UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
SAMI AGREBI

IMPLÉMENTATION FPGA D'UNE FFT À BASE D'ARITHMÉTIQUE LOGARITHMIQUE POUR LES SYSTÈMES OFDM

AOÛT 2012

Résumé

Le marché actuel des télécommunications est en constante évolution, dans le but d'améliorer les normes actuelles des réseaux de la téléphonie mobile. Les défis sont d'améliorer l'efficacité spectrale pour un meilleur débit de transmission tout en présentant une robustesse du système aux perturbations. La modulation OFDM (*Orthogonal Frequency Division Multiplexing*) représente une des techniques les plus récemment adoptées dans les systèmes de communication large bande. Cette dernière démontre des performances élevées lorsqu'elle fait face à un environnement à trajets multiples. La transformée rapide de Fourier (FFT – *Fast Fourier Transform*) présente l'élément clé dans la défincition de l'OFDM en assurant l'orthogonalité des porteuses. L'exécution de la FFT en technologie d'intégration à très grande échelle (ITGE, VLSI – *Very Large Scale Integration*) 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 système des nombres logarithmique a été introduit en remplacement au système linéaire afin de réduire la taille des opérateurs complexes en virgule fixe tels que la multiplication et

la division. D'ailleurs, ses derniers sont parmi les contraintes d'implémentation matérielle exigent le plus de ressource.

Ce travail présente l'étude de la FFT, à base d'arithmétique logarithmique, dans un récepteur OFDM en vue d'une implémentation efficace dans une technologie cible FPGA. L'étude présentée dans ce mémoire est une évaluation comparative entre la FFT à arithmétique logarithmique (Log-FFT) et la FFT conventionnelle dans le contexte d'une transmission OFDM. Nous proposons l'utilisation de la base logarithmique supérieure à 2 afin de réduire les ressources en virgule fixe.La FFT conventionnelle et Log-FFT ont été simulées et implémentées dans le but de déterminer leurs performances. Les critères d'évaluation sont basés sur la perte de performance exprimée par le rapport signal sur bruit (SNR – *Signal to Noise Ratio*) et la consommation en nombre de cellules logiques (*Logic Slice*). Pour la recherche du meilleur compromis, nous avons appliqué une fonction de coût *J* qui nous a permis de démontrer que la Log-FFT affiche une réduction de 55% par rapport à la FFT conventionnelle. Comparativement à l'utilisation d'une FFT à virgule flottante, notre approche résulte d'une perte de SNR au récepteur OFDM de seulement 0.2dB pour un taux d'erreur binaire (BER – *Bit Error Rate*) de 1%. De plus, dans ces mêmes conditions notre approche procure une réduction des ressources matérielles de 10% par rapport à la FFT virgule fixe conventionnelle demandant une longueur binaire accrue pour atteindre la même perte de SNR.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

4G	4^{ième} Génération
ADSL	Asymmetric Digital Subscriber Line
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
BPE	Butterfly Processing Element
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
DAB	Digital Audio Broadcasting
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
DSP	Digital Signal Processing
DVB	Digital Video Broadcasting-Terrestrial
FDM	Frequency Division Multiplexing
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDTV	High Definition Television
HDSL	High-by-rate Digital Subscriber Line
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol Interference

LNS	Logarithmic Number System
LTE	Long Term Evolution
LUT	Look Up Table
NLF	Negative Logarithmic Function
MC-CDMA	Multi-Carrier Code Division Multiple Access
OFDM	Orthogonal Frequency Division Multiplexing
PAPR	Peak-to-Average Pwer Ratio
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
SNR	Signal Noise Ratio
SQNR	Signal Quantified Noise Ratio
TDMA	Time Division Multiple Access
VHSIC	Very High Speed Integrated Circuit
VHDSL	Very High-by-rate Digital Subscriber Line
VHDL	Very High Description Language
VLSI	Very Large Scale Integration
WLAN	Wireless Local Area Network

Chapitre 1 Introduction

À l'époque actuelle, le domaine des télécommunications est en constante évolution à travers certains secteurs émergents tels que la téléphonie mobile, la radiodiffusion numérique(DAB) et la diffusion de vidéo numérique (DVB). Le succès de ces derniers repose sur l'efficacité spectrale, la robustesse par rapport aux propagations multitrajets, la consommation en puissance et la complexité d'implémentation. Une des techniques modernes, qui a permis d'atteindre ces performances intéressantes, est la modulation multiporteuse OFDM[1] (*Orthogonal Frequency Division Multiplexing*, en français: MROF – Multiplexage par Répartition orthogonale de la Fréquence). Elle a suscité l'intérêt de plusieurs chercheurs pour ses avantages tels que son débit binaire élevé, son efficacité spectrale sans compromettre la largeur de bande et la diminution des interférences inter symbole (ISI). L'OFDM a vu le jour dans les laboratoires de l'armée américaine vers les années 60 sauf qu'elle était confidentielle. Ce n'est que 30 ans plus tard qu'elle est devenue

[1] Par mesure de commodité lorsque cela ce pose, nous utiliserons les abréviations techniques anglophones

accessible. D'autre part, la transformée de Fourier rapide (TFR, en anglais: FFT : *Fast Fourier Transform*), sur laquelle repose l'OFDM, est un algorithme efficace pour calculer la transformée de Fourier discrète (TFD, en anglais: DFT : *Discret Fourier Transform*). Son usage est fondamental en traitement numérique du signal, notamment pour les nouveaux systèmes de communication [WID97].

Les pionniers du premier algorithme de la TFR, en 1965, Cooley et Turkey deux ingénieurs, ont ouvert la voie aux autres chercheurs [COO65]. Dans la même période, Mitchell a innové une nouvelle méthode pour traiter les multiplications et les divisions à base de nombre logarithmique, afin de concevoir un système qui viendrait remplacer celui des nombres flottants et fixes [MIT62]. La naissance de la technologie ITGE (Intégration à Très Grande Échelle, en anglais: VLSI – *Very Large Scale Integration*) et son évolution ont permis l'implémentation de ces algorithmes.

L'intérêt de ce mémoire porte sur l'implémentation d'une FFT à base d'arithmétique logarithmique dans un système OFDM. La plateforme matérielle sollicitée est le FPGA (*Field Programmable Gate Array*, en français : MPP - *matrice de portes programmables*). Cette dernière est de plus en plus exploitée en télécommunication, aéronautique, transport, etc. Elle est également utilisée dans le prototypage rapide d'ASIC (*Application Specific Integrated Circuit*). Parmi les avantages notables du FPGA, on retrouve : mise en marché rapide, court temps de conception et de développement et les faibles frais non récurrents.

1.1 Problématique

La modulation OFDM représente une des techniques prometteuses dans le milieu de la télécommunication. Elle est utilisée dans plusieurs dispositifs à haut débit tels que le WLAN (*wireless local area* network, en français : RLSF - *réseau local sans fil*), la HDTV (*High Definition Television*, en français : TVHD – *Télévision Haute Définition*) et les systèmes de communication 4G (4e génération). L'élément clé de cette modulation est la FFT, elle est définie comme étant l'un des algorithmes le plus utilisés en traitement numérique du signal. Munie de plusieurs multiplications, la FFT représente un défi lors de l'implémentation. D'autre part, le LNS (*Logarithmic Number System*, en français : SNL - *Système des Nombre Logarithmique*) a été introduit, comme une alternative au système linéaire, pour ses opérateurs arithmétiques avantageux tels que la multiplication et la division. Ces derniers deviennent, respectivement, une simple sommation et soustraction. Cependant, les opérations de sommation et soustraction logarithmique sont plus complexes à concevoir. D'un point de vue matériel, le LNS représente un virage intéressant pour le calcul de la FFT. Par contre malgré la perte de précision des données, la quantification est indispensable dans une implémentation. Dans chaque réalisation de projet, un compromis est essentiel entre les principaux enjeux tels que : la vitesse d'exécution, la surface du circuit, la consommation et la précision.

1.2 Objectifs

L'objectif principal de ce projet est d'étudier l'arithmétique logarithmique comme moyen de réduction de complexité et de consommation de circuit pour un récepteur OFDM. Cette étude est réalisée dans la perspective d'effectuer une implémentation de cette méthode et l'évaluation des ressources matérielles en technologie FPGA. Pour ce faire, trois sous-objectifs peuvent être ressortis:

1. Étude de l'application de l'arithmétique logarithmique sur la FFT dans un contexte OFDM.
2. Étude des architectures d'une FFT et d'une Log-FFT radix-2 : avantages et techniques d'implémentation.
3. Programmation et implémentation de la FFT conventionnelle et des BPE (*Butterfly Processing Element*) conventionnelle et logarithmique sur Matlab et en VHDL pour évaluation des performances dans une cible FPGA.
4. Évaluer les méthodes en termes de meilleur compromis performance et complexité d'implémentation.

1.3 Méthodologie

L'algorithme conventionnel de la FFT radix-2 est la plus utilisé dans les projets de traitement numérique des signaux. Par ce fait, nous l'adoptons comme algorithme de référence tout au long de ce projet. Cependant, son étude est nécessaire pour bien comprendre le fonctionnement de la BPE.

Dans un premier temps, nous programmons la BPE conventionnelle sur Matlab®, ensuite nous validons son bon fonctionnement en comparant ses résultats avec la fonction préprogrammée. Matlab® représente un outil de calcul et de simulation puissant. Il sera dorénavant notre plateforme, pour toutes manipulations telles que la vérification, l'évaluation et l'implémentation de tout algorithme indispensable pour le projet. La première étape du projet consiste à étudier le LNS et l'arithmétique logarithmique afin de concevoir la Log-FFT, figure 1.1. Nous la programmons sur Matlab®, ensuite nous l'évaluons pour valider son bon fonctionnement. La deuxième étape se traduit par la conception de la Log-FFT et la vérification de son bon fonctionnement, en l'évaluant par rapport à la FFT de Matlab®. La troisième étape débute par la mise en application d'une plateforme OFDM qui sera notre environnement de simulation de la Log-FFT. La quatrième étape, l'intégration de la FFT et de la Log-FFT dans la plateforme OFDM et l'évaluation des résultats entre les deux approches.

Après l'obtention de résultats de simulation concluants, la programmation Matlab® est terminée, et l'étape de l'implémentation matérielle débute. Dans la dernière étape, nous implémentons la BPE conventionnelle et logarithmique en VHDL. L'outil d'implémentation et de simulation utilisé pour ce fait est Modelsim PE ® de Mentor Graphics®. Cet outil permet de faire le modèle VHDL et d'effectuer des simulations des circuits étudiés à l'aide de modules '*testbench*'. Ces derniers représentent l'environnement respectant les comportements attendus des entrées/sorties relatives au projet principal.

Par la suite, l'étape de validation entre les deux programmes simulés, sur les deux plateformes Matlab® et Modelsim®, s'effectue à travers une cosimulation. À l'aide de Matlab® nous générons un flux de donnée avec une amplitude variable dans un intervalle donné. Nous le quantifions en longueurs binaires à étudier, puis l'injectons dans les BPE des deux plateformes. Les résultats de sorties seront comparés afin de déterminer l'erreur en virgule fixe des deux plateformes. Une fois que le seuil d'erreur est respecté, nous validons la structure de la Log-FFT. Finalement, la synthèse et l'implémentation de la Log-FFT sur FPGA sont réalisées par le logiciel Xilinx ISE 12.1.

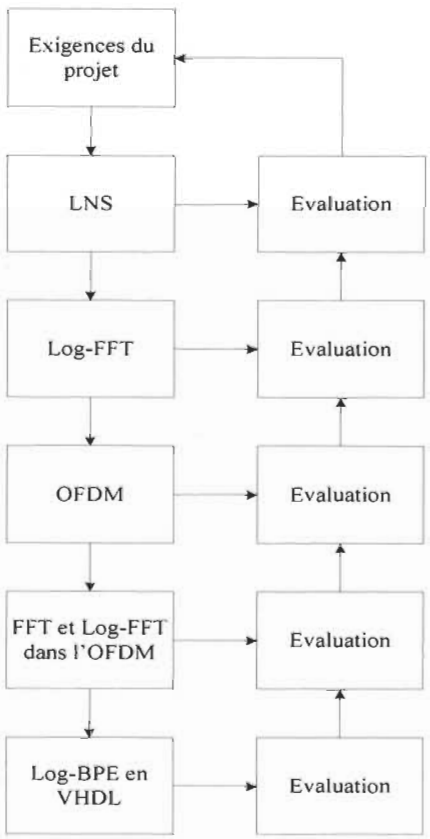


Figure 1.1 Modélisation de l'évaluation et la conception de la Log-FFT

L'étude de la transmission OFDM permettra d'approfondir les connaissances sur ce type de modulation, ainsi que son fonctionnement avec l'usage de la FFT au niveau du récepteur. Une cosimulation Matlab® / Modelsim® a été réalisée et se résume par l'utilisation de données générées par Matlab et injectées dans ModelSim afin de simuler le model VHDL des circuits étudié. La cosimulation a pour but de valider le bon fonctionnement des BPE afin de les implémenter sur FPGA. L'évaluation en ressources se réalise avec les outils de Xilinx.

1.4 Organisation du mémoire

Le reste du mémoire est structuré comme suit: le chapitre 2 introduit le concept de la communication OFDM et ses applications, ainsi que l'état de l'art de l'algorithme de la FFT et son implémentation en technologie ITGE. Ce chapitre présente également une définition de l'arithmétique logarithmique, ses architectures et circuits. Le chapitre 3 traite la FFT à base d'arithmétique logarithmique en virgule fixe. L'évaluation de la dynamique et la proposition d'une base logarithmique supérieure à 2 sont étudiées. Par la suite, nous introduisons la réduction des longueurs binaires et l'évaluation de la Log-FFT dans un récepteur OFDM. Le chapitre 4 décrit l'implémentation FPGA de la Log-FFT, les circuits pour l'arithmétique logarithmique et la Log-BPE radix-2, ensuite nous évaluons les ressources matérielles à l'aide de la synthèse et nous analysons la complexité des ressources. Enfin, le dernier chapitre fait l'objet de conclusions et des perspectives envisagées pour les travaux futures.

Chapitre 2
Circuits FFT pour l'OFDM

2.1 Communication OFDM

La transmission multi porteuse OFDM est une technique de modulation utilisée dans des systèmes audiovisuels tels que la radiodiffusion numérique (DAB) et la diffusion de vidéo numérique (DVB). Cette modulation a été très sollicitée ses dernières années et elle a été proposée dans plusieurs normes de télécommunications, à titre d'exemple, citons les réseaux locaux (WLAN – *Wireless Local Area Network*) IEEE 802.11 et les réseaux métropolitains (WMAN – *Wireless Metropolitan Area Network*), soit l'IEEE 802.16 et le LTE (*Long Term Evolution*).

Vers les années 50, les premiers dispositifs renfermant la modulation multiporteuse ont vu le jour dans les laboratoires militaires. Les premiers modèles d'OFDM ont été présentés par Chang en 1966 et Saltzberg en 1967 [NEE00]. L'utilisation réelle de la modulation multiporteuse était limitée et l'aspect pratique du concept a été remis en question.

Cependant, l'OFDM a été développé dans les travaux de Chang et Gibby en 1968 [CHA68], Weinstein et Ebert en 1971 [WEI71], Peled et Ruiz en 1980 [PEL80], et Hirosaki en 1981 [HIR80]. Ces derniers ont prouvé leurs capacités à créer le principe de modulation et démodulation du signal à l'aide d'une technique avancée en traitement numérique du signal, la FFT. Le choix de l'OFDM comme technique de transmission a pu être justifié par des études comparatives avec les systèmes d'une seule porteuse. Entre autres, un certain intérêt a été suscité pour la combinaison de la technique de transmission d'OFDM et l'accès multiple par répartition des codes (CDMA) dans les systèmes de canaux multiples MC-CDMA par Hara et Prasad en 1997 [MUH04]. L'OFDM est un domaine de recherche pertinent pour une mise en application dans les réseaux locaux sans fil et dans des applications de câble à bande large. La modulation multiporteuse a été adoptée comme technique de modulation pour les lignes d'abonnés numériques asymétriques (ADSL, 1.536 Mb/s), les lignes numériques de haut débit (HDSL, 1,6 Mb/s), les lignes d'abonnés numériques de haute vitesse (VHDSL, 100 Mb/s), la radiodiffusion numérique d'acoustique (DAB) et la radiodiffusion terrestre (HDTV).

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]. 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. L'orthogonalité permet également une haute efficacité spectrale, en effet la bande passante étant quasiment utilisée dans son intégralité. 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.

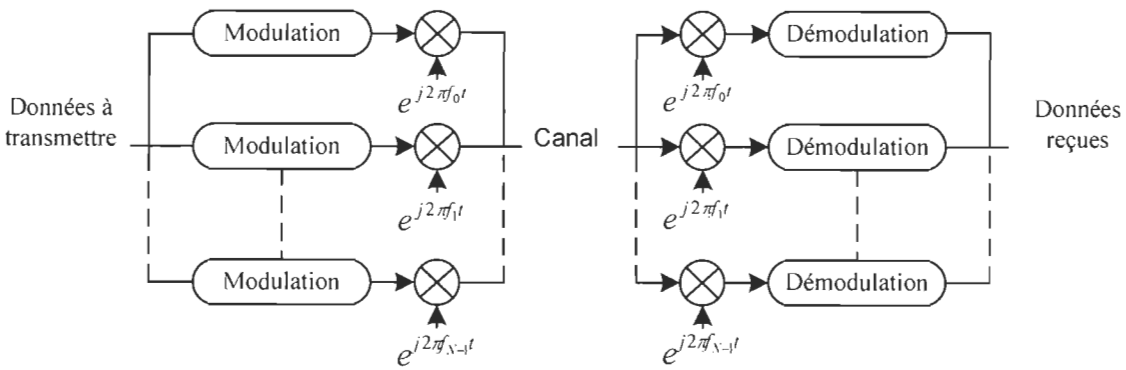


Figure 2.1 Modulation Multi-Porteuses

Dans un système OFDM [WID97] avec N sous-porteuses et un symbole de durée T, les fréquences de sous-porteuses, f_n, sont choisis 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 (2.1) et (2.2), le spectre de fréquence de *s(t)* consistera d'un chevauchement mutuel des sous-porteuses orthogonales, comme illustré à la figure 2.2.

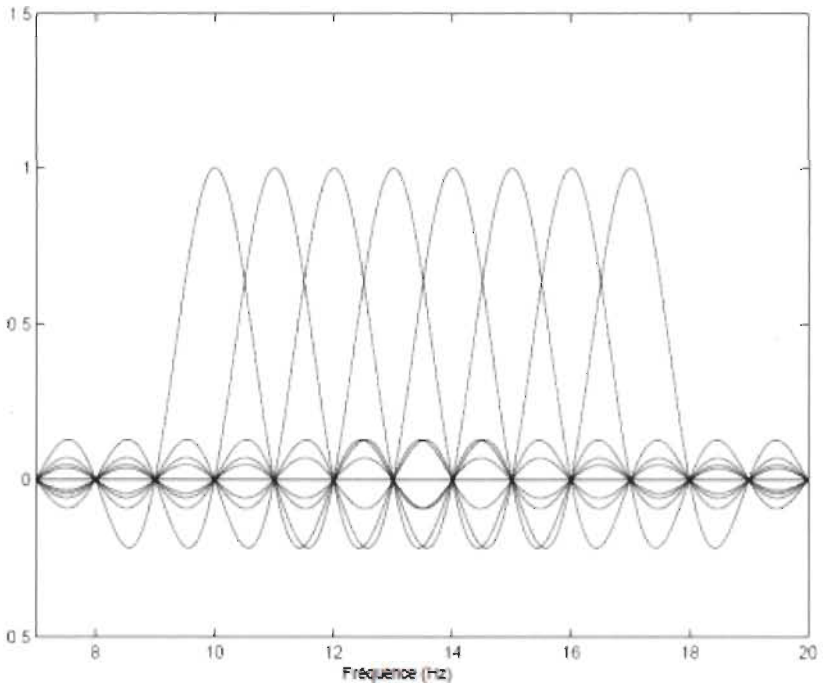


Figure 2.2 Spectre de sous-porteuses dans un système OFDM

À la réception, 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'estimation

de S_m est récupérée 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 idéales, $r(t) = s(t)$, l'estimé devient :

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

Considérant une orthogonalité idéale, nous avons alors:

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

2.1.2 Avantages et inconvénients de l'OFDM

Les techniques de modulation OFDM offrent plusieurs avantages par rapport aux méthodes de modulation conventionnelle à une seule porteuse. Un grand avantage de l'OFDM est sa simplicité due à l'utilisation d'IFFT (*Inverse Fast Fourier Transform*) [FEC99]. L'OFDM est basée sur la répartition de l'information à transmettre sur un grand nombre de sous porteuses orthogonales modulées à bas débit binaire de façon à rendre le canal radio mobile non dispersif en temps et non sélectif en fréquence. Cette technique réside dans le chevauchement du spectre des différentes sous porteuses produisant une efficacité spectrale

optimale. L'OFDM est une solution efficace au problème d'interférence inter-symboles (ISI) dans des canaux radio mobiles, grâce à l'insertion d'un intervalle de garde après chaque symbole utile [HEN02]. Cet intervalle a pour effet d'absorber l'effet multitrajets du canal et de maintenir la condition d'orthogonalité des sous porteuses au récepteur. La procédure de conservation d'orthogonalité pour l'OFDM est beaucoup plus simple comparée aux techniques de CDMA ou de TDMA (*Time Division Multiple Access*). L'OFDM peut être employée dans des applications à grande vitesse de multimédia avec un faible coût de service [NEE00]. La solution OFDM utilise d'une façon optimale la largeur de bande disponible, elle possède donc une haute efficacité spectrale [MUH04].

Malgré ses nombreux avantages, la modulation OFDM possède certains inconvénients, dont particulièrement celui du rapport de la puissance maximale et la puissance moyenne du signal OFDM, soit le problème du PAPR (*Peak-to-Average Power Ratio*) qui peut causer la saturation de l'amplificateur à l'émission et mener par la suite à une perte d'orthogonalité et de rayonnements hors de la bande passante. Un PAPR très élevé signifie que le signal possède une puissance maximale plus importante par rapport à sa puissance moyenne, et que des pics d'amplitude importante sont présents [MAY98]. Le rapport de PAPR est directement proportionnel au nombre de sous canaux utilisés dans le système OFDM. Par ailleurs, la modulation OFDM est extrêmement sensible aux erreurs de synchronisation qui conduisent à un TEB (Taux d'Erreur par Bit, en anglais: BER – *Bit Error Rate*) très élevé. Les méthodes existantes pour synchroniser le système OFDM sont basées sur l'utilisation des symboles pilotes ainsi que l'exploitation de la redondance du signal transmis [COX97]. Les symboles pilotes peuvent permettre une évaluation du canal à toutes les fréquences ou utiliser certains sous-canaux pour transmettre l'information

connue au récepteur. Ce dernier mesure l'atténuation subie par les porteuses pilotes à la réception et emploie cette information pour estimer l'atténuation des symboles de données aux autres sous-canaux.

2.1.3 Système à base de 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 suivante :

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

où $j^2 = -1$.

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 $t = kT / N$.

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

L'équation (2.8) est reconnue comme une IFFT 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'équation (2.9) :

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

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 $\{Sn\}$ de 0 à $N-1$ comme souhaité [WID97].

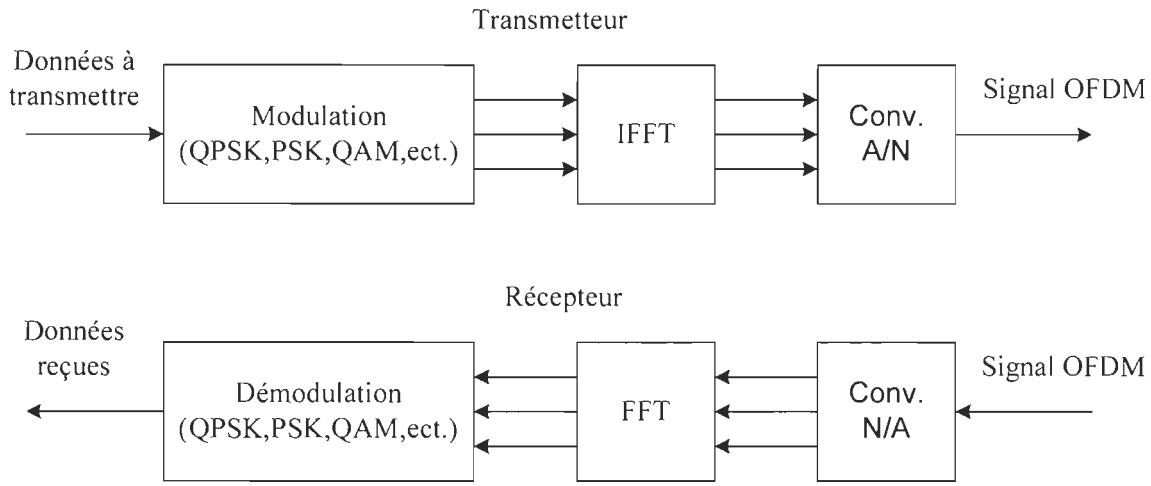


Figure 2.3 Système OFDM basé sur la FFT

La figure 2.3 représente un système OFDM basé sur la FFT. Le système OFDM, qui y figure, est 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 (Transformée de Fourier Discrète) et la TFDI (Transformées de Fourier Discrète Inverse), respectivement, mais plus efficace en terme de réduction de la complexité de calcul de sa mise en œuvre [LOU01].

À l'émission, l'IFFT est utilisée afin de réaliser le passage 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 bloc 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, multiples de la fondamentale. 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 des N sinusoïdes. 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].

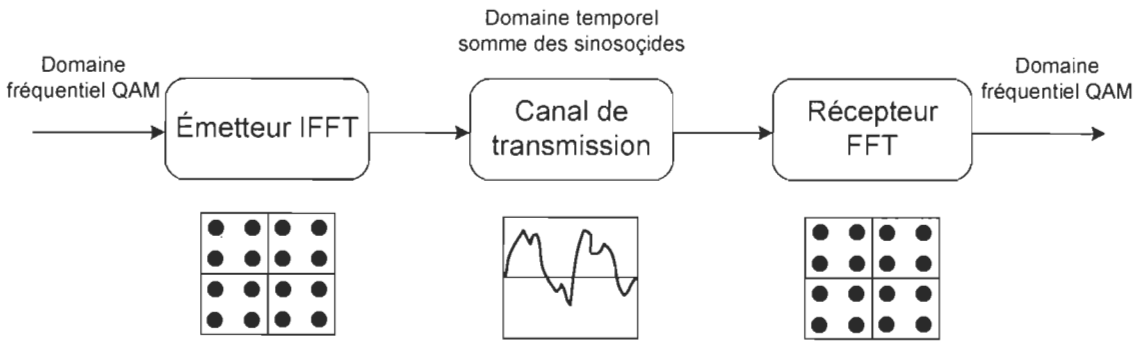


Figure 2.4 Passage du domaine fréquentiel au domaine temporel dans un système OFDM

2.2 Algorithme de la FFT

Pendant plusieurs décennies, la principale préoccupation des chercheurs était de développer un algorithme FFT qui minimise le nombre d'opérations requises pour calculer la transformée discrète de Fourier que nous rappelons ici

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

où $x_{[n]}$ est la séquence d'entrée, $X_{[k]}$ est la séquence de sortie, N est la longueur de la transformée, Les deux séquences $x_{[n]}$ et $X_{[k]}$ sont à valeur complexe.

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

Où W sont les coefficients de Fourier couramment appelé *twiddle factor*, figure 2.5, et $j^2 = -1$. Plusieurs propriétés caractéristiques de ces coefficients sont utilisées dans [VAN03], par exemple :

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

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

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

où * désigne le complexe conjugué.

Donc à partir de (2.10) et (2.14) nous pouvons déterminer la forme matricielle de la TFD suivante :

$$X_{[k]} = [B_N][x_n] \tag{2.15}$$

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

Les deux ingénieurs d'*IBM*, Cooley et Tukey, ont présenté leur approche en 1969, montrant que le nombre de multiplications requis pour calculer la TFD peut être réduit significativement en exploitant les propriétés de la figure 2.5. En reformulant les variantes des algorithmes FFT, l'intérêt a surgi à la fois pour chercher des applications pour cette puissante transformée et trouver diverses implémentations logicielles et matérielles pour la FFT.

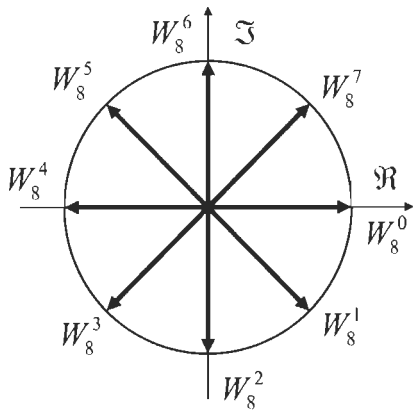


Figure 2.5 Affixes des coefficients de la TFD racine 8

Il existe des algorithmes FFT qui calculent la TFD plus rapidement et plus efficacement, tout en réduisant significativement les ressources et tailles des puces, où seront implémentés les FFT. L'exécution de ses algorithmes en technologie VLSI 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 (débit des résultats). D'où la nécessité d'augmenter les performances de ces algorithmes en trouvant une solution matérielle.

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

2.2.1 L'algorithme de la radix-2

L'algorithme FFT à base de radix-2 est l'un des algorithmes possédant la structure papillon (en anglais: *Butterfly*) la plus simple pour le calcul de la TFD. Radix-2 qui signifie en français radical-2 ou base-2, vu que la taille N de la FFT à calculer à base 2. L'appellation Butterfly est due au fait à son schéma en forme de papillon (voir figure 2.5). Dans la littérature, il existe deux algorithmes de la FFT: le DIT (*Decimation In Time*, en français: Décimation dans le temps) et le DIF (*Decimation In Frequency*, en français: Décimation en fréquence). 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 où N étant le nombre de points multiples de 2, 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 :

$$\left.\begin{aligned} x_1[n] &= x[2n] \\ x_2[n] &= x[2n+1] \end{aligned}\right\} \text{ avec } n = 0,1,\ldots,(N/2) - 1 \qquad (2.16)$$

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

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk} = \sum_{n=0\ (n\ pair)}^{N-1} x[n]W_N^{nk} = \sum_{n=0\ (n\ impair)}^{N-1} x[n]W_N^{nk}$$
$$= \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x[2n]W_N^{2nk} + \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x[2n+1]W_N^{(2n+1)k} \tag{2.17}$$

Et en utilisant (2.13) et (2.16), on trouve :

$$X[k] = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x_1[n]W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x_2[n]W_{N/2}^{nk}$$
$$= X_1[k] + W_N^k X_2[k] \tag{2.18}$$

où $X_1[k]$ 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 (2.18), 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 nombres complexes A et B, on calcule deux nouveaux nombres complexes Y et Z tels que :

$$Y = A + W_N^P B \quad \text{et} \quad Z = A - W_N^P B \tag{2.19}$$

avec W_N le coefficient de Fourier communément nommé *twiddle factor* et p un entier compris entre 0 et $N - 1$ afin de passer tous les échantillons. L'équation (2.19) est représentable schématiquement à la figure 2.5.

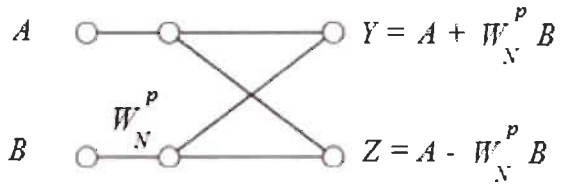


Figure 2.6 Structure DIT Butterfly

La figure 2.6 représente le schéma d'une structure Butterfly DIT de la FFT qui fera l'objet de l'implémentation dans la suite de ce travail. 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 TFD à 2 points. La figure 2.7 quant à elle, représente la structure d'une Butterfly DIF.

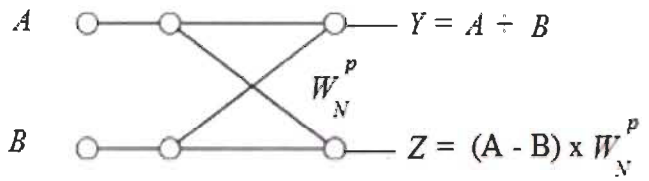


Figure 2.7 structure de la Butterfly DIF

Ci-dessus, les deux structures Butterfly utilisées respectivement pour le DIT et le DIF de la 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 (2.15).

Cette dernière peut être réécrite comme suit :

$$\mathbf{X} = \mathbf{B}_r \mathbf{x} \tag{2.20}$$

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

$$\mathbf{B}_r = \mathbf{W}_N^r \mathbf{T}_r \tag{2.21}$$

Et pour le DIF-FFT :

$$\mathbf{B}_r = \mathbf{T}_r \mathbf{W}_N^r \tag{2.22}$$

La matrice,

$$\mathbf{W}_N^r = diag\left(1, W_N^P, W_N^{2P}, ..., W_N^{(r-1)P}\right) \tag{2.23}$$

représente la matrice diagonale des *twiddle factor*, avec $p = 0, 1, ..., N/r^s-1$, $s = 0,1,...log_r N-1$ et $\mathbf{T}_r$ est une matrice l x m qui représente le *adder-tree* dans le Butterfly [WID97].

$$\mathbf{T}_r = \begin{bmatrix} w^0 & w^0 & w^0 & - & w^0 \\ w^0 & w^{N/r} & w^{2N/r} & - & w^{(r-1)N/r} \\ w^0 & w^{2N/r} & w^{4N/r} & - & w^{2(r-1)N/r} \\ - & - & - & - & - \\ w^0 & w^{(r-1)N/r} & - & - & w^{(r-1)^2 N/r} \end{bmatrix} = \left[\mathbf{T}_{r(l,m)}\right] \tag{2.24}$$

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

$$[T_r]_{l,m} = W_N^{\left[\left[\frac{lmN}{r}\right]\right]_N} \tag{2.25}$$

Avec $l = m = 0, ..., r-1$ et $\left[\left[\frac{lmN}{r}\right]\right]_N$ représente l'opération valeur absolue de $\frac{lmN}{r}$ modulo N

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

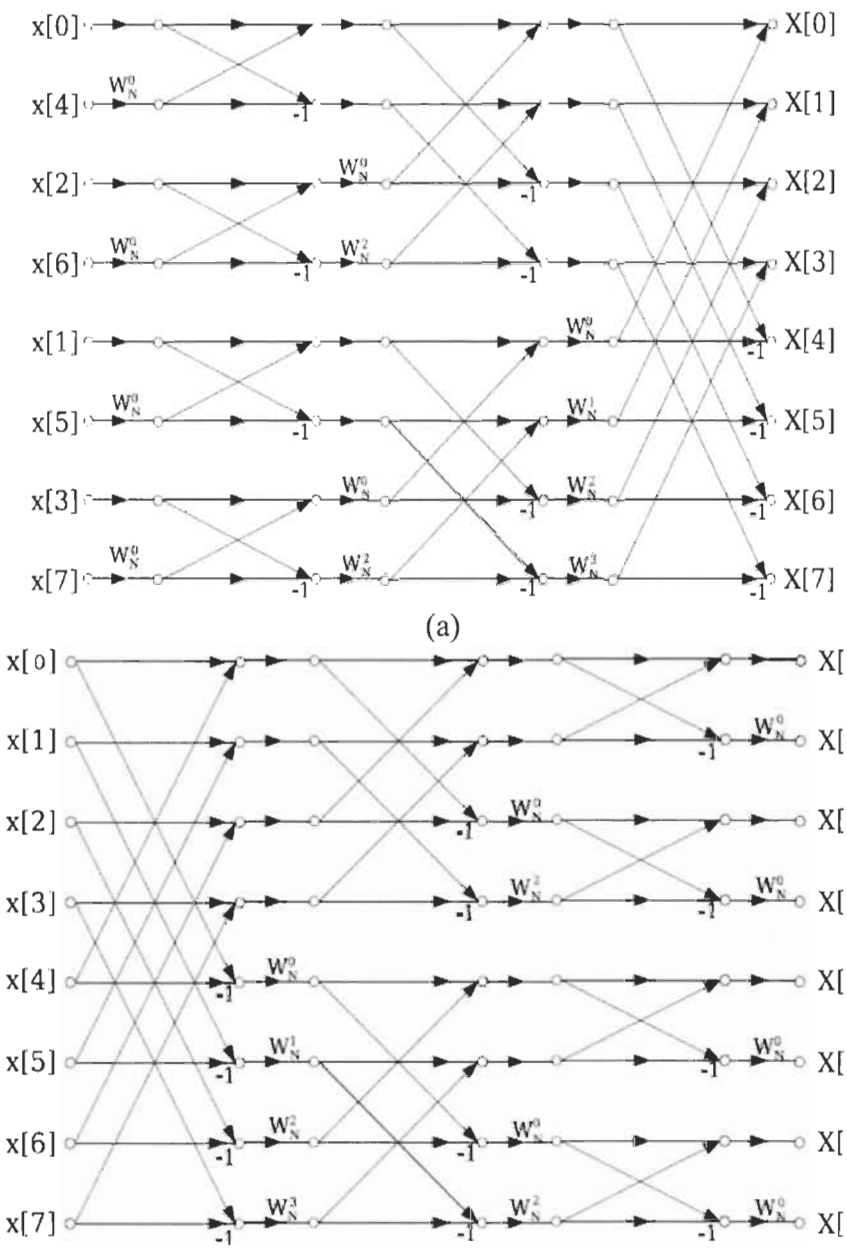


Figure 2.8 Diagramme de la FFT à base de Butterfly Radix-2 de type DIT (a) et DIF (b) pour N=8

Dans la figure 2.8, on représente la structure du Radix-2 DIT et DIF d'un FFT de longueur 8 points (N=8). On constate que la division en petit sous-système, nommé Butterfly, 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». Pour un Radix-2, la FFT se calcule en $\log_2 N$ étages (ou itérations), soit ici en trois étages 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, incluant les coefficients triviaux.

Le Radix-2 est l'algorithme FFT le plus utilisé actuellement. Il apparaît dans plusieurs articles [WAN07], [TAN02], [TAN04], grâce à son architecture simple. Plusieurs travaux sont effectués sur cet algorithme afin de trouver des méthodes plus élaborées pour réduire le nombre de multiplications et d'additions complexes. En effet, même si les structures de la figure 2.8 semblent simples, il n'en est rien au niveau de son implémentation en technologie VLSI à cause des accès mémoires et flots de données.

2.3 Arithmétique logarithmique

Depuis plusieurs années la recherche, dans le domaine du traitement numérique du signal, évolue afin d'améliorer et réduire certaines contraintes de la conception matérielle dans les plateformes VLSI telles que FPGA, ASIC, et DSP.

En général, les critères observés lors de la conception sont les ressources matérielles, le débit, la latence et la consommation électrique. Dans le cadre de ce projet, nous allons nous concentrer plus particulièrement aux ressources matérielles, car elles représentent le critère à améliorer tout en respectant les performances attendues de l'algorithme implémenté. Les ressources matérielles représentent les unités arithmétiques, les LUTs (*Look-Up Table*), les logiques combinatoires, les mémoires, etc; le débit est le nombre de résultats par seconde;

la latence est le temps d'exécution pour obtenir le premier résultat suite à l'application de la première donnée; enfin, la consommation électrique du circuit est mesurée en Watt. Le système des nombres logarithmiques a été introduit pour remplacer le système linéaire vu ses performances. Ses avantages sont très attrayants pour les algorithmes qui reposent sur des calculs complexes de longueur binaires élevés.

2.3.1 Système des nombres logarithmique (LNS – *Logarithm Number System*)

Le traitement numérique du signal est un domaine dans lequel il y a une multitude de manœuvres d'opérations arithmétiques. Certaines d'entre elles, tel que la multiplication, la division, la racine carrée, la puissance, etc.., sont couteuses en ressources matérielles. Le LNS permet, dans certaines situations, de tirer profit de ces opérations complexes.

Selon la norme IEEE de la représentation d'un nombre réel en virgule flottante est présentée à la figure 2.9. Nous avons conservé la nomenclature de IEE pour l'appellation de l'exposant E et la mantisse m, à ne pas confondre avec m utilisée à l'équation (2.26).

Signe S	**Exposent E**	**Mantisse m**
1 bit	E bits	m bits

Figure 2.9 IEEE 754 simples précisions en virgule flottante 32 bits

À l'aide des paramètres de la figure 2.9, le calcul de la valeur réelle d'un nombre binaire est effectué par l'équation (2.26).

$$N = -1^{S} \times 1.m \times 2^{E} \text{ et } 0 \le m < 1 \tag{2.26}$$

La figure 2.10 montre le système des nombres logarithmiques. Ce dernier possède une représentation semblable à un nombre réel.

Signe S	**Nombre L**	
1 bit	*i* bits entier	*f* bits de fraction

Figure 2.10 Représentation d'un nombre réel dans le LNS

Les équations (2.27) et (2.28) permettent le passage d'un nombre réel du domaine linéaire vers le domaine logarithmique ou réciproquement.

$$N = (1 - 2 \times S) \times 2^{L} \qquad \text{avec} \quad S = \begin{cases} 0 \text{ si } N \ge 0 \\ 1 \text{ si } N < 0 \end{cases} \tag{2.27}$$

$$\text{et} \qquad L = \begin{cases} log_2(N) \text{ si } |N| \ge \tau \\ log_2(\tau) \text{ si } |N| < \tau \end{cases} \tag{2.28}$$

Si nous analysons les équations (2.27) et (2.28), nous observons que le calcul du logarithme, d'un nombre réel N, se déroule sur deux étapes. Dans un premier temps, le signe de N est stocké dans S, ainsi S est un nombre binaire où 0 représente une valeur négative et 1 une valeur positive. Dans un deuxième temps, le logarithme L est calculé à partir de $|N|$. Cette approche est sollicitée pour se débarrasser des logarithmes complexes dus au nombre réel négatif. Le nombre τ est un seuil qui limite le logarithme L de tendre vers une valeur très négative, non représentable sur une longueur binaire ciblée. Une fois que ces paramètres sont fixés, nous pouvons commencer la conception.

Le modèle d'une application en LNS est représenté dans la figure 2.11. Les données d'entrée en nombre réelles sont converties en données logarithmiques à l'aide d'une Log-LUT. En fait, les *Look Up-Table* (LUT) sont des tables précalculées, qui renferment les valeurs logarithmiques d'une plage de données réelles. Après l'exécution de l'application, une Antilog-LUT est nécessaire pour ramener les résultats logarithmiques vers les résultats de sortie réels.

Lors de l'implémentation, les LUTs seront stockés dans des cellules mémoires du FPGA et permettront la conversion des données. D'ailleurs parmi les inconvénients du LNS, la taille et la largeur des LUTs peuvent devenir un handicap ressource et vitesse, étant donné qu'elles demandent des mémoires.

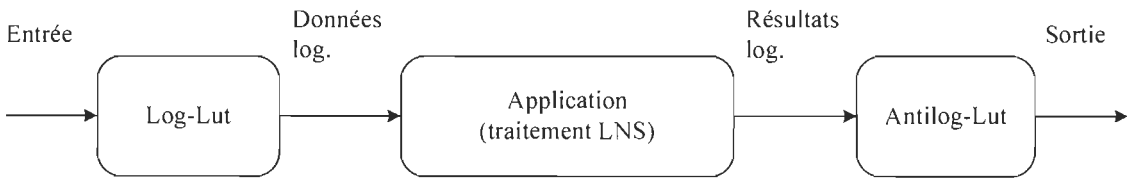


Figure 2.11 Application en LNS

2.4 Architecture pour arithmétiques logarithmiques

L'un des avantages de l'arithmétique logarithmique est la faible complexité des opérations coûteuses dans le système linaire. Pour expliquer l'exécution de l'arithmétique logarithmique, nous utiliserons des nombres réels A et B. Dans ce qui suit, nous utiliserons S_A pour le signe de A, S_B pour le signe de B, L_A pour le logarithme de A et L_B pour le logarithme de B. Le signe peut prendre les valeurs binaires 0 pour un nombre négatif et 1 pour un nombre positif.

2.4.1 Multiplication :

Le résultat de multiplication M est représenté comme suit :

$$M = A \times B \rightarrow \begin{cases} S_M = S_A \oplus S_B \\ L_M = L_A + L_B \end{cases} \tag{2.29}$$

Dans le système logarithmique, la multiplication de deux nombres est l'addition de ses deux logarithmes et son signe est un ou exclusif de ces deux entrées. S_M est le signe du résultat de la multiplication M et L_M son logarithme.

2.4.2 Division :

La division est représentée comme suit

$$D = A / B \rightarrow \begin{cases} S_D = S_A \oplus S_B \\ L_D = L_A - L_B \end{cases} \tag{2.30}$$

La division de deux nombres est la soustraction de ses deux logarithmes et son signe est un ou exclusif de ces deux entrées. S_D est le signe du résultat de la division D et L_D son logarithme.

2.4.3 Sommation / Soustraction

La sommation et la soustraction dans le système logarithmique représentent des opérateurs à complexité élevés, comparés aux autres opérateurs logarithmiques. Les étapes à suivre pour les réaliser de C=A+B sont lui suivantes :

À partir de la création d'un nombre temporaire L_T

$$L_T = L_A - L_B \tag{2.31}$$

nous définissons un signal de contrôle du signe

$$l_T = \begin{cases} 0 & \text{si } L_A \geq L_B \\ 1 & \text{si } L_A < L_B \end{cases} \tag{2.32}$$

En utilisant ce signal de contrôle binaire l_T et son complément $\overline{l_T}$, nous obtenons le signe du résultat comme suit :

$$S_C = l_T.S_B + \overline{l_T}.S_A \tag{2.33}$$

Pour obtenir le résultat de sommation ou soustraction, nous devons encore une fois se définir un signal de contrôle binaire *k* comme suit

$$k = S_A \oplus S_B \tag{2.34}$$

Ce signal de contrôle servira à sélectionner le résultat de sommation ou soustraction L_C comme suit :

$$L_C = \begin{cases} \max(L_A, L_B) + f_a(L_T) & \text{si } k = 0 \\ \max(L_A, L_B) + f_b(L_T) & \text{si } k = 1 \otimes |L_T| \geq \tau_1 \\ \tau_1 & \text{si } k = 1 \otimes |L_T| < \tau_1 \end{cases} \tag{2.35}$$

ou x représente l'opérateur binaire ET, fa et fb représentent deux fonctions logarithmiques défini comme suit

$$f_a(x) = \log_2\left(1 + 2^{-|x|}\right) \text{ , et } f_b(x) = \log_2\left(1 - 2^{-|x|}\right) \tag{2.36}$$

La sommation / soustraction logarithmique se traduit par une suite d'opération arithmétique. L'algorithme (2.31) repose sur une soustraction, une sommation et le calcul

de l'une des deux fonctions $f_a(x)$ *et* $f_b(x)$. D'ailleurs, la partie la plus complexe est le calcul des données $f_a(x)$ et $f_b(x)$. Cependant, dans certaines applications où la longueur binaire utilisée est inférieure à 8 bits, l'implémentation de ses tables est abordable.

Depuis quelques années, différents auteurs ont avancé plusieurs propositions qui touchent au système des nombres logarithmiques et son arithmétique. Mitchell [MIT62] a proposé une méthode qui traite les multiplications et les divisions, à base de nombres logarithmiques, pour les calculs effectués par ordinateur. L'approche conventionnelle utilisée, pour calculer ces derniers, était d'effectuer une série de sommations, soustractions et de décalage binaire. Le temps d'exécution étant très important, qu'effectué une simple addition ou soustraction, Mitchell a penché sa recherche sur l'utilisation de l'arithmétique logarithmique. L'un des inconvénients rencontrés était l'estimation du logarithme d'un nombre. Dans son époque, la mémoire d'un ordinateur était réduite ce qui rendait la tâche difficile d'incorporer des LUTs pour stocker des valeurs précalculées. Sa proposition était d'approximer le logarithme d'un nombre à partir de lui-même, d'un décalage et d'un compteur. La multiplication et la division nécessitaient une simple addition, soustraction et une opération de décalage.

L'inconvénient de la proposition de Mitchell est l'accumulation d'erreur d'approximation absolue avec seulement une précision de 3.5 bits. Ce qui lui faisait défaut, c'était la partie fractionnelle du nombre logarithmique.

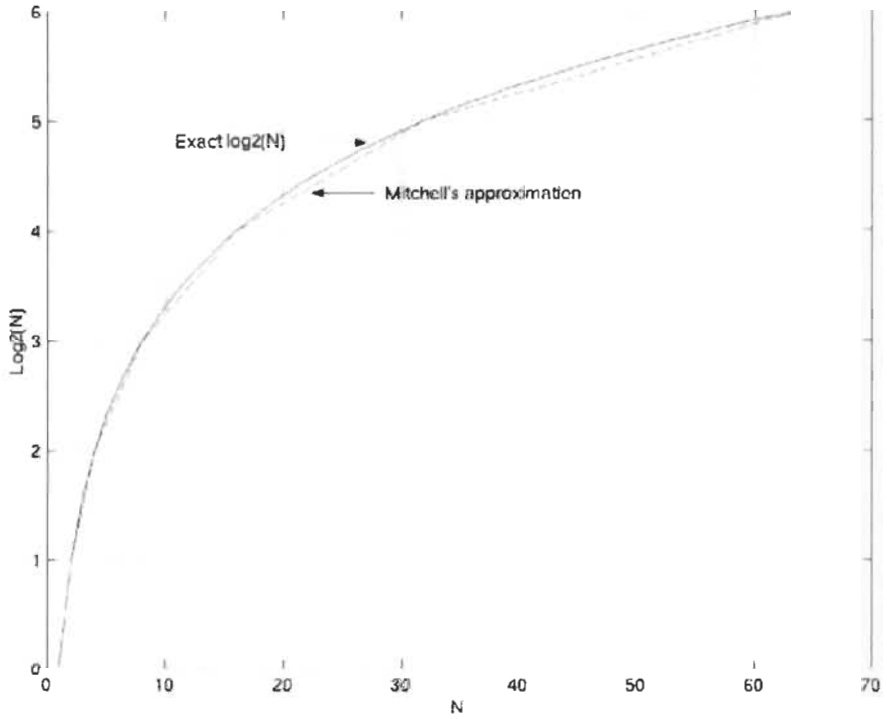


Figure 2.12 Approximation de Mitchell Vs Logarithme conventionnel [MIT62]

Dans la figure 2.12, les résultats trouvés par Mitchell montrent une bonne approximation des logarithmes sur une plage large de données. Certaines applications, en télécommunication par exemple, utilisent des fractions dans leurs traitements. La méthode de Mitchell constitue un handicap, ce qui a ouvert la porte à plusieurs autres auteurs pour développer d'autres techniques pour améliorer son approche.

Combet *et al.*, partie du principe de Mitchell, propose d'améliorer la précision du logarithme d'un nombre, en réduisant l'erreur d'approximation par un facteur de 6 [COM65]. Son principe est de diviser la plage de donnée réelle en quatre régions avec une approximation linéaire dans chacune d'entre elles. Les coefficients de linéarité sont mis à jour à l'aide de la réduction d'erreur résultante de l'approximation du logarithme. Cependant, les ressources matérielles et la latence de son circuit sont beaucoup moins

performantes que Mitchell, surtout si on divise par plus de régions pour une meilleure approximation.

Dans [HAL70], les auteurs ont développé un algorithme pour calculer approximativement le logarithme d'un nombre, la Sommation/Soustraction d'un logarithme et l'Antilogarithme. Ils divisent à leurs tours la plage de donnée en quatre sous-intervalles, ensuite ils choisissent l'approximation linéaire qui réduit l'erreur moyenne quadratique. Dans [SAN99], les auteurs ont proposé un autre type d'approximation des nombres entiers binaires. Leur contribution se résume à utiliser deux fonctions linéaires incorporées à l'aide de logique combinatoire, sans l'utilisation de multiplications. L'ajout de complexité pour leur algorithme n'est pas significatif par rapport à la méthode de Mitchell.

La proposition [ABE03] porte sur la conception d'un convertisseur 32bits binaire/binaire logarithmique seulement à base de logique combinatoire capable de calculer le logarithme d'un nombre en un cycle d'horloge. L'auteur a développé trois algorithmes pour ce fait. Les résultats, illustrés dans le Tableau 2.1, montrent la faible erreur d'approximation maximale en pourcentage après la conversion. Les trois propositions d'*Abed* sont comparées avec les autres auteurs cités auparavant.

Tableau 2.1 Comparaison des algorithmes d'approximation binaire du Logarithme d'un nombre de différents auteurs avec la méthode proposée par *Abed* [ABE03]

	Mitchell	SanGregory	Combet	Hall	Abed 2-regions	Abed 3-regions	Abed 6-regions
Regions de la mantisse	1	2	4	4	2	3	6
Bits de la mantisse	non	4	Tout les bits	Tout les bits	3	4	6
Max. Erreur de -ve(%)	0	-1.5403	-0.2669	-0.7282	-0.5544	-0.2684	-0.1529
Max. Erreur de +ve(%)	5.3605	0.4314	0.1849	0.1258	0.9299	0.4314	0.1538
Erreur absolue(%)	5.3605	1.9717	0.4518	0.9071	1.4843	0.6998	0.3067
Circuit de correction	non	simple	complexe	très complexe	simple	simple	simple

Le compromis à faire pour les différents auteurs se situe entre la complexité de l'implémentation et le nombre de bits de précision après l'approximation du logarithme.

Les chercheurs dans [LAY04] ont présenté une architecture, baptisée NLF (Negative Logarithmic Function), afin de calculer le logarithme et sa réciproque en utilisant peu de logiques combinatoires tout en obtenant de meilleurs résultats de précision. La NLF traite des nombres entiers, son équation est présentée en (2.36) :

$$f_{NLF}(x) = A_i - B_i \log_2(x+1) \tag{2.36}$$

Où A et B représentent deux coefficients positifs et i désigne la largeur binaire de la donnée d'entrée x. Pour la réalisation de cette fonction d'approximation au niveau matériel, *Layer* se sert des séries de Taylor / Mac Laurin du Logarithme présenté en (2.37) :

$$-\log_2(x+1) = \frac{1}{\ln 2}\sum_{n=1}^{\infty}(-1)^n.\frac{x^n}{n} \approx -\frac{n}{\ln 2} \tag{2.37}$$

Dans la figure 2.13, il a obtenu les résultats de l'erreur relative d'approximation en %. Cette dernière dépend de la largeur du mot binaire à l'entrée.

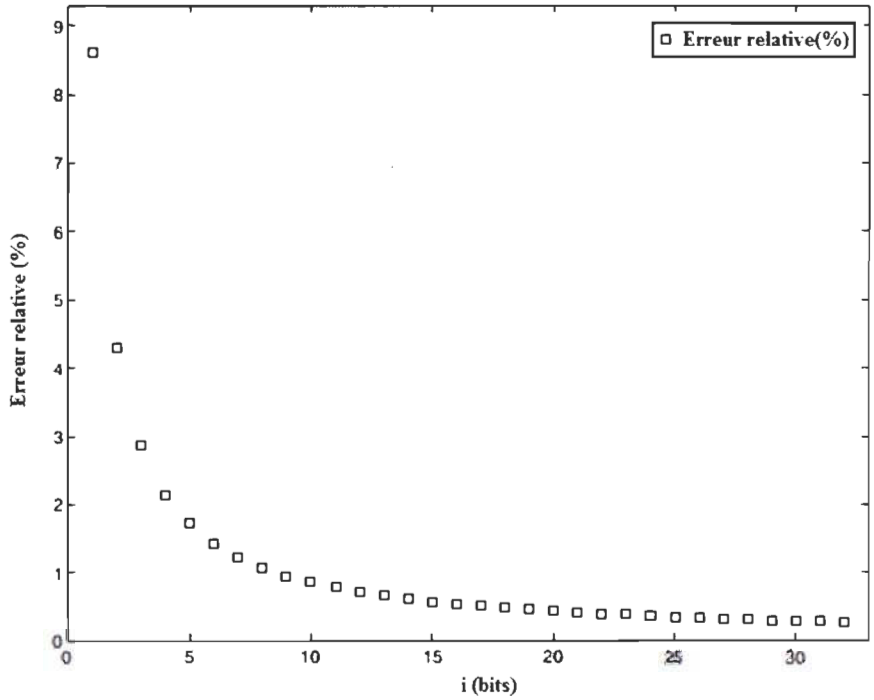


Figure 2.13 Erreur relative de l'approximation linaire logarithmique de *Layer* [LAY04]

Dans [MAH06], les auteurs ont proposé une nouvelle méthode pour améliorer la précision de l'algorithme de *Mitchell* dans le but d'effectuer une multiplication logarithmique. La proposition repose sur l'utilisation de la décomposition des opérandes. Cette technique permet de réduire les bits ayant '1' comme valeur dans les multiplicandes, ainsi que l'entité à approximer, de telle sorte que l'opérande d'entrée décomposé comporte moins de bits '1' que l'entrée originale. La décomposition de l'opérande réduit l'erreur de multiplication logarithmique de presque 45%. L'approche de *Mahalingam* n'apporte pas de correction à l'algorithme de *Mitchell*, par contre elle peut être jumelée avec un autre algorithme de correction proposée par les différents auteurs dans la littérature.

Brubaker est le premier à avoir utilisé les tables précalculées (Look-Up Table) du logarithme [BRU75]. De plus, il a étudié la multiplication logarithmique. Dans son

approche, l'erreur dépend de la largeur binaire utilisée pour les tables stockées dans la ROM (read only mémory). En effet, lorsque les deux entrées de la multiplication ont une largeur binaire de n bits, l'erreur maximale en % suit l'allure illustrée par la figure 2.14.

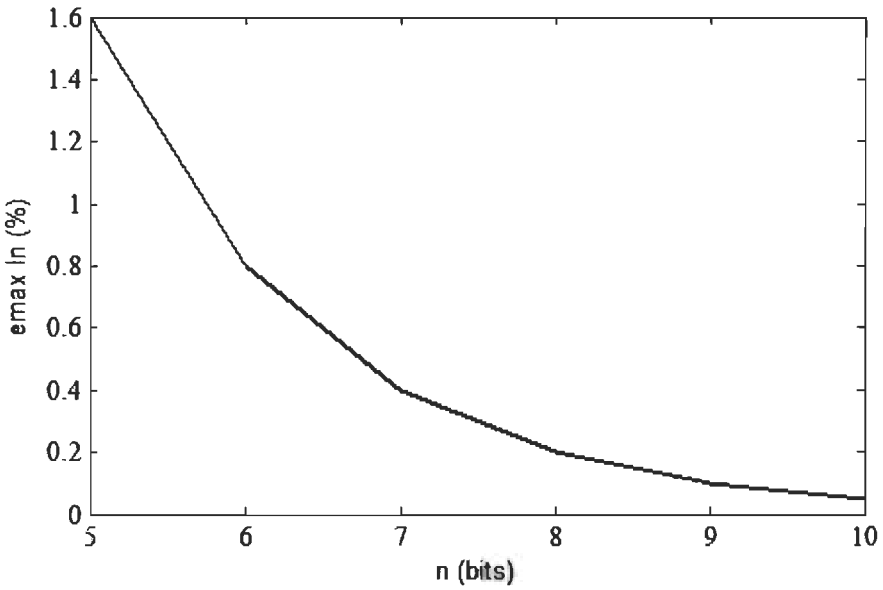


Figure 2.14 Erreur maximale de la multiplication logarithmique pour une longueur binaire n [BRU75].

Il s'est avéré que la taille et la largeur binaire des Lut utilisées, pour effectuer l'opération du Log et Antilog, jouent un rôle important sur l'erreur maximale accumulée.

Dans [TAN91], l'auteur a implémenté trois fonctions élémentaires : 2^x, $\log(x)$ et $\sin x$. Pour calculer le logarithme, il a utilisé une fonction d'approximation polynomiale sur plusieurs petits intervalles, avec $x \in [1,2]$.

Les deux auteurs dans [SCH99] ont amené une nouvelle proposition : la méthode des tables bipartites symétriques (SBTM). C'est une approche très rapide pour les fonctions d'approximation qui utilise des tables bipartites. Comparée aux autres méthodes de [BRU75] et [TAN91] qui utilisent le même principe, cette approche occupe moins de

mémoires, car les tables sont symétriques et l'une d'elles est menée par des zéros (zeros leading).

Dans [SUG09], **Suganth Paul** a proposé une méthode pour calculer le Log et l'Antilog d'un nombre binaire en utilisant la même architecture, lors de l'implémentation matérielle, qu'Arnold [ARN01] et Lewis [LEW89] sauf que ces deux derniers ne traitent que le Log(). L'approche repose sur l'utilisation de deux LUTs accompagnés d'une étape d'interpolation basée sur la logique combinatoire. Le nouvel aspect de sa technique c'est l'efficience de sa méthode d'interpolation qui se passe des multiplieurs et des diviseurs. L'avantage de sa proposition se situe au niveau du gain en ressources matériel, comparé aux autres propositions faites auparavant [BRU75], [LEW89], [TAN91] et [ARN01]. Le tableau 2.2 montre l'erreur et les bits de précision atteinte en fonction du nombre d'interpolations.

Tableau 2.2 Nombre de bits de précision pour une LUT de 64 nombres et de largeurs 16 bits [SUG09]

Nbr d'interplation	0	1	2	3	4	5	6	7	8
Erreur maximale x10(-3)	3.4147	1.7182	0.8834	0.4640	0.2538	0.1486	0.0959	0.0693	0.0564
Nbr de bits de précision	8.19	9.19	10.14	11.07	11.94	12.72	13.35	13.82	14.11

Comparée aux fonctions d'approximations par régions, l'utilisation des LUTs est avantageuse pour les applications qui nécessitent des entrées binaires avec plusieurs bits de fractions. En effet, dans l'application qui manipule les nombres logarithmiques, le compromis à faire se situe entre la taille et la largeur des LUTS versus le nombre de bits de précisions recherché.

Chapitre 3
FFT à base d'arithmétique logarithmique en virgule fixe

Les récentes avancées technologiques dans les systèmes sans fil requièrent une meilleure vitesse d'exécution, une réduction de surface et une faible consommation électrique.

Pour relever ce défi, certains chercheurs ont proposé des approches moins complexes du point de vue matériel tout en garantissant des performances acceptables. Dans la littérature, il existe plusieurs contributions qui ont mené à l'implémentation d'une FFT [WAN02], [ZHO09], [JIA04], [LIM05].

Étant un outil indispensable dans un récepteur OFDM, cette dernière a été exploitée avec l'arithmétique logarithmique dans le but de voir les avantages, les inconvénients et l'impact qu'elle pourrait avoir lors de l'implémentation.

Dans ce chapitre, nous allons explorer la Log-FFT dans un modèle simple ensuite dans un contexte de transmission OFDM.

3.1 Évaluation de la dynamique en virgule fixe

De nos jours, la mise en application d'un projet en microélectronique est sujette à plusieurs contraintes. Dans la plateforme FPGA, l'implémentation représente le fruit d'un compromis entre la surface occupée et la latence. Si nous nous penchons sur la première contrainte qui représente le critère principal de cette étude, nous remarquons qu'elle représente le défi actuel de plusieurs chercheurs [WAN03]. D'ailleurs, un des facteurs influents dans l'implémentation d'un projet est l'effet de quantification. Ce dernier joue un rôle important dans l'évaluation de la robustesse dans un environnement réel.

Pour ce faire, dans un premier temps, nous avons programmé la FFT radix-2 sous Matlab, et nous avons évalué la dynamique des données à travers les *Butterflies*. Ces dernières sont composées de 3 opérateurs arithmétiques complexes (une multiplication, une sommation et une soustraction). Nous avons sauvegardé tous les résultats obtenus, par ces opérations arithmétiques, dans des vecteurs et nous les avons observés. L'avantage de cette étape est de visualiser les amplitudes des données à prendre en considération pour la quantification.

La figure 3.1 illustre l'histogramme de l'opération de multiplication dans une FFT radix-2 de 64 points. Afin de réaliser ce tracé, nous avons considéré les conditions de simulation suivantes : La trame d'entrée de la FFT provient des données reçues au niveau du récepteur OFDM, celui dont le principe est affiché à la figure 2.3 mais le modèle complet est à la figure 3.8, cette plateforme est munie d'un générateur de donnée binaire, un modulateur 4 QAM, un canal sélectif en fréquence avec 6 possibilités de trajet suivi d'un générateur de

bruit ensuite la FFT et le démodulateur. Nous avons généré 6400 données que nous avons utilisées pour l'évaluation de la dynamique. Au niveau de la réception OFDM, nous avons observés que l'amplitude de la matrice de données reçue se situe entre +/- 6. Ces données ont nécessité l'utilisation de la FFT de 64 points 100 fois. Comme nous l'avons mentionné dans les objectifs, le but de ce projet est d'explorer la FFT dans un récepteur OFDM en appliquant l'arithmétique logarithmique.

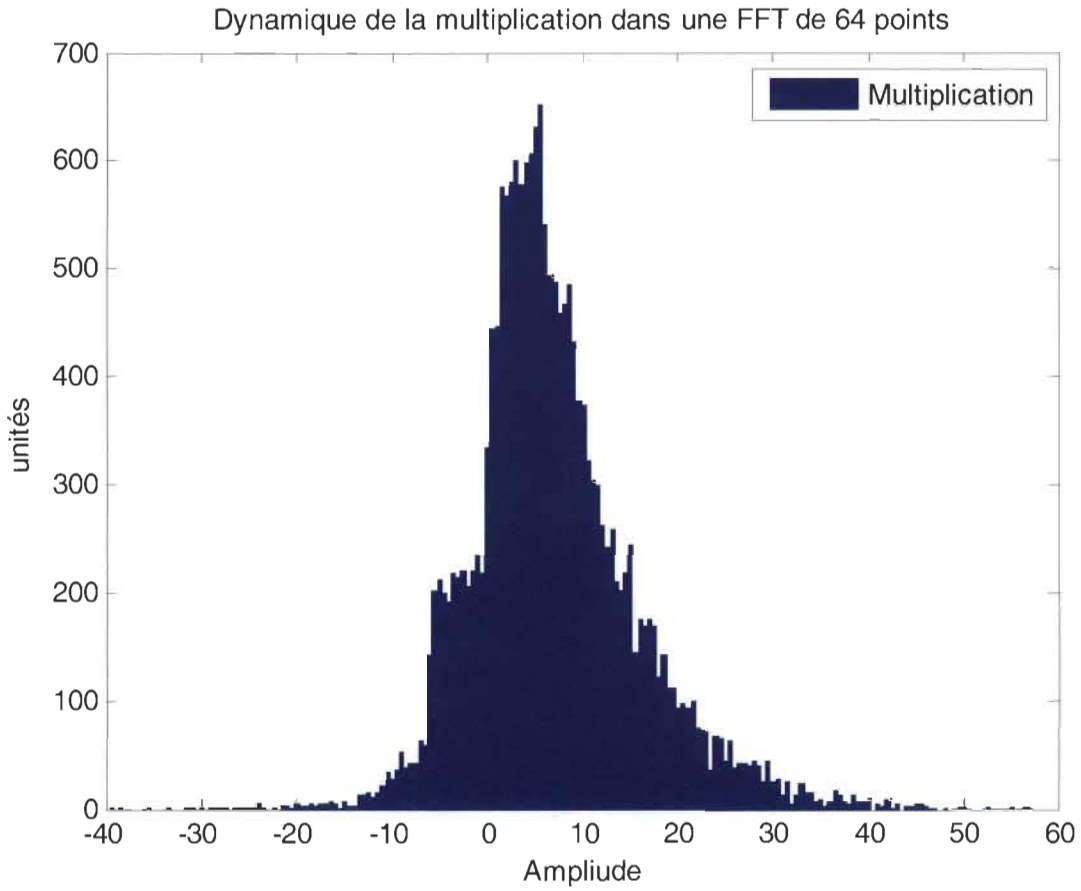


Figure 3.1 Dynamique de l'opération de multiplication pour une FFT radix-2 de 64 points

D'après la figure 3.1, nous constatons qu'une concentration importante des résultats obtenus après la première opération arithmétique de la FFT (la multiplication) présente des

amplitudes situées entre *-30* et *50*. Ces premiers résultats vont être réutilisés, par la suite, dans les deux opérations suivantes (la sommation et la soustraction) pour obtenir les résultats de sortie de la Butterfly, figure 3.2. D'après cette dernière, nous observons une augmentation en amplitude, par rapport à la multiplication. Les valeurs de sortie de la Butterfly varient entre *-60* et *80*. Cela là s'explique par l'addition ou la soustraction du résultat de la multiplication avec d'autres valeurs réelles.

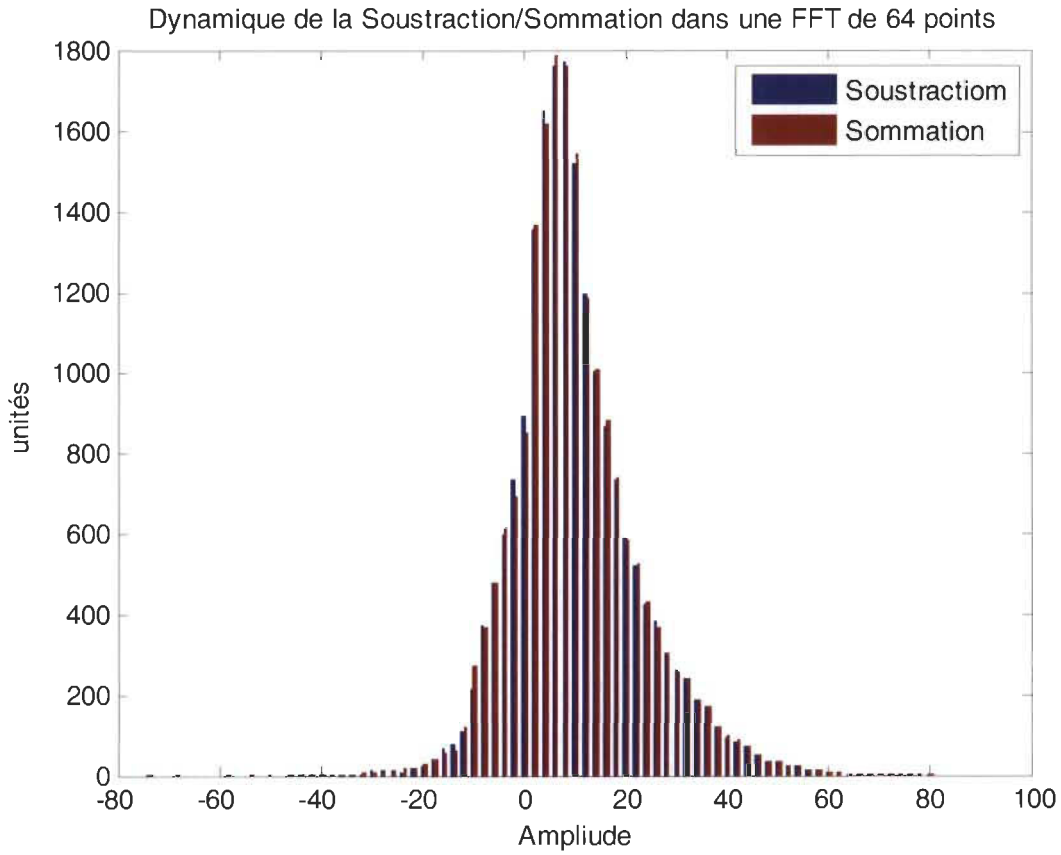


Figure 3.2 Dynamique des opérations soustraction/sommation pour une FFT radix-2 de 64 points

À partir de ces deux histogrammes, nous pouvons avoir une idée précise sur la variation de la dynamique à l'intérieur des Butterflies. Cela nous permet d'aller chercher une meilleure précision lors de la quantification. D'ailleurs, grâce aux figures 3.1 et 3.2 nous pouvons estimer que la longueur binaire minimale, pour contenir la précision nécessaire lors des prochaines simulations, est de 7 bits (6 bits entiers et 1 bit de signe). Les bits de fractions restent à déterminer en fonction des paramètres de la plateforme OFDM.

Cependant, ses histogrammes restent crédibles pour une FFT *radix-2* de 64 points avec un vecteur d'entrée qui varie entre ±6. Une variation du nombre de points de la FFT ou du vecteur d'entrée (comme dans le cas d'un récepteur OFDM) affectera considérablement les résultats de la sortie. L'accumulation due à une FFT supérieure à 64 points fait varier considérablement l'amplitude des données à l'intérieur des multiplieurs et des sommateurs/soustracteurs. Le même principe est observé lorsqu'on change l'intervalle de variation des données d'entrées.

À titre d'exemple, nous avons effectué le même exercice pour une FFT radix-2 de 2048 points (voir figure 3.3 et 3.4).

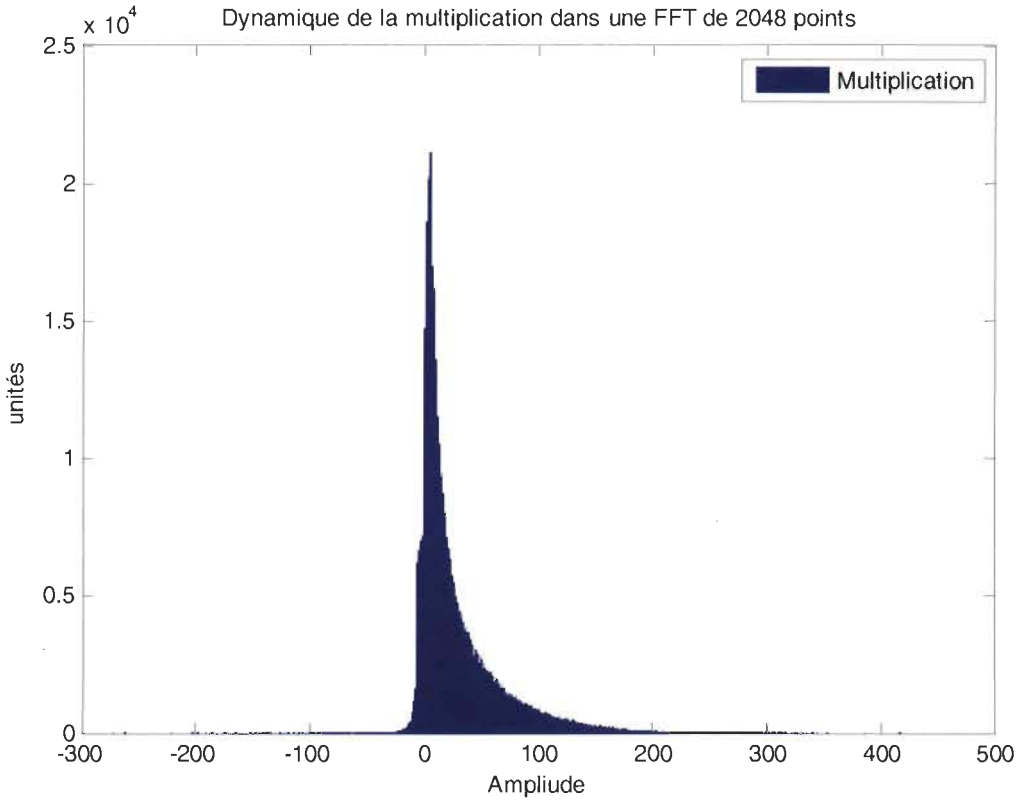


Figure 3.3 Dynamique de l'opération de multiplication pour une FFT radix-2 de 2048 points

Dans le cas d'un système OFDM, le choix d'utiliser une FFT avec un nombre de points élevés (*i.g* 512, 1024 ou 2048) servira à améliorer le débit de transmission. Par contre, la dynamique à l'intérieur de la FFT va être plus large telle que présentée à la figure 3.4 (les données qui ont géré ces résultats sont entre ±6)

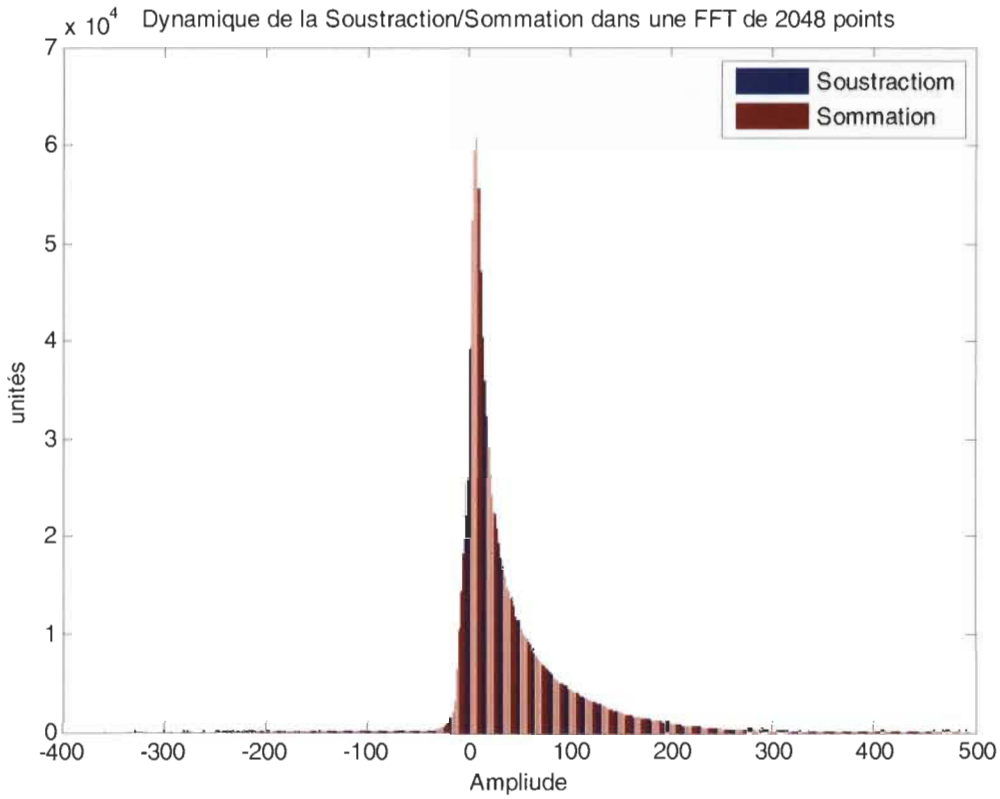


Figure 3.4 Dynamique des opérations de soustraction/sommation pour une FFT radix-2 de 2048 points

À partir des figures 3.4 et 3.3, nous pouvons affirmer que pour une FFT *radix-2* 2048 points il faudrait utiliser une longueur binaire de 10 bits (9 bits entiers et 1 bit de signe). En conclusion, l'étude de la variation de la dynamique à l'intérieur d'une FFT permet d'établir la longueur binaire minimale nécessaire pour l'effet de quantification afin de limiter les pertes de précision. Par ailleurs, les facteurs influents dans cette étude sont le nombre de points de la FFT et l'amplitude du vecteur d'entrée.

3.2 Proposition d'une base logarithmique supérieure à 2

Le système des nombres logarithmiques a été introduit en remplacement au système linéaire pour ses avantages alléchants au niveau de l'implémentation. Les opérateurs arithmétiques logarithmiques simples (la sommation et la soustraction) viennent en substitution à la multiplication et la division du domaine linéaire afin de réduire la complexité. Dans notre étude, nous avons programmé la FFT radix-2 avec l'approche linéaire et logarithmique, et nous l'avons exposé dans un environnement qui nous permettra d'estimer sa robustesse.

Pour ce faire, nous avons conçu, à la figure 3.5, le modèle suivant :

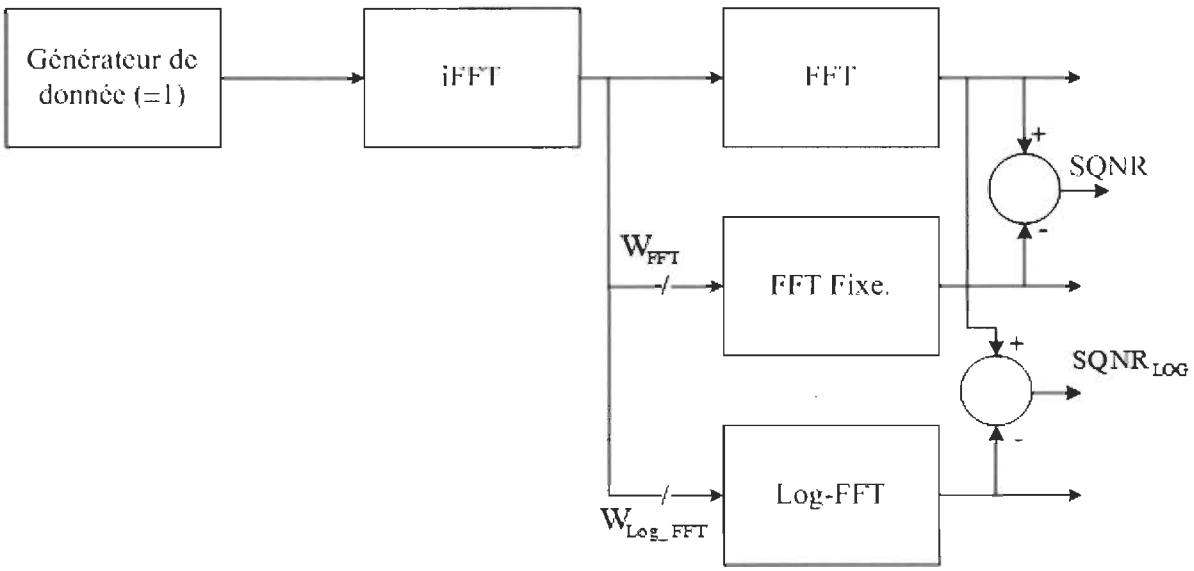


Figure 3.5 Modèle simple d'un système de communication à base de la FFT

Le système suivant permet de générer des données aléatoires, on applique l'iFFT (l'inverse de la FFT) ensuite la FFT afin d'avoir une estimation des données générées. Ce modèle représente un système de communication simple. Si nous regardons plus en détail les différentes FFT représentées, nous remarquons que la première FFT est la FFT standard de Matlab, une fonction préprogrammée qui requiert comme paramètre le nombre de points et le vecteur d'entrée. La FFT fixe représente une FFT quantifiée et enfin la Log-FFT désigne la fonction que nous avons créée à l'aide des opérateurs arithmétiques logarithmiques en virgule fixe. Afin d'étudier la robustesse de cette dernière, nous avons opté pour la méthode de comparaison du SQNR [WEI08]. La méthode du SQNR est décrite par l'équation (3.1) :

$$SQNR = 10\log_{10}\left(\frac{\left|Sortie_{idéale}\right|^2}{\left|Sortie_{idéale} - Sortie_{iestimée}\right|^2}\right) \tag{3.1}$$

La figure 3.6 représente le SQNR obtenu pour une FFT conventionnelle et une Log-FFT à base 2, 3 et 4. Dans cette simulation, nous avons utilisé plusieurs longueurs binaires (8 bits à 14 bits) lors de la quantification des FFT afin d'obtenir des résultats qui varient entre *17 dB* et *60 dB*.

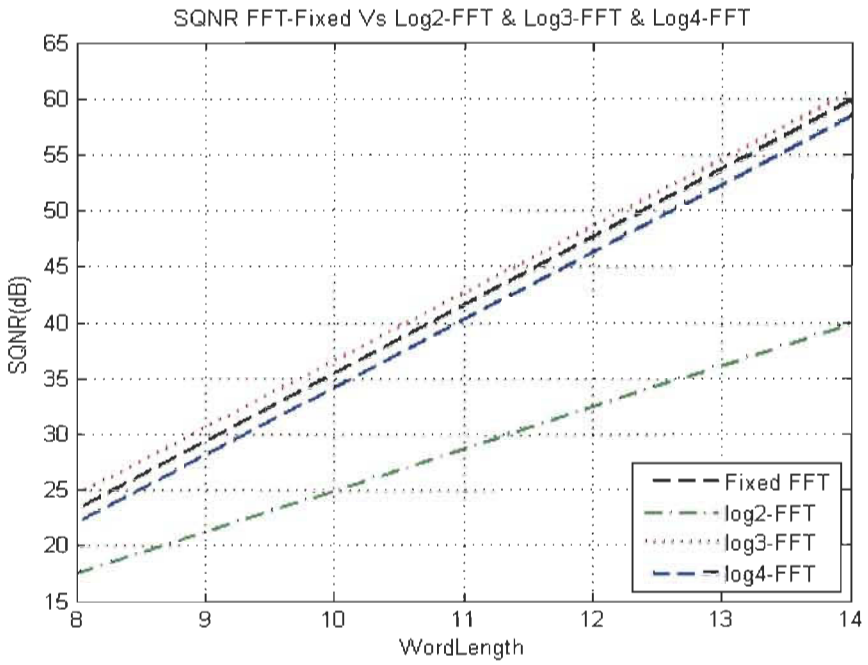


Figure 3.6 SQNR pour une FFT conventionnelle et une Log-FFT à base 2-3-4

D'après la figure 3.6, nous constatons que le SQNR de la FFT conventionnelle, situé entre celui de la Log-FFT à base 4 et à base 3, révèle une perte de *1 dB* en faveur de la Log-FFT à base 3.

Cependant, nous constatons également que la variation de la base du logarithme a un impact considérable sur les résultats obtenus. Plus particulièrement le passage de la base 2 à la base 3 signale un gain de *8* à *20 dB* en fonction de la longueur binaire. Par contre, une base supérieure à 3 implique des résultats moins performants comparés aux résultats de la FFT en virgule fixe.

3.3 Réduction du nombre de bits de fraction

La différence entre les différents résultats de la Log-FFT (base 2, 3 et 4) est située au niveau des conversions des données du domaine linéaire vers le domaine logarithmique,

plus exactement au niveau de la Log-Lut et l'AntiLog-Lut. D'ailleurs, si nous observons cette conversion en détail, figure 3.7, nous notons que plus la base augmente, plus la dynamique à l'intérieure des LUTs est réduite.

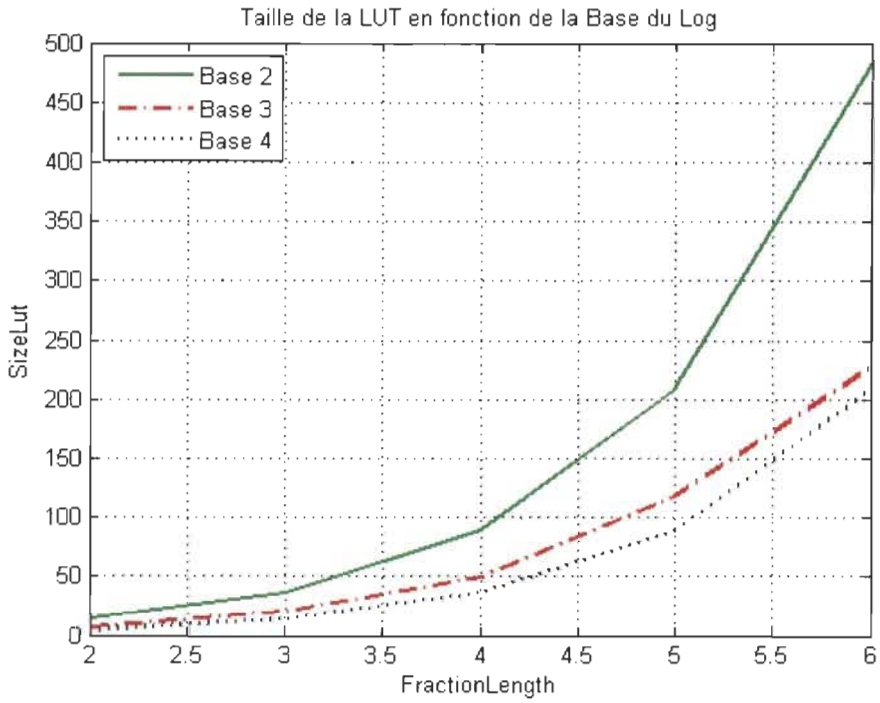


Figure 3.7 Taille de la Log-Lut en fonction de la base du logarithme

Or, il existe une base qu'on appellera « optimum » qui permet d'avoir de meilleurs résultats. Figure 3.6. Exemple : dans nos résultats du SQNR, nous observons que la base 3 possède de meilleurs résultats que la base 4.

En résumé, le nombre de bits de fraction dans la figure 3.7 définit la précision visée. Les LUTs sont influencés généralement par ce paramètre ainsi que la base du logarithme. Nous pouvons constater que, malgré le nombre de bits de fraction allouée pendant la quantification, le choix d'une base de logarithme supérieure à 2 implique une réduction de

la dynamique pendant la conversion des données. De ce fait, l'erreur de précision est moins significative lors du calcul des points de la Log-FFT.

3.4 Évaluation d'un récepteur OFDM en arithmétique logarithmique

Le système OFDM parait de plus en plus incontournable dans les nouveaux standards de communication mobiles de la prochaine génération. La technique multiporteuse permet de résister aux trajets multiples et d'aspirer à des débits élevés. Le système représenté à la figure 3.8 a été créé et utilisé pour intégrer l'arithmétique logarithmique.

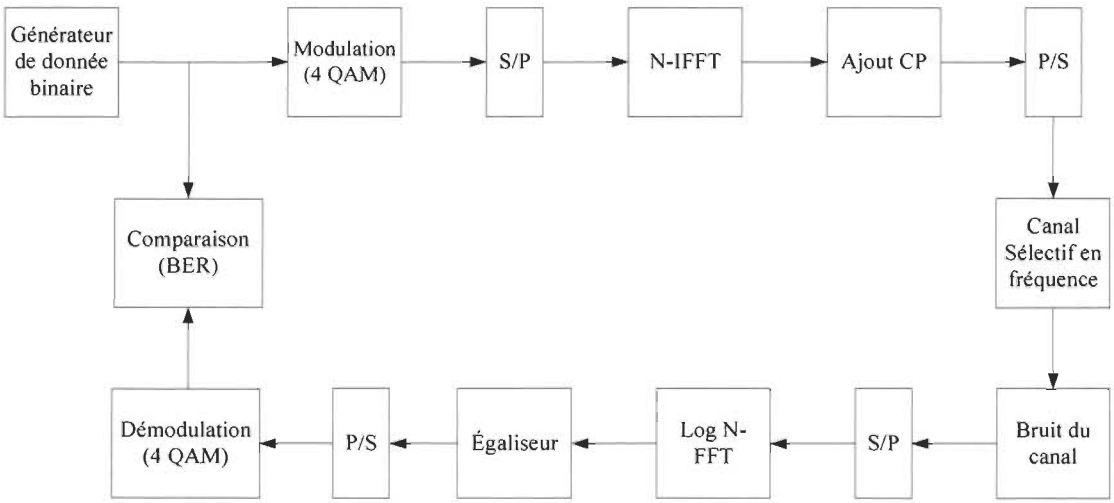


Figure 3.8 Plateforme OFDM

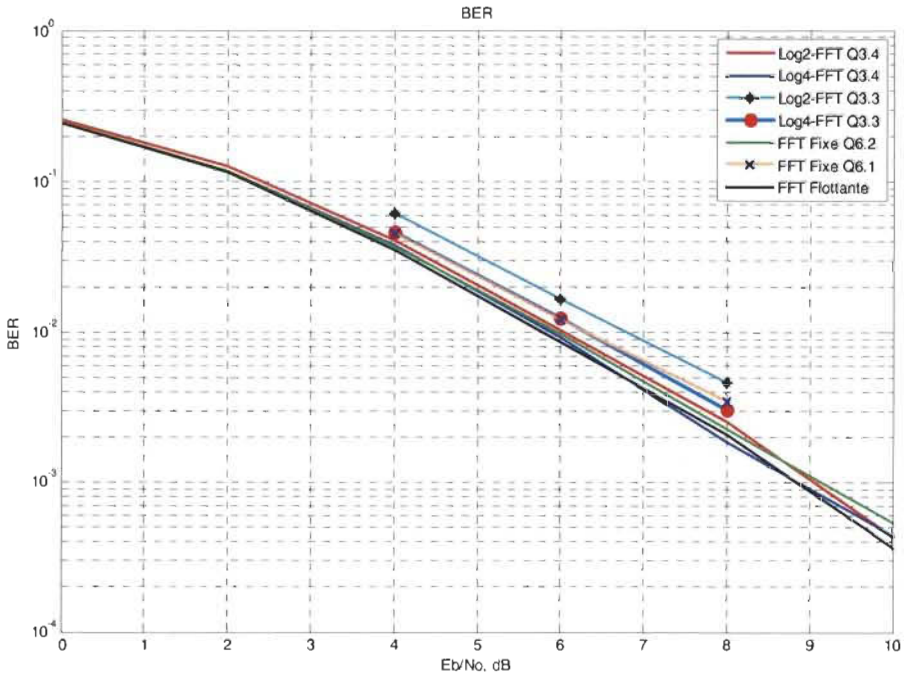
Le modèle de la plateforme OFDM est représenté par deux blocs reliés par un canal de transmission. Dans le premier bloc, le générateur de données permet de créer des symboles OFDM composés par des trames binaires. Ces derniers sont modulés en 4 QAM, multiplexés pendant leur passage dans l'IFFT et suivi par un ajout de préfixe cyclique pour être envoyés vers le récepteur. Pendant le passage dans le canal à trajets multiples, du bruit additif s'accumule et affecte le signal. Au niveau du deuxième bloc, la Log-FFT convertit

les données du domaine fréquentiel au domaine temporel, l'égaliseur permet de corriger partiellement le signal modulé et le démodulateur récupère une estimation du signal. La méthode utilisée pour la comparaison est le taux d'erreur binaire (BER). Pour les simulations, nous avons utilisé une Log-FFT de 64 points, avec un canal sélectif en fréquence de 6 possibilités de trajets avec un bruit additif, dont le niveau signal sur bruit (SNR) variant entre *0* et *10 dB*.

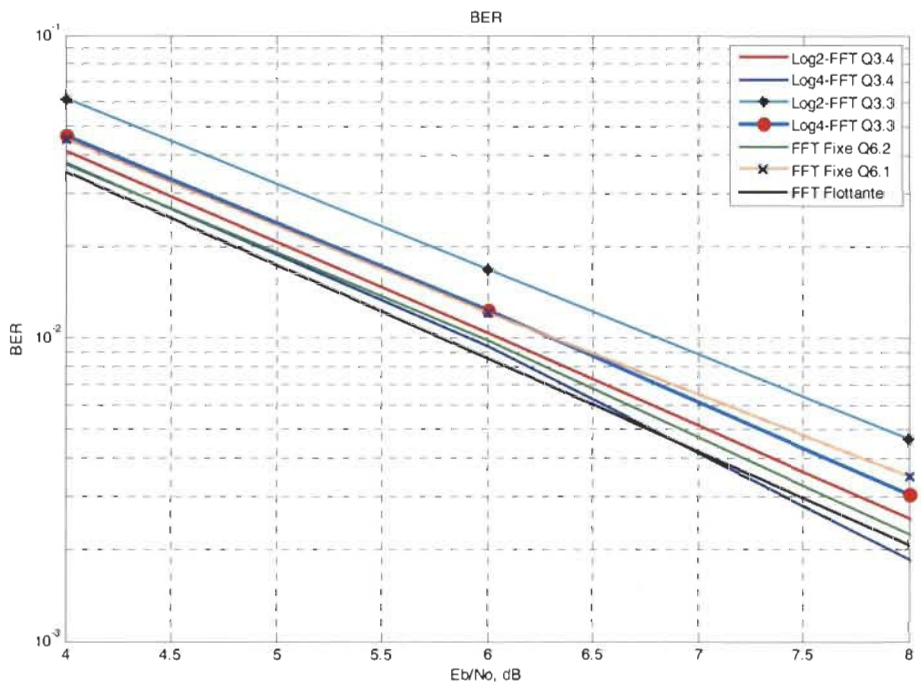
Dans ce projet nous avons opté pour la notation Qx.y, qui est une représentation de x bits d'entier et y bits de fraction, sans compter le bit de signe, pour une longueur binaire totale de x+y+1 [TMS03].

Lors de la quantification de la FFT, nous avons fixé les longueurs binaires à Q6.2 et Q6.1 (le premier chiffre à gauche représente le nombre de bits entiers, celui de droite le nombre de bits de fraction sachant que le bit de signe n'est pas inclus dans cette représentation). La log-FFT a été quantifiée en deux longueurs binaires, Q3.4 et Q3.3. Les résultats obtenus sont affichés dans la figure 3.9.

Le premier graphique illustre les différents BER sur une plage de *0* et *10 dB* et le deuxième est un zoom entre *4* et *8 dB* là ou l'erreur avoisine un BER de 1%. Les bases du logarithme exploré lors de ces simulations sont la base 2 et la base 4. Tel que démontré dans les résultats de SQNR, la base 3 affiche des résultats similaires à la base 4, cependant, la base 4 demande moins de ressources que la base 3. Dans la figure 3.9 (b), les BERs de la Log-FFT base 4 Q3.4 et la FFT Q6.2 représentent les meilleurs résultats obtenus, la Log-FFT base Q3.4 et la FFT Q6.1 sont aussi acceptables étant donné que pour un BER de 1% la différence de ces derniers, par rapport à la courbe de la FFT en virgule flottante, est inférieure ou égale à *0.5 dB*.



(a)



(b)

Figure 3.9 a) BER pour différentes FFT et b) agrandissement autour d'un BER de 1%

3.5 Réduction de la partie fractionnaire

Les différents résultats de BER obtenus dans la figure 3.9 démontrent que le nombre de bits de fraction est déterminant dans les performances de la FFT au niveau du récepteur OFDM. Comme introduit lors de l'évaluation de la dynamique, le nombre de bits de fraction permet d'accentuer la précision sur les performances de chaque courbe simulée. Pour interpréter ces courbes, nous allons analyser les résultats des courbes obtenus pour un $SNR = 6dB$ ensuite pour un BER aux alentours de *1%*.

Premièrement, nous observons que la Log-FFT à base 4 Q3.4, la FFT virgule fixe Q6.2 et Log-FFT à base 2 Q3.4 sont les 3 courbes qui représentent les meilleurs résultats par

rapport à la courbe réalisée en virgule flottante. Pour un $SNR = 6dB$, la courbe idéale (virgule flottante), la Log-FFT à base 4 Q3.4, la FFT virgule fixe Q6.2 et Log-FFT à base 2 Q3.4, enregistrent des BERs de *0.008793*, *0.0095*, *0.009717* et *0.0107* respectivement. Nous constatons que le passage d'un logarithme à base 2 pour un logarithme à base 4, joue un rôle important sur la réduction du taux d'erreur binaire (*0.0107* contre *0.0095)*. D'ailleurs, cela implique un gain de *12.63%*. Dans la même veine, grâce à l'approche logarithmique nous avons réalisé un gain binaire, d'un bit (8 bits pour la Log-FFT et 9 bits pour la FFT en virgule fixe). Ce gain au niveau de la longueur binaire avec l'utilisation d'une base supérieure à 2, plus spécifiquement 4, apportera un avantage significatif lors de l'implémentation.

Deuxièmement, en étudiant les courbes, de la Log-FFT à base 4 Q3.4, la FFT virgule fixe Q6.2, Log-FFT à base 2 Q3.4 et celle obtenue en virgule flottante, pour un BER fixe à *1%*.

Nous enregistrons les valeurs de *SNR* respectives suivantes : *5.90 dB*, *5.97dB*, *6.05dB* et *5.78 dB*. La perte de *dB* réalisée par toutes ces courbes par rapport à la courbe idéale (virgule flottante) est de *0.12dB* pour la Log-FFT à base 4 Q3.4, *0.19dB* pour la FFT virgule fixe Q6.2 et *0.27dB* pour la Log-FFT à base 2 Q3.4. D'une part les pertes de *dB* sont toutes inférieures à *0.3dB* et d'autre part la Log-FFT base 4 enregistre une perte plus faible en *dB* de *2%* par rapport à la Log-FFT base 2. Dit autrement, le cas Q3.4 pour atteindre un BER de 2%, le logarithme à base 4 permet un gain de 0.5 dB par rapport à une base 2 et cela tout en réduisant les ressources d'un rapport de 2, selon la figure 3.7.

Dans le prochain chapitre, ces résultats concluants seront déterminants pour l'aspect implémentation et surtout d'un point de vue gain en ressources matérielles. Aussi, grâce à ces résultats nous venons de réaliser un des objectifs de ce projet, l'efficacité d'utiliser une base logarithmique supérieure à 2 pour des meilleures performances BERvs*SNR*.

Chapitre 4
Implémentation FPGA de la FFT en arithmétique logarithmique

Précédemment, nous avons étudié la Log-FFT dans un récepteur OFDM et comparé ses résultats à une FFT conventionnelle. Dans ce chapitre, nous exposerons les architectures d'une *Butterfly* radix-2 conventionnelle et logarithmique, et comparerons leurs bons fonctionnements à l'aide d'une cosimulation Matlab-ModelSim.

Dans le cadre de ce projet, la technologie visée est le FPGA. Cette dernière a été choisie vu son importance dans l'industrie grâce à ses aspects reconfigurables. La technologie FPGA cible sera celle de la compagnie Xilinx, vu la disponibilité de la famille Xilinx à notre laboratoire. L'outil de synthèse, qui évalue les ressources de chaque architecture VHDL, validera la comparaison entre les deux modèles de *Butterfly*.

Comme dans les résultats de simulations Matlab, la longueur binaire est un facteur influant dans l'occupation des cellules du FPGA.

4.1 Circuit pour l'arithmétique logarithmique

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 *Butterflies*. On a ainsi bien défini les opérateurs et les éléments sensibles qui entrent dans la conception de l'architecture des *Butterflies* (conventionnelle et logarithmique).

Le logiciel utilisé pour le projet est Modelsim®. La plate-forme Matlab® servira quant à elle, à vérifier et valider le bon fonctionnement de l'architecture de Log-Butterfly créée sur Modelsim®. La validation du bon fonctionnement s'effectue en générant des données d'entrées avec Matlab qu'on injecte dans le modèle Matlab et ModelSim® ensuite nous comparons les deux résultats. Une fois que l'architecture est validée, on passe à l'étape de synthèse qui est effectuée avec l'outil Xilinx XST®. Figure 4.1

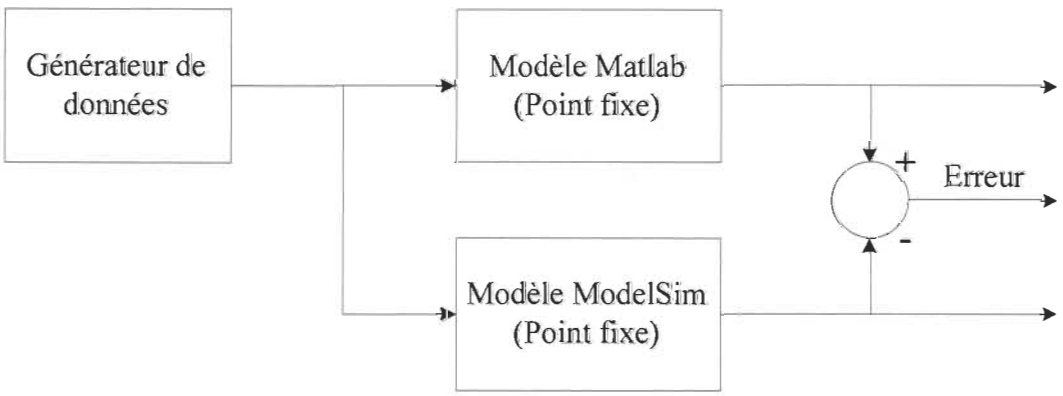


Figure 4.1 Diagramme de validation des modèles VHDL

4.1.2 Système des nombres logarithmiques

Le système des nombres logarithmique est constitué par des LUTs de conversion (Log-Lut et Antilog-Lut) et des opérateurs arithmétiques (multiplication/division et sommation/soustraction). Des représentations algorithmiques, reflétant la conception VHDL, seront exposées afin d'assimiler l'architecture de la Log-Butterfly. [WAN03]

4.1.2.1 Log-Lut/AntiLog-Lut

La Log-Lut est la première étape à laquelle sont confrontées les données de la Log-Butterfly.

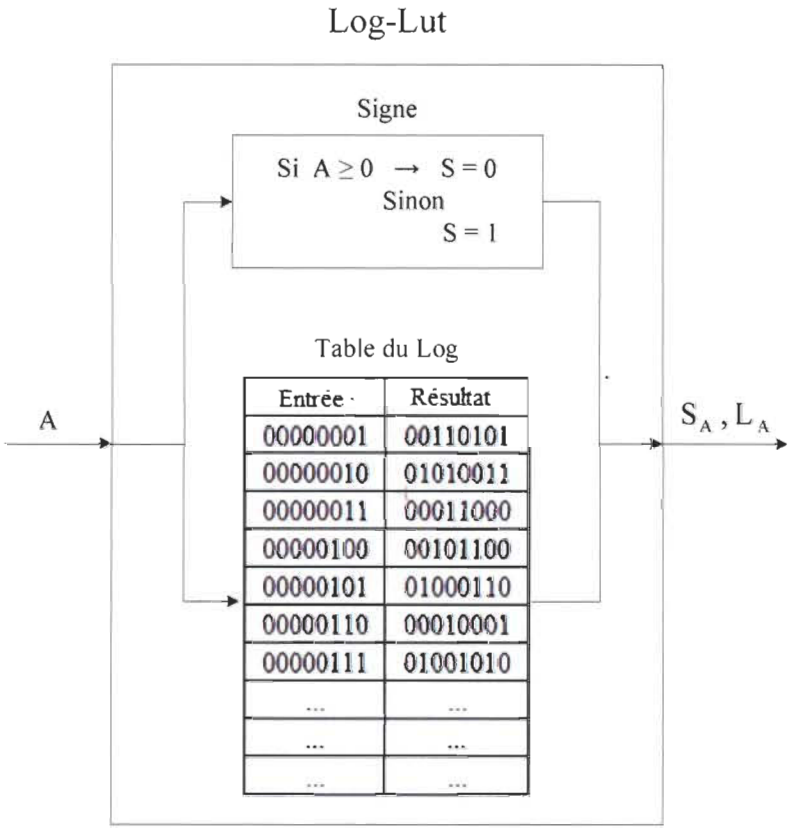


Figure 4.2 La représentation de la Log-Lut

Dans un premier temps, le signe de la donnée A est stocké dans la variable S, il prend la valeur 0 ou 1 dépendamment de ce dernier, figure 4.1. Ensuite, la valeur absolue de A est

utilisée comme une adresse qui permet de récupérer son logarithme dans la table du log. Cette table est un tableau prédéfini, conservé dans une mémoire, dans lequel chaque adresse mémoire pointe sur la valeur de son logarithme. Les valeurs binaires dans les tables du Log et AntiLog sont à titre représentatif. Finalement, le résultat à la sortie se traduit par l'ajout du bit de signe S au logarithme L saisi au tableau. Exemple : Si A est un nombre représenté sur 8 bits, son logarithme aura la même longueur par contre la sortie de la Log-Lut sera sur 9 bits.

Rappel : Dans le LNS, on applique le logarithme sur une donnée en valeur absolue pour d'éviter de manipuler des nombres complexes. D'où, l'importance de conserver le signe celle-ci.

La table du Log est influencée par 2 facteurs : la longueur binaire utilisée pendant la conception et la base du logarithme.

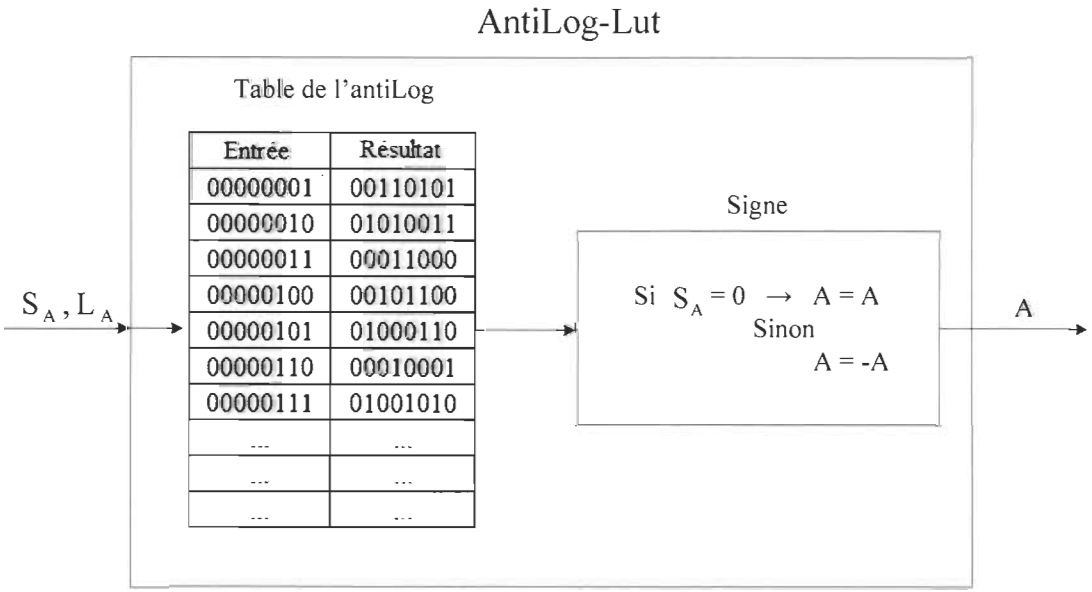


Figure 4.3 La représentation de l'AntiLog-Lut

Nous observons à la figure 4.2 le modèle de l'AntiLog-Lut. Contrairement à Log-Lut, l'Antilog-Lut permet de convertir le logarithme L et son signe S vers son équivalent linéaire. Après avoir déterminé le nombre A grâce à l'adresse L qui est son logarithme, le signe S divulgue l'information sur le signe de A. Maintenant que les LUTs ont été présentés, nous pouvons explorer mieux les opérateurs arithmétiques logarithmiques et profiter de leurs avantages pour la conception.

4.1.2.2 La multiplication

La multiplication logarithmique est l'opérateur arithmétique le plus rentable de point de vue matériel, la multiplication n-bits par n-bits dans le système linaire devient une simple sommation dans le LNS.

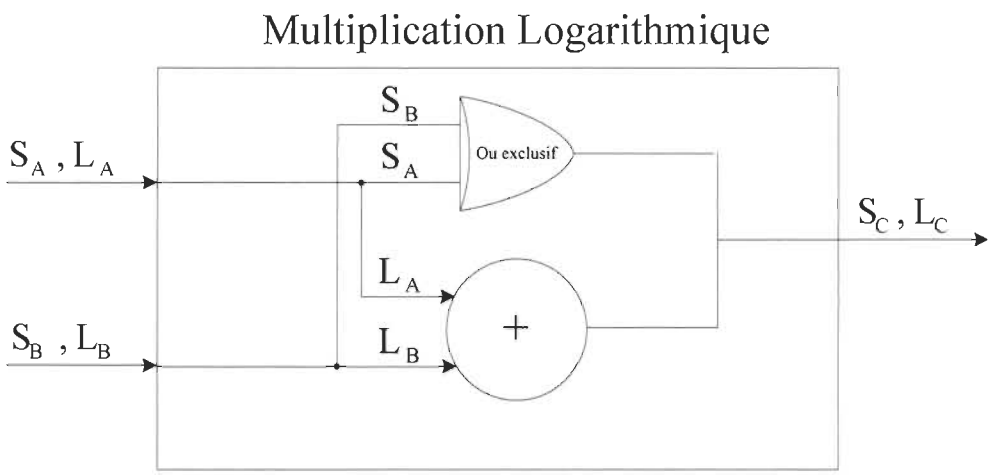


Figure 4.4 La représentation de la multiplication dans le LNS

Les deux données logarithmiques et leurs signes sont calculés séparément, figure 4.3, les L_A et L_B sont additionnés bit par bit et leurs signes S_A et S_B sont traités par la fonction logique « Ou exclusif ».

4.1.2.3 La sommation/soustraction

La sommation et la soustraction représentent le même opérateur arithmétique dans le LNS, le même algorithme est appliqué à part le signe S qui varie (si on désire une sommation, on garde le même état des variables S_A et S_B sinon on inverse le signe de la donnée qu'on veut soustraire). Comparée à la multiplication, la sommation/soustraction représente l'opérateur arithmétique le plus couteux en ressource matérielle.

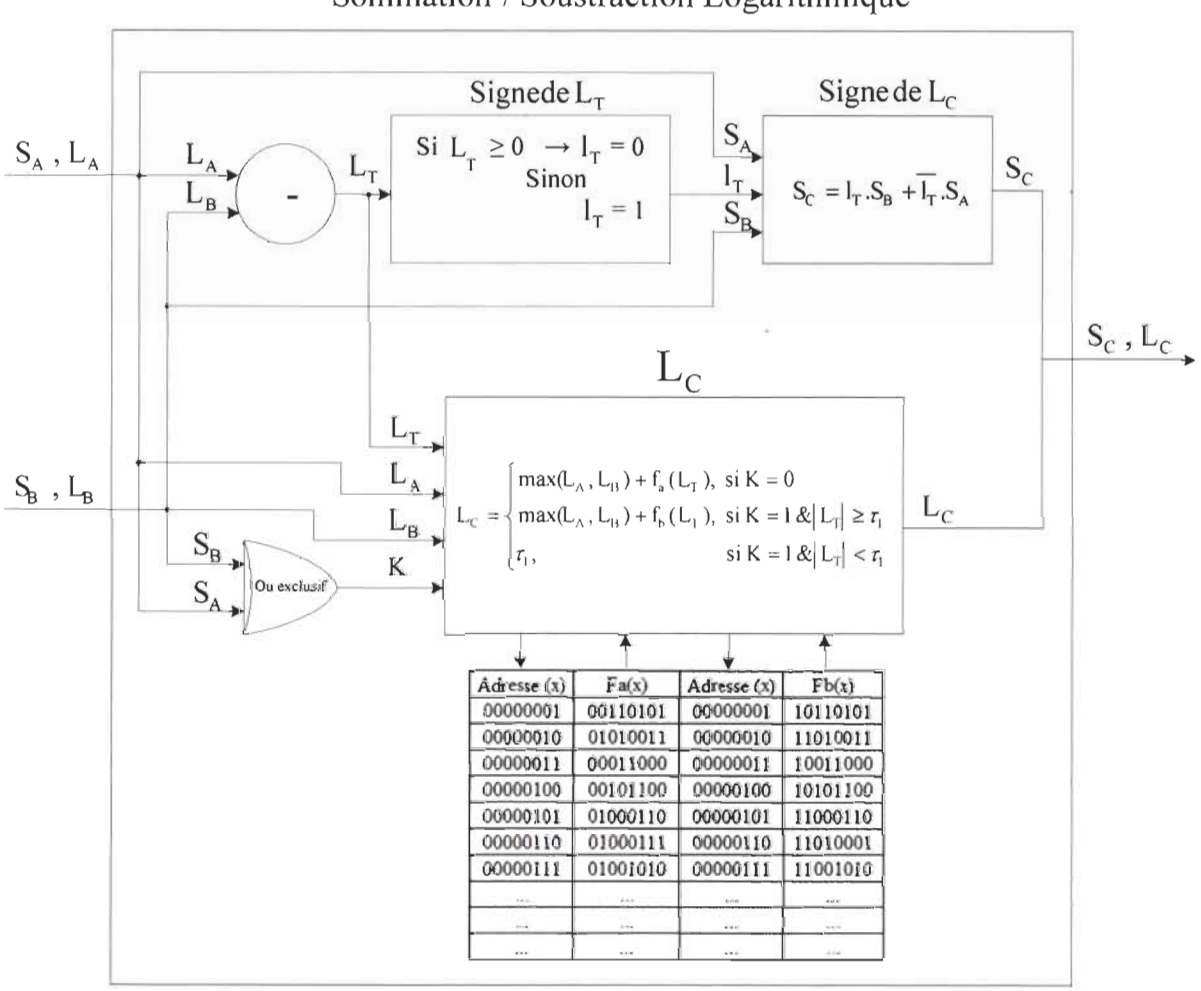


Figure 4.5 La représentation de la Sommation/Soustraction dans le LNS

La figure 4.3 illustre la conception de l'opération de sommation dans le domaine logarithmique. Cet opérateur arithmétique est composé de plusieurs étapes. Dans un premier temps, nous effectuons une opération de soustraction entre L_A et L_B pour générer, L_T, une variable temporaire. Celle-ci, permettra à l'aide de son signe l_T, de S_A et S_B de retrouver le signe du résultat de sortie S_C. Dans un deuxième temps, le calcul de L_C nécessite une opération de sommation, précédé par l'acheminement d'une donnée supplémentaire acquise dans l'une des deux tables Fa ou Fb. Ces dernières sont l'approximation des deux fonctions $F_a(x) = \log_b\left(1 + b^{-|x|}\right)$ et $F_b(x) = \log_2\left(1 - b^{-|x|}\right)$ avec b qui représente la base du logarithme utilisé. En résumé, la sommation logarithmique est constituée d'une opération de soustraction, un accès à l'une des deux tables Fa ou Fb et une opération de sommation. Contrairement à la multiplication, la partie la plus couteuse matériellement se situe au niveau des deux tables Fa et Fb. Comme démontré précédemment avec les log-Lut, Fa et Fb sont influencées par la longueur binaire et la base du logarithme.

4.1.3 Conclusion

Nous avons vu dans cette partie les différentes architectures utilisées pour la conception de la Log-Lut, l'Antilog-Lut, la multiplication et la sommation/soustraction logarithmiques. Nous avons observé aussi la complexité de l'opération de sommation par rapport à la multiplication plus spécifiquement au niveau des tables Fa et Fb. Tous ces

modèles ont été validés dans une cosimulation Matlab/ModelSim. Dans la prochaine partie, nous allons introduire la Log-Butterfly radix2.

4.2 Butterfly radix-2 en arithmétique logarithmique

Dans ce projet, nous avons choisi la FFT radix-2 comme sujet principal. Étant l'outil de calcul le plus sollicité en traitement numérique du signal et en télécommunication, nous l'avons conçu dans le domaine linéaire (conventionnel) et logarithmique. Ensuite, nous l'avons simulé, et observé son fonctionnement dans un récepteur OFDM. Dans cette section, nous avons programmé la Butterfly radix-2 conventionnelle et logarithmique sous ModelSim afin de comparer les résultats de synthèse et tirer des conclusions sur les deux approches.

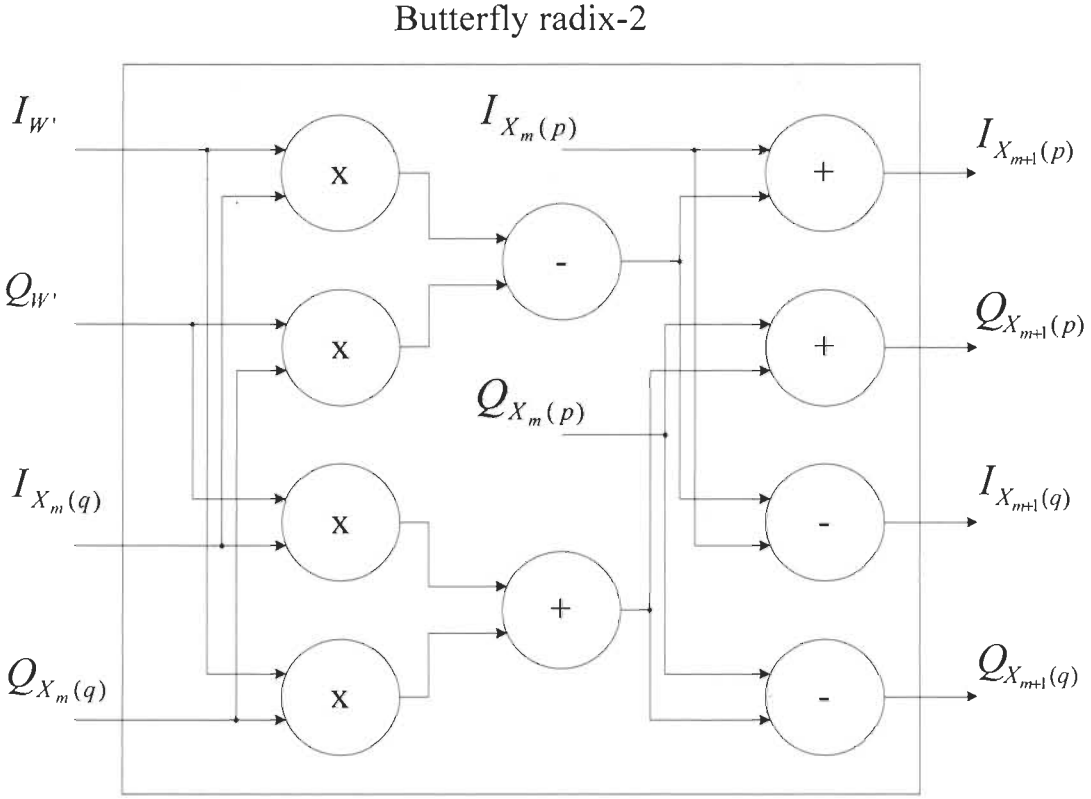


Figure 4.6 La représentation de la Butterfly radix-2

Comme vu au chapitre 2 à la figure 2.5 et selon l'équation (2.21), la Butterfly radix-2 est composée d'une multiplication complexe, une sommation et une soustraction complexe. Les équations (4.1) à (4.3) reprennent la conception effectuée à la figure 4.5.

$$C = W\, X_m(q) \tag{4.1}$$

où W est un des coefficients de Fourier (*Twiddle factor*) composé de sa partie réelle, I_w, et sa partie imaginaire Q_w, $W = I_W + jQ_W$. $X_m(p)$ est le vecteur des données d'entrées de la Butterfly et se propage dans la Butterfly afin d'obtenir les sorties suivantes

$$X_{m+1}(p) = X_m(p) + C \tag{4.2}$$

$$X_{m+1}(q) = X_m(p) - C \tag{4.3}$$

La variable C représente le résultat de la multiplication complexe de l'équation (4.1) décrite par sa partie réelle, I_C, et sa partie imaginaire, Q_C, à savoir

$$I_C = I_W\, I_{X_m(q)} - Q_W\, Q_{X_m(q)} \tag{4.4}$$

$$Q_C = I_W\, Q_{X_m(q)} + Q_W\, I_{X_m(q)} \tag{4.5}$$

Les équations (4.2) et (4.3) qui sont une sommation et une soustraction complexe, deviennent deux sommations et deux soustractions réelles. Une Butterfly radix-2 conventionnelle, présenté figure 4.5, nécessite 4 multiplieurs et 6 opérations de sommations/soustractions.

La Log-Butterfly radix-2 est représentée à la figure 4.6, cette figure a été simplifiée afin de mieux visualiser l'architecture générale. L'opérateur de sommation logarithmique, vu précédemment dans figure 4.4, a été réduit à une soustraction, le maximum entre L_A et L_B (les deux entrées à additionner), les tables Fa et Fb et la sommation finale (voir

figure 4.7). Autrement dit, nous avons simplifié le diagramme en éliminant la partie qui consiste à calculer les variables temporaires ainsi que les différents signes.

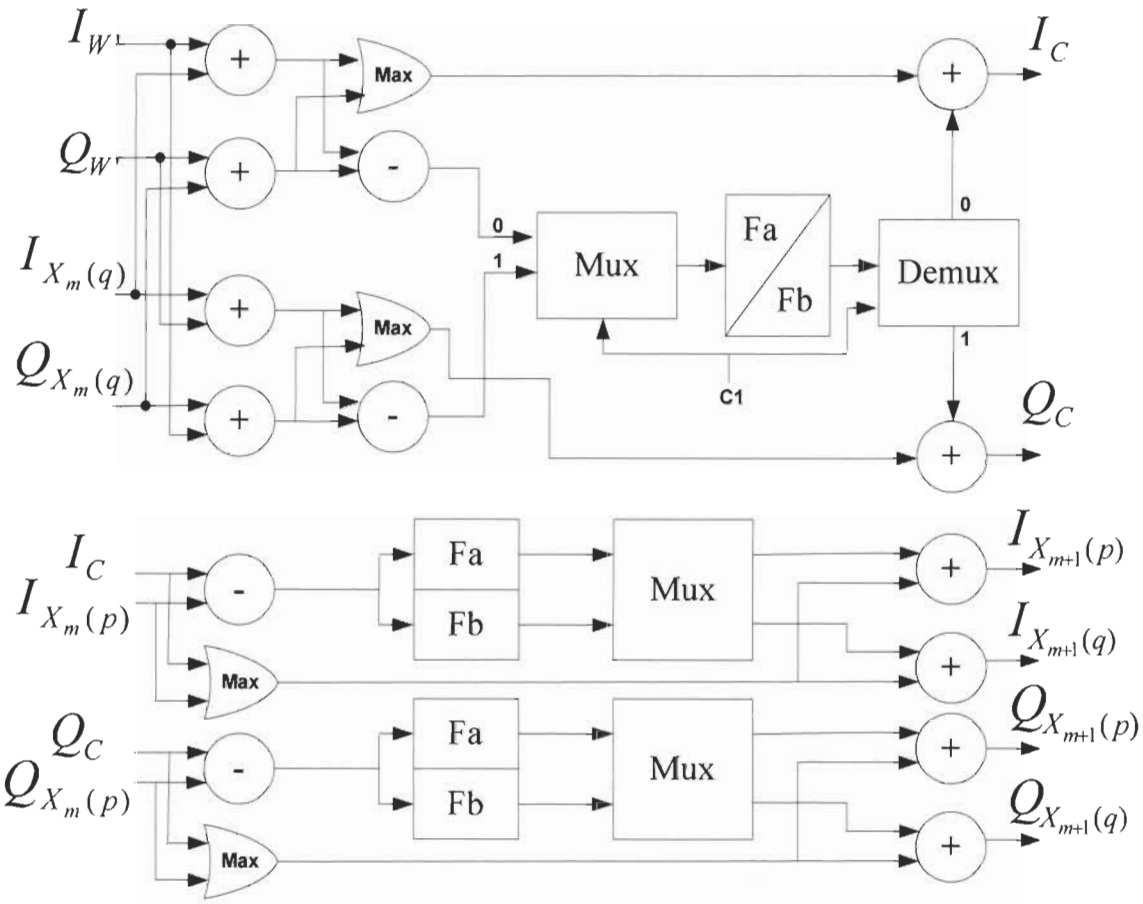


Figure 4.6 La représentation de la Log-Butterfly radix-2

La figure4.6 est constituée de deux parties : La première permet d'obtenir le résultat de l'opération de multiplication complexe, et la deuxième les deux résultats des deux opérations de sommation et soustraction complexes. Étant donné que la Butterfly radix-2 est un algorithme dont l'exécution est séquentielle, dans un premier temps, ensuite parallèle, ce choix d'architecture devient indispensable [WAN03].

Le premier bloc représente les équations (4.4) et (4.5), celles-ci sont constituées de 4 multiplications réelles, une soustraction et une sommation réelle. Dans la figure 4.6, les multiplications sont remplacées par des sommations et les deux opérations de sommation/soustraction par la représentation simplifiée de ces deux dernières dans le LNS.

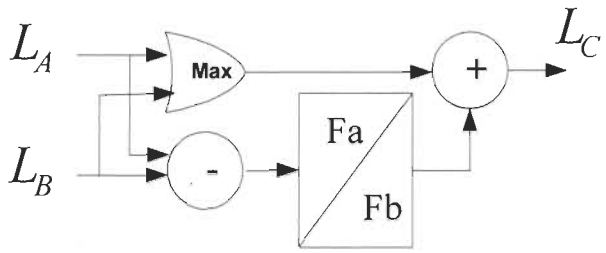


Figure 4.7 Représentation simplifiée de la Sommation/soustraction logarithmique

Nous avons utilisé une seule table Fa/Fb pour les deux opérations de sommation/soustraction, figure 4.6, afin de réduire les ressources, cependant il fallait ajouter un bloc Mux/Démux pour gérer l'accès à ces tables dans le but d'éviter toute perte d'information ou désynchronisation à la sortie.

La deuxième partie de la figure 4.6 représente les équations (4.2) et (4.3). Si nous développons celles-ci, nous obtenons :

$$I_{X_{m+1}(p)} = I_{X_m(p)} + I_C \tag{4.6}$$

$$I_{X_{m+1}(q)} = I_{X_m(p)} - I_C \tag{4.7}$$

$$Q_{X_{m+1}(p)} = Q_{X_m(p)} + Q_C \tag{4.8}$$

$$Q_{X_{m+1}(q)} = Q_{X_m(p)} - Q_C \tag{4.9}$$

Les équations (4.6) à (4.9) décrivent le résultat de la sommation et la soustraction complexes entre celui de la multiplication, obtenue dans (4.5) et (4.6), avec l'entrée $X_m(p)$. Pour implémenter cette partie, nous avons utilisé les mêmes ressources pour faire la sommation et la soustraction réelles. Dans le modèle de la sommation/soustraction logarithmique, figure 4.4, la variable K, qui dépend de S_A et S_B, définit si le résultat final L_C est le fruit d'une sommation ou une soustraction. En plus, K permet d'identifier laquelle des tables Fa ou Fb va être parcourue pour obtenir ce résultat final. Or, étant donné que nous effectuons la sommation et la soustraction réelle simultanément, les Tables, Fa et Fb vont être sollicités en même temps. Autrement dit, nous utilisons deux nombres réels pour exécuter deux opérations inverses, ce qui implique que la seule différence entre ces deux dernières se situe au niveau du signe, d'où une utilisation simultanée des deux tables Fa et Fb.

Enfin, pour additionner les valeurs obtenues des tables avec le Max des deux entrées, nous utilisons un bloc Mux qui est basé sur la valeur de K pour diriger le bon résultat à la bonne sortie. Le bon fonctionnement de l'architecture de la Butterfly radix-2 a été validé à l'aide d'une cosimulation Matlab/ModelSim qui a démontré sa robustesse.

4.3 Évaluation de synthèse pour FPGA

L'évaluation de la synthèse d'un projet est une étape qui consiste à estimer les ressources nécessaires pour l'implémentation dans un FPGA. Avant d'effectuer la synthèse, nous

avons élargi la gamme des résultats obtenus dans le chapitre précédent afin de mettre davantage de lumière sur l'apport de l'approche logarithmique dans ce projet.

4.3.1 Résultats de la plateforme OFDM

Nous avons saisi dans le Tableau 4.1 les différentes valeurs de BER, obtenues dans les simulations de la plateforme OFDM avec une FFT en virgule flottante et fixe, pour un *SNR* fixé à *6dB*. Plus la longueur binaire en virgule fixe augmente, plus les valeurs de BER diminue pour s'approcher de ceux de la FFT flottante. Nous observons une fluctuation entre les deux valeurs de Q6.4 et Q6.5 par rapport au BER flottant, cette variation s'explique par le nombre de symboles OFDM simulés.

Tableau 4.1 BER de la FFT en virgule fixe et flottante pour un *SNR de 6 dB*

Virgule Flottante		
0.008793		
WL (bits)	WL	Virgule Fixe
7	Q6.0	0.032478
8	Q6.1	0.012861
9	Q6.2	0.009716
10	Q6.3	0.008904
11	Q6.4	0.008685
12	Q6.5	0.009147

En réalité, plus le nombre de bits augmente en virgule fixe plus le résultat en BER devrait tendre cers celui en virgule flottante. D'où l'importance d'un nombre élevé de symboles OFDM lors des simulations afin d'assurer un bon lissage des résultats. À noter que Q6.4 présente un résultat légèrement inférieur à celui en virgule flottante.

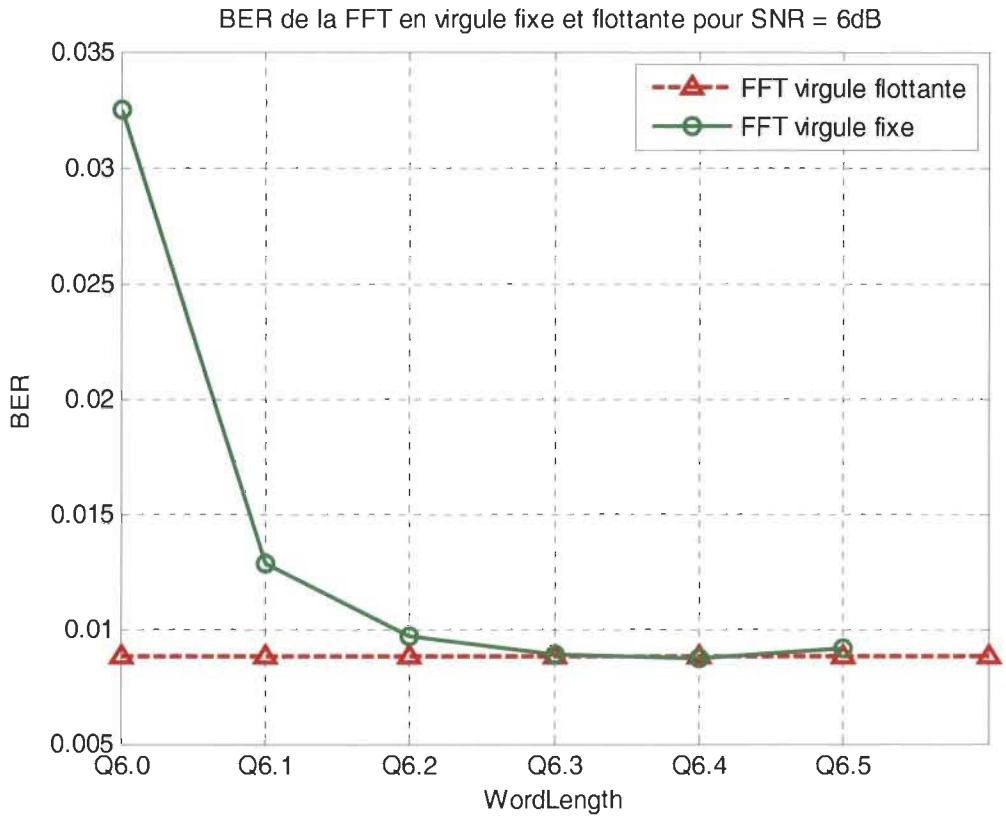


Figure 4.7 BER de la FFT en virgule fixe et flottante pour SNR = 6dB

Nous constatons, dans la Fig.4.6, qu'à partir de la longueur Q6.2 nous atteignons des résultats satisfaisants. Nous avons un BER de *0.009716* et le BER en virgule flottante est de *0.008793*. Autrement dit, la FFT en virgule fixe nécessite un minimum de 9 bits pour atteindre les résultats en virgule flottante.

Le tableau 4.2 représente les valeurs de BER obtenues par la Log-FFT pour des bases logarithmiques allant de 2 à 5 dans notre plateforme OFDM avec un *SNR fixé à 6dB*. Nous avons pris en considération plusieurs bases de logarithme afin de voir la variation du BER. Cela rajoute un deuxième facteur qui influence significativement la tendance du BER.

Tableau 4.2 BER de la Log-FFT en virgule fixe pour un *SNR = 6 dB*

		BER (SNR = 6 dB)						
WL (bits)	WL	2	2.5	3	3.5	4	4.5	5
6	Q3.2	0.1267	0.0422	0.0304	0.0289	0.0300	0.0384	0.0401
7	Q3.3	0.0172	0.0113	0.0118	0.0118	0.0127	0.0139	0.0142
8	Q3.4	0.0107	0.0092	0.0099	0.0095	0.0095	0.0096	0.0100
9	Q3.5	0.0095	0.0089	0.0087	0.0092	0.0090	0.0094	0.0093

Comme observé au tableau 4.2, plus la longueur binaire augmente, plus les valeurs de BER diminuent. L'aspect pertinent, avec la variation de base, est la diminution du BER quand la base argumente. Par contre, il existe une base « optimale » qui fournit les meilleurs résultats. Toute base supérieure à cette dernière, réduit la dynamique dans les LUTs et facilite la propagation de l'erreur dans les calculs de la FFT, cela détériore les valeurs du BER. Pour mieux illustrer ce constat, nous avons tracé les courbes à la figure 4.7 qui permet de mieux visualiser les différentes variations du BER en fonction de la longueur binaire ainsi que l'apport significatif du changement de base du logarithme. Les résultats peu intéressants de la Log-FFT Q3.2 n'ont pas été représentés afin d'alléger le graphique.

À partir de la figure 4.7, nous constatons que le premier facteur qui permet d'atténuer le BER est la longueur binaire. La courbe noire représente la Log-FFT Q3.5 et la courbe bleue la Log-FFT Q3.3. Le deuxième facteur qui permet une amélioration notable du BER et le changement de la base du logarithme. Chaque courbe possède sa base « optimale». Les deux premières courbes de la Log-FFT Q3.3 et Q3.4 atteignent leurs minimums du BER à *0.0113* et *0.0092* respectivement pour une base égale à 2.5. Par contre, la base optimale pour Log-FFT Q3.5 est égale à 3. Noter l'importance de la base dans le cas Q3.3. La Log-FFT qui nous intéresse dans de cas de figure est le cas Q3.4 avec une base égale à 2.5. Cette dernière satisfait les besoins établis au niveau du BER.

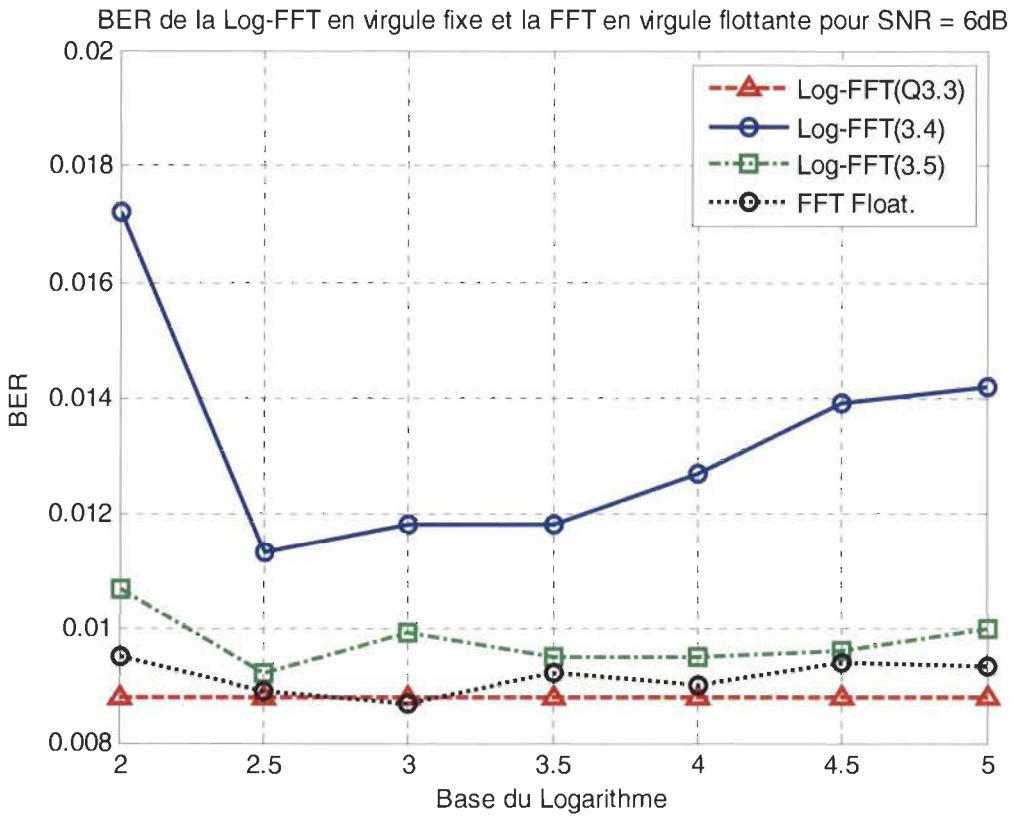


Figure 4.8 BER de la Log-FFT en virgule fixe et la FFT en virgule flottante pour SNR de *6dB*

En résumé, les deux courbes qui répondent aux meilleurs résultats sont la Log-FFT à base 2.5 avec Q3.4 avec une BER de 0.0092 et la FFT en virgule fixe Q6.2 offrant une BER de 0.009716. Noté que la méthode proposée atteint un BER inférieur à la méthode en virgule fixe tout en ayant un bit de moins dans sa représentation binaire.

Après que l'analyse du BER soit établie, nous nous intéressons maintenant sur l'aspect de la perte de performance, SNR_{loss}, des méthodes étudiées mesurée en dB dans la plateforme OFDM étudiée au chapitre 3. Cette mesure de perte de performance se définie comme étant l'écart du SNR entre la méthode en virgule flottante et celle en virgule fixe

pour obtenir un BER donné (BER de 1% dans notre cas). Ainsi, à partir des résultats de la figure 3.9, nous avons préparé deux tableaux qui présentent les pertes de performance mesurée en dB pour la plateforme OFDM afin d'atteindre un BER de 1%.

Tableau 4.3 Perte de performance, SNR_{loss}, de la FFT en virgule fixe pour un BER de 1%

WL (bits)	WL	Fixed Point
7	Q6.0	1.84232
8	Q6.1	0.53615
9	Q6.2	0.14074
10	Q6.3	0.01769

Le tableau 4.3 présente la perte performance, SNR_{loss}, de la FFT en virgule fixe par rapport à la virgule flottante servant de référence. Ces données sont générées à l'aide d'un calcul d'interpolation des résultats de la figure 3.9. À la figure 4.8, les SNR_{loss} sont présentées pour le cas de la méthode en virgule fixe pour différente longueur binaire de 7 à 10 bits. Comme attendu, la grandeur binaire influence considérablement la perte de performance où nous observons une perte quasi nulle pour Q6.3.

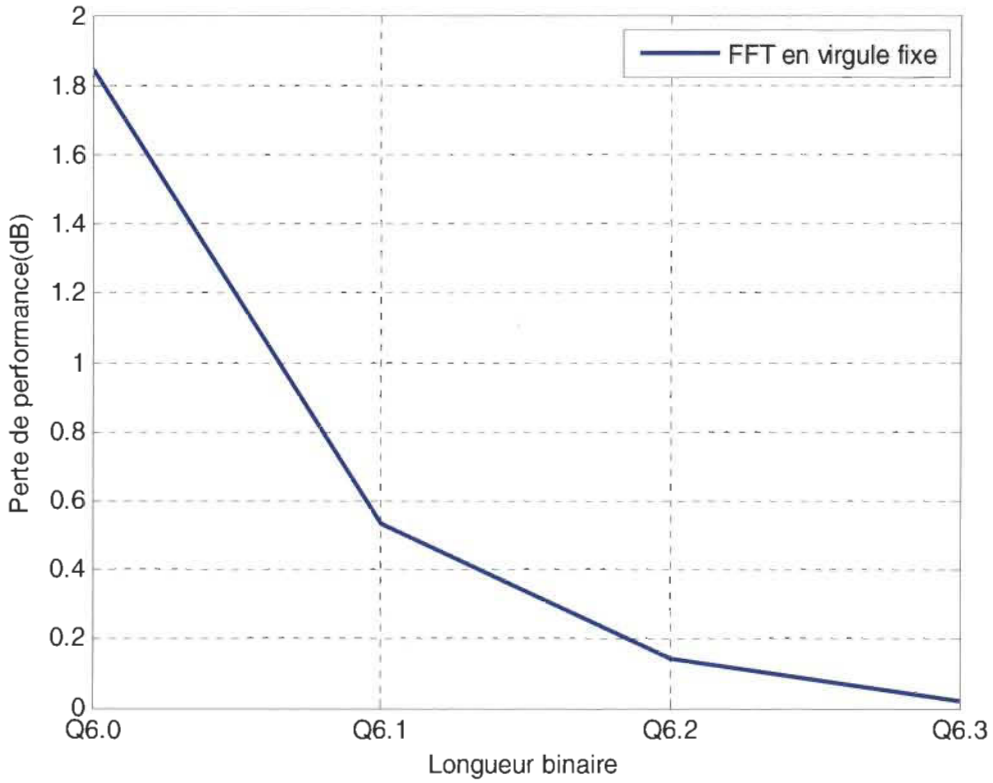


Figure 4.9 Perte de performance, SNR_{loss}, de la FFT en virgule fixe pour un BER de 1%

Les bits de fraction apportent une meilleure précision sur les résultats de sortie de la FFT, ce qui minimise cette perte en dB. Nous pouvons voir qu'à partir de Q6.2 les pertes de performance sont inférieures à *0.2dB*, ce qui est satisfaisant.

Les résultats de la Log-FFT proposée se dressent en deux parties. Dans un premier temps, le tableau 4.4 représente les pertes de performances de la Log-FFT en fonction de différente longueur binaire et de la base logarithmique. D'autre part, nous avons tracé un graphique de ses pertes pour analyser les avantages de cette approche.

Tableau 4.4 Perte de performance, SNR_{loss}, de la Log-FFT en virgule fixe pour un BER de 1%

		Base du logarithme b						
WL (bits)	WL	2	2.5	3	3.5	4	4.5	5
6	Q3.2	3.761708	2.21154473	1.74909418	1.677746	1.730418	2.078493	2.139573
7	Q3.3	0.946049	0.35369557	0.41474434	0.414744	0.518383	0.645688	0.675796
8	Q3.4	0.276767	0.06379931	0.16719688	0.109044	0.109044	0.123809	0.181368
9	Q3.5	0.109044	0.01705449	-0.0149925	0.063799	0.032809	0.094123	0.079043

Nous avons étudié au tableau 4.4 les pertes de performances de la Log-FFT pour les longueurs binaires Q3.2, Q3.3, Q3.4 et Q3.5. Nous avons utilisé un logarithme allant de la base 2 jusqu'à la base 5 par pas de variation de 0.5 pour voir l'impact sur les performances de la Log-FFT.

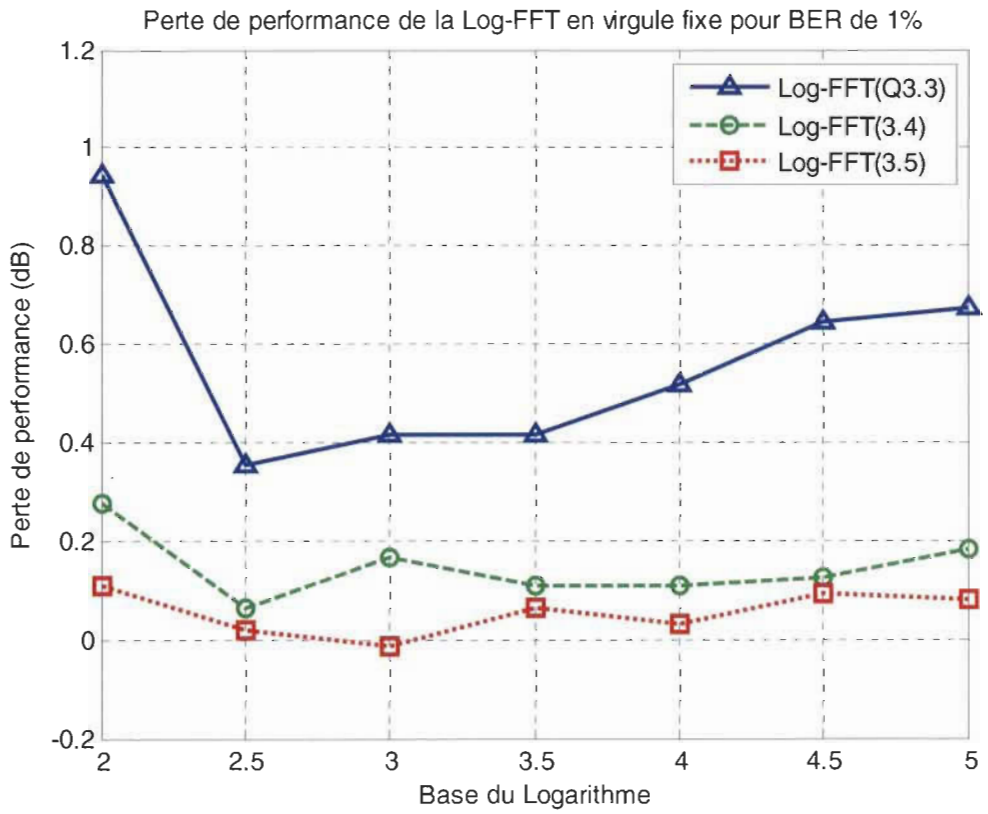


Figure 4.10 Perte de performance, SNR_{loss}, de la Log-FFT en virgule fixe pour un BER de 1%

Nous constatons que le SNR_{loss} de la Log-FFT s'établit de la longueur binaire la plus petite, Q3.3, vers la plus grande, Q3.5. Nous n'avons pas affiché la Log-FFT Q3.2 parce qu'elle accuse un SNR_{loss} élevée. Les résultats obtenus à la figure 4.9 démontrent que les bits de fractions améliorent significativement les performances de la Log-FFT. La base du logarithme joue un rôle important pour atténuer les pertes, sauf qu'il existe une base «optimale» qui définit le SNR_{loss} le plus faible, plus évident pour Q3.3. La base 2.5 permet d'atteindre un SNR_{loss} minimal de la Log-FFT Q3.3 et Q3.4. Par contre, la Log-FFT Q3.5 affiche son minimum à la base 3. D'ailleurs, la perte négative observée à la base 3, qui devient un gain, est en fait dû au nombre de symboles OFDM utilisé dans les simulations. En réalité, quand les pertes de performances atteignent 0 (SNR_{loss}=0), cela signifie que nous avons des résultats qui coïncident avec ceux obtenus en virgule flottante (la référence).

4.3.2 Synthèse sur une technologie FPGA

Dans cette section nous allons observer les résultats de synthèse des deux Butterflies radix-2 de la FFT en virgule flottante conventionnelle et à base logarithmique. Nous avons choisi la famille Virtex 4 et le modèle XC4VLX15, pour établir la synthèse. Dans le cadre de ce projet, le choix de la famille n'a pas vraiment d'impact sur les résultats mesurés. C'est pourquoi nous présenterons les résultats que pour le XC4VLX15. Cependant, nous avons réalisé nos synthèses sur les deux approches sans l'utilisation des multiplieurs câblés DSP48.

Le tableau 4.5 représente le nombre de Slices nécessaire pour la conception de la Butterfly en virgule fixe conventionnelle.

Tableau 4.5 Les ressources FPGA utilisées par la FFT en virgule fixe conventionnelle

WL (bits)	WL	Nombre de Slices
7	Q6.0	192
8	Q6.1	227
9	Q6.2	291
10	Q6.3	333
11	Q6.4	373
12	Q6.5	420

Dans la section précédente, nous avons constaté que plus la longueur binaire augmente, plus nous enregistrons une réduction significative du BER. Dans la synthèse, la longueur binaire représente le prix à payer en ressources matérielles. À partir de là, un compromis entre la consommation en ressource et la précision des calculs recherchées est indispensable.

Tableau 4.6 Les ressources FPGA utilisées par la Log-FFT en virgule fixe

		Base du logarithme b						
WL (bits)	WL	2	2.5	3	3.5	4	4.5	5
6	Q3.2	211	193	188	191	192	192	189
7	Q3.3	262	248	234	242	228	227	228
8	Q3.4	321	289	284	282	281	280	262
9	Q3.5	445	402	366	357	355	353	343

Ce compromis est le choix d'une longueur binaire qui respecte nos objectifs. La figure 4.10 montre la croissance linaire des ressources FPGA de la FFT conventionnelle en virgule fixe en fonction de la longueur binaire. Au tableau 4.6, les résultats de la Log-FFT sont classés en fonction de la longueur binaire et la base du logarithme. Comme

pour les résultats de la FFT à virgule fixe conventionnelle, la longueur binaire entraine une hausse des ressources.

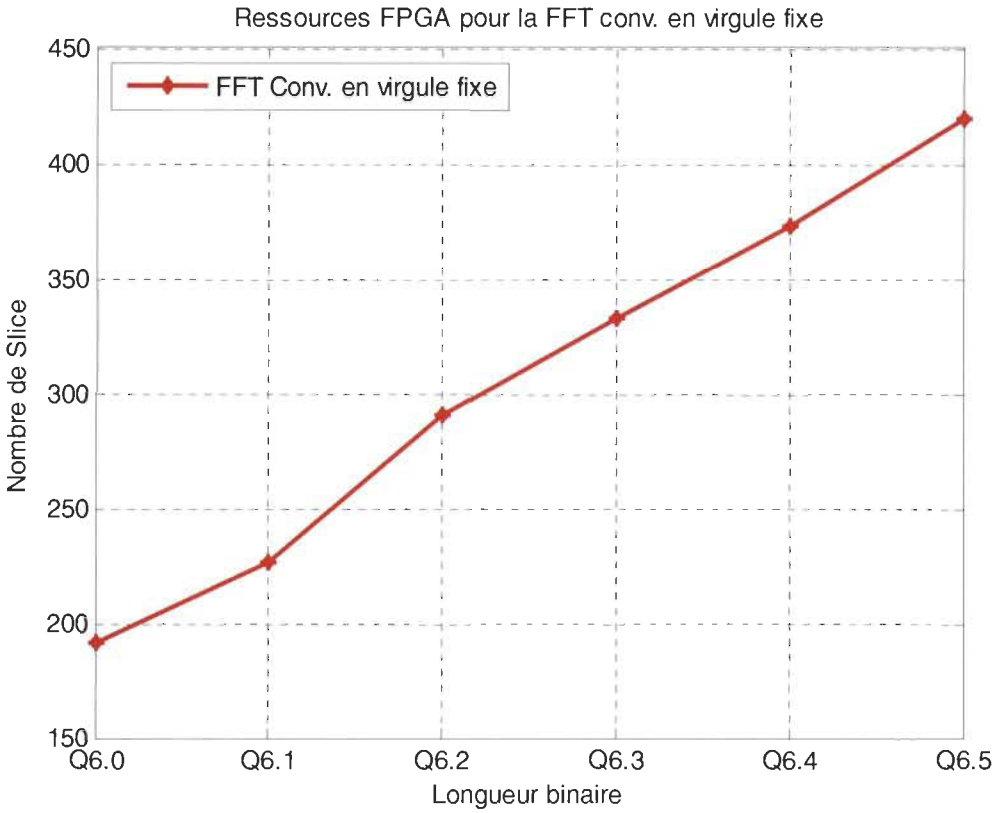


Figure 4.11 Les ressources FPGA utilisées par la FFT en virgule fixe conventionnelle

Cependant, la base du logarithme réduit ses derniers. Plus spécifiquement, la Log-FFT base 5 est celle qui occupe le moins de coût d'un point de vue ressource comparée aux autres base de logarithmes, figure 4.11. Cette faible consommation de slice résulte par la réduction des valeurs stockées à l'intérieur des LUTs. À titre de rappel, une base de logarithme croissante réduit la dynamique à l'intérieur des LUTs de conversion des logarithmes et antilogarithmes. Nous constatons aussi, à la figure 4.11, que pour la même longueur binaire, la FFT conventionnelle est plus performante que la Log-FFT. Sauf que dans nos résultats de simulations dans la plateforme OFDM, nous avons démontré que la

Log-FFT nécessitait un bit de moins que la FFT conventionnelle, ce qui implique un gain en ressource.

Ces résultats de synthèse sont extraits des autres tableaux qui renferment plus d'informations sur la synthèse des deux approches. Les tableaux A.1à A.9, dans la section Annexe, regroupent les résultats de synthèse dans un FPGA en fonction de la longueur binaire pour un logarithme à base 2, 3 et 4.

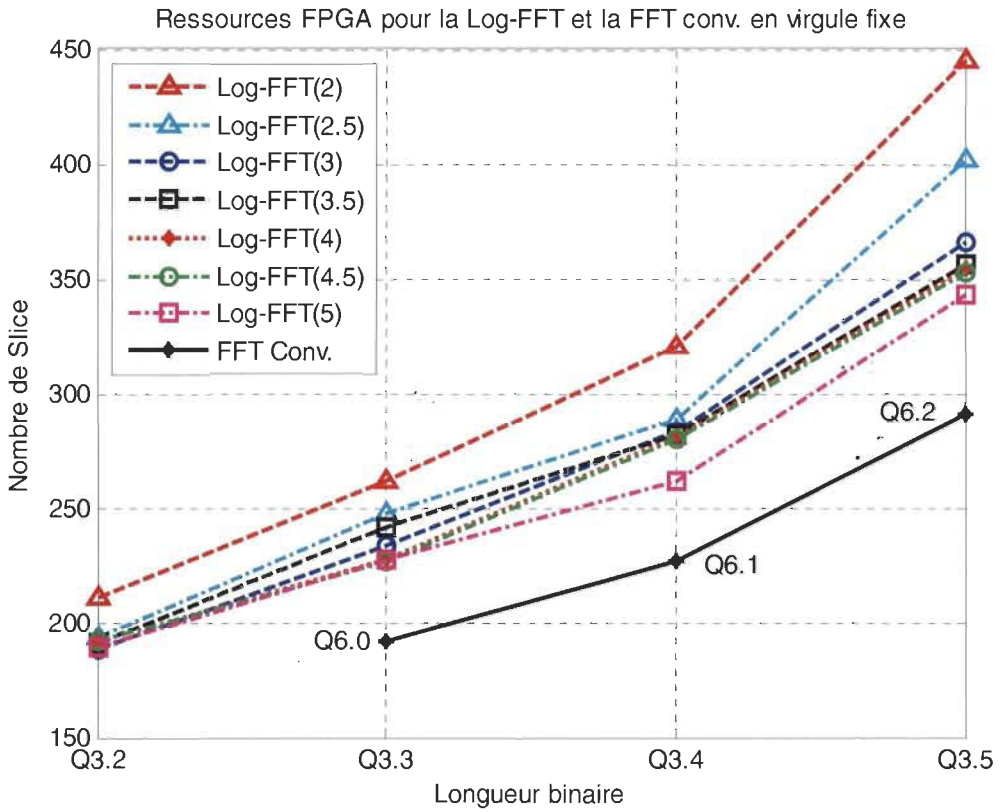


Figure 4.12 Les ressources FPGA utilisées par la Log-FFT

Les graphes A.1 à A.6 représentent les ressources matérielles utilisées par les deux approches pour différentes bases de logarithme et différentes longueurs binaires ainsi que le gain observé pour chacune de la Log-FFT avec 3 bits et 4 bits de fractions.

Nous constatons, dans les tableaux A.1 à A.6, que pour la même longueur binaire la FFT à virgule fixe conventionnelle est plus avantageuse. Cependant, à partir de 10 bits et plus, la Log-FFT commence à enregistrer un gain en ressources. Dans le cas de ce projet, nous nous ne comparons pas à la même longueur binaire entre les deux approches, nous avons vu auparavant que pour des résultats satisfaisants, la FFT conventionnelle nécessite 1 bit de plus que la Log-FFT. Ce qui implique l'étude d'une fonction de coût afin de mieux interpréter ses résultats

4.3.3 Fonction de coût

La fonction de coût est une formulation qui permet quantifier la mesure du meilleur compromis entre plusieurs approches étudiées. Cette formulation peut être établie selon plusieurs paramètres dépendamment du contexte de réalisation du projet. Pour notre étude, nous avons utilisé la fonction de coût selon l'équation (4.1) comme déjà établi dans [NOU08] et [NOU11].

$$J = SNR_{loss} \times R \tag{4.1}$$

où SNR_{loss} représente la perte de performance de la FFT exprimée en dB et R représente les ressources exprimée en nombre de Slices nécessaires pour l'implémentation sur FPGA.

Dans notre projet, on essaie de réduire ces deux paramètres afin d'une part, d'assurer une bonne performance, qui respecte la précision requise, et d'autre part d'utiliser le minimum de ressources matérielles pour la conception. La minimisation du critère J permettra de définir l'approche qui représente le meilleur compromis d'un point de vue performance versus ressources consommées.

Tableau 4.7 Fonction de coût $J = SNR_{loss} \times R$ (dB×Slice)

		Base du logarithme b								
WL (bits)	WL	2	2.5	3	3.5	4	4.5	5	Fixed-Point	
6	Q3.2	793.7	426.8	328.8	320.4	332.2	399.1	404.4		
7	Q3.3	247.9	87.7	97.1	100.4	118.2	146.6	154.1	Q6.0	353.7
8	Q3.4	88.8	18.4	47.5	30.8	30.6	34.7	47.5	Q6.1	121.7
9	Q3.5	48.5	6.9	-5.5	22.8	11.6	33.2	27.1	Q6.2	41.0

Au tableau 4.7, nous avons généré la fonction de coût J pour la Log-FFT de 6 bits à 9 bits pour des bases de logarithme allant de 2 à 5. Nous avons aussi calculé la fonction de coût de la FFT à virgule fixe conventionnelle pour une longueur binaire de 7 bits à 9 bits. Ensuite, à la figure 4.12, nous avons représenté les résultats de la Log-FFT en fonction de la longueur binaire et le coût. Chaque courbe représente les résultats des différentes bases de logarithme. Afin de mieux comparer les résultats, nous avons ajouté la courbe de la FFT en virgule fixe conventionnelle.

Dans un premier temps, nous constatons que la croissance de la longueur binaire réduit la fonction de coût pour toutes les méthodes. L'augmentation de la longueur binaire améliore la précision au niveau de la FFT ce qui implique la réduction de perte de performance d'une part. D'autre part, une longueur binaire importante augmente le nombre de Slices. Cependant, d'après les résultats de la figure 4.12, ce dernier argument n'a pas autant d'impact sur la fonction de coût que celle de la réduction de la perte de performance (SNR_{loss}).

Dans un deuxième temps, les meilleurs résultats d'un point de vue « coût le plus faible » appartiennent à la Log-FFT à base 2.5. La Log-FFT à base 2 représente la solution la plus coûteuse par rapport aux autres Log-FFT et se rapproche de la méthode à virgule fixe conventionnelle. Pour la Log-FFT Q3.4 à base 2 et à base 2.5, nous avons les coûts

respectifs de *89 dB×Slices* et *18 dB×Slices*. Cela représente une baisse de *79%*. La FFT à virgule fixe conventionnelle représente la courbe la plus coûteuse. Dans les sections précédentes lors de l'analyse du BER et perte de performance, nous avons retenu les longueurs binaires suivantes : Q3.4 pour la Log-FFT et Q6.2 pour la FFT conventionnelle.

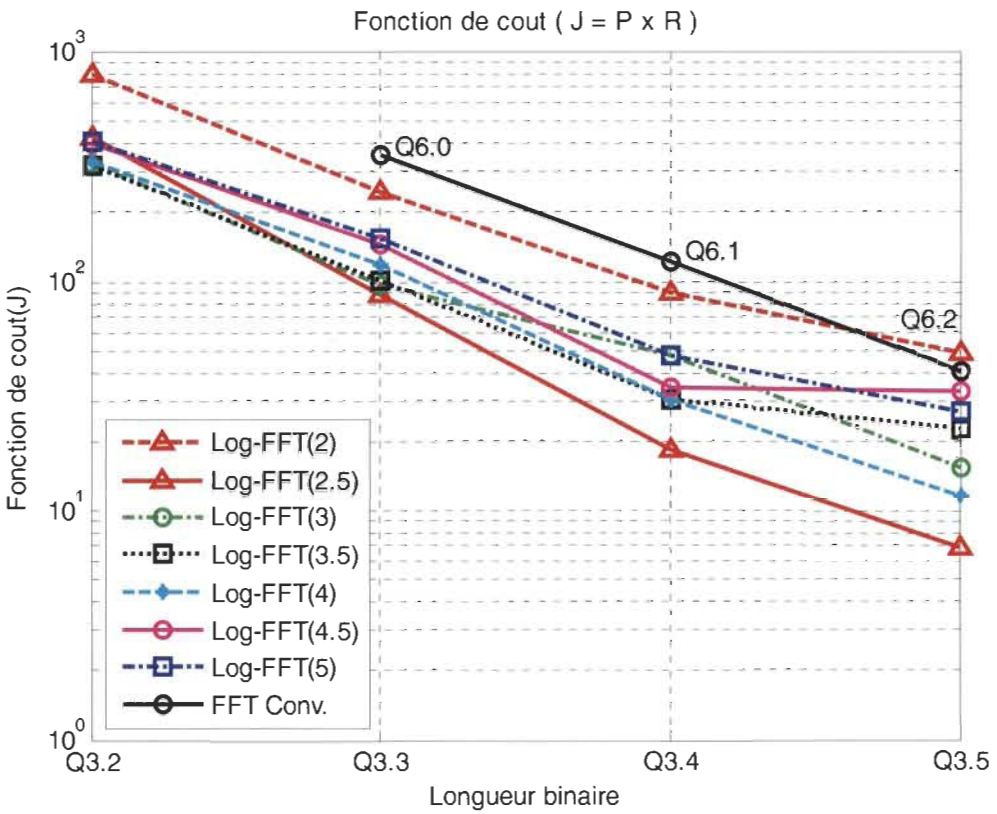


Figure 4.13 La fonction de coût de la FFT conventionnelle et la Log-FFT pour différente base et différentes longueurs binaires

Ces dernières représentent des coûts respectifs de *18 dB×Slices* et *41 dB×Slices*, une réduction significative de *55%* pour la Log-FFT.

4.3.4 Réduction des ressources pour un SNR_{loss} cible

La réduction de ressources est la différence entre le nombre de slices utilisé par les deux approches. Pour effectuer cette différence, nous avons établi une méthodologie pour l'approche conventionnelle et logarithmique. Celle-ci consiste à cibler une perte de performance donnée (SNR_{loss} = *1 dB, 0.5 dB, 0.2 dB, 0.1 dB, etc..*), ensuite dans les tableaux de la perte de performance 4.3 et 4.4, choisir la longueur binaire et la base (pour la Log-FFT) qui minimise ou égalise SNR_{loss} cible, tout en minimisant le nombre de Slices dans le tableau des ressources utilisées 4.5 et 4.6. Autrement dit, pour le premier point qui est une perte de performance de *1dB* nous avons choisi la longueur binaire, dans les tableaux 4.3 ensuite 4.5, qui a un SNR_{loss} inférieur ou égal à *1dB* et en même temps qui minimise le nombre de Slices. Dans ce cas, c'est Q6.1. Nous appliquons la même méthodologie avec la Log-FFT et nous obtenons la longueur binaire Q3.3 avec une base égale à 5. Le nombre de Slice pour ses deux longueurs binaire est *227* et *228* respectivement.

Le tableau 4.7 représente la réduction des ressources observée par la Log-FFT pour des SNR_{loss} constants. À travers ce résultat, nous démontrons que la Log-FFT enregistre un gain en ressource croissant pour chaque réduction du SNR_{loss} cible. Le point important se situe au niveau d'un gain en ressources de *30%* pour une faible perte de performance de l'ordre de *0.02 dB*. Le tableau 4.7 a été généré à l'aide des tableaux 4.3, 4.4, 4.5 et 4.6. Dans le cas où nous ciblons un SNR_{loss} à 0.2dB (cas normalement ciblé en pratique), nous notons que notre proposition apporte une réduction des ressources (Slice) de *10%*. Ce qui représente un gain significatif de réduction des ressources.

Tableau 4.8 Réduction des ressources pour une perte de performance cible SNR_{loss}

Perte (dB)	1	0.5	0.2	0.1	0.05	0.02
Conv_FFT	227	291	291	333	333	373
Log-FFT	228	248	262	289	289	262
Gain	0%	15%	10%	13%	13%	30%

La figure 4.13 présente la mise en graphique du gain de réduction des ressources de notre approche. Noté la croissance non monotone de réduction des ressources avec la réduction de perte de performance, ceci s'explique par l'aspect non linéaire occasionné par les limites et choix de configuration de la Log-FFT selon la base, la longueur binaire ainsi que par la structure d'exploitation des ressources définie en terme de Slice dans le FPGA.

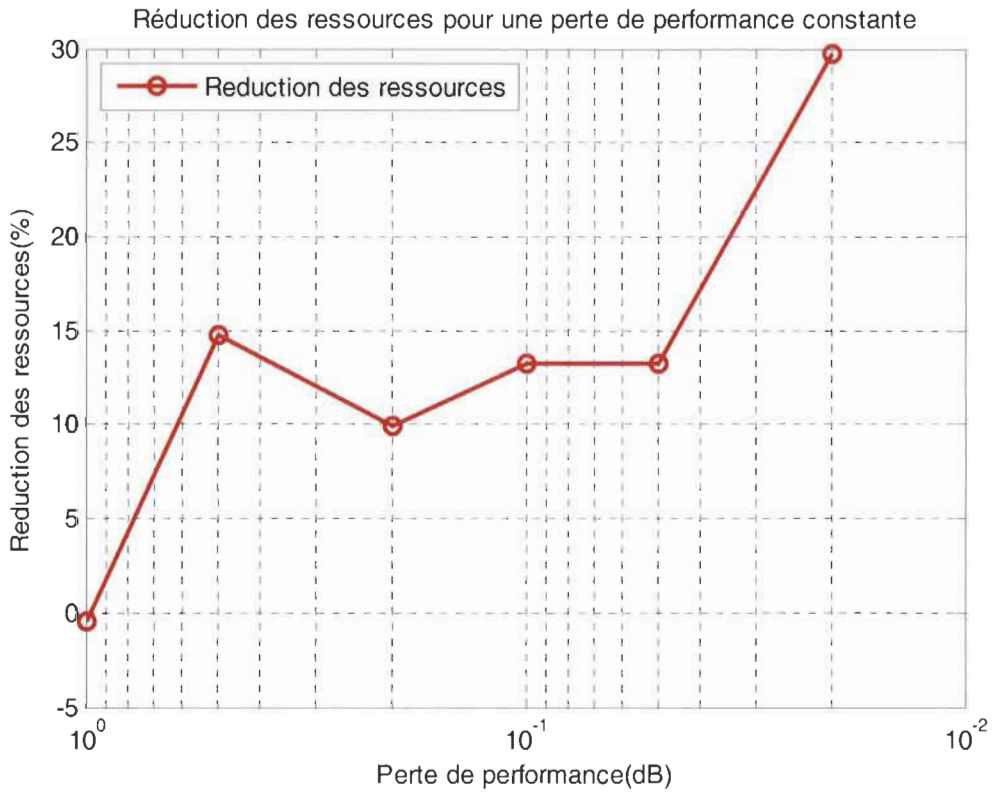


Figure 4.14 Réduction des ressources de la Log-FFT en fonction de la réduction de perte de performance SNR_{loss} de *1 dB, 0.5 dB, 0.2 dB, 0.05dB* et *0.02 dB.*

4.4 Conclusion

Dans ce chapitre nous avons exploré les différentes architectures des opérateurs arithmétiques logarithmiques. Nous avons présenté la Log-lut, l'Antilog-lut, la multiplication et la sommation/soustraction. Nous avons aussi introduit l'architecture VHDL d'une Butterfly radix-2 conventionnelle et d'une Log-Butterfly. Nous avons parcouru les éléments de la Log-Butterfly afin de bien assimiler son fonctionnement. Nous avons présenté dans la première partie de la section de l'implémentation, les différents graphes de BER, perte de performance du SNR et nombre de slices pour chacune des deux approches étudiées.

La FFT conventionnelle, avec une longueur Q6.2, montre un résultat de BER égal à *0.009716*. La perte de performance reliée à cette FFT est égale à *0.14dB* ce qui est très satisfaisant. D'un point de vue implémentation, les ressources utilisées sont égale *291 Slices.*

La Log-FFT affiche des résultats de BER acceptable, *0.0092*, à Q3.4 avec une base de logarithme égale à 2.5. D'un point de vue perte de performance, cette même Log-FFT affiche des pertes de l'ordre de *0.063 dB* ce qui est très faible. Enfin, l'implémentation de cette dernière nécessite *289 Slices*. En dernier lieu, nous avons établi une fonction de coût, qui permet d'estimer le coût de chacune des deux approches, basées sur la perte de performance multipliée par les ressources exprimées en nombre de slices. Cette méthode de comparaison nous a permis de montrer que la Log-FFT est performante, selon les résultats obtenus, avec une réduction de coût de *55%*. De plus, pour une perte de performance de *0.2dB*, notre approche permet une réduction des ressources de 10% par rapport à la FFT virgule fixe conventionnelle.

Chapitre 5 Conclusion

Le domaine des télécommunications est en constante évolution à travers certains secteurs émergents tels que la téléphonie mobile, la radiodiffusion numérique (DVB) et la diffusion de vidéo numérique (DVB). Le succès de ces derniers repose sur l'efficacité spectrale, la robustesse par rapport aux propagations multitrajets, la consommation en puissance et la complexité d'implémentation. Une des techniques modernes, qui a permis d'atteindre ces performances intéressantes, est la modulation multiporteuse OFDM. Elle a suscité l'intérêt pour ses avantages tels que son débit binaire élevé, son efficacité spectrale sans compromettre la largeur de bande et la diminution des interférences inter symbole (ISI).

L'OFDM est un domaine de recherche pertinent pour une mise en application dans les réseaux locaux sans fil et dans des applications de câble à bande large. Il a pour élément clé la FFT au niveau du récepteur. Ce dernier se doit d'être très performant en termes de compromis précision recherchée et ressource d'implémentation puisque les données sont traitées en temps réel. Le défi actuel des ingénieurs en microélectronique, c'est de concevoir un processeur FFT répondant au compromis surface de silicium versus la

précision de calcul. L'algorithme de la FFT le plus utilisé dans l'industrie est la radix-2. L'architecture de ce dernier représente donc l'élément principal de cette étude.

Le traitement numérique du signal est un domaine dans lequel il y a une multitude de manœuvres d'opérations arithmétiques. Certaines d'entre elles, tel que la multiplication, la division, la racine carrée, la puissance, etc.., sont couteuses en ressources matérielles. Le LNS permet, dans certaines situations, de tirer profit de ces opérations complexes. Par exemple, l'opération de multiplication devient une simple sommation et la division devient une simple soustraction. Le système des nombres logarithmiques a été introduit pour remplacer le système linéaire vu ses performances. Ses avantages sont très attrayants pour les algorithmes qui reposent sur des calculs complexes de longueur binaires élevés, ce qui permet une réduction de complexité significative au niveau de l'implémentation matérielle.

Dans ce projet, le LNS est une approche qui a permis de réduire la complexité de la FFT d'une part. D'autre part, la base du logarithme utilisée joue un rôle important dans la réduction des LUTs de conversion. D'ailleurs, l'usage d'une base supérieur à 2 apporte une réduction de la dynamique à l'intérieur des LUTs ce qui a permis une meilleure précision au niveau du calcul de la FFT. Pour rechercher le compromis optimal des deux méthodes à virgule fixe, la FFT conventionnelle et la Log-FFT, nous avons utilisé une fonction de coût *J*. Cette fonction de coût est basée sur le produit des deux paramètres suivants: la perte de performances et le nombre de slices nécessaire pour l'implémentation. L'approche offrant le meilleur compromis performance-complexité est celle qui minimisera ce critère. Nous avons pu démontrer par ces travaux que la Log-FFT

à base 2.5 avec une longueur binaire Q3.4 coute, selon le critère *J*, *55%* de moins que la FFT conventionnelle Q6.2.

De plus, à un taux d'erreur binaire de 1% du récepteur OFDM où nous notons une perte de performance de *0.2dB* du SNR, comparativement à une FFT à virgule flottante, notre approche permet une réduction des ressources de 10% par rapport à la FFT virgule fixe conventionnelle.

Parmi les travaux futurs à réaliser, nous suggérons la réalisation l'approche proposée pour une technologie cible d'un circuit à application spécifique (ASIC – *Application Specific Integrated Circuit*). Ces derniers sont reconnus pour être optimisés selon l'application, ce qui permet de meilleures performances. Dans un deuxième temps, nous voyons une étude plus approfondie dans l'évaluation par la simulation d'une Log-FFT de 2048 points et plus, dans une plateforme OFDM, en utilisant la puissance des ordinateurs parallèles.

Il est connu de la littérature que le milieu le plus avantageux pour le LNS est la manipulation des longueurs binaires élevée. Cependant, nous avons pu démontrer dans le cadre de nos travaux les avantages même dans le cas de faibles longueurs binaires.

Liste des références

[ABE03] K. H. Abed and R. E. Sifred, "CMOS VLSI implementation of a low-power logarithmic converter", *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1421-1433, Nov. 2003.

[ARN01] M. G. Arnold and M. D. Winkel, "A single-multiplier quadratic interpolator for LNS arithmetic," *in Proc. Int. Conf. Comput. Des.: VLSI Comput. Process. (ICCD)*, Washington, DC, p. 178, 2001.

[BEL98] M. Bellanger. *Traitement numérique du signal*. Les éditions Dunod. Paris, France, 1998.

[BRU75] T. A. Brubaker and J. C. Becker, "Multiplication using logarithms implemented with read-only memory," *IEEE Trans. Computers, vol.* C-24, no. 8, pp. 761–766, Aug. 1975.

[CHA68] R.W. Chang, and R.A. Gibby, "Theoretical Study of Performance of an Orthogonal Multiplexing Data Transmission Scheme," *IEEE Transactions on Communications*, vol. 16, n. 4, pp. 529-540, 1968.

[COM65] M. Combet, H. Zonneveld, and L. Verbeek, "Computation of the Base Two logarithm of Binary Numbers," *IEEE Trans. Electronic Computers*, vol. 14, pp.863-867, Dec.1965.

[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.

[COX97] T. M. Schmidl and D. C. Cox. "Robust Frequency and Timing Synchronization for OFDM," *IEEE Trans. on Communications*, vol. 45, Dec. 1997.

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

[FEC99] S. A. Fechtel, A. Blaickner: "Efficient FFT and equalizer implémentation for OFDM receivers," *IEEE Transactions on Consumer Electronics*, vol. 45, no. 4, pp. 1104-1107, Nov. 1999.

[HAL70] E.L. Hall, D.D. Lynch, and S.J. Dwyer III, "Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications," *IEEE Trans. Computers*, vol. 19, pp. 97-105, Feb. 1970.

[HEN02] W. Henkel, G. Taubock, P. Odiing, P. Borjesson, and N. Petersson, "The cyclic prefix of OFDM/DMT - an analysis," *in Intemational Zurich Seminar on Broadband Communications Access, Transmission and Networking*, Zurich, Switzeriand, Feb. 2002.

[HIR80] B. Hirosaki, "*An Analysis of Automatic Equalizers for Orthogonally Multiplexed QAM Systems,*" *IEEE Transactions on Communications*, vol. 28, issue l, pp 73-83, jan. 1980.

[JIA04] Min Jiang, Bing Yang, Yiling Fu, Anping Jiang, Xin-an Wang, Xuewen Gan, Baoying Zhao, Tianyi Zhang. 'Design of FFT processor with Low Power Complex Mutliplier for OFDM-based High-speed Wireless Applications', ISCIT 2004.

[KUN91] M. Kunt *et al. Techniques modernes de traitement numérique des signaux*. Presses polytechniques et universitaires romandes, Lausanne, 1991.

[LAY04] C. Layer, H. J. Pfleiderer, and C. Heer, "A scalable compact architecture for the computation of integer binary logarithms through linear approximation," *in Proc. Int. Symp. Circuits Syst*, pp. 421–424, May 2004.

[LEW89] D. M. Lewis and L. Yu, "Algorithm design for a 30-bit integrated logarithmic processor," *in Proc. 9th Symp. Comput. Arith.*, pp.192–199, Sep. 1989.

[LIM05] Myoung-seob Lim, Jung-yeol Oh, 'Area and Power Efficient Pipeline FFT Algorithm', SIPS 2005.

[MAN06] V. Mahalingam and N. Ranganathan, "An efficient and accurate logarithmic multiplier based on operand decomposition," *in Proc. 19th Int. Conf. VLSI Des. held jointly with 5th Int. Conf. Embedded Syst. Des.(VLSID)*, Washington, DC, pp. 393–398, 2006.

[MAY98] T. May, and H. Rohiing, "Reducing the Peak to Average Power Ratio in OFDM Radio Transmission Systems," *IEEE conference proceedings VTC*, pp. 2474-2478, 1998.

[MIT62] J.N. Mitchell, "Computer multiplication and division binary logarithms," *IRE Trans. Electron. Computers*, vol. 11, pp. 512-517, Aug. 1962.

[MUH04] M. I. Rahman, S. S. Das, F. H. P. Fitzek, "OFDM Based WLAN Systems," *Technical Report R-04-1002*; vol.2 ISBN87- 90834-43-7 ISSN0908-1224, Aalborg University 2004.

[MUL00] S. Muller, *OFDM for Wireless Communications: Nyquist Windowing, Peak-Power Réduction and Synchronization,* Shaker Verlag, 2000.

[NEE00] R.V. Née & R. Prasad, *OFDM for Wireless Multimedia Communications,* Artech House Publishers, 2000.

[NOU08] F. Nougarou, D. Massicotte, M. Ahmed-Ouameur, "Multiple Antennas and Multipass Structure For Adaptive Duplicated Filters and Interference Canceller in WCDMA Systems", IEEE-Int. Conf. on Circuits and Systems for Communications (ICCSC'2008), Shanghai, Chine, Mai 2008.

[NOU11] F. Nougarou, Classe d'algorithmes à faible complexité pour l'annulation d'interférences multiples à base de filtres adaptatifs dupliqués, thèse PhD. en génie électrique, UQTR, 2011.

[PEL80] A. Peled, A. Ruiz, "*Frequency domain data transmission using computational complexity algorithms,*" *ICASSP'80*, vol. 5, pp. 964-967,1980.

[SAN99] S. L. SanGregory, C. Brothers, D. Gallagher, and R. E. Siferd, "A fast lowpower logarithm approximation with CMOS VLSI implementation", in *Proc. IEEE Midw. Symp. Circuits Syst.*, vol. 1, pp. 388–391, Aug. 1999.

[SCH99] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Trans. Computers*, vol. 48, no. 8, pp. 842-847, Aug. 1999.

[SHE95] S. Shephard *et al.*, "Sipmple Coding Scheme to Reduce Peak Factor in QPSK Multicarrier Modulation," *IEEE Elec. Letters*, vol. 31, pp. 1131-1132, Jul. 1995.

[SUG09] S. Paul, N. Jayakumar, S. Khatri, "A Fast Hardware Approach for Approximate, Efficient Logarithm and Antilogarithm Computations, " *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.17, no.2,pp. 269- 277, Feb. 2009.

[TAN91] P. T. P. Tang, "Table-lookup algorithms for elementary functions and their Error analysis," *in Proc. 10th Symp. Comput. Arithmetic*, pp. 232–236, Jun, 1991.

[TAN02] Reduce FFT Memory Reference For Low Power Applications, Yiyan Tang, Yiangtao Jiang, and Yuke Wang Acoustics, Speech and, Signal Processing, 2002 Proceedings (ICASSP'02).

[TAN04] **High-Speed Assembly FFT Implementation with Memory Reference Reduction On DSP Processors,** Yiyan Tang; Wang, Y.; Jin-Gyun Chung;
Song, S.; Lim, M., Electronics, Circuits and Systems, 2004. ICECS 2004.

[TMS03] "TMS320C64x DSP Library Programmer's Reference", Literature Number: SPRU565B, Oct. 2003, (code DSP-radix-2, p. 4-9,4-10).

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

[WEI71] S. B. Weinstein et P. M. Ebert, "*Data transmission by frequency-division multiplexing using the discrète Fourier transform,*" *IEEE Transactions on Communication Technology*, Vol. COM-19, No 5, pp. 628-634, Oct. 1971.

[WEI08] Wei-Hsin Chang and Truong Q. Nguyen. 'On the Fixed-Point Accuracy Analysis of FFT Algorithms'

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

[ZHO09] Bin Zhou, Yingning Peng et David Hwang, 'Pipeline FFT Architectures, Optimized for FPGAs', Hindawi Publishing Corporation, International Journal of Reconfigurable Computing

Annexe

Nous présentons dans cette partie les résultats obtenus lors de la synthèse des circuits de BPE logarithmique et nous les comparons à la BPE conventionnelle. Ces résultats sont classés sous forme de tableaux. Des tableaux A1 à A3, nous trouvons les résultats pour une base logarithmique égale à 2. Des tableaux A4 à A6, nous présentons les résultats pour une base logarithmique de 3. Finalement, des tableaux A7 à A9, nous présentons les résultats pour une base logarithmique de 4. La BPE conventionnelle est synthétisée sans multiplieur câblé (DPS48) afin d'avoir une comparaison équitable entre les deux approches. D'un point de vue ressources matérielles, dans ce projet la Log-BPE enregistre un gain de 10% (voir explication au chapitre 4). Par contre, dans les délais de propagations la BPE conventionnelle enregistre un meilleur résultat vu le chemin critique complexe emprunté par la BPE logarithmique qui nécessite des accès aux LUT pour la réalisation des fonctions f_a et f_b des équations (2.36).

Tableau A.1 Les ressources FPGA pour une Log2-FFT avec 3 bits de fraction

Ressources utilisées (Slices)						
FractionLength	3 bits (à base 2)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	227	333	420	556	662
Log-FFT (x3 FaFb)	161	268	324	388	429	482
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.97	11.02	12.07	14.89	13.45	13.83
Consomation (W)	0.173	0.180	0.176	0.177	0.182	0.18
Gain ressource (%)	-4.3%	-15.3%	2.8%	8.2%	29.6%	37.3%
Gain vitesse (%)	-12.5%	-23.2%	-30.6%	-42.0%	-33.9%	-24.9%

Tableau A.2 Les ressources FPGA pour une Log2-FFT avec 4 bits de fraction

Ressources utilisées (Slices)						
FractionLength	4 bits (à base 2)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	227	333	420	556	662
Log-FFT (x3 FaFb)	160	321	360	448	470	520
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	9.36	12.60	14.05	15.19	14.82	14.86
Consomation (W)	0.175	0.181	0.177	0.177	0.18	0.183
Gain ressource (%)	-3.8%	-29.3%	-7.5%	-6.3%	18.3%	27.3%
Gain vitesse (%)	-16.1%	-32.9%	-40.4%	-43.2%	-40.0%	-30.1%

Tableau A.3 Les ressources FPGA pour une Log2-FFT avec 6 bits de fraction

Ressources utilisées (Slices)						
FractionLength	6 bits (à base 2)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	227	333	420	556	662
Log-FFT (x3 FaFb)	118	305	549	708	708	756
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.63	11.61	14.99	16.15	16.30	16.58
Consomation (W)	0.172	0.179	0.176	0.177	0.180	0.182
Gain ressource (%)	30.5%	-25.6%	-39.3%	-40.7%	-21.5%	-12.4%
Gain vitesse (%)	-9.0%	-27.1%	-44.1%	-46.6%	-45.5%	-37.3%

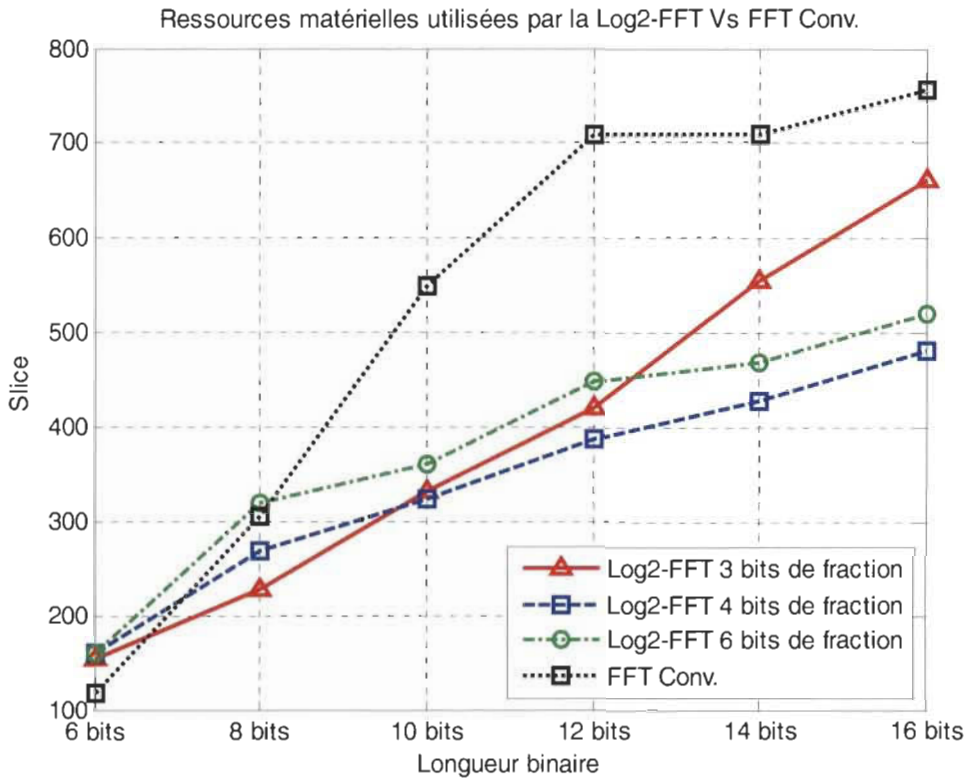


Figure A.1 Ressources matérielles de la Log2-FFT Vs FFT Conv.

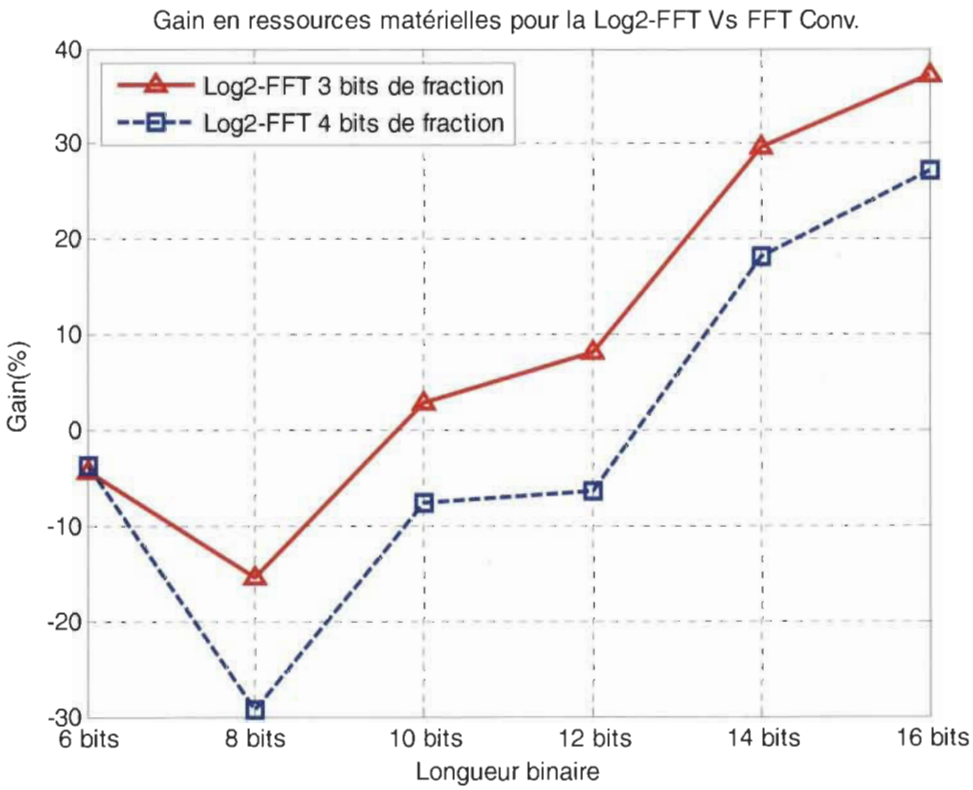


Figure A.2 Gain en ressources matérielles de la Log2-FFT Vs FFT Conv.

Tableau A.4 Les ressources FPGA pour une Log3-FFT avec 3 bits de fraction

Ressources utilisées (Slices)						
FractionLength	3 bits (à base 3)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	154	236	284	338	392	443
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.80	10.78	11.49	14.23	13.07	14.36
Consomation (W)	0.173	0.176	0.178	0.179	0.180	0.18
Gain ressource (%)	0.0%	-1.7%	18.0%	24.3%	41.8%	49.4%
Gain vitesse (%)	-10.8%	-21.5%	-27.1%	-39.4%	-32.0%	-27.6%

Tableau A.5 Les ressources FPGA pour une Log3-FFT avec 4 bits de fraction

Ressources utilisées (Slices)						
FractionLength	4 bits (à base 3)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	162	262	341	385	430	481
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	9.6	12.10	12.25	15.39	14.48	14.66
Consomation (W)	0.173	0.177	0.176	0.175	0.182	0.181
Gain ressource (%)	-4.9%	-11.5%	-1.8%	9.1%	29.3%	37.6%
Gain vitesse (%)	-18.2%	-30.1%	-31.6%	-43.9%	-38.6%	-29.1%

Tableau A.6 Les ressources FPGA pour une Log3-FFT avec 6 bits de fraction

Ressources utilisées (Slices)						
FractionLength	6 bits (à base 3)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	118	304	484	588	604	655
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.63	11.83	14.84	16.01	16.04	16.53
Consomation (W)	0.172	0.176	0.176	0.179	0.179	0.181
Gain ressource (%)	30.5%	-23.7%	-30.8%	-28.6%	-7.9%	1.1%
Gain vitesse (%)	-9.0%	-28.5%	-43.5%	-46.1%	-44.6%	-37.1%

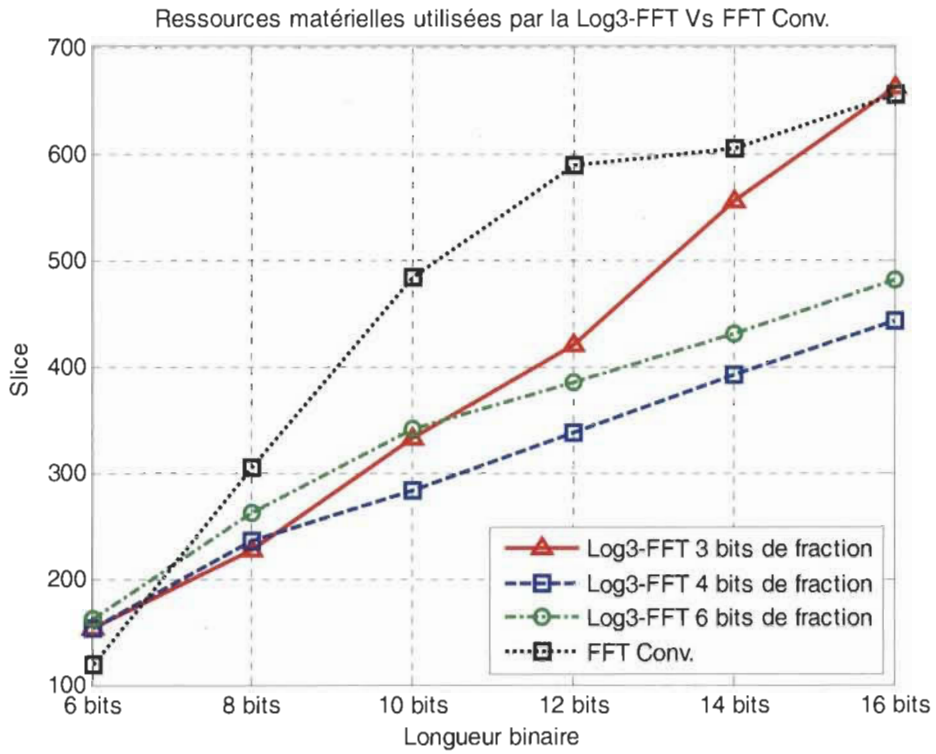


Figure A.3 Ressources matérielles de la Log3-FFT Vs FFT Conv.

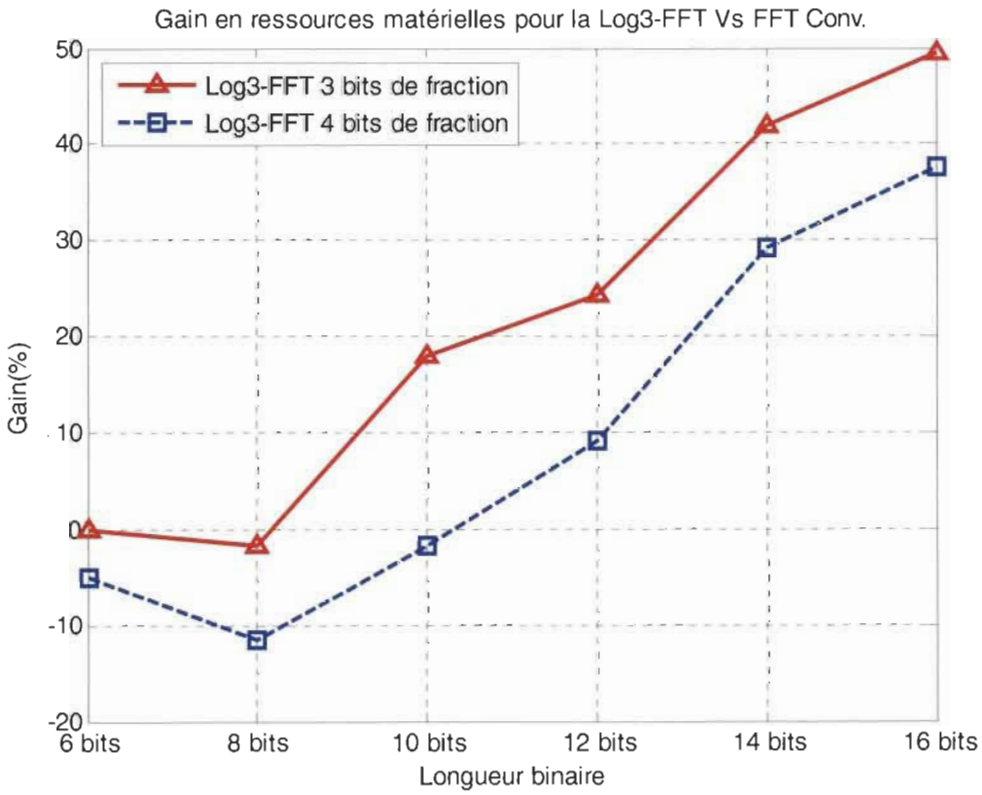


Figure A.4 Gain en ressources matérielles de la Log3-FFT Vs FFT Conv.

Tableau A.7 Les ressources FPGA pour une Log4-FFT avec 3 bits de fraction

Ressources utilisées (Slices)						
FractionLength	3 bits (à base 4)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	159	225	267	336	396	445
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	9.27	10.85	11.85	15.27	14.33	14.47
Consomation (W)	0.173	0.179	0.176	0.177	0.185	0.18
Gain ressource (%)	-3.1%	3.1%	25.5%	25.0%	40.4%	48.8%
Gain vitesse (%)	-15.3%	-22.0%	-29.3%	-43.5%	-38.0%	-28.2%

Tableau A.8 Les ressources FPGA pour une Log4-FFT avec 4 bits de fraction

Ressources utilisées (Slices)						
FractionLength	4 bits (à base 4)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	161	268	324	388	429	482
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.97	11.02	12.07	14.89	13.45	13.83
Consomation (W)	0.173	0.180	0.176	0.177	0.182	0.18
Gain ressource (%)	-4.3%	-13.4%	3.4%	8.2%	29.6%	37.3%
Gain vitesse (%)	-12.5%	-23.2%	-30.6%	-42.0%	-33.9%	-24.9%

Tableau A.9 Les ressources FPGA pour une Log4-FFT avec 6 bits de fraction

Ressources utilisées (Slices)						
FractionLength	6 bits (à base 4)					
WordLength	6 bits	8 bits	10 bits	12 bits	14 bits	16 bits
Conv-FFT(Sans Dsp48)	154	232	335	420	556	662
Log-FFT (x3 FaFb)	130	330	441	543	562	612
Délai Conv-FFT (ns)	7.85	8.46	8.38	8.63	8.89	10.39
Délai Log-FFT (ns)	8.06	11.70	14.18	15.67	14.71	15.20
Consomation (W)	0.174	0.180	0.176	0.180	0.182	0.181
Gain ressource (%)	18.5%	-29.7%	-24.0%	-22.7%	-1.1%	8.2%
Gain vitesse (%)	-2.6%	-27.7%	-40.9%	-44.9%	-39.6%	-31.6%

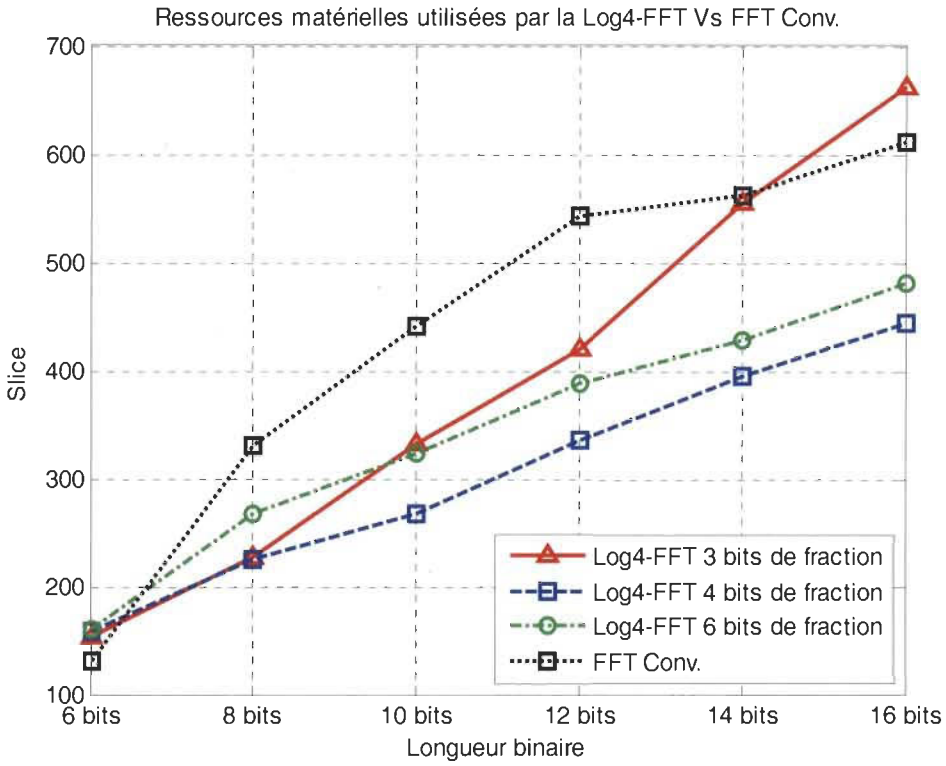


Figure A.5 Ressources matérielles de la Log4-FFT Vs FFT Conv.

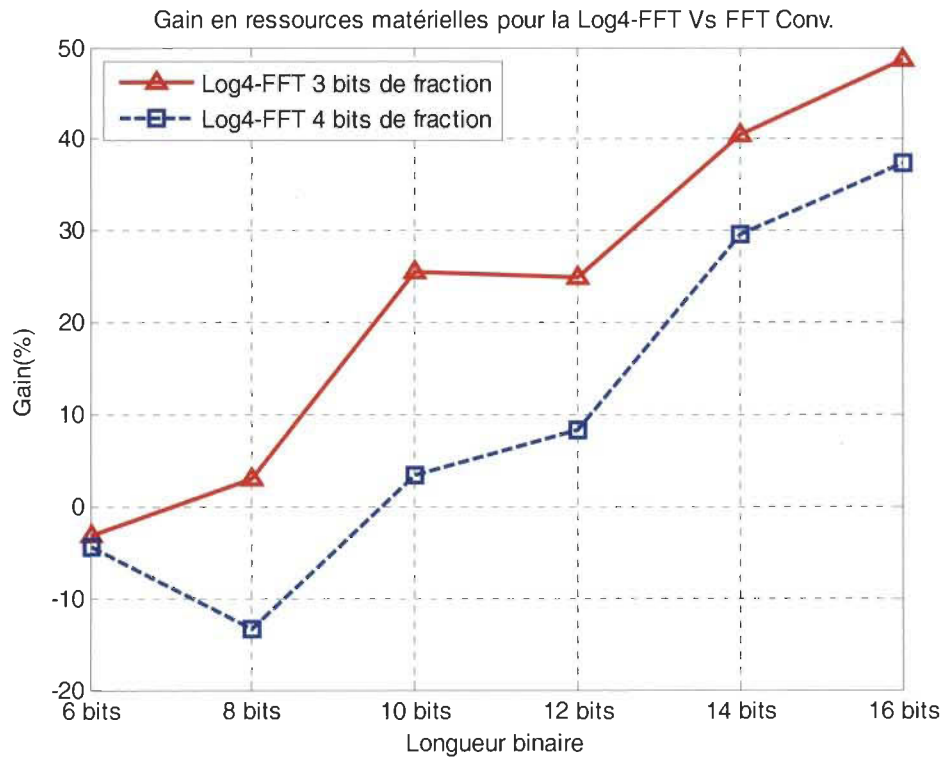


Figure A.6 Gain en ressources matérielles de la Log4-FFT Vs FFT Conv.