République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Mise en œuvre d'une architecture améliorée d'un processeur FFT sur FPGA

Zohra RABET

Katia Rania BITAM

Sous la direction de Mr. Mohamed Taghi Enseignant chercheur à ENP, Alger

Présenté et soutenu publiquement le 25/06/2023 auprès des membres du jury :

Président	M. Chrif	Larbes	Prof.	ENP, Alger
Promoteur	Mr. Mohamed	Taghi	Enseignant chercheur.	ENP, Alger
Examinatrice	Mme. Latifa	Hammami	Prof.	ENP, Alger

ENP 2023

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

Département électronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Mise en œuvre d'une architecture améliorée d'un processeur FFT sur FPGA

Zohra RABET

Katia Rania BITAM

Sous la direction de Mr. Mohamed Taghi Enseignant chercheur à ENP, Alger

Présenté et soutenu publiquement le 25/06/2023 auprès des membres du jury :

Président	M. Chrif	Larbes	Prof.	ENP, Alger
Promoteur	Mr. Mohamed	Taghi	Enseignant chercheur.	ENP, Alger
Examinatrice	Mme. Latifa	Hammami	Prof.	ENP, Alger

ENP 2023

10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.
www.enp.edu.dz

ملخص

تحويل فورييه سريع هو خوارزمية عددية تُستخدم لحساب تحويل فورييه متسلسل لإشارة بكفاءة. يتيح لنا تحليل الإشارة إلى مكونات ترددية مختلفة. نظرًا لأن تحويل فورييه سريع يكمن في أساس خوارزمية تحجيم الشرارة المستخدمة في معالجة رادار الفتحة الاصطناعية (SAR) ، فمن الضروري تقييم الخوارزمية وتعقيدها الحسابي لتصميم هندسة تركيبية فعالة لتحويل فورييه سريع.المساهمة الرئيسية لهذا البحث تكمن في تصميم معالج FFT مبني على مجمع مبني على المجال القابل لإعادة التشكيل بسرعة وكفاءة الموارد ، ويمكن تكوينه بعدد متغير من النقاط N N) $=8^k$).يظهر الدراسة المقارنة التي أجريت بين معالج FFT المقترح والمقالات المرجعية الأخيرة بشكل مرضٍ فعالية هذه الهندسة الجديدة.

كلمات مفتاحية : FPGA. SAR, FFT,

Abstract

The Fast Fourier Transform is a numerical algorithm used to efficiently compute the discrete Fourier transform of a signal. It allows for the decomposition of a signal into its frequency components.. Since the Fast Fourier Transform lies at the core of the chirp scaling algorithm used in synthetic aperture radar (SAR) processing, it is crucial to evaluate the algorithm and its computational complexity in order to design an optimal FFT hardware architecture. The main contribution of this research lies in the design of an FPGA-based FFT processor that is both fast, resource-efficient, and can be configured with a variable number of points, N ($N = 8^k$). The comparative study conducted between the proposed FFT processor and recent reference articles demonstrates favorably the efficiency of this new architecture in terms of hardware resource consumption.

Keywords : FFT, SAR, FPGA.

Résumé

La Transformée de Fourier Rapide (FFT) est un algorithme numérique utilisé pour calculer efficacement la transformée de Fourier discrète (TFD) d'un signal. Elle permet de décomposer un signal en ses composantes fréquentielles. Étant donné que la transformée de Fourier rapide (FFT) est au cœur de l'algorithme chirp scaling utilisé dans le traitement du radar à synthèse d'ouverture SAR, il est primordial d'évaluer l'algorithme ainsi que sa complexité de calcul afin de concevoir une architecture matérielle FFT optimale. La principale contribution de cette recherche réside dans la conception d'un processeur FFT sur FPGA qui est à la fois rapide, efficient en termes de ressources, et qui peut être configuré avec un nombre variable de points, N ($N = 8^k$). L'étude comparative menée entre le processeur FFT proposé et les articles de référence récents démontre de manière favorable l'efficacité de cette nouvelle architecture en terme de consommation de ressources matérielles.

Mots clés : FFT, SAR, FPGA.

Dédicace

“

À ma maman extraordinaire,

Ton amour inconditionnel, ta force de combattante et tes innombrables sacrifices ont été les piliers de ma vie. Sans toi, je ne serais pas la personne que je suis aujourd'hui. Tu es ma source de motivation constante, mon modèle de détermination et ma plus grande inspiration. Cette dédicace est une humble façon de te témoigner toute ma gratitude pour tout ce que tu as fait et continue de faire pour moi. Je t'aime infiniment, ma maman bien-aimée.

À ma soeur.D,

Au-delà du lien du sang, tu occupes une place spéciale dans mon cœur. Tu es ma confidente la plus fidèle, ma partenaire de crime et ma meilleure amie. Peu importe les hauts et les bas, je sais que je peux toujours compter sur toi pour être là, prête à m'écouter, me conseiller et m'encourager. Tu as su me réconforter lorsque j'étais triste, me soutenir lorsque j'étais découragé, et me célébrer lorsque j'ai réussi.

À mon frére,

Ta fierté pour mes accomplissements renforce ma détermination et me pousse à aller toujours plus loin. Tu célèbres mes victoires avec une sincérité contagieuse, me rappelant constamment que mes succès sont aussi les tiens. Ta persévérance face aux défis et ta détermination à surmonter les obstacles m'ont montré qu'avec une volonté solide, rien n'est impossible.

À Célia,

ma meilleure amie, mon amie d'enfance, Merci pour toutes ces années d'amitié si précieuse. Ta présence a apporté tant de bonheur et de réconfort dans ma vie. Tu es une personne exceptionnelle, et je suis heureuse de t'avoir à mes côtés.

À ma chère promo,

"ELN MATCH", avec laquelle j'ai partagé des moments mémorables. Merci d'avoir rendu cette expérience inoubliable.

À mes amis,

Hafsa, Hadjer, Ryhame, Idriss, Zaki et Walid qui ont été présents à mes côtés et m'ont apporté leur aide tout au long de mon parcours scolaire.
Merci.

”

- Katia Rania Bitam

“

À ma chère mère,

À mon cher père,

À mes deux adorables grand-mères,

Qui n'ont jamais cessé, de formuler des prières à mon égard, de me soutenir et de m'épauler pour que je puisse atteindre mes objectifs.

À mes chers frères, Mohammed et Hacen

À mes sœurs bien-aimées, Khaoula, Douaa et Khadidja,

Pour ses soutiens moral, leurs aides précieuses et supports dans les moments difficiles et leurs conseils précieux tout au long de mon parcours.

À toute ma famille,

À ma chére promos "ELN MATCH"

À tous mes autres ami(e)s, A tous ceux que j'aime et ceux qui m'aiment.

Merci !

”

- Zohra Rabet

Remerciements

Nous tenons à exprimer notre gratitude à notre encadrant de projet de fin de mémoire Mr.Taghi, pour son soutien tout au long de ce mémoire. Sa guidance, sa disponibilité et ses conseils.

Nous remercions tout particulièrement les membres du jury qui ont accepté de consacrer leur temps à examiner ce travail : Monsieur Cherif LARBES, Professeur à l'Ecole Nationale Polytechnique et Madame Latifa HAMMAMI, Professeur à l'Ecole Nationale Polytechnique.

Enfin, nous tenons à adresser nos remerciements et notre gratitude à tous nos professeurs de l'Ecole Nationale Polytechnique, qui nous ont guidé tout au long de nos cinq années d'études.

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

FFT *Fast Fourrier Transform.*

FPGA *Field Programmable Gate Array.*

SAR *Synthetic Aperture Radar*

CSA *Chirp Scaling Algorithm.*

DFT *Discrete Fourrier Transform.*

DIT *Decimation In Time.*

DIF. *Decimation In Frequency.*

MSB *Most Significant Bit.*

LSB *Least Significant Bit.*

SFG *Signal Flow Graph.*

BF *Butterfly*

MDC *Multi-path Delay Commutator.*

SDF *Single-path Delay Feedback.*

SDC *Single-path Delay Commutator.*

LUT *Look Up Table.*

DSP *Digital Signal Processing.*

ROM *Read Only Memory.*

RAM *Random Access Memory.*

UART *Universal Asynchronous Receiver Transmitter.*

FSM *Finite State Machine.*

VHDL *VHSIC Hardware Description Language.*

CORDIC *COordinate Rotation DIgital Computer.*

RSBQ *Rapport Signal sur Bruit de Quantification.*

SQNR *Signal to Quantization Noise Ratio.*

Introduction générale

La Transformée de Fourier est un outil fondamental dans le domaine du traitement du signal. Cet algorithme permet de convertir un signal du domaine temporel en son équivalent dans le domaine fréquentiel fournissant ainsi des informations utiles sur les signaux, telles que leurs spectres fréquentiels, leurs spectre de phase, leurs amplitudes etc… Cette transformation est essentielle pour diverses applications, telles que l'analyse spectrale, le filtrage, la compression et le traitement d'images.

À l'ère actuelle, il existe une demande croissante de calculs à grande échelle et de traitement en temps réel dans différents domaines. La FFT, dans ce contexte, a fait l'objet de beaucoup de travaux ayant menés à la conception d'algorithmes et d'architectures en vue d'aboutir à des réalisations efficaces en termes de rapidité, d'encombrement et de consommation. Des avancées significatives ont été réalisées dans l'accélération des calculs de la transformée de Fourier rapide grâce à l'utilisation de processeurs basés sur des matrices de portes programmables (FPGA). Ces circuits reconfigurables présentent de nombreux avantages, tels que leur flexibilité, leur capacité de traitement parallèle et de calcul rapide, ce qui en fait un choix idéal pour accélérer l'exécution de l'algorithme FFT.

Dans le cadre de ce projet de fin d'études, nous nous concentrons sur le développement d'une implémentation de la Transformée de Fourier rapide spécifiquement adaptée aux applications de Radar à ouverture synthétique. Le radar à ouverture synthétique est une technique de télédétection qui permet de générer des images détaillées de la surface terrestre en utilisant des signaux radar. Au sein de ces systèmes, le traitement FFT joue un rôle essentiel dans la formation des images et l'analyse des données. Il est, donc, intéressant d'entreprendre la mise en œuvre d'un processeur FFT capable de répondre aux exigences de traitement en temps réel.

Notre objectif est de développer, en s'appuyant sur les connaissances et les techniques existantes, un processeur FFT spécifiquement adapté aux applications SAR. Nous œuvrerons, dans cette optique, à optimiser les performances d'un tel processeur en termes de rapidité, des ressources mobilisées dans l'architecture cible, et ses capacités de traitement en temps réel. Cela, contribuera, à coup sûr , à l'amélioration la technologie du radar à ouverture synthétique

.

Contribution

- Un processeur FFT générique de N points (N= 8^k) dont la taille des données est variable. Cette flexibilité permet l'adaptation du système à différents besoins.
- Un schéma de multiplication a été élaboré afin de contourner l'utilisation des blocs DSP et des blocs ROM pour des FFT avec un grand nombre de points.
- Une comparaison a été réalisée entre le processeur FFT conçu et les travaux précédents, révélant que le processeur proposé mobilise moins de ressources matérielles.

Structure du mémoire

Le premier chapitre de ce mémoire est consacrée à la présentation des concepts fondamentaux utilisés dans la transformée de Fourier. Une étude approfondie des différents algorithmes FFT est réalisée. Les performances de chaque algorithme sont comparées en terme de consommation des ressources matérielle afin d'identifier ceux qui conviennent le mieux à notre implémentation. Ce chapitre contiendras également un état de l'art des recherches antérieures.

Le deuxième chapitre présente une analyse complète de la mise en œuvre matérielle des processeurs FFT et IFFT avec une explication détaillée de l'approche adoptée.

Dans le troisième chapitre, des approches spécifiques seront étudiées dans le but de réduire au minimum les ressources matérielles nécessaires lors de l'implémentation de la FFT. Une discussion détaillée de ces techniques est proposée, permettant ainsi d'explorer les voies menant à l'amélioration de l'efficacité du processeur FFT.

Enfin, le quatrième chapitre est réservé à la partie évaluations et expérimentations où sont présentées les performances du processeur proposé. Une comparaison est établie à des fins de validation avec les résultats obtenus à l'aide de MATLAB, ainsi que ceux concernant l'utilisation des ressources matérielles des Processeurs FFT décrits dans la littérature.

Chapitre 1

Transformée de fourier rapide pour les applications SAR

1.1 Définition

SAR (Synthetic Aperture Radar) est une technologie de télédétection utilisée pour l'observation de la Terre à partir de satellites, d'avions ou de drones. Contrairement aux systèmes de radar traditionnels qui envoient des impulsions électromagnétiques et mesurent le temps qu'il faut pour les réfléchir, le SAR envoie une série d'impulsions électromagnétiques tout en se déplaçant, ce qui crée une "ouverture synthétique". En analysant les ondes réfléchies, le SAR peut générer des images détaillées de la surface terrestre.

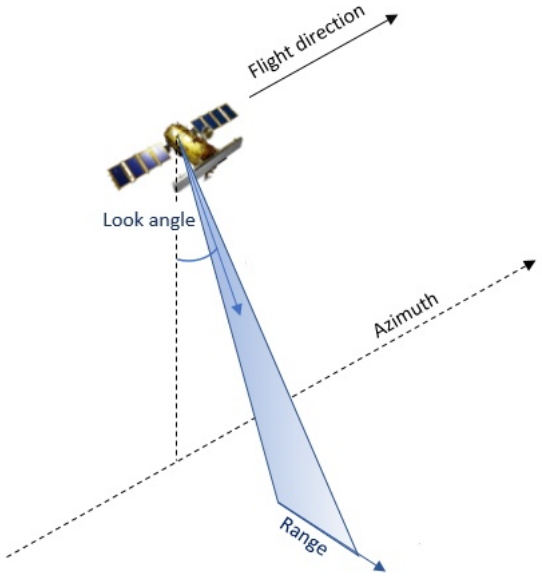


FIG. 1.1 : Radar à synthèse d'ouverture[1]

Dans le contexte du Radar à ouverture synthétique, les termes "range" et "azimuth" se réfèrent à deux directions importantes dans lesquelles les données SAR sont collectées et traitées.

Direction de portée (Range direction) : La direction de portée dans le SAR fait référence à la direction le long de la ligne de visée entre l'antenne radar et la surface terrestre. Dans cette direction, le radar mesure la distance entre l'antenne et la cible ou la surface terrestre. La direction de portée est perpendiculaire à la direction de vol de la plate-forme SAR (par exemple, un avion ou un satellite).
Direction d'azimut (Azimuth direction) : La direction d'azimut dans le SAR fait référence à la direction parallèle à la surface terrestre, généralement le long de la direction de vol de la plate-forme SAR.Dans cette direction, le radar mesure la position angulaire de la cible ou de la surface terrestre par rapport à la direction de vol de la plate-forme SAR.

Ensemble, les directions de portée et d'azimut forment un système de coordonnées bidimensionnel qui décrit la position spatiale des cibles ou de la surface terrestre dans les données SAR.

1.2 Algorithme Chirp scaling

L'algorithme chirp scaling est l'un des nombreux algorithmes de traitement SAR disponibles. Comme illustré dans la figure 1.2, cet algorithme comprend plusieurs étapes qui consistent à convertir le signal du domaine temporel au domaine fréquentiel (et vice versa), ainsi qu'à effectuer une multiplication par une matrice de coefficients essentielle pour la formation d'images.
Comme évoqué précédemment, les données SAR se présentent sous la forme de signaux bidimensionnels, l'algorithme reçoit donc une matrice Na x Nr, ou Na représente le nombre de points d'échantillonnage des données d'écho dans la direction azimutale, et Nr représente le nombre de point d'échantillonnage des données d'écho dans la direction de portée.

Il convient de souligner que notre mémoire ne se concentre pas spécifiquement sur la matrice de coefficients et à son processus d'acquisition. Pour une analyse plus détaillée, nous vous recommandons de consulter l'article , qui traite spécifiquement cette question. Dans la section suivante, nous examinerons la proportion de FFT dans cet algorithme afin de mettre en évidence l'importance de l'implémentation d'un processeur FFT rapide pour les calculs [2].

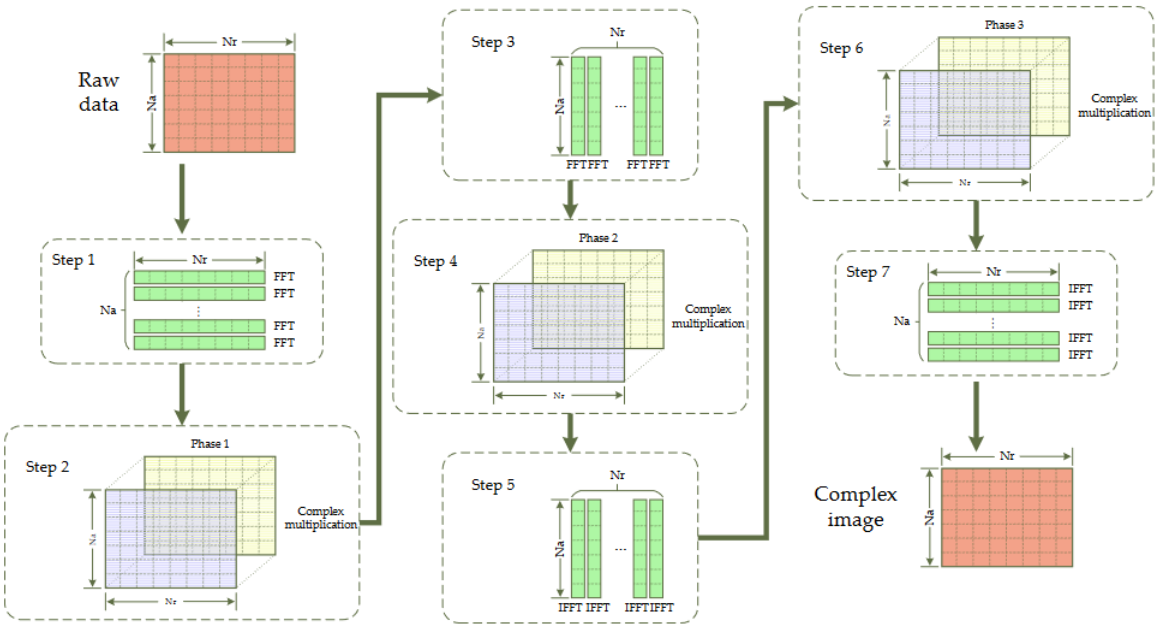


FIG. 1.2 : Diagramme de flux de l'algorithme chirp scaling [3].

1.3 FFT/ IFFT dans l'algorithme chirp scaling

Après avoir analysé les différentes étapes d'exécution de l'algorithme CS, il est clair que ce dernier utilise de manière significative la FFT (Transformée de Fourier rapide) et l'IFFT (Transformée de Fourier inverse rapide) pour le traitement des données du radar à ouverture synthétique.

La proportion W de FFT (IFFT) peut être définie comme le pourcentage des opérations de multiplication et d'addition réelles effectuées lors de l'opération FFT.
Selon le Tableau (1.1) [4], la valeur de la proportion W de la FFT varie légèrement en fonction de la taille de la matrice d'imagerie utilisée. Généralement, on constate que W est supérieur à 90% et peut atteindre jusqu'à 95% lorsque la taille de la matrice augmente.

Taille de l'image	256× 256	1024× 1024	4096× 4096	16384× 16384	65536× 65536
Charge de calcul FFT	10^7	2.1×10^8	4.1×10^9	7.6×10^{10}	1.2×10^{12}
Charge de calcul de compensation de phase	10^6	1.8×10^7	3×10^8	4.8×10^9	1.2×10^{10}
W-Value	89.8%	91.7%	93%	94%	94.7%

Tab. 1.1 : Proportion de la FFT pour différentes tailles d'images.

Par conséquent, l'utilisation d'un processeur FFT spécialisé, conçu spécifiquement pour répondre aux exigences du CSA, permettra une exécution efficace des opérations FFT/IFFT, réduisant ainsi le temps de calcul et améliorant les performances globales de l'algorithme. En utilisant des ressources matérielles dédiées, le processeur spécialisé peut gérer efficacement les calculs FFT à grande échelle impliqués dans l'imagerie SAR. Cela accélère le traitement des données et le temps d'imagerie, permettant des applications SAR en temps réel ou quasi temps réel.

Chapitre 2

Transformée de Fourier Rapide : Concepts et état de l'art

2.1 Introduction

L'objectif de ce chapitre est de fournir une compréhension approfondie des concepts fondamentaux et de l'état de l'art de la FFT. En explorant les différents algorithmes, les architectures et les avancées récentes, nous serons en mesure de prendre des décisions éclairées pour la conception et l'implémentation de la FFT dans notre mémoire.

2.2 Transformée de Fourier rapide

La transformée de fourier rapide est un algorithme efficace qui consiste à calculer la transformée de fourier discrete d'une façon rapide. Cet algorithme permet la réduction du nombre d'opérations, en particulier le nombre des multiplications et d'additions complexes [5] ce qui entraîne un temps de traitement plus court .
L'algorithme FFT accélère le calcul de la transformée de Fourier discrète d'un signal à N points en utilisant la technique "diviser pour régner". Cet algorithme divise le signal en sous-signaux de taille plus petite, puis combine les résultats obtenus pour obtenir la DFT complète.
Mathématiquement, le calcule de la DFT consiste à effectuer la somme des échantillons du signal x_n multipliés par les facteurs de phases W_N^{nr} .Ce calcul doit effectuer N^2 multiplications complexes et N(N-1) additions complexes .

$$X_r = \sum_{n=0}^{N-1} x_n W_N^{nr}, 0 \leq k \leq N-1, \tag{2.1}$$

$$W_N^{nr} = e^{-2\pi i \frac{nr}{N}}$$

tel que :

- N le nombre des échantillons .
- X_r : sont les échantillons de signal dans le domaine fréquentiel ..
- x_r sont les échantillons de signal dans le domaine temporel .
- W_N^{rn} sont les facteurs de phases.

Avec les algorithms de calcule de la fft ces nombres sont réduits à moins de $\frac{N}{2} log_2(N)$ opérations de multiplications et jusqu'à $N log_2(N)$ additions complexes [6].

2.3 Algorithmes

2.3.1 Décimation dans le temps DIT

L'algorithme décimation dans le temps DIT sert à diviser le signal d'entrée en deux séquences paire et impaire . les équations (2.2) montrent les expressions mathématiques utilisées pour diviser le signal d'entrée en utilisant l'algorithme radix 2 DIT .

$$
\begin{aligned}
X_r &= \sum_{n=0}^{N/2-1} x_{2n} \cdot W_N^{r(2n)} + \sum_{n=0}^{N/2-1} x_{2n+1} \cdot W_N^{r(2n+1)} \\
X_r &= \sum_{n=0}^{N/2-1} x_{2n} \cdot W_{N/2}^{rn} + \sum_{n=0}^{N/2-1} x_{2n+1} \cdot W_{N/2}^{rn} W_N^r \\
X_r &= \sum_{n=0}^{N/2-1} x_{2n} \cdot W_{N/2}^{rn} + W_N^r \sum_{n=0}^{N/2-1} x_{2n+1} \cdot W_{N/2}^{rn}
\end{aligned}
\tag{2.2}
$$

L'équation (2.3) montre que la transformée de Fourier discrète (DFT) est périodique. En effet, seule la moitié de la longueur de la DFT doit être calculée. Les facteurs de phase sont également périodiques, c'est-à-dire que $W_N^r = W_N^{r+N}$, comme le montre l'équation 2.2. Cette périodicité est exploitée pour accélérer le processus de calcul de la DFT.

$$
\begin{aligned}
&\sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{r+N}{N}} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{r}{N}} e^{-2\pi i n} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i n \frac{r}{N}} \\
&e^{-2\pi i n} = 1
\end{aligned}
\tag{2.3}
$$

2.3.2 Décimation en fréquence DIF

Une autre façon de calculer la transformée de fourier discrete DFT consiste a utiliser l'algorithme de décimation en fréquence DIF [7]. Cet algorithme divise la formule de la DFT en deux sommations, où l'une s'applique à la première moitié des entrées [0, N/2-1], tandis que l'autre s'applique à la seconde moitié [N/2, N-1]. Cette division est montrée dans les équations (2.4) :

$$
\begin{aligned}
X_r &= \sum_{n=0}^{N/2-1} x_n \cdot W_N^{rn} + \sum_{n=N/2}^{N-1} x_n \cdot W_N^{rn} \\
X_r &= \sum_{n=0}^{N/2-1} x_n \cdot W_N^{rn} + W_N^{nr/2} \sum_{n=0}^{N/2-1} x_{n+\frac{N}{2}} \cdot W_N^{rn} \\
X_r &= \sum_{n=0}^{N/2-1} \left(x_n + (-1)^r \cdot x_{n+\frac{N}{2}} \right) W_N^{rn} \\
W_N^{Nr/2} &= (-1)^r
\end{aligned}
\tag{2.4}
$$

Le graphe de butterfly de l'algorithme DIF est très similaire à celui d'un butterfly DIT. La principale différence réside dans le fait que la multiplication par les facteurs de phase est effectuée à la fin du butterfly au lieu du début.

2.3.3 Inversion de bits

L'inversion de bits consiste à inverser l'ordre des bits dans un mot binaire de gauche à droite.[8] Ainsi, les bits les plus significatifs (MSB) deviennent les moins significatifs (LSB) et les LSB deviennent les MSB.

Pour un nombre de points N= 2^n, cela signifie que chaque échantillon avec un indice i sera échangé avec l'échantillon ayant l'indice correspond à l'inversion des bits de i .
Par exemple pour des entrées de taille N= 8 ayant les indice suivant (0,1,2,3,4,5,6,7), l'algorithme d' Inversion de bits réorganise les échantillons comme suit (0,4,2,6,1,5,3,7)

$$\begin{aligned}
0 \rightarrow (000) &< \text{ bit-reversal } > (000) \rightarrow 0 \\
1 \rightarrow (001) &< \text{ bit-reversal } > (100) \rightarrow 4 \\
2 \rightarrow (010) &< \text{ bit-reversal } > (010) \rightarrow 2 \\
3 \rightarrow (011) &< \text{ bit-reversal } > (110) \rightarrow 6 \\
4 \rightarrow (100) &< \text{ bit-reversal } > (001) \rightarrow 1 \\
5 \rightarrow (101) &< \text{ bit-reversal } > (101) \rightarrow 5 \\
6 \rightarrow (110) &< \text{ bit-reversal } > (011) \rightarrow 3 \\
7 \rightarrow (111) &< \text{ bit-reversal } > (111) \rightarrow 7
\end{aligned}$$

Dans le cas d'une FFT DIT (Décimation In Time), les entrées doivent être arrangées selon l'ordre inverser avant d'effectuer les calculs de la FFT .En revanche, dans une FFT DIF (Décimation In Frequency), les résultats de la FFT doivent être réarrangés selon l'ordre du bit-reversal après avoir effectué les calculs.

2.3.4 Radix 2

Le Radix-2 est un algorithme de calcul de la FFT, qui requiert que le nombre d'échantillons N soit a base de 2.
L'algorithme radix-2 DIF est basé sur la décomposition de la séquence de sortie en paquets de longueur 2 , le premier contient des données d'indices pairs **X(2r)** et l'autre contient des données d'indices impaire **x(2r+1)** comme indiquée dans les équations (2.5) (2.6).

$$X_{2r} = \sum_{\kappa=0}^{N-1} x_n W_N^{n(2r)}, \quad \text{r} = 0, 1, \ldots, (\text{N}/2) - 1 \tag{2.5}$$

$$X_{2r+1} = \sum_{\kappa=0}^{N-1} x_n W_N^{n(2r+1)}, \quad \text{r} = 0, 1, \ldots, (\text{N}/2) - 1 \tag{2.6}$$

Grâce à la propriété périodique des facteurs de phases, on peut obtenir les échantillons de fréquence pairs (2.7) :

$$X_{2r} = \sum_{n=0}^{\frac{N}{2}-1} \left(x_n + x_{n+\frac{N}{2}} \right) W_{N/2}^{rn}, \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{2.7}$$

De même, les échantillons de fréquence impairs sont (2.8) :

$$X_{2r+1} = \sum_{n=0}^{\frac{N}{2}-1} \left(x_n - x_{n+\frac{N}{2}} \right) W_N^n W_{N/2}^{rn}, \quad r = 0, 1, \ldots, \frac{N}{2} - 1 \tag{2.8}$$

Les deux équations (2.7) et (2.8) peuvent être combinés dans une unité butterfly comme illustré dans la figure 2.1, Dans cette unité, les valeurs de x(n) et x(n+N/2) sont additionnées pour produire la première sortie, tandis que x(n+N/2) est soustrait de x(n) pour générer la deuxième sortie. Les deux sorties seront ensuite multipliées par les facteurs de phases appropriés.

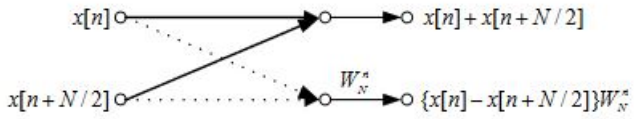


Fig. 2.1 : Graphe en butterfly de radix-2 DIF

2.3.5 Radix 4

En Comparant avec le radix-2, l'avantage de développer le radix-4 DIF est de réduire le nombre d'opérations arithmétiques complexes .Ce dernier nécessite que la taille du séquence d'entrée soit un multiple de 4 .
Dans le radix-4 DIF la séquence de fréquences de sortie est divisée en quatre paquets d'indices suivants **4r, 4r+1 , 4r+2, 4r+3** [9] .
l'équations (2.1) peut réécrire en termes de quatre sommations comme suit (2.9), (2.10), (2.11), (2.12) :

$$X_{4r+l} = \sum_{n=0}^{N-1} x_n W_N^{(4r+l)n}, \mathrm{r} \in \left[0, \frac{N}{4} - 1 \right] \tag{2.9}$$

$$X_{4r+l} = \sum_{n=0}^{N/4-1} x_n W_N^{(4r+l)n} + \sum_{n=N/4}^{N/2-1} x_n W_N^{(4r+l)n} + \sum_{n=N/2}^{3N/4-1} x_n W_N^{(4r+l)n} + \sum_{n=3N/4}^{N-1} x_n W_N^{(4r+l)n}. \tag{2.10}$$

$$X_{4r+l} = \sum_{n=0}^{N/4-1} x_n W_N^{(4r+l)n} + \sum_{n=0}^{N/4-1} x_{n+\frac{N}{4}} W_N^{(2(4r+l))n} + \sum_{n=0}^{N/4-1} x_{n+\frac{N}{2}} W_N^{(3(4r+l))n} + \sum_{n=0}^{N/4-1} x_{n+3\frac{N}{4}} W_N^{(4(4r+l))n}. \tag{2.11}$$

$$X_{4r+l} = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x_n + x_{n+\frac{N}{4}} W_4^l + x_{n+\frac{2N}{4}} W_4^{2l} + x_{n+\frac{3N}{4}} W_4^{3l} \right\} W_{\frac{N}{4}}^{rn}, \quad l = 0, 1, \ldots, 3 \qquad (2.12)$$

A partir de l'équation (2.12) , nous pouvons illustrer le graphe en butterfly de radix-4 comme indiqué dans la figure 2.2[10].

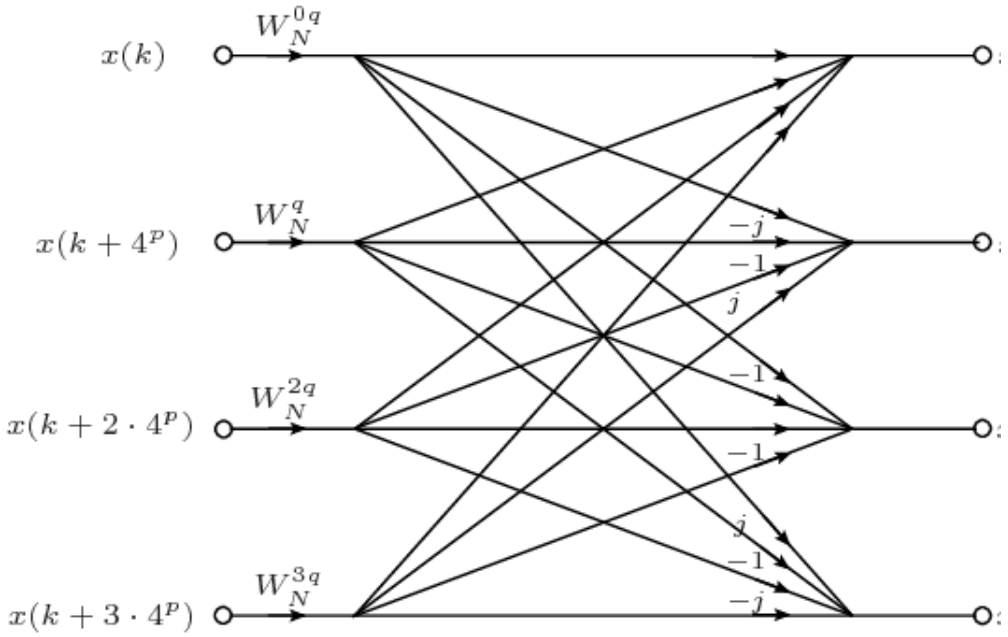


FIG. 2.2 : Graphe en butterfly de radix-4 DIF

2.3.6 Radix 2^2

L'algorithme radix-2^2 est développé pour maintenir une séquence de taille multiple de 4 tout en minimisant la complexité de l'algorithme par rapport au radix-4.
Pour y parvenir, l'algorithme radix-2^2 utilise une approche qui remplace le graphe en butterfly de radix-4 par deux étages de radix-2. Cette substitution se fait en remplaçant l'indice **l** dans l'équation (2.12) par $\mathbf{2}l_2 + l_1$. Cette nouvelle représentation permet de simplifier le processus de calculs .

Voici l'équation obtenue (2.13) :

$$X_{4k+2l_2+l_1} = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x_n + x_{n+\frac{N}{4}} W_4^{2l_2+l_1} + x_{n+\frac{2N}{4}} W_4^{2(2l_2+l_1)} + x_{n+\frac{3N}{4}} W_4^{3(2l_2+l_1)} \right\} W_{\frac{N}{4}}^{kn} \quad l_1, l_2 = 0, 1 \qquad (2.13)$$

La figure 2.3 [11] représente le graph en butterfly de l'algorithme radix 2^2 DIF .

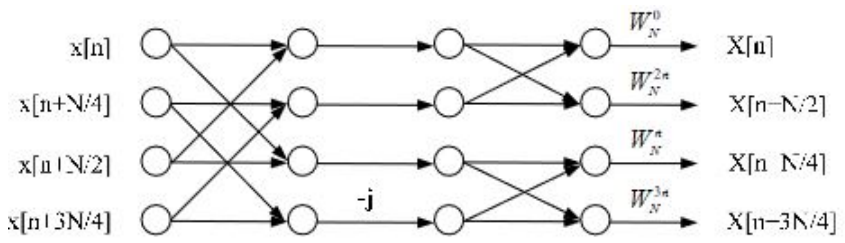


FIG. 2.3 : Graphe en butterfly de radix-2^2 DIF

2.3.7 Radix 8

L'avantage de l'algorithme radix-8 par rapport aux algorithmes FFT radix-2 et radix-4 est de réduire le nombre des opérations complexes et d'accélérer le processus de calcul . Cet algorithme nécessite une séquence d'entrées en base de 8.
Pour cela, l'indice k dans l'équation (2.1) est remplacé par 8 parties distinctes : **8k+l**.

Elle peut s'écrire sous cette forme (2.16) :

$$X_{8r+l} = \sum_{n=0}^{N-1} x_n W_N^{(8k+l)n}, \mathrm{r} \in \left[0, \frac{N}{8} - 1\right] \tag{2.14}$$

$$X_{8r+l} = \sum_{n=0}^{N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=N/8}^{N/4-1} x_n W_N^{(8r+l)n} + \sum_{n=N/4}^{3N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=3N/8}^{N/2-1} x_n W_N^{(8r+l)n} + \sum_{n=N/2}^{5N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=5N/8}^{6N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=6N/8}^{7N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=7N/8}^{N-1} x_n W_N^{(8r+l)n}. \tag{2.15}$$

$$X_{8r+l} = \sum_{n=0}^{N/8-1} x_n W_N^{(8r+l)n} + \sum_{n=0}^{N/4-1} x_{n+\frac{N}{8}} W_N^{(2(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+\frac{N}{4}} W_N^{(3(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+3\frac{N}{8}} W_N^{(4(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+\frac{N}{2}} W_N^{(5(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+5\frac{N}{8}} W_N^{(6(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+3\frac{N}{4}} W_N^{(7(8r+l))n} + \sum_{n=0}^{N/8-1} x_{n+7\frac{N}{8}} W_N^{(8(8r+l))n}. \tag{2.16}$$

On sort $W_N^{nl} W_{\frac{N}{8}}^{nl}$ dans l'équation (2.16) en remplaçant W_N^{8nl} par $W_{\frac{N}{8}}^{nl}$ on obtient cette formule (2.17) :

$$X_{8k+l} = \sum_{n=0}^{\frac{N}{8}-1} \left\{ \left(x_n + x_{n+2\frac{N}{8}} W_4^l + x_{n+4\frac{N}{8}} W_4^{2l} + x_{n+6\frac{N}{8}} W_4^{-l} \right) + \left(x_{n+\frac{N}{8}} + x_{n+3\frac{N}{8}} W_4^l + x_{n+5\frac{N}{8}} W_4^{2l} + x_{n+7\frac{N}{8}} W_4^{-l} \right) W_8^l \right\} W_N^{nl} W_{\frac{N}{8}}^{nr}, \quad l = 0, 1, \ldots, 8 \tag{2.17}$$

Le graphe en papillon de l'algorithme radix 8 DIF avec 8 points est illustré dans la Figure 2.4 [12].

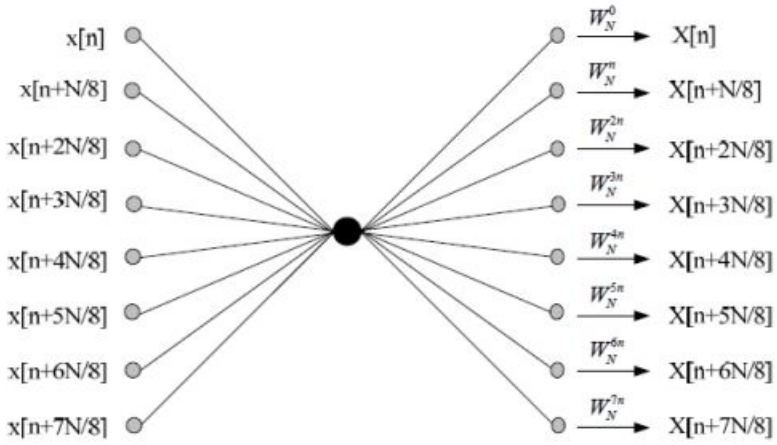


FIG. 2.4 : Graphe en butterfly de radix-8 DIF

2.3.8 Radix 2^3

Lorsque le nombre de points FFT est élevé et est un multiple de 8, il devient avantageux d'utiliser l'algorithme FFT radix-2^3 pour réaliser des économies sur les ressources matérielles et minimiser la complexité de l'algorithme afin de faciliter sa mise en œuvre. La réalisation de cet algorithme se fait en remplaçant **l** dans l'expression (2.17) par $4l_1 + 2l_2 + l_3$ où l_1, l_2, l_3 varient de 0 à 1.
Nous obtenons l'équation (2.18)

$$\begin{aligned} X_{8r+4l_3+2l_2+l_1} = & \sum_{n=0}^{N/8-1} x_n W_N^{(8r+4l_3+2l_2+l_1)n} + \sum_{n=0}^{N/8-1} x_{n+N/8} W_N^{(8r+4l_3+2l_2+l_1)(n+N/8)} + \\ & \sum_{n=0}^{N/8-1} x_{n+N/4} W_N^{(8r+4l_3+2l_2+l_1)(n+N/4)} + \sum_{n=0}^{N/8-1} x_{n+3N/8} W_N^{(8r+4l_3+2l_2+l_1)(n+3N/8)} + \\ & \sum_{n=0}^{N/8-1} x_{n+N/2} W_N^{(8r+4l_3+2l_2+l_1)(n+N/2)} + \sum_{n=0}^{N/8-1} x_{n+5N/8} W_N^{(8r+4l_3+2l_2+l_1)(n+5N/8)} + \\ & \sum_{n=0}^{N/8-1} x_{n+6N/8} W_N^{(8r+4l_3+2l_2+l_1)(n+3N/4)} + \sum_{n=0}^{N/8-1} x_{n+7N/8} W_N^{(8r+4l_3+2l_2+l_1)(n+7N/8)}. \end{aligned} \tag{2.18}$$

Après simplification nous obtenons (2.19) :

$$X_{8k+4l_3+2l_2+l_1} = \sum_{n=0}^{\frac{N}{8}-1} \left\{ \left(x_n + x_{n+4\frac{N}{8}} W_2^{l_1} + W_2^{l_2} W_4^{l_1} \left(x_{n+2\frac{N}{8}} + x_{n+6\frac{N}{8}} W_2^{l_2} \right) \right) \right.$$
$$\left. + \left(x_{n+\frac{N}{8}} + W_2^{l_2} x_{n+5\frac{N}{8}} + W_2^{l_2} W_4^{l_1} \left(x_{n+3\frac{N}{8}} + W_2^{l_1} x_{n+7\frac{N}{8}} \right) \right) W_8^{4l_1+2l_2+l_1} \right\} W_8^{n(4l_1+2l_2+l_1)} W_{N/8}^{nk}. \tag{2.19}$$

Un stage en butterfly de l'algorithme FFT radix-2^3 est particulièrement composé de trois unités en butterfly de radix-2 simple. Cela permet d'implémenter efficacement un graphe en butterfly de radix-8 . Le schéma en butterfly de radix-2^3 est illustré dans la figure 2.5[13].

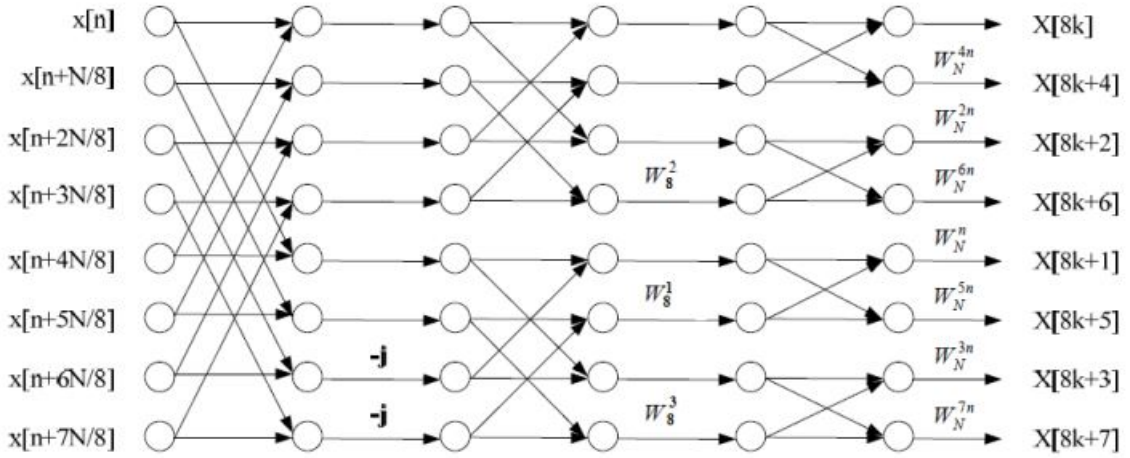


FIG. 2.5 : Graphe en butterfly de radix-2^3 DIF

2.3.9 Analyse comparative des différents algorithms radix

Pour un petit nombre de points à transformer, l'algorithme radix-2 est généralement préféré en raison de sa faible consommation de ressources et de sa facilité d'implémentation. Cependant, pour un grand nombre de points, les algorithmes radix-4 et radix-8 offrent une meilleure efficacité en termes de rapidité mais leur mise en œuvre est plus complexe et nécessitent une grande consommation de ressources logiques [14] [15].

L'algorithme radix-2^3 vise à trouver un équilibre entre la réduction du nombre d'opérations complexes , la minimisation de la consommation de ressources logiques et la rapidité de calcul ce qui est l'objectif de notre mémoire [15].

2.4 Architectures

Dans cette section, nous allons explorer les différentes architectures disponibles pour l'implémentation de la FFT. Chacune de ces architectures présente des avantages et des limitations spécifiques. En examinant ces différentes approches, nous pourrons mieux comprendre les choix et les compromis à prendre en compte lors de la mise en œuvre de la FFT.

2.4.1 Architecture parallèle

L'architecture parallèle est obtenue par une projection directe du graphe de flux de signal.la figure 2.6 [16] illustre l'architecture parallèle pour N = 8.

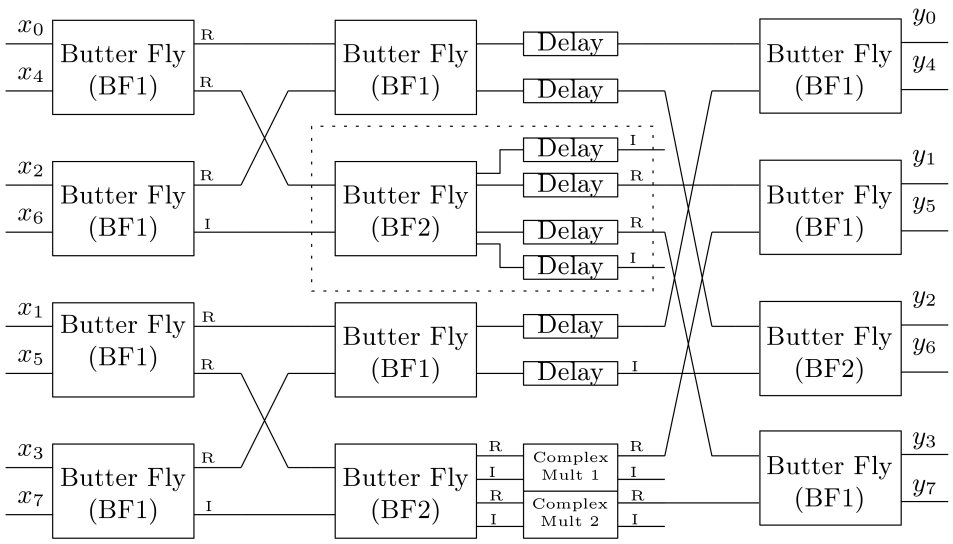


Fig. 2.6 : Architecure parallèle

Les échantillons arrivent en parallèle et en même temps sur les entrées des BF de la rangée gauche et les composantes spectrales sont récupérées simultanément aux sorties des PEs de la rangée droite. L'architecture parallèle est utilisée pour sa rapidité et son efficacité dans le calcul de la FFT. Cependant pour implémenter cette architecture, il est nécessaire d'utiliser $(\frac{N}{r})\log(N)$ blocs de traitement élémentaires, où r représente la base du radix ce qui signifie que les ressources matérielles nécessaires augmentent rapidement avec la taille N de la FFT.

De plus, un des problèmes majeurs de cette architecture est la faible utilisation des BPE, car les données d'entrée sont traitées séquentiellement tandis que les données de sortie sont générées en parallèle. Cela signifie que certaines ressources matérielles peuvent rester inutilisées pendant certains cycles d'horloges.

2.4.2 Architecture pipeline

Le pipelining est une technique utilisée pour améliorer les performances de calcul en divisant une tâche en plusieurs étapes et en permettant à chaque étape de fonctionner de manière concurrente. Ce chevauchement d'opérations permet d'obtenir un débit élevé et de réduire le temps nécessaire pour terminer une tâche.Toutefois, cette approche implique une logique beaucoup plus complexe.

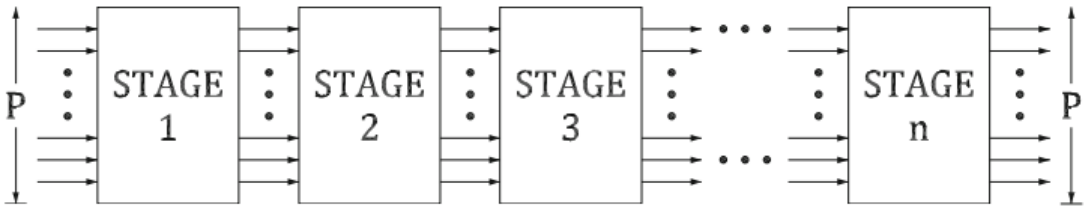


Fig. 2.7 : Architecture pipeline

La figure 2.7 [17]illustre une architecture pipeline où chaque étage dispose de P entrées et P sorties, et le flux de données s'écoule en continu à un débit de P données par cycle d'horloge. Une méthode efficace pour obtenir de telles architectures en pipeline pour différents algorithmes FFT est de projeter le graphe de flux de signal (SFG) perpendiculairement au flux de données, puis d'obtenir le matériel nécessaire 2.8.

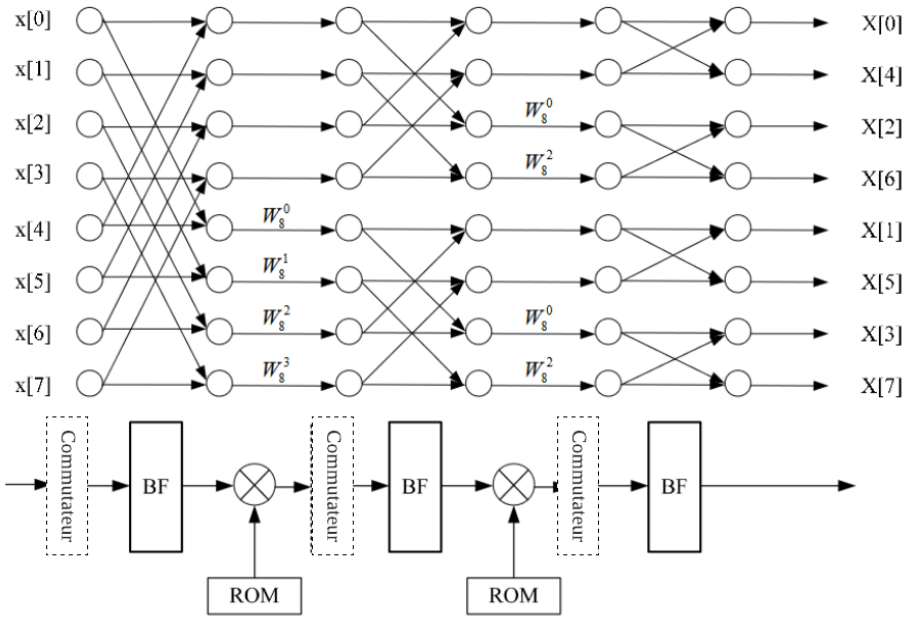


Fig. 2.8 : Projection orthogonale du graphe de flux de signal [17]

Dans le système, plusieurs unités de traitement Butterfly (BF) sont utilisées pour traiter les données. Chaque unité de traitement BF est responsable d'effectuer $\frac{N}{r}$ opérations consécutives. Pour faciliter le flux des données, l'unité "Commutateur" est utilisée pour réarranger les données entre les différentes unités BF. Cela permet de garantir un cheminement fluide des données à travers le système. À chaque niveau du pipeline, chaque unité BF effectue des calculs complexes sur r données, ce qui permet de générer r données intermédiaires en sortie.

Le premier point de différence concerne le nombre de chemins de données utilisés pour traiter les échantillons. Il existe deux types d'architectures :

- celles à chemin unique (Singlepath ; S)
- celles à chemin multiple (Multi-path ; M).

Le deuxième point de différence concerne la stratégie de mémorisation utilisée pour créer les différents retards nécessaires à l'ordonnancement des échantillons. Dans l'ensemble, il existe deux stratégies de stockage tampon pour le processeur FFT basé sur une architecture en pipeline :

- l'architecture à commutateur de retard (DC)
- l'architecture à rétroaction de retard (DF).

En fonction de ces deux critères, nous pouvons diviser les architectures en deux types.

- Radix-r SDF : Single-path Delay Feedback.
- Radix-r SDC : Single-path Delay Commutator .
- Radix-r MDC : Multi-path Delay Commutator

Multi-path Delay Commutator MDC

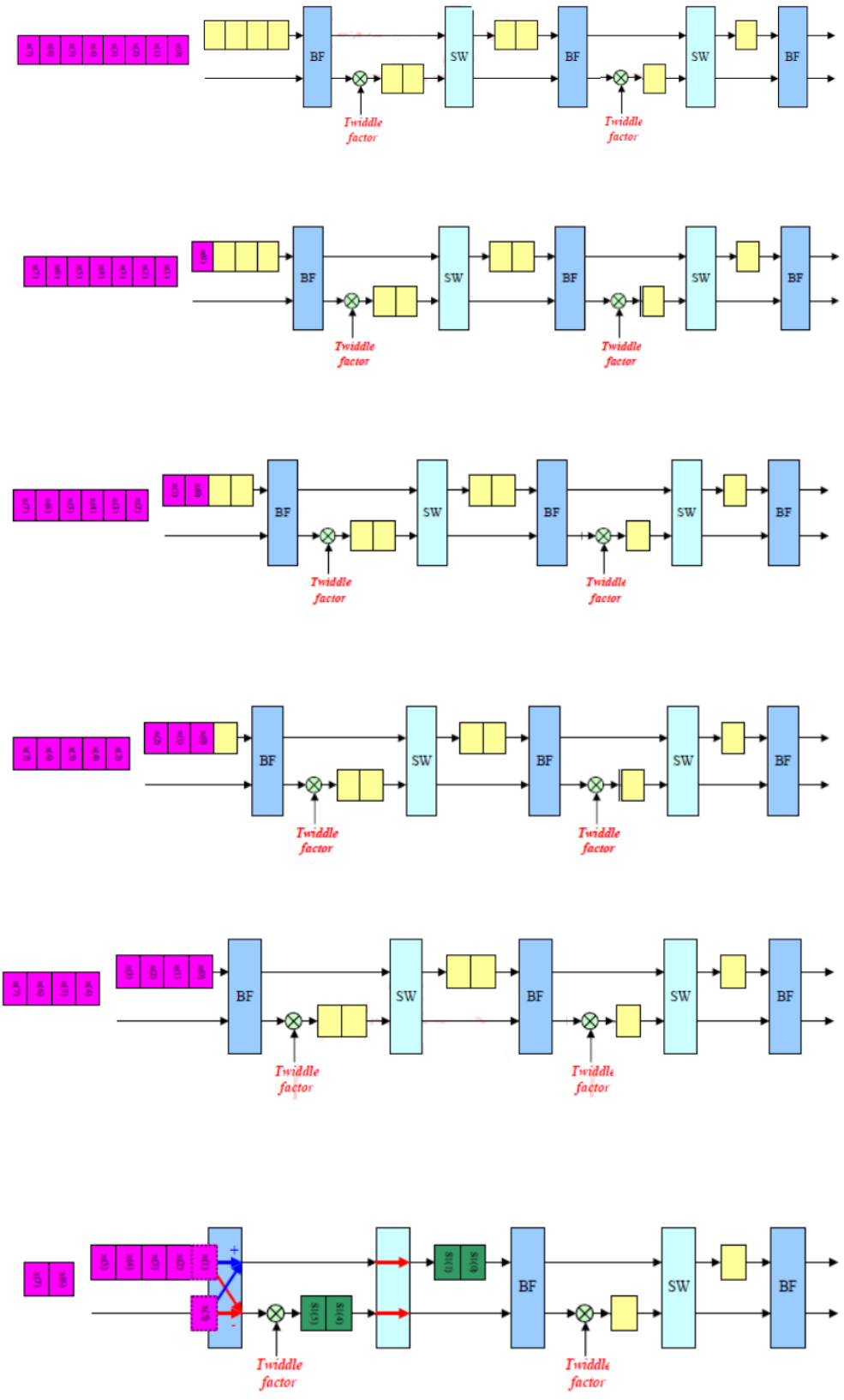
Dans une architecture MDC, La séquence de données d'entrée est divisée en plusieurs chemins, généralement égaux à la base de l'algorithme de la FFT . Chaque chemin reçoit un sous-ensemble des données d'entrée.

Dans le contexte d'un radix 2, l'entrée est séparée en deux flux de données parallèles. La première moitié des données d'entrée est retardée à l'aide d'un registre, permettant ainsi au premier module de traitement de recevoir les données xi et xi+$\frac{N}{2}$ en tant que entrées respectives.

Pour s'assurer que les paires de valeurs d'entrée correctes arrivent aux papillons, des registres a décalage et un commutateur de délais sont utilisés. Le commutateur de délais réordonne les valeurs avant qu'elles ne soient transmises au papillon suivant.
À la différence de l'architecture SDF, il n'est pas nécessaire de renvoyer les données intermédiaires vers le registre. Cette approche permet d'atteindre un débit de données

supérieur à celui de l'architecture SDF.

Le processus de fonctionnement de cette architecture se décline en plusieurs étapes comme illustré dans les figures 2.9 [18] :



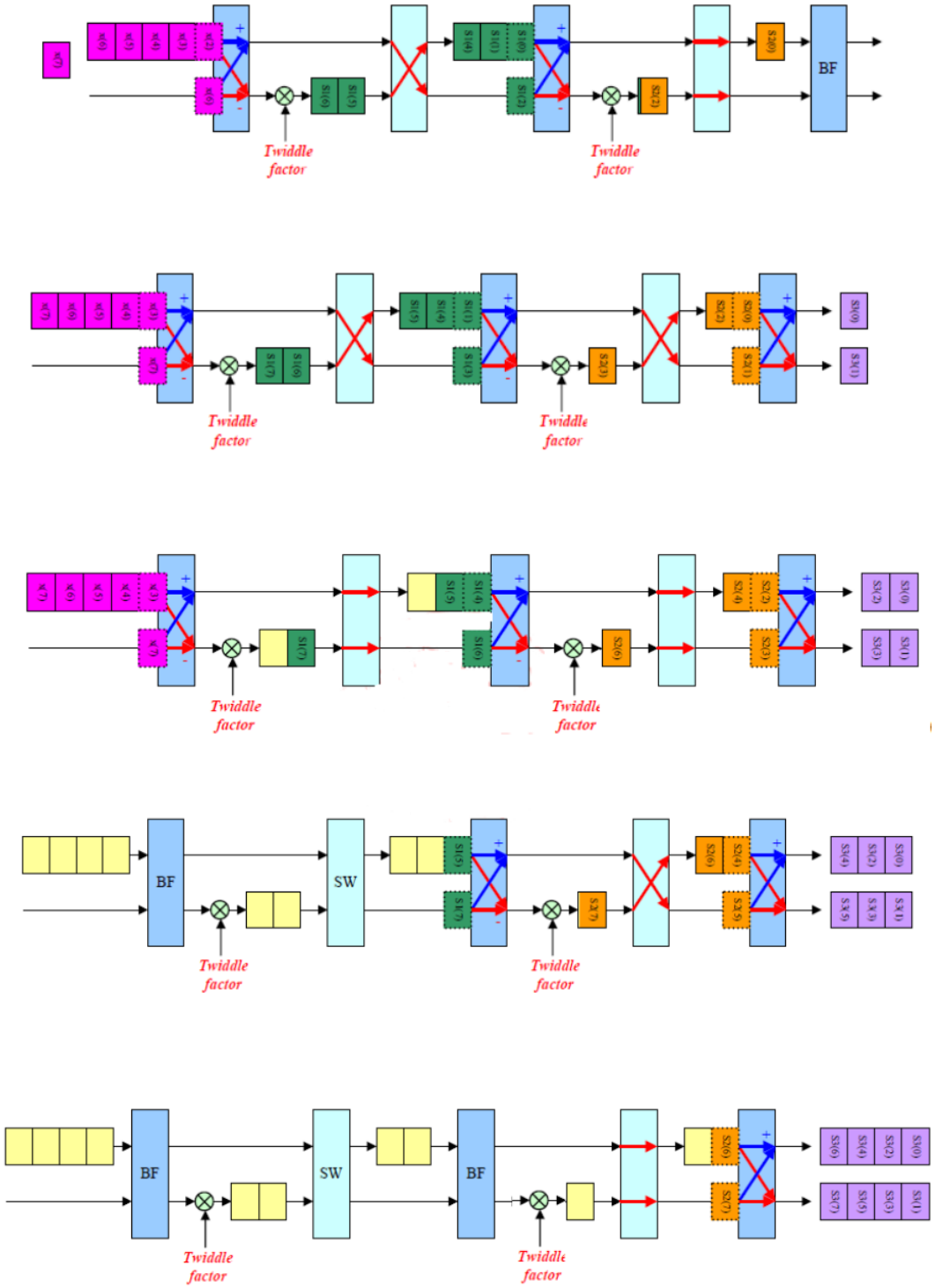


FIG. 2.9 : Fonctionnement de l'architecture MDC

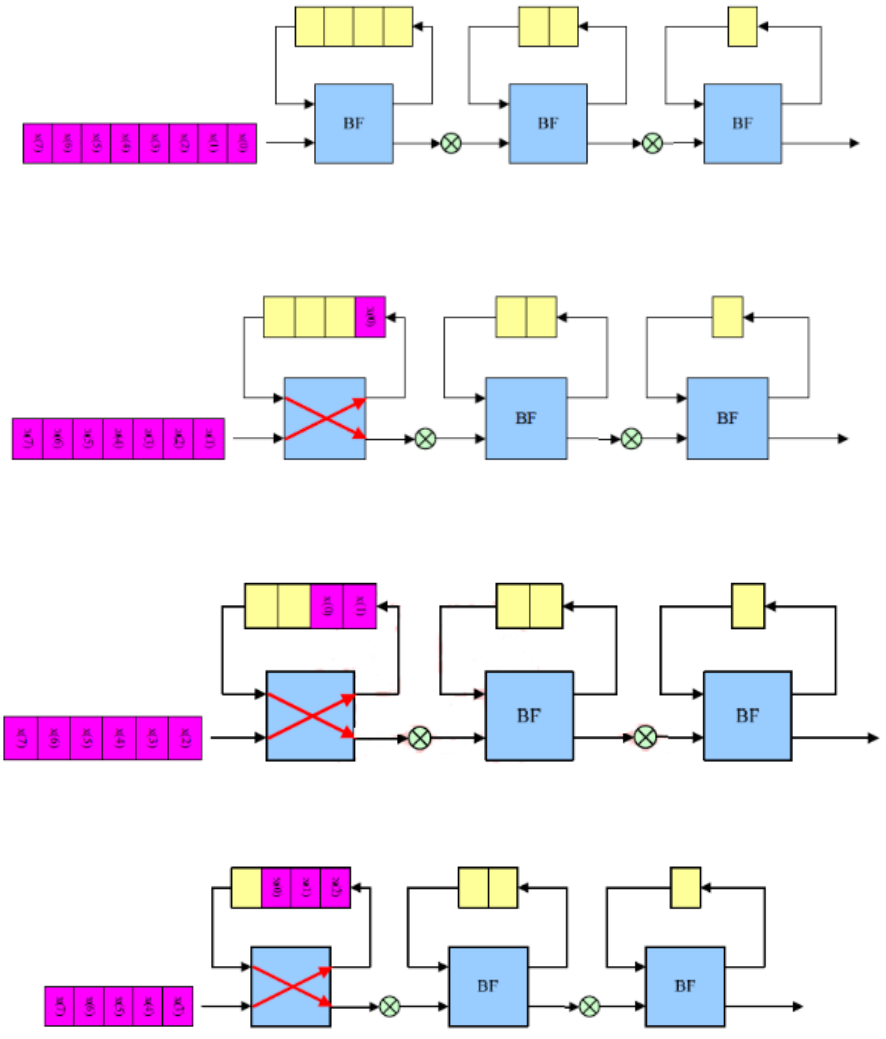
Single-path Delay Feedback SDF

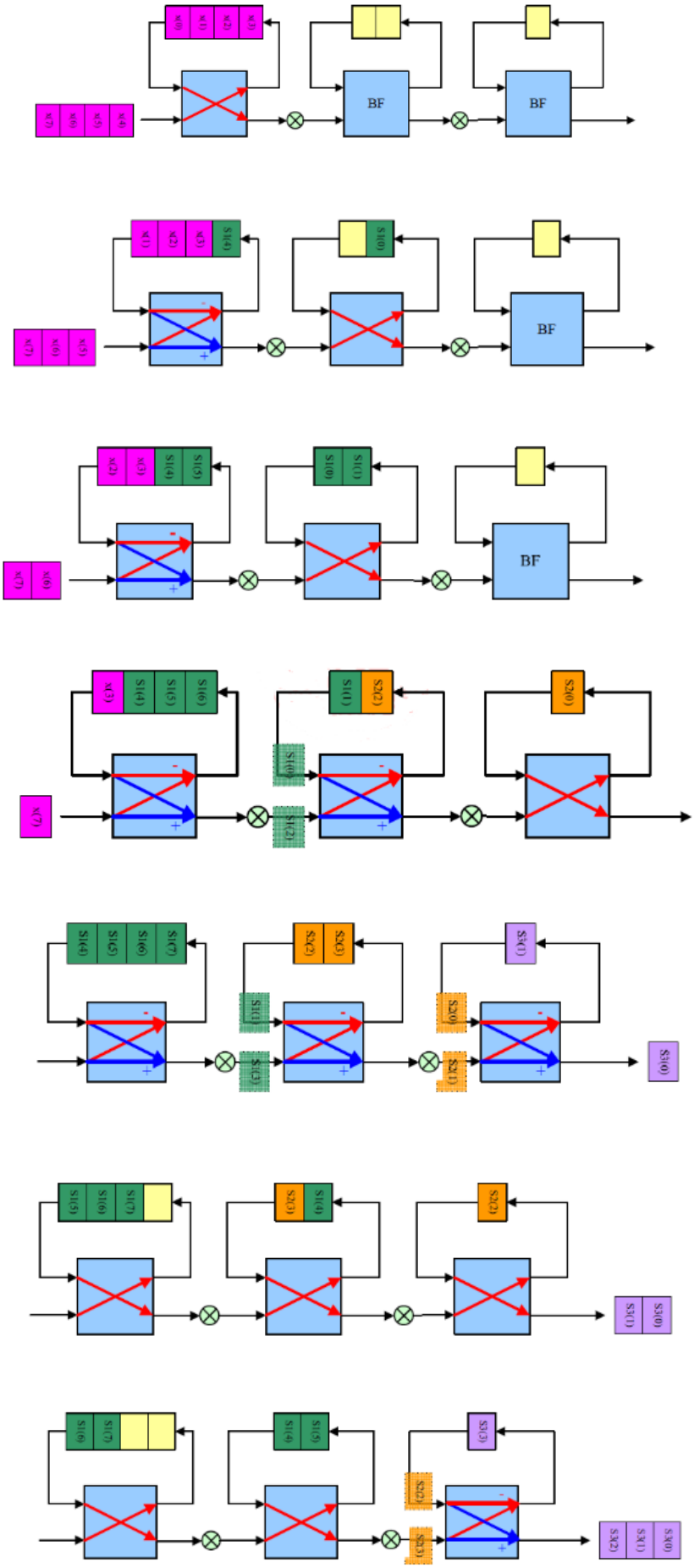
Dans cette architecture, chaque bloc de traitement possède une seule entrée et une seule sortie. Comme illustré dans les figures 2.10[18] , les données issues des blocs butterfly sont enregistrées dans des registres à décalage. Grâce à la propagation des séquences de données d'entrée le long d'un seul chemin, un unique multiplicateur complexe est utilisé

après chaque bloc butterfly, ce qui se traduit par une réduction de la complexité matérielle. Cependant, l'architecture SDF peut fonctionner à une vitesse relativement basse, et la synchronisation des données nécessite une gestion complexe des contrôles.

Le processus de cette architecture est le suivant :

- Les données d'entrée correspondant aux indices de 0 à N/2 sont stockées dans le registre à décalage. Les unités butterfly radix-2 opèrent sur les données du registre ainsi que sur les données d'entrée restantes correspondant aux indices de N/2 à N.
- Les données résultant des opérations d'addition des papillons sont transmises au 2eme étage, tandis que les résultats de soustraction sont renvoyés au registre à décalage .
- Une fois que les N/2 points issus de l'addition passent à l'étage suivant, les données de soustraction provenant du registre sont d'abord traitées par l'unité de multiplication avant de pouvoir passer à l'étage suivant.
- Le fonctionnement des étages suivants est similaire à celui du premier étage.




BF
BF
BF
BF

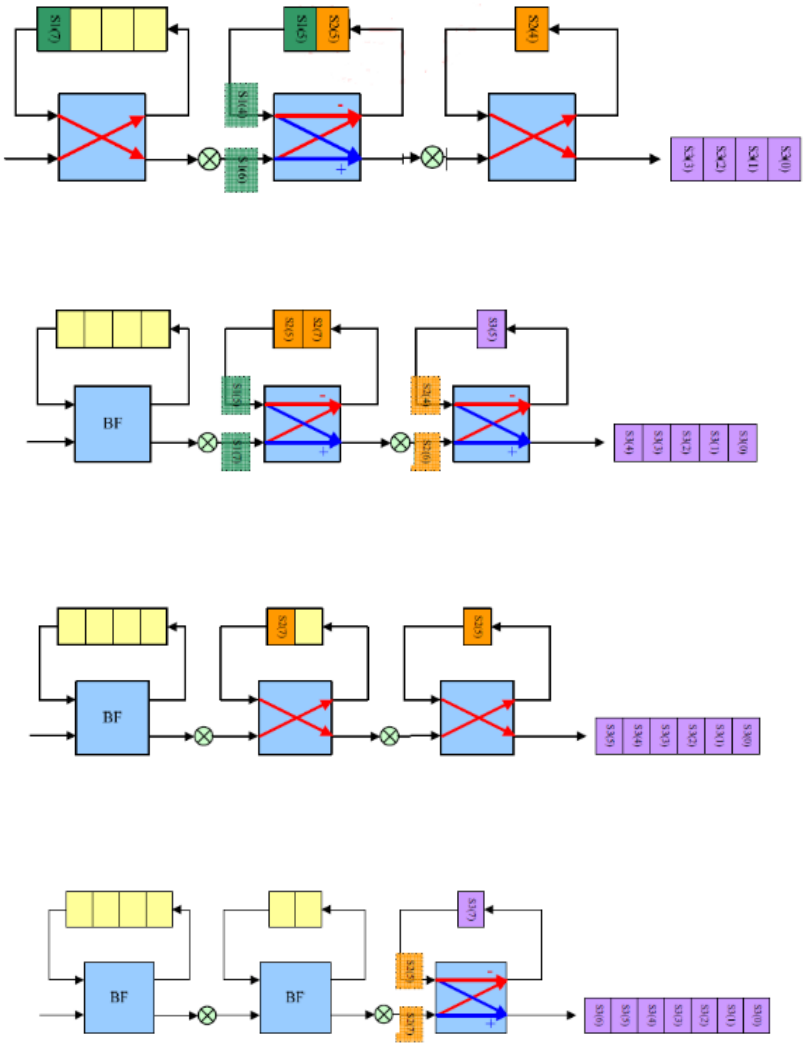


FIG. 2.10 : Fonctionnement de l'architecture SDF

2.4.3 Single-path Delay Commutator SDC

L'architecture SDC, contrairement à l'architecture MDC et SDF, utilise un commutateur de délais avec une seule entrée et plusieurs sorties. Le nombre de sorties du commutateur varie en fonction du nombre d'entrées du BF. Cette configuration conduit à une utilisation matérielle plus élevée que celle de l'architecture MDC [19].
Cependant, l'architecture SDC fonctionne à un débit binaire plus faible en raison de son unique chemin d'accès pour les entrées et les sorties.

2.5 Revue de littérature

Dans cette revue de l'état de l'art, nous explorerons les différentes approches et les avancées récentes dans le domaine de l'implémentation de la FFT sur FPGA. Nous examinerons les architectures matérielles proposées, les algorithmes utilisés et les résultats expérimentaux obtenus dans la littérature.

En étudiant ces travaux, nous pourrons analyser les avancées réalisées en termes de performances et d'utilisation des ressources dans les différentes approches d'implémentation de la FFT sur FPGA. Cette comparaison nous permettra de mieux comprendre les forces et les limites de chaque approche, ainsi que les opportunités d'amélioration.

[20] a proposé un processeur FFT pour les applications SAR a base de l'algorithme de décomposition en fréquence radix-2. Cette approche a donné de meilleurs résultats par rapport au cœur IP Altera, avec une réduction de 33,33 des DSP et de 77,96 des LUT.

l'article [3], a implémenter l'algorithme FFT radix-23 en utilisant une structure SDF. Pour générer les facteurs de phase, une méthode basée sur Cordic a été utilisée. Cette approche a permis d'éliminer l'utilisation des blocs ROM pour stocker les facteurs de phase.

Des modifications ont été apportées par [21]en proposant un schéma de multiplication entièrement basé sur Cordic. Cela a permis de réaliser la multiplication de nombres complexes sans avoir recours aux blocs multiplicateurs DSP. L'implémentation proposée s'est avérée avoir une consommation des ressources moindre par rapport aux implémentations précédentes.

D'autres schémas de multiplication ont été proposés dans la littérature. Dans cet article [22], des multiplicateurs Vedic ont été introduits, ce qui permet de réduire la longueur totale des opérations de multiplication, de plus, un additionneur à compresseur modifié 4:2 a été utilisé pour accélérer les multiplicateurs Vedic.Cette implémentation a réduit le nombre de registres de tranche à 87,8 et les LUTs de tranche à 48 par rapport au multiplicateur Vedic conventionnel.

En outre, l'article[23], a mis en œuvre des architectures FFT feedforward avec deux multiplicateurs différents, ces derniers ont été comparées en termes d'utilisation de l'espace sur le FPGA. Il est observé que le multiplicateur basé sur l'algorithme de Booth modifié offre une meilleure optimisation de l'espace par rapport à l'architecture FFT avec le multiplicateur Vedic.

Afin de réduire davantage la consommation de ressources, le choix de l'architecture et de l'algorithme utilisé par le processeur FFT est primordial.
En termes d'algorithme FFT, l'article [24] a mené une analyse comparative des performances de deux approches d'implémentation de la FFT sur FPGA : l'algorithme radix-2 et l'algorithme radix-2^2. Les résultats ont démontré que les algorithmes de la série radix-2^k présentaient un taux d'utilisation des ressources logiques nettement plus élevé que les algorithmes de la série radix-k.

D'un autre côté, He Chen [3] a exploré une approche différente en combinant l'algorithme radix-2^3 et l'algorithme radix 4, ce qui est connu sous le nom de radix mixte. L'utilisation de l'algorithme radix-2^3 a permis de réduire considérablement la consommation de ressources du multiplicateur par rapport aux algorithmes radix-2 et radix-4. De plus, le radix mixte a permis l'utilisation d'une structure à quatre canaux, ce qui a

significativement réduit le temps de calcul de la FFT.

L'article [25] présente un processeur FFT utilisant un algorithme basé sur le radix-8 DIT (Decimation in Time). Dans cet algorithme, le calcul sur place est utilisé pour une utilisation efficace des mémoires.
D'autres part, [26] a proposé une amélioration du radix 8, ce qui réduit le nombre de registres de tranche de 87,8 et les LUTs de tranche de 48 par rapport à l'algorithme radix-8 conventionnel.

En termes d'architecture, [17] a proposé des architectures MDC de type radix-2^k, en se concentrant spécifiquement sur les conceptions radix-22, radix-23 et radix-24. Ces architectures ont démontré une efficacité matérielle supérieure par rapport aux conceptions 2^k de l'architecture SDF.

Dans [21], l'architecture MDC (Multiple Delay Commutator) traditionnelle a été optimisée et une architecture MDC améliorée a été proposée. L'architecture proposée améliore le débit et la vitesse de calcul en réduisant l'utilisation des unités de retard. De plus, le flux de données dans cette architecture reste continu, contribuant ainsi à son efficacité.

Une autre contribution notable provient de [27], qui a présenté une architecture combinée efficace appelée architecture FFT en pipeline Single-path Delay Commutator-Feedback (SDC-SDF) radix-2. Cette conception intègre $\log_2 N - 1$ étapes SDC et 1 étape SDF. Le bloc de traitement SDC atteint une utilisation des ressources matérielles de 100% en partageant des ressources arithmétiques communes grâce à une approche multiplexée dans le temps.

Ces avancées dans les architectures FFT offrent une meilleure efficacité matérielle, des besoins réduits en ressources, un débit plus élevé et une vitesse de calcul améliorée. Elles ont le potentiel de bénéficier considérablement aux applications exigeantes qui reposent sur des calculs FFT.

2.6 Conclusion

Cette étude approfondie des concepts fondamentaux ainsi que de l'état de l'art fournira les fondements essentiels à notre recherche et nous permettra de positionner notre travail par rapport aux avancées déjà réalisées dans le domaine.

Chapitre 3

Structure et conception

3.1 Introduction

Dans ce chapitre, nous présenterons tout d'abord l'architecture proposée pour l'implémentation de la FFT sur FPGA. Nous décrirons les différentes étapes de conception et de modélisation nécessaires pour réaliser cette architecture.Par la suite, nous examinerons en détail la conception du processeur FFT, qui est la partie centrale de notre implémentation.Les éléments clés de cette conception seront également détaillées, notamment les décisions relatives à l'architecture choisie.
Enfin, nous discuterons des simulations effectuées pour évaluer les performances de notre conception. Ces simulations nous permettront de valider l'efficacité de notre approche de conception de la FFT sur FPGA.

3.2 Architecture proposée

Différents algorithmes FFT présentent des caractéristiques distinctes en termes de consommation de ressources matérielles et de performances. Le bon choix du radix peut permettre d'optimiser l'utilisation des ressources, réduire les calculs redondants et améliorer l'efficacité globale de l'implémentation de la FFT.
Le tableau 3.1 [14] présente l'occupation du multiplicateur, de l'unité de mémoire et de l'unité papillon lors de l'utilisation de la structure MDC pour mettre en œuvre une FFT de N points.

Algorithmes	**Multiplieurs**	**Additionneurs**	**Mémoires**
Radix 2 MDC	$\log_2(N-2)$	$2\log_2(N)$	1.5N-2
Radix 2^2 MDC	$\log_2(N-2)$	$2\log_2(N)$	1.5N -2
Radix 4 MDC	$3\log_4(N-3)$	$8\log_4(N)$	2.5N-4
Radix 8	$7\log_8(N-7)$	$24\log_8(N)$	4.5N-8
Radix 2^3	$\log_8(N-2)$	$2\log_8(N)$	1.5N -2

Tab. 3.1 : Comparaison des différents radix en terme de consommation matérielle [14]

Lorsqu'il s'agit d'un petit nombre de points à transformer, l'algorithme radix-2 est généralement préféré en raison de sa faible consommation de ressources et de sa facilité d'implémentation. Cependant, pour un grand nombre de points, les algorithmes radix-4 et radix-8 offrent une meilleure efficacité en termes de vitesse, mais leur mise en œuvre est plus complexe et nécessite une consommation plus élevée de ressources logiques [14]. Dans notre mémoire, notre objectif est de trouver un équilibre entre la réduction du nombre d'opérations complexes, la minimisation de la consommation de ressources logiques et la rapidité de calcul. C'est pourquoi nous avons exploré l'algorithme radix-2^3.

En ce qui concerne l'architecture, les architectures MDC se démarquent par leur débit remarquable, comme en témoigne [17]. Leur capacité à atteindre des performances élevées en termes de traitement de données les positionne comme le choix idéal, même pour les

applications les plus exigeantes en termes de ressources.

Dans l'architecture que nous proposons, nous combinons les avantages du radix-2^3 et de la structure de circuit MDC pour créer une implémentation FFT performante. Cette combinaison nous permet d'obtenir une flexibilité accrue ainsi qu'une une utilisation réduite de la mémoire et des ressources logiques.

3.3 Étapes de conceptions et modélisation

En tant que première étape du processus de l'implémentation de l'algorithme FFT radix 2^3 pour les plates-formes matérielles, il est crucial de disposer d'un modèle de référence pouvant servir de base pour les tests et le débogage. Dans ce cas, nous utiliserons Matlab pour développer et modéliser l'algorithme FFT à virgule fixe, le Radix 2^3 .

Matlab offre un environnement polyvalent pour le développement et la simulation d'algorithmes, ce qui nous permet de valider la précision et les performances de notre algorithme avant de passer à la mise en œuvre matérielle.

Nous commençons d'abord par déterminer la représentation des données la plus appropriée pour notre implémentation. Comme nous le savons, un nombre à virgule fixe peut être représenté en deux parties : la partie entière et la partie fractionnaire. Ainsi, couramment, les concepteurs mesurent la plage dynamique du signal à l'aide d'un modèle à virgule flottante (qui ne souffre pas de débordement) afin d'obtenir la largeur en bits requise pour la partie entière. La largeur en bits de la partie fractionnaire est choisie en fonction de la précision souhaitée.
Pour déterminer la largeur minimale suffisante des bits fractionnaires requise pour une implémentation matérielle optimale, nous avons réalisé une analyse en calculant l'erreur entre les représentations en virgule fixe de différentes largeurs de bits et leurs réalisations en virgule flottante correspondantes.

Une fois les paramètres définis, la prochaine étape consiste à exécuter l'algorithme FFT. Cette exécution nous fournira deux composants essentiels pour notre implémentation : les facteurs de phases nécessaires au calcul FFT et les résultats intermédiaires obtenus à chaque étape. Ces résultats intermédiaires seront essentiels pour vérifier la justesse de notre conception lors de la phase de simulation VIVADO. De plus, il est nécessaire de sauvegarder les données d'entrée et les facteurs de phases dans des fichiers .coe. Ces fichiers serviront de données d'initialisation pour les ROMs et les RAMs utilisés dans notre architecture proposée, garantissant que les données nécessaires sont facilement accessibles pour le processus de mise en œuvre.

Dans la dernière étape de notre implémentation, nous générons le fichier de flux binaire (bitstream), qui contient les données de configuration pour le FPGA. Ce fichier de flux binaire est ensuite chargé sur la carte FPGA, ce qui lui permet d'exécuter l'algorithme FFT basé sur l'architecture conçue. La carte FPGA traitera les données d'entrée

et effectuera le calcul FFT. Les données de sortie résultantes seront transmises de la carte FPGA à MATLAB via une communication UART (Universal Asynchronous Receiver-Transmitter).

Les données reçues à Matlab seront traitées et tracées afin de visualiser la représentation du domaine fréquentiel du signal d'entrée. Cela nous permet de vérifier le bon fonctionnement de notre implémentation et de s'assurer que le calcul FFT est effectué correctement. En examinant le tracé du domaine fréquentiel, nous pouvons vérifier l'exactitude et la fonctionnalité de notre implémentation FFT.
La figure 3.1 présente un récapitulatif des étapes expliquées précédemment.

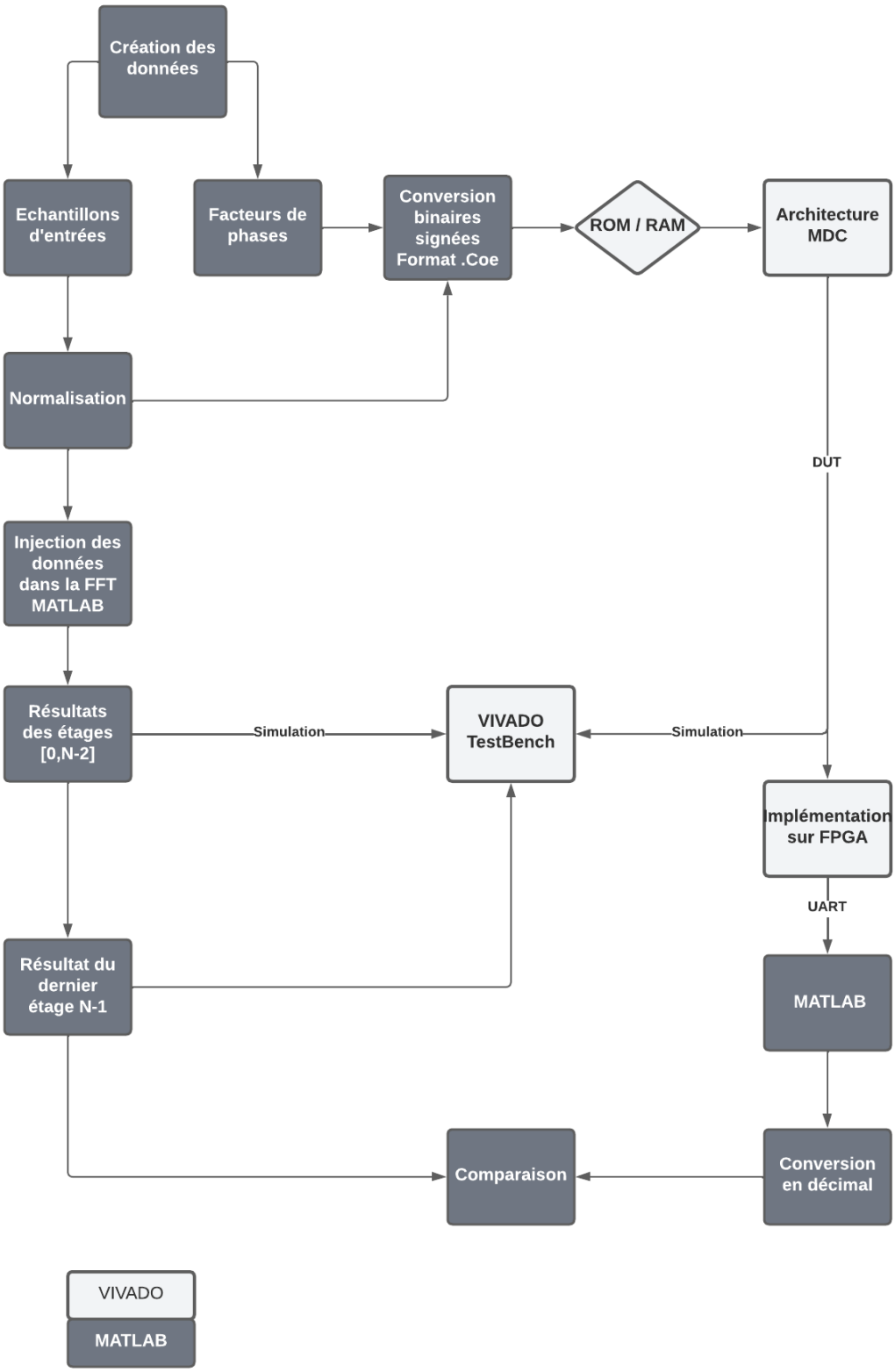


FIG. 3.1 : Étape de conception du processeur FFT

3.4 Choix du format de données

Bien que les circuits intégrés spécialisés offrent des avantages en termes de coût et d'efficacité énergétique, la présence de bruit résultant de l'arithmétique en virgule fixe doit être soigneusement prise en compte lors de la conception et de la mise en œuvre des algorithmes.

pour déterminer le nombre de bits nécessaire, nous implémentons l'algorithme 2^3 sur MATLAB. En calculant le rapport signal sur bruit de quantification pour une FFT de 4096 points pour différents nombres de bits, nous obtenons les résultats de la figure 3.2. Afin de trouver un compromis entre la précision et la consommation des ressources, nous optons pour une taille de 16 bits. Étant donné que les données d'entrée sont normalisées, seulement deux bits sont réservés pour la partie entière, tandis que le reste est utilisé pour la partie fractionnaire. Cela garantit un rapport signal sur bruit supérieur à 60 dB.

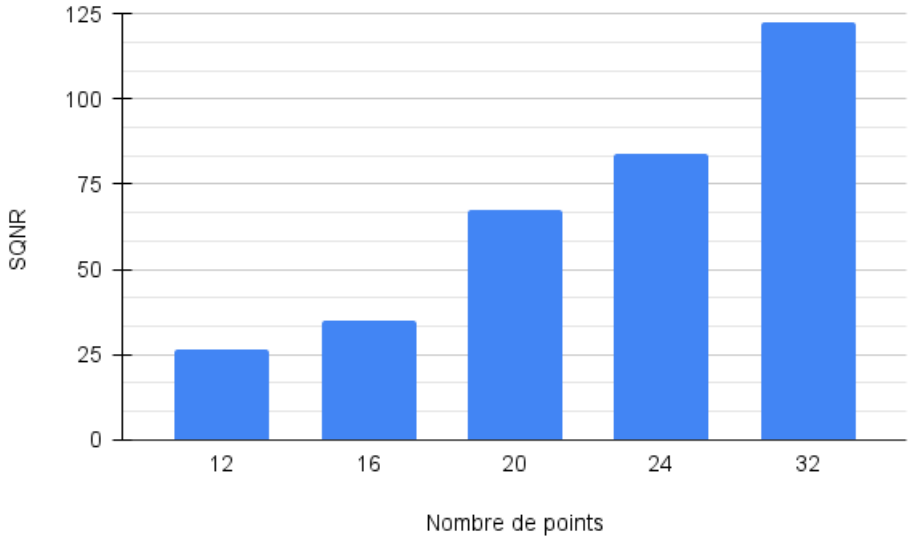


Fig. 3.2 : La valeur de SQNR en fonction du nombre de bits

3.5 Conception du processeur FFT

Après avoir présenté toutes les informations nécessaires sur les étapes de conception de la FFT radix-2^3 dans la section précédente, nous abordons maintenant la conception de l'architecture pipeline radix-2^2 MDC , Nous examinerons en détail les principaux éléments qui la composent, ainsi que le rôle de chaque bloc.

L'architecture du pipeline radix-2^3 MDC DIF à N points comprend $n = log_2(N)$ étages comme illustré dans la figure 3.3.
Les données d'entrées sont stockées dans une mémoire RAM et sont ensuite transmises au cœur de l'FFT pour effectuer la transformation. Comme nous avons utilisé l'algorithme

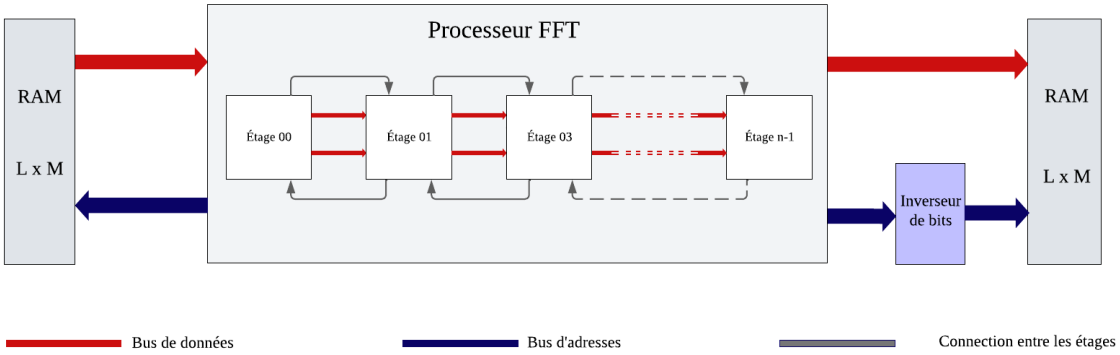


FIG. 3.3 : L'architecture FFT

DIF (Decimation in Frequency), la sortie du dernier papillon de la FFT doit être réarrangée. Pour accomplir cela, le bloc "bit reverse" génère des adresses réorganisées qui sont ensuite utilisées par le bloc RAM pour stocker les sorties du processeur FFT de manière ordonnée. Cela garantit que les résultats de la FFT sont correctement alignés.

La figure 3.4 offre une visualisation complète d'un étage du processeur.
Les bus de données sont représentés par des signaux de couleur rouge, tandis que les bus d'adresses sont indiqués par des signaux de couleur bleue. Les signaux de commande, émanant du bloc contrôleur, sont quant à eux représentés par des signaux de couleur grise.

Une fois le module FFT activé, l'unité de contrôle génère les adresses mémoire requises pour extraire les données stockées dans la RAM.
Le générateur d'adresses fournit les adresses nécessaires pour récupérer les facteurs de rotation présentées dans le tableau 3.2 à partir de la mémoire ROM . Ces facteurs de rotation sont ensuite transférés dans des registres qui seront utilisés ultérieurement dans le processus de multiplication.

Algorithme	Étage							
	1	2	3	4	5	6	7	8
Radix-2^3	W_4	W_8	W_N	W_4	W_8	$W_{N/8}$	W_4	W_8

TAB. 3.2 : Les facteurs de phases des différents étages de la FFT radix-2^3 à N-point

Une fois les données préparées, l'unité de calcul du butterfly est déclenchée. Le résultat de la soustraction est multiplié par le facteur de rotation approprié et stocké dans un registre à décalage, tandis que le résultat de l'addition est directement enregistré dans le registre deux.Ce processus est répliquer jusqu'à ce que les deux registres atteigne leurs capacité maximale, activant ainsi un signal "full".

Lorsque le signal "full" est activé, le commutateur inverse le cheminement des données présentes à son entrée afin de permettre l'alignement des données pour l'étage suivant.

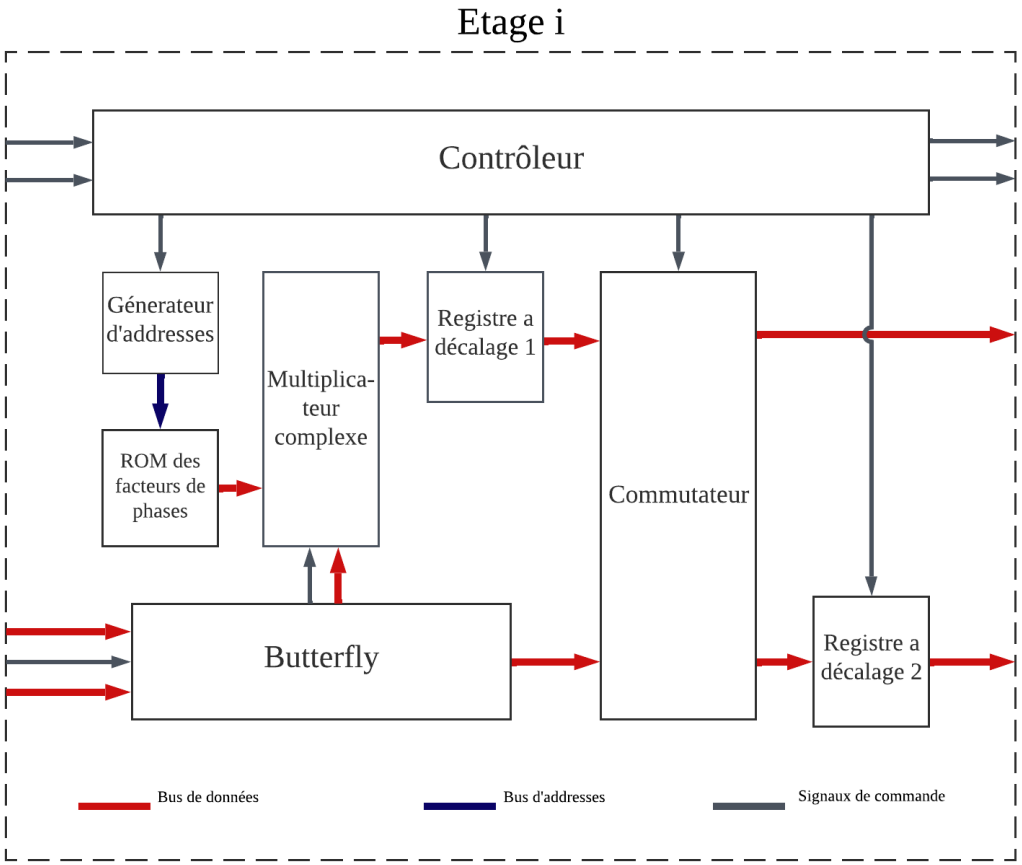


FIG. 3.4 : Architecture interne d'un étage du processeur

Ce processus est réitéré jusqu'à ce que tous les registres soient remplis et le signal "full" est à nouveau activé. À ce stade, le commutateur reprends sa position initiale permettant l'acheminement des données issues de la multiplication vers l'étage suivant.
Une représentation améliorée du flux des données entre les registres a été présentée Au chapitre deux, section deux.

Chaque étage de l'architecture effectue les mêmes opérations. excepté le dernier étage qui se compose uniquement du bloc butterfly générant les sorties finales de la FFT.

3.5.1 Unité de contrôle

Afin d'exécuter l'algorithme FFT de manière efficace, il est nécessaire d'avoir un circuit spécialisé qui détermine quand et quelles opérations doivent être effectuées. Le circuit est connu sous le nom d'unité de contrôle qui peut être réalisé par une machine à états finis afin d'imposer l'ordre des étapes souhaitées. Cela permet d'assurer la coordination interne entre les différents blocs qui composent chaque étage, ainsi que la synchronisation externe entre les différents étages qui forment le processeur FFT. Le contrôleur FSM nécessite

cinq entrées pour son fonctionnement :

- La broche **clk**, qui est le signal d'horloge utilisé pour piloter la FSM.
- La broche **reset**, qui est la broche de réinitialisation activée à l'état bas pour remettre la FSM dans un état initial.
- La broche **Go**, qui est utilisée pour déclencher le processeur et démarrer les opérations.
- Les entrées **request_in** et **receive**, qui permettent d'interagir avec l'étage précédent et de recevoir les données nécessaires T.

Ce bloc produit également huit sorties :

- La sortie **request_out**, qui est utilisée pour établir la connexion avec l'étage suivant du processeur.
- La sortie **en_stage**, qui indique le démarrage de l'étage suivant.
- Les sorties **add_wi** et **add_wr**, qui correspondent aux adresses appropriées dans la mémoire ROM pour lire les facteurs de phase nécessaires.
- De plus, il génère les sorties **enw_fifo1**, **enw_fifo2**, **enr_fifo1** et **enr_fifo2** pour effectuer les opérations de lecture et d'écriture dans les registres à décalage.

La figure 3.5 représente l'unité de contrôle.

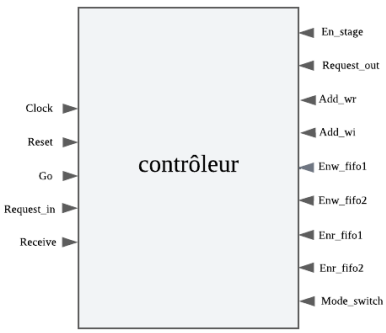


Fig. 3.5 : Unité de contrôle

Généralement, cette machine d'état se compose de quatre états pour gérer les opérations : **IDLE, DATA_WRITE, DATA_READ_WRITE** et **DATA_READ**.

L'état IDLE est le point de départ où les signaux tels que les adresses et les compteurs sont initialisés. Lorsque le signal **GO** est activé, les différents blocs sont déclenchés et la machine d'état passe à l'état DATA_WRITE.

Dans l'état DATA_WRITE , le contrôleur transfère les données de la mémoire vers l'unité butterfly et récupère les facteurs de phases correspondants pour l'unité multiplicateur. Les registres à décalage sont utilisés uniquement en mode écriture, tandis que l'unité commutateur est configurée au mode "0". Cette séquence d'opérations est répétée jusqu'à ce que les registres FIFO soient remplis. Ensuite, la machine d'état passe à l'état DATA_READ_WRITE.

En état DATA_READ_WRITE, les données sont lues et écrites dans les registres FIFO, et l'unité commutateur est mise à l'état "1". Ces opérations sont répétées **n/n_stag** fois, où n représente la moitié du nombre de points de la FFT et n_stag est l'indice de l'étage. Ensuite, la machine d'état passe à l'état DATA_READ.

Dans l'état DATA_READ , les registres sont en mode lecture et l'unité switch revient au mode "0". Le contrôleur reste dans cet état jusqu'à ce que les registres soient vides. Ensuite, la machine d'état revient à l'état IDLE.

La figure 3.6 illustre la machine d'état du contrôleur.

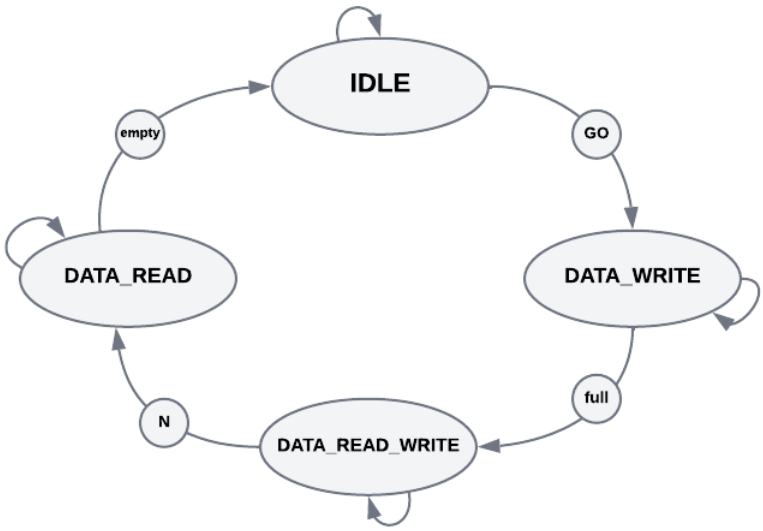


Fig. 3.6 : Contrôleur FSM

Le contrôleur FSM répète ce processus **n_stag** fois afin de traiter tous les échantillons de la FFT.

3.5.2 Unité Butterfly

L'unité butterfly est l'unité responsable des opérations arithmétiques (additions et soustractions) entre deux points DFT.
La figure 3.7 présente les signaux d'entrée et de sortie de l'entité, ainsi que le schéma

interne du circuit.

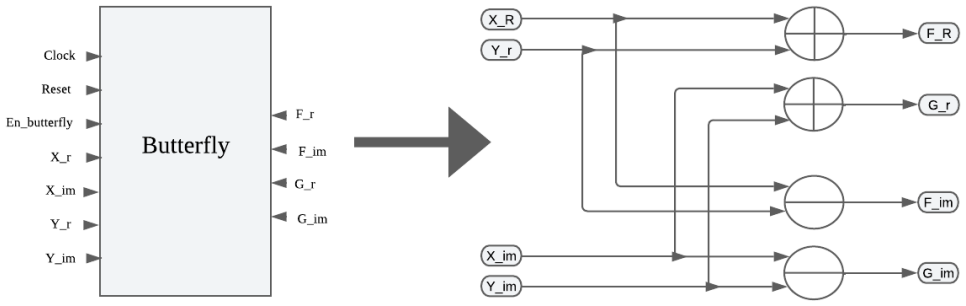


FIG. 3.7 : Unité Butterfly

Dans cette unité, les signaux d'entrée **Clock** et **Reset** permettent de gérer l'horloge et la réinitialisation synchronisée du bloc. Les quatre ports d'entrée **X_r**, **X_im**, **Y_r** et **Y_im** correspondent aux parties réelles et imaginaires des entrées.

Lorsque de nouvelles données sont disponibles, le signal d'entrée **en** est activé,ce qui déclenche l'opération dans l'unité papillon.

à l'intérieur de l'unité de calcul papillon , on observe deux composants principaux : un additionneur complexe et un soustracteur complexe chacun de ces composants est implémenté à l'aide de deux additionneurs réels qui additionnent respectivement la partie réelle et la partie imaginaire des signaux de manière appropriée afin de produire les sorties **F_r**, **F_im**, **G_r** et **G_im** qui représentent respectivement les parties réelles et imaginaires des données provenant de l'unité butterfly .

Une fois que ces sorties sont calculées, le signal **en_multiplication** est activé ce qui indique au bloc suivant qu'il peut commencer ses propres opérations.

On note que tous les stages dans l'architecture FFT contiennent le même bloc butterfly .

3.5.3 Unité de multiplication

Le bloc multiplicateur réalise la multiplication des les données proviennent de l'unité butterfly par les facteurs de rotation. Cette opération est plus complexe qu'une simple multiplication réelle. Elle est réalisée en utilisant quatre multiplications réelles et 2 opérations d'addition/soustraction comme indiqué dans la figure 3.8.

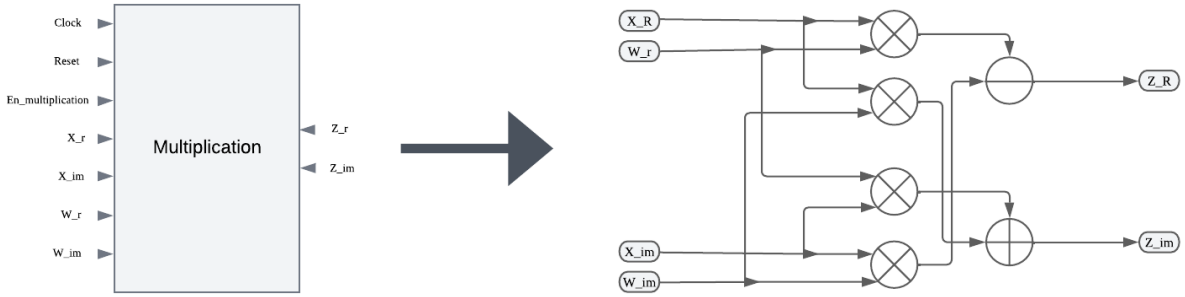


FIG. 3.8 : Unité multiplication

Les entrées **Clock** et **Reset** sont respectivement l'horloge et le signal de réinitialisation actif bas. Les ports **X_r** et **X_im** représentent respectivement les parties réelle et imaginaire des données provenant de l'unité papillon, tandis que **W_r** et **W_im** représentent les parties imaginaire et réelle des facteurs de rotation correspondants, qui proviennent de la ROM. Lorsque ces données sont prêtes à être utilisées en entrée, le signal **En_multiplication** provenant de l'unité butterfly est mis à un pour déclencher l'opération de multiplication complexe. Les résultats de cette multiplication sont affectés aux sorties **zr** et **zim**.

Il est important de noter que ce bloc se trouve uniquement dans les étages 1 et 4 d'un papillon de radix 2^3 simple, car pour les autres étages, les deux sorties de l'unité papillon doivent être multipliées par les facteurs de rotation, ce qui nécessite un autre bloc multiplicateur . donc ce bloc est remplacé par un autre appelé **multiplicateur fg** qui contient deux multiplicateurs simples intégrés.

3.5.4 ROM des coefficients de facteur de phase

Les facteurs de phase sont stockés dans des blocs de ROM synchrones en utilisant le cœur **IP Dual Port ROM** disponible dans **Vivado** . La ROM est conçue comme une mémoire en lecture seule, ce qui signifie qu'elle ne peut être que lue. Elle est initialisée avec les facteurs de rotation générés par **MATLAB** et enregistrés dans des fichiers au format **.coe**. Chaque fichier contient les parties réelles et imaginaires des facteurs de phase correspondants, comme expliqué dans la section précédente.

La taille de la ROM dépend du nombre de points et de la largeur des données d'entrée du processeur FFT. Pour un processeur FFT de 64 points, avec une largeur d'entrée fixée à 16 bits (5 bits pour la partie réelle, 10 bits pour la partie imaginaire et 1 bit de signe), les facteurs de phases sont également représentés sur 16 bits. La largeur de l'adresse est fixée à 6 bits, car il y a 32 coefficients de facteur (32 parties imaginaires et 32 parties réelles). Par conséquent, la taille de la ROM est de 16x32.

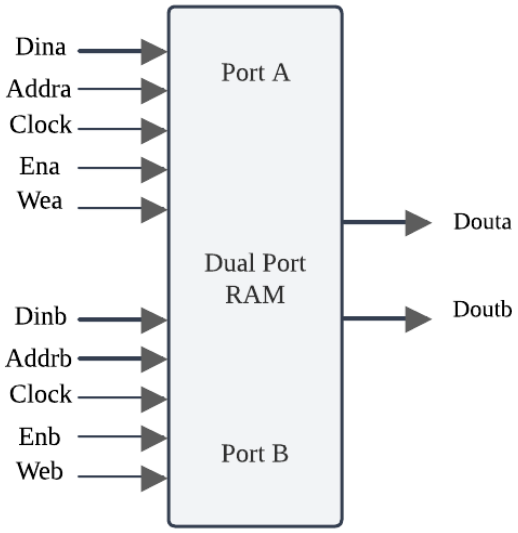


Fig. 3.9 : Dual port ROM

L'architecture du **Dual Port ROM** permet d'accéder aux deux données simultanément en utilisant les entrées **addra** et **addrb** pour récupérer les parties imaginaires et réelles des coefficients respectivement. Chaque opération de lecture prend 2 cycles d'horloge pour s'achever, conformément à la spécification de conception.

Les autres ports incluent les entrée **clka, clkb** pour la synchronisation, les entrée **ena, enb** pour activer le bloc, **wea, web** qui permettent d'activer ou de désactiver l'écriture de données dans la mémoire ROM, ainsi que les sorties **douta** et **doutb** qui représentent respectivement les parties réelles et imaginaires des facteurs de phase comme illustré sur la figure 3.9 [28] . Ces sorties sont ensuite utilisées comme entrées pour l'unité de multiplication.

3.5.5 Commutateur

Le commutateur de délai est l'unité responsable dans l'acheminement des données vers l'étage suivant. Il se présente sous la forme d'un multiplexeur qui est contrôlé par le signal **mode** provenant du bloc contrôleur.

La figure 3.10 présente le schéma interne du circuit.

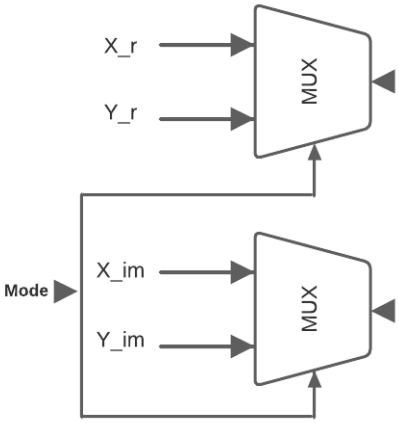


Fig. 3.10 : Fonctionnement interne du switch

La figure 3.11 illustre le fonctionnement du commutateur en mode 0, tandis que la figure 3.12 représente le fonctionnement en mode 1.

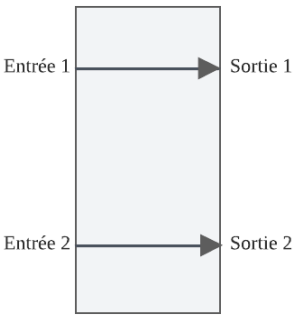


Fig. 3.11 : Commutateur en mode 0

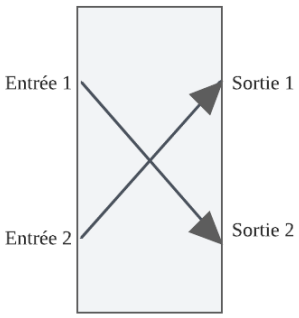


Fig. 3.12 : Commutateur en mode 1

3.5.6 Registre à décalage

Le registre à décalage FIFO (First In First Out) est un type particulier de registre qui fonctionne selon le principe du premier entré, premier sorti. Il est conçu pour stocker et déplacer des données de manière séquentielle, où la première donnée à être entrée est également la première à être extraite. Ce registre permet l'organisation des données avant leur transmission à l'étage suivant.

La méthode la plus courante pour construire un tampon FIFO consiste à ajouter un circuit de contrôle simple à un ensemble de mémoire générique, tel qu'une mémoire vive (RAM). Nous pouvons organiser l'ensemble de mémoire générique sous la forme d'une file circulaire et utiliser deux pointeurs pour marquer le début et la fin du tampon FIFO. Le

schéma conceptuel est illustré dans la figure 3.13. Le premier pointeur, appelé pointeur d'écriture (étiqueté "wr ptr"), pointe vers le premier emplacement vide devant le tampon. Lors d'une opération d'écriture, les données sont stockées dans cet emplacement prévu, et le pointeur d'écriture avance vers le prochain emplacement (c'est-à-dire incrémenté de 1). Le deuxième pointeur, appelé pointeur de lecture (étiqueté "rd ptr"), pointe vers la fin du tampon. Lors d'une opération de lecture, les données sont récupérées et le pointeur de lecture avance d'un emplacement, libérant ainsi efficacement l'emplacement pour de futures opérations d'écriture.

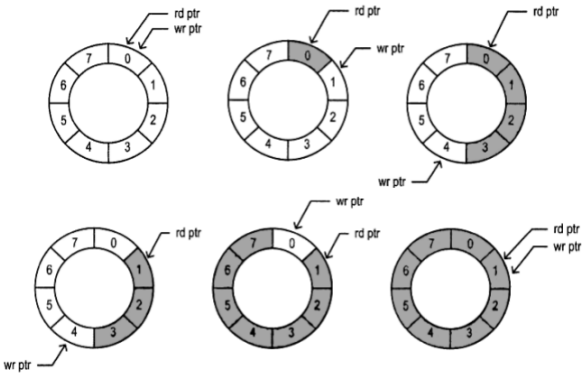


Fig. 3.13 : Schéma décrivant les opérateurs de lecture/écriture du registre à décalage

Fig. 3.14 : Registre à décalage

Le bloc séquentiel 3.14 possède plusieurs signaux d'entrée, notamment **write_en**, **read_en** et **data_in**, ainsi qu'un signal de sortie appelé **data_out**. Ces signaux d'entrée permettent de contrôler les opérations du bloc, tels que l'autorisation d'écriture **write_en**, l'autorisation de lecture **read_en** et l'entrée de données **data_in**. Le signal de sortie **data_out** renvoie la donnée extraite ou lue du bloc séquentiel.
Au front montant de l'horloge, si le signal "wr" est activé et que le tampon n'est pas plein, les données d'entrée correspondantes seront échantillonnées et stockées dans le tampon. Le signal "re" peut être interprété comme un signal de "retrait". S'il est activé au front montant et que le tampon n'est pas vide, le pointeur de lecture du FIFO avance d'une position et rend l'emplacement actuel disponible. Après les délais internes de l'incrémentation et du routage, de nouvelles données de sortie sont disponibles sur le port de sortie du FIFO.

3.5.7 Stratégie de pipelining

L'un des défis de la mise en œuvre d'une architecture pipeline est que le pipeline continue de fonctionner et de générer des sorties même lorsque les entrées de ce dernier ne sont pas encore validés, Cela exige une attention particulière en ce qui concerne le pipelining afin de s'assurer que les données sont traitées en temps opportun pour que l'étape suivante puisse les recevoir.
La figure 3.15 représente les signaux de requête de données entre les étages du processeur FFT.

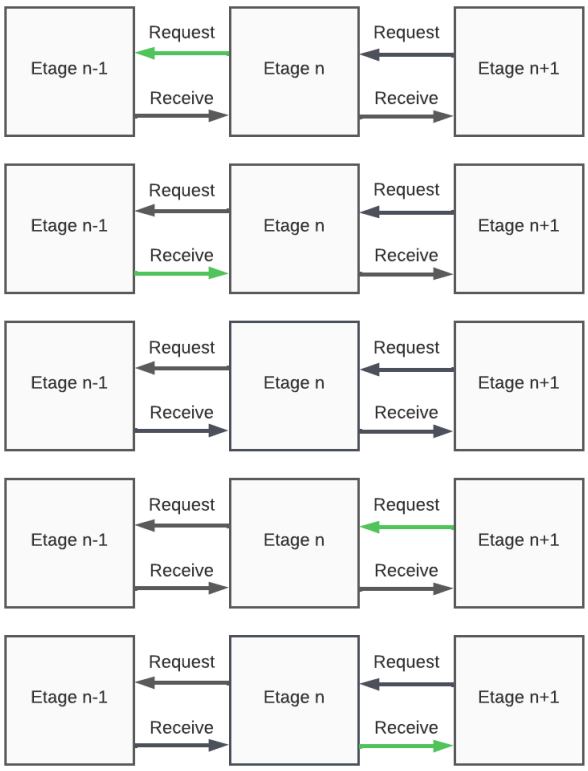


FIG. 3.15 : Illustration de la stratégie pipeline

3.5.8 Manipulation de la précision des données

Conformément à ce qui a été évoqué dans la section 3.4, Il est important de prendre en compte le débordement arithmétique lors de la manipulation de données numériques (additions, soustraction) de manière à éviter des résultats inattendus ou des erreurs de calcul.

Étant donné que les données sont normalisées, avec une valeur maximale de '1', nous avons cherché à exploiter toute la précision de notre système. Nous avons alloués deux bit pour représenter la partie entière des données d'entrée d'entrées. À mesure que nous progressons à travers chaque étape d'un butterfly, la partie entière s'étendra de manière dynamique, nous permettant de tirer le meilleur parti des bits disponibles pour la précision come montrée dans la figure 3.16. Cette approche nous permet d'utiliser efficacement

le nombre maximum de bits tout en maintenant la précision et l'efficacité de notre représentation.

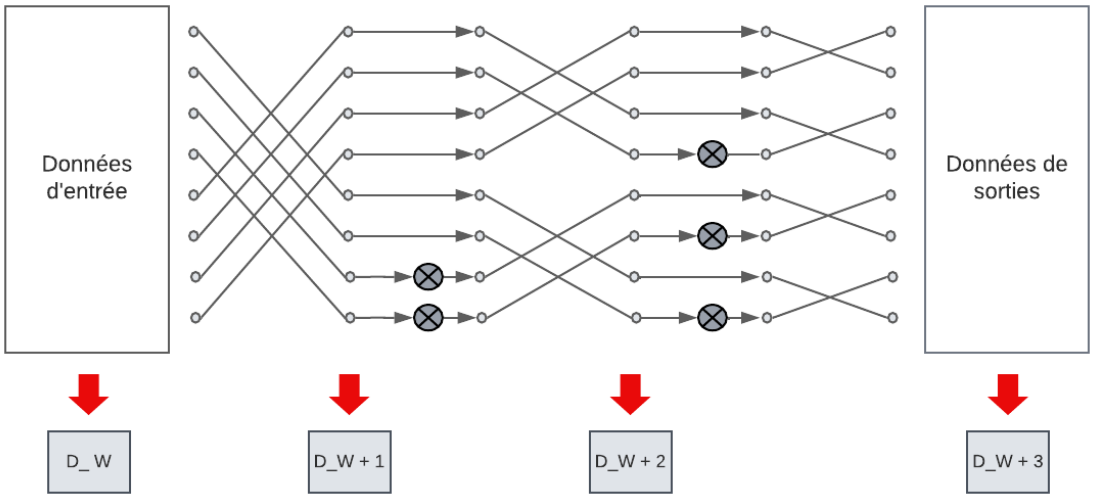


FIG. 3.16 : Ajustement de la longueur du mot

À l'entrée de l'architecture à pipeline, la largeur des bits des données est supposée **D_W**, ce qui indique la largeur des bits des parties réelles et imaginaires des données. Lorsque les données passent à travers l'unité butterfly, elles participent à des opérations d'addition et de soustraction. Afin de s'ajuster à l'augmentation du nombre de bits résultant de ces opérations, la partie entière s'étend d'un bit à chaque sortie d'un étage du pipeline.

3.6 Simulation

La simulation et l'utilisation d'un testbench sont des étapes cruciales dans le processus de développement de circuits numériques. La simulation permet de vérifier le fonctionnement attendu d'une conception avant sa mise en œuvre réelle, ce qui permet de détecter et de corriger les erreurs dès les premières étapes du développement.

En utilisant un testbench, qui est un module écrit en VHDL spécialement conçu pour générer des stimuli et vérifier les résultats, on peut fournir des entrées contrôlées à la conception et observer les sorties correspondantes. Cela permet de valider le comportement fonctionnel, d'analyser les performances, et de détecter d'éventuels problèmes ou comportements inattendus. De plus, le testbench permet de reproduire différentes situations de fonctionnement et d'explorer les scénarios critiques.
Nous utilisons le simulateur Vivado pour effectuer la simulation et vérifier le comportement de l'architecture conçue. Afin de garantir que le circuit est temporellement correct, notre test bench inclura les instructions suivantes. :

Activation du signal "reset" afin d'initialiser tous les signaux de l'architecture. Cela permet de garantir un état initial cohérent avant le début du traitement. Après avoir activé le signal "reset", nous attendons un cycle d'horloge pour permettre à l'initialisation de se propager à travers le circuit. Ensuite, nous désactivons le signal "reset" pour permettre au circuit de commencer ses opérations normales.

Activation du signal "GO" : Nous activons le signal "GO" pour démarrer le processus de traitement de l'architecture. Ce signal indique au circuit qu'il doit commencer ses opérations.

Ajout d'une instruction "wait" : Nous ajoutons une instruction "wait" pour permettre au design de se lancer et de progresser dans ses opérations. Cette attente est généralement nécessaire pour s'assurer que toutes les étapes du circuit sont correctement exécutées.

Utilisation du compteur du dernier étage comme indicateur : Nous utilisons le compteur du dernier étage de l'architecture comme un indicateur pour vérifier si toutes les opérations de l'architecture sont terminées. Lorsque le compteur atteint sa valeur finale, cela signifie que toutes les étapes du circuit ont été exécutées.

Réinitialisation du signal "GO" : Une fois que les opérations sont terminées, nous remettons le signal "GO" à zéro pour éviter de relancer le circuit. Cela garantit que seules les données souhaitées sont extraites lors d'une seule itération du circuit.

En suivant ces étapes, nous pouvons vérifier si le circuit conçu fonctionne correctement sur le plan temporel, en s'assurant que les opérations sont exécutées dans l'ordre approprié et que les données sont traitées comme prévu.
la figure 3.17, illustre la simulation du circuit sur Vivado.

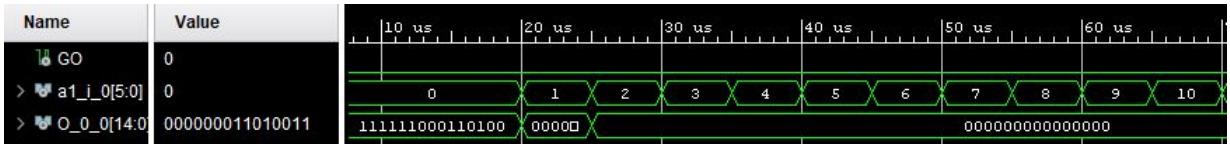


Fig. 3.17 : Simulation du processeur FFt pour N = 64 points

3.7 Synthèse et implémentation

Après avoir décrit le circuit en VHDL, la prochaine étape consiste à effectuer la synthèse et l'implémentation du code sur Vivado.

La synthèse du circuit transforme le code en une représentation structurelle appelée netlist. Cette netlist indique comment les différents blocs logiques et bascules du circuit sont connectés.
La netlist obtenue est ensuite utilisée dans l'étape d'implémentation où elle est adaptée

aux ressources spécifiques du FPGA et routée pour créer la configuration physique du circuit. La carte FPGA Artix7-50T a été choisie comme plateforme pour mener à bien cette imolémentation. Cette carte offre une capacité de traitement suffisante pour accueillir l'architecture et les fonctionnalités de notre processeur FFT.

L'étape d'implémentation fournit également des rapports détaillés qui donnent des informations telles que l'utilisation des ressources du FPGA, l'analyse temporelle et la consommation de puissance de notre conception. Ces informations sont détaillé ci dessous :

3.7.1 Rapport des ressources utilisées

Le rapport d'utilisation est un document généré par l'outil Vivado, qui donne un aperçu détaillé de la manière dont les ressources du FPGA sont utilisées par le design créé. Il fournit des informations précieuses sur l'utilisation des différentes ressources FPGA, telles que les tranches LUTS, la mémoire et les DSP, ainsi que sur la configuration des signaux d'entrée et de sortie.

La figure 3.18 présente le rapport de consommation matérielles générer par Vivado.

Resource	Utilization	Available	Utilization %
LUT	1710	32600	5.25
LUTRAM	192	9600	2.00
FF	1764	65200	2.71
BRAM	6	75	8.00
DSP	32	120	26.67
IO	26	210	12.38

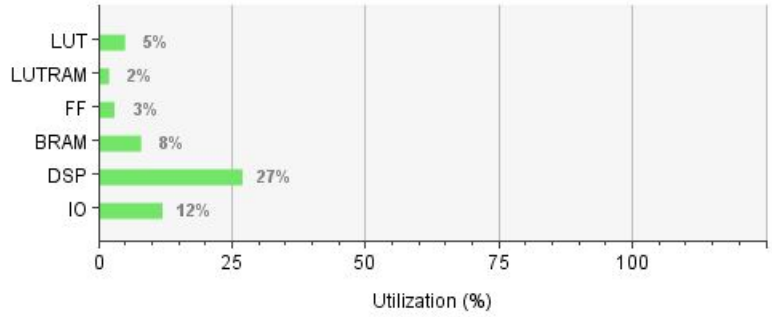


Fig. 3.18 : Consommation matérielles

3.7.2 Rapport de la consommation de puissance

Le rapport de consommation d'énergie, connu sous le nom de "Power Report", est généré par l'outil Vivado dans le cadre de la conception FPGA. Ce rapport détaille la consommation de puissance des ressources utilisées de la carte FPGA.
La figure 3.19 présente le rapport de consommation matérielles générer par Vivado.

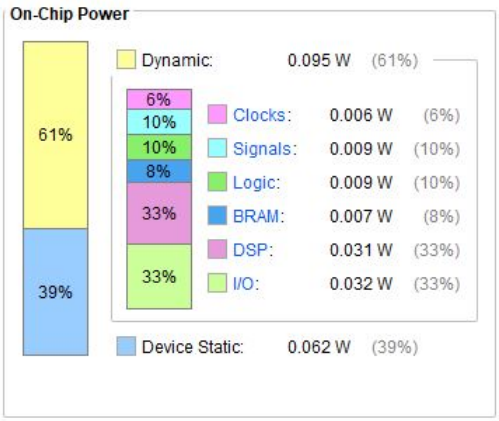


FIG. 3.19 : Consommation de puissance

3.7.3 Analyse temporelle

Dans le cadre du processus d'implémentation, les outils FPGA effectuent une analyse temporelle (*timing analysis*) du design. Cette analyse nous fournit deux indicateurs temporels : le temps d'établissement *WSN - Setup Time* et le temps de maintien *WHN - Hold Time*.

- Le temps d'établissement est la durée nécessaire pour que l'entrée d'une bascule soit stable avant un front d'horloge.
- Le temps de maintien est la durée minimale requise pour que l'entrée d'une bascule soit stable après un front d'horloge.

Si ces durée sont négative, cela signifie que le design ne réussit pas la contrainte temporelle *timing constraint*.

La figure 3.20 représente l'analyse temporelle de notre design. Il est clairement visible que les deux indicateurs *WHN* et *WSN* sont positifs ce qui indique la validité temporelle de notre implémentation.

FIG. 3.20 : L'analyse temporelle du design

3.8 Conclusion

Ce chapitre a abordé la structure et la conception de l'implémentation de la FFT sur FPGA. Nous avons présenté une architecture proposée pour cette mise en œuvre, mettant l'accent sur les étapes de conception et de modélisation, ainsi que sur la conception du processeur FFT et les simulations associées.

Les informations et les résultats obtenus jusqu'à présent nous fournissent une base solide pour poursuivre l'amélioration de notre implémentation de la FFT sur FPGA. Le prochain chapitre se concentrera spécifiquement sur les stratégies d'optimisation que nous pouvons mettre en œuvre pour améliorer les performances et l'efficacité de notre système.

Chapitre 4

Amélioration de l'architecture

4.1 Introduction

Dans ce chapitre, nous explorerons d'autres méthodes de multiplications qui permettent de contourner l'utilisation des DSP et des blocs RAM pour des FFT avec un grand nombre de points. Nous discuterons des techniques qui exploitent les ressources génériques du FPGA pour réaliser les calculs de la FFT de manière efficace et sans compromettre les performances.

4.2 Optimisation des ressources

Dans les architectures FFT traditionnelles, il est courant d'employer des mémoires ROM pour stocker les facteurs de rotation, ces derniers servent à effectuer les multiplications complexes à chaque étage de l'architecture. Par conséquent, pour une FFT à N points, il est nécessaire de disposer de $Log_2(N)$ blocs de mémoire ROM. Chaque bloc ROM contient 2N facteurs de rotation, représentant les parties réelles et imaginaires.

En outre, les multiplications complexes sont réalisées dans des blocs DSP, qui sont des ressources matérielles spécialisées sur les FPGA conçues pour accélérer les opérations de traitement du signal. Chaque bloc DSP peut effectuer des opérations de multiplication et d'accumulation de manière très efficace.

Cependant, à mesure que le nombre de points FFT augmente, le besoin en blocs ROM et en blocs DSP augmente également. Ces ressources sont souvent limitées sur les FPGA, ce qui rend cette approche traditionnelle inefficace en termes de consommation des ressources matérielles.

Afin de remédier à cela, l'architecture présentée dans le chapitre précédant est modifiée dans le but d'éliminer l'utilisation des blocs ROM et de substituées les multiplications complexes basées sur les DSP par des circuits logiques exploitant les ressources LUTs.

Avant d'aborder la description des circuits logiques, il est important de définir ce que sont les blocs LUTs et les blocs DSP.

Blocs LUTs : Ces blocs sont utilisées pour réaliser des opérations logiques, telles que les portes logiques (ET, OU, NON),les opérations booléennes et les fonctions de décision. Elles peuvent également être configurées pour implémenter des fonctions arithmétiques, des multiplexeurs et d'autres opérations logiques complexes.
La figure 4.1 présente le schéma interne d'un bloc LUT.

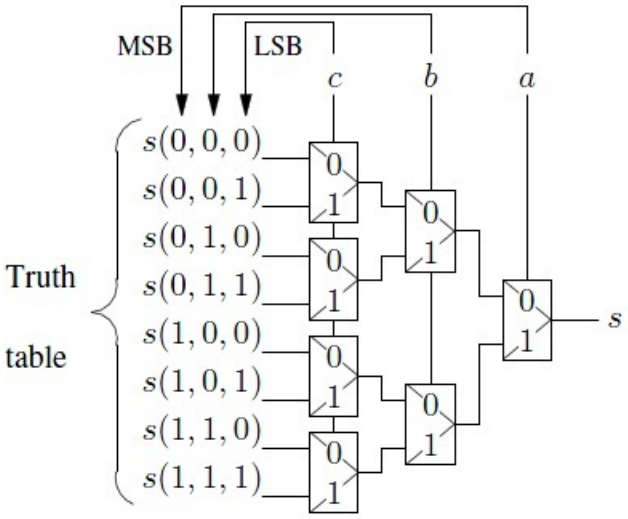


FIG. 4.1 : Schéma interne d'un bloc LUT [29]

Blocs DSP : Un bloc DSP est un composant matériel constitué d'un multiplieur dédié, d'un accumulateur, d'un opérateur arithmétique, d'une logique de contrôle et de registres d'entrée/sortie. Il permet d'effectuer des multiplications rapides, des opérations d'addition et de soustraction, et dispose de fonctionnalités telles que des modes d'arrondi et des accumulateurs en virgule flottante.
La figure 4.2 présente le schéma interne d'un bloc LUT.

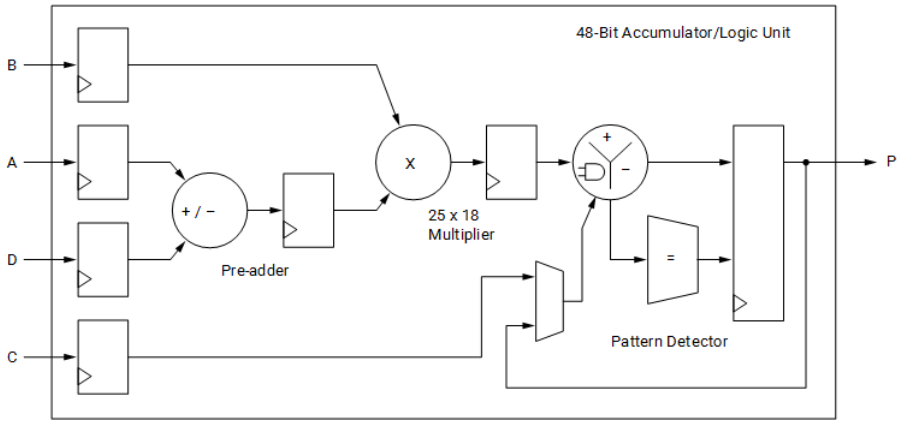


FIG. 4.2 : Schéma interne d'un bloc DSP [30]

Dans l'architecture améliorée, les multiplications complexes aux étages d'indice 3i et 3i+1 (étages 0, 1, ...) seront réalisées à l'aide de circuits logiques basés sur des additions et des décalages. De plus, les multiplications par les facteurs non constants aux étages d'indice 3i+2 (étages 2, 5, ...) seront remplacées par l'utilisation d'un bloc CORDIC. Ces deux approches seront expliquées en détail dans les sections suivantes.

4.3 Multiplicateurs à coefficient constant

Les coefficients de rotation tels que W_8^k et W_4^k sont des multiplicateurs à coefficient constant permettant de réaliser des opérations de multiplication d'une manière plus efficace car ces derniers sont connus à priori.
En utilisant uniquement des additionneurs, des décalages, des inversions entre les parties réelles et imaginaires et des changements de signes, il est possible de concevoir des circuits matériels optimisés pour effectuer ces multiplicateurs. Cela nous permet d'obtenir des performances améliorées en termes de temps d'exécution et de consommation d'énergie tout en limitant la complexité et les ressources requises pour les calculs.

Les facteurs de phases des différents étages de la FFT radix-2^3 à N-point sont présentées dans le tableau 4.1.

Algorithme	Étage							
	1	2	3	4	5	6	7	8
Radix-2^3	W_4	W_8	W_N	W_4	W_8	$W_{N/8}$	W_4	W_8

Tab. 4.1 : Facteurs de phases des différents étages de la FFT radix-2^3

4.3.1 Multiplication par W_4^k

D'après le tableau 4.1, le premier stage du processeur élémentaire contient un multiplieur par les facteurs de rotations $W_4^k = e^{-j\frac{2\pi k}{4}}$.

Le terme W_4^k est périodique, lorsque l'indice k dépasse la valeur 3, ce dernier peut être réécrit sous la forme suivante : $W_4^k = W_4^{4+m} = W_4^m$, où m varie de 0 à 3. Cela signifie que la multiplication par les facteurs de rotation W_4^k revient à la multiplication par l'un des quatre facteurs W_4^0, W_4^1, W_4^2 et W_4^3 comme exprimé dans les équations (4.8). Ces facteurs peuvent être considérés comme des coefficients constants.

$$\begin{aligned}
(a+jb)W_4^0 &= (a+jb)\cdot(1) &&= a+jb \\
(a+jb)W_4^1 &= (a+jb)\cdot(-j) &&= b-ja \\
(a+jb)W_4^2 &= (a+jb)\cdot(-1) &&= -a-jb \\
(a+jb)W_4^3 &= (a+jb)\cdot(j) &&= -b+ja
\end{aligned} \tag{4.1}$$

Par conséquent, les données pour lesquelles k = 0 n'ont pas besoin d'être multiplier.De même, si k ϵ [4,2, 3], les données sont déphasées de 270 ◦ , 180 ◦ et 90 ◦ , ce qui correspond

à des multiplications complexes par -j , -1 et j , respectivement. Ces rotations sont considérées comme triviales, car elles peuvent être calculées en intervertissant les composantes réelles et imaginaires et/ou en changeant le signe des données.

Le schéma illustré dans la figure 4.3 représente le circuit de multiplication par les facteurs W_4^k.

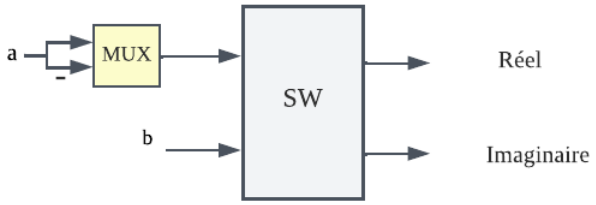


FIG. 4.3 : Circuit de multiplication par le facteur W_4^k.

4.3.2 Multiplication par W_8^k

Le terme W_8^k est également périodique, de ce fait, la multiplication par les facteurs de rotation W_8^k revient à la multiplication par l'un des huit facteurs W_8^0, W_8^1 ... W_8^6 et W_8^7 comme exprimé dans les équations (4.2) .

$$\begin{aligned}(a+jb)W_8^1 = (a+jb)W_8^5 = (a+jb) * \frac{\sqrt{2}}{2}(1-j) \\ (a+jb)W_8^3 = (a+jb)W_8^7 = (a+jb) * \frac{\sqrt{2}}{2}(-1+j)\end{aligned} \tag{4.2}$$

Les multiplications par les facteurs constants W_8^n sont effectuées à l'étage d'indice 3i+1. Ces multiplications sont remplacées par des circuits logiques afin d'éviter une utilisation excessive des ressources matérielles.
Les données correspondant à k = [0, 2, 4, 6] sont associées à des multiplications par 1, -j, -1 et j respectivement. Ces multiplications seront calculées de la même manière que les multiplications par les facteurs W_4^k. Cela signifie que les mêmes circuits logiques seront utilisés pour effectuer ces opérations.
De plus, les multiplications par W_8^1, W_8^3, W_8^5 et W_8^7 peuvent être simplifiées en utilisant des opérations de décalage et un ensemble des additionneurs.

Les figures 4.4 et 4.5 présentent le circuit de multiplication par le facteur W_8^n et la multiplication par le facteur $\frac{\sqrt{2}}{2}$ respectivement.

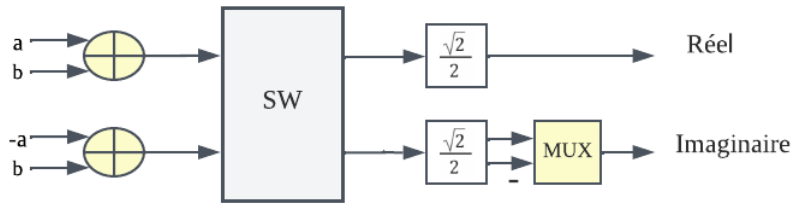


FIG. 4.4 : Circuit de multiplication par le facteur W_8^n

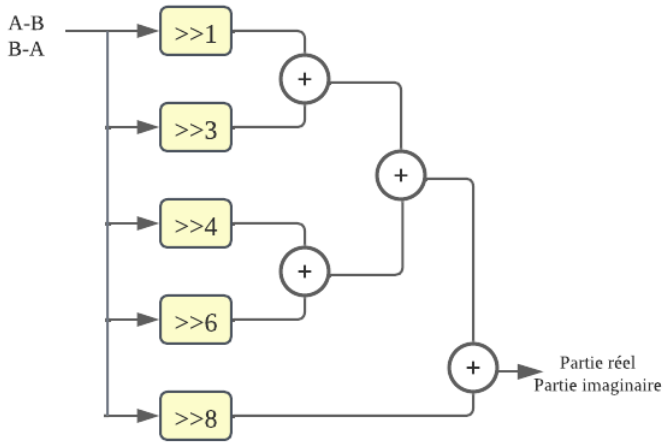


FIG. 4.5 : Circuit de multiplication par le facteur $\frac{\sqrt{2}}{2}$

4.3.3 Multiplication par W_N^n

Comme mentionné précédemment, l'algorithme 2^3 nécessite des multiplications par des facteurs non constants à chaque trois étages de son exécution. Dans ce contexte, il devient inefficace de concevoir un circuit distinct pour chaque multiplication, car cela entraînerait une utilisation excessive des ressources matérielles.
Pour résoudre ce problème, Nous avons opté pour l'introduction de l'algorithme CORDIC. Cet algorithme permet de calculer un ensemble de fonctions arithmétiques, y compris la multiplication, la division, le sinus, le cosinus, l'arctangente et les fonctions hyperboliques.

Lorsque l'algorithme CORDIC est utilisé pour effectuer des rotations complexes, la procédure est basée sur le fait que tout angle de rotation, θ, peut être décomposé en une somme de M angles prédéfinis, θ_i selon l'équation 4.3.

$$\theta_i = \sum_{m}^{M} \theta_i \tag{4.3}$$

Où m et M représentent respectivement le premier et le dernier angle utilisé.

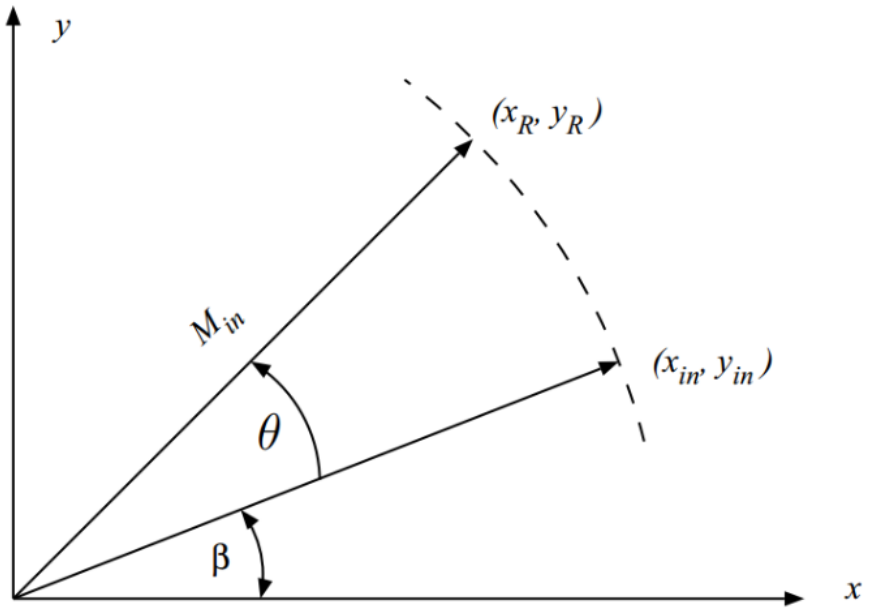


FIG. 4.6 : Rotation d'un vecteur dans le plan Oxy

Lorsque nous effectuons une rotation du vecteur (xi, yi) par un angles θ, comme le montre la figure 4.6, sa nouvelle coordonnée peut être exprimée comme suit :

$$\begin{aligned} x_{i+1} &= x_i \cos(\theta) - y_i \sin(\theta) \\ y_{i+1} &= y_i \cos(\theta) + x_i \sin(\theta) \end{aligned} \tag{4.4}$$

En factorisant le $\cos(\theta)$ on obtient :

$$\begin{aligned} x_{i+1} &= \cos(\theta)(x_i - y_i \tan(\theta)) \\ y_{i+1} &= \cos(\theta)(x_i + y_i \tan(\theta)) \end{aligned} \tag{4.5}$$

La propriété de la fonction tangente, nous donne que : $\tan(\theta) = \delta_i 2^{-i}$ où $\delta_i \in -1, 1$.

Par conséquent, le processus itératif de l'algorithme CORDIC pour la rotation peut être représenté par les equations 4.6 :

$$\begin{aligned} x_{i+1} &= K(x_i - \delta_i y_i 2^{-i}) \\ y_{i+1} &= K(y_i + \delta_i x_i 2^{-i}) \\ z_{i+1} &= z_i - \delta_i \theta_i \end{aligned} \tag{4.6}$$

Où K représente le facteur de compensation et est égale a : $K = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}$ Pour un nombre assez grand d'itération K = 0.60725.
La figure 4.7 présente les composants du système Cordic.

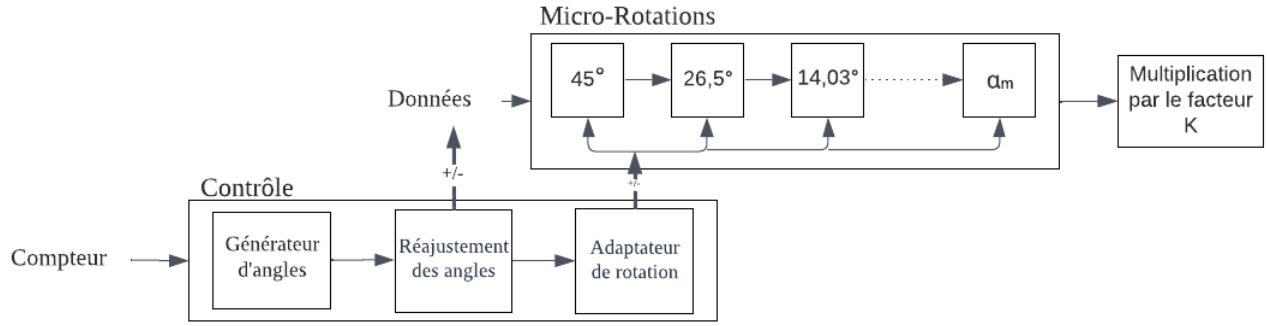


FIG. 4.7 : Composants du système Cordic

Générateur d'angles

Le générateur d'angles obtient les séquences d'angles de chaque étage de la FFT grâce à l'équation (4.7).

$$z = -\frac{2\pi}{N} \cdot n \cdot (k_1 + 2k_2 + k_3) \tag{4.7}$$

À l'exception de N, les autres valeurs changent à chaque cycle d'horloge en fonction de la valeur du tableau 4.2 :

Paramétre	Générateur 1	Générateur 2	Générateur 3
N	4096	512	64
n	0, 1, 2, .., 511	0, 1, 2, .. , 63	0, 1, 2,.. , 7
$K_1 + 2K_2 + K_3$	0, 4, 2, 6, 1, 5, 3, 7	0, 4, 2, 6, 1, 5, 3, 7	0, 4, 2, 6, 1, 5, 3, 7

TAB. 4.2 : Les valeurs de changement de chaque paramètre dans l'équation 4.7

Réajustement des angles

Après avoir obtenu les angles de rotations, la prochaine étape consiste à reformuler le problème de sorte que la rotation souhaitée soit inférieure à 45 degrés. Cela est nécessaire car la rotation maximale que l'algorithme CORDIC peut effectuer est une rotation de 45 degrés. Au-delà de 45 degrés, les angles deviennent de plus en plus petits.
Par conséquent, il est essentiel d'effectuer une pré-rotation sur le vecteur d'entrée afin que la rotation restante soit inférieure ou égale à 45 degrés.

Pour que tout soit à +/- 45 degrés, nous devons vérifier non seulement dans quel quadrant se trouve notre demande de phase, mais aussi dans quel segment de 45 degrés de ce quadrant se trouve l'angle, comme le montre la figure 4.8

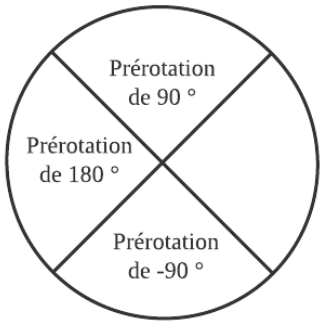


FIG. 4.8 : Prérotation des angles

Afin de placer l'angle dans l'intervalle approprié, nous pouvons utiliser des rotations triviales de 180 degrés et 90 degrés. Ces rotations permettent de déplacer l'angle dans la plage souhaitée comme indiquer dans les équations 4.8.

$$
\begin{aligned}
\mathbf{0} < \theta < \mathbf{45} \\
x &= x; \\
y &= y; \\
\text{phase} &= \text{phase}; \\
\mathbf{45} < \theta < \mathbf{90} \\
x &= -y; \\
y &= x; \\
\text{phase} &= \text{phase}; \\
\mathbf{90} < \theta < \mathbf{135} \\
x &= -y; \\
y &= x; \\
\text{phase} &= \text{phase}; \\
\mathbf{135} < \theta < \mathbf{180} \\
x &= -x; \\
y &= -y; \\
\text{phase} &= \text{phase} - 90; \\
\mathbf{180} < \theta < \mathbf{225} \\
x &= -x; \\
y &= -y; \\
\text{phase} &= \text{phase} - 90; \\
\mathbf{225} < \theta < \mathbf{270} \\
x &= y; \\
y &= -x; \\
\text{phase} &= \text{phase} - 180; \\
\mathbf{270} < \theta < \mathbf{315} \\
x &= y; \\
y &= -x; \\
\text{phase} &= \text{phase} - 180; \\
\mathbf{315} < \theta < \mathbf{360} \\
x &= x; \\
y &= y; \\
\text{phase} &= \text{phase};
\end{aligned}
\tag{4.8}
$$

Adaptateur de rotation

L'algorithme CORDIC est implémenté au moyen d'une structure de pipeline parallèle. La Figure 4.9 [31]est une représentation du pipeline à 16 niveaux utilisé pour mettre en œuvre l'algorithme CORDIC. Chaque étape de l'itération CORDIC est réalisée par une unité d'itération conforme aux règles de l'algorithme. La structure du pipeline est identique à tous les niveaux. La Figure 4.10 [32] illustre le schéma de la structure logique de l'unité d'itération CORDIC, qui comprend un décaleur, un additionneur et un soustracteur. À chaque cycle d'horloge, le contrôleur détermine le décalage à effectuer vers la droite et le type d'opération à réaliser en fonction du bit de symbole Zi de la valeur de l'accumulateur d'angle.
L'angle initial Z0 est défini comme l'angle de rotation souhaité du vecteur, et la valeur de l'accumulateur d'angle se rapproche de zéro après 16 itérations. La précision de l'algorithme peut être améliorée en augmentant le nombre d'itérations. En général, il faut effectuer N itérations CORDIC pour obtenir une précision de sortie de n bits.

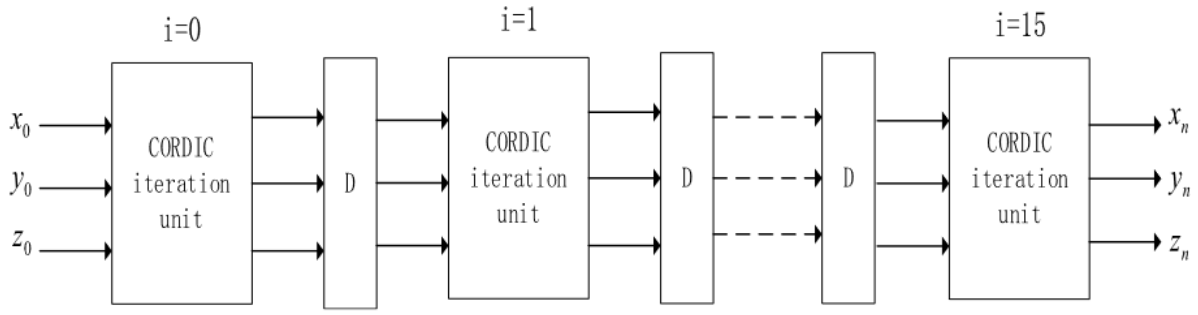


FIG. 4.9 : Structure du pipeline a 16 étage de l'algorithme Cordic

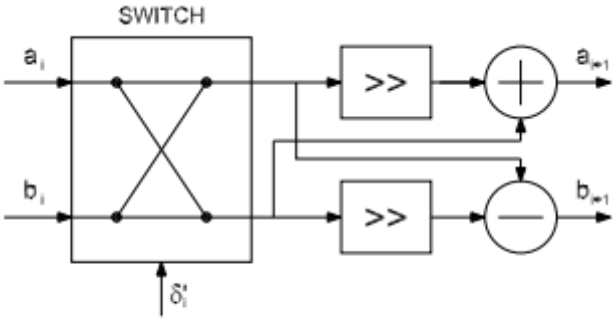


Figure 8.5: Basic CORDIC processor.

FIG. 4.10 : schéma de la structure de l'unité d'itération Cordic

Facteur de compensation

Comme mentionné précédemment, l'algorithme CORDIC est caractérisé par un gain constant K qui lui est associé. Ce facteur constant peut être implémenté à l'aide du circuit illustré par la figure 4.11 :

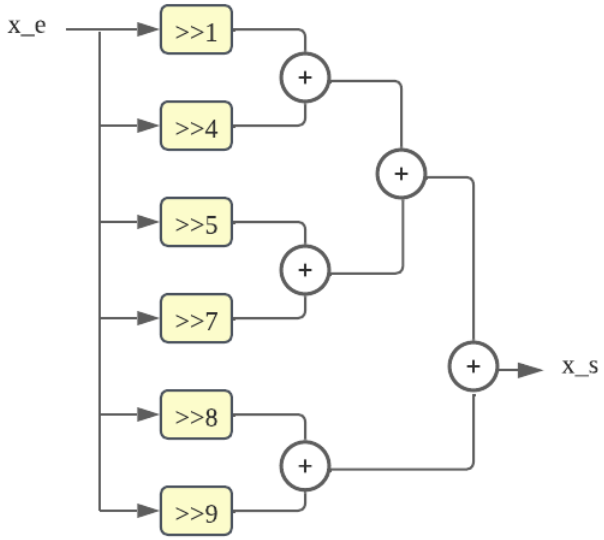


FIG. 4.11 : Facteur de compensation K

4.4 Simulation

Après avoir effectué des modifications sur notre architecture, il est essentiel de relancer la simulation afin de vérifier le bon fonctionnement du processeur avant de procéder à sa synthèse et à son implémentation.

la figure 4.12, illustre la simulation du circuit sur Vivado.

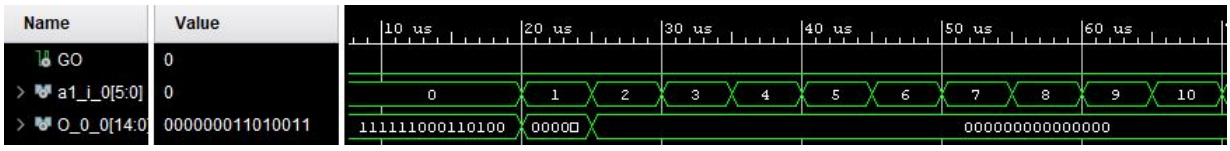


FIG. 4.12 : Simulation du processeur FFt pour N= 64

4.5 Synthèse et implémentation

4.5.1 Rapport des ressources utilisées

En observant le tableau de consommation des ressources, on constate que notre objectif d'éliminer la consommation des blocs ROM et DSP a été atteint. Les 2 BRAM restantes sont les mémoires de stockages des données d'entrées et sorties. La conversion des opérations de multiplication en circuits logiques est responsable de l'augmentation des tranches LUT. La figure 4.13 présente le rapport de consommation matérielles générer par Vivado.

Resource	Utilization	Available	Utilization %
LUT	4899	32600	15.03
LUTRAM	192	9600	2.00
FF	3332	65200	5.11
BRAM	2	75	2.67
IO	28	210	13.33

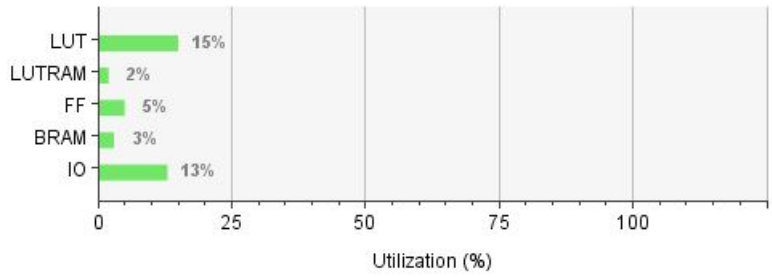


Fig. 4.13 : Consommation matérielles de l'implémentation améliorée

4.5.2 Rapport de la consommation de puissance

La consommation de puissance reste constante par rapport à l'implémentation précédente. Cependant, la puissance qui était auparavant consommée par les DSP et les BRAM est désormais utilisée par les signaux et les ressources logiques de la carte.

La figure 4.14 présente le rapport de consommation matérielles générer par Vivado.

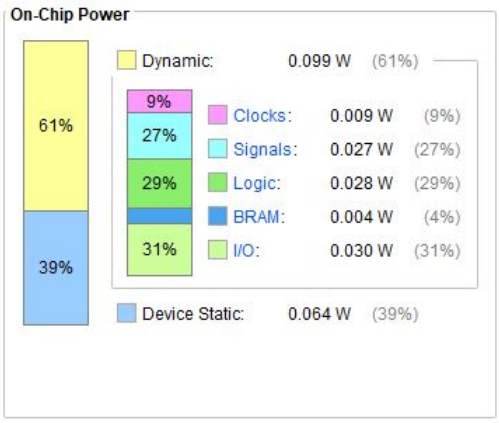


FIG. 4.14 : Consommation de puissance de l'implémentation améliorée

4.5.3 Analyse temporelle

La figure 4.15 présente l'analyse temporelle du processeur.

FIG. 4.15 : L'analyse temporelle du processeur FFT améliorée

Une diminution du WSN est observée en raison de l'augmentation du temps de propagation des signaux entre deux registres synchrones. Cette augmentation est causée par l'utilisation d'un circuit de multiplication combinatoire.

4.6 Conclusion

Dans ce chapitre, nous avons mis en évidence l'importance de la réalisation de circuits pour les facteurs à coefficient constant, ce qui permet de réduire les ressources nécessaires et d'éviter les opérations de multiplication coûteuses. De plus, nous avons présenté l'algorithme CORDIC comme une alternative efficace aux DSP et aux blocs RAM pour les multiplications W_N^n.

Cette approche offre la possibilité de réaliser des FFT pour un grand nombre de points sans dépendre des DSP et des blocs RAM, ce qui se traduit par une flexibilité accrue et

une meilleure adaptabilité aux contraintes spécifiques des applications.

Selon le tableau 4.3, nous évaluons l'erreur relative des résultats des multiplications complexes réalisées à l'aide des blocs optimisés par rapport à ceux obtenus à partir de DSP. L'erreur relative est une mesure du pourcentage de divergence entre une valeur approximative et une valeur exacte.

Il est observé que ces méthodes d'optimisation n'ont aucun impact sur la précision des résultats du processeur FFT.

Multiplication	Multiplicateurs à coefficient constant	Cordic
Erreur relative %	0	0.07

Tab. 4.3 : Erreur relative de la multiplication cordic et les multiplicateurs à coefficient constant

Chapitre 5

Évaluation expérimentale et résultats

5.1 Introduction

Dans ce chapitre, nous procédons à l'évaluation détaillée de notre implémentation. Nous évaluerons d'abord la précision de nos résultats en les comparant à ceux générés par l'exécution de la fonction FFT de Matlab, le SQNR (le rapport signal à bruit de quantification) étant la métrique utilisée à cet effet . Nous présenterons, par la suite, les résultats concernant les ressources matérielles consommées comparativement aux autres réalisations de la FFT décrites dans la littérature ainsi qu'à la FFT de l'IP core Xilinx. Ces évaluations constituent un moyen pour mesurer le degré de fiabilité et d'efficacité de notre implémentation par rapport à d'autres approches.

5.2 Évaluations de la précision

Pour évaluer le fonctionnement de notre processeur FFT, nous utilisons quatre signaux différents dans la partie expérimentale. Ces signaux comprennent un signal sinusoïdal réel, un signal sinusoïdal complexe et le signal chirp. Les expressions des quatres signaux de test sont présentées dans le tableau 5.1.

Le signal Chirp : Le signal de chirp, également connu sous le nom de signal à modulation de fréquence linéaire (LFM), est un type de signal qui présente une fréquence croissante ou décroissante de manière linéaire au fil du temps.Il est utilisé dans les systèmes radar pour atteindre une résolution de portée élevée, c'est-à-dire pour distinguer des cibles proches les unes des autres en termes de distance [2].

Signal de test	L'expression du signal de test
Signal sinusoïdale réel	$0.5\sin(2\pi\mathrm{t}) + 0.5\cos(2\pi 3\mathrm{t})$
Signal sinusoïdale complexe	$0.5\exp(\mathrm{j}2\pi\mathrm{t}) + 0.5\exp(\mathrm{j}2\pi 3\mathrm{t})$
Signal chirp	$chirp(t, f_0, (N-1)Te, f_1)$

Tab. 5.1 : Signaux de test

Nous mesurons la précision de notre processeur FFT en comparant ses résultats avec ceux de la FFT de MATLAB. Cela est réalisé en calculant l'erreur de quantification SQNR, pour différents nombres de points (64, 512 et 4096 points).

Signal sinusoïdale

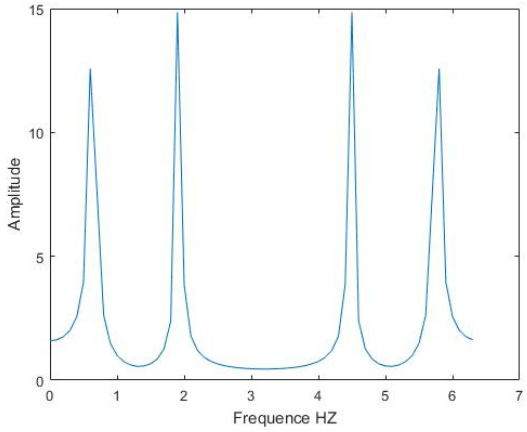


FIG. 5.1 : FFT à 64 points d'un signal sinusoïdale sur FPGA

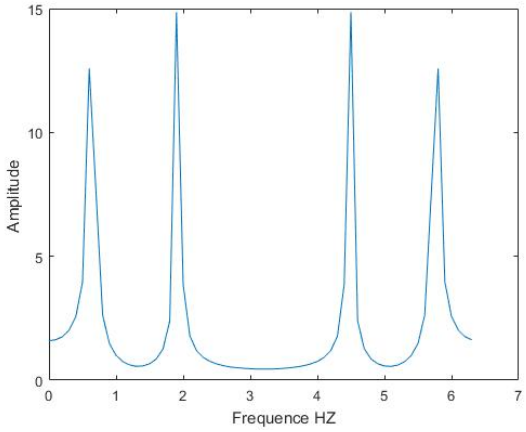


FIG. 5.2 : FFT à 64 points d'un signal sinusoïdale sur Matlab

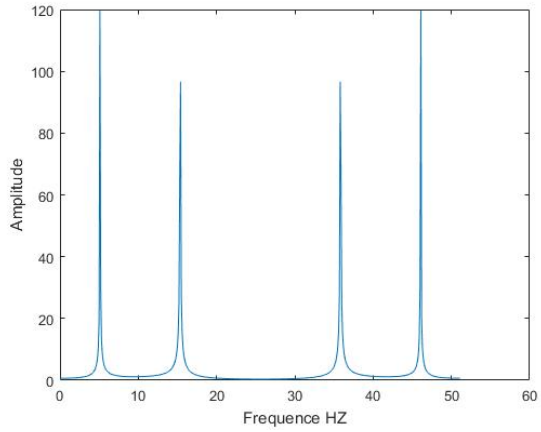


FIG. 5.3 : FFT à 512 points d'un signal sinusoïdale sur FPGA

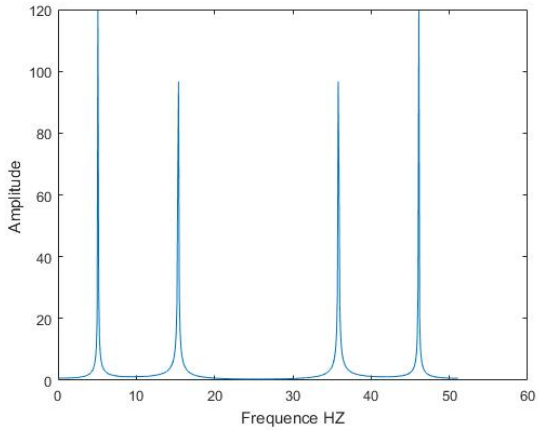


FIG. 5.4 : FFT à 512 points d'un signal sinusoïdale sur Matlab

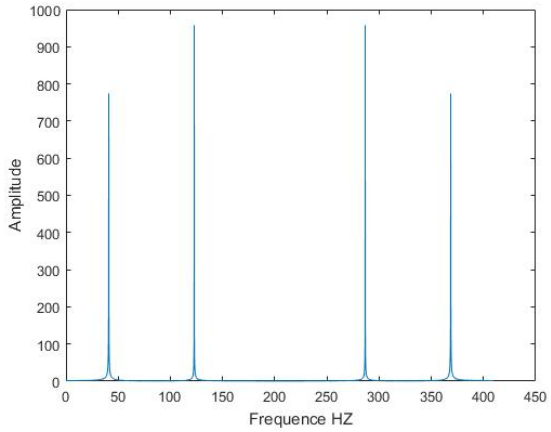


FIG. 5.5 : FFT à 4096 points d'un signal sinusoïdale sur FPGA

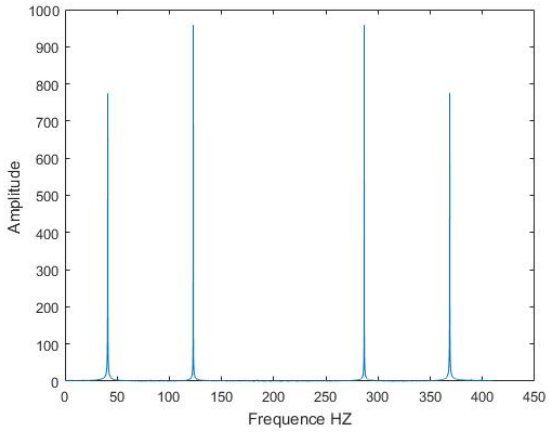


FIG. 5.6 : FFT à 4096 points d'un signal sinusoïdale sur Matlab

Nombre de points	64	512	4096
SQNR (dB)	65	51	40

TAB. 5.2 : SQNR du signal sinusoïdale

Signal complexe

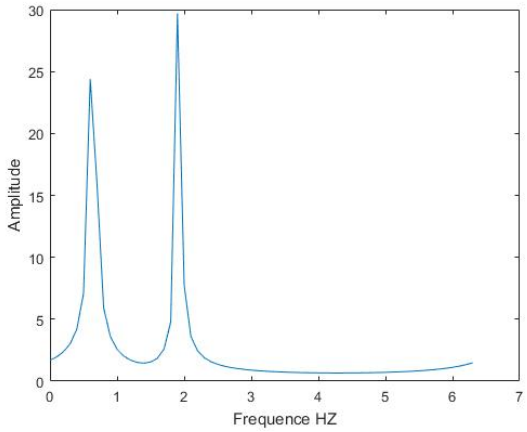


FIG. 5.7 : FFT a 64 points d'un signal complexe sur FPGA

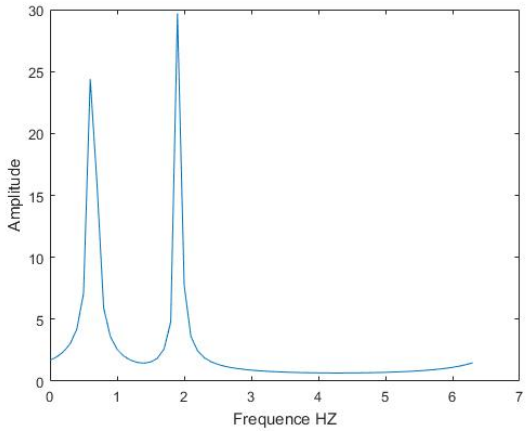


FIG. 5.8 : FFT a 64 points d'un signal complexe sur Matlab

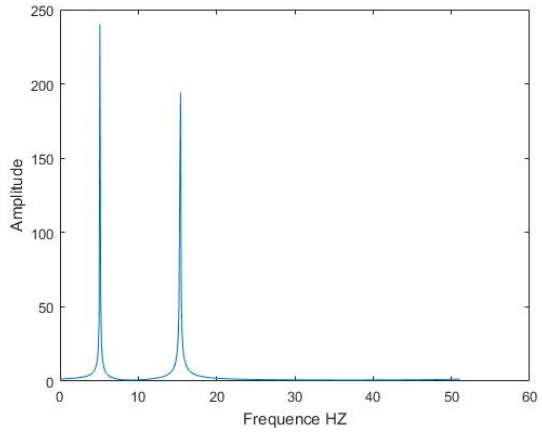


FIG. 5.9 : FFT a 512 points d'un signal complexe sur FPGA

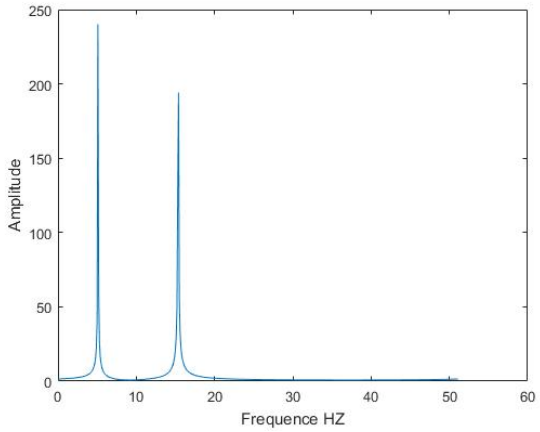


FIG. 5.10 : FFT a 512 points d'un signal complexe sur Matlab

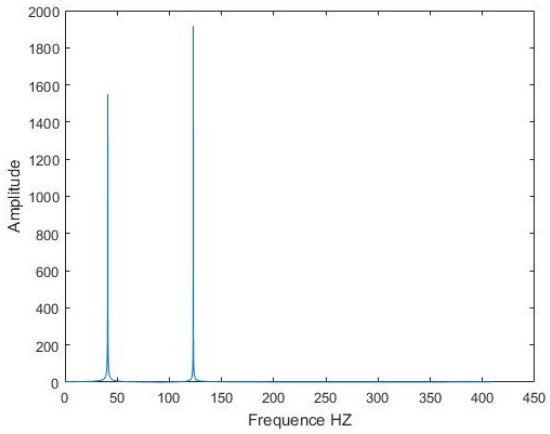


FIG. 5.11 : FFT a 4096 points d'un signal complexe sur FPGA

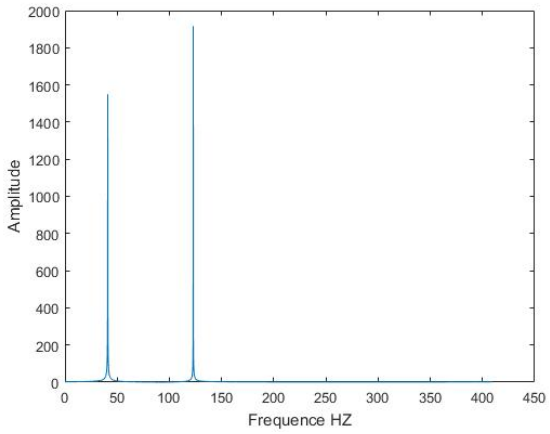


FIG. 5.12 : FFT a 4096 points d'un signal complexe sur Matlab

Nombre de points	64	512	4096
SQNR (dB)	66	53	42

TAB. 5.3 : SQNR du signal complexe

Signal chirp

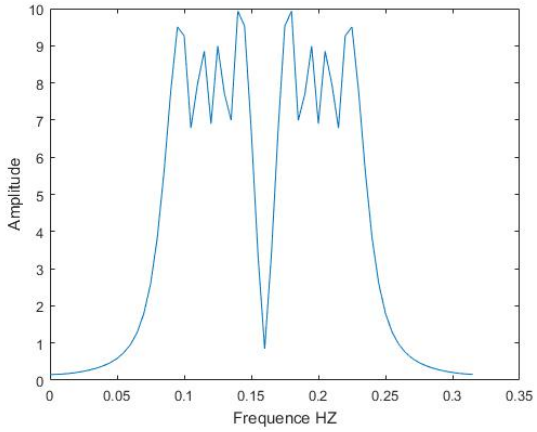


FIG. 5.13 : FFT a 64 points d'un signal chirp sur FPGA

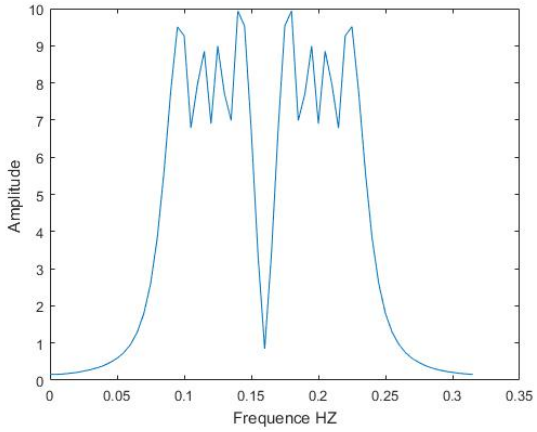


FIG. 5.14 : FFT a 64 points d'un signal chirp sur Matlab

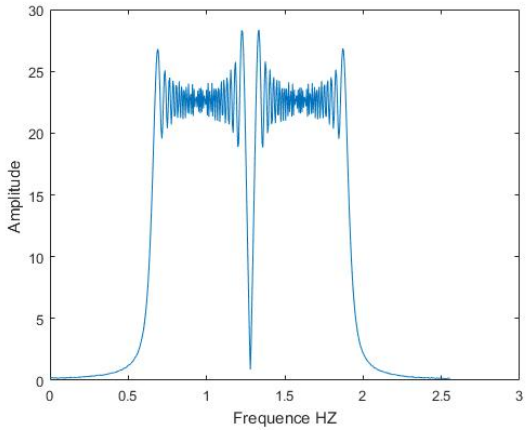


FIG. 5.15 : FFT a 512 points d'un signal chirp sur FPGA

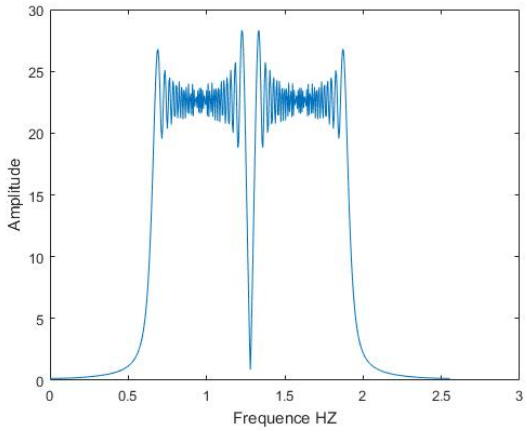


FIG. 5.16 : FFT a 512 points d'un signal chirp sur Matlab

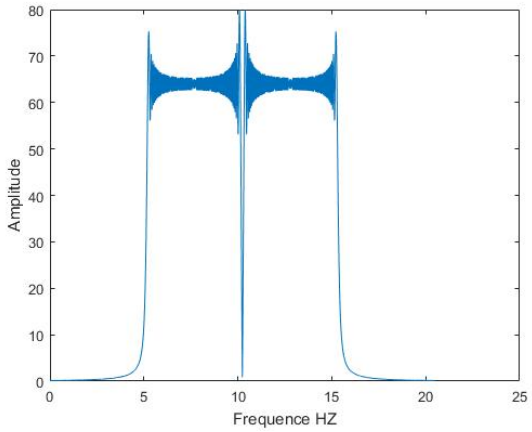


FIG. 5.17 : FFT a 4096 points d'un signal chirp sur FPGA

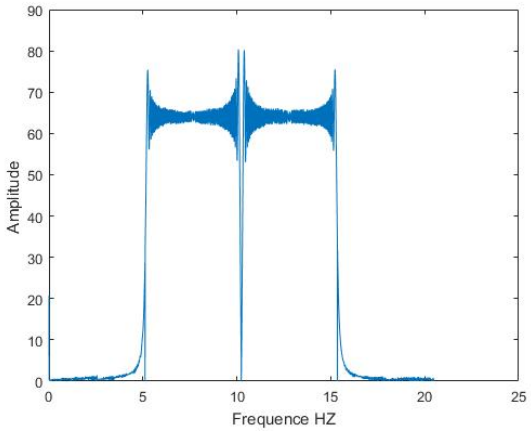


FIG. 5.18 : FFT a 4096 points d'un signal chirp sur Matlab

Nombre de points	64	512	4096
SQNR (dB)	62	53.2	37

TAB. 5.4 : SQNR du signal chirp

Commentaire

Les résultats obtenus démontrent que notre processeur parvient à obtenir le domaine fréquentiel des signaux mentionnés précédemment. Un tableau de mesures du rapport signal sur bruit de quantification a été présenté pour différents nombres de points et différents signaux 5.2 5.3 5.4.
La variation du rapport SQNR est attribuée à la réduction du nombre de bits dans la partie fractionnaire. Dans le cas de 512 points, 5 bits sont utilisés pour représenter la partie fractionnaire, tandis que pour 4096 points, seulement 2 bits sont utilisés. Pour améliorer le rapport signal sur bruit, il est nécessaire d'augmenter le nombre de bits à une valeur supérieure à 16 bits. Une approche complémentaire consisterait à examiner différentes méthodes d'arrondissement, comme suggéré dans l'article [33]

5.3 Évaluations de la consommation matérielle

Cette section est basée sur une analyse comparative des performances de notre architecture proposée en termes de consommation des ressources matérielles par rapport à l'IP core FFT de Xilinx, ainsi que sur quelques articles récents [31], [34], [21].
Les IP cores de Xilinx sont des modules de propriété intellectuelle préconçus et pré-validés, intégrés dans les logiciels de conception tels que Vivado et ISE. L'IP core FFT de Xilinx offre diverses options de configuration, telles que l'architecture, le nombre de points FFT, la taille de l'échantillon et la précision des données.

l'IP core FFT de [35] a une architecture pipeline streaming IO, qui met en série plusieurs unités de traitement Radix-2 [36], ainsi qu'une largeur de données de 16 bits.

Lors de la comparaison de l'utilisation des ressources matérielles entre différentes implémentations, il est important de prendre en compte trois métriques clés : les blocs LUTs, les blocs DSP ainsi que les BRAM. La description de ces indicateurs de performance a été détaillée dans le chapitre précédent.

Tab. 5.5 : Comparaison de performances entre l'architecture proposée et quelques travaux antérieurs

Points	Algorithmes	Architectures	Largeur du mot	Multiplications	LUTs	DSP	Block RAM
512	[35]	R2,PSI	16-bit PFX	AC,MC	15,469	0	1.5
	[31]	R2,FP,DDC	16-bit PFX	R2CORDIC	12,109	0	0
	[34]	R2,FP,MDC	16-bit PFX	R16CORDIC	11,302	0	0
	[21]	R2,FP,DDC	16-bit PFX	CORDIC	9981	0	0
	Proposé	$R2^3$,FP,MDC	16-bit PFX	CORDIC	8617	0	0
4096	[35]	R2,PSI	16-bit PFX	AC,MC	21,350	0	5.5
	[31]	R2,FP,DDC	16-bit PFX	R2CORDIC	26,543	0	0
	[34]	R2,FP,MDC	16-bit PFX	R16CORDIC	20,615	0	0
	[21]	R2,FP,DDC	16-bit PFX	CORDIC	19,757	0	0
	Proposé	$R2^3$,FP,MDC	16-bit PFX	CORDIC	11,313	0	0

R2: Radix-2 FFT, FP : Fully pipelined, PP : Parallel Pipelined, AC : Addition Complexe, MC : Multiplication, DDC : Double-path Delay Commutator, MDC : Multipath Delay Commutator,PFX : Point Fixe, PSI : Pipelined Streaming I/Os

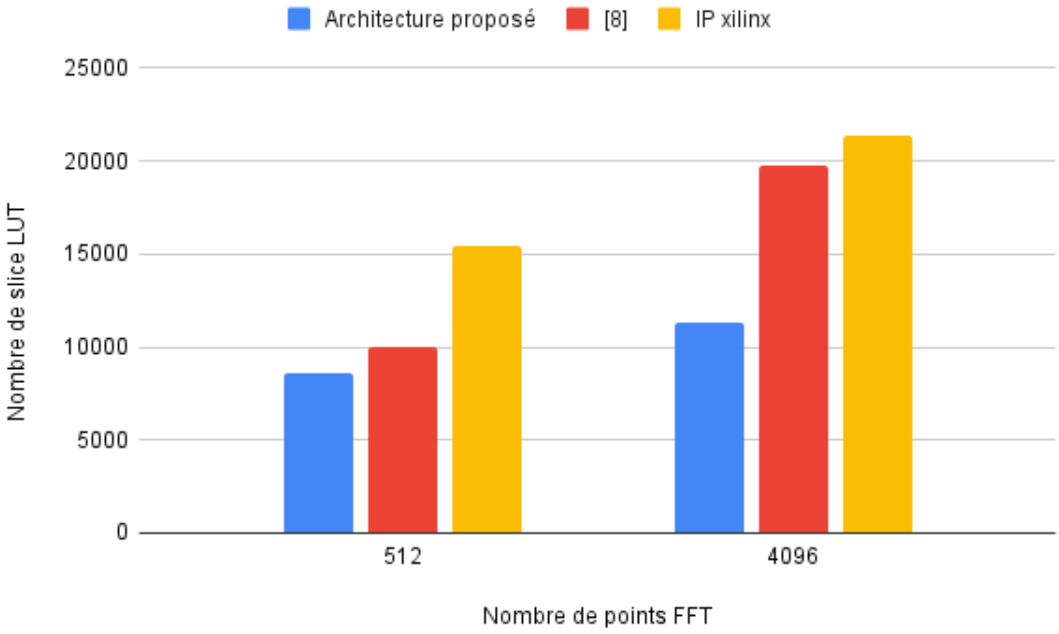


FIG. 5.19 : Consommation matérielles LUTS

En se référant au tableau 5.5 des consommations matérielles, nous avons observé que notre implémentation de l'algorithme FFT présente en moyenne une réduction de 47% de l'utilisation des blocs LUTs par rapport à l'IP core FFT de Xilinx.

Pour faciliter la comparaison avec les articles mentionnés dans le tableau, seule la dernière référence est prise en compte, car cette dernière a déjà démontré de meilleures performances par rapport aux articles précédents.

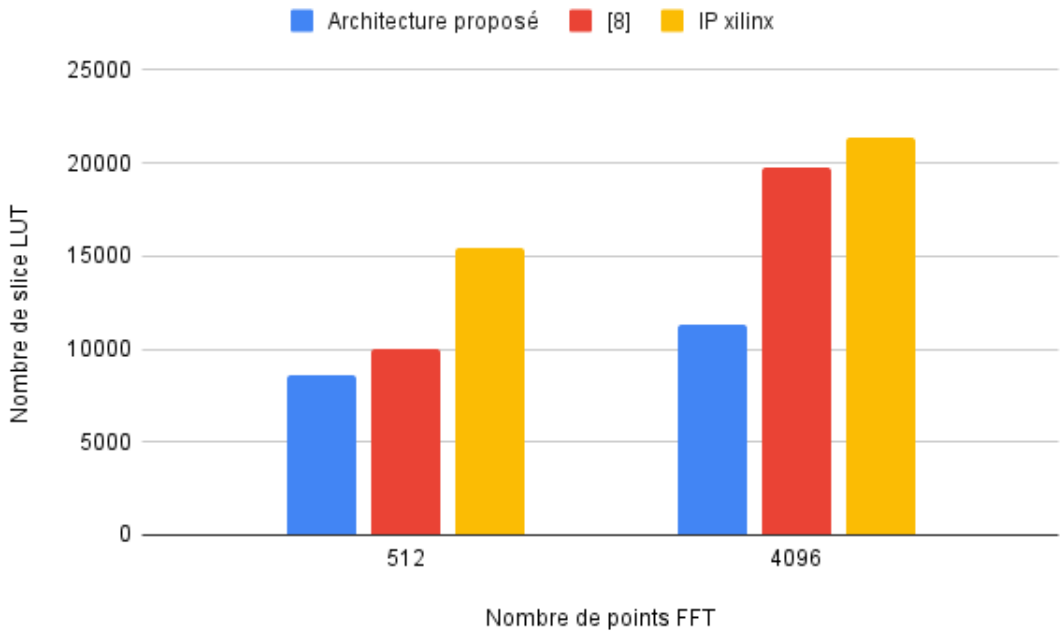


Fig. 5.20 : Consommation matérielles LUTS

À partir de la figure 5.20, on peut observer que, à mesure que le nombre de points de la FFT augmente, l'amélioration en pourcentage de l'utilisation des slices LUT est meilleure dans notre implémentation proposée par rapport à l'implémentation DDC radix-2. La réduction des ressources est possible grâce à l'algorithme radix 2^3, l'utilisation du CORDIC modifié pour la multiplication de nombres complexes ainsi qu'aux circuits de décalage et d'addition simples pour la multiplication par une les facteurs de phases constants.

5.4 Conclusion

L'implémentation de du processeur FFT proposée constitue une alternative prometteuse aux implémentations existantes, avec des améliorations significatives en termes d'utilisation des ressources. Ces résultats confirment l'efficacité de l'algorithme CORDIC proposé et de l'utilisation de circuits à base de décalage et d'addition pour la multiplication constante.

Conclusion

L'objectif de ce mémoire est de concevoir un processeur FFT sur une carte FPGA pour les applications SAR. Le but principal est d'accélérer le temps de calcul nécessaire pour effectuer la FFT, tout en réalisant une implémentation efficace en termes de consommation matérielle.

L'étude de différents algorithmes et architectures nous a permis de prendre connaissance des avantages et des inconvénients associés à chaque approche. Le choix de ces approches a été influencé par notre objectif de minimiser la consommation des ressources matérielles tout en gardant une bonne précision.

Lors de notre première tentative d'implémentation, nous avons constaté que les blocs DSP occupent 26% des blocs DSP disponibles sur la carte FPGA Artix-7. Cette situation a posé problème lorsque nous avons essayé d'implémenter un nombre de points plus grand pour le calcule de la FFT. De plus, la taille des mémoires ROM augmente de manière linéaire avec le nombre de points N.

Face à ces problème, une exploration de méthodes alternatives pour réaliser les multiplications s'est avérée nécessaire afin de contourner l'utilisation des blocs DSP et des blocs ROM. Dans l'architecture finale proposée, nous utilisons des circuits logiques basés sur des opérations d'addition et de décalage pour effectuer les multiplications par les facteurs constants, tandis que l'algorithme CORDIC est utilisé pour les autres facteurs.

Pour finir, une comparaison approfondie a été réalisée entre le processeur FFT conçu dans ce mémoire et les travaux antérieurs portant sur des architectures similaires. Cette comparaison a été réalisée en prenant la consommation de ressources matérielles en tant que critère.

Les résultats de cette comparaison ont clairement démontré que le processeur FFT proposé nécessite significativement moins de ressources matérielles par rapport aux travaux antérieurs. Cela se traduit par une utilisation plus efficace des ressources disponibles sur la carte FPGA, ce qui est d'une importance cruciale pour les applications à contraintes de ressources, telles que les applications SAR.

Bibliographie

2. CRUZ, Helena ; VÉSTIAS, Mário ; MONTEIRO, José ; NETO, Horácio ; DUARTE, Rui Policarpo. A review of synthetic-aperture radar image formation algorithms and implementations : a computational perspective. *Remote Sensing*. 2022, t. 14, n° 5, p. 1258.
3. LI, Yongrui ; CHEN, He ; XIE, Yizhuang. An FPGA-Based Four-Channel 128k-Point FFT Processor Suitable for Spaceborne SAR. *Electronics*. 2021, t. 10, n° 7. ISSN 2079-9292. Disp. à l'adr. DOI : 10.3390/electronics10070816.
4. WANG, Shiyu ; ZHANG, Shengbing ; HUANG, Xiaoping ; AN, Jianfeng ; CHANG, Libo. A Highly Efficient Heterogeneous Processor for SAR Imaging. *Sensors*. 2019, t. 19, n° 15. ISSN 1424-8220.
5. MARTI-PUIG, Pere. Two Families of Radix-2 FFT Algorithms With Ordered Input and Output Data. *IEEE Signal Processing Letters*. 2009, t. 16, n° 2, p. 65-68. Disp. à l'adr. DOI : 10.1109/LSP.2008.2003993.
6. NUSSBAUMER, Henri J. The Fast Fourier Transform. In : *Fast Fourier Transform and Convolution Algorithms*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1981, p. 80-111. ISBN 978-3-662-00551-4. Disp. à l'adr. DOI : 10.1007/978-3-662-00551-4_4.
7. SAIDI, Ali G. Decimation-in-time-frequency FFT algorithm. *Proceedings of ICASSP '94. IEEE International Conference on Acoustics, Speech and Signal Processing*. 1994, t. iii, III/453-III/456 vol.3.
8. SIMEK, Adam ; ŠIMEČEK, Ivan. Design of a modern fast Fourier transform and cache effective bit-reversal algorithm. *International Journal of Parallel, Emergent and Distributed Systems*. 2023, t. 38, n° 3, p. 229-248. Disp. à l'adr. DOI : 10.1080/17445760.2023.2179049.
9. SCHLEMON, Maron ; NAGHMOUCHI, Jamin. FFT Optimizations and Performance Assessment Targeted towards Satellite and Airborne Radar Processing. In : *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 2020, p. 313-320. Disp. à l'adr. DOI : 10.1109/SBAC-PAD49847.2020.00050.
10. CHAMI, Mouhcine ; DI MARTINO, Joseph ; PIERRON, Laurent et al. Real-time signal reconstruction from short-time Fourier transform magnitude spectra using FPGAs. In : *5th. International Conference on Information Systems and Economic Intelligence-SIIE 2012*. 2012.

11. KUMAR, G Ganesh ; SAHOO, Subhendu K ; MEHER, Pramod Kumar. 50 years of FFT algorithms and applications. *Circuits, Systems, and Signal Processing*. 2019, t. 38, p. 5665-5698.
12. BAISHYA, Dr Ranjit. International Journal for Research in Applied Science & Engineering Technology Vol 8 Issue IV. 2020.
13. BAHTAT, Mounir ; BELKOUCH, Said ; ELLEAUME, Philippe ; LE GALL, Philippe. Instruction scheduling heuristic for an efficient FFT in VLIW processors with balanced resource usage. *EURASIP Journal on Advances in Signal Processing*. 2016, t. 2016, p. 1-21.
14. MONCEF, Boutarene. *Systéme communication MIMO-OFDM sur FPGA*. 2018. Mém. de mast. Ecole Nationale Polytechnique.
15. TSAI, Pei-Yun ; CHEN, Chia-Wei ; HUANG, Meng-Yuan. Automatic IP generation of FFT/IFFT processors with word-length optimization for MIMO-OFDM systems. *EURASIP Journal on Advances in Signal Processing*. 2011, t. 2011, p. 1-15.
17. GARRIDO, Mario ; GRAJAL, J. ; SANCHEZ, M. A. ; GUSTAFSSON, Oscar. Pipelined Radix-2^k Feedforward FFT Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2013, t. 21, n° 1, p. 23-32. Disp. à l'adr. DOI : 10.1109/TVLSI.2011.2178275.
18. LIN, Hsin-Te. *A NEW VARIABLE-LENGTH FFT PROCESSOR FOR IEEE 802.16 STANDARD*. 2007. Mém. de mast. Tatung University.
20. QURESHI, Fahad ; ALAM, Syed Asad ; GUSTAFSSON, Oscar. 4k-point FFT algorithms based on optimized twiddle factor multiplication for FPGAs. In : *2010 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*. 2010, p. 225-228. Disp. à l'adr. DOI : 10.1109/PRIMEASIA.2010.5604921.
21. ZHAO, Yupu ; LV, Hong ; LI, Jun ; ZHU, Lulu. High performance and resource efficient FFT processor based on CORDIC algorithm. *EURASIP Journal on Advances in Signal Processing*. 2022, t. 2022, n° 1, p. 23. ISSN 1687-6180. Disp. à l'adr. DOI : 10.1186/s13634-022-00855-6.
22. DHANASEKAR, S. ; BRUNTHA, P. Malin ; AHMED, L. Jubair ; VALARMATHI, G. ; GOVINDARAJ, V. ; PRIYA, C. An Area Efficient FFT Processor using Modified Compressor adder based Vedic Multiplier. In : *2022 6th International Conference on Devices, Circuits and Systems (ICDCS)*. 2022, p. 62-66. Disp. à l'adr. DOI : 10.1109/ICDCS54290.2022.9780676.
23. MONICA, V ; GANGADHARAIAH, S L ; NARAYANAPPA, C K. FPGA Based Efficient Feed forward FFT Architecture. In : *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2018, p. 1692-1697. Disp. à l'adr. DOI : 10.1109/RTEICT42901.2018.9012113.
24. SANTHOSH, Lakshmi ; THOMAS, Anoop. Implementation of radix 2 and radix 22 FFT algorithms on Spartan6 FPGA. In : *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. 2013, p. 1-4. Disp. à l'adr. DOI : 10.1109/ICCCNT.2013.6726840.

25. SUN, Mingxi ; TIAN, Liyu ; DAI, Dongmin. Radix-8 FFT processor design based on FPGA. In : *2012 5th International Congress on Image and Signal Processing*. 2012, p. 1453-1457. Disp. à l'adr. DOI : 10.1109/CISP.2012.6469786.

26. CHANDU, Y ; MARADI, Megha ; MANJUNATH, Amogh ; AGARWAL, Priyanka. Optimized High Speed Radix-8 FFT Algorithm Implementation on FPGA. 2018, p. 430-435. Disp. à l'adr. DOI : 10.1109/ICOEI.2018.8553791.

27. YANG, Chen ; CHEN, He. A efficient design of a real-time FFT architecture based on FPGA. 2013.

29. LOHRKE, Heiko ; SCHOLZ, Philipp ; BOIT, Christian ; TAJIK, Shahin ; SEIFERT, Jean-Pierre. Automated detection of fault sensitive locations for reconfiguration attacks on programmable logic. In : *ISTFA 2016*. ASM International, 2016, p. 336-341.

31. NGUYEN, Ngoc Hung ; KHAN, Sheraz Ali ; KIM, Cheol-Hong ; KIM, Jong-Myon. A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms. *Microprocessors and Microsystems*. 2018, t. 60, p. 96-106. ISSN 0141-9331. Disp. à l'adr. DOI : https://doi.org/10.1016/j.micpro.2018.04.003.

32. GÁLVEZ, Mario Garrido. Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time. In : 2009.

33. TORRES, Cesar ; ZHOU, Bin ; PENG, Yingning ; HWANG, David. Pipeline FFT Architectures Optimized for FPGAs. *International Journal of Reconfigurable Computing*. 2009, t. 2009, p. 219140. ISSN 1687-7195. Disp. à l'adr. DOI : 10.1155/2009/219140.

34. CHANGELA, Ankur ; ZAVERI, Mazad ; VERMA, Deepak. FPGA implementation of high-performance, resource-efficient Radix-16 CORDIC rotator based FFT algorithm. *Integration*. 2020, t. 73, p. 89-100. ISSN 0167-9260. Disp. à l'adr. DOI : https://doi.org/10.1016/j.vlsi.2020.03.008.

35. XILINX, I. LogiCORE IP Fast Fourier Transform v8. 0, Product Specifications DS808. 2012.

Webographie

1. L3HARRIS GEOSPATIAL. *Sentinel-1 Intensity Analysis Tutorial* [https://www.l3harrisgeospatial.com/docs/Sentinel1IntensityAnalysisTutorial.html]. 2023.

16. DIGITAL SYSTEM DESIGN. *FPGA Implementation of 8-Point FFT* [https://digitalsystemdesign.in/fpga-implementation-of-8-point-fft/]. 2023. Accessed : June 21, 2023.

19. DIRLIK, S. *A comparison of FFT processor designs*. 2013. Aussi disponible à l'adresse : http://essay.utwente.nl/72179/.

28. *Memory Design* [Available at : https://digitalsystemdesign.in/memory-design/]. [s. d.]. Accessed : June 21, 2023.

30. *Xilinx* [https://www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/uwa1504034294196.html]. 2017. Accessed : June 21, 2023.

36. *Xilinx Documentation* [https://docs.xilinx.com/r/en-US/pg109-xfft/Pipelined-Streaming-I/O]. [s. d.]. Accessed : June 21, 2023.