UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR

YOUSSEF ACHOURI

IMPLÉMENTATION EFFICACE DE LA FFT POUR DES COMMUNICATIONS OFDM

DÉCEMBRE 2010

Résumé

Afin de répondre aux besoins actuels des utilisateurs de la téléphonie cellulaire, les laboratoires de recherches en télécommunication travaillent fort afin d'assurer une bonne migration de la 3G (Troisième Génération) à la 4G (Quatrième Génération). Cette dernière a pour but d'offrir plus de performances en termes de débit et plus de rapidité que l'ancienne.

La transformée discrète de Fourier (DFT – *Discrete Fourier Transform*) représente une méthode de base en traitement numérique des signaux pour l'analyse des signaux numériques. Son exécution en technologie d'intégration à très grande échelle représente un problème non trivial quand il s'agit de respecter les contraintes de l'application en termes de consommation de puissance, coût d'implémentation et vitesse de calcul.

Le futur réseau 4G LTE (*Long Term Evolution*) du 3GPP, utilisera de nouvelles techniques, comme l'OFDM (*Orthogonal Frequency Division Multiplexing*) et la MIMO (*Multi Input Multi Output*). La modulation numérique OFDM représente la modulation des futurs systèmes de communication large bande et a comme base de traitement la FFT.

Ce travail présente l'évaluation d'une nouvelle formulation de la DFT, la 'JFFT' en vue d'une implémentation plus efficace que les propositions conventionnelles. Cette nouvelle formulation comprend de nouveaux générateurs d'adressages des données et coefficients ainsi que des processeurs élémentaires (BPE – *Butterfly Processing Element*) selon les radix-*r* utilisés. Ce mémoire concerne l'évaluation des BPE de la JFFT. L'étude présentée

dans ce mémoire est une évaluation comparative de l'implémentation de coprocesseurs conventionnels et JFFT pour radix-2, radix-4 et radix-8 sur FPGA. Cette évaluation se base sur plusieurs critères : débit de la BPE, débit du coprocesseur, latence, fonction de performance globale et autre. Plusieurs coprocesseurs FFT conventionnels et JFFT ont été étudiés et évalués afin de générer des données comparatives. Les coprocesseurs FFT conventionnels sont utilisés comme référence de plancher puisqu'ils sont les plus communs. Par contre, les coprocesseurs FFT sur FPGA de l'article [ZHO09] sont considérés comme référence récente à surpasser grâce à leurs architectures améliorées. Les coprocesseurs JFFT ont été étudiés sur FPGA afin de connaître leur complexité et leurs performances. Ils représentent donc l'élément principal de ce travail.

Leur performance a dépassé nos attentes puisqu'ils offrent des résultats plus que satisfaisants; on obtient un gain de performance de 370,3 % sur la Spartan-3 et de 231,95% sur la Virtex-4 comparé à l'architecture FFT de référence R2^{2}SDF. Le calcul de la FFT est 10 fois plus rapide pour le cas 1 du Radix-8 JFFT sur Spartan-3, 8 fois sur Virtex-4 et 6 fois sur Virtex-E, comparé aux coprocesseurs FFT conventionnels et ceux de l'article de référence [ZHO09].

Finalement, ce projet permet de valider l'efficacité de la nouvelle formulation de la DFT, la 'JFFT' en termes de performances de puissances de calcul et de rapidité pour une utilisation dans des communications OFDM.

Remerciements

Je tiens à remercier sincèrement Monsieur Daniel Massicotte, professeur régulier au département génie électrique et directeur du Laboratoire des Signaux et Systèmes Intégrés(LSSI), qui, en tant que Directeur de recherche, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Mes remerciements s'adressent également à Monsieur Marwane Jaber, doctorant au LSSI, pour sa générosité et la grande patience dont il a su faire preuve malgré ses charges académiques et professionnelles.

Je tiens aussi à remercier Monsieur François Nougarou, doctorant au LSSI, pour ses précieux conseils techniques, son humour et son soutien moral.

J'exprime ma gratitude à Monsieur Adel Omar Dahmane, professeur régulier au département génie électrique et Monsieur Simon Delisle, ingénieur au département génie électrique pour leurs conseils et leur aide si précieuse.

Je n'oublie pas mes parents pour leur contribution, leur soutien et leur patience.

Enfin, j'adresse mes plus sincères remerciements à mes deux colocataires et à tous mes amis, qui m'ont toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

Merci à tous et à toutes.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

3G	3^{ième} Génération
4G	4^{ième} Génération
WCDMA	Wideband Code Division Multiple Access
UMTS	Universal Mobile Telecommunications System
HSPA	High Speed Packet Access
LTE	Long Term Evolution
3GPP	3rd Generation Partnership Program
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiplexing Access
FDM	Frequency Division Multiplexing
MIMO	Multi Input Multi Output
SISO	Single Input Single Output
ISI	Inter Symbol Interference
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
FPGA	Field Programmable Gate Array
DFT	Discrete Fourier Transform
DIT	Decimation In Time
DIF	Decimation In Frequency
LSSI	Laboratoire des Signaux et Systèmes Intégrés

JFFT	Jaber Fast Fourier Transform
VHDL	Very High Description Language
SFG	Signal Flow Graph
BPE	Butterfly Processing Element
R2MDC	Radix2 Multipath Delay Commutator
SQNR	Signal Quantization Noise Ratio
MIMO	Multi Input Multi Output
MMP	Modulation Multi-Porteuse
BPSK	Binary Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
QAM	Quadrature Amplitude Modulation
DAB	Digital Audio Broadcasting
DVB-T	Digital Video Broadcasting-Terrestrial
WLL	Wireless Local Loop
DSP	*Digital Signal Processing*
SRFFT	Split Radix Fast Fourier Transform
PFA	Prime Factor Algorithm
WFTA	Winograd Fourier Transform Algorithm
CRT	*Chineese Remainder Theorem*
VLSI	Very Large Scale Integration
PE	Processing Element
SDF	Single-path Delay Feedback
SDC	Single-path Delay Commutator

MDC	Multi-path Delay Commutator
MC-MRMDC	Multi Channel - Mixed Radix Multi-path Delay Commutator
VHSIC	Very High Speed Integrated Circuit
LUT	Look Up Table

Chapitre 1 Introduction

De nos jours, l'incroyable croissance que connait le marché des télécommunications, plus précisément la téléphonie sans fils, fait en sorte que la bande passante allouée n'est plus suffisante pour supporter le grand nombre de nouveaux utilisateurs qui ne cesse d'augmenter.

Avec le développement considérable des réseaux sans fils 3G et grâce aux nouveaux téléphones cellulaires 3G basées sur la technologie d'accès multiples WCDMA (*Wideband Code Division Multiple Access*)[BOU07], les opérateurs télécoms offrent de plus en plus de services de données (internet, visiophonie, courriel avec pièces jointes volumineuses…). La demande en termes de débits binaires devient de plus en plus accrue. Dans son tout début, la troisième génération de téléphonie mobile 3G de l'UMTS (*Universal Mobile Telecommunications System*) a été conçue pour assurer un débit binaire de l'ordre 2Mbits/s. Malgré son évolution de la 3G à la 3G+ qui maintenant assure un débit binaire de l'ordre de 15-20 Mbits/s en *downlink* (données entrantes) et 5-10 Mbits en *uplink* (données sortantes), grâce aux techniques HSPA et HSPA+ (*High Speed Packet Access*), cette génération a du mal à répondre aux besoins en termes de débits binaires actuels[FRI07].

C'est pour cette raison que les ingénieurs télécoms des industriels et des opérateurs sont déjà entrain de concevoir le futur réseau 4G (*4^{ième} Génération*). Pour la téléphonie mobile large bande, le projet actuel s'appelle le LTE (*Long Term Evolution*) développé au sein de

la 3GPP (*3rd Generation Partnership Program*). Ce projet vise à produire les normes pour le réseau de 4^{ième} génération. Le LTE sera mis en service d'ici la fin 2010-début 2011, la figure 1 montre son évolution parallèle avec les autres protocoles de communication 3G actuellement utilisés [QUA09].

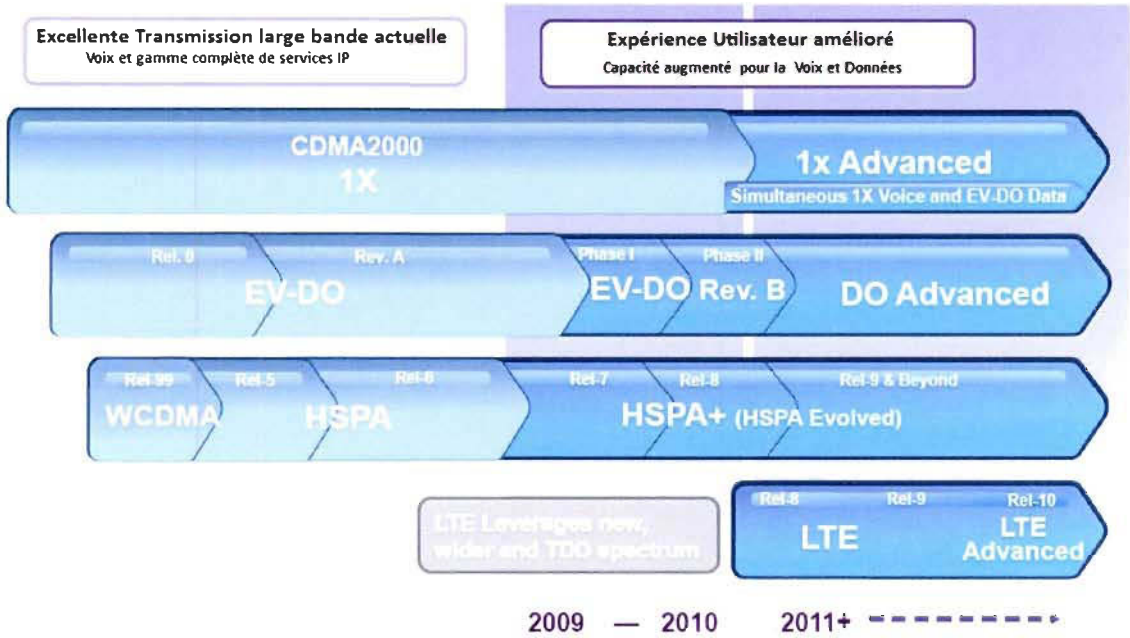


Figure 1.1 : Évolution parallèle du LTE avec la 3G

On espère ainsi atteindre des vitesses de débit binaire de l'ordre de 100 Mbits/s et plus, on verra donc le débit binaire multiplié par un facteur de 10 par rapport à la 3G+ actuelle.

Le futur réseau 4G LTE du 3GPP, utilisera de nouvelles techniques, l'OFDM (*Orthogonal Frequency Division Multiplexing*) et la MIMO (*Multi Input Multi Output*) [FAZ03]. L'OFDM a été choisie comme solution pour la modulation grâce à ces nombreux avantages dont les plus importants sont : la bonne exploitation de la bande passante ainsi que la diminution des Interférences Inter Symbole (*ISI : Inter Symbol Interference*); deux points

très essentiels pour justifier l'utilisation de cette technique. Malgré que cette technique ait été découverte depuis les années 60 dans les laboratoires de l'armée américaine, elle n'a pu voir le jour que vers la fin des années 90. Grâce aux avancés considérables qu'a connu la microélectronique, on arrive maintenant a exploité cette technique qui se base sur un des algorithmes les plus utilisés et les plus fondamentaux en traitement du signal numérique, la Transformée de Fourier Rapide connu sous le nom de FFT (*Fast Fourier Transform*). La FFT est un algorithme efficace pour calculer la TFD (*Transformée de Fourier Discrète*) puisque qu'il réduit considérablement le nombre d'opérations arithmétiques par rapport à la TFD. La FFT est rendue donc une opération essentielle dans les nouveaux systèmes de communications [WID97].

Plusieurs algorithmes FFT ont été développés depuis l'invention du premier algorithme FFT par les deux ingénieurs d'IBM, W. Cooley et John W. Tukey en 1965 [COO65]. L'évolution de la technologie VLSI a facilité l'implémentation de ces divers algorithmes FFT sur des processeurs dédiés, appelés 'Processeurs FFT'. L'intérêt de la présente étude porte sur l'implémentation au niveau matériel du Processeur FFT : l'élément essentiel pour la modulation OFDM. La cible matérielle visée est le FPGA (*Field Programmable Gate Array*). Les FPGA sont de plus en plus utilisés dans les applications télécoms, leur flexibilité, leur rapidité et leurs architectures parallèles les rendent des composants microélectroniques parfaits pour la modulation OFDM. Leur flexibilité est une propriété très recherchée puisque les nouveaux systèmes de télécommunications large bande doivent d'une part supporter les services des réseaux 3G basés sur la technologie WCDMA, et d'autre part, assurer la bonne migration vers la future génération de réseaux 4G [SAN08].

1.1 Problématique

Le calcul direct d'une TFD requière N^2 nombre d'opérations (multiplications et additions), N étant la taille de la Transformée. Avec l'évolution des diverses applications des systèmes de communications, la taille de N ne cessent d'augmenter ainsi que le nombre d'opérations [JAB09]. Durant ces 40 dernières années, plusieurs algorithmes, on été élaborés justement pour alléger le nombre d'opérations de la DFT. Comme mentionné à l'introduction, ces algorithmes sont connus sous le nom d'algorithmes FFT. Ces derniers rendent possible l'implémentation de la DFT sur puce électronique. Le premier algorithme FFT connu sous le nom de Cooley-Tukey ou Radix2 DIT (Decimation In Time) requière N.logN nombre d'opérations au lieu de N^2 pour le calcul direct de la DFT [COO65], où r est le Radix. Ceci démontre très bien la diminution concrète du nombre d'opérations à effectuer. Cet algorithme'Radix2' a apporté une nouvelle approche de calcul appelé 'papillon' connu sous le nom '*Butterfly Computation*'. Le défit maintenant, est de concevoir des Processeurs FFT très rapides pour remédier à la demande de vitesse d'exécution des nouveaux systèmes de télécommunication large bande. Des Processeurs FFT qui utilisent le moins de ressources matérielles (moins coûteux) et qui consomment le moins de puissance (économie d'énergie). Depuis, plusieurs algorithmes ont été développés et étudiés pour rendre plus efficace l'implémentation des Processeurs FFT, parmi eux le Radix-2 introduit par (Cooley-Tukey), le Split Radix Algorithm [TZE09], Winograd Fourier Transform Algorithm (WFTA)[WIN75].

Les algorithmes FFT les plus utilisés en industrie sont le Radix-2 et Radix-4 conventionnels. L'architecture de leurs BPE (*Butterfly Processor Element*) facile à implémenter les privilégie par rapport à des algorithmes FFT de Radix plus grand. Plus on utilise un Radix plus grand, plus les opérations arithmétiques pour calculer une FFT de N point augmentent. Aussi avec un Radix plus grand l'architecture du BPE devient alors plus complexe : calcul arithmétique plus complexe, plus de connexions, augmentation de nombres phases et augmentation du délai du chemin critique (*Critical Path Delay*). Mais l'avantage de l'utilisation d'un Radix plus grand, réside dans la diminution du nombre de multiplications ainsi que la diminution du nombre d'étages pour exécuter la FFT. En sachant qu'une multiplication prend plus d'espace en silicium qu'une simple addition et que la diminution du nombre d'étage diminue la charge d'interconnexion et d'accès mémoires, l'utilisation de BPE de Radix plus grand est bénéfique au niveau de l'implémentation matérielle. Quelques laboratoires de recherches se sont penchés sur l'implémentation de BPE avec Radix plus grand mais ont échoués à cause des multiplieurs qui s'ajoutent au chemin critique de la butterfly [JAB09].

Une nouvelle formulation de la DFT, développée au sein de notre laboratoire LSSI (Laboratoire des Signaux et Systèmes Intégrés), permet l'utilisation d'une architecture de BPE avec Radix plus grand sans avoir un grand impact sur les deux points les plus essentiels dans l'implémentation de la FFT : la charge de calcul à effectuer et le chemin critique (un multiplieur complexe et quelques additionneurs). Cette nouvelle solution s'appelle la JFFT [JAB09].

1.2 Objectif

L'objectif principal de ce projet, est l'évaluation de la nouvelle formulation de la DFT nommée 'la JFFT' en vue d'une implémentation plus efficace que les propositions conventionnelles. Cette évaluation est faite en termes de temps de calcul et de ressources matérielles d'implémentation en technologie FPGA. Les sous-objectifs permettant de rencontrer l'objectif principal sont :

1. Étude des algorithmes FFT et leur implémentation en vu d'une exécution dans un contexte de communication OFDM.
2. Étude de la nouvelle formulation JFFT proposé dans les travaux du LSSI. Plus spécifiquement les structures BPE Radix2, Radix4 et Radix8.
3. Étude des architecture FFT pipelinées et parallèles : avantages et techniques d'implémentation.
4. Programmation et implémentation des BPE (*Butterfly Processing Element*) FFT conventionnels et JFFT sur Matlab® et en VHDL pour évaluation des performances sur plusieurs plateforme FPGA (Virtex-E, Virtex-5, Virtex-5…).

1.3 Méthodologie

Les algorithmes conventionnels : Radix2, Radix4 et Radix8 sont étudiés, ils serviront tout au long du projet d'algorithmes de références puisqu'ils sont les plus utilisés en industrie. Leur étude nous permet de comprendre la technique avec laquelle sont extrait les SFG

(*Signal Flow Graph*) des BPE (*Butterfly Processing Element*) à partir des équations mathématiques qui régissent les algorithmes FFT conventionnels.

On programme les BPE Conventionnels sur Matlab® et on vérifie leurs bons fonctionnements. L'outil Matlab® sera notre plateforme pour effectuer les vérifications et les évaluations sur tout les algorithmes qui seront programmés et implémentés durant le projet. Après étude de la JFFT, on programme cette fois les BPE JFFT sur notre plateforme Matlab®. On testera sur cette dernière, comme pour les BPE conventionnels, leurs bons fonctionnements.

Une fois cette étape accomplie, on programmera tous les BPE conventionnels et JFFT en VHDL. L'outil choisit pour faire la simulation VHDL est Modelsim PE Student Edition 6.4b® de Mentor Graphics®. Les BPE programmés en VHDL, seront tous testés sur Modelsim® avec des programmes 'testbench' conçus pour chaque BPE.

Pour valider les structures VHDL des BPE afin de les utiliser pour constituer le Processeur FFT. On effectue une cosimulation Matlab® / Modelsim®. Matlab® générera des données normées entre 0.99 et -1 (vecteurs d'entrée et *twiddle factor*) en point fixes (8 bits, 12 bits, 14 bits, 16 bits, …, 24 bits) puis injectera les données dans les BPE programmées sur Matlab® et sur Modelsim®. L'erreur entre les sorties des BPE en points fixes seront comparées. Cette erreur validera la structure du BPE. On se servira alors de ces BPE pour constituer les coprocesseurs FFT.

Dans le projet, seule la partie réception du dispositif de communication OFDM sera considérée puisque cette partie utilise un Processeur FFT. La partie émission du module de

communication OFDM utilise par contre un Processeur IFFT (*Inverse Fast Fourier Transform*).

Les Processeurs FFT programmés dans le projet ont une architecture pipelinée. Cette architecture est choisie grâce à son bon compromis vitesse/surface sur silicium. Les architectures programmées sont : R2MDC (*Radix2 Multipath Delay Commutator*), R4MDC et R8MDC conventionnelles et JFFT. L'effet de quantification et la dynamique de signaux sont étudiés pour les différentes architectures pipelinées. Afin de calculer le SQNR (Signal *Quantization Noise Ratio*), plusieurs longueurs de données sont attribuées aux données et plusieurs méthodes d'arrondi sont utilisés pour des FFT de tailles N différentes (256, 512, 1024, 4096, 8192…)

Enfin, la synthèse FPGA des BPE ainsi que leur implémentation FPGA sont effectués sur Xilinx ISE 9.1. Les résultats obtenus démontreront l'efficacité réelle de la nouvelle formulation 'JFFT' en termes de vitesse de calcul et des ressources matériels d'implémentation en technologie FPGA.

Pour récapitulation, l'évaluation des différents algorithmes FFT s'est faite suivant une méthodologie de travail suivant quatre étapes : étude théorique de l'OFDM et des algorithmes FFT, étude de la nouvelle formulation JFFT proposé dans les travaux du LSSI [JAB09], étude des architecture FFT pipelines et parallèles et la modélisation ainsi que l'implémentation des BPE constituants les coprocesseurs FFT conventionnels et JFFT sur Matlab® et en VHDL pour évaluation des performances sur plusieurs familles de FPGA.

L'étude de la modulation OFDM nous a permis d'approfondir sur son principe de fonctionnement qui repose sur l'utilisation du coprocesseur FFT afin de démoduler le signal à transmettre en fréquentiel au niveau du récepteur. Nous avons aussi approfondi sur les

différents algorithmes FFT ce qui a facilité la compréhension du fonctionnement de chaque algorithme et puis d'effectuer un tri selon les performances de chacun.

L'étude des différentes architectures FFT pipelines et parallèles, nous a aidé à approfondir nos connaissances sur les différentes architectures pipelines et parallèles dérivées et surtout nous a permis de choisir l'architecture la plus adéquate pour notre besoin.

La cosimulation Matlab® / Modelsim® a contribué à valider le bon fonctionnement des BPE pour but de les implémenter sur FPGA.

L'évaluation des différentes architectures FFT a nécessité la modélisation directe des architectures FFT conventionnelles et JFFT en VHDL sans passer par un logiciel qui génère du code VHDL. Les BPE ont été programmés d'une façon optimale afin de réduire leur complexité et d'obtenir des résultats satisfaisants pour comparaison avec la référence.

1.4 Organisation du mémoire

Dans un premier temps, le chapitre 2 introduit à la section 2.1 la technique de modulation des nouveaux systèmes de communication 4G, l'OFDM. Son principe de fonctionnement, les éléments de bases utilisées dans l'émetteur et récepteur OFDM ainsi que ses avantages est inconvénients seront décrits. La section 2.2 présente les différentes applications de l'OFDM et leurs standards. Dans la section 2.3, on présentera le principe de la DFT, on décrira ces équations ainsi que ses éléments de base. La section 2.4 par contre présentera les algorithmes FFT les plus connus. On verra dans ces sous-sections les différents algorithmes FFT conçus et les plus utilisés. La nouvelle formulation de la DFT, la JFFT

proposé par notre laboratoire sera présentée à la section 2.5. Une comparaison entre les algorithmes conventionnels et ceux de la JFFT sera présentée dans la sous-section 2.5.1.

Le Chapitre 3 discutera les différentes architectures FFT qui peuvent être envisagés. Les sections 3.1 et 3.2 décrivent respectivement les architectures parallèles et pipeline. La section 3.3 présente l'étude de l'effet de quantification faite sur la plateforme Matlab® sur toutes les architectures étudiées. Une comparaison des résultats de cette étude sera effectuée dans la sous-section 3.3.3.

Le chapitre 4 abordera l'implémentation des BPE sur FPGA (Simulation VHDL, Synthèse FPGA). La section 4.1 introduira les principes et les éléments sensibles qui composent les BPE. La section 4.2, discutera les architectures des BPE. Ses sous-sections présenteront respectivement les architectures des BPE conventionnels et JFFT. La section 4.3 fera l'objet de l'étude théorique pour estimer les chemins critiques (temps de calcul) et les ressources matérielles. La section 4.4 exposera les résultats pratiques obtenus en termes de calcul et de ressources matérielles FPGA qui feront objet de discussions.

Enfin, le chapitre 5 présentera la conclusion générale du projet de recherche.

Chapitre 2 L'OFDM ET LA FFT

2.1 La communication OFDM

Un des éléments clés du LTE (*Long Term Evolution*) est l'utilisation de la technologie OFDM en tant que porteuse du signal, ainsi que l'utilisation de ses systèmes d'accès associés, l'OFDMA (*Orthogonal Frequency Division Multiplexing Access*) et la SC-FDMA (*Single Carrier Frequency Division Multiple Access*). L'OFDM est une méthode de Modulation Multi-Porteuse (MMP), MCM – *Multi Carrier Modulation*) utilisée grâce à ses propriétés intéressantes, particulièrement pour les systèmes de transmissions mobiles à haut débit de données tel que le WLAN, WiMax et bien sûr le 4G LTE. L'OFDM présente de nombreux avantages, y compris sa robustesse contre l'évanouissement du signal par trajets multiples et contre les interférences. Cette technologie a été utilisée depuis les années 60 par les militaires pour leurs systèmes de communications câblés. Malgré son excellent comportement spectral, l'OFDM n'a été commercialisé que récemment, principalement à cause du coût élevé de son implémentation. Le développement technologique, surtout dans le domaine de la microélectronique a rendu son implémentation commerciale possible [WID97].

2.1.1 Principe

Le principe de l'OFDM consiste à diviser sur un grand nombre de sous-porteuses le signal numérique à transmettre. Comme si l'on combinait le signal à transmettre sur un grand nombre de systèmes de transmission indépendants de fréquences porteuses différentes. La figure 2.1 montre comment les sous-porteuses sont multiplexées par la FDM (*Frequency Division Multiplexing*) [WID97]. Pour que les fréquences des porteuses soient les plus proches possibles et ainsi transmettre le maximum d'information sur une portion de bande passante donnée, l'OFDM utilise des fréquences porteuses orthogonales. Les signaux des différentes porteuses se chevauchent mais grâce à l'orthogonalité elles n'interfèrent pas entre elles. Le signal à transmettre est généralement répété sur différentes fréquences porteuses. Ainsi dans un canal de transmission avec des chemins multiples où certaines fréquences seront détruites à cause de la combinaison destructive de chemins, le système OFDM sera tout de même capable de récupérer l'information perdue sur d'autres fréquences porteuses qui n'auront pas été détruites. Chaque porteuse est modulée indépendamment en utilisant des modulations numériques: BPSK (*Binary Phase Shift Keying*), QPSK (*Quadrature Phase Shift Keying*), QAM-16 (*Quadrature Amplitude Modulation*), QAM-64.

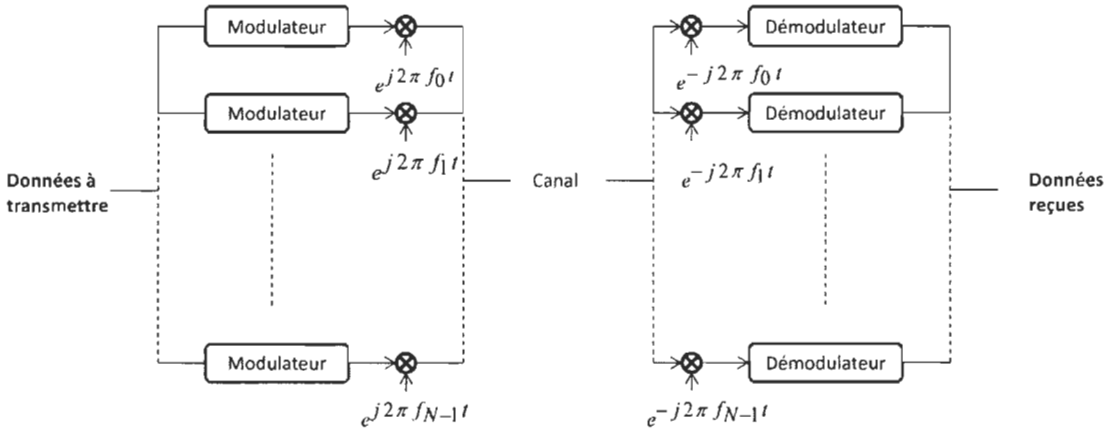


Figure 2.1 : Modulation Multi-Porteuses MMP

Dans un système OFDM [WID97] avec N sous-porteuses et un symbole de durée T, f_0 la fréquence initiale et les fréquences de sous-porteuses, f_n, sont choisies tel que:

$$f_n = f_0 + \frac{n}{T} \qquad 0 \le n \le N-1 \tag{2.1}$$

Et la fonction de base est choisie tel que

$$g_n(t) = \begin{cases} e^{j2\pi f_n t} & 0 \le t \le T \\ 0 & \text{sinon} \end{cases} \tag{2.2}$$

Le signal OFDM, $s(t)$, transmis est formé par la modulation des fonctions de base, $g_n(t)$, par les symboles à transmettre, S_n, d'alphabet complexe fini

$$s(t) = \sum_{n=0}^{N-1} S_n . g_n(t) \qquad 0 \le t \le T \tag{2.3}$$

Si les fréquences de sous-porteuses et les fonctions de bases sont choisies selon les équations (1) et (2), le spectre de fréquence de *s(t)* consistera d'un chevauchement mutuel des sous-porteuses orthogonales, comme illustré à la figure 2.2.

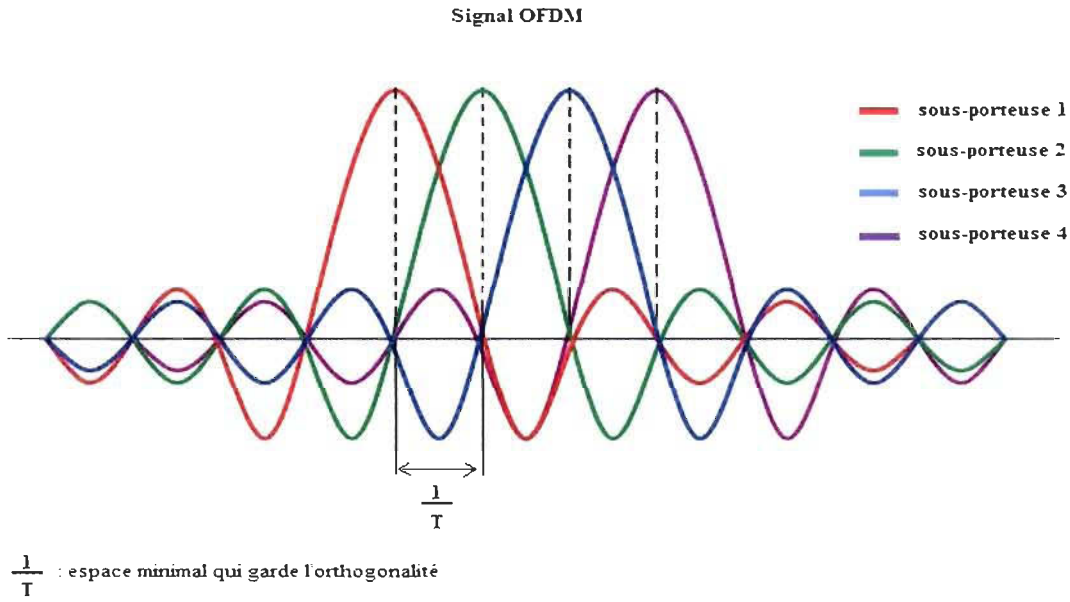


Figure 2.2 : spectre de 4 sous-porteuses dans un système OFDM

Dans le récepteur, le signal reçu *r(t)* est corrélé avec les conjugués complexes de chaque fonction de base dans le but de récupérer les symboles transmis. Par exemple, l'estimé de S_m est récupéré en corrélant *r(t)* par $g_m^*(t)$ d'après :

$$\hat{S}_m = \frac{1}{T}.\int_0^T r(t).g_m^*(t) \tag{2.4}$$

Si on considère que la transmission et l'orthogonalité sont idéals, *r(t)* = *s(t)*, l'estimé devient :

$$\hat{S}_m = \frac{1}{T}\cdot\int_0^T \left\{\sum_{n=0}^{N-1} S_n \cdot g_n(t)\right\}\cdot g_m{}^*(t) = S_m \tag{2.5}$$

Considérant une orthogonalité idéale, nous avons:

$$\int_0^T g_n(t)\cdot g_m{}^*(t) = \begin{cases} T & n=m \\ 0 & \text{sinon} \end{cases} \tag{2.6}$$

Dans la suite, nous présenterons la réalisation des équations (2.3) et (2.4).

2.1.2 Système basé sur la FFT

Une implémentation directe du système basé sur la figure 2.1 est possible mais compliquée et coûteuse. Toutefois, en exécutant la modulation en bande large et en utilisant le traitement numérique du signal, le coût de l'implémentation peut être réduit. Le signal *s(t)* transmis à travers le canal, doit être formé selon l'équation (2.3). Cette dernière peut être réécrite sous la forme :

$$s(t) = \sum_{n=0}^{N-1} S_n \cdot e^{j2\pi\frac{n}{T}t} \cdot e^{j2\pi f_0 t} \qquad 0 \le t \le T \tag{2.7}$$

Cela peut être considéré comme un signal large bande $s_B(t)$ qui est modulé par la fréquence f_0. Une version discrète du signal large bande, $s_B[k]$, peut être obtenu en configurant $t = kT/N$.

$$S_B[k] = \sum_{n=0}^{N-1} S_n . e^{j2\pi \frac{nk}{N}} \qquad k \in [0,1,..,N-1] \tag{2.8}$$

L'eq. (2.8) est reconnue comme une IFFT (*Inverse Fast Fourier Transform*) de N-point composée de séquences complexes $\{S_n\}$ de *0* à *N-1*, à l'exception du facteur scalaire *1/N*. Donc, le signal large bande $s_B(t)$ peut être construit en exécutant l'IFFT sur les symboles à transmettre. On convertit après le signal temporel discret résultant en un signal temporel continu [WID97].

Dans le but de trouver *s(t)*, le signal large bande doit être modulé par la fréquence f_0 selon l'eq. (9) :

$$s(t) = s_B(t) . e^{j2\pi f_0 t} \qquad 0 \le t \le T \tag{2.9}$$

Afin de simplifier la notation mathématique, notez que les explications ci-dessus, considèrent l'intervalle d'un seul symbole.

Du côté du récepteur les symboles transmis peuvent être récupérés par la démodulation du signal reçu *r(t)* afin d'obtenir le signal large bande $r_B(t)$. Après échantillonnage de ce dernier, on effectue la FFT sur la séquence $r_B(k)$. Si la transmission est parfaite, $r_B(t)$ est égale à $s_B(t)$ et $r_B(k)$ est égale à $s_B(k)$, et par conséquent, les résultats du calcul de la FFT sont équivalents à la séquence de symboles transmis $\{S_n\}$ de *0* à *N-1* comme souhaité

[WID97]. La figure 2.3 représente un système OFDM basé sur la FFT.

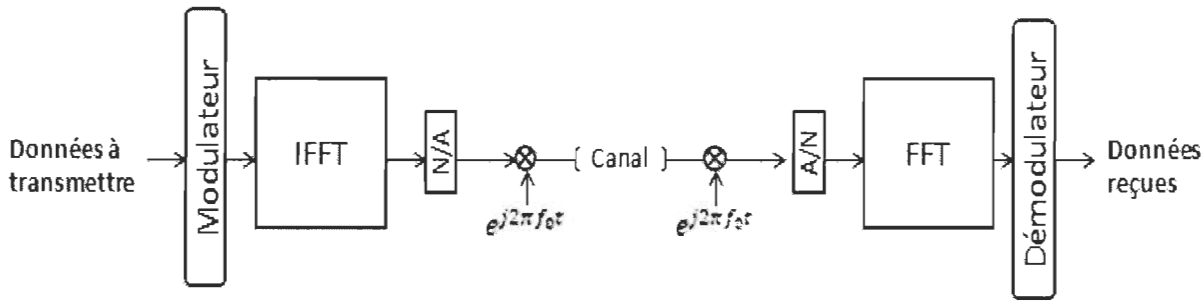


Figure 2.3 : système OFDM basé sur la FFT

La figure 2.3 représente un système OFDM simplifié. Dans la pratique, les systèmes OFDM sont implémentés en utilisant une combinaison de blocs FFT (*Fast Fourier Transform*) et l'IFFT (Inverse *Fast Fourier Transform*) qui sont mathématiquement équivalentes à la TFD (Transformées de Fourier Discrète) et la TFDI (Transformées de Fourier Discrète Inverse), respectivement, mais plus efficace en terme de rédaction de la complexité de calcul à mettre en œuvre [LOU01].

On remarque que dans la partie émission, l'IFFT est utilisée puisque qu'on passe du domaine fréquentiel au domaine temporel. Au niveau de l'émetteur, le système OFDM traite les symboles de la source (par exemple, des symboles QPSK ou QAM) dans le domaine fréquentiel. Ces symboles sont utilisés comme entrées du bloc IFFT qui amène le signal dans le domaine temporel. Le block IFFT prend ces N symboles à la fois, N étant le nombre de sous-porteuses dans le système. Chaque symbole d'entrée a une période T appelé période de symbole. D'après les équations (2.1) et (2.2), on rappelle que les

fonctions de bases pour l'IFFT sont N sinusoïdes orthogonales de fréquences différentes. Chaque symbole d'entrée possède un poids complexe pour chaque fonction de base sinusoïdale. Étant donné que les symboles d'entrée sont complexes, la valeur du symbole détermine à la fois l'amplitude et la phase de la sinusoïde pour cette sous-porteuse. La sortie IFFT est la somme de toutes les sinusoïdes N. Ainsi, le bloc IFFT fournit un moyen simple pour moduler les données sur des sous-porteuses orthogonales N. Le bloc de N échantillons de sortie de l'IFFT forme un seul symbole OFDM. La durée du symbole OFDM est NT, où T est la période du symbole [LOU01].

Après quelques traitements supplémentaires, le signal temporel qui résulte de l'IFFT est transmis à travers le canal. Au niveau du récepteur, le bloc FFT est utilisé pour traiter le signal reçu et le convertir en spectre fréquentiel. Idéalement, la sortie de la FFT s'agira du signal fréquentiel initial (données à transmettre) qui a été transformé en signal temporel par l'IFFT au niveau de l'émetteur pour le but d'être transmis à travers le canal. Lorsqu'on reporte les symboles reçus dans le plan complexe, les échantillons de sortie FFT formeront une constellation, comme la 16-QAM. Toutefois, il n'y a pas de notion de constellation pour un signal temporel. Donc, lorsque ce dernier est placé dans le plan complexe, il forme un nuage de points sans forme régulière. Ainsi, au niveau du récepteur, tout traitement du signal qui utilise le concept de constellation doit s'effectuer dans le domaine fréquentiel. Le schéma de la figure 2.4 [LOU01] représente le passage du domaine fréquentiel au domaine temporel dans un système OFDM utilisant la modulation QAM [LOU01].

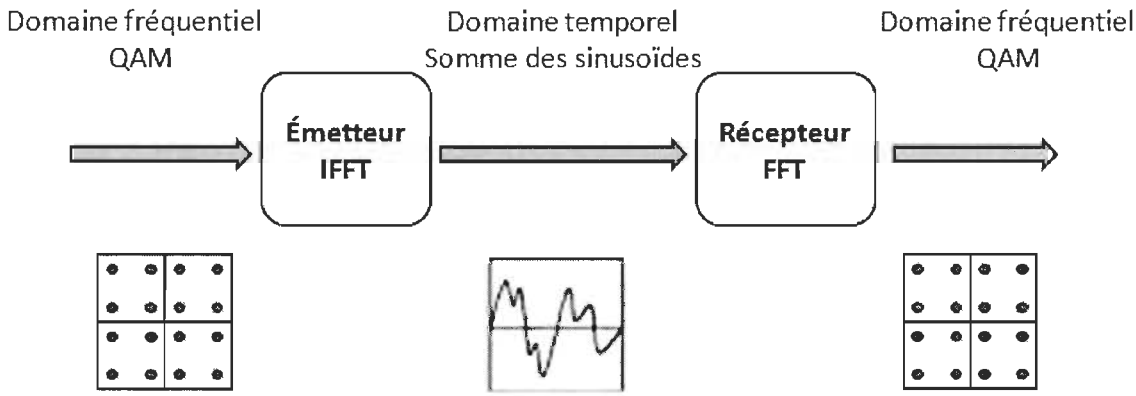


Figure 2.4 : Passage du domaine fréquentiel au domaine temporel dans un système OFDM

2.2 Applications et Standards

La modulation OFDM est utilisée dans plusieurs systèmes de communication tels que les systèmes européens de diffusion audio et vidéo numérique, le DAB (*Digital Audio Broadcasting*) et le DVB-T (*Digital Video Broadcasting-Terrestrial*). L'OFDM est aussi utilisée dans les réseaux sans fil : WLAN (*Wireless Local Area Network*) ainsi que les WLL (*Wireless Local Loop*). Les tableaux 2.1 à 2.3 [FAZ03], ci-dessous, résument les paramètres clés des standards de ces systèmes de communications multi-porteuses.

Tableau 2.1 Standards des systèmes de diffusion DAB et DVB-T

Paramètres	DAB			DVB-T	
Bande Passante	1.5 MHz			8 MHz	
Nombre de sous-porteuses N	192 (256 FFT)	384 (512 FFT)	1536 (2k FFT)	1705 (2k FFT)	6817 (8k FFT)
Durée de symbole T	125 µs	250 µs	1 ms	224 µs	896 µs
Espacement entre porteuses	8 kHz	4 kHz	1 kHz	4.464 kHz	1.116 kHz
Intervalle de garde	31 µs	62 µs	246 µs	$T/32$, $T/16$, $T/8$, $T/4$	
Modulation	D-QPSK			QPSK, 16-QAM, 64-QAM	
Débit Maximum	1.7 Mbit/s			31.7 Mbit/s	

Tableau 2.2 Standards du WLAN

Paramètres	IEEE 802.11a
Bande Passante	20 MHz
Nombre de sous-porteuses N	52 (64 FFT)
Durée de symbole T	4 µs
Espacement entre porteuses	312.5 kHz
Intervalle de garde	0.8 µs
Modulation	BPSK, QPSK, 16-QAM, et 64-QAM
Débit Maximum	54 Mbit/s

Tableau 2.3 Standards du WLL

Paramètres	IEEE 802.16a	
Bande Passante	De 1.5 à 28 MHz	
Nombre de sous-porteuses N	256 (OFDM mode)	2048 (OFDMA mode)
Durée de symbole T	De 8 à 125 µs (dépendamment de la bande passante)	De 64 à 1024 µs (dépendamment de la bande passante
Intervalle de garde	De 1/32 jusqu'à 1/4 de T	
Modulation	QPSK, 16-QAM, et 64-QAM	
Débit Maximum (canal de 7Mhz)	Jusqu'à 26 Mbit/s	

Les tables 2.1 à 2.3 montrent les paramètres essentiels des différents standards des systèmes de communications multi-porteuses. On remarque, pour chaque standard, la bande passante alloué, la modulation utilisée ainsi que le nombre de sous-porteuses N utilisé. Ce dernier, qui varie selon l'application de 64 points à 8096 points et plus, représente le nombre de points que la FFT devra calculer.

L'objectif de cette section, est d'introduire à la modulation OFDM et d'expliquer comment la FFT et l'IFFT peuvent être utilisées dans les systèmes OFDM. Plusieurs aspects de l'OFDM ne seront pas discutés dans ce mémoire (comme l'ajout de l'intervalle de garde, les interférences inter-symboles…) puisque c'est au-delà des objectifs du projet qui est l'implémentation efficace de la FFT.

2.3 Principe de la DFT

Dans le domaine du traitement numérique du signal, la Transformée de Fourier Discrète (TFD), ou en anglais DFT (*Discrete Fourier Transform*) joue et jouera un rôle très

important dans l'analyse, la conception et l'implémentation des algorithmes et systèmes futurs du traitement du signal temporel discret. La DFT est un procédé mathématique qui reste au centre du traitement prenant place à l'intérieur d'un processeur de signal numérique. La DFT peut, par exemple, calculer la réponse fréquentielle d'un signal, et peut servir comme une étape intermédiaire dans des techniques plus élaborées de traitement numérique du signal comme la modulation OFDM abordée dans la section 2.1 de ce chapitre [WAN07], [JAB08]. Son application est aussi utilisée dans plusieurs domaines technologiques tels que les télécommunications, le biomédical, le traitement séismique, etc. …

La DFT d'un signal discret *x(n)* peut être directement calculée par l'équation. (2.10),

$$X(n) = \sum_{n=0}^{N-1} x_{(n)} W_N^{nk} \quad , \quad k \in [0, \dots, N-1] \tag{2.10}$$

Avec *x(n)* et *X(n)*, respectivement, séquence d'entrée et séquence de sortie. Les deux séquences *x(n)* et *X(n)* sont des séquences de nombres complexes. *N* est la longueur de la transformée, elle est de valeur finie pour limiter les calculs exécutés par le processeur numérique qui lui aussi est limité par la taille de sa mémoire.

L'équation de définition de la DFT fournit une relation entre deux ensembles de N nombres complexes qui s'écrit d'une manière commode sous une forme matricielle, en posant :

$$W_N^{nk} = e^{-j(2\pi/N)nk} \quad \text{avec} \quad j^2 = -1 \tag{2.11}$$

Les affixes des nombres, les W_N^{nk}, appelés *coefficients de base de la DFT* ou plus communément (en anglais '*Twiddle factor*'), se trouvent sur le cercle unité comme le montre la figure 2.5.

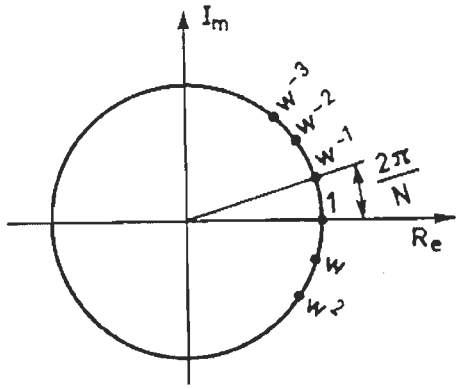


Figure 2.5 affixes des coefficients de la DFT

Plusieurs propriétés caractéristiques de ce coefficient sont utilisées [VAN03], par exemple :

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n} \tag{2.12}$$

$$W_N^{2kn} = W_{N/2}^{kn} \tag{2.13}$$

$$W_N^{k(N-n)} = (W_N^{kn})^* \tag{2.14}$$

où * désigne le complexe conjugué.

Les équations (2.15) et (2.16) montrent comment on déduit la forme matricielle de la DFT directe.

$$X(k) = \sum_{n=0}^{N-1} x(n) w_N^{nk} , \quad k \in [0, \dots, N-1] \ <=>$$

$$X(0) = x_{(0)}W_N^0 + x_{(1)}W_N^0 + x_{(2)}W_N^0 + \cdots + x_{(N-1)}W_N^0$$

$$X(1) = x_{(0)}W_N^0 + x_{(1)}W_N^1 + x_{(2)}W_N^2 + \cdots + x_{(N-1)}W_N^{(N-1)}$$

$$X(2) = x_{(0)}W_N^0 + x_{(1)}W_N^2 + x_{(2)}W_N^4 + \cdots + x_{(N-1)}W_N^{2(N-1)} \quad (2.15)$$

$$X(3) = x_{(0)}W_N^0 + x_{(1)}W_N^3 + x_{(2)}W_N^6 + \cdots + x_{(N-1)}W_N^{3(N-1)}$$

$$\cdot$$

$$\cdot$$

$$X(N-1) = x_{(0)}W_N^0 + x_{(1)}W_N^{(N-1)} + x_{(2)}W_N^{2(N-1)} + \cdots + x_{(N-1)}W_N^{(N-1)^2}$$

<=>

$$X(k) = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & & W_N^0 & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & \cdots & W_N^{(N-2)} & W_N^{(N-1)} \\ W_N^0 & W_N^2 & W_N^4 & & W_N^{2(N-2)} & W_N^{2(N-1)} \\ & \vdots & & \ddots & \vdots & \\ W_N^0 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-2)(N-1)} & W_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x_{(0)} \\ x_{(1)} \\ x_{(2)} \\ \vdots \\ \vdots \\ x_{(N-1)} \end{bmatrix} \quad (2.16)$$

qui peut être représenté sous la forme matricielle suivante :

$$X(k) = [B_N][x_n] \quad (2.17)$$

Avec $[B_N]$ la matrice des coefficients et $[x_n]$ la matrice d'entrée de la DFT.

Donc la Transformée de Fourier Discrète est une décomposition d'un signal échantillonné, composé de sinusoïdes (exponentielles complexes). À partir de l'équation (2.10), on constate que la complexité de calcul de la DFT s'exprime en $O(N^2)$, soit il augmente avec le carré de la longueur de la transformée et donc devient cher pour des N larges. Ainsi par exemple, pour parachever une multiplication complexe, plusieurs additions et soustractions, en plus de quelques autres opérations, sont nécessaires. C'est pour ces raisons que la DFT

n'est pas utilisée pour le traitement du signal temps réel. En exploitant les propriétés de symétrie et de périodicité de la DFT, plusieurs méthodes efficaces ont été développées pour calculer la DFT et ainsi diminuer significativement la charge de calcul. Dans ce chapitre, nous aborderons ces méthodes desquelles résultent des algorithmes rapides pour le calcul de la DFT connus sous le nom de FFT (*Fast Fourier Transform*) [WID97] [JAB08].

2.4 Algorithmes FFT

Pendant plusieurs décennies, la principale préoccupation des chercheurs était de développer un algorithme FFT qui minimise le nombre d'opérations requises. Depuis que les deux ingénieurs d'*IBM*, *Cooley* et *Tukey*, ont présenté leur approche montrant que le nombre de multiplications requis pour calculer la DFT peut être réduit significativement en exploitant les propriétés de la figure 2.5 formulant la variantes des algorithmes FFT, l'intérêt a surgi à la fois pour chercher des applications pour cette puissante transformée et trouver divers implémentations logicielles et matérielles pour la FFT.

Il existe des algorithmes FFT qui calculent la DFT plus rapidement et plus efficacement et réduisent significativement les tailles des puces où seront implémentés tel que les DSP (*Digital Signal Processing*) et FPGA (*Field Programmable Gate Array*). L'exécution de ses algorithmes en technologie d'intégration à très grande échelle représente un problème non trivial quand il s'agit de respecter les contraintes de l'application en termes de

consommation de puissance, coût d'implémentation et vitesse de calcul. D'où, la nécessité d'augmenter les performances de ces algorithmes en trouvant une solution matérielle.

Tous les algorithmes cités dans ce chapitre ont été élaborés pour réduire la charge de calcul en termes de multiplications et d'additions complexes et chacun d'entre eux possède ces propres caractéristiques. Ces méthodes d'algorithmes rapides sont toutes basées sur une approche appelée 'diviser pour régner'. Le principe de cette méthode est de diviser un grand problème en petit sous-problèmes facile à résoudre.

Dans le cas de la FFT, la division en sous-problèmes signifie que les données d'entrée $x_{(n)}$ sont divisées en sous-ensembles sur lesquels la DFT est calculée. Ensuite, à partir des résultats intermédiaires, la DFT des données initiales est reconstruite. Si cette stratégie est appliquée récursivement sur les DFT intermédiaires, on obtient un algorithme FFT [COO65], [WID97], [JAB08].

2.4.1 Algorithmes à facteur commun (algorithmes conventionnels)

Les algorithmes les plus connus et les plus utilisés sont les algorithmes FFT où N est une puissance entière de deux, $N = 2^M$, où M un entier. Grâce à ces algorithmes, il est possible de réduire le nombre d'opérations nécessaires à un ordre de grandeur de $N.log_2(N) = N \times M$ [VAN03].

Le tableau 2.4 donne un résumé des valeurs N^2, $N.log_2(N)$, $N/log_2(N)$ pour diverses valeurs de N. La dernière colonne montre de combien la vitesse de calcul peut être augmentée par l'utilisation de la FFT au lieu du calcul direct de DFT [VAN03].

Tableau 2.4 Comparaison du nombre d'opérations dans un calcul direct de la DFT et dans la FFT [VAN03]

N	N^2	$N.log_2(N)$	$N/log_2(N)$
2	4	2	2.00
4	16	8	2.00
8	64	24	2.67
16	256	64	4.00
32	1024	160	6.40
64	4096	384	10.67
128	16 384	896	18.29
256	65 536	2048	32.00
512	262 144	4608	56.89
1024	1 048 576	10 240	102.40
2048	4 194 304	22 528	186.18

Dans d'autres littératures, on emploie le mot Radix-r ou base-b, qui signifient le nombre d'entrée de chaque sous-systèmes DFT qui compose l'algorithme FFT. Toutefois les algorithmes sont nommés Radix-r. Par exemple si $r = 2$, l'algorithme s'appellera Radix-2 où $N = r^M = 2^M$. Dans la section suivante, nous verrons de plus près, les trois algorithmes conventionnels FFT les plus communs: Radix-2, Radix-4 et Radix-8.

2.4.1.1 Radix 2

L'algorithme FFT à base de Radix-2 est un des algorithmes possédant la structure butterfly la plus simple pour le calcul de la DFT. Radix-2 qui signifie en français radical-2 ou base-2, veut que la taille *N* de la FFT à calculer soit à base de 2. Sa structure est très simple, appelé structure Butterfly (papillon) à cause de son schéma en forme de papillon.

Il existe deux algorithmes le DIT FFT (*Decimation In Time*, Décimation dans le temps) et DIF FFT (*Decimation In Frequency*, Décimation en fréquence). Dans ce travail nous nous intéresserons uniquement à la version DIT car elle présente, selon [CHA08], une meilleure précision en virgule fixe que la version DIF.

Le Radix-2 utilise la technique 'diviser pour régner' [COO65], donc à l'aide de cette technique, il divise la FFT en sous système de N/2 points, puis calcule chaque sous-système tout seul pour obtenir à la fin les composantes du spectre fréquentiel du signal.

La démonstration ci-dessous montre la méthode 'diviser pour régner' utilisée dans l'algorithme Radix-2 ainsi que la méthode appliquée pour obtenir la structure en papillons.

Prenons un signal discret $x[n]$ de longueur N (N est pair). Définissons alors deux nouveaux signaux $x_1[n]$ et $x_2[n]$ chacun de longueur $N/2$ et qui sont constitués par les échantillons pairs et impairs de $x[n]$ respectivement :

$$\left.\begin{aligned} x_1[n] &= x[2n] \\ x_2[n] &= x[2n+1] \end{aligned}\right\} \text{ avec n = 0, 1, … , (N/2) - 1} \qquad (2.18)$$

En utilisant les coefficients W_N, on trouve comme DFT de $x[n]$:

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk} = \sum_{n=0(n\ pair)}^{N-1} x[n]W_N^{nk} + \sum_{n=0(n\ impair)}^{N-1} x[n]W_N^{nk}$$

$$= \sum_{n=0}^{(\frac{N}{2})-1} x[2n]W_N^{2nk} + \sum_{n=0}^{(\frac{N}{2})-1} x[2n+1]W_N^{(2n+1)k} \quad (2.19)$$

Et en utilisant (2.13) et (2.18), on trouve :

$$X[k] = \sum_{n=0}^{(\frac{N}{2})-1} x_1[n]W_{N/2}^{nk} + W_N^k \sum_{n=0}^{(\frac{N}{2})-1} x_2[n]W_{N/2}^{nk}$$

$$= X_1[k] + W_N^k\ X_2[k] \quad (2.20)$$

$X_1[k]$ et $X_2[k]$ représentent les TFD à N/2 points de $x_1[n]$ et $x_2[n]$ [VAN03].

À partir de l'équation (2.20), on constate que le calcul de X[k] peut se faire à l'aide d'une structure en treillis de la FFT, plus communément appelée 'butterfly diagram' littéralement 'structure en papillon'. Cette dernière représente une opération dans laquelle, partant de deux nombre complexes *A* et B, on calcule deux nouveaux nombres complexes *Y* et *Z* tels que :

$$Y = A + W_N^p\ B \quad et \quad Z = A - W_N^p\ B \quad (2.21)$$

Avec W_N le *twiddle factor* bien connu et *p* un entier compris entre 0 et $N-1$.

L'équation (2.21) est représentable schématiquement, la figure 2.5 montre son schéma :

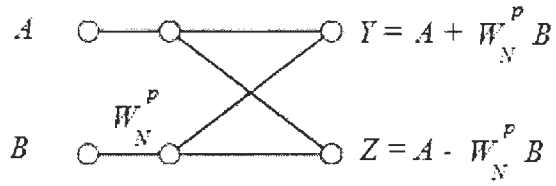


Figure 2.5 structure DIT butterfly

La figure 2.5 représente le schéma d'une structure butterfly DIT de la FFT. On remarque que cette structure, avec $p = 0$, représente exactement la relation qui existe entre les échantillons à l'entrée $x[0] = A$ et $x[1] = B$ et les échantillons en sortie $X[0] = Y$ et $X[1] = Z$, dans une DFT à 2 points. La figure 2.6 quant à elle, représente la structure d'une butterfly DIF.

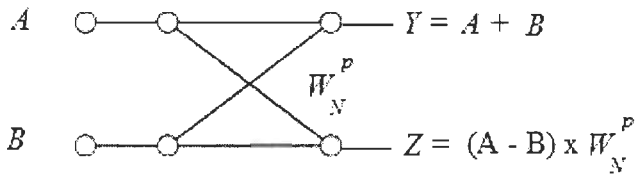


Figure 2.6 structure butterfly DIF

Ci-dessus, les deux structures butterfly utilisées respectivement pour le DIT-FFT et le DIF FFT. Les facteurs de rotation dits «*twiddle factor*» pour le DIT sont multipliés par l'entrée B puis le produit est, ou ajouté ou retranché, à A. Par contre pour le DIF on additionne les entrées A et B pour la première sortie puis pour la deuxième sortie, on soustrait B de A et on multiplie par le *twiddle factor*.

En général, la structure butterfly est l'opération de base du Radix-r PE (Processeur Élémentaire) qui compose l'algorithme FFT, où r entrées sont combinées pour donner r sorties via l'opération déjà décrite dans l'équation (2.17).

Cette dernière peut être réécrite comme tel :

$$X = B_r \; x \tag{2.22}$$

Où $\mathbf{x}=[x(0), x(1),...x(r-1)]^T$ est le vecteur d'entrée et $X=[X(0), X(1),...X(r-1)]^T$ est le vecteur de sortie. $\mathbf{B}_r$ est une matrice r × r qui peut être exprimée pour le DIT-FFT:

$$B_r = W_N^r \times T_r \tag{2.23}$$

Et pour le DIF-FFT :

$$B_r = T_r \times W_N^r \tag{2.24}$$

La matrice,

$$W_N^r = diag(1,\ w_N^p\ ,\ w_N^{2p}, \ldots\ldots\ldots,\ w_N^{(r-1)p}\ ) \tag{2.25}$$

représente la matrice diagonale des *twiddle factor*, avec $p = 0, 1, ..., N/r^s-1$,

$s = 0,1,...log_r\ N-1$ et T_r est une matrice qui représente le adder-tree dans le butterfly [WID97].

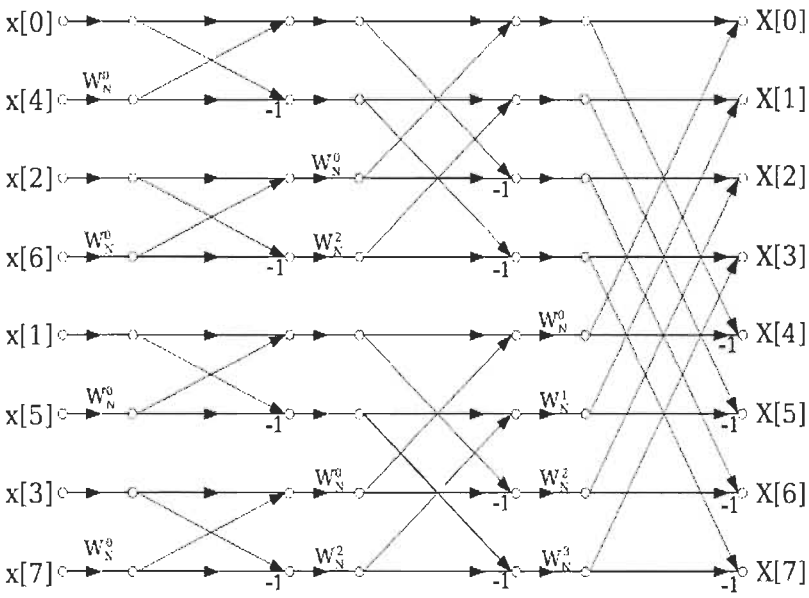
$$\mathbf{T}_r = \begin{bmatrix} w^0 & w^0 & w^0 & - & w^0 \\ w^0 & w^{N/r} & w^{2N/r} & - & w^{(r-1)N/r} \\ w^0 & w^{2N/r} & w^{4N/r} & - & w^{2(r-1)N/r} \\ - & - & - & - & - \\ w^0 & w^{(r-1)N/r} & - & - & w^{(r-1)^2N/r} \end{bmatrix} = \left[\mathbf{T}_{r(l,m)}\right] \tag{2.26}$$

On redéfinit $[\mathbf{T}_{r\,(l,m)}]$ comme l'élément de la matrice $\mathbf{T}_r$ à la $l^{\text{ième}}$ ligne et la $m^{\text{ième}}$ colonne. Et on réécrit (2.26) comme :

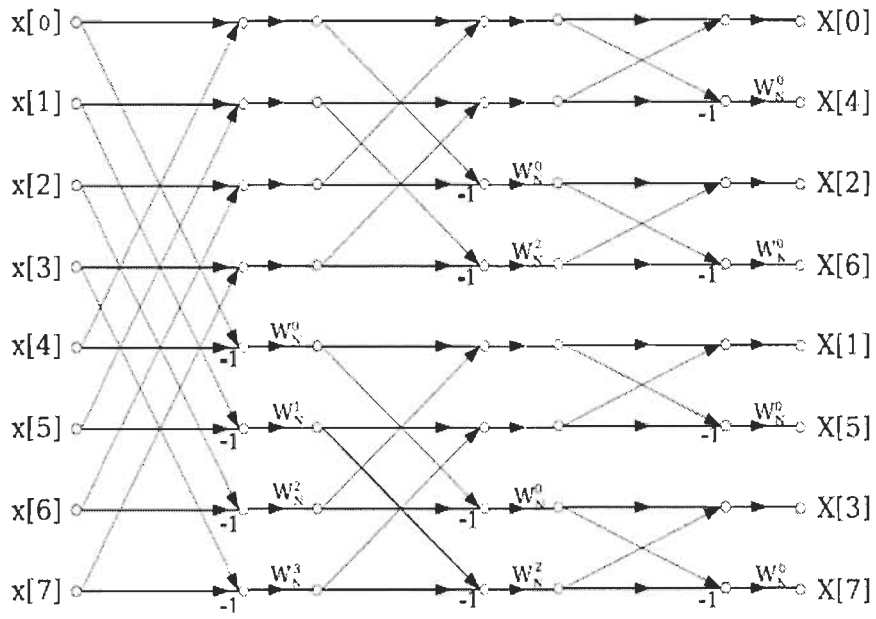
$$[T_r]_{l,m} = W_N^{\left[\left[\frac{lmN}{r}\right]\right]_N} \tag{2.27}$$

Avec $l = m = 0, 1, r-1$ et $[\![x]\!]$ représente l'opération x modulo N

La figure 2.7 représente les structures des deux algorithmes Radix-2 DIT et DIF :



(a)



(b)

Figure 2.7 Diagramme de la FFT à base de Butterfly Radix-2 de type DIT (a) et DIF (b) pour N=8.

Dans la figure 2.7 ci-dessus, on représente la structure du Radix-2 DIT et DIF. On constate que la division en petit sous-système, nommé butterfly (structures papillons), commence du côté de l'entrée (signal temporel), d'où le nom de l'algorithme «décimation dans le temps». Le Radix-2 DIF possède les mêmes propriétés que le précédent sauf que la division se fait au niveau de la fréquence «décimation en fréquence». La taille de la FFT est N=8. Pour un Radix-2, la FFT se calcule en log2 N étages, soit ici en trois étages puisque N=8 et on aura N/2 butterfly (sous-systèmes) à calculer par étage, soit 4 pour notre cas ici. On ajoute aussi que le Radix-2, DIT ou DIF, réduit l'ordre de calcul de N^2 à $N/2\ \log_2 N$ multiplications complexes et de N^2-N à $N\ \log_2 N$ additions complexes.

Le Radix-2 est l'algorithme FFT le plus utilisé actuellement. Il apparaît dans plusieurs articles de l'IEEE [WAN07], [TAN02], [TAN04], grâce à sa simple architecture. Plusieurs travaux sont effectués sur cet algorithme afin de trouver des méthodes plus élaborées pour réduire le nombre de multiplications et d'additions complexes.

2.4.1.2 Radix4

Le Radix-4 est un algorithme FFT intéressant puisqu'il réduit le nombre de multiplications complexes de 25% par rapport au Radix-2 [WID97], [BEL98]. À condition que la taille de la FFT soit une puissance de 4.

La matrice adder-tree et la matrice des *twiddle factor* sont données ci-dessous.

$$\cdot\ T_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \qquad \mathrm{W}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & w_N^P & 0 & 0 \\ 0 & 0 & w_N^{2P} & 0 \\ 0 & 0 & 0 & w_N^{3P} \end{bmatrix} \tag{2.28}$$

Les éléments de la matrice T_4, est obtenue par une méthode géométrique, figure 2.8 ci-dessous.

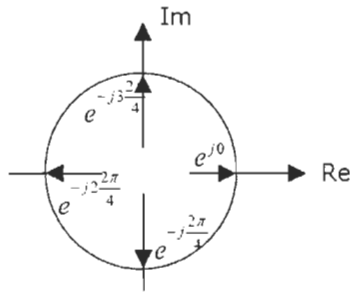


Figure 2.8 : méthode géométrique

Les *twiddle factor* sont calculés à partir du cercle trigonométrique de la figure 2.8, ils prennent donc les valeurs (1, -1, j, -j). Après quelques simplifications :

$$T_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad (2.29)$$

À partir de T_4 et W_4, on obtient la structure simplifié du *butterfly* Radix-4, représenté dans la figure 2.9 ci-dessous.

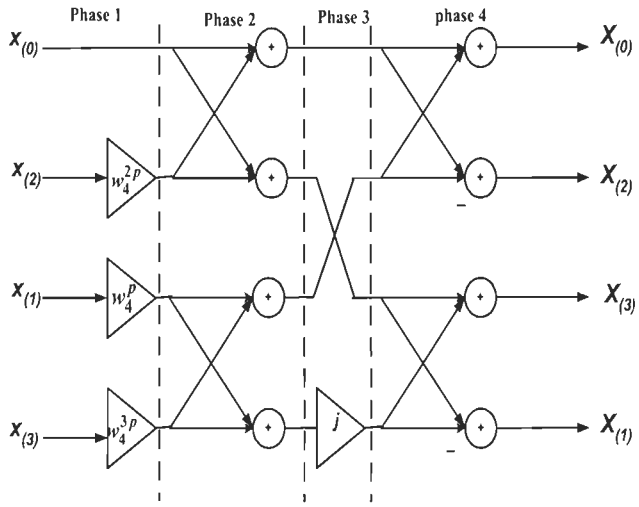


Figure 2.9 : PE_Butterfly du Radix-4 DIT

La figure 2.9 représente le SFG (*Signal Flow Graph*) du PE Radix-4 DIT. Tous les signaux d'entrées et de sorties du SFG sont de valeurs complexes. Puisque c'est un algorithme DIT, on constate que les twidles factor sont placés aux PIN d'entrées du PE.

Les entrées du PE ne sont pas dans l'ordre pour des considérations de dessin.

Note : Le nombre de multiplications complexes est réduit par rapport au Radix-2 pour une taille de FFT plus grande voir le tableau 2.5.

2.4.1.3 Radix 8

La plupart des implémentations des algorithmes 'Cooley et Tukey' utilisent des versions d'algorithmes Radix-2 ou Radix-4, bien que les Radix supérieurs comme le Radix-8 ont plusieurs avantages évidents par exemple, moins d'accès mémoire, et moins de multiplications complexes, voir ci-dessous le tableau 2.5 [WID97]. La raison est sans doute que l'implémentation de Radix supérieurs, exige une grande complexité du Processeur Élémentaire (PE) et impose une restriction sur la longueur N de la transformée. D'où la nécessité de s'intéresser de plus près à ces Radix supérieurs pour les rendre moins complexes donc plus facile à implémenter. Dans ce projet, on s'intéresse au Radix-8 et un des buts principaux est de comparer les deux PE Radix-8 conventionnel et JFFT en termes de ressources matérielles et de vitesse de calcul [WID97].

Tableau 2.5 Nombre d'opérations d'une FFT de 4096 points pour différents Radix

Opération	Radix-2	Radix4	Radix-8
Multiplications complexes	22528	15360	10752
Multiplications réelles	0	0	8192
Additions complexes	49152	49152	49152
Additions réelles	0	0	8192
Accès mémoire	49152	24576	16384

Le tableau 2.5 confirme que le Radix-8 utilise moins de multiplications complexes et moins d'accès mémoire. Ces deux propriétés le rendent un algorithme très intéressant à implémenter.

Ci-dessous les matrices adder-tree (T_8) et twidle factor (W_8) de l'algorithme Radix-8 :

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w_8 & -j & -jw_8 & -1 & -w_8 & j & jw_8 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -jw_8 & j & w_8 & -1 & jw_8 & -j & -w_8 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -w_8 & -j & jw_8 & -1 & w_8 & j & -jw_8 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & jw_8 & j & -w_8 & -1 & -jw_8 & -j & w_8 \end{bmatrix} \quad (2.30)$$

$$W_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_N^p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_N^{2p} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_N^{3p} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_N^{4p} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_N^{5p} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_N^{6p} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_N^{7p} \end{bmatrix} \quad (2.31)$$

On constate que la matrice adder-tree T_8 contient des multiplications par $W_8 = e^{-j2\pi/8} = e^{-j\pi\backslash 4}$, $-W_8$, jW_8 et $-jW_8$ qui ne sont pas des multiplications triviales. Des méthodes pour implémenter ces multiplications seront discutées dans le chapitre 4. Ces dernières permettent de diminuer le nombre d'additions et multiplications complexes. On constate aussi que la matrice T_8 exige 56 additions complexes. Pourtant en la factorisant, on peut

effectuer le même calcul en 24 additions complexes. La factorisation est démontrée en (2.32) ci-dessous.

$$T_8 = \begin{bmatrix} 1&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&-j \\ 0&1&0&0&0&j&0&0 \\ 0&0&1&0&0&0&1&0 \\ 1&0&0&0&-1&0&0&0 \\ 0&0&0&1&0&0&0&j \\ 0&1&0&0&0&-j&0&0 \\ 0&0&1&0&0&0&-1&0 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&-j \\ 0&1&0&0&0&j&0&0 \\ 0&1&0&0&0&-j&0&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&-1&0&0&0&1&0 \\ 0&0&0&-jw_8&0&0&0&-w_8 \\ 0&0&0&jw_8&0&0&0&w_8 \end{bmatrix} \cdot \begin{bmatrix} 1&0&0&0&1&0&0&0 \\ 1&0&0&0&-1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&-1&0&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&1&0&0&0&-1&0 \\ 0&0&0&1&0&0&0&1 \\ 0&0&0&1&0&0&0&-1 \end{bmatrix}$$

(2.32)

En effectuant cette factorisation, on déduit qu'un seul *butterfly* Radix-8 utilise le même nombre d'addition complexes que 12 *butterfly* Radix-2 [WID97].

À partir de T_8 et W_8, on obtient la structure simplifiée du *butterfly* Radix-8, représentée dans la figure 2.10 ci-dessous.

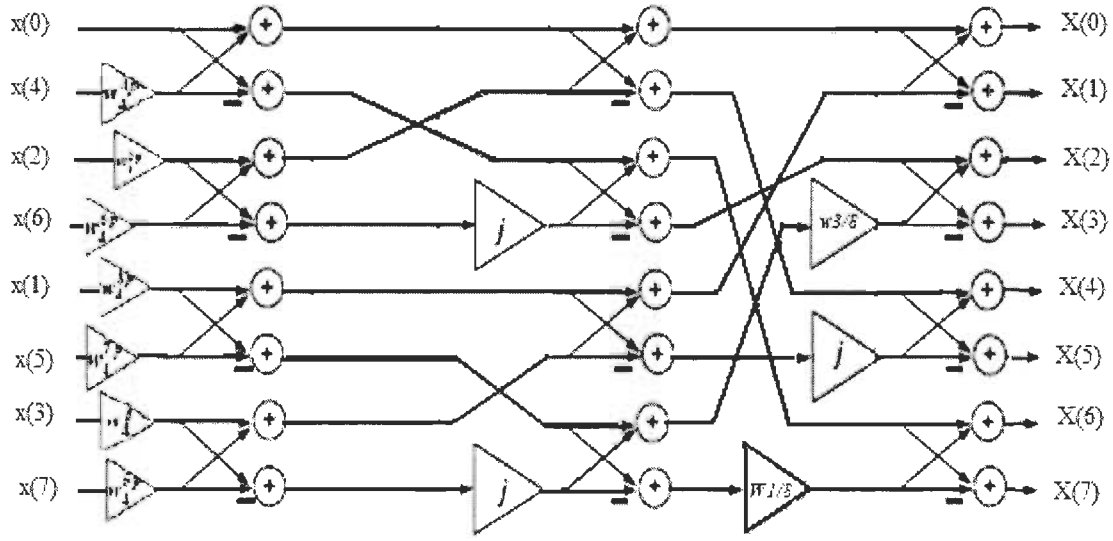


Figure 2.10 : Structure *Butterfly* du Radix-8 DIT

La figure 2.10 représente le SFG (*Signal Flow Graph*) du PE Radix-8 DIT. Tous les signaux d'entrées et de sorties du SFG sont de valeurs complexes. Puisque c'est un algorithme DIT, on constate que les twidles factor sont placés aux PIN d'entrées du PE.

Les entrées du PE ne sont pas dans l'ordre pour des considérations de dessin. On constate aussi les multiplications complexes internes considérées comme non-triviales. Ces dernières rendent le chemin critique du *butterfly* plus complexe. Malgré cela, le Radix-8 reste très prometteur puisque il utilise moins d'opérations et moins de d'accès mémoire.

2.4.2 Split-Radix FFT (SRFFT)

L'algorithme Split-Radix FFT (SRFFT) a été clairement décrit et nommé pour la première fois en 1984 par Duhamel and Hollman [DUH84]. Cet algorithme utilise moins d'opérations arithmétiques (multiplications et d'additions) que n'importe quel autre algorithme de puissance 2.

L'algorithme SRFFT est une variante de l'algorithme Cooley-Tukey, puisqu'il utilise un mélange du Radix-2 et du Radix-4. Il exprime de manière récursive une DFT de longueur N, en fonction de d'une DFT plus petites de longueur N/2 et deux petites DFT de longueur N/4 [JON06], [PRO96].

L'équation 2.33 démontre la dérivation de l'algorithme Split-Radix :

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1}\left(x(2n)e^{-\frac{j2\pi(2n)k}{N}}\right) + \sum_{n=0}^{\frac{N}{4}-1}\left(x(4n+1)e^{-\frac{j2\pi(4n+1)k}{N}}\right)$$

$$+ \sum_{n=0}^{\frac{N}{4}-1}\left(x(4n+3)e^{-\frac{j2\pi(4n+3)k}{N}}\right)$$

$$= DFT_{N/2}[x(2n)] + W_N^k\, DFT_{\frac{N}{4}}[x(4n+1)] + W_N^{3k}\, DFT_{N/4}[x(4n+3)]$$

(2.33)

La figure 2.11 représente la structure du *butterfly* Split-Radix résultant de l'équation 2.33

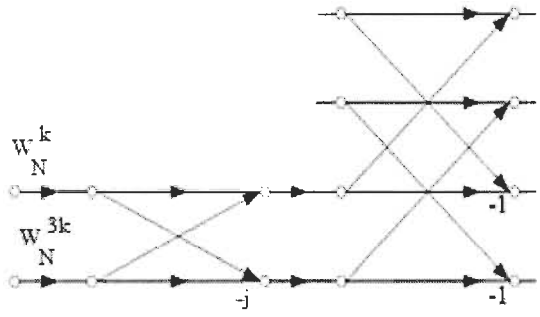


Figure 2.11 : Structure *butterfly* du Split-Radix DIT

Grâce à sa structure mixée, le SRFFT réutilise des *twiddle factor* communs entre le Radix-2 et la Radix-4. Ainsi, le nombre de multiplications non-triviales diminue et des fois remplacé par des additions. En effectuant un mélange approprié des équations régissant le Radix-2 et le Radix-4, on arrive à obtenir un algorithme FFT utilisant moins d'opérations arithmétiques. Le tableau 2.6 donne le nombre multiplications et additions réelles non-triviales pour calculer une DFT de *N*-point complexes.

Tableau 2.6 nombre multiplications et additions réelles non-triviales pour calculer une DFT de N-point complexes [DUH86]

N	Multiplications réelles				Additions réelles			
	Radix2	Radix4	Radix8	Split Radix	Radix2	Radix4	Radix8	Split Radix
16	24	20	-	20	152	148	-	148
32	88	-	-	68	408	-	-	388
64	264	208	204	196	1032	976	972	964
128	712	-	-	516	2504	-	-	2308
256	1800	1392	-	1284	5896	5488	-	5380
512	4360	-	3204	3076	13566	-	12420	12292
1024	10248	7856	-	7172	30728	28336	-	27652

Inconvénients Split-Radix :

- À cause de sa structure irrégulière, le SRFFT est un algorithme très difficile à programmer. Des programmes réussis ont été écris dans [HEI86] mais pour des machines uni-processeur. Il serait difficile de coder le SRFFT pour des vecteurs ou des machines multiprocesseurs.

2.4.3 *Prime Factor Algorithm* et *Winograd Fourier Transform Algorithm*

Le PFA (*Prime Factor Algorithm*) est un algorithme FFT qui redéfinit une DFT de taille $N= N_1N_2$, en une DFT à deux dimensions $N_1 \times N_2$; où N_1 et N_2 sont des nombres premiers entre eux. Pour exemple, on considère une DFT de N points, de séquences d'entrées $x(n)$ et de séquences de sortie $X(k)$, où $x(n)$ et $X(k)$ sont décomposées comme tel : $x(n_1, n_2) = x(N_2n_1+ n_2)$ et $X(k_1, k_2) = X(k_1+N_1k_2)$, respectivement. La DFT peut être calculée par :

$$X(k_1, k_2) = \sum_{n_2} \sum_{n_1} x(n_1, n_2)\, W_{N_1}^{(n_1+\alpha n_2)k}\, W_{N_2}^{n_2(k_2+\beta k_1)} \tag{2.34}$$

Avec α et β des entiers tel que $\alpha\, N_2+\beta\, N_1 = 1$ modulo N_1N_2 [ZHE94].

Le WFTA (*Winograd Fourier Transform Algorithm*) a été présenté pour la première fois en juillet 1975 par S. Winograd de l'*IBM Thomas J.Watson Research Center* [WIN75]. Cet algorithme utilise une forte propriété d'indexage de données basé sur le théorème CRT (*Chineese Remainder Theorem*) ainsi utilise moins de multiplications que le PFA, mais cela en dépit de quelques additions en plus. Le WFTA est plus avantageux que le PFA pour des tailles FFT, plus grandes [WIN75] [BUR08]. Le tableau 2.7 présente une comparaison des nombres d'opérations entre algorithmes FFT à facteur commun et les algorithmes PFA et WFTA [JON07].

Tableau 2.7 : Comparaison de nombres d'opérations entre algorithmes FFT à facteur commun et les algorithmes PFA et WFTA

	Taille FFT	Multiplications réelle	Additions réelles	Multiplications + Additions
Radix2	1024	10248	30728	40976
Split Radix	1024	7172	27652	34824
PFA	1008	5804	29100	34904
WFTA	1008	3548	34416	37964

Le tableau 2.7, confirme que les algorithmes PFA et WFTA demande moins de multiplications réelles que les algorithmes à facteur commun comme le Radix-2 ou le Split

Radix. On constate aussi que pour une même taille de FFT, l'algorithme Split Radix est meilleur puisqu'il utilise moins d'opérations en total que le PFA et le WFTA.

Le WFTA reste un algorithme difficile à programmer et rarement utilisé en pratique. En général pour tout signaux arbitraires, on choisit un nombre d'échantillons de puissance 2 afin d'utiliser; par exemple; le Radix-2 (Algorithme à Facteur commun). Plusieurs travaux de recherches sont effectués sur le PFA et le WFTA afin de les rendre moins complexes et plus simple à programmer [BUR81].

2.5 JFFT

Il a été démontré que la méthode de simplification 'adder tree' n'a pas fourni une solution complète pour le problème de la FFT en raison de l'augmentation de la complexité des butterfly pour des Radix supérieurs, [WID97]. La solution du problème réside dans la structure de la matrice 'adder tree' et la matrice des *twiddle factor*. Ainsi, si nous sommes attentifs aux éléments de la matrice 'adder tree' T_r, ainsi qu'à ceux de la matrice des *twiddle factor* W_N^r, nous remarquons que les deux contiennent des *twiddle factor*. Ainsi, en contrôlant la variation des *twiddle factor* lors du calcul complet de la FFT, nous pouvons fusionner les matrices des *twiddle factor* et 'adder tree' en une seule phase de calcul.

En fonction de l'équation (2.23), B_r est le produit de la matrice des *twiddle factor* W_N par la matrice 'adder tree' T_r. En définit $W_{N(u,v,s)}$ comme l'ensemble de la matrice des *twiddle factor*,

$$W_{N(u,v,s)} = diag(w_{N(0,v,s)},\ w_{N(1,v,s)}, \dots\dots\dots,\ w_{N(r-1,v,s)}) \qquad (2.35)$$

Les indices utilisés sont $u = 0,1,\ldots, r\text{-}1$, $v = 0,1,\ldots, V\text{-}1$ et $s = 0,1,\ldots, S\text{-}1$ où r est le Radix-r, V est le nombre de mot ($V = N/r$), et S est le nombre d'étages ($S = \log_r N$). À partir de l'équation (2.26), on peut définir tout les éléments de la matrice comme ($l^{ième}$ ligne, $m^{ième}$ colonne).

$$[W_N]_{l,m(u,v,s)} = \begin{cases} w_N^{[\![\lfloor v/u^s \rfloor l u^s]\!]_N} & pour\ l=m \\ 0 & ailleurs \end{cases} \tag{2.36}$$

$l = 0,1,\ldots, r\text{-}1$, $m = 0,1,\ldots, r\text{-}1$, et $\lfloor x \rfloor$ représente la partie entière de x.

Ainsi, la structure butterfly Radix-r modifiée et proposée est exprimée pour le traitement DIF comme suit:

$$[B_r]_{l,m(v,s)} = w_N^{\left[\!\left[\frac{lmN}{r} + \lfloor v/r^s \rfloor l r^s\right]\!\right]_N} \tag{2.37}$$

En conséquence, l'opération du Radix-r BPE pour la FFT est formulée par le vecteur colonne :

$$X_{(u,v,s)} = B_r x \tag{2.38}$$

Où la $l^{ième}$ sortie est :

$$X_{(v,s)}[l] = \sum_{m=0}^{r-1} x_{(v,s)}[m]\, w_N^{\left[\!\left[\frac{lmN}{r} + \lfloor v/r^s \rfloor l r^s\right]\!\right]_N} \tag{2.39}$$

Avec le même raisonnement comme ci-dessus, on peut dériver le BPE Radix-r DIT FFT.

Les équations de (2.36) à (2.39) décrivent la fusion des matrices des *twiddle factor* et 'adder tree' ainsi que la gestion des données injectées dans les BPE à travers les étages de la structure pipeline en fonction du nombre de points N de la FFT, le mot et l'étage. La gestion des données utilise un système très performant qui gère l'adressage des données

dans le coprocesseur FFT lors du calcul de la FFT. Ce système est l'*Address Generator*. On rappelle que dans ce mémoire, l'*Address Generator* ne sera pas étudié mais plutôt les performances de vitesse ainsi que les ressources matérielles des BPE JFFT comparés au BPE conventionnels et cela surtout pour les BPE de Radix plus grand comme le Radix-8.

L'idée proposée dans la JFFT qui est la fusion des matrices des *twiddle factor* et 'adder tree' en une seule phase de calcul est expliquée dans ce qui suit. Les équations qui décrivent le fonctionnement du BPE Radix-8 conventionnel de la figure 2.10 sont décris ci-dessous :

$$X_0 = (x_0 + x_4M_4) + (x_2M_2 + x_6M_6) + (x_1M_1 + x_5M_5) + (x_3M_3 + x_7M_7)$$

$$X_1 = (x_0 - x_4M_4) - i(x_2M_2 - x_6M_6) + \alpha(x_1M_1 - x_5M_5) - i\alpha(x_3M_3 - x_7M_7)$$

$$X_2 = (x_0 + x_4M_4) - (x_2M_2 + x_6M_6) - i(x_1M_1 + x_5M_5) + i(x_3M_3 + x_7M_7)$$

$$X_3 = (x_0 - x_4M_4) + i(x_2M_2 - x_6M_6) - i\alpha(x_1M_1 - x_5M_5) + \alpha(x_3M_3 - x_7M_7)$$

$$X_4 = (x_0 + x_4M_4) + (x_2M_2 - x_6M_6) - \alpha(x_1M_1 + x_5M_5) - (x_3M_3 + x_7M_7)$$

$$X_5 = (x_0 - x_4M_4) - i(x_2M_2 - x_6M_6) - \alpha(x_1M_1 - x_5M_5) + i\alpha(x_3M_3 - x_7M_7)$$

$$X_6 = (x_0 + x_4M_4) - (x_2M_2 + x_6M_6) + i(x_1M_1 + x_5M_5) - i(x_3M_3 + x_7M_7)$$

$$X_7 = (x_0 - x_4M_4) + i(x_2M_2 - x_6M_6) + i\alpha(x_1M_1 - x_5M_5) - \alpha(x_3M_3 - x_7M_7) \quad (2.40)$$

Où $x=[x_0, x_1, \ldots x_{(r-1)}]^T$ est le vecteur d'entrée et $X=[X_0, X_1, \ldots X_{(r-1)}]^T$ est le vecteur de sortie. Les M_n où $n = 0 \ldots r\text{-}1$ sont les *twiddle factor* et α un facteur où $\alpha = \exp(-i * \pi/4)$.

Les équations (2.40) représentent les opérations effectuées par le BPE Radix8 conventionnel.

La modification proposée par la JFFT se base sur l'intégration du facteur α dans le calcul de la butterfly à deux entrées contenu dans les équations du BPE Radix-8. Ainsi on fusionne le facteur α avec les *twiddle factor*, là où il existe dans les équations de la butterfly. On prend les termes où α est utilisé :

Par exemple, on a :

$$\alpha(x_1M_1 + x_5M_5) = \alpha M_1x_1 + \alpha M_5x_5 \tag{2.41}$$

Supposons que :

$$\alpha M_1 = M_8 \; et \; \alpha M_5 = M_9 \tag{2.42}$$

Donc (2.41) devient

$$\boldsymbol{\alpha(x_1M_1 + x_5M_5) = \alpha M_1x_1 + \alpha M_5x_5 = x_1M_8 + x_5M_9} \tag{2.43}$$

Cette manipulation de distribution de facteur permet de fusionner le facteur α avec les *twiddle factor* de la butterfly, ainsi on regroupe tout les *twiddle factor* à l'entrée du BPE. Toutefois, dans le cas du BPE Radix-8, on ajoute quatre nouveau *twiddle factor* donc 4 multiplications complexes en plus que le BPE Radix-8 conventionnel.

Les équations (2.44) ci-dessous, représentent la nouvelle formulation modifiée du BPE JFFT8.

$$X_0 = (x_0 + x_4M_6) + (x_2M_3 + x_6M_9) + (x_1M_1 + x_5M_7) + (x_3M_4 + x_7M_{10})$$

$$X_1 = (x_0 - x_4M_6) - i\,(x_2M_3 - x_6M_9) + (x_1M_2 - x_5M_8) - i\,(x_3M_5 - x_7M_{11})$$

$$X_2 = (x_0 + x_4M_6) - (x_2M_3 + x_6M_9) - i\,(x_1M_1 + x_5M_7) + i\,(x_3M_4 + x_7M_{10})$$

$$X_3 = (x_0 - x_4M_6) + i\,(x_2M_3 - x_6M_9) - i\,(x_1M_2 - x_5M_8) + (x_3M_5 - x_7M_{11})$$

$$X_4 = (x_0 + x_4M_6) + (x_2M_3 + x_6M_9) - (x_1M_1 + x_5M_7) - (x_3M_4 + x_7M_{10})$$

$$X_5 = (x_0 - x_4M_6) - i\,(x_2M_3 - x_6M_9) - (x_1M_2 - x_5M_8) + i\,(x_3M_5 - x_7M_{11})$$

$$X_6 = (x_0 + x_4M_6) - (x_2M_3 + x_6M_9) + i\,(x_1M_1 + x_5M_7) - i\,(x_3M_4 + x_7M_{10})$$

$$X_7 = (x_0 - x_4M_6) + i\,(x_2M_3 - x_6M_9) + i\,(x_1M_2 - x_5M_8) - (x_3M_5 - x_7M_{11})$$

(2.44)

Les équations (2.35) représentent le fonctionnement du BPE DIT JFFT8. À partir de ces dernières, on obtient le SFG du BPE JFFT8 représenté dans la figure 2.12.

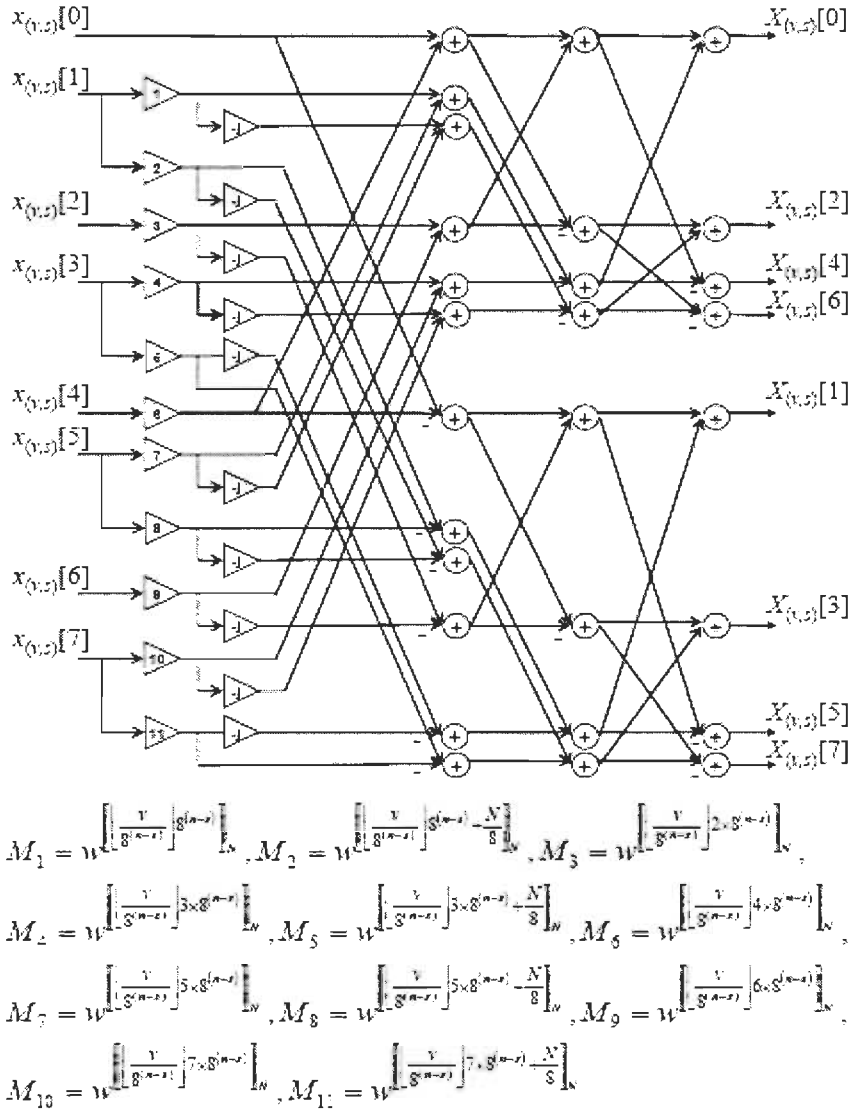


Figure 2.12 Structure butterfly du BPE DIT JFFT8

À partir de la figure 2.12, on remarque dans la structure du BPE JFFT que la multiplication par les *twiddle factor* est effectuée au début de la structure en une seule phase, contrairement au BPE Radix-8 conventionnel. On constate aussi que le BPE JFFT8 contient 4 *twiddle factor* de plus que le BPE Radix-8 conventionnel. Ceci est dû au fusionnement

des *twiddle factor* des matrices 'adder tree' et *twiddle factor* comme expliqué ci-dessus. Les formules pour calculer les *twiddle factor* sont aussi données dans la figure 2.12 et on remarque qu'ils dépendent du nombre de point de la FFT, du mot et de l'étage.

La figure 2.13 ci-dessous représente la structure du BPE JFFT4.

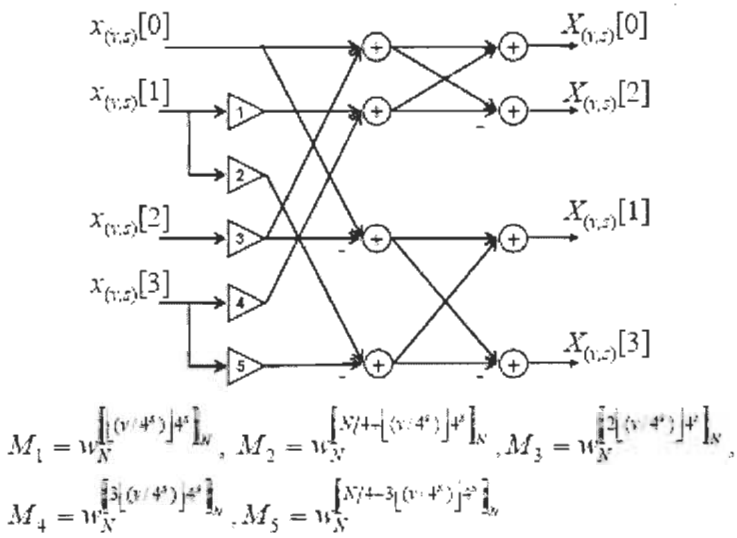


Figure 2.13 Structure butterfly du BPE DIT JFFT4

On remarque que les *twiddle factor* sont aussi regroupés pour être calculés en une seule phase. La différence entre le BPE Radix-4 conventionnel et le BPE JFFT4 n'est pas très grande contrairement au BPE Radix-8 avec le BPE JFFT8. On constate que le BPE JFFT4 utilise deux *twiddle factor* de plus que le BPE Radix-4 conventionnel. Toutefois, le BPE JFFT4 effectue le calcul avec une phase de moins que le BPE Radix-4 conventionnel. Les résultats d'implémentation pour la vitesse et les ressources matérielles seront présentés au chapitre 4.

2.5.1 Comparaison BPE conventionnel et BPE JFFT

La nouvelle modification proposée par la JFFT, permet de diminuer le temps de calcul pour chaque BPE. En microélectronique, le temps de calcul minimal d'un dispositif est représenté par le chemin critique. Le chemin critique est donc le chemin le plus optimal par lequel passe les données afin de donner un premier résultat en sortie du BPE.

Afin de comparer les structures des BPE Radix-r proposées avec les BPE Radix-r conventionnelles, on assume que les données présentes à l'entrée du BPE sont dans le bon ordre. Les figures 2.14 et 2.15 représentent les chemins critiques du BPE Radix-8 conventionnel et JFFT8 respectivement.

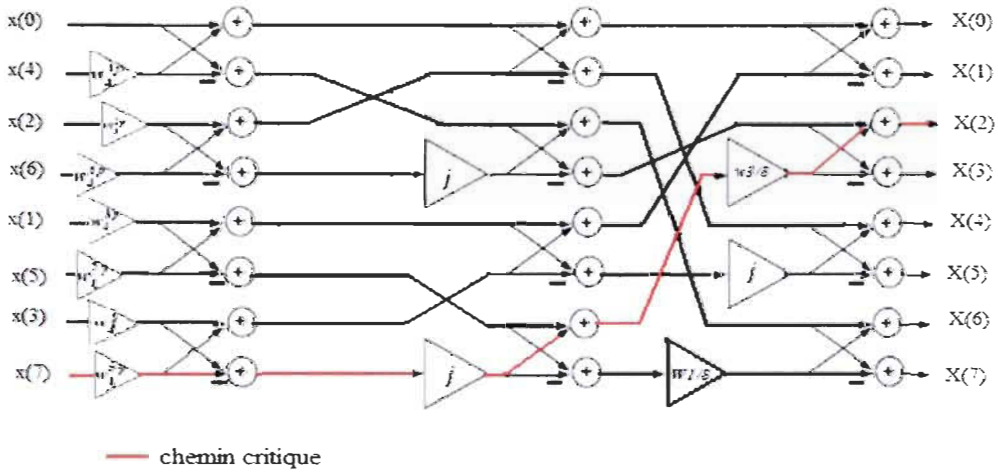


Figure 2.14 chemin critique BPE Radix-8 conventionnel

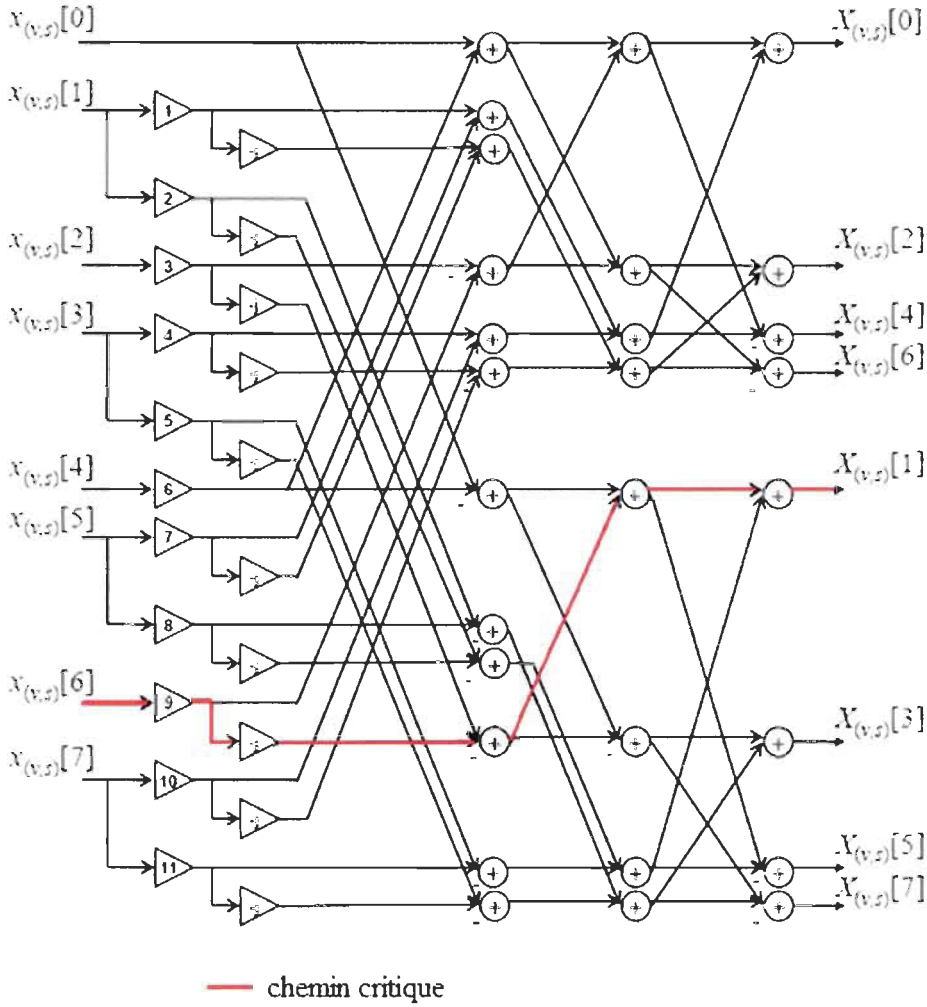


Figure 2.15 chemin critique BPE JFFT-8 proposé

Les figures 2.14 et 2.15 représentent les chemins critiques du BPE Radix-8 conventionnel et JFFT8 respectivement. On constate que le chemin critique du BPE Radix-8 conventionnel a besoin de quatre multiplications de valeur complexes et trois additions de valeurs complexes tandis que le BPE JFFT8 demande une multiplication complexe et trois additions complexes. On remarque une diminution significative des opérations au niveau du chemin critique du Radix-8 JFFT. Pour les autres Radix 2 et 4, le Radix-4 par exemple, le chemin critique du BPE Radix-4 conventionnel a besoin de deux multiplications de valeur

complexes et deux additions complexes tandis que le BPE JFFT4 demande une multiplication complexe et deux additions complexes. Par contre les BPE Radix-2 conventionnel et le BPE Radix-2 JFFT ont une structure identique. Les résultats d'implémentation pour la vitesse et les ressources matérielles seront présentés au chapitre 4.

Chapitre 3 Architectures FFT et effet de quantification

Avec l'évolution des systèmes de télécommunication sans fil, le besoin de vitesse d'exécution en temps réel ainsi que la complexité matérielle ne cessent d'augmenter. Il est donc nécessaire de concevoir des architectures VLSI de coprocesseurs FFT (éléments clé de l'OFDM) qui répondent aux exigences de vitesse (haut débit binaire) et qui sont moins complexe au niveau de l'implémentation matérielle, [YUN03]. Il existe plusieurs architectures dédiées à l'implémentation VLSI des coprocesseurs FFT dans la littérature. Dans ce chapitre on présentera les deux architectures les plus communes : l'architecture parallèle et l'architecture pipeline. On expliquera leurs différentes caractéristiques, avantages et inconvénients et on justifiera le choix pour lequel on a opté qui est l'architecture pipeline. On note que les architectures discutées dans ce chapitre considéreront l'implémentation d'algorithmes FFT à facteur commun (Radix-r et SRFFT ou Mixed-Radix).

3.1 Architecture parallèle

L'architecture parallèle est l'architecture la plus rapide et la plus efficace pour les FFT de courtes tailles *N*. Cette architecture est une reproduction directe du mappage du SFG (*Signal Flow Graph*) de la FFT avec un PE (*Processing Element*), en français, (*Processeur Élémentaire*) dédié pour chaque butterfly du SFG. Cette architecture requière $(N/r).\log_r N$ processeurs élémentaires(PE). On constate donc que son besoin en ressources matérielles augmente rapidement avec la taille *N* de la FFT. C'est pour cette raison que la plupart des implémentations sont limitées à des tailles courtes de *N*, avec *N* pouvant aller jusqu'à 64 points, [WID97].

3.1.1 Exemples d'implémentations d'architectures parallèles

La figure 3.1 représente l'implémentation de l'architecture parallèle du SFG Radix-2 de type DIT (avec *N*=8) de la figure 2.7.a.

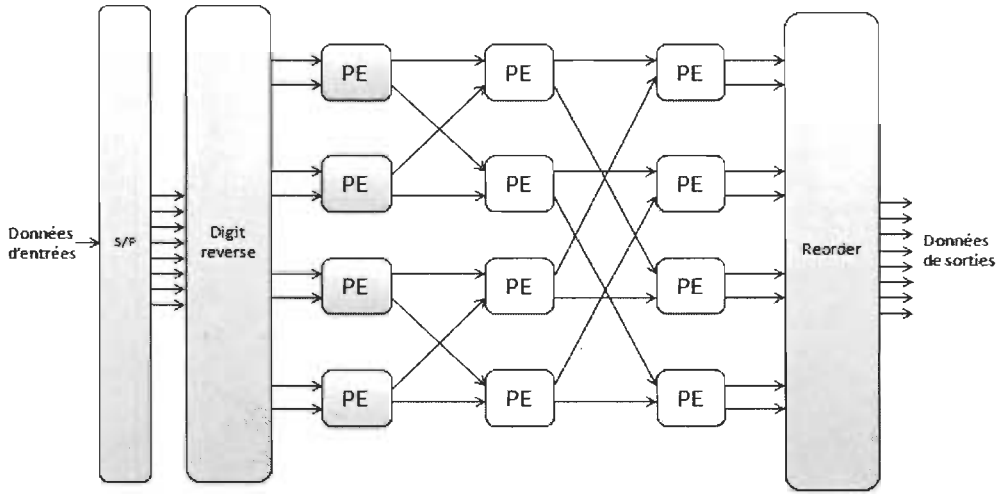


Figure 3.1 architecture parallèle du Radix-2 (N=8)

Un des problèmes de cette architecture, excepté le grand besoin de ressources matérielles, est la faible utilisation des PE puisque les données d'entrée sont séquentielles tandis que les données de sorties sont générées en parallèle. Ce problème peut être résolu si on alimente les entrées des PE par N trames de données parallèles à chaque cycle d'horloge. On verra donc l'utilisation des PE augmentée à 100% (Figure3.1). On constate aussi l'ajout de deux modules 'digit reverse' et 'reorder' qui assure que les données entrent et sortent dans le bon ordre pour l'obtention du bon calcul de la FFT, [WID97]. La figure 3.2 montre un autre exemple de l'architecture parallèle, celui du Radix-4 avec N=16 points.

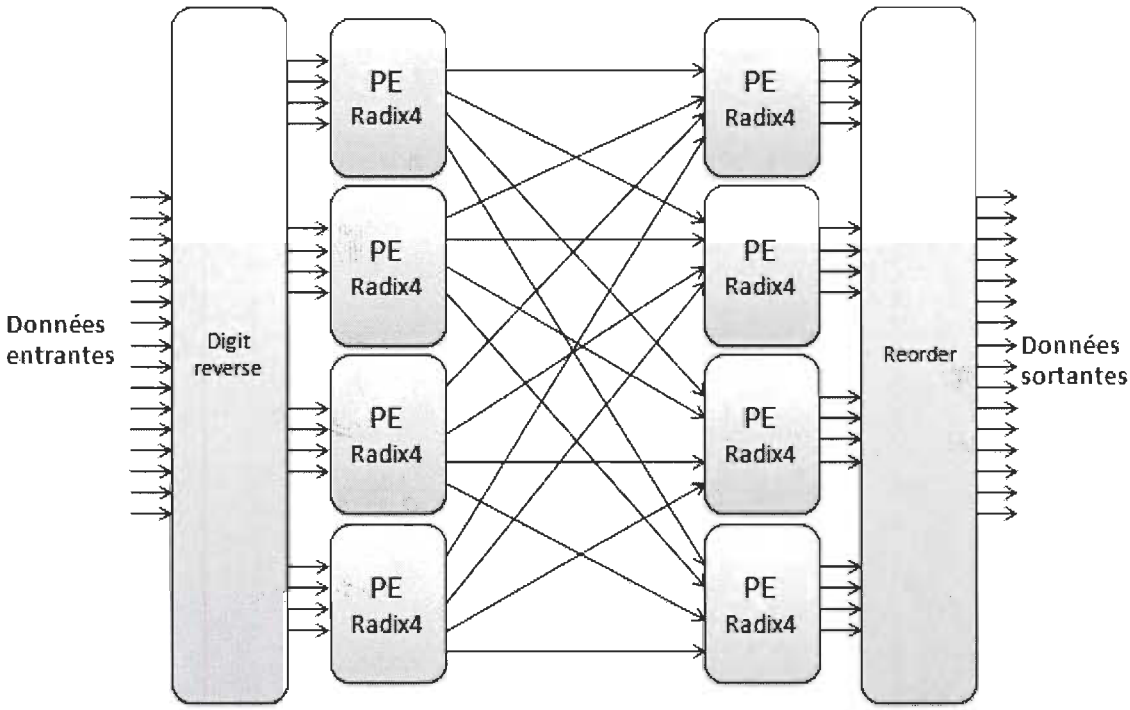


Figure 3.2 Architecture parallèle du Radix-4 (N=16)

Comme cité dans la définition de l'architecture parallèle, les PE (*Processeurs Élémentaires*) représentent les butterfly dans les SFG des algorithmes FFT. D'autres

algorithmes utilisant deux sortes de PE de Radix différents comme le SRFFT (*Split Radix FFT*) peuvent être implémentés en architecture parallèle. La figure 3.3 ci-dessous représente l'architecture parallèle du SRFFT (4/2) pour N=32 points.

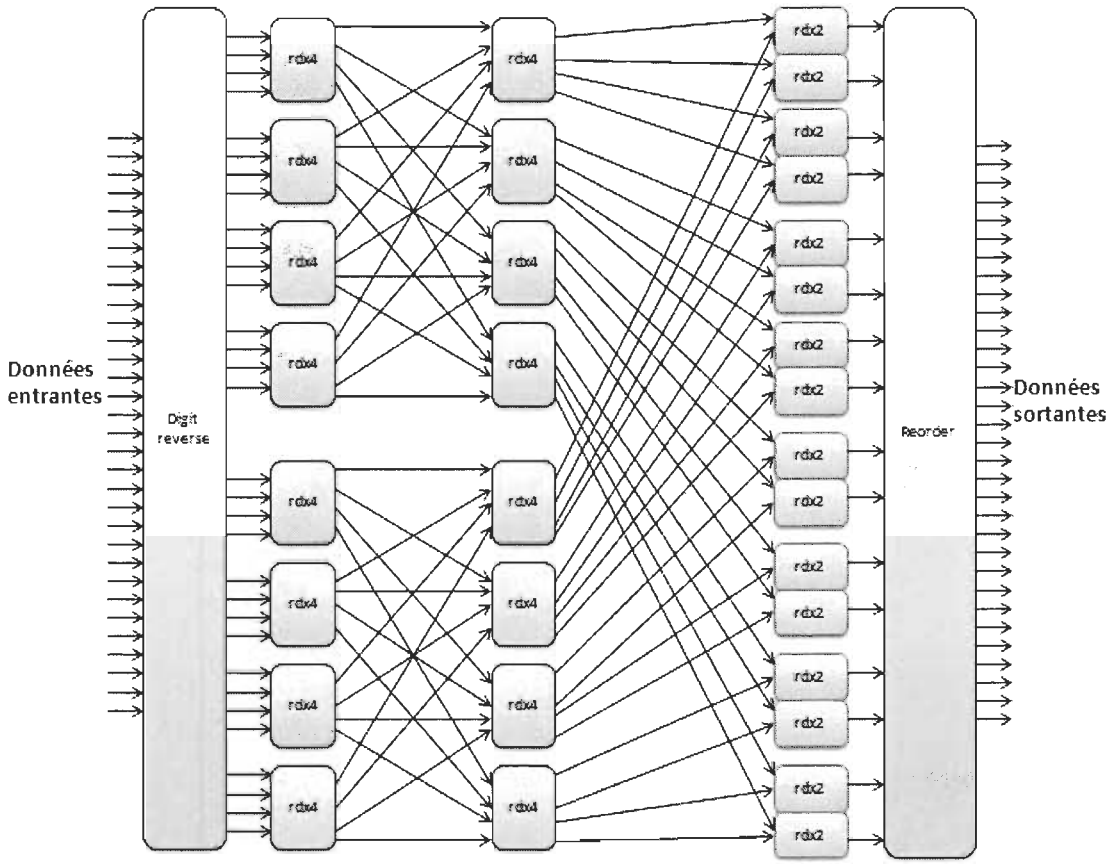


Figure 3.3 Architecture parallèle du SRFFT-4/2 (N=32)

À partir de la figure 3.3, on remarque que plus la taille N augmente, plus l'architecture prend d'espace silicium. Ce problème handicape l'implémentation de l'architecture parallèle et la limite à des petites tailles de FFT. Cependant, elle reste très attrayante, sinon

une des meilleures solutions d'implémentation si la taille de la FFT ne dépasse pas 64 points.

D'autres avantages pour cette architecture peuvent être cités dont :

- L'implémentation efficace sur FPGA (grâce à la structure parallèle Manhattan)
- Utilisation de butterfly optimisés
- Réduction du nombre des *twiddle factor* (réutilisation des *twiddle factor*)
- Aucune limite de pipeline
- Peut atteindre une fréquence d'horloge de 400 MHz sur les nouvelles FPGA Virtex5 de Xilinx.

3.1.2 Discussion

Malgré que l'architecture parallèle nous permette de calculer la FFT de façon efficace et rapide, elle ne pourrait répondre aux exigences des standards des nouveaux systèmes de communications basés sur l'OFDM. Si on se base sur le tableau 1.3.1, les FFT à calculer ont de grandes tailles qui varient entre 256 points à 8096 points. Ceci est dû au grand nombre d'informations numériques contenues dans les signaux OFDM traités par les systèmes OFDM. Par exemple, le système MIMO-OFDM avec k antennes de transmission et de réception se compose de k processeurs OFDM large bande en parallèle, donc k coprocesseurs FFT sont nécessaires. Pour cette raison, la complexité matérielle augmente k fois. Dans notre cas, l'architecture parallèle n'est donc pas le choix idéal à utiliser pour

implémenter la FFT pour ses systèmes puisqu'elle nécessite énormément de ressources matérielles, donc ne donne pas le meilleur compromis vitesse/surface silicium. Heureusement plusieurs autres architectures alternatives sont utilisées pour le calcul de la FFT et respectent le compromis vitesse/surface silicium, une des plus utilisées actuellement est l'architecture pipeline.

3.2 Architecture pipeline

Au cours de ces deux dernières décennies, plusieurs conceptions d'architectures pipeline ont été développées et implémentées. L'architecture pipeline est une des architectures les plus populaires et les plus adaptées pour les applications du traitement de signal où le haut débit de données est une condition dominante, [SWA84]. Depuis, l'architecture pipeline offre un bon compromis complexité matérielle/taux de traitement de données pour les systèmes de communications à haut débit, [SWA84], [YUN03], [SAN07], [HSI09]. La figure 3.4 représente une architecture pipeline de base, [YUN03].

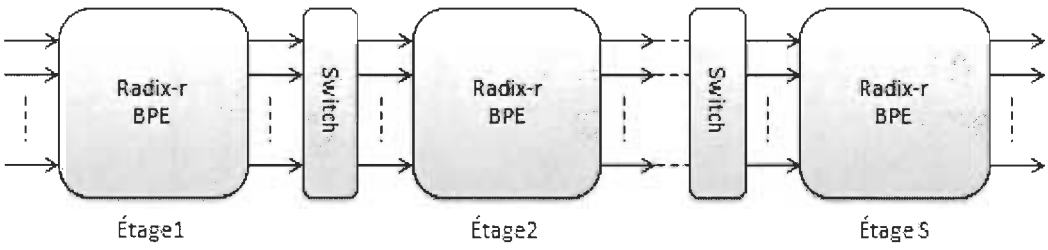


Figure 3.4 architecture pipeline de base

Comme le montre la figure 3.4, l'architecture se compose d'un nombre de BPE (Butterfly Processor Element) qui sont les unités de calcul des butterfly et d'un nombre de modules 'Switch'. Ces derniers réorganisent les données entre les BPE et peuvent contenir des modules de délais. À chaque étage du pipeline, un BPE, prend r données de valeurs complexes, effectue son calcul, puis génère r données intermédiaires à la sortie. On a donc un débit de données r fois la fréquence d'horloge. Par exemple avec un BPE Radix-2 fonctionnant à R MHz, les données sont traitées à une fréquence de $2R$ MHz, puisque deux données sont traitées au cours de chaque période d'horloge. De même pour un Radix-4 fonctionnant à R MHz, la fréquence de traitement sera de $4R$ MHz. Donc plus le Radix du BPE est grand, plus on obtient un meilleur débit binaire, [YUN03].

Il existe plusieurs structures d'architecture pipeline, on peut les divisés en trois types : l'architecture SDF (*Single-path Delay Feedback*), l'architecture SDC (*Single-path Delay Commutator*) et l'architecture MDC (*Multi-path Delay Commutator*), [YUN03].

3.2.1 Architecture Radix-r SDF

L'architecture SDF (*Single-path Delay Feedback*) est l'architecture la plus appropriée pour les systèmes OFDM au niveau de la complexité matérielle. Puisqu'il existe plusieurs algorithme Radix-r, où r est le nombre d'entrées du BPE, on retrouve plusieurs architectures SDF, tel que la R2SDF (*Radix-2 Single-path Delay Feedback*), la R4SDF, la R8SDF et la R2^3SDF. L'architecture SDF est considérée comme la meilleure approche afin de calculer la FFT pour un système OFDM à un seul canal parce qu'elle requière le moins

de ressources matérielles et qu'elle possède une seule entrée et sortie ce qui la rend très appropriée pour un système SISO-OFDM (*Single Input Single Output Orthogonal Frequency Division Multiplex*), [HSI09], [SAN07].

La figure 3.5.a et 3.5.b ci-dessous, représente deux architectures SDF, la R2SDF et la R4SDF.

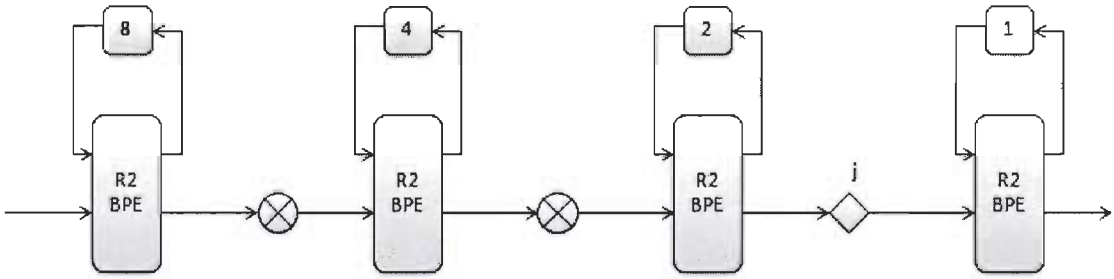


Figure 3.5.a Architecture R2SDF (N = 16 points)

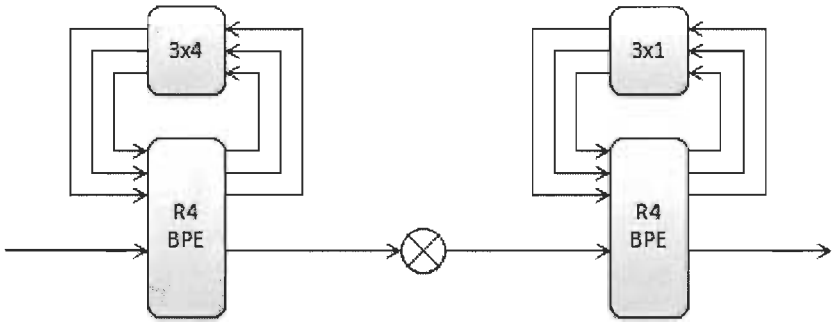


Figure 3.5.b Architecture R4SDF (N = 16 points)

Les figures 3.5.a et 3.5.b représentent les architectures R2SDF et R4SDF respectivement pour N =16 points. On remarque que l'architecture SDF utilise les registres de manières plus efficaces en stockant les sorties des BPE dans les registres à décalage de la chaîne à rétroaction. Dans l'architecture SDF, la séquence de données d'entrée passe par le chemin

d'accès unique puis est réorganisée par l'unité de calcul BPE. À cause du chemin unique d'accès, un seul multiplieur complexe est utilisé après chaque BPE. La complexité matérielle est donc faible. Toutefois, l'architecture fonctionne à une faible vitesse et le contrôle de synchronisation des données demeure complexe, [YUN03].

Puisque l'architecture SDF possède un seul chemin d'accès d'entrée et de sortie et a un faible débit binaire, plusieurs coprocesseurs FFT R4SDF doivent être utilisés pour un système MIMO-OFDM. Par exemple, pour un système MIMO-OFDM à 4 canaux, quatre architectures R4SDF doivent être utilisées (une pour chaque canal). On constate que la complexité matérielle augmente avec le nombre de canaux du système MIMO-OFDM, [HSI09].

3.2.2 Architecture Radix-r SDC

L'architecture pipeline Radix-r SDC utilise le même nombre de multiplieurs complexes que l'architecture pipeline SDF grâce à son chemin unique d'accès. Elle utilise par contre un commutateur de délais qui possède une seule entrée et plusieurs sorties. Le nombre de sorties du commutateur varie selon le nombre d'entrée du BPE. La figure 3.6 représente une architecture pipeline R4SDC.

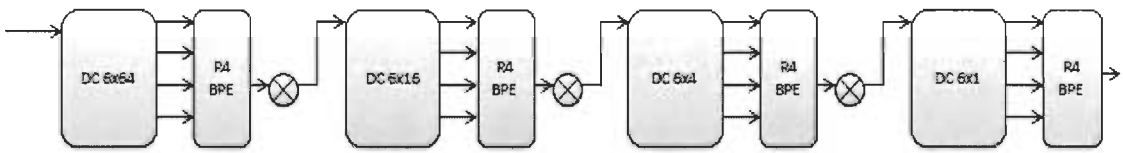


Figure 3.6 Architecture R4SDC ($N = 256$ points)

À chaque coup d'horloge, le commutateur de délais injecte les données au BPE. Après avoir effectué les calculs, le BPE injecte le résultat dans le chemin unique d'accès pour l'étage suivant. L'architecture SDC requière un système de contrôle plus complexe que les autres architectures pipeline. Et comme l'architecture SDF, elle opère à un faible débit binaire puisqu'elle possède un seul chemin d'accès d'entrée et de sortie. Pour un système MIMO-OFDM, elle a aussi donc besoin de plusieurs coprocesseurs pour chaque canal du système. Ceci augmente la complexité matérielle de son implémentation, [YUN03], [HSI09].

3.2.3 Architecture Radix-r MDC

L'architecture pipeline Radix-r MDC (*Multi-path Delay Commutator*) est l'architecture pipeline la plus populaire et la plus utilisée pour l'implémentation des systèmes de communications qui demandent un haut débit binaire [SAN07]. Plusieurs implémentations existent. On trouve le R2MDC (*Radix-2 Multi-path Delay Commutator*), le R4MDC, le R8MDC et d'autres architectures plus élaborées comme le R2^3MDC et le MC-MRMDC (*Multi Channel - Mixed Radix Multi-path Delay Commutator*).

L'architecture MDC est l'architecture la plus adaptée aux systèmes MIMO-OFDM parce qu'elle possède multiple chemin d'entrées et de sorties. Cette propriété permet aux coprocesseurs FFT utilisant cette architecture d'opérer à haute vitesse avec un haut débit. Par contre cette architecture utilise plus de surface silicium que les autres architectures

pipeline, mais reste très attrayante et reste l'implémentation pipeline la plus classique grâce à son compromis vitesse/surface silicium.

Les figures 3.7.a et 3.7.b représentent deux architectures pipeline R2MDC et R4MDC.

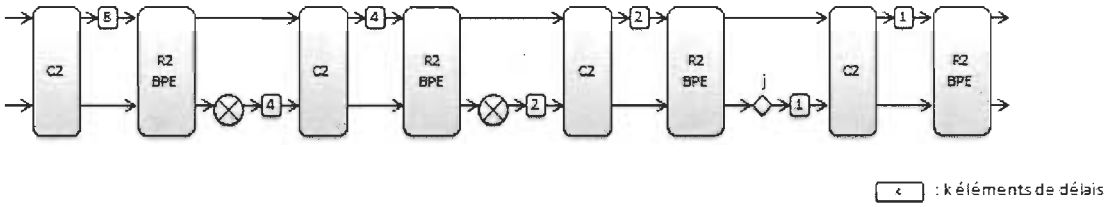


Figure 3.7.a Architecture R2MDC (N = 16 points)

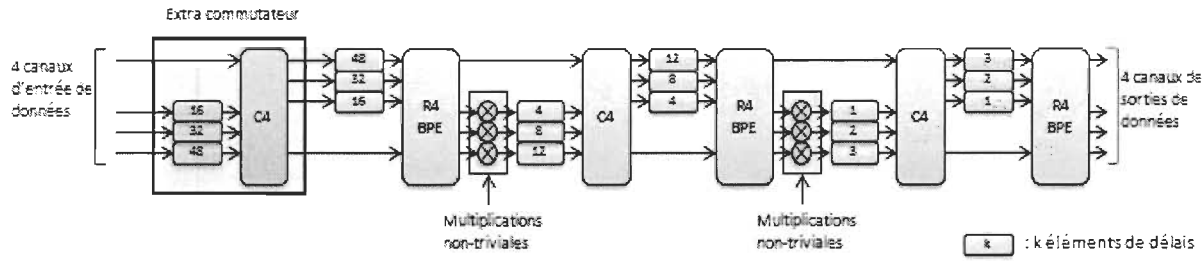


Figure 3.7.b Architecture R4MDC (N = 64 points)

Les figures 3.7.a et 3.7.b représentent respectivement deux architectures pipeline R2MDC pour N=16 points et R4MDC à 4 canaux pour N=64 points. Contrairement à l'architecture SDF, l'architecture MDC de la figure 3.7.b par exemple, utilise multiple chemins d'entrées et multiple chemins de sorties. Le nombre de ces derniers dépend du nombre d'entrées du BPE. Par conséquent, un coprocesseur FFT R4MDC à 4-canaux peut traiter simultanément

quatre canaux de données indépendants, de plus, le système de contrôle des données inter-étage est très simple à gérer. On précise que l'architecture R4MDC à 4 canaux de la figure 3.7.b à la différence de l'architecture conventionnelle R4MDC, requière un extra commutateur. Ce dernier n'augmente pas la complexité matérielle de l'architecture puisqu'il est implémenté par de simples éléments de délais et un mixeur (*multiplexer*). Le rôle de cet extra commutateur est de réorganiser les quatre flux d'entrées de données avant de les injecter dans l'architecture. La figure 3.8 représente le diagramme du commutateur de délais utilisé dans l'architecture proposée par [YUN03].

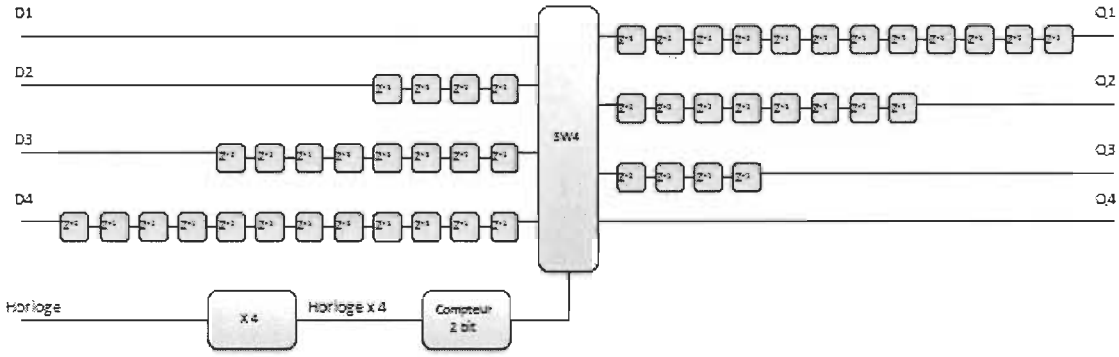


Figure 3.8 diagramme du commutateur de délais

La complexité matérielle du coprocesseur FFT RPMDC, avec *P* la valeur du Radix, peut être réduite en utilisant d'autres algorithmes comme le Mixed-Radix plus communément le Split-Radix, [SWA84] [SAN07].

La figure 3.9 représente l'architecture MC-MRMDC d'un coprocesseur FFT utilisé pour un système de communication MIMO-OFDM.

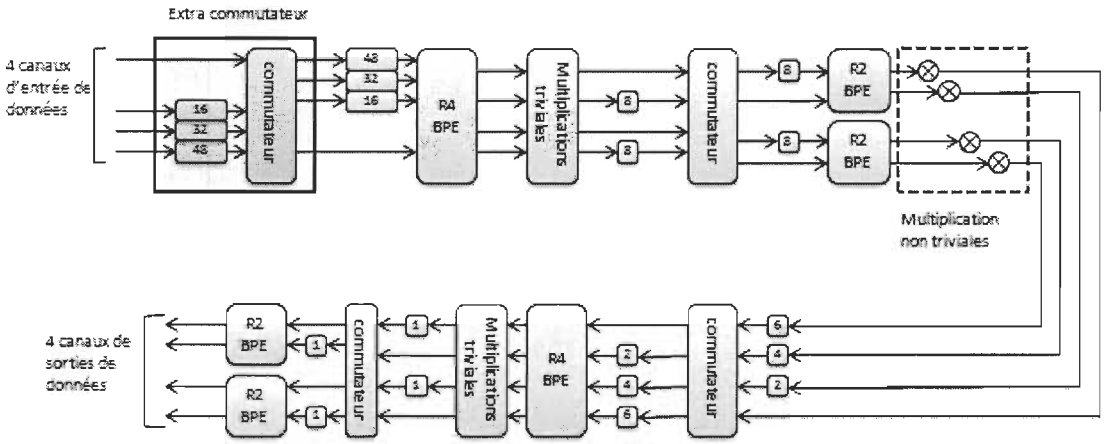


Figure 3.9 Architecture MC-MRMDC ($k = 4$, $N = 64$ points)

La figure 3.9 représente l'architecture MC-MRMDC (*Multi Channel - Mixed Radix Multi-path Delay Commutator*) présentée dans [SAN07] pour $N = 64$ points et $k = 4$, où k est le nombre de canaux MIMO. Cette architecture plus efficace, utilise l'algorithme Mixed-Radix est permet de diminuer le nombre de multiplications non-triviales donc diminue la surface silicium utilisée. On constate à partir des figures 3.7.b et 3.8 que le coprocesseur FFT MC-R4MDC utilise 6 multiplications non-triviales tandis que le coprocesseur FFT MC-MRMDC utilise que 4 multiplications non-triviales, [SAN07].

Le tableau 3.1 expose une comparaison des exigences matérielles de plusieurs architectures FFT pipelines y compris le $R2^2SDF$, le $R2^3SDF$ et le MRMDC. La constante T désigne le nombre des additionneurs requis pour implémenter une multiplication triviale et $[a]$ représente le plus petit entier supérieur à a.

Tableau 3.1 Comparaison en termes d'exigences matérielles entre différentes architectures FFT pipelines pour k MIMO canaux, [SAN07].

	Nombre de Processeurs (P)	Multiplieurs complexes/P	Additionneurs complexes/P	Taille mémoire/P	Contrôle
R2SDF	k	$\log_2 N$ - 1	$2\log_2 N$	$N-1$	simple
R4SDF	k	$\log_4 N$ - 1	$8\log_4 N$	$N-1$	moyen
R4SDC	k	$\log_4 N$ - 1	$3\log_4 N$	$2N-2$	complexe
R2^{2}SDF	k	$\log_2 N$ - 1	$4\log_4 N$	$N-1$	simple
R2^{3}SDF	k	$2(\log_8 N-1)$	$(62T)\log_8 N$	$N-1$	moyen
R2MDC	$[k/2]$	$\log_2 N$ - 2	$2\log_2 N$	$3N/2-2$	simple
R4MDC	$[k/4]$	$3(\log_4 N-1)$	$8\log_4 N$	$5N/2-4$	simple
MRMDC	$[k/4]$	$4(\log_8 N-1)$	$(12+3T)\log_8 N$	$5N/2-4$	simple

D'après le tableau 3.1, pour $k = 4$ et $N = 64$, l'architecture MC-MRMDC sauve 2 multiplications non-triviales comparativement à l'architecture MC-MR4MDC et 76 additionneurs complexes comparativement à l'architecture MC-R2^{3}SDF.

On constate qu'en utilisant l'architecture pipeline MDC, on utilise moins de processeurs, ce qui réduit significativement la surface silicium utilisée. Par exemple; pour implémenter un système MIMO-OFDM à 4 canaux; on aura besoin de 4 processeurs utilisant l'architecture pipeline SDF au lieu d'un seul processeur utilisant l'architecture MDC. On réalise ainsi un grand gain en termes de surface. Ceci justifie l'utilisation de l'architecture pipeline MDC dans la plupart des implémentations VLSI des coprocesseurs FFT. Cependant, on remarque que la taille mémoire par BPE est plus élevé pour les méthodes MDC comparativement au SDF, [SAN07].

3.3 Effet de quantification

3.3.1 Architecture MDC utilisée

Après l'étude des différentes architectures pipelines, il a été démontré que l'architecture MDC (*Multi-path Delay Commutator*) est la plus appropriée pour les nouveaux systèmes de télécommunication LTE de $4^{ième}$ génération basés sur la MIMO-OFDM. L'architecture MDC offre le meilleur compromis vitesse/surface silicium, donc permet d'obtenir des hauts débits binaires avec une utilisation de surface matérielle très bien prise en charge par les nouvelles familles FPGA de Xilinx.

L'architecture utilisée dans notre projet est une architecture semblable à celle de la figure (3.4) par contre le contrôle des données et des délais ainsi que le flux de données inter-étages s'effectue grâce à l'unité de contrôle '*Address Generator*' déjà cités au chapitre 2.

Une FFT de taille $N = r^S$ est programmée sur Matlab avec S étages où chaque étage exécute un BPE Radix-r (*Butterfly Processing Element*), avec r le nombre d'entrée du BPE.

Par exemple pour $N = 4096$ si $r = 2$, le nombre d'étages est S = 12. Par contre pour $r = 8$, le nombre d'étage $S = 4$.

3.3.2 Méthode et outils de vérification

Nous étudierons l'effet de quantification sur deux architectures de BPE, ainsi les architectures étudiées et comparées sont : R2MDC, R8MDC, R16MDC, J8MDC et J16MDC.

Généralement dans un algorithme de traitement de signal, chaque opération de multiplication peut introduire une erreur due à l'opération d'arrondi ou à la troncation. Cette erreur est appelée erreur de quantification arithmétique. En plus les coefficients '*twiddle factors*' sont utilisés avec une longueur binaire limitée, cependant on constate une perte due aux valeurs inexactes des coefficients '*twiddle factors*', [CHA08]. Cette dernière est appelée erreur de quantification des coefficients. Trois cas seront étudiés, programmés et simulés pour calculer le SQNR (*Signal-to-Quantization-Noise Ratio*) dans les mêmes conditions que l'article de référence [CHA08] mais pour des tailles de FFT de 512 et 4096 points:

Cas1: On fera varier la longueur binaire des coefficients (*twiddle factors*) de 4 à 16 bits et on fixera les autres variables (tailles des données, tailles registres internes…) à 16 bits.

Cas2: On fixera la taille des coefficients à 8 bits et on variera les autres variables de 8 à 24 bits.

Cas3: On fera varier cette fois; et la longueur binaire des coefficients et la longueur binaires des autres variables; entre 4 et 24 bits.

La figure 3.10 représente la méthode de comparaison du SQNR selon la référence, [CHA08].

Dans tous ces cas de simulation, nous avons ajusté le nombre de bit d'entier qui maximise le SQNR. En effet, le nombre d'entier varie en fonction de la longueur *N* de la FFT. Le nombre de bit entier est le même pour FFT conventionnelle et JFFT. Pour des longueurs N de 512 et 4096 points le nombre de bits entiers considéré est de 1 bit entier pour les *twiddle factors* et 2 bits entiers pour les données d'entrées r et variables internes. Nous avons observé que l'amplitude du signal d'entrée faisait en sorte que 2 bits entiers optimisait le SQNR pour 512 et 4096 points ainsi que pour la FFT et la JFFT. De plus, une mise à l'échelle de r à la sortie de chaque étage n'apporte pas plus de précision.

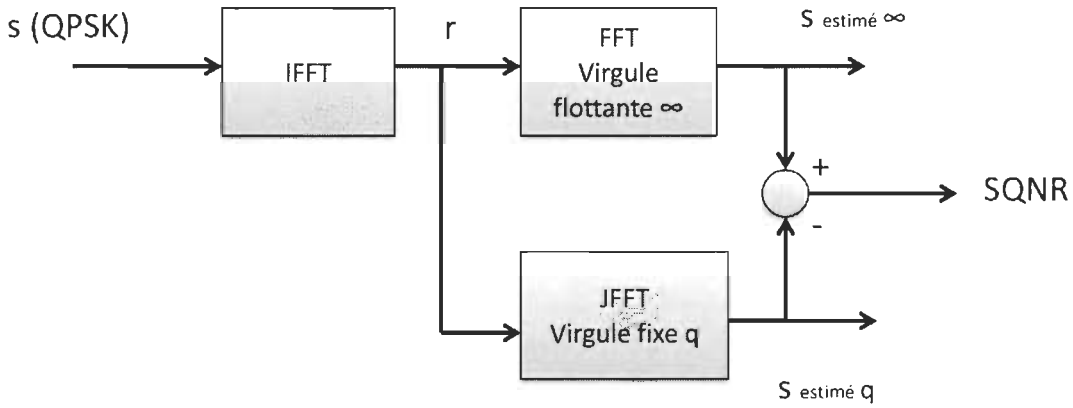


Figure 3.10 Méthode de calcul du SQNR, [CHA08]

La figure 3.10 traduit la méthode utilisée pour calculer le SQNR. Le signal *s* est le signal envoyé en QPSK, $s \in \{\pm 1 \pm j\}$, le signal *r* est le signal reçu. Le signal *s estimé* ∞ avec l'opérateur mathématique ∞ qui signifie dans notre cas en virgule flottante et le signal *s estimé* q avec q qui fait référence à la virgule fixe. La différence entre ces deux derniers signaux nous donne le SQNR.

3.3.3 Résultats de quantification

Selon notre évaluation préliminaire, en télécommunications, un SQNR supérieur à 30 dB semble garantir une bonne qualité de communications, soit un faible niveau de taux d'erreur à 10 dB. C'est pourquoi, nous nous intéresseront aux longueurs binaires offrant des performances autour d'un SQNR de 30 dB.

Après simulation des architectures R2MDC, R2MDC, R8MDC, J8MDC et J16MDC, nous avons obtenu les graphes représentant le SQNR en fonction des coefficients '*twiddle factors*' et des autres variables.

Les figures 3.11 et 3.12 représentent les résultats de quantification issus du premier cas.

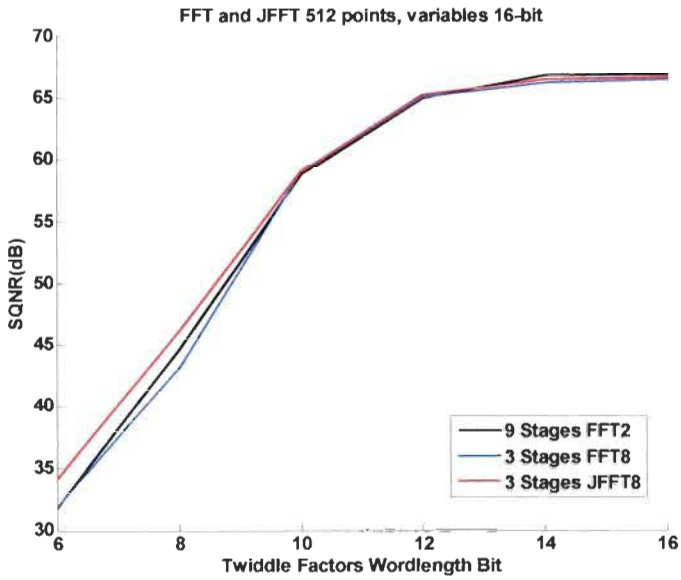


Figure 3.11 Comparaison du SQNR obtenu par la FFT2, FFT8 et JFFT8 pour N=512 points (cas1)

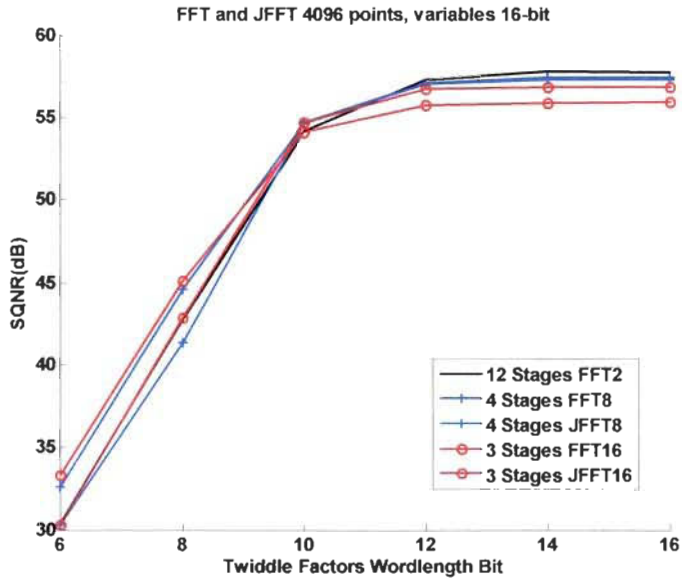


Figure 3.12 Comparaison du SQNR obtenu par la FFT2, FFT8, FFT16, JFFT8 et JFFT16 pour N=4096 points (cas1)

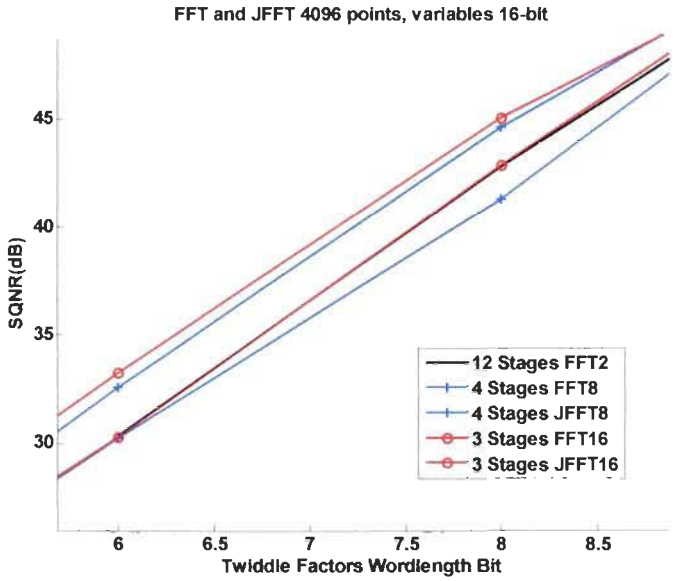


Figure 3.13 Zoom sur la figure 3.12 sur le SQNR obtenu par la FFT2, FFT8, FFT16, JFFT8 et JFFT16 pour N=4096 points (cas1)

Selon les graphes des figures 3.11, 3.12 et 3.13 on constate que les architectures JFFT8 et JFFT16 ont un gain de 3 dB par rapport aux architectures FFT8 et FFT16 conventionnelles pour une longueur binaire de *twiddle factors* de seulement 6 bits et une longueur binaires fixe de 16 bits pour les autres variables.

NB : le graphe de la figure 3.13 est un zoom de la figure 3.12.

Les figures 3.14 et 3.15 représentent les résultats de quantification du deuxième cas.

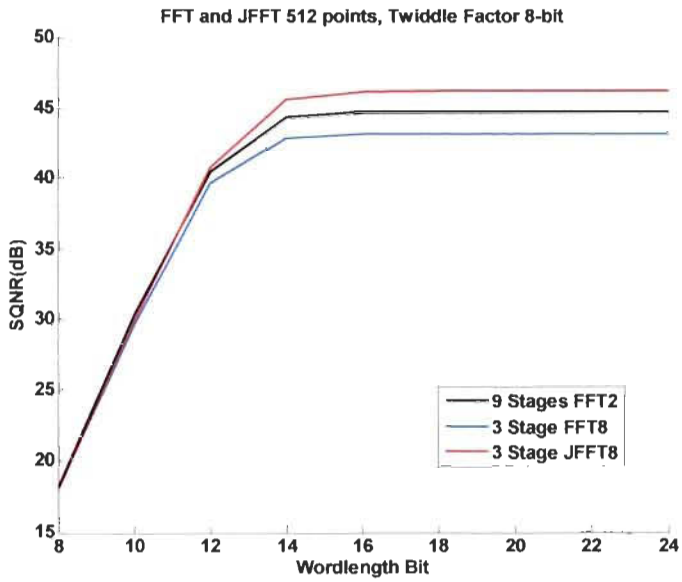


Figure 3.14 Comparaison du SQNR obtenu par la FFT2, FFT8 et JFFT8 pour N=512points (cas2)

À partir du graphe de la figure 3.14 pour $N = 512$ points, on constate que la méthode JFFT8 fourni un gain de 3 dB par rapport à la FFT2 à partir d'une longueur binaire 14 bits pour les tailles des données et les autres variables avec une longueur binaire fixe de 8 bits pour les *twiddle factors*.

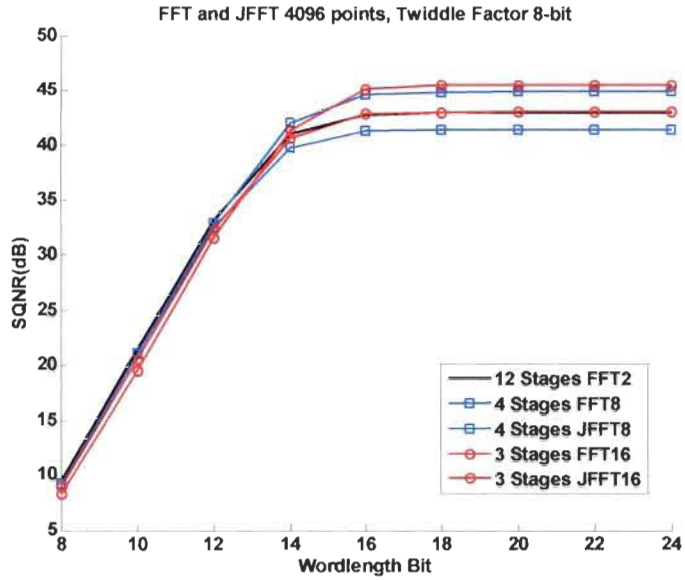


Figure 3.15 Comparaison du SQNR obtenu par la FFT2, FFT8, FFT16, JFFT8 et JFFT16 pour N=4096 points (cas2)

Selon le graphe de la figure 3.15 pour N = 4096 points, on constate que les architectures JFFT8 et JFFT16 commence à être plus intéressante avec un gain de 3 dB par rapport aux architectures FFT8 et FFT16 conventionnelles à partir d'une longueur binaire 16 bits pour les tailles des données et les autres variables avec une longueur binaire fixe de 8 bits pour les *twiddle factors*. Il s'agit d'un gain intéressant pour la version DIT de la FFT [CHA08]. On constate aussi qu'à partir de la longueur binaire 16 bits, le gain devient fixe malgré l'augmentation de la longueur binaire des autres variables.

Les figures 3.16 et 3.17 représentent les résultats de quantification issus du troisième cas.

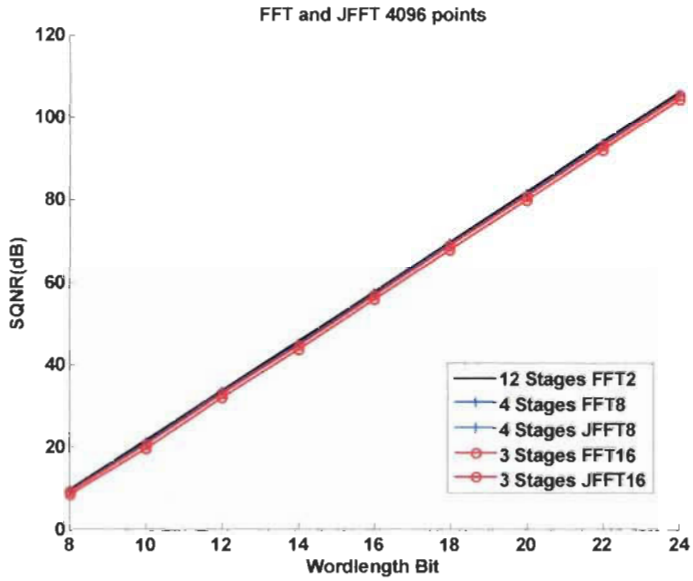


Figure 3.16 Comparaison du SQNR obtenu par la FFT2, FFT8, FFT16, JFFT8 et JFFT16 pour N=4096 points (cas3)

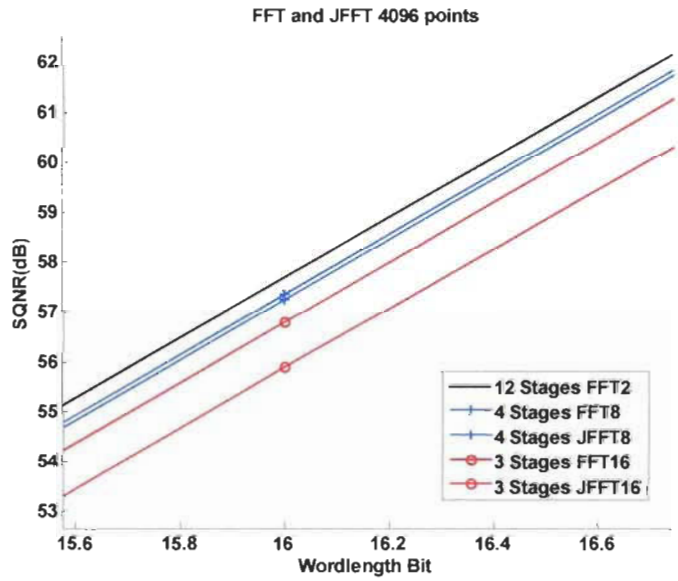


Figure 3.17 Zoom sur la figure 3.16 sur le SQNR obtenu par la FFT2, FFT8, FFT16, JFFT8 et JFFT16 pour N=4096 points (cas3)

Selon les graphes des figures 3.16 et 3.17, on constate le gain que réalise la JFFT8 et JFFT16 par rapport aux architectures FFT8 et FFT16 conventionnelles. On constate aussi que plus la longueur des variables (coefficients, données...) est grande plus le SQNR augmente. Contrairement aux cas 1 et cas 2 où l'on remarque une saturation à partir de la longueur binaire 16 bits.

Chapitre 4 Implémentation de la JFFT

Dans ce chapitre, nous étudierons l'implémentation des BPE (*Butterfly Processing Element*) éléments clé de la FFT. Nous exposerons en détail les étapes qui nous en permis de concevoir leurs architectures (BPE conventionnels et JFFT). Nous testerons le bon fonctionnement de ces dernières à l'aide d'une cosimulation Matlab-Modelsim. Par la suite, nous exposerons une étude théorique du chemin critique des architectures en termes de temps de calcul et nous évaluerons théoriquement les ressources matérielles nécessaires pour les implémenter. La partie d'implémentation (synthèse) viendra confirmer l'efficacité réelle de la nouvelle formulation «JFFT» en termes de temps de calcul et des ressources matériels d'implémentation en technologie FPGA (*Field Programmable Gate Array*).

Ce chapitre représente aussi les résultats d'implémentation obtenu des différentes architectures de BPE (Butterfly Processing Element). Les résultats sont exprimés dans plusieurs tableaux selon la famille de FPGA. Nos résultats sont comparés avec les résultats obtenus par une nouvelle étude présentée par [ZHO09].

Les tableaux fournissent un résumé précis des résultats obtenus selon une taille de FFT choisie de 4096 points, une taille de données et de TWF fixée à 16 bit.

4.1 Principes et éléments sensibles

4.1.1 Principes

La programmation de la FFT effectuée sur Matlab®, pour l'étude de l'effet de quantification, nous a permis de valider le bon fonctionnement des BPE. On a ainsi bien définit les opérateurs et les éléments sensibles qui rentrent dans la conception des architectures des BPE. Pour la conception de ces dernières, on utilisera un langage de description de circuits électroniques : le 'VHDL'. Le VHDL appelé aussi VHSIC (*Very High Speed Integrated Circuit*), est un langage communément utilisé pour décrire le fonctionnement des structures électroniques, de concevoir des circuits logiques programmable (synthèse) et de les simuler pour valider leur fonctionnement, [DOU02]. En VHDL, tout composant (dans le sens logiciel) est décrit sous deux aspects :

- L'interface avec le monde extérieur, décrite dans une section dénommée ***entity*** **(entité)**.
- L'implémentation elle-même, décrite dans une section dénommée **architecture**.

C'est donc l'architecture qui contient la fonction matérielle qu'on souhaite implémenter, [DOU02].

Le logiciel utilisé pour le projet est Modelsim®. La plate-forme Matlab® servira quant à elle, à vérifier et valider le bon fonctionnement des entités 'architectures des BPE' créées sur Modelsim®. Une fois que les architectures sont validées et fonctionnelles, on passe à l'étape de synthèse qui est effectuée avec l'outil Xilinx XST®. Les FPGA ciblées pour implémentation sont la famille Spartan-3, la famille Virtex-E, la famille Virtex-4 et la

famille Virtex-5. Cette dernière étape nous permettra d'évaluer les performances de la nouvelle formulation FFT, «la JFFT», en termes de temps de calcul et de ressources matériels. Le tableau 4.1 résume les outils utilisés pour les différentes étapes de conception des BPE.

Tableau 4.1 outils de conception

Étape de conception	Outils
Simulation VHDL	Modelsim PE 6.4
Synthèse VHDL	Xilinx XST ISE 9.1
Cible FPGA	Famille Spartan-3, Famille Virtex-4, Famille Virtex-E et Famille Virtex-5
Vérification	Matlab R2007b

Dans les parties qui suivent on construira les entités qui composent les coprocesseurs élémentaires. Il est important de savoir que les données qui seront traitées par nos coprocesseurs BPE seront comprises entre -1 et 1 (normalisation). Nous devrons donc utiliser la représentation suivante (pour 16 bits) :

Bit 15 = signe (0 pour positif et 1 pour négatif)

Bit 14 = 2^{-1}, Bit 13 = 2^{-2}, Bit 12 = 2^{-3}, Bit 11 = 2^{-4}, Bit 10 = 2^{-5}, Bit 9 = 2^{-6}, Bit 8 = 2^{-7}, Bit 7 = 2^{-8}, Bit 6 = 2^{-9}, Bit 5 = 2^{-10}, Bit 4 = 2^{-11}, Bit 3 = 2^{-12}, Bit 2 = 2^{-13}, Bit 1 = 2^{-14},

Bit 0 = 2^{-15}.

Les nombres négatifs sont représentés en notation en complément à 2.

4.1.2 Éléments sensibles

D'après les équations des BPE, on peut remarquer que les opérations de base pour calculer la FFT sont la multiplication complexe et l'addition ou soustraction complexe. De plus,

dans la littérature, les performances des différents algorithmes FFT sont souvent comparés par rapport au nombre d'opérations de multiplications et d'additions complexes, [DUH90], [WID97], [JIA04], [LIM05], [FAN06], [ZHO09]. Dans ce projet, les BPE de différents Radix utiliseront plusieurs entités : registres, multiplieurs, additionneurs et soustracteurs etc…

Lors de la programmation de ces entités, on a fait en sorte que tous, prennent en compte le débordement appelé aussi l'«overflow». Vous trouverez leurs programmes dans les références [ACH10]. Ils ont ainsi été simulés et testés sur Modelsim dans un premier temps, puis valider par Matlab à l'aide d'une cosimulation Modelsim-Matlab. La figure 4.1 représente le diagramme de méthode de cosimulation Modelsim-Matlab utilisée dans le projet.

4.1.2.1 Utilisation de l'architecture pipeline

Le choix de l'architecture pour implémenter la FFT est un élément très sensible et très important. Dans le chapitre 3, on a justifié notre choix pour l'architecture pipeline. Donc, comme dit précédemment, l'architecture pipeline est une des architectures les plus populaires et les plus adaptées pour les applications du traitement de signal où le haut débit de données est une condition dominante, [SWA84]. Dans notre projet, on étudie et utilise particulièrement l'architecture pipeline MDC puisqu'elle propose un faible temps de latence, une forte utilisation et exploitation des BPE et surtout requière une faible utilisation de mémoire comparée à plusieurs autres architectures pipeline. La figure 3.4 représentée dans le chapitre 3, montre la structure de l'architecture pipeline utilisée lors de notre projet.

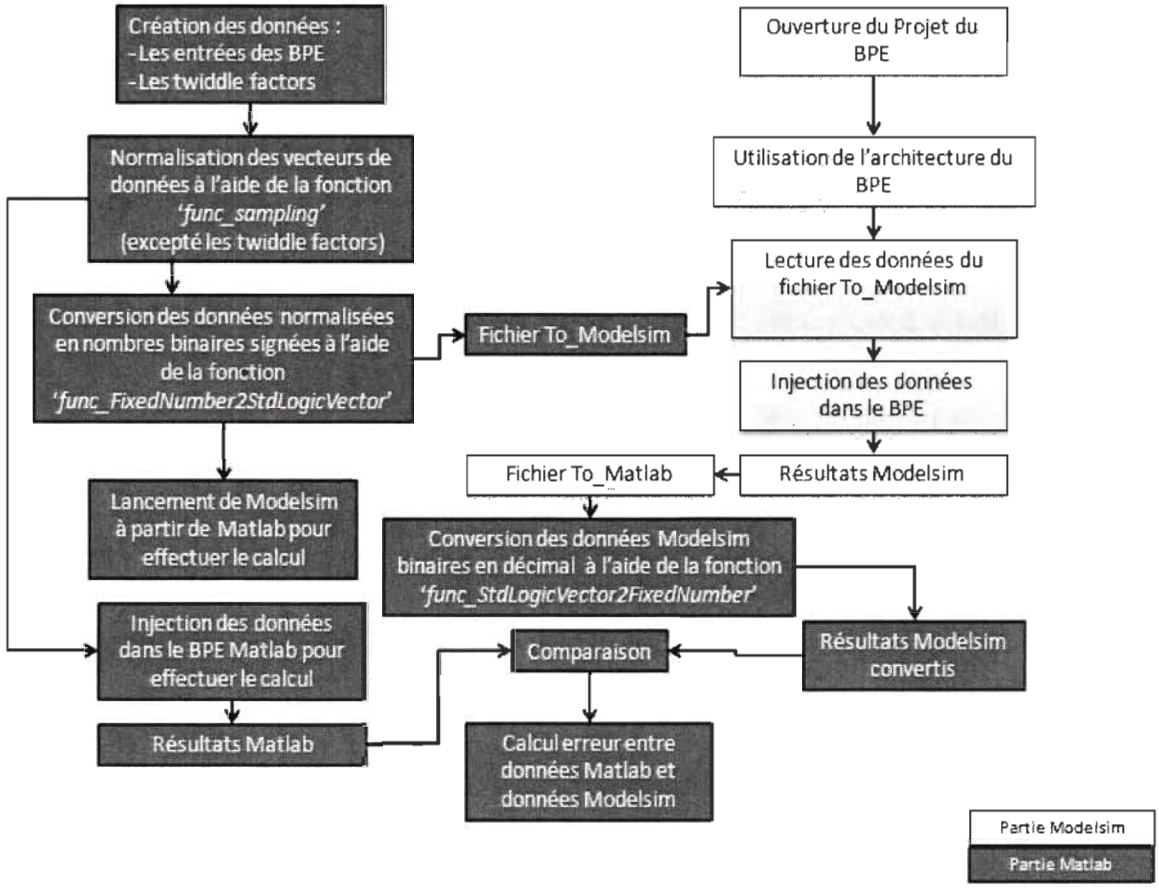


Figure 4.1 Diagramme de la méthode de cosimulation Matlab-Modelsim

Cette architecture répond très bien aux exigences des normes et standards de l'ITU en termes de latence, vitesse de calcul, utilisation de mémoire surtout pour un système de transmission et réception MIMO-OFDM [SAN07]. Les auteurs de l'article [SAN07] démontrent que pour implémenter un système MIMO-OFDM à 4 canaux; on aura besoin de 4 processeurs utilisant l'architecture pipeline SDF au lieu d'un seul processeur utilisant l'architecture MDC. On réalise ainsi un grand gain en termes de surface. Ceci justifie l'utilisation de l'architecture pipeline MDC dans la plupart des implémentations VLSI des coprocesseurs FFT.

4.1.2.2 Entité multiplieur complexe

On retrouve le multiplieur de nombres complexes dans pratiquement tous les algorithmes de traitement de signal tel que les filtres adaptatifs (LMS, Kalman…), la FFT, algorithmes d'égalisation des canaux etc., [JIA04], [LIM05], [FAN06]. Le multiplieur complexe est considéré comme un des éléments les plus sensibles dans l'architecture des BPE puisqu'il utilise une surface silicium importante dans l'architecture complète des différents BPE. Et puisque les systèmes de communication numérique actuels sont de plus en plus exigeants : beaucoup de données à traiter et le plus rapidement possible, ce module se doit d'être plus performant, très rapide et surtout doit consommer le moins de surface silicium, [JIA04], [LIM05]. La multiplication de deux nombres complexes peut être représentée de la façon suivante :

$$(A + jB) * (C + jD) = (AC - BD) + j(AD + BC) = R_e + j I_{mag} \qquad (4.1)$$

Avec : Re = AC-BD, partie réelle du produit

Imag = AD+BC, partie imaginaire du produit

Le multiplieur de nombres complexes fait intervenir quatre multiplications réelles (AC, BD, AD, BC), une addition et une soustraction. Les nombre A, B, C et D sont considérés des nombres binaires en complément à deux, signés (premier bit = bit de signe). Une autre manière de décrire une multiplication de nombres complexes est la suivante :

$$\begin{cases} R_e = AC - BD = (A - B) * D + A * (C - D) \\ I_{mag} = AD + BC = (A - B) * D + B * (C + D) \end{cases} \qquad (4.2)$$

Avec cette méthode de factorisation, On passe de quatre multiplications réelles, une addition et une soustraction à trois multiplications, trois additions et deux soustractions. En

sachant que la surface en circuit électronique qu'occupe un multiplieur est trois fois supérieur à celle d'un additionneur, [JAB09]. Cette méthode de factorisation, est très bénéfique puisqu'elle rend le multiplieur complexe plus rapide en prenant moins d'espaces sur le circuit électrique.

La figure 4.2 décrit le schéma du multiplieur complexe selon la méthode de factorisation.

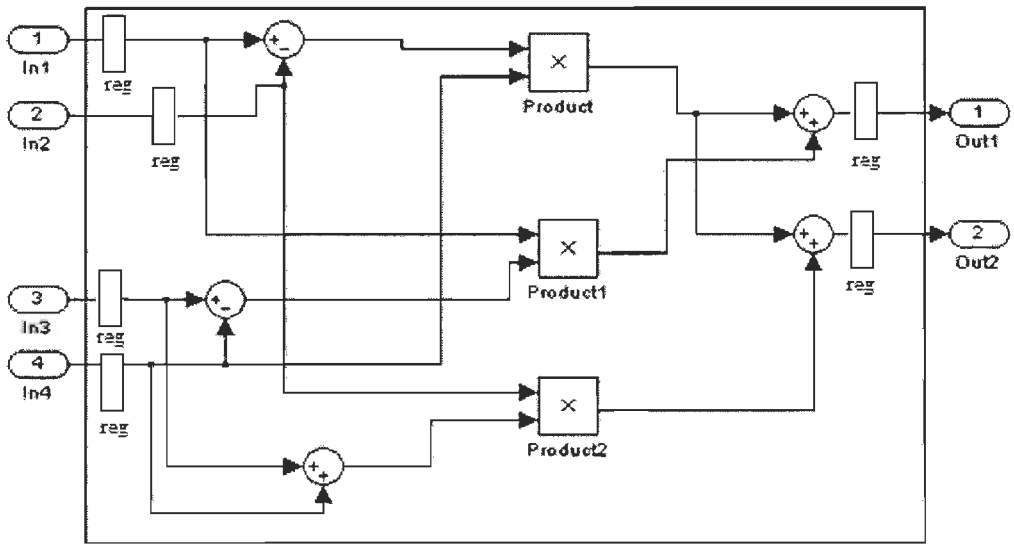


Figure 4.2.a Schéma de multiplieur complexe cas0

La figure 4.2.a représente le schéma du multiplieur complexe qui sera utilisé dans la conception des BPE. Admettant que les entrées sont codés sur Nbits=16 bits. Au niveau de la multiplication dans le module (multiplieur complexe), la taille des données est multipliée fois 2 (x2) soit Nbits=32 bits. Donc les additionneurs qui suivront cette opération, verront la taille de leurs entrées doubler soit Nbits =32 bits.

Pour les modules qui viendront après l'entité multiplieur complexe, on insère une entité troncation qui va rendre les données à leur taille première qui est de Nbits=16 bits.

L'avantage de la troncation à ce niveau est de réduire la taille des additionneurs donc on aura des additionneurs plus petits, donc moins de surface utilisée sur le circuit microélectronique. Cependant, de cette opération résultera un problème qui est «le bruit de troncation » puisqu'on a une perte de données : suppression des 16 bit les moins significatifs.

Pour l'étude de l'implémentation, on ajoute des registres internes pour effectuer des étages pipeline à l'intérieur du multiplieur complexe. Le pipeline augmente considérablement la fréquence du module. Une latence due aux registres vient s'ajouter. Par contre la cadence pour avoir des résultats à la sortie du module BPE est ainsi augmentée.

Pour notre étude on considère quatre cas de multiplieur complexe, le cas0 est celui où le multiplieur complexe n'est pas pipeliné, figure 4.2.a. Le cas1 est celui où on ajoute des registres avant chaque additionneur réel, soustracteur réel et multiplieur réel du module multiplieur complexe, que nous appellerons dans la suite pipeline complet, figure 4.2.b. Le cas2 est celui où on ajoute des registres après les multiplieurs réels, figure 4.2.c. Et finalement dans le cas3, on vient ajouter des registres avant les multiplieurs réels, figures 4.2.d. Les figures 4.2.b, c et d, ci-dessous représentent les cas1, 2 et 3 du multiplieur complexe.

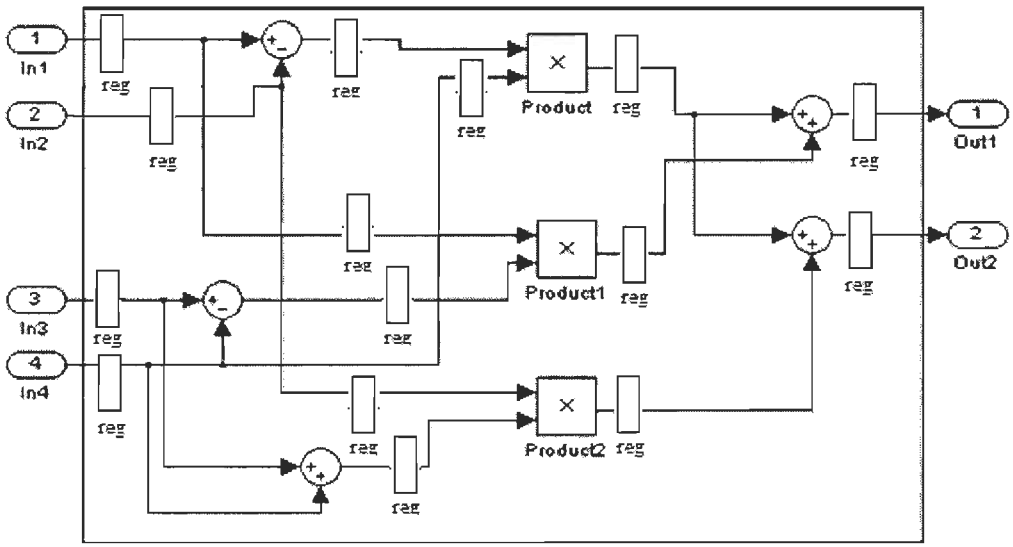


Figure 4.2.b Schéma de multiplieur complexe cas1

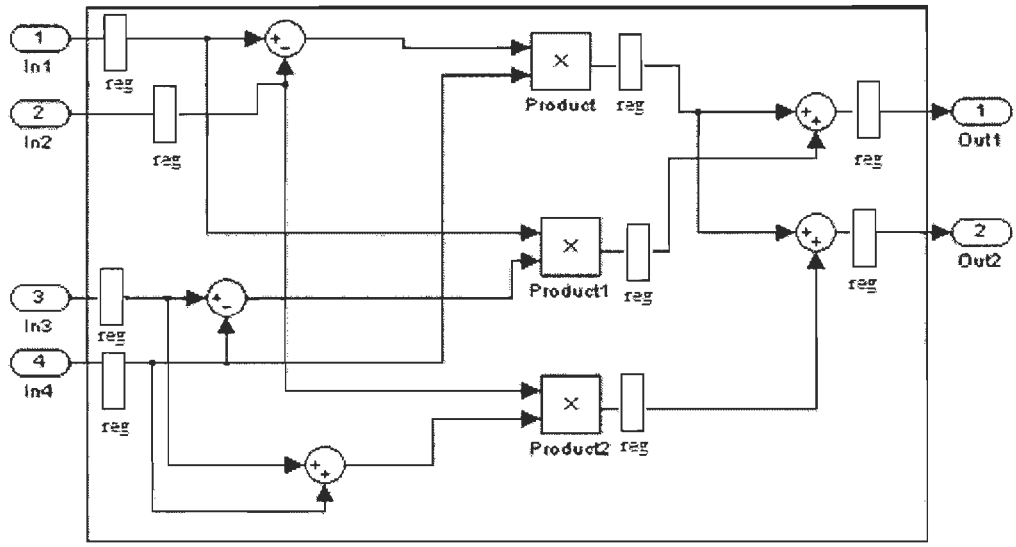


Figure 4.2.c Schéma de multiplieur complexe cas2

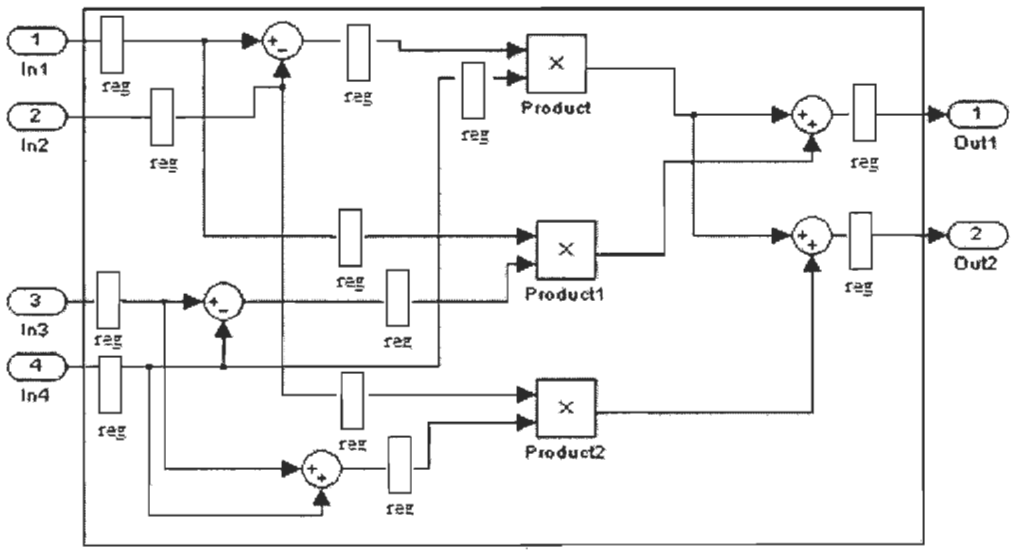


Figure 4.2.d Schéma de multiplieur complexe cas3

4.2 Conception des architectures

Dans cette partie nous verrons les étapes de conception des BPE conventionnels et JFFT. Toutes les figures représentant les BPE utilisent le multiplieur complexe du cas1, 'pipeline complet'.

4.2.1 Butterfly conventionnel

4.2.1.1 Radix-2

L'architecture de l'entité butterfly Radix-2 conventionnel est représentée dans la figure 2.5 où A et B sont les entrées du butterfly et W_N appelé coefficient de la FFT ou encore «*twiddle factor*». Le butterfly de la figure 2.5 calcule la FFT de deux points. Cette structure butterfly représente une partie clé pour l'accomplissement des autres architectures des BPE du projet. Il est aussi nécessaire de mentionner que c'est une butterfly de l'algorithme

Radix-2 DIT et que durant le projet toutes les architectures seront de type DIT. Le schéma de l'architecture de l'entité Radix-2 DIT est représenté par la figure 4.3 ci-dessous :

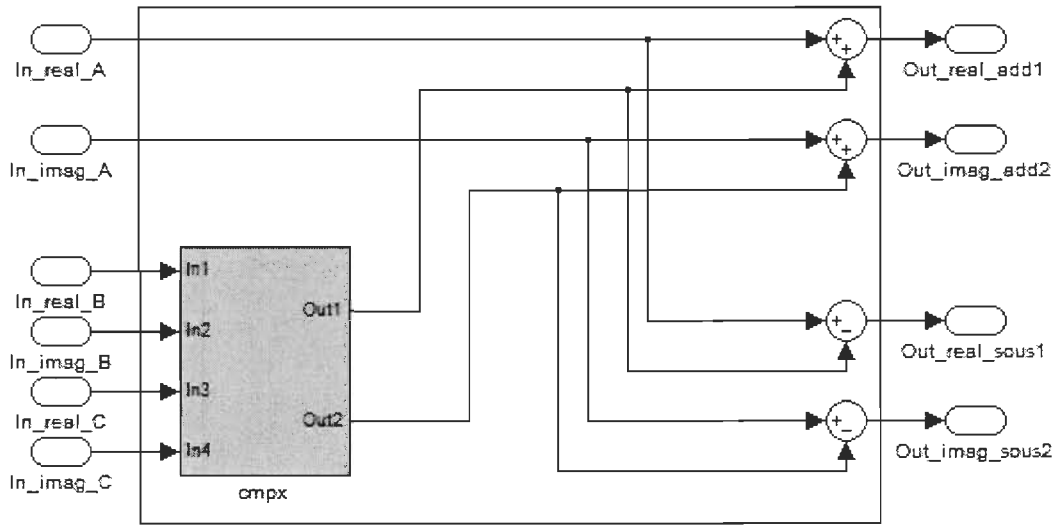


Figure 4.3 Entité Radix-2 DIT

Cette entité se compose de l'entité du multiplieur complexe, de deux entités addition et de deux entités soustraction. On note que les entrées sont traitées en réels et imaginaires séparément. Le programme VHDL de cette entité est disponible voir [ACH10].

Dans ce projet, on notera aussi que les BPE Radix-2 conventionnel et JFFT sont identiques.

4.2.1.2 Radix-4

Le SFG du BPE Radix-4 conventionnel est représentée dans la figure 2.9 (chapitre 2). La figure 4.4 ci-dessous, représente les différentes entités qui vont constituer le BPE Radix-4.

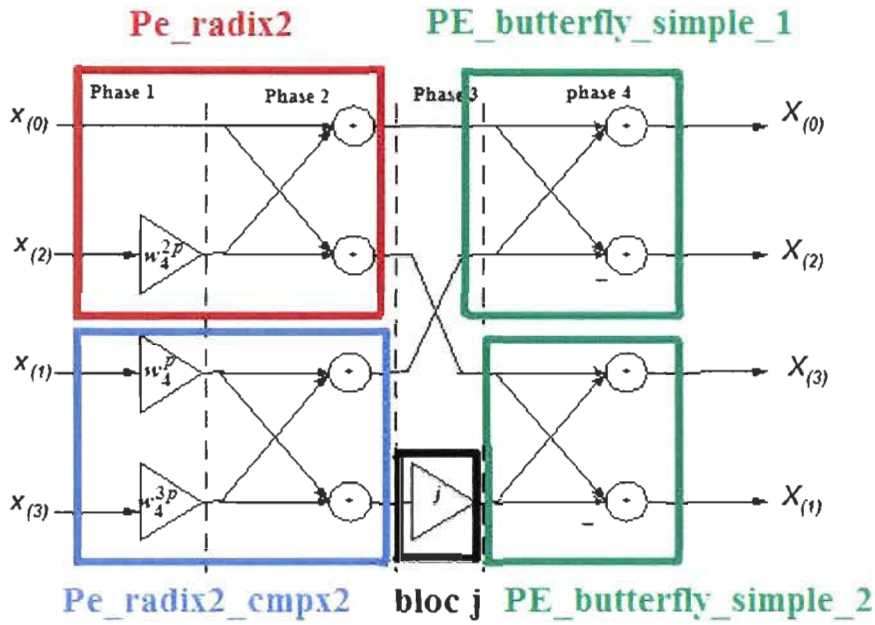


Figure 4.4 Entités constituants le BPE Radix-4

La figure 4.4 représente le SFG du BPE Radix-4 et décrit les trois principales entités qui constituent le BPE Radix-4 : l'entité Pe_Radix2, l'entité Pe_Radix2_cmpx2 et l'entité Pe_butterfly_simple. Ces entités seront programmées sur VHDL est constitueront l'architecture du BPE Radix-4. On remarque que le calcul du butterfly se fait en 4 phases. Dans la phase 1, les entrées du BPE Radix-4 sont multipliées par les *twiddle factor*. La phase 2, calcule les butterfly simples. Dans la phase 3, la deuxième sortie de la butterfly 2 est multipliée par le «bloc j». Finalement dans la phase 4, les données sont injectées dans les butterfly simples. On remarque qu'on peut regrouper les phases 1 et 2 en créant l'entité Pe_Radix2 et l'entité Pe_Radix2_cmpx2 (figure 4.3). Les données intermédiaires sorties par ces dernières seront injectées à leur tour dans les entités Pe_butterfly_simple. Celles sorties de l'entité Pe_Radix2_cmpx2 passera par le «bloc *j*» avant d'être injectées dans

l'entité Pe_butterfly_simple_2. L'entité Pe_Radix2 est présentée ci-dessus par la figure 4.3. Les entités Pe_Radix2_cmpx2 et Pe_butterfly_simple sont représentées dans les figures 4.4 et 4.6 respectivement.

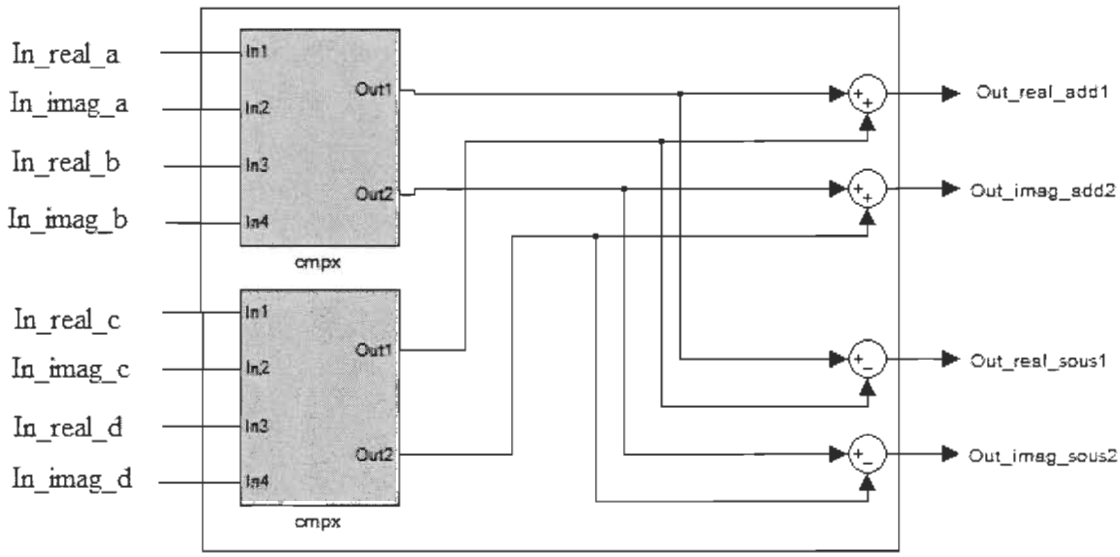


Figure 4.5 entité Pe_Radix2_cmpx2

On remarque que l'entité Pe_Radix2_cmpx2 se compose de deux entités multiplieur complexe (d'où la nomenclature cmpx2), de deux entités addition et de deux entités soustraction. Les deux multiplieurs complexes servent à la multiplication des données par les *twiddle factor*.

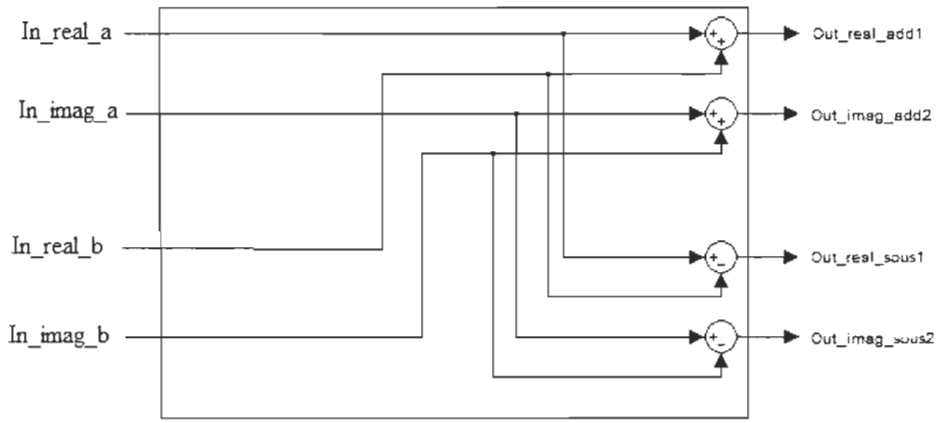


Figure 4.6 entité Pe_butterfly_simple

La figure 4.6 représente l'entité Pe_butterfly_simple. Cette dernière traduit une opération butterfly simple représentée dans la figure 4.7 ci-dessous. Elle utilise deux entités addition (une pour la partie réelle et l'autre pour la partie imaginaire) et deux entités soustraction (partie réelle et partie imaginaire).

La figure 4.7 (a et b) explique la structure Pe_butterfly_simple en démontrant la séparation des entrées en réels et imaginaires.

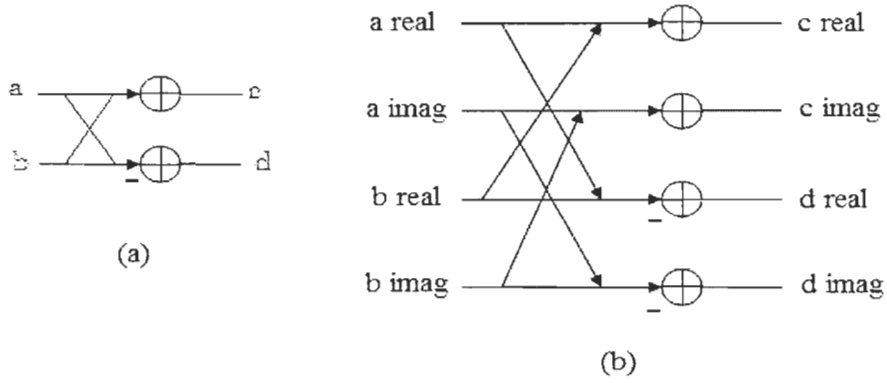


Figure 4.7 : (a) butterfly Radix2 (b) butterfly Radix2 avec les entrées/sorties séparées en réels et imaginaires

Pour la phase 3 du SFG du BPE Radix-4, on remarque que les données sont injectées dans le «bloc *j*». Une solution a été trouvée pour éviter une multiplication par *j*. Grâce à la modularité dans l'adder tree, la multiplication par *j* devient très triviale, elle peut être résolue en inversant les entrées et les sorties du butterfly. Dans ce qui suit nous expliquons la méthode de simplification du «bloc *j*».

La figure 4.7.b est générée par les équations suivantes :

$$\begin{cases} a_{real} + b_{real} = c_{real} \\ a_{imag} + b_{imag} = c_{imag} \\ a_{real} - b_{real} = d_{real} \\ a_{imag} - b_{imag} = d_{imag} \end{cases} \quad (4.3)$$

En multipliant l'entrée b de la butterfly de la figure 4.7.a par le «bloc *j*» (voir ci-dessous la figure 4.8), on inverse les entrées 3 et 4 de la butterfly de la figure 4.7.b.

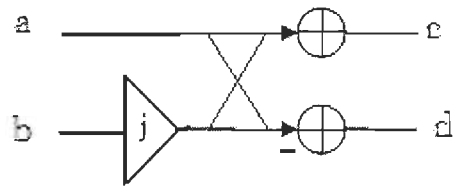


Figure 4.8 : multiplication de l'entrée b par le «bloc *j*»

On démontre cette inversion dans ce qui suit :

On a:

$$b = b_{real} + j\, b_{imag} \quad (4.4)$$

On a aussi :

$$b \times j = (b_{real} + j\, b_{imag}) \times j = -b_{imag} + j \times b_{real} \quad (4.5)$$

Donc, on remplace dans les équations (4.3) l'entrée b_{real} par $-b_{imag}$ et b_{imag} par b_{real} , puis (4.3) devient :

$$\begin{cases} a_{real} - b_{imag} = c_{real} \\ a_{imag} + b_{real} = c_{imag} \\ a_{real} + b_{imag} = d_{real} \\ a_{imag} - b_{real} = d_{imag} \end{cases} \quad (4.6)$$

On remet l'équation (4.6) dans l'ordre afin de garder la structure primaire du butterfly de la figure 4.7.b et on obtient :

$$\begin{cases} a_{real} + b_{imag} = d_{real} \\ a_{imag} + b_{real} = c_{imag} \\ a_{real} - b_{imag} = c_{real} \\ a_{imag} - b_{real} = d_{imag} \end{cases} \quad (4.7)$$

Donc les équations (4.7), nous donnent la nouvelle structure du butterfly représentée ci-dessous dans la figure (4.9).

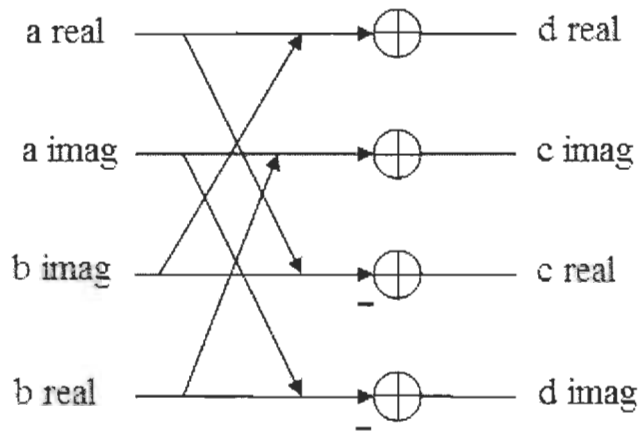


Figure 4.9 : structure butterfly résultante après multiplication par «bloc *j*»

La butterfly représentée dans la figure 4.9 ci-dessus remplacera le «bloc *j*» et l'entité Pe_butterfly_simple_2 dans l'architecture de l'entité finale du BPE Radix-4.

La figure 4.10 représente l'architecture de l'entité BPE Radix-4 programmée en VHDL.

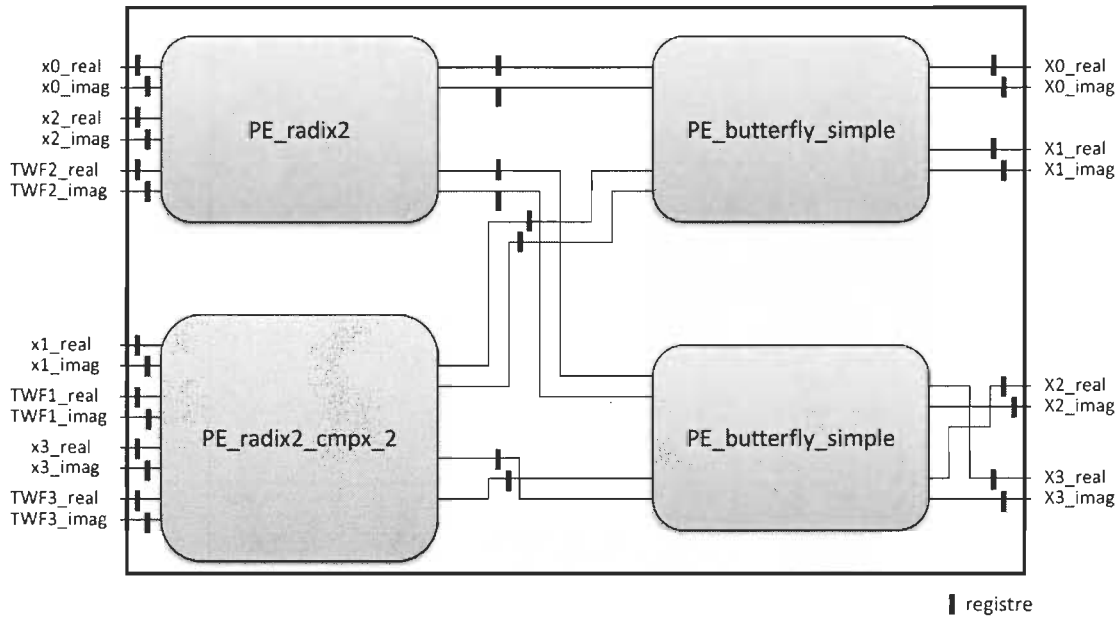


Figure 4.10 Architecture de l'entité BPE Radix-4 conventionnel

La figure 4.10 représente l'entité du BPE Radix-4 conventionnel. Cette dernière est donc constituée de quatre entités. On remarque au niveau de l'entité 'Pe_butterfly_simple', en bas à droite du schéma, qu'elle utilise la nouvelle structure de la figure 4.9. On constate l'inversement des entrées et sorties afin d'évité la multiplication par le «bloc *j*».

4.2.1.3 Radix-8

L'architecture simplifiée du butterfly Radix-8 conventionnel est représentée dans la figure 2.10. On constate que le calcul se fait en plusieurs étapes et la structure du butterfly est plus complexe puisqu'elle possède des multiplications triviales et non triviales qui viennent s'ajouter au chemin critique du BPE. Les multiplications non triviales sont celles où les données sont multipliées par des *twiddle factor* internes contenus dans la matrice adder-tree T_8. Les multiplications triviales quant à elles, sont celles où les données passent par le «bloc *-j*». Comme pour le cas du Radix-4, ici la multiplication par le «bloc *–j*» peut être simplifiée en effectuant quelques modifications sur les équations générant la structure butterfly qui contiendra le «bloc *–j*». La figure 4.11 ci-dessous représente la structure butterfly contenant la multiplication par le «bloc *–j*».

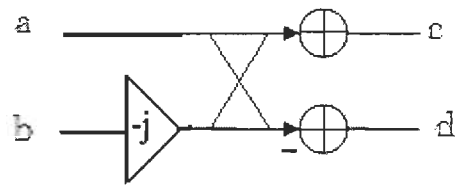


Figure 4.11 : multiplication de l'entrée b par le «bloc *-j*»

On démontre la simplification de la multiplication triviale de la figure 4.11 par ce qui suit :

À partir de l'équation (4.4) et l'équation (4.8) ci-dessous :

$$b \times (-j) = \left(b_{real} + j\, b_{imag}\right) \times (-j) = b_{imag} + j \times (-b_{real}) \qquad (4.8)$$

On déduit après multiplication par *-j* que b_{real} de l'équation (4.4) devient b_{imag} dans l'équation (4.8) et b_{imag} de (4.4) devient $-b_{real}$ dans (4.8).

Donc :

$$\begin{cases} b_{real} = b_{imag} \\ b_{imag} = -b_{real} \end{cases} \tag{4.9}$$

On remplace donc b_{real} et b_{imag} de (4.9) dans l'équation (4.3) par leurs nouvelles valeurs b_{imag} et $-b_{real}$ respectivement.

On obtient :

$$\begin{cases} a_{real} + b_{imag} = c_{real} \\ a_{imag} - b_{real} = c_{imag} \\ a_{real} - b_{imag} = d_{real} \\ a_{imag} + b_{real} = d_{imag} \end{cases} \tag{4.10}$$

On remet les équations (4.10) dans l'ordre afin de respecter la structure de base du butterfly, on obtient :

$$\begin{cases} a_{real} + b_{imag} = c_{real} \\ a_{imag} + b_{real} = d_{imag} \\ a_{real} - b_{imag} = d_{real} \\ a_{imag} - b_{real} = c_{imag} \end{cases} \tag{4.11}$$

Donc les équations (4.11), nous donnent la nouvelle structure du butterfly représentée ci-dessous dans la figure 4.12.

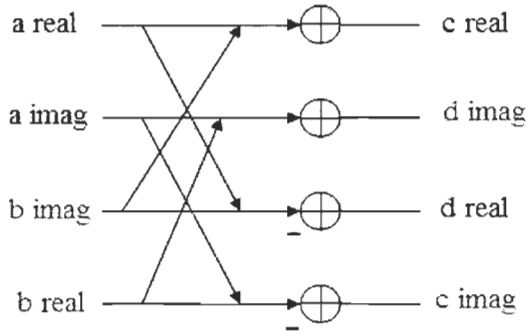


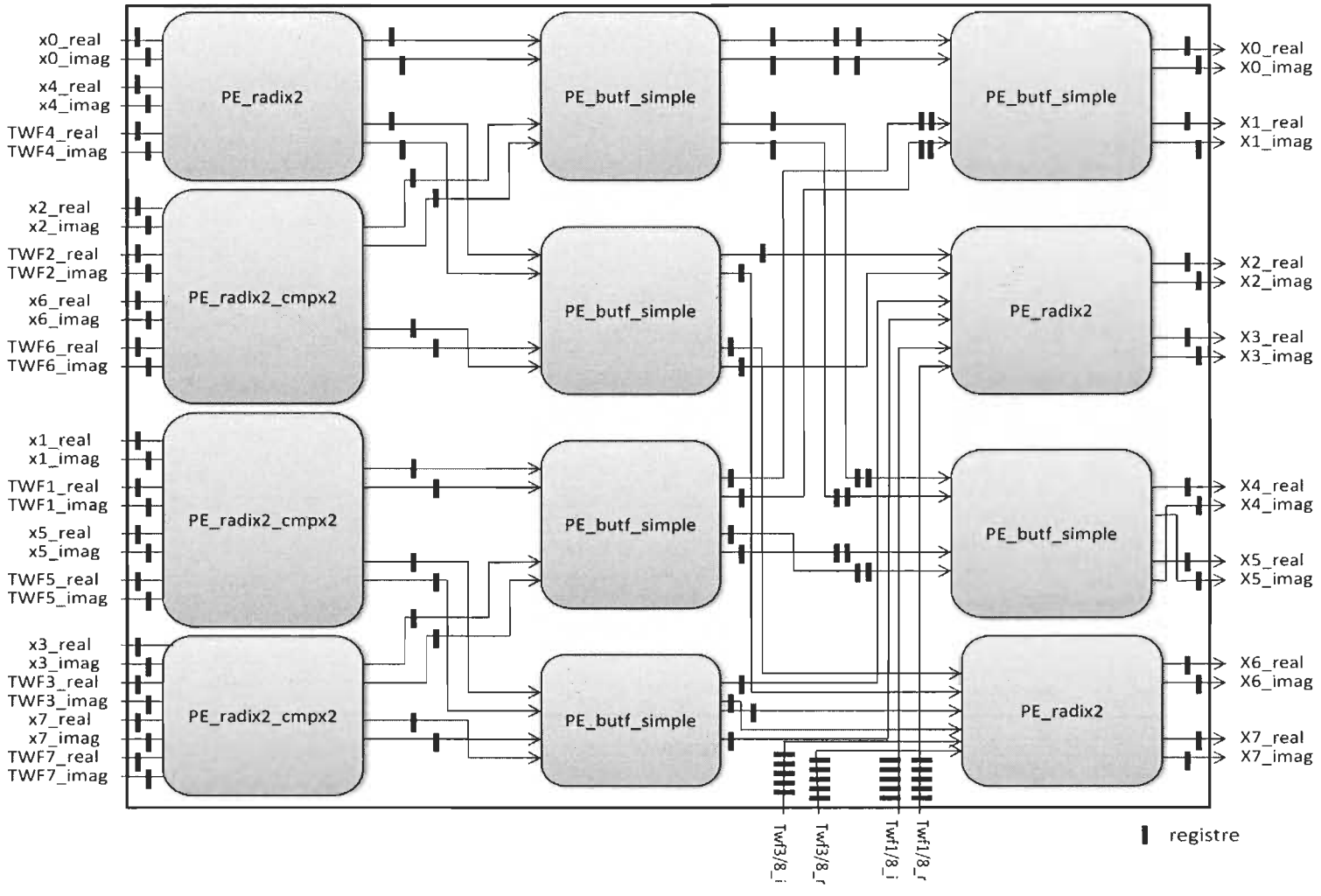
Figure 4.12 : structure butterfly résultante après multiplication par le «bloc $-j$»

La butterfly représentée dans la figure 4.12 ci-dessus remplacera le «bloc $-j$» et l'entité Pe_butterfly_simple dans l'architecture de l'entité finale du BPE Radix-8.

La figure 4.13 ci-dessous, représente l'architecture de l'entité BPE Radix-8 programmée en VHDL.

L'architecture du BPE Radix-8 est donc constituée de douze entités : trois entités Pe_Radix2, trois entités Pe_Radix2_cmpx2 et six entités Pe_butterfly_simple. On constate que tout les «blocs $-j$» sont intégrés dans les entités Pe_butterfly_simple, en inversant les entrées et les sorties, ainsi toutes les multiplications triviales sont éliminées. Il reste donc les multiplications non triviales par les *twiddle factor* ($e^{-2\pi/8}$et $e^{-3\pi/8}$) qui seront intégrées dans les deux Pe_Radix2 de la dernière phase du calcul du BPE.

Figure 4.13 Architecture de l'entité BPE Radix-8 conventionnel



4.2.2 Butterfly JFFT

4.2.2.1 JFFT4

L'architecture simplifiée du butterfly JFFT4 est représentée dans la figure 2.13.

On remarque dans la structure du BPE JFFT que la multiplication par les *twiddle factor* est effectuée au début de la structure en une seule phase. Cette nouvelle structure utilise une structure *butterfly* différente de celle du BPE conventionnel. La figure 4.14 représente cette nouvelle structure.

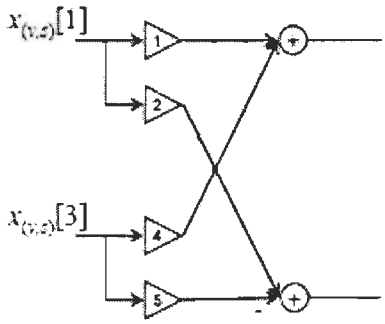


Figure 4.14 nouvelle structure butterfly dans la JFFT

La structure présentée dans la figure 4.14 utilise quatre multiplieurs complexes, un additionneur complexe et un soustracteur complexe. Et puisque sa structure ne respecte pas la structure de base de la *butterfly*, sa conception en VHDL traduira exactement les modules qui la composent. La figure 4.15 représente l'entité de la nouvelle structure butterfly dans la JFFT.

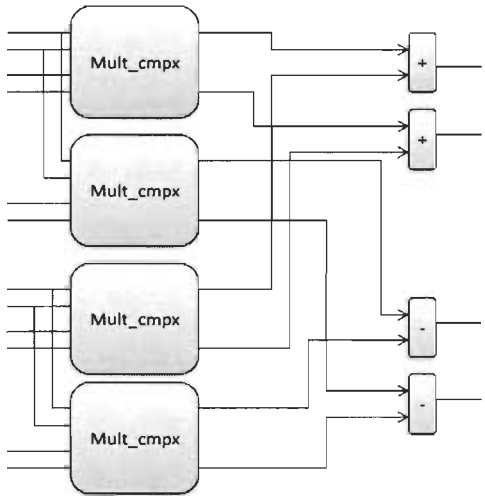


Figure 4.15 nouvelle structure butterfly dans la JFFT

La figure 4.16 ci-dessous, représente l'entité du BPE JFFT4.

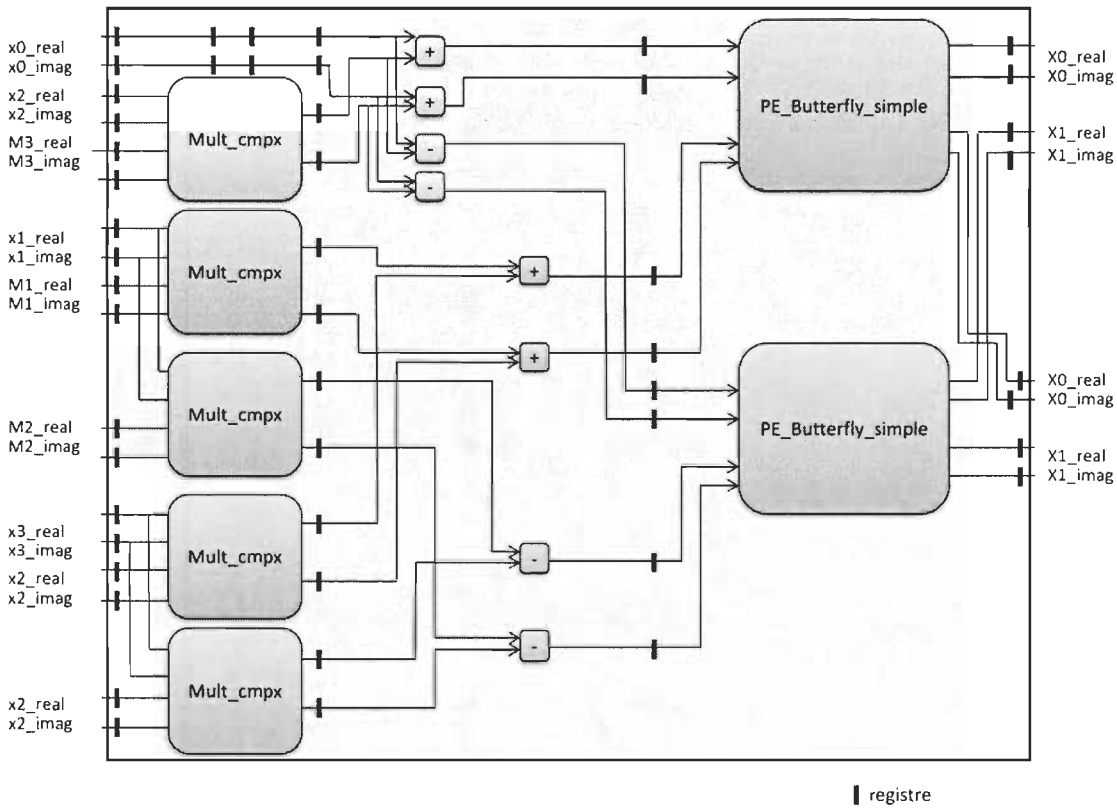


Figure 4.16 Architecture de l'entité BPE JFFT4

Le BPE JFFT4 se compose d'une entité PE_Radix2, de deux entités PE_butterfly_simple et de (quatre multiplieurs complexes, un additionneur complexe et un soustracteur complexe) qui composent l'entité de la nouvelle structure butterfly JFFT. Le programme VHDL de ce BPE est disponible voir [ACH10].

4.2.2.2 JFFT8

L'architecture simplifiée du *butterfly* JFFT8 est représentée dans la figure 2.12. On remarque dans la structure du BPE JFFT que la multiplication par les *twiddle factor* est effectuée au début de la structure en une seule phase, contrairement au BPE Radix-8 conventionnel. Ainsi les multiplications non triviales qui existent au milieu du calcul du BPE Radix-8 conventionnel sont intégrées au début de l'architecture du BPE JFFT8.

Cette nouvelle structure proposée par la JFFT, rend le chemin critique du BPE de Radix élevé moins complexe, [JAB09].

À partir du SFG du BPE JFFT8 figure 2.12, on remarque que sa structure utilise des multiplieurs par le «bloc $-j$» et on remarque aussi la forme de nouvelles *butterfly*. La figure 4.17 représente une de ces nouvelles structures butterfly Radix-8.

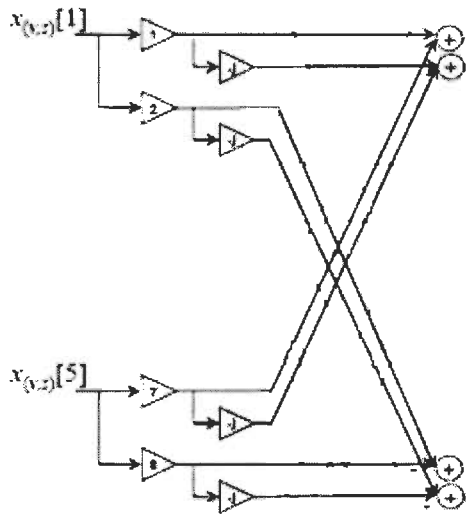


Figure 4.17 structure d'entrée Butterfly JFFT8 avec multiplieurs *twiddle factor* et « bloc –j»

La figure 4.17 représente une des butterfly d'entrée du BPE JFFT8. On constate que les données sont injectées dans deux butterfly imbriquées. Dans la première butterfly, les données sont multipliées par les *twiddle factor* seulement, par contre dans la deuxième après être multipliées par les *twiddle factor*, les données passent par le «bloc *–j*». Pour éliminer les multiplications triviales résultantes du «bloc *–j*», on a recours à quelques simplifications. À partir des figures 4.18.a et 4.18.b, on déduit les équations des opérations de la deuxième butterfly qui contient le «bloc *–j*» et on montre la façon avec laquelle on simplifie le SFG pour la programmation sur VHDL.

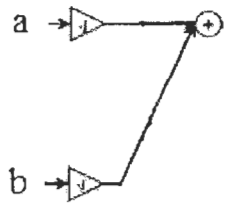


Figure 4.18.a Partie addition bloc $-j$

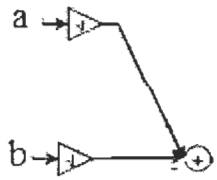


Figure 4.18.b Partie soustraction bloc $-j$

À partir de la figure 4.18.a, on déduit les équations suivantes :

On a :

$$a \times (-j) = (a_{real} + j\, a_{imag}) \times (-j) = a_{imag} + j \times (-a_{real})$$

$$b \times (-j) = (b_{real} + j\, b_{imag}) \times (-j) = b_{imag} + j \times (-b_{real})$$

(4.12)

Donc l'équation pour l'opération 'a + b' est la suivante:

$$a \times (-j) + b \times (-j) = \left(a_{imag} + j \times (-a_{real})\right) + \left(b_{imag} + j \times (-b_{real})\right)$$

$$= (a_{imag} + b_{imag}) + j\,(-a_{real} - b_{real})$$

(4.13)

À partir de l'équation (4.13), on remarque que la partie réelle de 'a + b' devient $(a_{imag} + b_{imag})$ et la partie imaginaire de 'a + b' devient $(-a_{real} - b_{real})$.

Afin de modifier la structure de la figure 4.18.a et éliminer la multiplication par $-j$, on adopte la structure de la figure 4.19 ci-dessous :

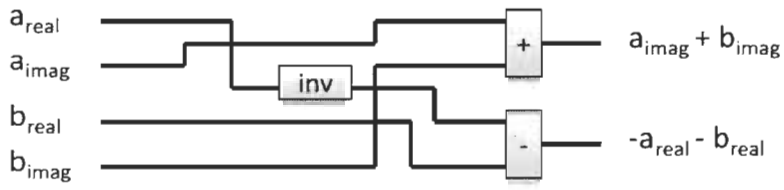


Figure 4.19 Structure 'a + b' éliminant la multiplication par *-j*

Dans la figure 4.19 ci-dessus, on remarque la méthode utilisée pour simplifier la structure de la figure 4.18.a. On constate que pour obtenir le $-a_{real}$, on crée une entité inverseur. L'entité inverseur n'ajoute pas de complexité matérielle puisqu'elle a besoin que de quelques portes logiques, donc n'influence aucunement sur la complexité du chemin critique du BPE.

On reprend la même méthode pour la structure de la figure 4.18.b. À partir des équations (4.12), on en déduit l'équation du module 'a - b' :

$$a \times (-j) - b \times (-j) = \left(a_{imag} + j \times (-a_{real})\right) - \left(b_{imag} + j \times (-b_{real})\right)$$

$$= \left(a_{imag} - b_{imag}\right) + j\left(-a_{real} + b_{real}\right) \tag{4.14}$$

On obtient alors la structure représentée dans la figure 4.20 ci-dessous :

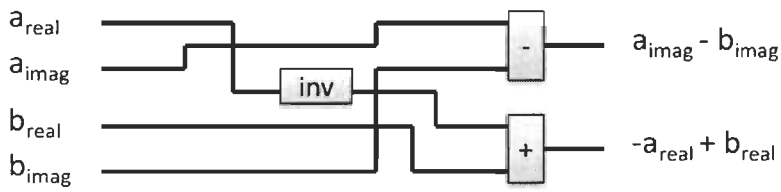
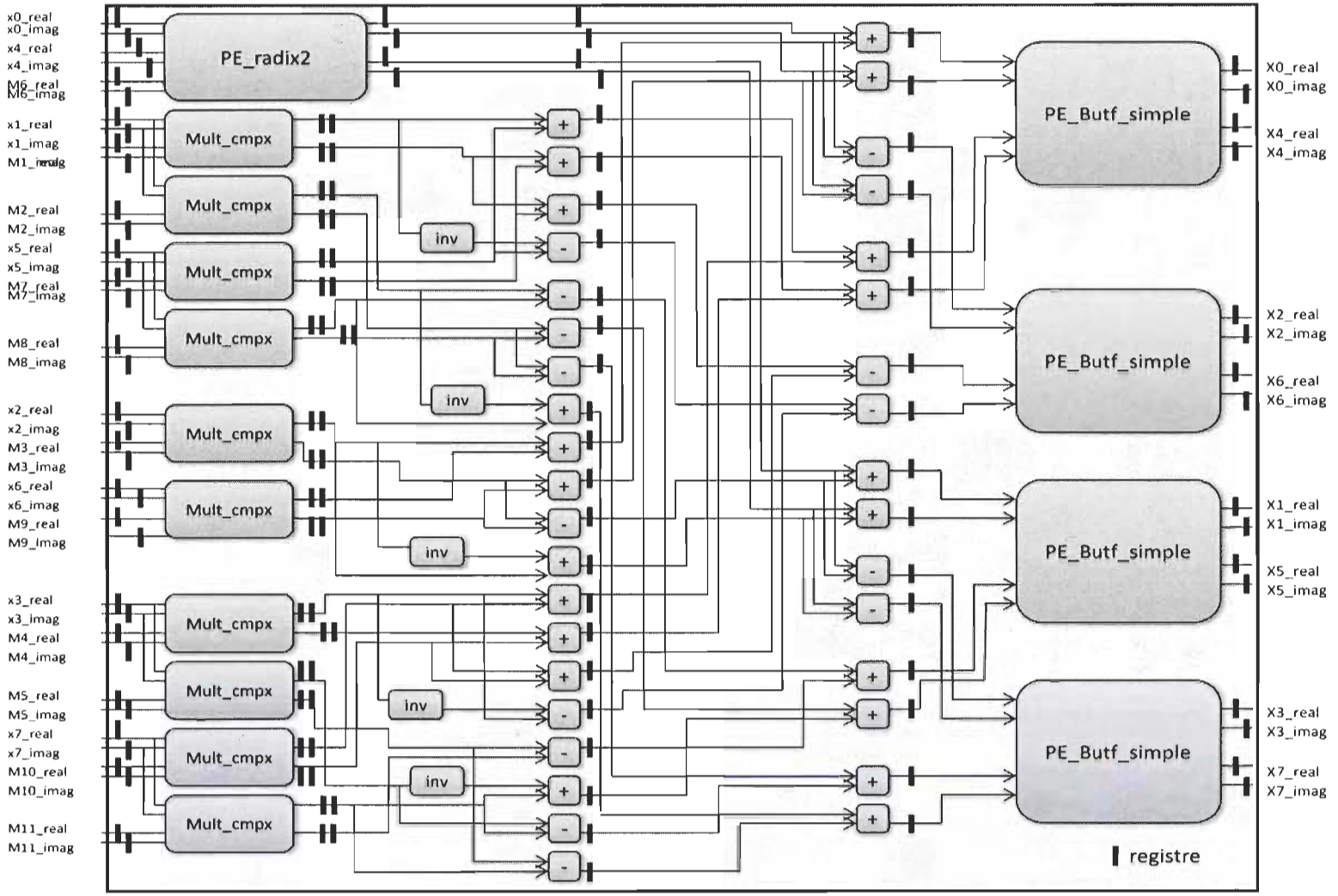


Figure 4.20 Structure 'a - b' éliminant la multiplication par *-j*

La figure 4.21 ci-dessous représente l'architecture de l'entité BPE JFFT8 conçue en VHDL.

Figure 4.21 Architecture de l'entité BPE JFFT8



L'entité BPE JFFT8 se compose d'une entité 'Pe_Radix2', quatre entités 'Pe_butterfly_simple', dix entités 'multiplieurs complexes', vingt et une entités 'additionneurs réels', quinze entités 'soustracteur réels' et cinq entités 'inverseur'.

Avant son implémentation sur FPGA, l'architecture BPE JFFT8 de la figure 4.21 ci-dessus a été testée comme toutes les autres architectures à l'aide de la cosimulation Matlab-Modelsim de la figure 4.1. Les résultats de cosimulation du BPE JFFT8 sont représentés dans les figures 4.22 et 4.23 ci-dessous. Pour les résultats de cosimulation des autres architectures veuillez-vous référer au rapport technique du LSSI [ACH10].

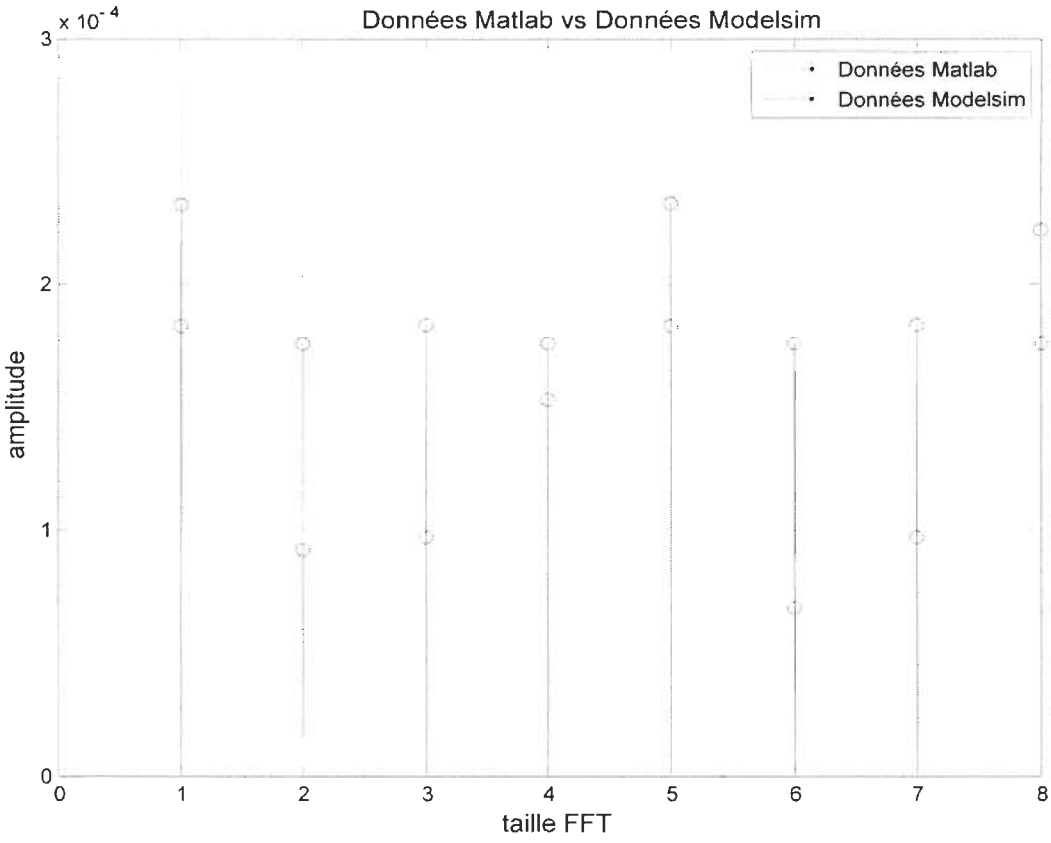
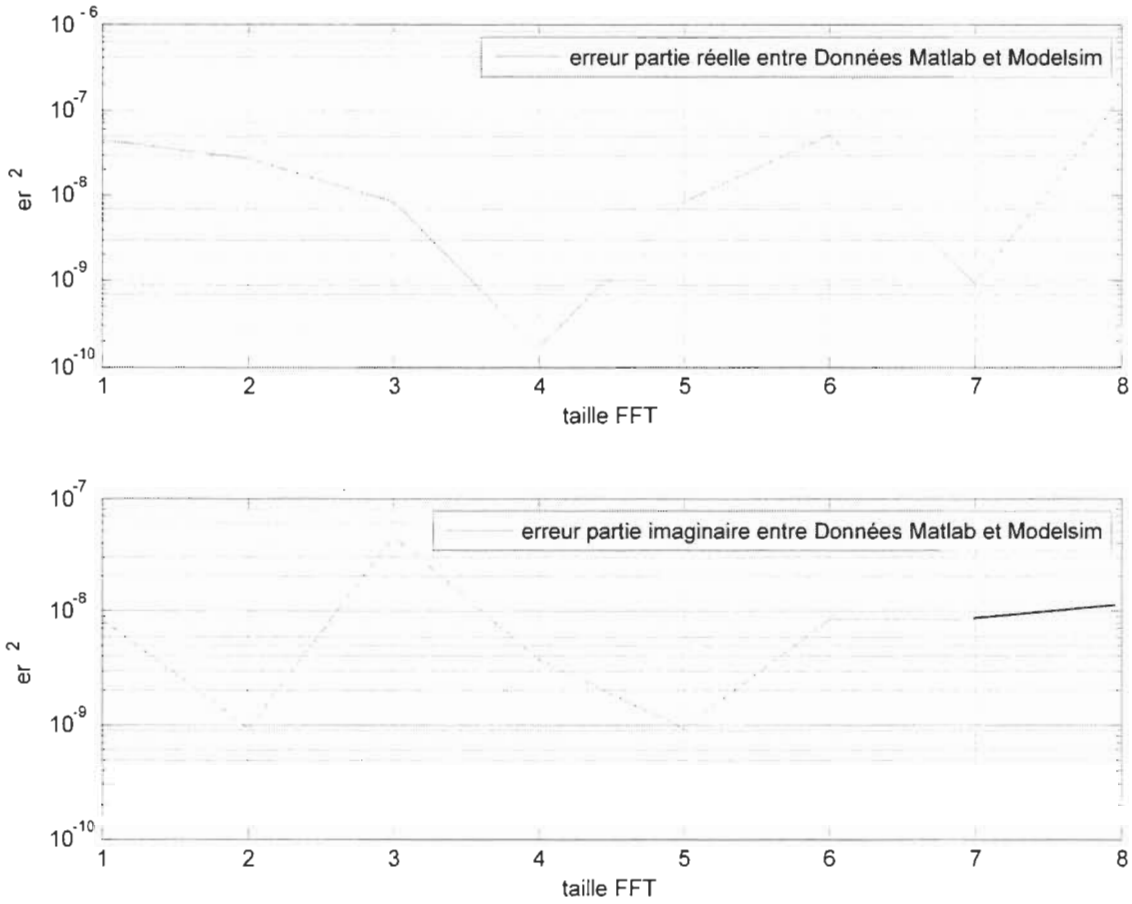


Figure 4.22 Erreur entre les résultats complexes fournis par les BPE JFFT8 de Matlab® et Modelsim®



Figures 4.23 Erreur au carré entre les résultats (partie réelle et partie imaginaire) générés par la cosimulation Matlab® et Modelsim® des BPE JFFT8

Les résultats de cosimulation pour le BPE JFFT8 illustrés dans les figures 4.22 et 4.23 démontrent le bon fonctionnement du BPE puisque l'erreur entre les données fournies par Matlab® et celles fournies par Modelsim® est très faible.

4.3 Étude théorique du chemin critique et des ressources matérielles

Dans cette partie, on étudie les performances des BPE JFFT comparés aux BPE des Radix conventionnels.

Les résultats de performances se composent en deux éléments :

- **Le temps de calcul** : chemin critique du BPE
- **Ressources matérielles** : unité arithmétique élémentaire en terme d'additionneur complet binaire (FA – Full Adder)

4.3.1 Chemin critique – Critical path

À partir des figures (2.14) et (2.15) qui représentent respectivement les chemins critiques des BPE Radix-8 conventionnel et JFFT8. On constate que le chemin critique du BPE Radix-8 conventionnel a besoin de quatre multiplications de valeur complexes et trois additions de valeurs complexes tandis que le BPE JFFT8 demande une multiplication complexe et trois additions complexes. On remarque une diminution significative des opérations au niveau du chemin critique du Radix-8 JFFT, [JAB09].

On calcule le chemin critique pour les autres Radix, on remarque donc que le Radix-4 a besoin de deux multiplications de valeur complexes et deux additions complexes tandis que le BPE JFFT4 demande une multiplication complexe et deux additions complexes. Les BPE Radix-2 conventionnel et le BPE Radix-2 JFFT ont par contre une structure identique, [JAB09].

Le chemin critique pour le Radix-2 est de 1 CTM +1 CTA.

Si TM=4TA, CTA=2TA, CTM= 3TM+5TA, Le temps du chemin critique = 4.75 (x TM).

Avec : TM : Temps d'une multiplication réelle

TA : Temps d'une addition réelle

CTM : Temps d'une multiplication complexe

CTA : addition complexe

Le tableau 4.2 ci-dessous résume les chemins critiques des différents Radix conventionnels et JFFT.

Tableau 4.2 Chemins critiques des différents Radix conventionnels et JFFT, [JAB09]

Butterfly	Temps de calcul [xTM]		Gain
	Conventionnel	JFFT	
Radix-2	4.75	4.75	1
Radix-4	5.25	5.00	1.05
Radix-8	10.25	5.25	1.95

D'après le tableau 4.2, on remarque que pour un Radix plus grand le gain en temps de calcul est très significatif, puisqu'on constate que pour le Radix-8, que le BPE JFFT est à peu près deux fois plus rapide que le BPE conventionnel.

4.3.2 Ressources matérielles

Pour une étude théorique, les ressources matérielles seront calculées à base d'unités arithmétiques élémentaires en terme d'additionneur complet binaire (FA – Full Adder).

Le tableau 4.3 ci-dessous représente le nombre de ressources en termes de Full-Adder avec (Multiplieurs (Mult) de 16bits et additionneurs (Add) en 32 bit).

Tableau 4.3 nombre de ressources en termes de Full-Adder avec (Multiplieurs de 16bits et additionneurs en 32 bit), [JAB09].

Butterfly	Conventionnels			JFFT		
	Mult	Add	FA	Mult	Add	FA
Radix-2	3	9	1056	3	9	1056
Radix-4	9	31	3296	15	41	5152
Radix-8	27	93	9888	33	111	12000

Les ressources matérielles sont calculées de la sorte :

Nombre de ressources = (nombre de Mult * nombre de bit^2 du Mult) + (nombre de Add * nombre de bit du Add)

Exemple pour Radix2 : $3*16^2 + 9*32 = 1056$

À partir des tableaux 4.2 et 4.3, pour le cas du Radix-8, on constate qu'avec juste 17,6% de surface silicium en plus le BPE JFFT8 est à peu près deux fois plus rapide que le Radix-8 conventionnel.

4.4 Résultats d'implémentation sur FPGA

Dans cette partie on présente les résultats d'implémentation obtenu des différentes architectures de BPE (Butterfly Processing Element) Radix-8 conventionnel et JFFT. Les résultats sont exprimés dans plusieurs tableaux selon la famille de FPGA. Nos résultats sont comparés avec les résultats obtenus par une nouvelle étude présentée par [ZHO09].

Les tableaux fournissent un résumé précis des résultats obtenus selon une taille de FFT choisie de 4096 points, une taille binaire de données et de TWF (Twiddle Factor) fixée à 16 bit.

L'évaluation de l'exécution de la FFT se base sur le compromis surface silicium/vitesse.

Cependant plusieurs paramètres rentrent en jeu afin de trouver le meilleur compromis surface silicium/vitesse. En parlant de surface silicium, on parle des ressources matérielles nécessaires pour implémenter la FFT. Pour la vitesse, on parle de fréquence d'exécution de la FFT, de débit et de latence.

4.4.1 Critères d'évaluation

Il existe plusieurs critères d'évaluation de l'exécution de la FFT. Un des critères les plus importants est la latence. L'importance de réduire la latence est que les normes et standards de la 3GPP deviennent de plus en plus exigeants sur la réduction de la latence. Les normes stipulent que les systèmes doivent traiter une trame dans l'intervalle de 10 ms. La latence correspond au délai entre la transmission et la réception du message. Ce qui oblige les sous-systèmes de transmission/réception de réduire au minimum les délais de latence. Un autre critère aussi important est le coût d'implémentation. La meilleure méthode d'implémentation pour une exécution d'une FFT complète, doit avoir une fréquence d'exécution rapide, tout en utilisant moins de ressources matérielles sur le FPGA. Dans notre projet, on compare différentes méthodes d'implémentation cas0, cas1, cas2 et cas 3; le cas0 est celui où le multiplieur complexe n'est pas pipeliné, le cas1 est le ' pipeline complet', le cas2 est celui où on ajoute des registres avant et après les multiplieurs réels 'pipeline partiel'; sur les BPE Radix-8 conventionnels et JFFT avec deux méthodes d'implémentation utilisées par [ZHO09] et cela sur plusieurs paramètres :

- Nombre de multiplieur câblés ou DSP48 : chaque FPGA possède des multiplieurs câblés ou des blocs appelés DSP48. Ces derniers sont des blocs développés par Xilinx pour les nouvelles versions de FPGA et sont utilisés pour maximiser la vitesse d'exécution de la multiplication. On remarque ainsi une augmentation considérable dans la fréquence d'exécution de la FFT.
- Fréquence de fonctionnement et délai : paramètres clés pour la comparaison des performances de la méthode d'implémentation. Le délai est calculé en nanoseconde (ns) et représente le temps de propagation des signaux à travers l'architecture, il constitue la limite de fonctionnement du circuit. La fréquence est calculée quant à elle en (MHz). Plus la fréquence de fonctionnement est élevée plus l'architecture est rapide.
- Nombre de cycles : représente le nombre de cycle d'horloge nécessaire pour exécuter une FFT complète ou la latence.
- Débit : Nombre d'échantillon de sortie de la structure en pipeline par le temps d'un cycle d'horloge.
- Fonction de performance (MS/s/slice) : dans les FPGA de xilinx, les slices sont des cellules logiques élémentaires, composées de 2 LUT et 2 bascules. Les LUT (*Look Up Table*) sont des générateurs de fonctions. Le nombre et l'architecture des LUT et des bascules dans une slice varient d'une famille FPGA à une autre. La fonction dévaluation de la performance exprimée en MS/s/slice représente le débit en Méga Échantillon par seconde par slice, les ressources. Plus la fonction performance

(MS/s/slice) est grande mieux c'est. Noter que nous ne considérons pas dans les ressources le nombre de multiplieur câblé, ceci étant fait séparément.

- Coût du BPE : représente le nombre de slices utilisés par le BPE multiplié par le délai du BPE (temps de propagation des signaux à travers le BPE).
- Coût de la FFT : représente le nombre de slices utilisés par la FFT multiplié par le temps d'exécution de la FFT en (µs). Ce dernier représente à son tour le nombre de cycle divisé par la fréquence de fonctionnement.
- Temps de transformée : c'est le temps nécessaire pour l'exécution d'une FFT complète en (µs).

4.4.2 Résultats d'implémentation

Les tableaux de 4.4 à 4.23 représentent les résultats d'implémentation des architectures BPE conventionnelles et JFFT tous deux Radix-8 pour les familles FPGA Virtex-E (xcv1600e-8-bg560), Spartan-3 (xc3s1000-5-fg456), Virtex-4 (xc4vlx100-12-ff1148) et Virtex-5 (xc5vlx110t-3-ff1136) respectivement. Tous les tableaux comparent les méthodes FFT par rapport à une taille précise de 4096 points, à une taille de données et de facteur (*Twiddle Factor*) fixée à 16 bits, au nombre de slices utilisés, au délai et fréquence d'horloge maximale, à la latence, au temps de transformée complète, au coût BPE et FFT, et finalement son évaluation de performance en MS/s/Slice.

Les tableaux 4.4, 4.10 et 4.16 affichent les résultats d'implémentation pour 1024 échantillons des méthodes FFT proposées dans l'article de référence [ZHO09] pour les trois familles Virtex-E, Spartan-3 et Virtex-4 respectivement. On constate que les deux

méthodes; la R2^{2}SDF et la R4SDC; proposées par [ZHO09], offrent le meilleur compromis surface silicium/vitesse puisqu'elles ont une fréquence de fonctionnement et un débit plus élevés avec moins de latence que les autres méthodes tout en utilisant peu de ressources matérielles. Dans le tableau 4.4, on constate aussi que la R2^{2}SDF et la R4SDC ont la meilleure performance (MS/s/Slice) par rapport aux autres méthodes de la littérature. Il est important de noter que nous avons évalué le cœur des étages pipelines des structures R2^{2}SDF et R4SDC proposées par [ZHO09] en utilisant nos outils de synthèse décrit à la section 4.1.1. Le cœur étant le multiplieur complexe selon le cas 1 et représenté à la figure 4.2.b. Nous avons atteint une fréquence d'horloge maximale de 118,3 MHz, 72,1 MHz et 228,8 MHz dans, respectivement, Spartan 3, le Virtex E, Virtex 4 de Xilinx. Ce qui est bien en dessous des fréquences présentées par [ZHO09], à savoir une réduction de 4,47%, 24,1% et 3,83% respectivement. Cependant, dans la suite, nous avons considéré les fréquences présentées dans [ZHO09] en guise de comparaison. R2^{2}SDF et R4SDC représentent les méthodes de références pour notre étude, plus particulièrement R2^{2}SDF qui présente une performance légèrement plus élevée que R4SDC.

Dans les tableaux 4.5, 4.11 et 4.17, on estime les données des deux meilleures méthodes, R2^{2}SDF et R4SDC pour une taille FFT de 4096 points pour pourvoir comparer les résultats en références avec les résultats de notre méthode proposée du Radix-8, 'la J8MDC'. On extrapole à la taille FFT de 4096 points puisque c'est la seule taille multiple de radix-2, radix-4 et radix-8. En conséquence, les résultats d'implémentation sont donc une estimation pour 4096 points. Nous assumons que la fréquence d'horloge est indépendante du nombre de point puisqu'elle dépend principalement du multiplieur complexe.

Les tableaux 4.6, 4.12 et 4.18 pour les familles Virtex-E, Spartan-3 et Virtex-4 respectivement, représentent les résultats de comparaison du gain en coût entre notre FFT proposée et les références. On note que l'architecture R2^{2}SDF est légèrement meilleure que la R4SDC, en termes de fonction de performance. Cependant le gain en performance calculé résulte de la comparaison entre notre FFT proposée J8MDC avec tous ces cas et l'architecture R2^{2}SDF. On remarque un gain en coût de 63,6% pour le J8MDC pour le cas1 et un gain en coût de 26,3% pour le cas3 par rapport à R2^{2}SDF, pour la Virtex-E. Le gain en coût est de 131,9% pour le cas 1 suivi de 73,5% pour le cas 3, pour la Virtex-4. Pour la famille Spartan-3, on constate un gain très important et très significatif qui est de 269,9% pour le cas1 et 253,3% pour le cas3. Une autre remarque importante par rapport à la fréquence de fonctionnement, c'est qu'on constate que cette dernière, augmente très significativement dans les Virtex-4 et Virtex-5, puisqu'on arrive à une fréquence maximale de 228,8 MHz et 277 MHz sur Virtex-4 et Virtex-5 respectivement au lieu de 72,1 MHz et 118,1 MHz sur Virtex-E et Spartan-3 respectivement. Cette augmentation est due à l'utilisation des Blocs DSP48 existants dans les Virtex-4 et 5 seulement.

Aussi, pour le Virtex-E, on n'arrive jamais à atteindre la fréquence maximale des deux architectures R2^{2}SDF ou R4SDC. Même en utilisant le cas 1 de la J8MDC où on utilise deux étages pipeline au lieu d'un seul comme pour les architectures de référence, la fréquence maximale est de 72,1 MHz pour la J8MDC cas 1 au lieu de 95 MHz pour la R2^{2}SDF. Mais malgré baisse en fréquence, la J8MDC cas 1, reste plus performante que la référence par rapport à la fonction de performance, à la latence et au débit par surface en MS/s/Slice.

Les tableaux 4.7, 4.13 et 4.19 pour les familles Virtex-E, Spartan-3 et Virtex-4 respectivement, représentent les résultats de comparaison du gain en latence entre notre architecture FFT proposée J8MDC et celles des références, la R2^{2}SDF et la R4SDC. Comme pour la comparaison du gain de performance, le gain en latence calculé résulte de la comparaison entre notre FFT proposée J8MDC avec tous ces cas et l'architecture R2^{2}SDF.

Selon les tableaux 4.7, 4.13 et 4.19, on souligne un gain maximal en latence de 88,9% pour le J8MDC cas1 pour le Spartan-3. Sinon, on enregistre un gain en latence qui se situe entre 88,9% et 73,7% pour tous les autres circuits FPGA.

Les tableaux 4.8, 4.14, 4.20 et 4.22 comparent notre architecture FFT proposée J8MDC et l'architecture conventionnelle R8MDC pour les familles Virtex-E, Spartan-3, Virtex-4 et Virtex-5 respectivement. Les meilleurs résultats sont donnés par la Spartan-3 pour le cas3 de la J8MDC puisqu'on constate un gain de performance de 151%, une réduction en temps de calcul de 59,3% et un débit par surface de 0,1001 MS/s/Slice.

Les tableaux 4.9, 4.15, 4.21 et 4.23 fournissent les résultats de comparaison entre notre architecture proposée J8MDC et l'architecture conventionnelle R8MDC par rapport à la latence pour les familles Virtex-E, Spartan-3, Virtex-4 et Virtex-5 respectivement. La réduction de latence la plus significative est de 59,4% donnée par l'architecture J8MDC cas3 sur Spartan-3.

Tableau 4.4 Résultats d'implémentation des méthodes de références [ZHO09] sur Virtex-E

VirtexE	Taille FFT	Taille données	Taille TWF	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée (pipeline plein)		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice
								Cycles	Temps (µs)				
Amphion	1024	13	13	1639	17,5	57,0	5097	4096	71,9	14,2	28754	117778	0,0087
Xilinx	1024	16	16	1968	12,0	83,0	4096	4096	49,3	20,8	23711	97120	0,0106
Sundance	1024	16	10	8031	20,4	49,0	1320	1320	26,9	49,0	163898	216345	0,0061
Suksawas	1024	16	16	7365	12,2	82,0	1099	1024	12,5	82,0	89817	91973	0,0111
R2^2SDF[09]	1024	16	16	5008	10,5	95,0	1042	1024	10,8	95,0	52716	53981	0,0190
R4SDC[09]	1024	16	16	7052	10,6	94,2	1041	1024	10,9	94,2	74862	76659	0,0134

Tableau 4.5 Résultats estimés pour 4096 points des méthodes de [ZHO09] sur Virtex-E

VirtexE	Taille FFT	Taille données et TWF	Mult. Cablé	Slices*	Délai (ns)	Freq. (MHz)	Latence (cycles)*	Temps de transformée		Débit (MÉ/s)	Coût* BPE	Coût* FFT	MS/s/Slice*
								Cycles*	Temps (µs)*				
R2^2SDF[09]	4096	16	18	6010	10,5	95,0	4114	4096	43,1	95,0	63259	259109	0,0158
R4SDC[09]	4096	16	18	8462	10,6	94,2	4113	4096	43,5	94,2	89834	367962	0,0111

* Estimation pour 4096 points

Tableau 4.6 Comparaison de performance entre JFFT8 et les FFT de références sur Virtex-E

VirtexE	Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Gain de performance sans Mult. (%)
								Cycles	Temps (µs)					
R2^2SDF[09]*	4096	16	18	6010	10,5	95,0	4114	4096	43,1	95,0	63259	259109	0,0158	0,0
R4SDC[09]*	4096	16	18	8462	10,6	94,2	4113	4096	43,5	94,2	89834	367962	0,0111	-29,6
J8MDC Cas0	4096	16	132	32404	93,5	10,7	608	512	47,9	85,5	3031200	1551974	0,0026	-83,3
J8MDC Cas1	4096	16	132	22320	13,9	72,1	616	512	7,1	577,2	309378	158401	0,0259	63,6
J8MDC Cas2	4096	16	132	23008	20,4	49,1	612	512	10,4	392,6	468811	240031	0,0171	7,9
J8MDC Cas3	4096	16	132	22340	17,9	55,7	612	512	9,2	445,9	400847	205233	0,0200	26,3

* Estimation pour 4096 points

Tableau 4.7 Comparaison de la latence entre 'JFFT8' et les FFT de références sur Virtex-E

VirtexE	Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Latence + FFT		Gain en Latence (%)
								Cycles	Temps (µs)		Cycles	Temps (µs)	
R2^2SDF[09]*	4096	16	18	6010	10,5	95,0	4114	4096	43,1	95,0	8210	86,4	0,0
R4SDC[09]*	4096	16	18	8462	10,6	94,2	4113	4096	43,5	94,2	8209	87,1	-0,8
J8MDC Cas0	4096	16	132	32404	93,5	10,7	608	512	47,9	85,5	1120	104,8	-21,2
J8MDC Cas1	4096	16	132	22320	13,9	72,1	616	512	7,1	577,2	1128	15,6	81,9
J8MDC Cas2	4096	16	132	23008	20,4	49,1	612	512	10,4	392,6	1124	22,9	73,5
J8MDC Cas3	4096	16	132	22340	17,9	55,7	612	512	9,2	445,9	1124	20,2	76,7

* Estimation pour 4096 points

Tableau 4.8 Comparaison entre la FFT proposée 'JFFT8' et la FFT conventionnelle sur Virtex-E

VirtexE		Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Réduction temps de calcul (%)	Gain de performance (%)
									Cycles	Temps (µs)						
Radix-8 Conv.	Cas 0	4096	16	108	26832	138,1	7,2	608	512	70,7	57,9	3706814	1897889	0,0022	-	-
	Cas 1	4096	16	108	25156	13,9	71,9	620	512	7,1	575,5	349694	179043	0,0229	-	-
	Cas 2	4096	16	108	25788	20,4	49,0	616	512	10,5	391,8	526488	269562	0,0152	-	-
	Cas 3	4096	16	108	25772	22,4	44,7	616	512	11,5	357,6	576520	295178	0,0139	-	-
Radix-8 JFFT	Cas 0	4096	16	132	32404	93,5	10,7	608	512	47,9	85,5	3031200	1551974	0,0026	32,3	22,3
	Cas 1	4096	16	132	22320	13,9	72,1	616	512	7,1	577,2	309378	158401	0,0259	0,3	13,0
	Cas 2	4096	16	132	23008	20,4	49,1	612	512	10,4	392,6	468811	240031	0,0171	0,2	12,3
	Cas 3	4096	16	132	22340	17,9	55,7	612	512	9,2	445,9	400847	205233	0,0200	19,8	43,8

Tableau 4.9 Comparaison de la latence entre JFFT8 et la FFT conventionnelle sur Virtex-E

VirtexE		Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Latence + FFT		Gain en Latence (%)
									Cycles	Temps (µs)		Cycles	Temps (µs)	
Radix-8 Conv.	Cas 0	4096	16	108	26832	138,1	7,2	608	512	70,7	57,9	1120	154,7	-
	Cas 1	4096	16	108	25156	13,9	71,9	620	512	7,1	575,5	1132	15,7	-
	Cas 2	4096	16	108	25788	20,4	49,0	616	512	10,5	391,8	1128	23,0	-
	Cas 3	4096	16	108	25772	22,4	44,7	616	512	11,5	357,6	1128	25,2	-
Radix-8 JFFT	Cas 0	4096	16	132	32404	93,5	10,7	608	512	47,9	85,5	1120	104,8	32,3
	Cas 1	4096	16	132	22320	13,9	72,1	616	512	7,1	577,2	1128	15,6	0,6
	Cas 2	4096	16	132	23008	20,4	49,1	612	512	10,4	392,6	1124	22,9	0,5
	Cas 3	4096	16	132	22340	17,9	55,7	612	512	9,2	445,9	1124	20,2	20,1

Tableau 4.10 Résultats d'implémentation des méthodes de références [ZHO09] sur Spartan-3

Spartan3	Taille FFT	Taille données et TWF	Slices	Délai (ns)	Freq. (MHZ)	Latence (cycles)	Temps de transformée (pipeline plein)		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/S/Slice
							Cycles	Temps (µs)				
R2^2SDF[09]	1024	16	2802	10,5	95,3	1042	1024	10,8	95,3	29417	30123	0,0340
R4SDC[09]	1024	16	4409	8,1	123,8	1041	1024	8,3	123,8	35602	36457	0,0281

Tableau 4.11 Résultats estimés pour 4096 points des méthodes de [ZHO09] sur Spartan-3

Spartan3	Taille FFT	Taille données et TWF	Slices*	Délai (ns)	Freq. (MHZ)	Latence (cycles)*	Temps de transformée		Débit (MÉ/S)	Coût* BPE	Coût* FFT	MS/S/Slice*
							Cycles*	Temps(µs)*				
R2^2SDF[09]	4096	16	3362	10,5	95,3	4114	4096	43,0	95,3	35301	144592	0,0283
R4SDC[09]	4096	16	5291	8,1	123,8	4113	4096	33,1	123,8	42723	174993	0,0234

* Estimation pour 4096 points

Tableau 4.12 Comparaison de performance entre JFFT8 et les FFT de références sur Spartan-3

Spartan 3	Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Gain de performance sans Mult.
								Cycles	Temps (µs)					
R2^2SDF[09]*	4096	16	18	3362	10,5	95,3	4114	4096	43,0	95,3	35301	144592	0,0283	0,0
R4SDC[09]*	4096	16	18	5291	8,1	123,8	4113	4096	33,1	123,8	42723	174993	0,0234	-17,4
J8MDC Cas0	4096	16	132	14340	87,6	11,4	608	512	44,9	91,3	1256471	643313	0,0064	-77,5
J8MDC Cas1	4096	16	132	9028	8,5	118,3	616	512	4,3	946,1	76341	39086	0,1048	269,9
J8MDC Cas2	4096	16	132	8168	11,4	87,7	612	512	5,8	701,9	93091	47662	0,0859	203,4
J8MDC Cas3	4096	16	132	8312	9,6	104,0	612	512	4,9	831,8	79945	40932	0,1001	253,3

* Estimation pour 4096 points

Tableau 4.13 Comparaison de la latence entre 'JFFT8' et les FFT de références sur Spartan-3

Spartan3	Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHZ)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Latence + FFT		Gain en Latence (%)
								Cycles	Temps (µs)		Cycles	Temps (µs)	
R2^2SDF[09]*	4096	16	18	3362	10,5	95,3	4114	4096	43,0	95,3	8210	86,2	0,0
R4SDC[09]*	4096	16	18	5291	8,1	123,8	4113	4096	33,1	123,8	8209	66,3	23,1
J8MDC Cas0	4096	16	132	14340	87,6	11,4	608	512	44,9	91,3	1120	98,1	-13,9
J8MDC Cas1	4096	16	132	9028	8,5	118,3	616	512	4,3	946,1	1128	9,5	88,9
J8MDC Cas2	4096	16	132	8168	11,4	87,7	612	512	5,8	701,9	1124	12,8	85,1
J8MDC Cas3	4096	16	132	8312	9,6	104,0	612	512	4,9	831,8	1124	10,8	87,5

* Estimation pour 4096 points

Tableau 4.14 Comparaison entre la FFT proposée 'JFFT8' et la FFT conventionnelle sur Spartan-3

Spartan 3		Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Réduction temps de calcul (%)	Gain de performance (%)
									Cycles	Temps (µs)						
Radix-8 Conv.	Cas 0	4096	16	108	9040	90.7	11.0	608	512	46.4	88.2	819684	419678	0.0098	-	-
	Cas 1	4096	16	108	10796	15.3	65.5	620	512	7.8	524.3	164736	84345	0.0486	-	-
	Cas 2	4096	16	108	9308	21.1	47.5	616	512	10.8	380.0	195943	100323	0.0408	-	-
	Cas 3	4096	16	108	8508	23.6	42.4	616	512	12.1	338.8	200882	102852	0.0398	-	-
Radix-8 JFFT	Cas 0	4096	16	132	14340	87.6	11.4	608	512	44.9	91.3	1256471	643313	0.0064	3.4	-34.8
	Cas 1	4096	16	132	9028	8.5	118.3	616	512	4.3	946.1	76341	39086	0.1048	44.6	115.8
	Cas 2	4096	16	132	8168	11.4	87.7	612	512	5.8	701.9	93091	47662	0.0859	45.9	110.5
	Cas 3	4096	16	132	8312	9.6	104.0	612	512	4.9	831.8	79945	40932	0.1001	59.3	151.3

* Estimation pour 4096 points

Tableau 4.15 Comparaison de la latence entre JFFT8 et la FFT conventionnelle sur Spartan-3

Spartan3		Taille FFT	Taille données et TWF	Mult. Cablé	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Latence + FFT		Gain en Latence (%)
									Cycles	Temps (µs)					Cycles	Temps (µs)	
Radix-8 Conv.	Cas 0	4096	16	108	9040	90.7	11.0	608	512	46.4	88.2	819684	419678	0.0098	1120	101.6	-
	Cas 1	4096	16	108	10796	15.3	65.5	620	512	7.8	524.3	164736	84345	0.0486	1132	17.3	-
	Cas 2	4096	16	108	9308	21.1	47.5	616	512	10.8	380.0	195943	100323	0.0408	1128	23.7	-
	Cas 3	4096	16	108	8508	23.6	42.4	616	512	12.1	338.8	200882	102852	0.0398	1128	26.6	-
Radix-8 JFFT	Cas 0	4096	16	132	14340	87.6	11.4	608	512	44.9	91.3	1256471	643313	0.0064	1120	98.1	3.4
	Cas 1	4096	16	132	9028	8.5	118.3	616	512	4.3	946.1	76341	39086	0.1048	1128	9.5	44.8
	Cas 2	4096	16	132	8168	11.4	87.7	612	512	5.8	701.9	93091	47662	0.0859	1124	12.8	46.1
	Cas 3	4096	16	132	8312	9.6	104.0	612	512	4.9	831.8	79945	40932	0.1001	1124	10.8	59.4

* Estimation pour 4096 points

Tableau 4.16 Résultats estimés pour 4096 points des méthodes de [ZHO09] sur Virtex-4

Virtex4	Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHZ)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/S/Slice
								Cycles	Temps (µs)				
R2^2 SDF[09]	1024	16	16	2256	4,2	235,6	1042	1024	4,3	235,6	9576	9805	0,104
R4SDC[09]	1024	16	16	3064	4,6	219,2	1041	1024	4,7	219,2	13978	14314	0,072

Tableau 4.17 Résultats d'implémentation des méthodes de références [ZHO09] extrapolés, sur Virtex-4

Virtex4	Taille FFT	Taille données et TWF	DSP48	Slices*	Délai (ns)	Freq. (MHZ)	Latence (cycles)*	Temps de transformée		Débit (MÉ/S)	Coût* BPE	Coût* FFT	MS/S/Slice*
								Cycles*	Temps(µs)*				
R2^2SDF[09]	4096	16	19	2707	4,2	235,6	4114	4096	17,4	235,6	11491	47066	0,087
R4SDC[09]	4096	16	19	3677	4,6	219,2	4113	4096	18,7	219,2	16774	68705	0,060

* Estimation pour 4096 points

Tableau 4.18 Comparaison de performance entre JFFT8 et les FFT de références sur Virtex-4

Virtex4	Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Gain de performance sans DSP48
								Cycles	Temps (µs)					
R2^2SDF[09]*	4096	16	19	2707	4,2	235,6	4114	4096	17,4	235,6	11491	47066	0,0870	0,0
R4SDC[09]*	4096	16	19	3677	4,6	219,2	4113	4096	18,7	219,2	16774	68705	0,0596	-31,5
J8MDC Cas0	4096	16	132	8472	34,7	28,8	608	512	17,8	230,5	294072	150565	0,0272	-68,7
J8MDC Cas1	4096	16	88	9072	4,4	228,8	616	512	2,2	1830,7	39645	20298	0,2018	131,9
J8MDC Cas2	4096	16	88	8168	6,7	149,7	612	512	3,4	1197,2	54579	27944	0,1466	68,4
J8MDC Cas3	4096	16	88	8312	6,4	156,9	612	512	3,3	1254,9	52989	27130	0,1510	73,5

* Estimation pour 4096 points

Tableau 4.19 Comparaison de la latence entre 'JFFT8' et les FFT de références sur Virtex-4

Virtex4	Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHZ)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Latence + FFT		Gain en Latence (%)
								Cycles	Temps (µs)		Cycles	Temps (µs)	
R2^2SDF[09]	4096	16	19	2707	4,2	235,6	4114	4096	17,4	235,6	8210	34,8	0,0
R4SDC[09]	4096	16	19	3677	4,6	219,2	4113	4096	18,7	219,2	8209	37,4	-7,5
J8MDC Cas0	4096	16	132	8472	34,7	28,8	608	512	17,8	230,5	1120	38,9	-11,6
J8MDC Cas1	4096	16	88	9072	4,4	228,8	616	512	2,2	1830,7	1128	4,9	85,9
J8MDC Cas2	4096	16	88	8168	6,7	149,7	612	512	3,4	1197,2	1124	7,5	78,4
J8MDC Cas3	4096	16	88	8312	6,4	156,9	612	512	3,3	1254,9	1124	7,2	79,4

* Estimation pour 4096 points

Tableau 4.20 Comparaison entre la FFT proposée 'JFFT8' et la FFT conventionnelle sur Virtex-4

Virtex4		Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Réduction temps de calcul (%)	Gain de performance (%)
									Cycles	Temps (µs)						
Radix-8 Conv.	Cas 0	4096	16	108	8472	34,7	28,8	608	512	17,8	230,5	294072	150565	0,0272	-	-
	Cas 1	4096	16	108	9120	4,4	229,6	620	512	2,2	1836,5	39727	20340	0,2014	-	-
	Cas 2	4096	16	108	7560	6,7	149,6	616	512	3,4	1196,7	50539	25876	0,1583	-	-
	Cas 3	4096	16	108	6732	8,0	125,6	616	512	4,1	1004,9	53593	27440	0,1493	-	-
Radix-8 JFFT	Cas 0	4096	16	132	7180	50,9	19,7	608	512	26,0	157,3	365204	186984	0,0219	-46,5	-19,5
	Cas 1	4096	16	88	9072	4,4	228,8	616	512	2,2	1830,7	39645	20298	0,2018	-0,3	0,2
	Cas 2	4096	16	88	8168	6,7	149,7	612	512	3,4	1197,2	54579	27944	0,1466	0,0	-7,4
	Cas 3	4096	16	88	8312	6,4	156,9	612	512	3,3	1254,9	52989	27130	0,1510	19,9	1,1

* Estimation pour 4096 points

Tableau 4.21 Comparaison de la latence entre JFFT8 et la FFT conventionnelle sur Virtex-4

Virtex4		Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Latence – FFT		Gain en Latence (%)
									Cycles	Temps (µs)					Cycles	Temps (µs)	
Radix-8 Conv.	Cas 0	4096	16	108	7180	50,9	19,7	608	512	26,0	157,3	365204	186984	0,0219	1120	57,0	-
	Cas 1	4096	16	108	9120	4,4	229,6	620	512	2,2	1836,5	39727	20340	0,2014	1132	4,9	-
	Cas 2	4096	16	108	7560	6,7	149,6	616	512	3,4	1196,7	50539	25876	0,1583	1128	7,5	-
	Cas 3	4096	16	108	6732	8,0	125,6	616	512	4,1	1004,9	53593	27440	0,1493	1128	9,0	-
Radix-8 JFFT	Cas 0	4096	16	132	8472	34,7	28,8	608	512	17,8	230,5	294072	150565	0,0272	1120	38,9	31,8
	Cas 1	4096	16	88	9072	4,4	228,8	616	512	2,2	1830,7	39645	20298	0,2018	1128	4,9	0,0
	Cas 2	4096	16	88	8168	6,7	149,7	612	512	3,4	1197,2	54579	27944	0,1466	1124	7,5	0,4
	Cas 3	4096	16	88	8312	6,4	156,9	612	512	3,3	1254,9	52989	27130	0,1510	1124	7,2	20,2

* Estimation pour 4096 points

Tableau 4.22 Comparaison entre la FFT proposée 'JFFT8' et la FFT conventionnelle sur Virtex-5

Virtex5		Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Réduction temps de calcul (%)	Gain de performance (%)
									Cycles	Temps (µs)						
Radix-8 Conv.	Cas 0	4096	16	108	5888	40,4	24,7	608	512	20,7	197,9	237964	121837	0,0336	-	-
	Cas 1	4096	16	108	12224	3,6	277,7	620	512	1,8	2221,6	44019	22538	0,1817	-	-
	Cas 2	4096	16	108	9604	5,5	181,7	616	512	2,8	1453,5	52860	27065	0,1513	-	-
	Cas 3	4096	16	108	8092	6,7	150,0	616	512	3,4	1199,8	53957	27626	0,1483	-	-
Radix-8 JFFT	Cas 0	4096	16	132	9764	27,2	36,7	608	512	13,9	293,9	265796	136087	0,0301	32,6	-10,5
	Cas 1	4096	16	88	9072	3,6	277,5	616	512	1,8	2220,4	32686	16735	0,2447	-0,1	34,7
	Cas 2	4096	16	88	8168	5,5	181,6	612	512	2,8	1452,7	44981	23030	0,1779	-0,1	17,5
	Cas 3	4096	16	88	8312	5,3	190,3	612	512	2,7	1522,6	43671	22360	0,1832	21,2	23,6

* Estimation pour 4096 points

Tableau 4.23 Comparaison de la latence entre JFFT8 et la FFT conventionnelle sur Virtex-5

Virtex5		Taille FFT	Taille données et TWF	DSP48	Slices	Délai (ns)	Freq. (MHz)	Latence (cycles)	Temps de transformée		Débit (MÉ/S)	Coût BPE	Coût FFT	MS/s/Slice	Latence – FFT		Gain en Latence (%)
									Cycles	Temps (µs)					Cycles	Temps (µs)	
Radix-8 Conv.	Cas 0	4096	16	108	5888	40,4	24,7	608	512	20,7	197,9	237964	121837	0,0336	1120	45,3	-
	Cas 1	4096	16	108	12224	3,6	277,7	620	512	1,8	2221,6	44019	22538	0,1817	1132	4,1	-
	Cas 2	4096	16	108	9604	5,5	181,7	616	512	2,8	1453,5	52860	27065	0,1513	1128	6,2	-
	Cas 3	4096	16	108	8092	6,7	150,0	616	512	3,4	1199,8	53957	27626	0,1483	1128	7,5	-
Radix-8 JFFT	Cas 0	4096	16	132	9764	27,2	36,7	608	512	13,9	293,9	265796	136087	0,0301	1120	30,5	32,6
	Cas 1	4096	16	88	9072	3,6	277,5	616	512	1,8	2220,4	32686	16735	0,2447	1128	4,1	0,3
	Cas 2	4096	16	88	8168	5,5	181,6	612	512	2,8	1452,7	44981	23030	0,1779	1124	6,2	0,3
	Cas 3	4096	16	88	8312	5,3	190,3	612	512	2,7	1522,6	43671	22360	0,1832	1124	5,9	21,5

* Estimation pour 4096 points

4.4.3 Synthèse d'évaluation et analyse

Les tableaux de 4.24 à 4.27 représentent la synthèse de l'évaluation de performances de l'architecture FFT proposée, conventionnelle et celle de l'article de référence [ZHO09] en termes de coût (MS/s/Slice), de latence, de temps de transformée et de multiplieurs câblés et DSP48 utilisés pour une FFT de 4096 points.

D'après le tableau 4.24, le coût est maximisé pour le cas 1 de la J8MDC pour la Spartan-3 et la Virtex-4. Par rapport à l'architecture FFT de référence R2^{2}SDF, le gain est de 370,3 % sur la Spartan-3 et de 231,95% sur la Virtex-4. Par contre, si on considère la comparaison entre notre architecture FFT proposée et conventionnelle, le meilleur coût est souligné par le cas 1 de la J8MDC pour la Spartan-3 et Virtex-5.

Tableau 4.24 Synthèse de l'évaluation du coût en MS/s/Slice d'une FFT de 4096 points

		Coût (MS/s/Slice)			
Méthode		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [09]		0,0283	0,0158	0,0870	-
R4SDC [09]		0,0234	0,0111	0,0596	-
Radix-8 Conv.	Cas 0	0,0098	0,0022	0,0219	0,0336
	Cas 1	0,0486	0,0229	0,2014	0,1817
	Cas 2	0,0408	0,0152	0,1583	0,1513
	Cas 3	0,0398	0,0139	0,1493	0,1483
Radix-8 JFFT	Cas 0	0,0064	0,0026	0,0272	0,0301
	Cas 1	0,1048	0,0259	0,2018	0,2447
	Cas 2	0,0859	0,0171	0,1466	0,1779
	Cas 3	0,1001	0,0200	0,1510	0,1832

Le tableau 4.25 ci-dessous, représente la synthèse de l'évaluation de la latence d'une FFT de 4096 points. On constate une augmentation de la vitesse sur toutes les familles de FPGA. Les meilleurs résultats sont effectués par l'architecture J8MDC cas1. On remarque une

diminution significative de la latence on comparant la latence de l'architecture FFT de références. Par contre, quand on compare le cas1 de la Radix-8 FFT conventionnelle avec celui de la JFFT, on constate qu'il y a une augmentation de vitesse que sur la Spartan-3.

Tableau 4.25 Synthèse de l'évaluation de la latence d'une FFT de 4096 points

		Latence (µs)			
Méthode		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [09]		86,2	86,4	34,8	-
R4SDC [09]		66,3	87,1	37,4	-
Radix-8 Conv.	Cas 0	101,6	154,7	57,0	45,3
	Cas 1	17,3	15,7	4,9	4,1
	Cas 2	23,7	23,0	7,5	6,2
	Cas 3	26,6	25,2	9,0	7,5
Radix-8 JFFT	Cas 0	98,1	104,8	38,9	30,5
	Cas 1	9,5	15,6	4,9	4,1
	Cas 2	12,8	22,9	7,5	6,2
	Cas 3	10,8	20,2	7,2	5,9

D'après le tableau 4.26, le meilleur résultat en termes de temps de calcul d'une FFT de 4096 points est encore effectué par le cas1 du Radix-8 JFFT. En comparant le cas 1 du Radix-8 JFFT par rapport à l'architecture FFT de référence $R2^2SDF$, le calcul de la FFT est 10 fois plus rapide pour le cas 1 du Radix-8 JFFT sur Spartan-3, 8 fois sur Virtex-4 et 6 fois sur Virtex-E.

Tableau 4.26 Synthèse de l'évaluation du temps de calcul d'une FFT de 4096 points

		Temps de calcul (µs)			
Méthode		Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [09]		43,0	43,1	17,4	-
R4SDC [09]		33,1	43,5	18,7	-
Radix-8 Conv.	Cas 0	46,4	70,7	26,0	20,7
	Cas 1	7,8	7,1	2,2	1,8
	Cas 2	10,8	10,5	3,4	2,8
	Cas 3	12,1	11,5	4,1	3,4
Radix-8 JFFT	Cas 0	44,9	47,9	17,8	13,9
	Cas 1	4,3	7,1	2,2	1,8
	Cas 2	5,8	10,4	3,4	2,8
	Cas 3	4,9	9,2	3,3	2,7

Selon le tableau 4.27, on constate que l'architecture FFT Radix-8 JFFT utilise le plus grand nombre de multiplieurs câblés sur la Spartan-3 et Virtex-E. Par contre, pour le Virtex-4, l'architecture Radix-8 conventionnelle utilise plus de DSP48.

On constate que malgré sa forte utilisation de multiplieurs câblés, l'architecture JFFT proposée dans le projet reste très performante vis-à-vis de l'architecture conventionnelle et des architectures de références.

Tableau 4.27 Synthèse de l'évaluation de l'utilisation des multiplieurs câblés ou DSP48 d'une FFT de 4096 points

	multiplieur cablés		DSP48	
Méthode	Spartan 3	Virtex E	Virtex 4	Virtex 5
R2^2SDF [09]	18,0	18,0	19,0	-
R4SDC [09]	18,0	18,0	19,0	-
Radix-8 Conv.	108,0	108,0	108,0	108,0
Radix-8 JFFT	132,0	132,0	88,0	88,0

Chapitre 5 Conclusion

Les opérateurs en télécommunication offrent de plus en plus de services de données (internet, visiophonie, courriel avec pièces jointes volumineuses, ...). La troisième génération de téléphonie mobile (3G+) a du mal à répondre aux besoins en termes de débits binaires actuels. C'est pour cette raison que les ingénieurs en télécommunication des industriels et des opérateurs sont entrain de concevoir le futur réseau de téléphonie mobile large bande 4G. Le projet actuel s'appelle LTE (*Long Term Evolution*) développé au sein de la norme 3GPP utilisera de nouvelles techniques de modulation, dont l'OFDM. L'OFDM a été choisie comme solution pour la modulation grâce à ces nombreux avantages dont les plus importants sont : la bonne exploitation de la bande passante ainsi que la diminution des interférences inter symbole. L'OFDM a pour élément clé un coprocesseur FFT/IFFT dans le transmetteur. Ce dernier se doit d'être très performant en termes de compromis vitesse de calcul et ressource d'implémentation puisque les données sont traitées en temps réel. Le défit actuel des ingénieurs en microélectronique, c'est de concevoir un coprocesseur FFT répondant au compromis surface silicium versus vitesse de calcul. L'architecture du coprocesseur représente donc l'élément principal de l'étude. Les

algorithmes FFT les plus utilisés en industrie sont le radix-2 et radix-4 conventionnels. L'architecture de leurs BPE facile à implémenter les privilégie par rapport à des algorithmes FFT de radix plus grand. Plus on utilise un grand radix, plus les opérations arithmétiques pour calculer une FFT de N points diminue en contre partie d'une architecture du BPE plus complexe. La motivation de l'utilisation d'un grand radix réside donc dans la diminution globale du nombre de multiplication et addition complexes par la diminution du nombre d'étages pour exécuter la FFT. De ce fait, une nouvelle formulation de la transformée discrète de Fourier a été développée au sein de notre laboratoire LSSI, nommée JFFT [JAB09]. Ceci afin de permettre l'utilisation d'une architecture des processeurs élémentaires (BPE – *Butterfly Processing Element*) avec radix plus grand tout en visant une augmentation des performances du coprocesseur FFT, à savoir, augmentation du débit et réduction de la latence en rapport aux ressources d'implémentation nécessaire et ceci pour différent radix.

L'étude présentée dans ce mémoire est une évaluation comparative de l'implémentation de coprocesseurs conventionnels et JFFT pour radix-2, radix-4 et radix-8 sur FPGA. Cette évaluation se base sur plusieurs critères : débit de la BPE, débit du coprocesseur, latence, fonction de performance globale et autre. Plusieurs coprocesseurs FFT conventionnels et JFFT ont été étudiés et évalués afin de générer des données comparatives. Les coprocesseurs FFT conventionnels sont utilisés comme référence de plancher puisqu'ils sont les plus communs. Par contre, les coprocesseurs FFT sur FPGA de l'article [ZHO09] sont considérés comme référence récente à surpasser grâce à leurs architectures améliorées. Les coprocesseurs JFFT ont été étudiés sur FPGA afin de connaître leur complexité et leurs performances. Ils représentent donc l'élément principal de ce travail. Leur performance a

dépassé nos attentes puisqu'ils offrent des résultats plus que satisfaisants; on obtient un gain de performance de 370,3 % sur la Spartan-3 et de 231,95% sur la Virtex-4 comparé à l'architecture FFT de référence R2^{2}SDF. Le calcul de la FFT est 10 fois plus rapide pour le cas 1 du Radix-8 JFFT sur Spartan-3, 8 fois sur Virtex-4 et 6 fois sur Virtex-E, comparé aux coprocesseurs FFT conventionnels et ceux de l'article de référence [ZHO09].

Références

[ACH10] Y. Achouri, 'Annexe: Implémentation efficace de la FFT pour des communications OFDM', Laboratoire des signaux et systèmes intégrés, Université du Québec à Trois-Rivières, 2010.

[BEL98] M. Bellanger, 'Traitement numérique du signal, Théorie et pratique', Dunod, Paris 1998.

[BUR81] C.S. Burrus et P.W. Eschenbacher, 'An In-Place, In-Order Prime Factor FFT Algorithm', Int. Conference on Acoustics, Speech and Signal Processing, Atlanta, Georgia, USA, March 30 - April 01, 1981.

[BUR08] C.S. Burrus, M. Frigo, S.G. Johnson, S.G. Johnson, M. Pueschel, I. Selesnick. 'Fast Fourier Transforms', Connexions Rice University, Houston, Texas, 2008.

[CHA08] W.H. Chang, T.Q. Nguyen, "On the Fixed-Point Accuracy Analysis of FFT Algorithms", IEEE Trans. On Signal Processing, vol. 56, no. 10, octobre 2008, pp. 4673-4682.

[COO65] J.W. Cooley and J.W. Tukey, "An Algorithm For The Machine Calculation Of Complex Fourier Series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[DOU02] D.L. Perry, 'VHDL: Programming by Example', 4ième édition.

[DUH84] P. Duhamel et H. Hollman, ' Split-Radix algorithms', Electronics Letters, Jan 5 1984.

[DUH90] P.Duhamel, M.Vetterli, 'Fast Fourier Transforms: A tutorial review and a state of the art', Signal Processing 19 p. 259-299, Elsevier, 1990.

[FAN06] C.-P. Fan, M.-S. Lee and G.-A. Su, 'A Low Multiplier and Multiplication Costs 256-point FFT Implementation with Simplified Radix-24 SDF Architecture', Asia Pacific Conference on Circuits and Systems, Singapore, 4-7 December 2006.

[FAZ03] K. Fazel, S. Kaiser, 'Multi-carrier and spread spectrum systems', John Wiley & Sons

[FRI07] A.Friedmann, 'Understanding OFDMA, the interface for 4G wireless', EE Times-India journal, page 1-3, 2007

[HEI86] M.T. Heideman H.V. Sorensen and C.S. Burrus, 'On computing the split-Radix', IEEE Transactions on Acoustics, Speech and Signal Processing, , volume 34, issue 2, Feb 1986.

[HSI09] Hsiang S. Hu, Hsiao-Y. Chen and Shyh-J. Jou, 'Novel FFT Processor with Parallel-In-Parallel-Out in Normal Order', VLSI Design Automation and Test Symposium, Page(s): 150 – 153, 2009.

[JAB08] M. Jaber, "Multitraitement parallèle fréquentiel pour l'annulation des interférences", examen doctoral, Université du Québec à Trois-Rivières, septembre 2008.

[JABM09] M. Jaber, 'Address Generator for the Fast Fourier Transform', US-6,993,547B2 and European patent application PCT/US01/07602.

[JAB09] M. Jaber, D. Massicotte. 'A New FFT Concept for Efficient VLSI Implementation: Butterfly Processing Element'. Int. Conference on Digital Signal Processing, July 5-7, Santorini, Greece 2009.

[JIA04] M. Jiang, B. Yang, Y. Fu, A. Jiang, Xin-a. Wang, X. Gan, B. Zhao, T. Zhang, 'Design of FFT processor with Low Power Complex Mutliplier for OFDM-based High-speed Wireless Applications', Int. Symposium on Communications and Information Technology, vol. 2, page 639, 2004.

[JON06] Douglas L. Jones,'Split-Radix FFT Algorithms', Connexion project, Version 1.5: Nov 2, 2006.

[JON07] Douglas L. Jones, 'Choosing the Best FFT Algorithm', connexions project, version 1.3. Feb 6, 2007.

[LIM05] M.-S. Lim, J.-y. Oh, 'Area and Power Efficient Pipeline FFT Algorithm', The IEEE Workshop on Signal Processing Systems, page 520, 2-4 Nov. 2005.

[LOU01] M. Pugel and L. Litwin, 'The principles of OFDM', RFdesign.com journal, page 30-48, janvier 2001.

[PRO96] John G. Proakis, Dimitris G.Manolakis, 'Digital Signal Processing, Principles, Algorithms, and applications', Third Edition, Prentice hall.

[QUA09] LTE Release 8 and beyond, QUALCOMM, September 2009.

[SAN07] S. Lee, Y.Jung, J.Kim, 'Low Complexity Pipeline FFT Processor For MIMO-OFDM Systems', Institute of Electronics, Information and Communication Engineers Electron Express, Vol. 4, No. 23, pp.750-754, 2007.

[SAN08] M. Santirini, 'FPGAs : Primed For a Prominent Role in the 4G Wireless Network', Xcell Journal, Xilinx, Inc, Issue 65, third quarter 2008

[SWA84] E.E. Swartzlander, Jr., Wendell K. W. Young, Saul J. Joseph, 'A Radix-4 Delay Commutator for Fast Fourier Transform Processor Implementation', IEEE Journal of Solid-State Circuits, Page(s): 702 – 709, 1984.

[TAN02] Y. Tang, Y. Jiang, and Y. Wang, ' Reduce FFT Memory Reference For Low Power Applications', International conference on Acoustics Speech and Signal Processing, Orlando, Florida, USA, May 13-17, 2002.

[TAN04] Y. Tang; Wang, Y. Lim, J.G. Chung, S. Song, 'High-Speed Assembly FFT Implementation with Memory Reference Reduction On DSP Processors', International Conference on Electronics, Circuits and Systems, December 13-15, Tel-Aviv, Israel, 2004.

[TZE09] T.Y. Sung, H.S. Hsin, Y.P. Cheng, 'Low-power and high-speed CORDIC-based split-Radix FFT processor for OFDM systems', Digital Signal Prosessing Volume 20, Issue 2, March 2010, Pages 511-527, elsevier

[VAN03] A.W.M. Van Den Enden, N.A.M.Verhoeckx, 'Traitement Numérique Du Signal' 3^{e} édition, Dunod 2003

[WAN07] Y. Wang, Y.Tang, Y. Jiang, J.G. Chung, S.S. Song and M.S. Lim, ' Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors', IEEE Transaction on Signal Processing , vol. 55, no 5, pp. 2338 – 2349, 2007.

[WID97] T. Widhe, "Efficient Implementation of FFT Processing Elements",Linköping studies in Science and Technology, Thesis No. 619, Linköping University, Sweden, June 1997.

[WIN75] S. Winograd, private communications, July 1975.

[YUN03] Y.Jung, H. Yoon, and J. Kim, 'New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Applications', International Conference on Consumer Electronics, Feb 2003, Hong Kong.

[ZHE94] R. Zheng, 'A new prime factor FFT algorithm and its index mapping', International Conference on Industrial Technology, Dec 5-9, Guangzhou , China 1994.

[ZHO09] B. Zhou, Y. Peng et D. Hwang, 'Pipeline FFT Architectures, Optimized for FPGAs', Hindawi Publishing Corporation, International Journal of Reconfigurable Computing, Volume 2009, 9 pages, 2009.