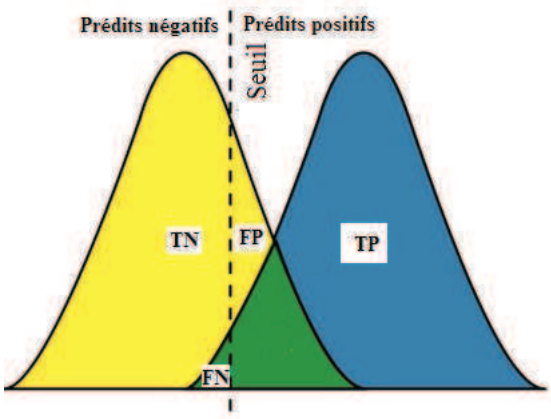


FIGURE A.1 – Courbe gaussienne

avec
TP : True Positive : les prédits positifs qui le sont vraiment.
FP : False Positive : les prédits positifs qui sont en fait négatifs.
TN : True Negative : les prédits négatifs qui le sont vraiment.
FN : False Negative : les prédits négatifs qui sont en fait positifs.

	Positif	Negatif
Positif	TP	FP
Negatif	FN	TN

Tableau A.1 – Matrice de confusion

Cette courbe est une visualisation de la séparation des observations v dans les deux classes indépendantes Y et Z. Quatre différents cas sont possibles, les cas de détection et non détection, et les cas de fausse alarme et de non détection fausse, qui constituent des erreurs du classifieur, l'objectif étant de déterminer le seuil qui constituera le meilleur compromis. Afin de mesurer graphiquement la performance du classifieur, on utilise la courbe ROC "Receiver Operating Characteristic" (ROC) [127], courbe paramétrique représentant le taux de détection vrai (appelé également sensitivité ou probabilité de détection vraie), en fonction du taux de fausse alarme (appelé également 1-spécificité ou probabilité fausse alarme), suivant la valeur du seuil de détection. Soit, en appelant H_0 l'hypothèse que l'image cible soit celle du sujet 0 et H_1 que ce soit celle du sujet 1, D_0 que le sujet 0 soit détecté et D_1 que le sujet 1 soit détecté, on a :

$$FPR = P(D_1|H_0) + P(D_0|H_1) \tag{A.3}$$

et

$$TPR = P(D_0|H_0) + P(D_1|H_1) \tag{A.4}$$

Pratiquement, la valeur du seuil est variée graduellement de 0 à 1. Selon la valeur de l'estimation $\hat{\theta}(v)$ et du seuil s, le classifieur prend une décision, D_1 ou D_2, permettant la construction d'une matrice de confusion pour chaque valeur du seuil, comptabilisant le nombre de détections, de non détections, de fausses alarmes et de non détections fausses :

Les False Positive Rate (FPR) et True Positive Rate (TPR) sont finalement définis de la façon suivante

$$TPR = \frac{TP}{TP + FN} \tag{A.5}$$

et

$$FPR = \frac{FP}{TN + FP} \tag{A.6}$$

En traçant le TPR en fonction du FPR pour chaque valeur du seuil s, on obtient la courbe ROC paramétrée par le seuil s (Figure A.2).

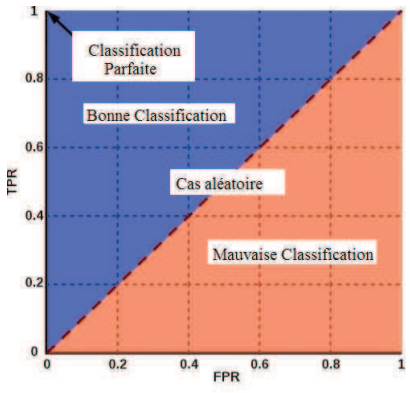


FIGURE A.2 – Courbe ROC

Le point dans le coin supérieur gauche du graphique, appelé " Perfect Classification ", correspond au cas optimal, ou de détection singulière, apparaissant lorsque les densités de probabilité des vecteurs X et Y ont des supports disjoints. Lorsqu'un point de la courbe apparait sur la diagonale, la classification est dite aléatoire. Cette diagonale divise donc l'espace de la courbe en deux parties : la bonne classification (au dessus de la diagonale) et la mauvaise classification (sous la diagonale). Une façon de représenter numériquement la performance d'un classifieur

est donc de considérer l'Area Under Curve (AUC) [128]. On a donc, logiquement, $AUC = 0.5$ lors d'une classification aléatoire et $AUC > 0.5$ lors d'une bonne classification. Bien que cette métrique peut constituer une indication de la performance du classifieur, il s'agit d'une perte d'information par rapport à la courbe ROC elle-même.

Annexe B

Schéma bloc d'une réalisation directe de la FFT 64 points

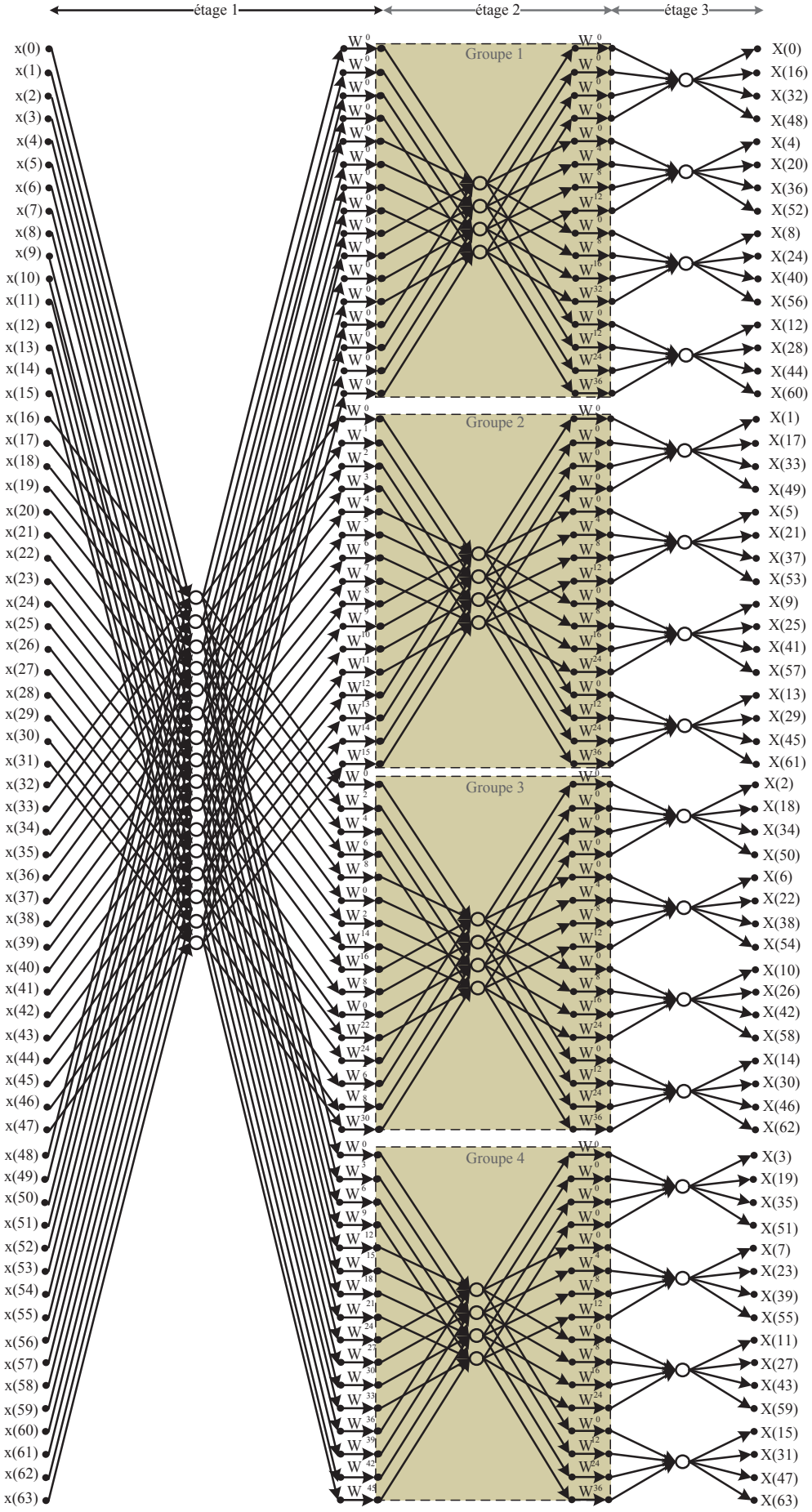


FIGURE B.1 – Schéma bloc d'une réalisation directe de la FFT 64 points

Annexe C

Architecture proposée pour le calcul de la DCT 8×8

C.1 Algorithme

La représentation matricielle pour la réalisation directe dans le cas où $N = 8$, est donnée par :

$$Y = AXA^T \tag{C.1}$$

Avec A$=\begin{pmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{pmatrix}$

et $\begin{cases} a = \cos(\frac{\pi}{16}); \quad b = \cos(\frac{2\pi}{16}); \\ c = \cos(\frac{3\pi}{16}); \quad d = \cos(\frac{4\pi}{16}); \\ e = \cos(\frac{5\pi}{16}); \quad f = \cos(\frac{6\pi}{16}); \\ g = \cos(\frac{7\pi}{16}) \end{cases}$

Cette multiplication peut être écrite sous la forme :

$$Y = BCXC^TB \tag{C.2}$$

Avec $B = \begin{pmatrix} d & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & d & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & f & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g \end{pmatrix}$ et $C = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & c_a & e_a & g_a & -g_a & -e_a & -c_a & -1 \\ 1 & f_b & -f_b & -1 & -1 & -f_b & f_b & 1 \\ 1 & -g_c & -a_c & -e_c & e_c & a_c & g_c & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -a_e & g_e & c_e & -c_e & -g_e & -a_e & -1 \\ 1 & -b_f & b_f & -1 & -1 & b_f & -b_f & 1 \\ 1 & -e_g & c_g & -a_g & a_g & -c_g & e_g & -1 \end{pmatrix}$

Et pour chaque élément x_y de la matrice C, $x_y = \frac{x}{y}$.

B est une matrice diagonale la multiplication peut ainsi s'écrire sous la forme :

$$Y = CXC^T \otimes E \tag{C.3}$$

$$\text{Avec } E = \begin{pmatrix} d^2 & da & db & dc & d^2 & de & df & dg \\ ad & a^2 & ab & ac & ad & ae & af & ag \\ bd & ba & b^2 & bc & bd & be & bf & bg \\ cd & ca & cb & c^2 & cd & ce & cf & cg \\ d^2 & da & ab & dc & d^2 & de & df & dg \\ ed & ea & eb & ec & ed & e^2 & ef & eg \\ fd & fa & fb & fc & fd & fe & f^2 & fg \\ gd & ga & gb & gc & gd & ge & gf & g^2 \end{pmatrix}$$

Ainsi, l'équation permettant le calcul de la DCT 2D est divisée en deux parties. Une première partie qui représente une multiplication matricielle avec une matrice symétrique et une deuxième partie constitué par une matrice qui peut être utilisée comme facteur d'échelle. Par conséquent l'équation utilisé pour le calcul de la DCT 2D 8x8 est donnée par :

$$Y = CXC^T \otimes E \tag{C.4}$$

C.2 Evaluation du nombre d'opérateurs

En utilisant le paramètre de symétrie au niveau de la matrice C le nombre d'opérateurs arithmétiques sera réduit. En effet, la réalisation directe de cette multiplication en se basant sur le graphe de la figure, cette réalisation utilise 176 multiplicateurs et 448 additionneurs/soustracteurs. De plus, on peut aussi utiliser les propriétés de symétrie suivantes :
$g_c + a_e = 2$; $b_f - f_b = 2$;$e_g - c_a = 2$;$4 * e_a + 2 = c_g$;$4 * g_c + \frac{g_e}{4}$
Le nombre total d'opérateurs arithmétiques peut ainsi passer à 128 multiplieurs et 496 additionneurs. Le tableau table représente ainsi le nombre final d'opérateurs arithmétiques utilisés pour le calcul de la DCT 2D 8x8 et une comparaison avec des réalisations directes basés sur d'autres algorithmes.

	Loeffler	Chen	Proposée
Multliplieurs	176	208	128
Additionneur	416	464	496

Tableau C.1 – Comparaison en termes de nombre d'opérateurs entre l'algorithme proposée et celui de Chen et le Loeffler

Annexe D

Production Scientifique

Yousri Ouerhani, Maher Jridi and Ayman Alfalou. "Area-Delay Efficient FFT Architecture Using Parallel Processing and New Memory Sharing Technique". Journal of Circuits, Systems, and Computers (JCSC) vol 21, No 6 (2012).

Yousri Ouerhani, Maher Jridi, Ayman Alfalou and Christian Brosseau. "Optimized pre-processing input plane GPU implementation of an optical face recognition technique using a segmented phase only composite filter". Optics Communication, Vol 289, (2013), pp. 33-44.

Yousri Ouerhani, Maher Jridi and Ayman Alfalou. "Fast Face Recognition Approach Using a Graphical Processing unit "GPU"". IEEE International Conference on Imaging Systems and Techniques (IST), pp.80-84, 1-2 July 2010.

Yousri Ouerhani, Maher Jridi and Ayman Alfalou. "Implementation Techniques of High-order FFT into low-cost FPGA". IEEE 54^{th} International Midwest Symposium on Circuits and Systems (MWSCAS), pp.1-4, 7-10 Aug. 2011.

Maher Jridi, **Yousri Ouerhani** and Ayman Alfalou. "Scalable DCT engine for image and video multistandard compression". Submitted to SPIE 2013

M. Elbouz, A. Alfalou, C. Brosseau, M. S. Alam, S. Qasmi, **Y. Ouerhani**. "Sensitivity of optical correlation to color change of target images". SPIE 2012.

Bibliographie

[1] A. Alkholidi, "Conception d'un Système de Cryptage Image/Vidéo par Fusion d'Informations," Thèse de doctorat, UBO (ED SICMA 0373), 2007.

[2] A. V. Lugt, "Signal Detection by Complex Filtering," *IEEE Transactions on information theory*, vol. 10, pp. 139–145, 1964.

[3] A. Alfalou et C. Brosseau, *Understanding Correlation Techniques for Face Recognition : From Basics to Applications*, 2010.

[4] A. Alkholidi, A. Cottour, A. Alfalou, H. Hamam, et G. Keryer, "Real-time optical 2D wavelet transform based on the JPEG2000 standards," *The European Physical Journal Applied Physics*, vol. 44, pp. 261–272, 2008.

[5] A. Alfalou, "Implantation optique de corrélateurs multivoies temps-réel," Thèse de doctorat, Université de Rennes I, 1999.

[6] C. S. Weaver et J. W. Goodman, "A Technique for Optically Convolving two Functions," *Applied Optics*, vol. 5, pp. 1248–1249, 1966.

[7] J. W. Goodman, *Introduction to Fourier Optics*. MacGraw-Hill, New York (NY), 1968.

[8] J. Horner et P. Gianino, "Phase-Only Matched Filtering," *Applied Optics*, vol. 23, pp. 812–816, 1984.

[9] J. L. Horner, "Metrics for assessing pattern-recognition performance," *Applied Optics*, vol. 31, n. 2, pp. 165–166, 1992.

[10] A. Oppenheim et J. Lim, "The Importance of Phase in Signals," in *Proceedings of the IEEE*, vol. 69, n. 5, 1981, pp. 529–541.

[11] J. L. de Bougrenet de la Tocnaye, E. Quémener, et Y. Pétillot, "Composite Versus Multichannel Binary Phase-Only Filtering," *Applied Optics*, vol. 36, 1997.

[12] B. V. K. V. Kumar, "Tutorial Survey of Composite Filter Designs for Optical Correlators," *Applied Optics*, vol. 31, 1992.

[13] A. Alfalou, G. Keryer, et J. L. de Bougrenet de la Tocnaye, "Optical Implementation of Segmented Composite Filtering," *Applied Optics*, vol. 38, 1999.

[14] J. E. Rau, "Detection of Differences in Real Distributions," *OSA*, vol. 56, pp. 1450–1494, 1966.

[15] T. Acharya et P.-S. Tsai, *JPEG2000 Standard for Image Compression : Concepts, Algorithms and VLSI Architectures*. WILEY-INTERSCIENCE, 2005.

[16] D. S. Taubman, M. W. Marcellin, et M. Rabbani, "JPEG2000 : Image Compression Fundamentals, Standards and Practice," *Electronic Imaging*, vol. 11, 2002.

[17] G. E. Moore, "Cramming More Components into Integrated Circuits," *Electronics*, vol. 38, pp. 114–117, 1965.

[18] ITRS, ""More-than-Moore" White Paper," Rapport technique, 2009.

[19] "International Technology Roadmap for Semiconductors," pp. 8–9, 2005.

[20] ITRS, "Design. Technical report, International Technology Roadmap For Semiconductors," Rapport technique, 1999.

[21] R. Ernst, D. Ziegenbein, K. Richter, T. Braunschweig, L. Thiele, et J. Teich, "Hardware/Software Codesign of Embedded Systems - The SPI Workbench," in *Proceedings IEEE Workshop on VLSI*, 1999, pp. 9–17.

[22] G. D. Micheli, R. Ernst, et W. Wolf, Editeurs., *Readings in Hardware/Software Co-Design.* Morgan Kaufmann, 2001.

[23] VSI ALLIANCE, "Architecture Document Version 1.0," Rapport technique, 1997.

[24] M. Keating et P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs.* Springer Publishing Company, Incorporated, 2007.

[25] T. C. project at IBM Research. (2010.) The cell project. IBM Research. [En ligne] Adresse : http://www.research.ibm.com/cell/.

[26] E. Lindholm et S. Oberman. (2007) The nvidia geforce 8800 gpu. [En ligne] Adresse : http://www.hotchips.org/archives/hc19/2_Mon/HC19.02/HC19.02.01.pdf.

[27] L. Bossuet, "Exploration de l'Espace de Conception des Architectures Reconfigurables," Thèse de doctorat, Université de Bretagne Sud, 2004.

[28] M. Shafique, "Architectures for Adaptive Low-Power Embedded Multimedia Systems," Thèse de doctorat, Karlsruhe Institute of Technology, 2011.

[29] S. Park, D. R. Shires, et B. Henz, "Coprocessor Computing with FPGA and GPU," in *DoD HPCMP Users Group Conference*, 2008.

[30] V. Garcia, "Suivi d'Objets d'Intérêt dans une Séquence d'Images :Des Points Saillants aux Mesures Statistiques," Thèse de doctorat, Université de Nice - Sophia Antipolis, 2008.

[31] Owens, D. John, Luebke, David, Govindaraju, Naga, Harris, Mark, Kruger, Jens, Lefohn, E. Aaron, Purcell, et J. Timothy, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, n. 1, pp. 80–113, 2007.

[32] [En ligne] Adresse : http://www.intel.com

[33] [En ligne] Adresse : http://www.nvidia.fr

[34] L. Buatois, "Algorithmes sur GPU de visualisation et de calcul pour des maillages non-structurés," Thèse de doctorat, Institut National Polytechnique de Lorraine, 2008.

[35] Xilinx. [En ligne] Adresse : www.xilinx.com

[36] E. Ahmed et J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 288–298, 2003.

[37] M. Elbouz, A. Alfalou, et C. Brosseau, "Fuzzy logic and optical correlation-based face recognition method for patient monitoring application in home video surveillance," *Optical Engineering*, vol. 50, 2011.

[38] G. D. Voelz, *Computational Fourier Optics : A MATLAB Tutorial.* SPIE Press Tutorial Vol TT89, 2011.

[39] A. Alfalou, C. Brosseau, B. E. Benkelfat, S. Qasmi, et I. Léonard, "Towards all-numerical implementation of correlation," pp. 839 809–839 809–7, 2012.

[40] N. Gourier, D. Hall, et J. L. Crowley, "Estimating Face Orientation from Robust Detection of Salient Facial Features," in *Proceedings of Pointing 2004, ICPR, International Workshop on Visual Observation of Deictic Gestures*, 2004.

[41] J. P. Egan, *Signal detection theory and ROC analysis*, sér. Series in Cognition and Perception. Academic Press, 1975.

[42] N. Prasad, V. Shameem, U. Desai, et S. Merchant, "Improvement in target detection performance of pulse coded Doppler radar based on multicarrier modulation with fast Fourier transform (FFT)," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 151, n. 1, pp. 11 – 17, 2004.

[43] Y. Chen, Y.-W. Lin, Y.-C. Tsao, et C.-Y. Lee, "A 2.4-Gsample/s DVFS FFT Processor for MIMO OFDM Communication Systems," *IEEE Journal of Solid-State Circuits*, vol. 43, n. 5, pp. 1260 –1273, 2008.

[44] J. W. Cooley et J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Computing*, vol. 19, pp. 297–301, 1965.

[45] A. V. Oppenheim, R. W. Schafer, et J. R. Buck, *Discrete-Time Signal Processing (2nd Edition) (Prentice-Hall Signal Processing Series).* Prentice Hall, 1999, ch. 11, pp. 775–802.

[46] H. Sorensen, M. Heideman, et C. Burrus, "On computing the split-radix FFT," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, n. 1, pp. 152 – 156, 1986.

[47] D. B. Glenn, "A Fast Fourier Transform Algorithm for Real-Valued Series," *Communications of the ACM*, vol. 11, pp. 703–710, 1968.

[48] D. Kolba et T. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 25, n. 4, pp. 281 – 294, 1977.

[49] "Real-time implementation of the split-radix FFT - An algorithm to efficiently construct local butterfly modules," *Signal Processing*, vol. 71, n. 3, pp. 291–299, 1998.

[50] Y. Achouri, "Implémentation efficace de la FFT pour des communications OFDM," Mémoire de master, Université Du Québec à Trois-Rivières, Décembre 2010.

[51] C. V. Loan, *Computational Frameworks for the Fast Fourier Transform.* Society for Industrial and Applied Mathematics (SIAM), 1992.

[52] J.-P. Perez-Seva, "Les optimisations d'algorithmes de traitement de signal sur les architectures modernes parallèles et embarquées," Thèse de doctorat, Université de Nice Sophia Antipolis, 2009.

[53] H. Shousheng et M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *International Symposium on Signals, Systems, and Electronics*, 1998, pp. 257 –262.

[54] H.-G. Yeh et G. Truong, "Speed and area analysis of memory based FFT processors in a FPGA," in *Wireless Telecommunications Symposium, 2007. WTS 2007*, 2007, pp. 1 –6.

[55] L. R. Rabiner et B. Gold, *Theory and Application of Digital Signal Processing.* Inc. NJ : Prentice-Hall, 1975.

[56] E.H. Wold and A.M. Despain, "Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementations," *IEEE Transactions on Computers*, vol. C-33, n. 5, pp. 414 –426, 1984.

[57] G. Bi et E. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, n. 12, pp. 1982 –1985, 1989.

[58] H. H. He et M. Torkelson, "A new approach to pipeline FFT processor," in *The 10th International Parallel Processing Symposium, 1996., Proceedings of IPPS '96*, 1996, pp. 766 –770.

[59] B. Zhou, Y. Peng, et D. Hwang, "Pipeline FFT Architectures Optimized for FPGAs," *Int. Journal of Reconfigurable Computing*, 2009.

[60] Xilinx, "LogiCORE IP Fast Fourier Transform v7.0," Rapport technique, 2009.

[61] B. Guoan et L. Gang, "Pipelined structure based on radix-2^2 FFT algorithm," in *IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2011, pp. 2530 –2533.

[62] A. Despain, "Fourier Transform Computers Using CORDIC Iterations," *IEEE Transactions on Computers*, vol. C-23, n. 10, pp. 993 – 1001, 1974.

[63] M. A. Sanchez, M. Garrido, M. L. Lpez, et J. Grajal, "Implementing the FFT algorithm on FPGA platforms : A comparative study of parallel architectures," *International conference on Design and Technology of Integrated Systems*, 2004.

[64] L. Yang, K. Zhang, H. Liu, J. Huang, et S. Huang, "An efficient locally pipelined FFT processor," *IEEE Transaction on Cicuits and System*, vol. 53, pp. 585–589, 2006.

[65] [En ligne] Adresse : http://www.xilinx.com/ipcenter.

[66] W.-H. Chang et T. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms," *IEEE Transactions on Signal Processing*, vol. 56, n. 10, pp. 4673 –4682, 2008.

[67] L. Tan et L. Wang, "Oversampling Technique for Obtaining Higher Order Derivative of Low-Frequency Signals," *IEEE Transactions on on Instrumentation and Measurement*, vol. 60, pp. 3677–3684., 2011.

[68] M. Hassan et F. Shalash, "FPGA Implementation of an ASIP for high throughput DFT/DCT 1D/2D engine," *Circuits and Systems (ISCAS)*, pp. 1255–1258, 2011.

[69] I. Y. Soon, S. N. Koh, et C. K. Yeo, "Noisy speech enhancement using discrete cosine transform," *Speech Communication*, vol. 24, pp. 249 – 257, 1998.

[70] V. P. Vishwakarma, S. Pandey, et M. N. Gupta, "An Illumination Invariant Accurate Face Recognition with Down Scaling of DCT Coefficients," *Journal of Computing and Information Technology*, vol. 18, pp. 53–67, 2010.

[71] H. Y. Huang, C. H. Yang, et W. H. Hsu, "A Video Watermarking Technique Based on Pseudo-3-D DCT and Quantization Index Modulation," *IEEE Transactions on Information Forensics and Security*, vol. 5, n. 4, pp. 625 –637, dec. 2010.

[72] U. Koc et K. Liu, "DCT-Based Motion Estimation," *IEEE Transactions on Image Processing*, vol. 7, n. 7, pp. 948 –965, jul 1998.

[73] M. Jridi et A. Alfalou, "Joint Optimization of Low-Power DCT Architecture and Efficient Quantization Technique for Embedded Image Compression," in *VLSI-SoC : Forward-Looking Trends in IC and Systems Design*. Springer Boston, 2012, vol. 373, pp. 155–181.

[74] W. B. Pennebaker et J. L. Mitchell, *JPEG - Still Image Data Compression Standard*. Newyork : International Thomsan Publishing, 1993.

[75] G. Strang, "The Discrete Cosine Transform," *SIAM Review*, vol. 41, pp. 135–147, 1999.

[76] J. F. Blinn, "What's the Deal with the DCT ?" *In IEEE Computer Graphics and Applications*, vol. 13, pp. 78 – 83, 1993.

[77] N. Ahmed, T. Natarajan, et K. R. Rio, "Discrete Cosine Transform," *IEEE Transactions on Computer*, vol. C-23, pp. 90–93, 1974.

[78] D. Slawecki et W. Li, "DCT/IDCT Processor Design for High Data Rate Image Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, n. 2, pp. 135 –146, 1992.

[79] S. White, "Applications of Distributed Arithmetic to Digital Signal Processing : a Tutorial Review," *IEEE ASSP Magazine*, vol. 6, n. 3, pp. 4 –19, 1989.

[80] A. Madisetti et A. Willson, "A 100 MHz 2-D 8*times*8 DCT/IDCT processor for HDTV applications ," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, n. 2, pp. 158 –165, apr 1995.

[81] Y. Sungwook et E. Swartziander, "DCT Implementation with Distributed Arithmetic," *IEEE Transactions on Computers*, vol. 50, n. 9, pp. 985 –991, sep 2001.

[82] D. W. Kim, T. W. Kwon, J. M. Seo, J. K. Yu, K. Lee, J. H. Suk, et J. R. Choi, "A Compatible DCT/IDCT Architecture Using Hardwired Distributed Arithmetic," in *ISCAS*, 2001, pp. 457–460.

[83] A. Shams, W. Pan, A. Chidanandan, et M. Bayoumi, "A Low Power High Performance Distributed DCT Architecture," in *Proceedings. IEEE Computer Society Annual Symposium on VLSI*, 2002, pp. 21 –27.

[84] ISO/IEC, "Coding of Audio Visual Objects : part2. Visual," *ISO/IEC 14496 - 2(MPEG-4 Part2)*, 1999.

[85] W. Chen, C. Smith, et S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Transactions on Communications*, vol. 25, n. 9, pp. 1004 – 1009, sep 1977.

[86] B. Lee, "A New Algorithm to Compute the Discrete Cosine Transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, n. 6, pp. 1243 – 1245, dec 1984.

[87] M. Vitterli et H. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations," *Signal Processing*, vol. 6, pp. 264 – 278, 1984.

[88] N. Suehiro et M. Hatori, "Fast Algorithms for the DFT and Other Sinusoidal Transforms," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, n. 3, pp. 642 – 644, 1986.

[89] H. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, n. 10, pp. 1455 – 1461, oct 1987.

[90] C. Loeffler, A. Ligtenberg, et G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications," in *International Conference on Acoustics, Speech, and Signal Processing*, 1989, pp. 988 –991, vol. 2.

[91] P. Duhamel et H. H'Mida, "New 2nDCT Algorithms Suitable for VLSI Implementation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, apr 1987, pp. 1805 – 1808.

[92] B. G. Kashef et A. Habibi, "Direct Computation of Higher-Order DCT Coefficients from Lower-Order DCT Coefficients," *Applications of Digital Image Processing VII*, vol. 504, pp. 425 – 431, 1984.

[93] P. -Z. Lee and F. Y. Huang, "New Recursive Algorithms for Computing the 1-D and 2-D Discrete Cosine Transform," Institute of Information Science, Academia Sinica, Rapport technique, 1993.

[94] R. Haralick, "A Storage Efficient Way to Implement the Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. C-25, n. 7, pp. 764 –765, 1976.

[95] B. Tseng et W. Miller, "On Computing the Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. C-27, n. 10, pp. 966 –968, oct. 1978.

[96] M. Narasimha et A. Peterson, "On the Computation of the Discrete Cosine Transform," *IEEE Transactions on Communications*, vol. 26, n. 6, pp. 934 – 936, jun 1978.

[97] D. Hein et N. Ahmed, "On a Real-Time Walsh-Hadamard/Cosine Transform Image Processor," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-20, n. 3, pp. 453 –457, aug. 1978.

[98] H. W. Jones, D. N. Hein, et S. C. Knauer, "The Karhunen-Lobve, Discrete Cosine and Related Transforms via the Hadamard Transform," in *Int. Telemeter. Conf., Los Angeles, CA*, 1978.

[99] H. C. Chen, "A Memory-Efficient Realization of Cyclic Convolution and its Application to Discrete Cosine Transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 445 – 453, 2005.

[100] P.K. Meher, "Systolic Designs for DCT Using a Low-Complexity Concurrent Convolutional Formulation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 1041–1050, 2006.

[101] Z. Wang, "Fast Algorithms for the Discrete W Transform and for the Discrete Fourier Transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, n. 4, pp. 803 – 816, aug 1984.

[102] M. Ghanbari et D. Pearson, "Fast Cosine Transform Implementation for Television Signals," *IEE Proceedings For Communications, Radar and Signal Processing*, vol. 129, n. 1, pp. 59 –68, 1982.

[103] L.R. Rabiner and B. Gold, *Theory and application of digital signal processing*, L.R. Rabiner and B. Gold, Editeur., 1975.

[104] C.-Y. Pai, W. Lynch, et A. Al-Khalili, "Low-Power Data-Dependent 8*times*8 DCT/IDCT for Video Compression," *IEE Proceedings Vision, Image and Signal Processing*, vol. 150, n. 4, pp. 245 – 255, 2003.

[105] C.A.Christopoulos, J.Bormans, J.Cornelis, et A.N.Skodras, "The Vector-radix fast cosine transform : Pruning and complexity analysis," *Signal Processing*, vol. 43, pp. 197–205, 1995.

[106] I. E. Richardson, *H.264 and MPEG-4 Video Compression : Video Coding for Next Generation Multimedia*, 1er ed. Wiley, 2003.

[107] Mohamed AMOUD, "Design et implémentation sur FPGA d'un algorithme DES," Thèse de doctorat, Université de Montréal, 2008.

[108] Maher JRIDI, "Etude, Modélisation et Amélioration des Performances des Convertisseurs Analogique Numérique Entrelacés dans le Temps," Thèse de doctorat, Université Bordeaux 1, 2007.

[109] Benoît MARTIN, "Etude et conception dŠun étage de mise en forme dŠimpulsions ultra-large-bande de forte puissance," Thèse de doctorat, Université de Limoges, 2008.

[110] Y. LAKYS, "Filtres a frequence agile totalement actifs : Theorie generale et circuits de validation en technologie sige bicmos," Thèse de doctorat, LŠUNIVERSITÉ BORDEAUX 1, 2009.

[111] Xilinx, "LogiCORE IP DDS Compiler v4.0," Rapport technique, 2011.

[112] J. Horner, B. Javidi, et J. Wang, "Analysis of the Binary Phase-Only Filter," *Optics Communication*, vol. 91, pp. 189–192, 1992.

[113] P. Duhamel et M. Vetterli, "Fast fourier-transforms - A tutorial review and a state-of the art," *IEEE Signal Processing Society*, vol. 4, n. 19, pp. 259–299, 1990.

[114] C. Myungjin and M. Abhijit and J. Bahram, "3D passive photon counting automatic target recognition using advanced correlation filters," *Optics Letters*, vol. 36, n. 6, pp. 861–863, 2011.

[115] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[116] A. Alfalou, M. Elbouz, A. Mansour, et G. Keryer, "New spectral image compression method based on an optimal phase coding and the RMS duration principle," *Journal of Optics*, vol. 12, 2010.

[117] P. H. P. I. (PHPID). [En ligne] Adresse : http://www.ecse.rpi.edu/~cvrl/database/other_Face_Databases.htm

[118] A. Alfalou, M. Elbouz, M. Jridi, et A. Loussert, "A new simultaneous compression and encryption method for images suitable to optical correlation," *Optics and Photonics for Counterterrorism and Crime Fighting*, vol. 7486, 2009.

[119] I. Leonard, A. Alfalou, et C. Brosseau, "Spectral optimized asymmetric segmented phase-only correlation filter," *Applied Optics*, vol. 51, pp. 2638–2650, 2012.

[120] [En ligne] Adresse : http://www.youtube.com/watch?v=-Ca8apgsebY

[121] M. AMMAR, "Optimisation D'un Schéma De Codage D'image à Base D'une TCD. Application à Un Codeur JPEG Pour L'enregistrement Numérique à Bas Débit," Thèse de doctorat, 'Ecole Nationale Supérieure des télécommunications, 2004.

[122] G. Côté, B. Erol, M. Gallant, et F. Kossentini, "H.263+ : Video coding at low bit rates," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 849–866, 1998.

[123] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[124] H. David, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, n. 9, pp. 1098–1101, 1952.

[125] G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop*, vol. 23, pp. 149–162, 1979.

[126] Isabelle Léonard, "Reconnaissance des objets manufacturés dans des vidéos sous marines," Thèse de doctorat, UBO, 2012.

[127] M. Barret, *Traitement Statistique du signal*, 2009, ch. Courbe COR et performance des détecteurs à seuils, pp. 138–145.

[128] J. A. Hanley et B. J. McNeil, "The meaning and use of the area under a receiver operating cha-racteristic (roc) curve," *Radiology*, vol. 143(1), pp. 29–36, 1982.