ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing

PAR
MARIE-EVE GRANDMAISON

CONCEPTION D'UN MODULE RECONFIGURABLE DE FFT

MONTRÉAL, LE 28 JANVIER 2005

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE :

M. Jean Belzile, directeur du mémoire
Département de génie électrique à l'École de technologie supérieure

M.Claude Thibeault, codirecteur
Département de génie électrique à l'École de technologie supérieure

M. François Gagnon, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Christian Gargour, membre du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 25 JANVIER 2005
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

CONCEPTION D'UN MODULE DE RECONFIGURABLE FFT

Marie-Eve Grandmaison

SOMMAIRE

La transformée rapide de Fourier (FFT) sert à extraire l'information fréquentielle d'un signal temporel. Elle est souvent employée en traitement numérique du signal et certaines de ses applications requièrent les performances d'un processeur dédié. La plupart des modules de FFT/IFFT disponibles commercialement manquent de flexibilité et de paramétrisation permettant leur réutilisation dans diverses applications. Ce mémoire porte sur la conception d'un module synthétisable de transformée rapide de Fourier qui répond au besoin de reconfigurabilité tout en traitant efficacement un flot sériel de données complexes. La taille de la transformée ainsi que les largeurs binaires des interfaces et des bus à l'interne sont paramétrées avant l'étape de la synthèse. Le choix entre une transformée directe ou inverse et entre l'entrée en ordre naturel et en ordre des bits renversé sont aussi disponibles.

Le module de FFT conçu se base sur une architecture matérielle de type pipelinée radix-2^2 qui a l'avantage de réutiliser les sous-modules. L'accroissement des besoins en ressources est logarithmique avec l'augmentation de taille de la transformée. Comparativement, une croissance par palier est obtenue avec une architecture de type réutilisation du papillon. Vu l'utilisation de délais en rétroaction et l'absence de parallélisation des données, l'architecture choisie demande peu de ressources mémoire et a une faible latence. La précision des résultats issus du calcul de la FFT dépend en grande partie des paramètres de configuration et des étapes d'arrondis. Des simulations ont permis d'identifier les effets de la quantification sur les performances. En comparant avec des cores commerciaux, notamment les *logicores* de Xilinx, on trouve que pour une configuration semblable, le module de FFT conçu utilise moins de logique et de ressources dédiées.

On présente aussi l'utilisation du module reconfigurable de FFT pour la réalisation d'un filtre dans le domaine fréquentiel. Ce type de filtre nécessite deux modules de FFT et demande moins de ressources qu'un FIR pour une vitesse de traitement similaire. Une étude de la précision des résultats de filtrage face au bruit de quantification est aussi effectuée.

RECONFIGURABLE FFT CORE DESIGN

Marie-Eve Grandmaison

ABSTRACT

Fast Fourier Transforms (FFT) are used to extract frequency information from a sequence of samples in the time domain. Hardware or firmware implementations of the FFT/IFFT are found in many digital signal processing applications. However, most commercially available cores lack in reconfigurability thus impeding reuse and increasing design time. This master's thesis presents the design of a scalable and parameterizable at synthesis FFT/IFFT core. The transform size, binary word length of inputs, outputs, and twiddle factors are all configurable. Parameters to select between FFT/IFFT and from ordered or bit-reversed inputs/outputs are also available.

The design is based on a pipelined radix-2^2 architecture where data parallelization is not required. The design allows real-time processing of the serial input signal, while adding minimal processing latency and optimizing memory usage. The building blocks forming the FFT are reused, therfore the resources increase logarithmically with transform size. By oppositon, a parallel FFT architecture would have increased in a step fashion. The results accuracy rely on configuration settings and on round-off methods. Simulations were performed to measure quantization effects and to identify the influence of each parameter. When comparing the designed FFT with commercial ones, like Xilinx Logicore's, for similar configuration our core uses less logic and embedded resources.

The design of a parameterize frequency-domain filter using the FFT is also presented. This kind of filter requires two FFT/IFFT cores, a complex multiplier and buffers. As the time response for a given filter increases, the frequency processing becomes attractive from a hardware resources point of view. For practical cases, the crossover point between frequency-domain and time-domain (FIR) implementations may be as low as 8 complex coefficients. The accuracy performance of both kind of filters is also studied.

AVANT-PROPOS ET REMERCIEMENTS

Ce mémoire a été réalisé au sein du LACIME (Laboratoire de communication et d'intégration de la microélectronique) de l'École de technologie supérieure, en collaboration avec la chaire Ultra-Electronics TCS en communication sans-fils.

Je remercie les Professeurs Jean Belzile et Claude Thibeault pour leur support et leurs conseils qui m'ont permis de mener à bien le projet.

Merci aussi à mes collègues du LACIME avec qui j'ai eu des discussions qui m'ont permis de trouver des solutions ou inspiré de nouvelles idées.

Je tiens également à remercier Sébastien pour sa patience et la pertinence de ses commentaires.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET SIGLES

Abréviation	Définition
W^{nk}_N	Facteur de phase
α	Facteur de mise en forme ou d'excès de bande passante
ASCI	Application Specific Integrated Circuit (Circuit intégrés à application spécifique)
b	Largeur d'un mot binaire
BF	Butterfly (Unité d'architecture papillon)
BMULT	Block Multiplier (Multiplicateur dédié)
BRAM	Block RAM (Bloc de mémoire RAM dédié)
CORDIC	COordinate Rotation Digital Computer (Calculateur numérique de rotation de coordonnée)
CS(f)	Cosinus surélevé en fréquence
cs(t)	Cosinus surélevé dans le temps
CW	Coefficient Wordlength (Largeur binaire des coefficients)
DC	Delay Commutator (Commutateur à délais)
DF	Delay Feedback (Délai en rétroaction)
DFT	Discrete Fourier Transform (Transformée discrète de Fourier)
DIF	Decimation In Frequency (décimation en fréquence)
DIT	Decimation In Time (décimation dans le temps)
dmath	Signal provenant du modèle mathématique quantifié
dref	Signal de référence dans le temps
DSP	Digital Signal Processor (Processeur numérique du signal)
dvhdl	Signal provenant des simulations VHDL
f	Variable continue en fréquence
FDF	Filtre dans le domaine fréquentiel
FFT	Fast Fourier Transform (Transformée rapide de Fourier)
FIFO	First In, First Out (Premier entré, premier sorti)
f_{in}	Fréquence d,arrivée des échantillons

FIR	Finite Impulse Response (Réponse impultionnelle finie)
FPGA	Field Programmable Gate Array (Réseau de portes programmable)
HDL	Hardware Description Language (Langage de description du matériel)
IDFT	Inverse Discrete Fourier Transform (Transformée discrète de Fourier inverse)
IFFT	Inverse Fast Fourier Transform (Transformée rapide de Fourier inverse)
IFW	Input Frequency Wordlength (Largeur binaire d'entrée en fréquence)
IW	Input Wordlength (Largeur binaire d'entrée)
k	Variable discrète en fréquence
L	Taille d'un bloc de données
LSB	Lest Signifiant Bit (Bit le moins significatif)
LMS	Least Mean Square (Plus petite moyenne carré)
LUT	Look-Up Table (Table de conversion)
M	Nombre de coefficient du filtre
MSB	Most Signifiant Bit (Bit le plus significatif)
N	Taille de la transformée
n	Variable discrète dans le temps
OFDM	Orthogonal Frequency Division Multiplexing (Multiplexage par répartition orthogonale de la fréquence)
OFW	Output Frequency Wordlength (Largeur binaire de sortie en fréquence)
OW	Output Wordlength (Largeur binaire de sortie en fréquence)
r	Base
RAM	Random Access Memory (Mémoire vive)
ROM	Read Only Memory (Mémoire morte)
s	Puissance de deux indiquant la mise à l'échelle, $S=2^s$
S	Facteur de mise à l'échelle
SQNR	Signal to Quantization Noise Ratio (Rapport signal à bruit de quantification)
$SQNR_{R2^2PC}$	SQNR du module conçu
$SQNR_{ref}$	SQNR de référence
t	Variable continue dans le temps

v	Puissance de deux indiquant la taille, $N=2^v$
VHDL	VHSIC HDL
VHSIC	Very High Speed Integrated Circuits (Circuit intégré à très grande vitesse)
VLSI	Very Large Scale Integration (Intégration à très grande échelle)
$X(f)$	Signal continu en fréquence
$X(k)$	Signal discret en fréquence
$x(n)$	Signal discret dans le temps
$x(t)$	Signal continu dans le temps

INTRODUCTION

L'algorithme de la transformée rapide de Fourier (FFT) fait parti du top dix des algorithmes du 20^{e} siècle qui ont eu le plus grand impact sur les sciences pratiques et le génie (Dongarra & Sullivan, 2000). Cet algorithme permet de décomposer un signal en ses composantes périodiques avec une complexité de calcul réduite. En effet, l'approche classique de la transformée discrète de Fourier demande une complexité de $O(N^2)$ tandis que la FFT requiert seulement $O(N\log_2 N)$. Cet algorithme est omniprésent dans les applications de traitement numérique des signaux. On retrouve son utilisation dans divers champs de connaissances, notamment dans les domaines de l'audio, du radar, des télécommunications . . .

Pour réaliser une FFT, on peut utiliser une application logicielle ou une unité de traitement dédiée. L'approche logicielle est flexible et configurable, mais elle offre des performances limitées. En général, l'approche dédiée ou matérielle est plus rapide, cependant elle est moins polyvalente. Ce travail porte sur la conception d'une architecture matérielle reconfigurable de transformée rapide de Fourier. Une attention particulière sera faite pour minimiser les ressources tout en maintenant une fréquence d'opération élevée et une précision adéquate des résultats. Le module est paramétré de manière à le rendre réutilisable dans de nombreuses applications. La réalisation d'un filtre numérique utilisant le module de FFT/IFFT conçu sera aussi présentée. La programmation et l'optimisation de FFT logicielle n'entrent pas dans le cadre de ce travail. La structure de ce document suit la méthodologie de conception utilisée pour réaliser le module reconfigurable de FFT/IFFT.

Le premier chapitre présente l'historique de l'algorithme de la transformée rapide de Fourier. Les simplifications réduisant la complexité arithmétique sont résumées, puis on

expose la représentation graphique des algorithmes les plus répandus et on définit certains termes utiles à la compréhension de la suite du travail.

Le deuxième chapitre porte sur la définition des besoins. Ceux-ci sont énoncés sous la forme d'une spécification. Ensuite, on présente une revue de la littérature qui permet d'étudier les différentes architectures matérielles de FFT existantes. Les deux principales approches, parallèle et pipelinée, sont analysées et comparées en fonction des besoins. Puis, le choix de l'architecture la plus appropriée est fait.

Le partitionnement du design de manière hiérarchique est présenté au troisième chapitre. Le contrôle local plutôt que global pour minimiser les interconnexions et faciliter la réutilisation des blocs a été privilégié. L'approche employée pour paramétrer le design est expliquée et illustrée par un exemple de configuration. On précise le fonctionnement et les particularités de chaque bloc, puis on définit toutes ses interfaces et paramètres de configuration.

Le chapitre 4 porte sur la caractérisation du module sous deux aspects : la précision des résultats et les ressources requises lors de l'implémentation dans une puce programmable. L'environnement de test et les métriques employées sont d'abord expliqués. Ensuite, on présente et analyse les performances obtenues en variant un ou plusieurs paramètres de configuration. Ainsi, cela permet de comprendre ce qui influence le bruit de quantification et estimer les ressources nécessaires pour une application précise.

Le module configurable de FFT/IFFT est comparé à des *cores* commerciaux de FFT au chapitre 5. D'une part, on présente les spécifications de *cores* synthétisables de différentes provenances pour lesquels la fiche technique est l'unique source d'information. D'autre part, la comparaison est approfondie avec les *cores* de Xilinx

auxquels on a accès. Elle se fera au niveau de l'usage de ressources du FPGA et de la précision des résultats obtenue pour une paramétrisation similaire.

Le chapitre 6 présente un exemple d'utilisation du module de FFT/IFFT pour la réalisation d'un filtre dans le domaine fréquentiel configurable. D'abord, l'étude de la complexité de ce type d'architecture par rapport à un filtre FIR traditionnel est réalisée. Puis, la méthodologie de conception utilisée pour réaliser le module de FFT est reprise pour le filtre dans le domaine fréquentiel. Les blocs formant le filtre sont détaillés, l'environnement de test est présenté et les résultats sont analysés.

Une conclusion permet ensuite de faire la synthèse du travail. On y présente aussi des possibilités d'utilisation du module reconfigurable de FFT pour des applications de télécommunication sans-fils à haut débit.

CHAPITRE 1

ALGORITHME DE LA FFT

1.1 Historique

1.1.1 Transformée de Fourier

La transformée de Fourier a été élaborée par le célèbre physicien et mathématicien français Jean Baptiste Joseph Fourier (1768-1830). Parmi les outils de calcul nommés en son honneur on retrouve les séries de Fourier, les intégrales de Fourier et la transformée de Fourier. La théorie développée par Fourier se retrouve souvent sous l'appellation analyse de Fourier dans les ouvrages de mathématique (Kreyszig, 1999). La théorie et les méthodes qu'il a développées sont très utiles en ingénierie, notamment dans le domaine du traitement du signal et pour la résolution d'équations différentielles.

La biographie de Fourier se retrouve dans plusieurs ouvrages (Bayart, 2004; Boyer & Merzbach, 1991; Mankiewicz, 2001; Maor, 2002) et elle est résumée ici. Fourier devint orphelin à l'age de neuf ans et fut ensuite placé à l'école militaire afin de poursuivre son éducation. En 1794, il reçut son diplôme de l'École normale supérieure, la première promotion. Ses professeurs furent Lagrange (1736-1813), de Laplace (1749-1827) et de Monge (1746-1818). Lors de la révolution française, Fourier fut emprisonné et sauvé de justesse de la guillotine. Puis en 1798, il partit avec les expéditions de Napoléon en Égypte et devint le gouverneur du sud de l'Égypte. Après la défaite de Napoléon en 1801, il retourna en France pour y faire une carrière administrative et universitaire. Le problème de la distribution spatiale de la chaleur dans un corps solide et de ses variations dans le temps l'intéressa énormément. De 1802 à 1807, il trouva les équations, puis une méthode pour résoudre le problème, ce qui est maintenant connu comme étant l'analyse de Fourier. Ce n'est qu'en 1822 à Paris qu'il publia *Théorie analytique de la chaleur.*

Avant de publier, il dut longuement défendre et prouver sa théorie devant ses contemporains notamment Lagrange, Laplace et Poisson (1781-1840). Le principe à la base de sa théorie dit qu'une fonction mathématique peut être représenté en une somme infinie de fonctions en sinus et en cosinus. Il devient ensuite secrétaire de la section mathématique l'académie des sciences.

La transformée de Fourier permet le passage du domaine du temps au domaine de la fréquence (Proakis & Manolakis, 1996). L'équation 1.1 est celle d'une transformée de Fourier directe avec $x(t)$ la fonction dans le temps et $X(f)$ la fonction en fréquence. La transformée inverse est aussi possible et son équation est donnée en 1.2.

$$x(t)=\int_{-\infty}^{\infty} X(f)\, e^{j2\pi ft}\, dt \tag{1.1}$$

$$X(f)=\int_{-\infty}^{\infty} x(t)\, e^{-j2\pi ft}\, dt \tag{1.2}$$

1.1.2 Transformée discrète de Fourier

Les équations 1.1 et 1.2 correspondent à des intégrales analytiques de moins l'infini à l'infini. Dans ce travail, vu l'utilisation de l'électronique numérique pour réaliser une transformée rapide Fourier, on doit se limiter à des représentations discrètes. Nous utiliserons donc la version discrète qui se nomme la transformée discrète de Fourier (DFT) (Oppenheim, Schafer, & Buck, 1999; Proakis & Manolakis, 1996; Rabiner & Gold, 1975). Malgré son nom, cet outil n'a pas été développé par Fourier. Il a plutôt été développé par le célèbre mathématicien Carl Friedrich Gauss (1777-1855) pour le calcul d'orbites (Heideman, Johnson, & Burrus, 1984). Gauss s'est basé sur les travaux de Lagrange qui avait développé une DFT en sinus, et d'Euler (1707-1783) sur les séries trigonométriques. De nos jours, on retrouve la DFT dans de nombreuses applications de

traitement numérique des signaux. L'équation de la DFT (Proakis & Manolakis, 1996) est

$$X(k)=\sum_{n=0}^{N-1} x(n)W_N^{kn} \qquad 0 \leqslant k \leqslant N-1 \tag{1.3}$$

et celle de la transformée inverse discrète de Fourier (IDFT) est

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-kn} \qquad 0 \leqslant n \leqslant N-1. \tag{1.4}$$

avec $x(n)$ le vecteur temporel, $X(k)$ le vecteur fréquentiel et N la taille de ces vecteurs. Le coefficient du produit W_N est nommé facteur de phase et correspond à une coordonnée sur le cercle unitaire complexe:

$$W_N = \mathrm{e}^{-j2\pi/N} \tag{1.5}$$

Le calcul direct d'une DFT correspond à la sommation d'un produit pour chaque point. Ceci équivaut à une quantité de calculs considérable: N additions et N multiplications complexes pour obtenir chaque résultat, donc N^2 opérations pour une transformée complète. Avec l'utilisation de processeurs et de puces dédiées pour calculer numériquement la DFT, optimiser ce calcul devient très avantageux.

1.1.3 Transformée rapide de Fourier

Le calcul direct de la DFT et de l'IDFT n'est pas efficace, puisqu'on ne tire pas profit des propriétés de symétrie (éq. 1.6) et de périodicité (éq. 1.7) des facteurs de phase sur le cercle unitaire complexe.

$$\text{Symétrie}: \quad W_N^{k+N/2} = -W_N^k \tag{1.6}$$

$$\text{Périodicité: } W_N^{k+N} = W_N^k \tag{1.7}$$

Les algorithmes de transformée rapide de Fourier (FFT, *Fast Fourier Transform*) permettent de faire une DFT de manière efficace et de réduire la complexité des calculs à l'ordre $C N \log_2 N$, où C est un coefficient inférieur ou égal à un et qui dépend de l'algorithme choisi. En effet, il existe de nombreux algorithmes de FFT qui découlent tous de l'approche diviser pour conquérir (*divide-and-conquer approch*).

Dans l'histoire récente, la découverte de l'algorithme de FFT est attribuée à Cooley et Tukey (1965). L'historique de cette redécouverte est racontée dans (J.W. Cooley, 1992). On nous fait part que John Tukey et un de ses collègues ont eu l'idée de la FFT en 1963 et qu'ils ont mandaté James W. Cooley de développer le concept et d'écrire un programme. Cooley ne se doutait pas de l'impact que ces simplifications auraient dans le monde de l'informatique et du traitement de signal, il n'a donc par priorisé ce travail. Suite à des pressions de son entourage, il a finalement publié un article en 1965. Cette publication a favorisé l'essor de nombreux autres articles expliquant des applications, améliorations et variantes de l'algorithme. En revanche, quelqu'un travaillant dans le domaine de l'océanographie a contacté Cooley pour lui signaler qu'il avait déjà programmé l'équivalent d'un algorithme de FFT radix-2 et qu'il avait pris la méthode dans l'article (Danielson & Lanczos, 1942). Cet article, resté dans l'oubli, exposait la FFT comme un moyen de faire la DFT à la main ou même comme une méthode pour valider cette dernière. Si l'on remonte plus loin dans l'histoire, en 1805 en Gauss aurait développé une manière efficace de calculer la DFT semblable à l'algorithme de Cooley et Tukey, mais il ne l'aurait pas publié (J.W. Cooley, 1992; Heideman, Johnson, & Burrus, 1984). Ainsi, la FFT aurait été développée avant la transformée de Fourier achevée en 1807 mais publiée en 1822. La publication de l'algorithme de Gauss eut lieu de manière posthume en 1866, en latin néo-classique ce qui a nuit à sa diffusion . . .

1.2 Survol du développement mathématique et représentation graphique

1.2.1 Approche diviser pour conquérir

L'algorithme de la FFT et ses variantes sont détaillées dans de nombreux ouvrages, notamment (Oppenheim, Schafer, & Buck, 1999; Proakis & Manolakis, 1996; Rabiner & Gold, 1975). Ce travail présente un bref développement mathématique pour illustrer les simplifications possibles de l'approche diviser pour conquérir. Pour plus de détails, il est conseillé de se référer à un de ces ouvrages. Le principe à la base de l'algorithme consiste à faire un changement de variable de l'indice n sur deux dimensions entières. Par exemple avec M pour les colonnes et L pour les rangées, on a

$$N = ML \tag{1.8}$$

$$n = Ml + m \qquad 0 \leqslant l \leqslant L-1 \quad et \quad 0 \leqslant m \leqslant M-1 \tag{1.9}$$

$$k = Mp + q \qquad 0 \leqslant p \leqslant L-1 \quad et \quad 0 \leqslant q \leqslant M-1 \tag{1.10}$$

Avec ce changement de variable, les points de la séquence à transformer se retrouvent dans un arrangement matriciel rangé/colonne et l'équation de la DFT devient

$$X(p,q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)} \quad . \tag{1.11}$$

Le terme $W_N^{(Mp+q)(mL+l)}$ peut être simplifié de la manière suivante :

$$W_N^{MLmp} = 1, \quad W_N^{mqL} = W_M^{mq} \quad \text{et} \quad W_N^{plM} = W_L^{pl} \quad . \tag{1.12}$$

De là on obtient l'équation

$$X(p,q)=\sum_{l=0}^{L-1}\left\{W_N^{lq}\left[\sum_{m=0}^{M-1}x(l,m)W_M^{mq}\right]\right\}W_L^{lp}\ . \qquad (1.13)$$

Pour résoudre l'équation 1.13, on procède d'abord à des DFT de M points correspondant à la sommation à l'intérieur entre crochets. Ensuite, la matrice résultante subit une multiplication point à point par les facteurs de phase W_N^{lq} et, enfin, des DFT sur L points sont effectuées en suivant l'autre dimension. En effectuant ces étapes, la complexité passe de l'ordre de N^2 à l'ordre $N(M+L)$. On peut aussi récursivement faire d'autres changements de variable. En prenant un nombre premier pour L, on peut ensuite factoriser M jusqu'à obtenir seulement des DFT ayant des tailles de nombre premier.

1.2.2 Algorithme radix-2

Lorsque tous les facteurs de N sont les mêmes, la FFT a la particularité de pouvoir être calculée de manière régulière et d'avoir une représentation graphique facile à interpréter. En prenant la base deux, $N=2^v$, on obtient les algorithmes qu'on retrouve dans la littérature sous le nom radix-2 . Suite au changement de variable successifs, on obtient l'opération mathématique « papillon » qui doit être effectuée sur chaque groupe de deux points. Cette opération se nomme papillon en raison de la forme de sa représentation graphique. Un papillon radix-2 contient une addition, une soustraction et une multiplication par un facteur de phase. Toutes ces opération sont effectuées sur des nombres complexes. Deux types de papillons existent puisque l'approche diviser pour conquérir peut être utilisée avec un changement de variable selon les rangées ou les colonnes. Le premier type nommé décimation en fréquence (DIF) est défini par la multiplication suivant la somme croisée comme l'illustre la figure 1. Le deuxième type nommé décimation dans le temps (DIT) est défini par la multiplication précédant les sommes, illustré à la figure 2. Les figures 3 et 4 illustrent des transformées de N=16

respectivement DIF et DIT. Sur ces figures, les cercles correspondent à la partie additive du papillon, une addition et une soustraction pour le cas présenté ici. Cette représentation graphique sera aussi utilisée dans les chapitres suivants.

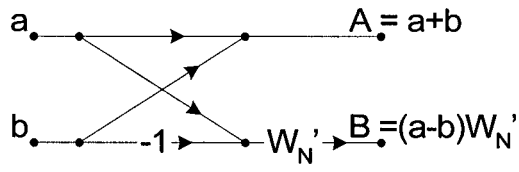


Figure 1 Papillon radix-2 DIF

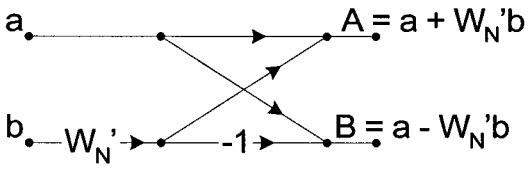


Figure 2 Papillon radix-2 DIT

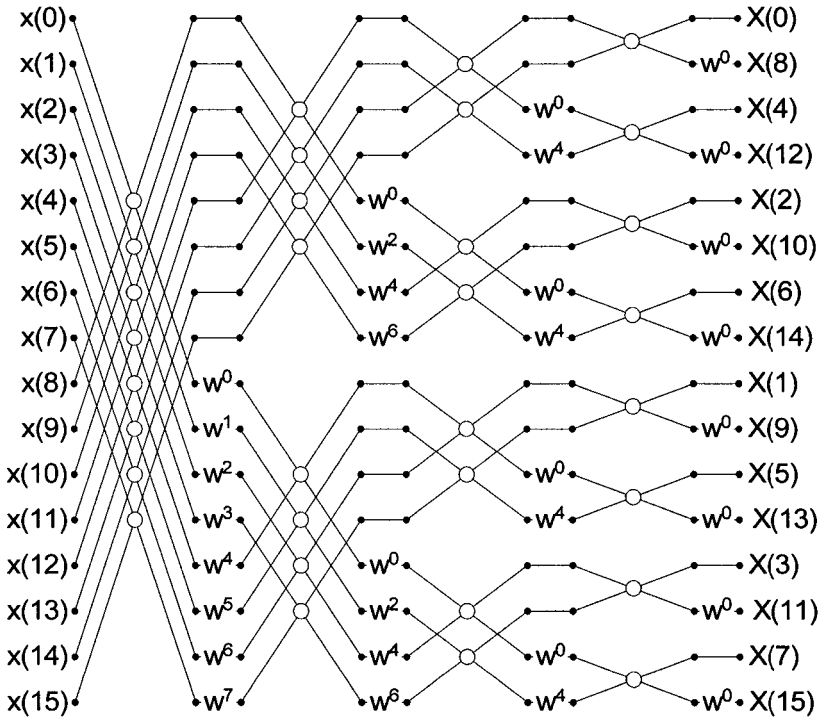


Figure 3 FFT radix-2 DIF sur 16 points

Une FFT radix-2 sur N points utilise $N/2$ rangées et $\log_2 N$ colonnes de papillons, ce qui fait au total $N\log_2 N$ additions complexes et $\frac{N}{2}\log_2 N$ multiplications complexes. Ce qui se compare avantageusement aux N^2 additions et N^2 multiplications complexes de la DFT. Sur la figure 3, on remarque que la séquence de points en entrée est en ordre naturel, mais que la sortie n'est pas ordonnée. L'inverse se produit pour l'algorithme DIT (fig. 4). Dans la littérature, cet ordre non naturel est nommé en anglais *bit reverse order.* Dans ce travail, on y fera référence par le nom ordre binaire renversé. On peut prévoir cet ordre en prenant l'indice du point en représentation binaire sur v bits, MSB à gauche, et en lisant cette représentation binaire dans l'ordre inverse, de droite à gauche. Par exemple, prenons le point $X(10)$ qui est sorti en $5^{ème}$ position à la figure 3. La représentation binaire de 10 sur quatre bits correspond à «1010». Lu à l'envers, on trouve

«0101» qui est effectivement 5 en binaire. Certains algorithmes hybrides DIF/DIT permettent d'obtenir l'ordonnancement naturel en entrée et en sortie.

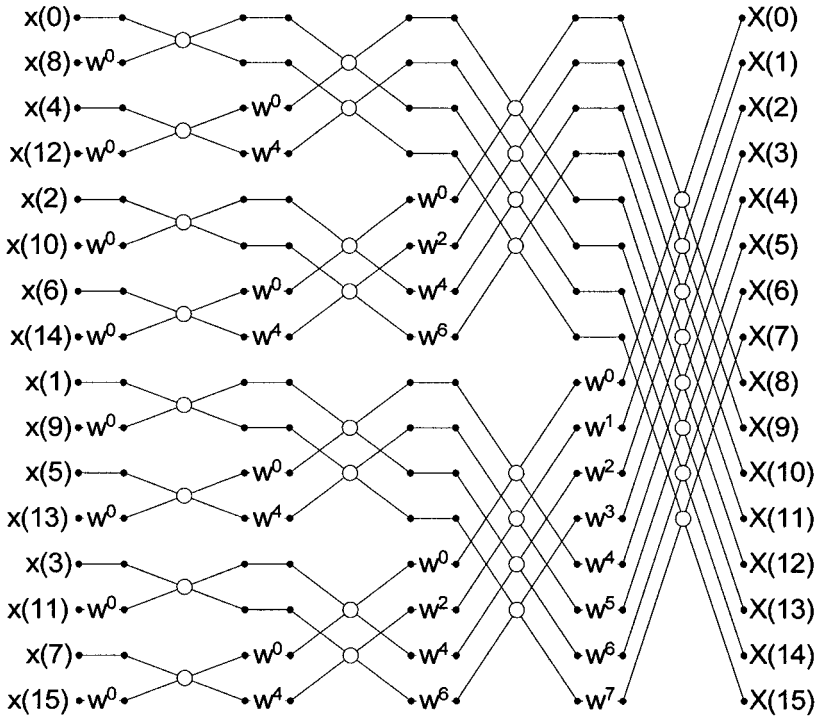


Figure 4 FFT radix-2 DIT sur 16 points

1.2.3 Algorithme radix-4

On peut utiliser d'autres bases que deux ou une combinaison de plusieurs bases, comme pour l'algorithme *split-radix* (Proakis & Manolakis, 1996), pour réduire davantage la complexité multiplicative de la DFT. Dans la suite de ce travail nous utiliseront l'algorithme de FFT radix-4, où quatre factorise *N*. En effet, on peut tirer profit des multiplications triviales par 1, -1, j et -j qui n'ont pas à être réalisées par une vraie multiplication. Le papillon radix-4 contient huit additions ou soustractions et trois multiplications complexes. Le papillon est illustré à la figure 5 et la représentation graphique de l'algorithme radix-4 à la figure 6 où les cercles représentent la partie additive dans laquelle quatre points se rencontrent.

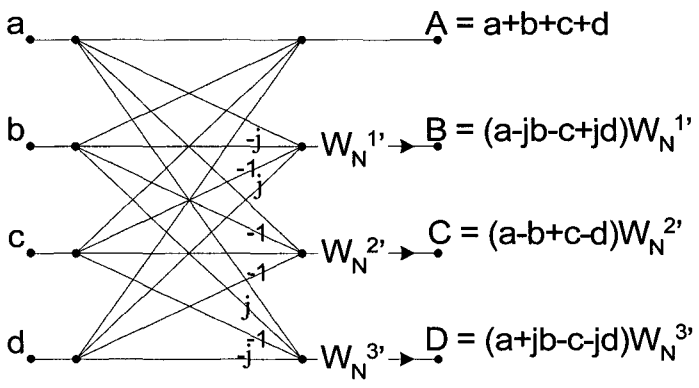


Figure 5 Papillon radix-4

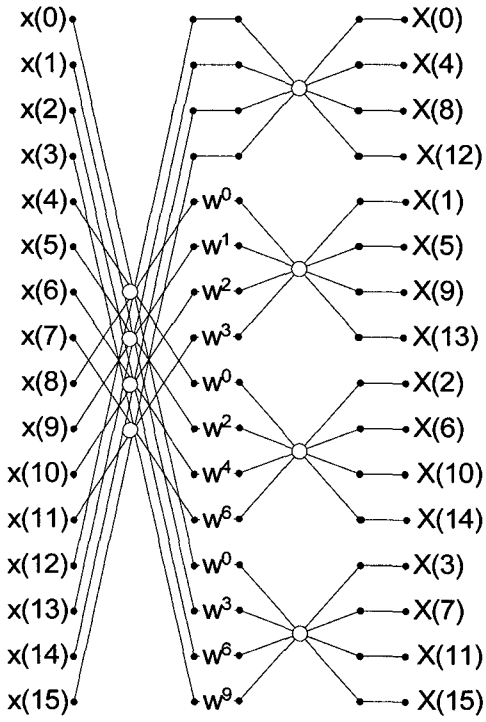


Figure 6 FFT radix-4 DIF sur 16 points

Une FFT requiert $N/4$ colonnes et $\log_4 N$ rangées de papillons. La complexité de cet algorithme est de $N\log_2 N$ additions complexes et $\frac{3N}{8}\log_2 N$ multiplications complexes. La version DIT de cet algorithme existe aussi, mais elle n'est pas illustrée ici. Au lieu de l'ordre binaire renversé en sortie, on trouve l'ordre renversé des groupes de deux bits puisque la base est quatre (2^2). Dans la littérature, cet ordre est nommé *digit reverse order*.

1.3 Conclusion

En résumé, l'algorithme de la FFT a un passé chargé et certainement un futur bien rempli, vu les multiples utilisation possibles de cet outils de traitement du signal. On retrouve plusieurs variantes des algorithmes de FFT radix-2 et radix-4. En général, celles-ci se font par rapport à l'ordre dans lequel les opérations papillon sont exécutées (Rabiner & Gold, 1975). Des algorithmes utilisant des bases différentes de deux ou de quatre ou utilisant une combinaison de différentes bases existent aussi. Les algorithmes de FFT peuvent aussi bien être programmés en logiciel qu'implémenté en matériel. Ce travail porte sur la conception matérielle ou embarquée d'un module de FFT configurable. Les architectures intéressantes à un tel design sont principalement celles utilisant les algorithmes de types radix-2 ou radix-4. Le prochain chapitre traitera du choix de l'architecture la plus appropriée.

CHAPITRE 2

SPÉCIFICATIONS ET CHOIX DE L'ARCHITECTURE

2.1 Spécifications

Cette section expose les spécifications que le module de transformée de Fourier doit remplir, notamment au niveau de la reconfigurabilité. Le module conçu doit être flexible et polyvalent. Cela permettra sa réutilisation dans diverses applications.

2.1.1 Langage et technologie pour l'implémentation

Le module doit être programmé en langage de type HDL (Hardware Description Language) pour faciliter la compréhension et permettre l'emploi de variables de configurations. Une approche modulaire et hiérarchique permet la réutilisation des blocs conçus, elle est donc à privilégier. Le langage choisi pour la réalisation de la FFT/IFFT est le VHDL (Very High Speed Integrated Circuit Hardware Description Language) et le code doit être entièrement synthétisable. Les paramètres de configuration doivent être fixés avant la synthèse au niveau hiérarchique supérieur dans le code. La conception ne doit pas être dédiée à une plate forme spécifique, ni à une technologie précise. En revanche, le module est principalement ciblé pour être utilisé dans des plates-formes de type FPGA (Field Programmable Gate Array). Des attributs peuvent servir à optimiser le module pour une certaine plate-forme, mais l'implémentation dans d'autres technologies doit rester possible.

2.1.2 Ressources et vitesse

Les ressources matérielles limitées telles que la logique de calcul et de contrôle ainsi que la mémoire doivent êtres minimisées pour laisser de l'espace au reste du design. En

contrepartie, la fréquence d'opération doit être maximisée afin d'avoir les meilleures performances. Il n'y a pas de valeur précise à atteindre quant à la vitesse requise pour réaliser une transformée, puisque c'est un paramètre qui varie en fonction de l'application de traitement du signal et de la technologie. La latence, délai en cycles d'horloge entre l'entrée du premier échantillon d'une FFT et la sortie du premier point en fréquence, doit aussi être minimisée.

2.1.3 Paramètres fixes

Toutes les données traitées sont complexes, c'est-à-dire qu'elles possèdent une partie réelle et une partie imaginaire. Les nombres sont représentés en complément à deux et on considère le point décimal à la suite du MSB. Ainsi pour un mot de b bits, la plage de représentation des nombres se situe entre $1-2^{-(b-1)}$ et -1. L'incrément entre deux nombres consécutifs est de $2^{-(b-1)}$.

Le module de transformée de Fourier reçoit ses données d'entrée, des échantillons temporels dans le cas d'une FFT, de façon sérielle. La sortie est aussi produite de manière sérielle. Le module doit posséder un port d'entrée et un port de sortie, chacun ayant la largeur d'une donnée complexe. Le module a aussi une entrée pour l'horloge `clk` et une pour un signal d'activation de l'horloge `ce`. Les données n'ont pas besoin d'entrer à un rythme constant, mais les échantillons doivent arriver de manière synchrone avec l'horloge. Il doit aussi posséder un mécanisme pour la synchronisation avec des modules extérieurs, tel qu'un signal pour lui indiquer le début d'une séquence et un signal pour informer que les résultats sont valides.

Le module doit être en mesure de fonctionner en temps réel, c'est-à-dire que pour chaque donnée qui entre, une donnée est produite en sortie. Une fois que la dernière donnée d'une séquence de N points est fournie au module, celui-ci est en mesure d'accepter la première donnée de la séquence suivante. Aucune étape intermédiaire de traitement ne

doit rompre le rythme. On doit aussi pouvoir opérer le module pour réaliser une seule transformée, puis d'attendre la suivante.

Les résultats de la FFT font suite à plusieurs étapes de calcul qui peuvent introduire des erreurs dues à la représentation discrète des nombres et à l'utilisation de la notation en point fixe. Ces erreurs de quantification doivent être les plus faibles possible. Il est même souhaitable de caractériser ces erreurs et d'identifier leurs sources.

2.1.4 Paramètres reconfigurables

Plusieurs paramètres du module doivent être configurables pour répondre aux besoins des différentes applications de traitement numérique du signal nécessitant une transformée de Fourier. Ces paramètres sont énumérés et détaillés dans les sous-sections suivantes.

2.1.4.1 Taille

La taille de la transformée de Fourier doit être paramétrée. Ainsi, toutes les tailles de puissance de deux peuvent être sélectionnées puisque l'algorithme de la FFT se prête bien à faire des transformées de taille puissance de deux. L'espace occupé par le module croît en fonction de la taille de la transformée. La limite pour la taille maximale n'est pas imposée par le module lui-même, mais par les ressources logiques et mémoires requises. Seulement faire des transformées ayant des tailles puissances de quatre, comme c'est le cas avec certains algorithmes, serait trop limitatif.

2.1.4.2 Largeur des mots binaires

L'espace occupé par le module ainsi que la précision des calculs sont largement dépendant de la largeur utilisée par les mots binaires.

2.1.4.2.1 Entrée et sortie

La largeur des ports en entrée et en sortie doit pouvoir être choisie indépendamment. Des valeurs de l'ordre de 5 à 20 bits comme largeur de mot binaire sont acceptables. En tout temps, on retrouve la même taille pour la partie réelle et pour la partie imaginaire.

2.1.4.2.2 Facteurs de phase

La largeur binaire des facteurs de phase doit aussi être un paramètre configurable. La précision des résultats et l'utilisation de ressources mémoires en dépendent. Des largeurs binaires semblables à celles des données, soit de 5 à 20 bits.

2.1.4.2.3 Interne

Les largeurs des différents bus de données internes doivent être paramétrées de manière à conserver ou à améliorer la précision des résultats. Pour ce faire, des mécanismes d'augmentation de la largeur des mots binaires pour empêcher le débordement et/ou pour réduire la largeur, tels que l'arrondi ou la troncation, doivent être employés.

2.1.4.3 Type de transformée

Le passage du domaine du temps au domaine de la fréquence, autant que celui du domaine de la fréquence au domaine du temps doivent être possibles. L'usager du module doit avoir le choix entre une FFT ou une IFFT. Ces deux transformées contiennent essentiellement les mêmes mécanismes mathématiques. Dans le cas de l'IFFT, il n'est pas nécessaire d'effectuer la division par N comprise dans l'équation 1.2 de l'IDFT. Ainsi, seul le signe de l'exponentielle complexe exécutant une rotation de phase change.

2.1.4.4 Ordonnancement de l'entrée

En général dans les différentes architectures de FFT, si les données arrivent au module de manière chronologique (ordre naturel), elles ressortent en ordre des bits renversé (voir section 1.2.2). L'inverse est aussi vrai: une entrée en ordre renversé donnera une sortie en ordre naturel. Le choix de l'ordonnancement en entrée doit être disponible pour permettre la cascade d'une FFT à l'entrée ordonnée et d'une IFFT à l'entrée en ordre des bits renversé. Cette dernière produira une sortie ordonnée.

2.1.4.5 Mise à l'échelle

La mise à l'échelle correspond à une division par deux pour éviter le débordement, puisqu'un bit plus significatif peut s'ajouter suite à une addition ou une soustraction. Elle peut aussi être vue comme l'accroissement de la plage dynamique de représentation des nombres. La transformée de Fourier est un calcul propice au débordement puisque la puissance moyenne des points discrets en fréquence $X(k)$ est d'un facteur N supérieur à celle des points dans le temps $x(n)$. Ceci peut être expliqué par la relation de Parceval discrète exprimée par l'équation 2.1 (Rabiner & Gold, 1975).

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 \tag{2.1}$$

Par exemple, le pire cas d'entrée se trouve à être une sinusoïde de même fréquence pour la partie réelle et pour la partie imaginaire et d'amplitude un. La FFT, d'un tel signal échantillonné en N points équidistants, occupe une seule raie fréquentielle avec une amplitude de valeur N. Dans ce cas, pour éviter les débordements et respecter la représentation des nombres en point fixe, on doit faire une mise à l'échelle de N. Dans l'étude faite par Knight et Kaiser (1979), il est prouvé que le nombre de division par deux nécessaires, s, peut être approximé par l'équation 2.2. Dans cette expression, v =

$\log_2 N$, p_o et p_i sont respectivement le rapport amplitude crête sur moyenne quadratique à l'entrée (temps) et à la sortie (fréquence).

$$s \simeq \frac{v}{2} + \log_2\left(\frac{p_o}{p_i}\right) \leqslant v+1 \tag{2.2}$$

La borne supérieure de la mise à l'échelle pour ne pas faire de débordement se trouve à être $2N$. Par contre, des mises à l'échelle de moindre importance peuvent être suffisantes selon la nature du signal d'entrée.

2.2 Comparaison des architectures matérielles

Les algorithmes de FFT basés sur l'approche *divide-and-conquer* exposés à la section 1.2 ont été maintes fois réalisés, autant sous forme logicielle que matérielle. La comparaison que l'on retrouve dans cette section, est uniquement faite au niveau des architectures dédiées à une réalisation matérielle. La FFT peut aussi être réalisée avec un processeur ou un DSP. Les sujets touchant une implémentation logicielle sont à l'extérieur du cadre de ce travail.

Il est possible de tirer profit de la structure régulière de l'algorithme pour réduire la complexité de l'implémentation matérielle. Voyons d'abord les techniques et tendances fréquemment employées et ensuite certaines approches plus marginales. Les deux approches traditionnelles sont l'architecture à réutilisation du papillon et l'architecture pipelinée. On pourra ensuite faire le choix de l'architecture qui répond le mieux aux spécifications.

2.2.1 Architectures à réutilisation du papillon

2.2.1.1 Fonctionnement

L'architecture à réutilisation du papillon consiste à utiliser la même unité de traitement, généralement un papillon radix-r, pour faire le plus d'opérations possibles. On sait que graphiquement l'algorithme de la FFT peut se représenter par N/r rangées et $\log_r(N)$ colonnes de calcul papillon. Ainsi, $(N/r)\log_r(N)$ opérations papillons radix-r sont requises pour effectuer une transformée de N points. Le tableau I donne les valeurs du nombre d'opérations papillons pour diverses tailles et pour les bases (r) les plus fréquemment employées: deux et quatre. Peu importe sa taille, on pourrait utiliser un seul papillon pour faire tout le calcul d'une transformée. C'est le principe qui est à la base de l'architecture à réutilisation du papillon. Pour ce faire, il suffit de mémoriser les résultats intermédiaires issus de chaque calcul du papillon. Des algorithmes dit d'adressage *in-place* ou sur place (Rabiner & Gold, 1975) permettent d'utiliser un seul tampon de N espaces mémoire pour stocker tous les résultats. Un adressage précis permet de lire et d'écrire les cases mémoire sans conflit, généralement en suivant les colonnes d'un algorithme de FFT du haut vers le bas. D'autres tampons mémoire sont aussi nécessaires à l'entrée et à la sortie d'une telle architecture pour permettre l'arrivée continue des échantillons.

Tableau I

Nombre de papillons nécessaires pour réaliser une FFT de taille *N*

N	**Opération papillon radix-2**	**Opération papillon radix-4**
4	4	1
8	12	-
16	32	8
32	80	-
64	192	48
256	1024	256
1 024	5 120	1 280
4 096	24 552	6 144
16 384	114 688	28 672

Il existe différentes méthodes de contrôle et d'adressage pour l'allocation sur place de la mémoire. Des multiplexeurs et/ou un bloc d'ordonnancement sont nécessaires pour faire se rencontrer les bonnes données issues des tampons de mémoire. Pour générer les adresses, le contrôleur contient en général des compteurs pour la rangée et la colonne ainsi que des registres à décalage. Johnson (1992) explique l'algorithme permettant la construction matérielle du générateur d'adresses d'une mémoire multibanque. Son approche nécessite des commutateurs (*barrel shifter*) avant et après l'unité de traitement et des multiplexeurs. Il fait aussi mention du mécanisme d'accès aux facteurs de phase dans une ROM. D'autres articles exposent des techniques d'adressage sur place différentes où la séquence de traitement des papillons est modifiée pour simplifier le contrôleur et l'accès aux facteurs de phase (Hasan & Arslan, 2002a; Ma & Wanhammar, 2000). Par exemple, la génération des adresses pour les données intermédiaires, contenues dans quatre banques de mémoire, et pour la table des facteurs de phase se fait conjointement.

Par souci de minimiser l'espace, Son, Jo, Sunwoo et Kim, (2002) montrent l'architecture d'une FFT de 256 points qui utilise un seul processeur radix-4 et seulement quatre banques de 64 cases mémoires chacune. Le mécanisme d'adressage sur place utilisé y est présenté. Le traitement d'un flot continu de données n'est cependant pas possible avec cette architecture sans ajouter d'autres tampons de mémoire. À l'opposé, l'architecture présentée dans (Szedo, Yang, & Dick, 2001) utilise deux unités de traitement et quatre tampons divisés chacun en quatre banques de mémoire à double ports. Cette dernière architecture permet de traiter un flot continu de données et de faire des FFT/IFFT sur 64, 256 ou 1024 points.

Analysons maintenant l'architecture à réutilisation du papillon qui serait utilisée pour la conception d'un module reconfigurable. On considère qu'une opération papillon requiert un cycle d'horloge. Ceci est possible si r données entrent dans le papillon en même temps et que le traitement de celui-ci est pipeliné. On considère aussi que les échantillons complexes sériels entrent dans la transformée à raison d'un par cycle et sont ensuite parallélisés. Un seul papillon radix-2 est nécessaire pour faire une transformée en temps réel sur quatre points, mais plus de papillons sont requis pour des tailles supérieures. Un papillon radix-2 requiert six additionneurs et quatre multiplicateurs.

D'après le tableau I, une transformée de 256 points prend 256 cycles en utilisant un papillon radix-4. C'est le cas limite pour atteindre le temps réel avec un seul papillon. Toutes les tailles inférieures nécessitent un seul papillon radix-4 et les tailles supérieures jusqu'à 2^{16} en nécessitent deux. Dans le reste de la discussion, nous nous concentrerons sur l'architecture à réutilisation du papillon de type radix-4. Cette architecture est illustrée à la figure 7. Augmenter la taille en gardant le même nombre de processeurs occasionne l'ajout de mémoire dans les banques et l'augmentation du nombre de passages des données dans le ou les unités de traitement. Le passage d'une à deux unités de traitement provoque l'ajout d'un tampon de N cases mémoires et des signaux de contrôle supplémentaires. Les papillons radix-4 sont bien adaptés à faire des

transformées de taille qui sont égales à des puissance de quatre. On peut aussi les utiliser pour faire des transformées de taille puissance de deux, mais le contrôle devient plus complexe et/ou il faut forcer des entrées du papillon à zéro. L'architecture ne s'en trouve donc pas réduite. Vu la présence de tampons à l'entrée et à la sortie, les deux types d'ordonnancement sont disponibles.

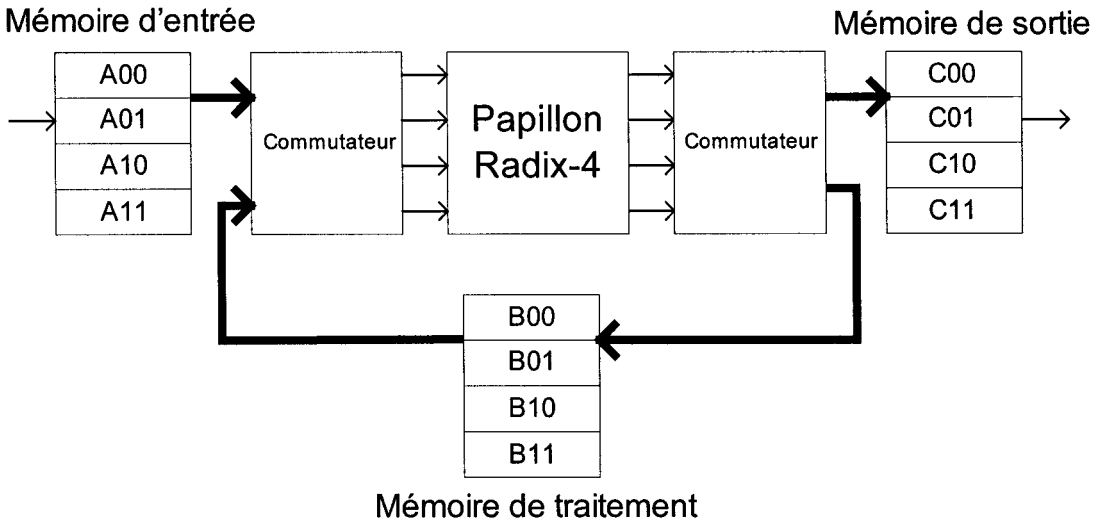


Figure 7 Schéma de l'architecture à réutilisation du papillon radix-4 pour une taille inférieure ou égale à 256

2.2.1.2 Ressources

Voyons maintenant les ressources requises par une architecture à réutilisation du papillon radix-4. Pour faire une comparaison générale, nous ne considérerons pas la complexité des mécanismes d'accès à la mémoire, de commutation et de contrôle. Pour des tailles inférieures ou égales à 256, un papillon radix-4 et trois tampons de mémoire sont requis (figure 7). Le papillon contient huit additionneurs ou soustracteurs complexes ainsi que trois multiplicateurs complexes, ce qui fait 22 additionneurs et 12 multiplicateurs non complexes. Chacun des trois tampons est divisé en quatre banques de taille $N/4$ ayant des ports distincts en entrée et en sortie. Le tampon à l'entrée permet

de paralléliser les échantillons, un autre permet de stocker les résultats intermédiaires et le troisième permet de sortir les résultats en série. Ces mémoires peuvent être utilisées de manière ping-pong, c'est-à-dire ayant des rôles interchangeables.

Trois facteurs de phase sont envoyés aux multiplicateurs à chaque cycle. Différentes façons peuvent être employées pour générer ces coefficients. La mémoire nécessaire pour stocker les facteurs de phase est en général de $3/4N$ espaces par multiplicateur complexe. On peut tirer profit de la symétrie des coefficients ayant des adresses avec une représentation binaire semblable pour réduire la taille des ROM. Une technique nécessite une ROM de $N/8+1$ espace avec de l'adressage et du traitement particulier (Hasan & Arslan, 2002b). On peut aussi envisager une seule mémoire de type ROM ayant trois ports de sortie.

La taille N=1024 est un cas particulier où l'on peut tirer profit du fait que le dernier étage ne requière pas de multiplications (Szedo, Yang, & Dick, 2001). Les deux unités de traitement sont donc un papillon radix-4 standard et un papillon radix-4 sans multiplicateur. On a ainsi besoin de 38 additionneurs, 12 multiplicateurs et quatre tampons de 1024 espaces. Pour les tailles supérieures, il faut deux unités de traitement complètes, donc 44 additionneurs, 24 multiplicateurs et 4 tampons.

Le pourcentage d'utilisation des ressources varie selon la taille de la FFT. La figure 8 détaille les étapes de fonctionnement pour trois tailles. On voit que l'entrée et la sortie d'une séquence de N points se produit à raison d'une donnée par cycle. N données se rencontrent en $N/4$ cycles pour faire le traitement correspondant à une colonne de l'algorithme, représenté par un carré avec une lettre à l'intérieur sur la figure 8. On remarque que l'unité de traitement fonctionne à 75% pour N=64 et à 100% pour N=256. Dans le cas de N=1024, le papillon standard fonctionne à 100% et le papillon modifié ne fonctionne qu'à 25%.

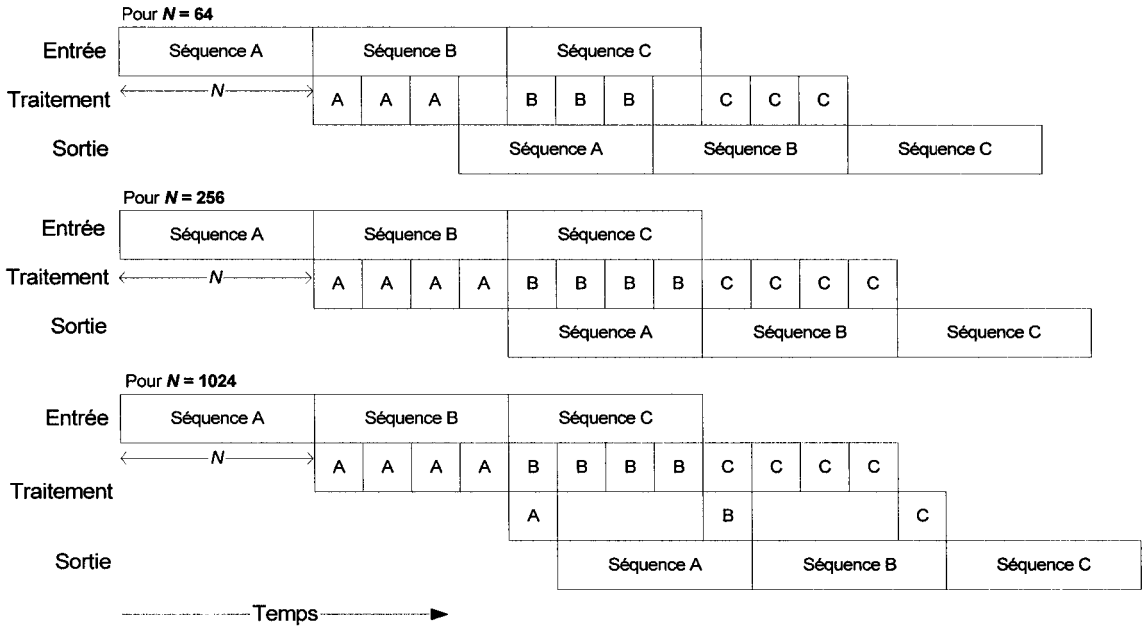


Figure 8 Latence requise par une architecture à réutilisation du papillon pour des FFT de 64, 256 et 1024 points

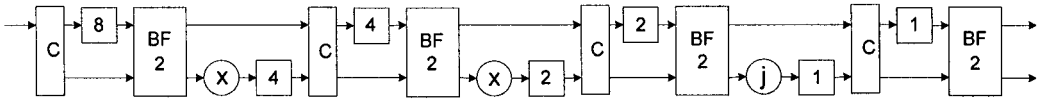
2.2.1.3 Latence

Le temps de transition des données entre l'entrée et la sortie est grandement dépendant de la taille de la transformée. Il se divise de la manière suivante: temps pour entrer dans le 1er tampon (N cycles) et temps de traitement ($\frac{N}{4}\log_4 N$ cycles + nombre d'étages de pipeline dans le papillon). La latence correspond donc à $N + \frac{N}{4}\log_4 N$ cycles, en omettant le pipeline interne qui permet une plus grande fréquence d'horloge. La figure 8 détaille cette latence pour différentes tailles.

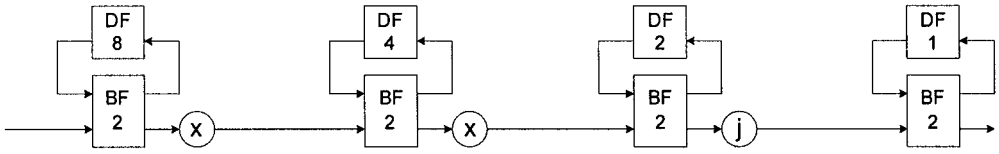
2.2.2 Architectures pipelinées

2.2.2.1 Fonctionnement

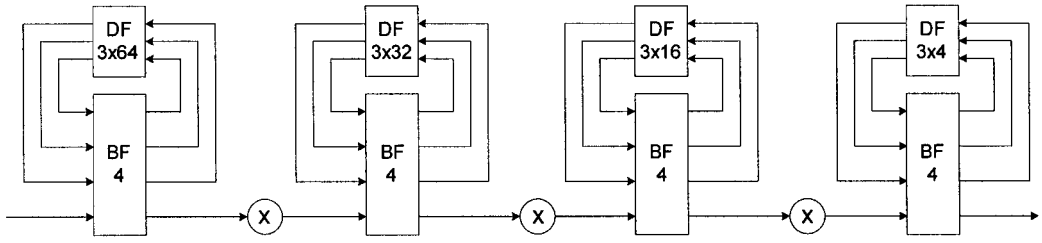
L'architecture pipelinée consiste à implémenter en matériel l'équivalent d'une rangée de l'algorithme de FFT. Pour ce faire, $\log_r N$ unités de traitement sont nécessaires. Une telle architecture est conçue pour recevoir un flot continu de données et pour produire les résultats au même rythme. Des tampons d'entrée et de sortie ne sont pas requis, puisque la parallélisation des données n'a pas à être faite. Il existe différentes approches de gestion de la mémoire et de commutation permettant aux bonnes données de se rencontrer. Les articles (He & Torkelson, 1996, 1998) dressent un bref historique de ces méthodes que nous reprenons ici par la figure 9 et le tableau II. Habituellement, dans une approche pipelinée, on sépare les opérations de multiplication des opérations d'addition et de soustraction. Un papillon radix-*r* (BF*r* sur la figure 9) représente les opérations additives seulement. Les références aux publications des designs originaux des FFT pipelinées sont présentées à même le tableau II.



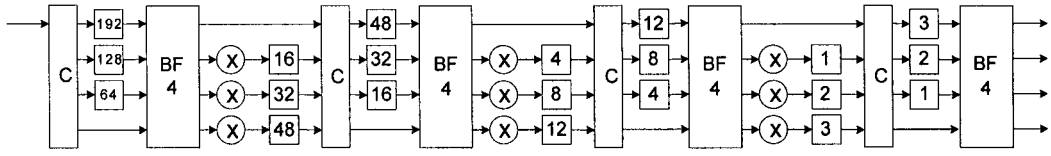
a) R2MDC : *Radix-2 Multi-path Delay Commutator* (*N=16*)



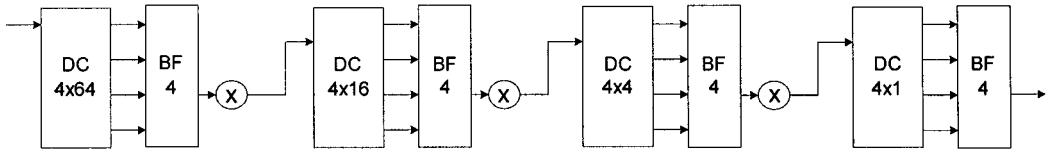
b) R2SDF : *Radix-2 Single-path Delay feedback* (*N=16*)



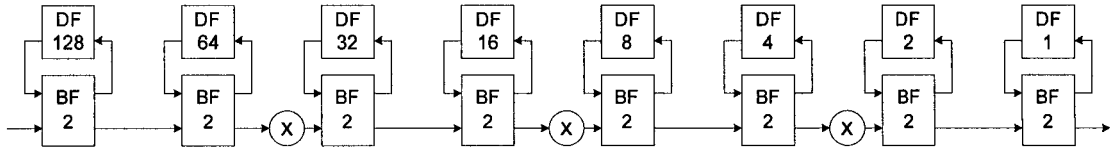
c) R4SDF : *Radix-4 Single-path Delay feedback* (*N*=256)



d) R2MDC : *Radix-4 Multi-path Delay Commutator* (*N*=256)



e) R2SDC : *Radix-4 Single-path Delay Commutator* (*N*=256)



f) R2^{2}SDF : *Radix-2^2 Single-path Delay Feedback* (*N*=256)

Légende: BF*r* représente la partie additive d'un papillon radix-*r* ;
C indique un commutateur;
DF (*Delay Feedback*) représente un délai en rétroaction;
les boîtes contiennent un chiffre indiquant le nombre d'unités de délai.

Figure 9 Diverses architectures matérielles de FFT pipelinée tiré de (He & Torkelson, 1998): a) R2MDC, b) R2SDF, c) R4SDF, d) R4MDC, e) R2SDC et f) R2^{2}SDF

Tous les chiffres du tableau II sont donnés pour des unités de calcul complexe. Pour les besoins de comparaison, on considère que la taille de la transformée N est une puissance de quatre.

Tableau II

Complexité des architectures pipelinées

Architectures (référence)	**Add./sous. complexes dans les BF** (Pourcentage d'utilisation)	**Multiplicateurs complexes** (Pourcentage d'utilisation)	**Élément de délais**	**Contrôle**
a) **R2MDC** (Rabiner & Gold, 1975)	$4\log_4 N$ (50 %)	$2\log_4 N$-2 (50 %)	$3N/2$-2	simple
b) **R2SDF** (Wold & Despain, 1984)	$4\log_4 N$ (50 %)	$2\log_4 N$-2 (50%)	N-1	simple
c) **R4SDF** (Despain, 1974)	$8\log_4 N$ (25 %)	$\log_4 N$-1 (75 %)	N-1	moyen
d) **R4MDC** (Rabiner & Gold, 1975; Swartzlander et al., 1992; Swartzlander et al., 1984)	$8\log_4 N$ (25 %)	$3(\log_4 N$-1) (25 %)	$5N/2$-4	simple
e) **R4SDC** (Bi & Jones, 1989; Bidet, Castelain, Joanblanq, & Senn, 1995)	$3\log_4 N$ (100 %)	$\log_4 N$-1 (75 %)	$2N$-2	complexe
f) **R2^{2}SDF** (He & Torkelson, 1996, 1998)	$4\log_4 N$ (50 %)	$\log_4 N$-1 (75 %)	N-1	simple

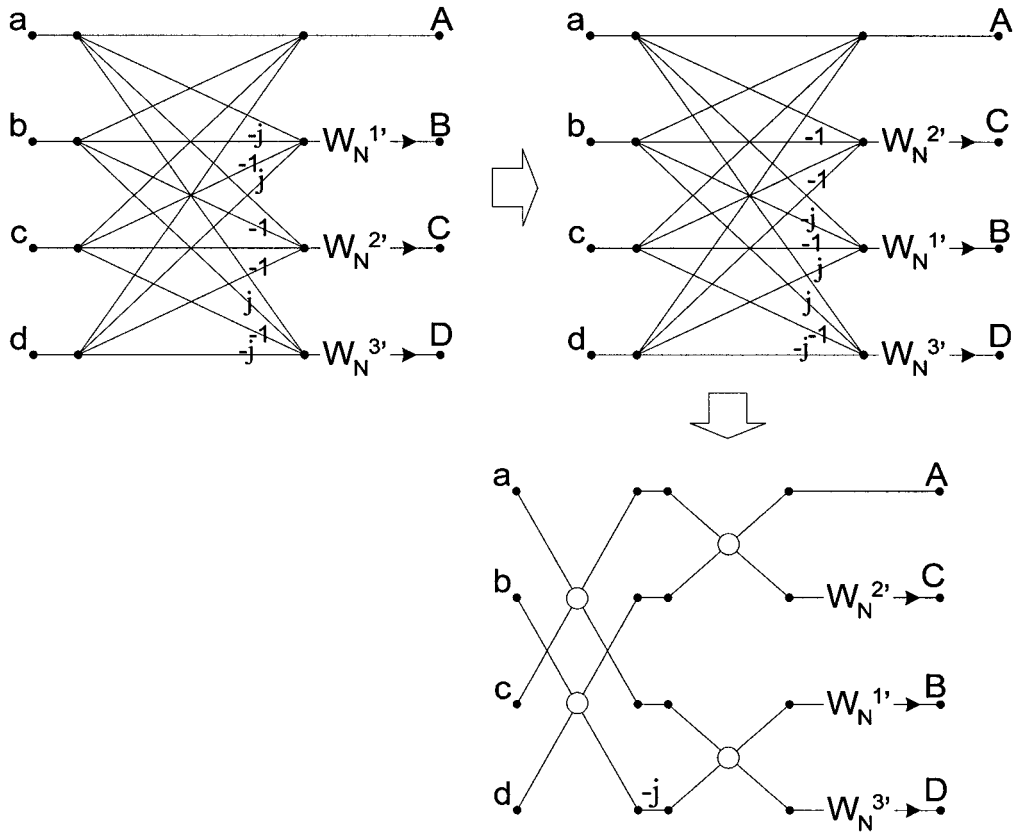
En observant la figure 9, on remarque qu'il y a deux tendances pour synchroniser les données: soit l'utilisation de délai en rétroaction (DF pour D*elay Feedback*) ou soit l'utilisation de commutateurs à délai (DC pour *Delay Commutator*). La quantité de mémoire nécessaire pour la réalisation dépend beaucoup de l'architecture pipelinée choisie. La première approche DF est toujours plus efficace pour l'utilisation de cette ressource que l'approche DC. Les architectures radix-4 à un trajet (R4SDF, R4SDC et R2^{2}SDF) utilisent mieux les multiplicateurs que l'architecture radix-4 multi-trajet (R4MDC). Elles requièrent aussi moins de multiplicateurs que les architectures radix-2 correspondantes (R2MDC et R2SDF) puisque moins d'étages de traitement sont séparés par une multiplication pour une taille de FFT donnée. Par contre, celles possédant le papillon radix-2 (R2MDC, R2SDF, R2^{2}SDF) utilisent mieux les ressources de celui-ci et nécessitent moins d'additionneurs et de soustracteurs. À la lumière de ces résultats, l'architecture pipelinée la plus pertinente à implémenter est celle utilisant l'algorithme R2^{2}SDF. Elle sera détaillée dans les sections suivantes.

2.2.2.2 Cas particulier de l'algorithme Radix-2^2

L'algorithme R2^{2}SDF a la particularité d'utiliser la partie additive des papillons radix-2 tout en ayant la complexité multiplicative d'un algorithme radix-4. C'est de là que proviennent son nom et ces propriétés intéressantes à une implémentation matérielle. Cet algorithme a été expliqué pour la première fois par He et Torkelson (1996). Des implémentations VLSI en ont été faites pour une FFT de 256 points (Vergara, Strum, Eberle, & Gyselinckx, 1998) et pour une FFT de 1024 points (He & Torkelson, 1998). L'architecture semble être facilement paramétrable pour obtenir une FFT de taille variable et pour répondre aux spécifications préalablement exposées.

Les articles préalablement mentionnés expliquent en détail le développement mathématique du radix-2^2, c'est pourquoi il n'est par repris ici. Seule une explication graphique est présentée dans cette section. La figure 10 nous montre la décomposition

d'un papillon radix-4 standard en quatre structures papillons radix-2 (parties additives seulement) séparées par l'opération triviale -j. Pour ce faire, on doit inverser l'ordre de sortie du point B et C, puis découper le traitement en deux étapes.



Les cercles correspondent à la partie additive des papillons radix-2: le résultat de l'addition se trouve sur la branche du haut et de la soustraction sur la branche du bas

Figure 10 Transformation du papillon radix-4 pour obtenir la structure radix-2

La figure 11 permet d'avoir une vue générale de l'algorithme radix-2^2 utilisant la décimation en fréquence. Les cercles correspondent à la partie additive des papillons radix-2. On trouve le résultat de l'addition sur la branche du haut et de la soustraction sur la branche du bas. On peut aussi remarquer que l'ordre des facteurs de phase est modifié par rapport à celui de l'algorithme radix-4 standard et que l'ordonnancement des points en sortie est l'ordre binaire renversé.

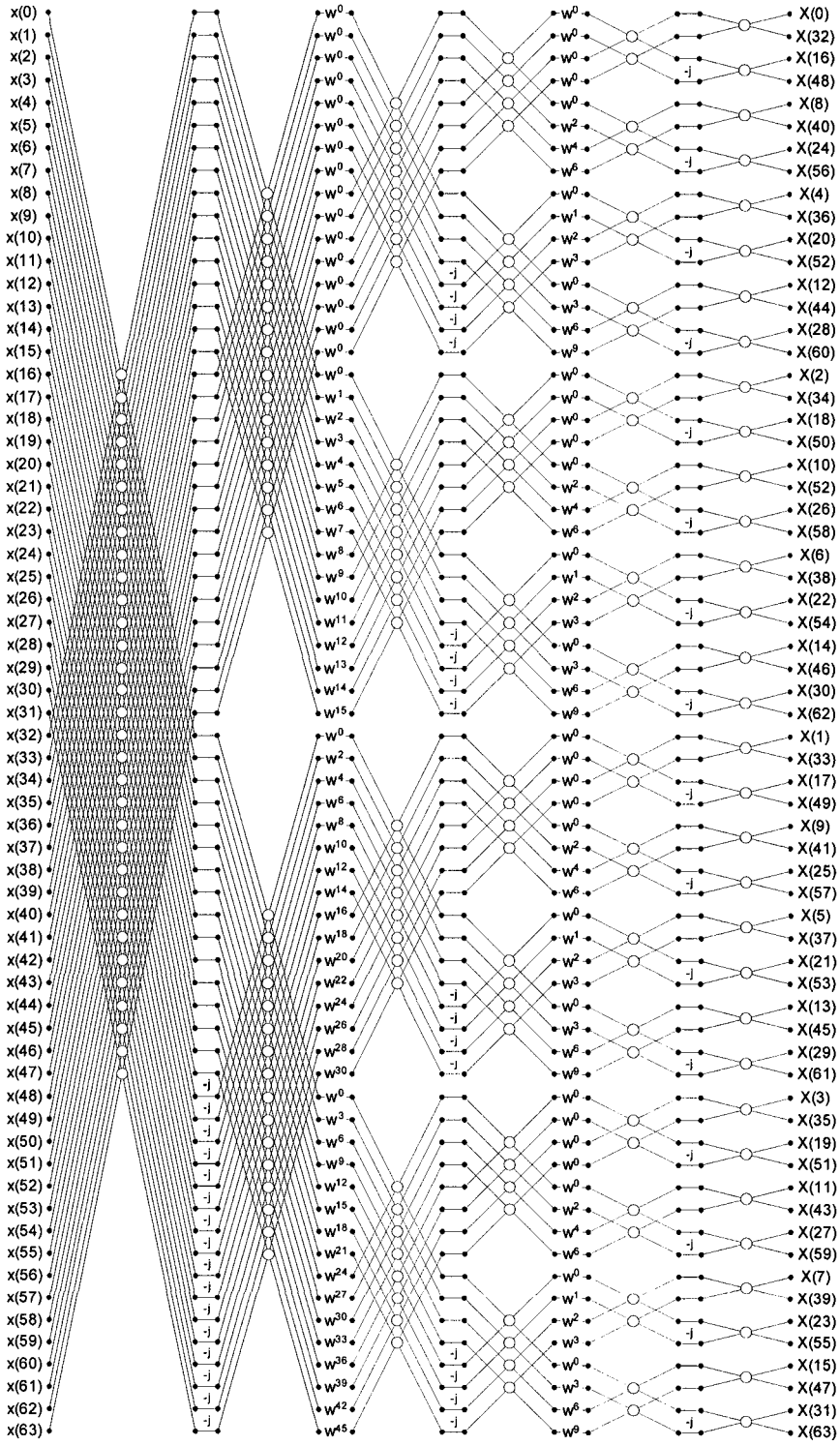


Figure 11 Représentation de l'algorithme Radix-2^2 DIF d'une FFT de 64 points

2.2.2.2.1 Ressources

L'architecture du R2^{2}SDF (figure 9f) est constituée de $\log_2 N$ papillons de type radix-2 contenant chacun un additionneur et un soustracteur complexe, donc quatre additionneurs en complément à deux. Un multiplexeur est aussi nécessaire dans chaque papillon. Celui-ci permet de laisser passer les données inchangées ou de faire le traitement, ce qui confère un taux d'utilisation de 50% . Dans le cas d'une transformée de N une puissance de quatre, un multiplicateur complexe (quatre multiplicateurs et deux additionneurs) est nécessaire entre chaque groupe de deux papillons. Ce regroupement (l'équivalent d'un étage de radix-4) est répété $\log_4 N$ fois pour former une transformée de N points, à l'exception de l'étage de sortie qui ne nécessite pas de multiplication. Les transformées de taille puissance de deux qui ne sont pas une puissance de quatre sont aussi possibles avec cette architecture. L'étage d'entrée étant un seul papillon radix-2 et un multiplicateur complexe. La mémoire nécessaire est divisée en $\log_2 N$ lignes à délai de taille décroissante. En effet, une mémoire de type FIFO accompagne chaque papillon et elle prend une des tailles suivantes: $N/2$, $N/4$, $N/8$, ... 1, respectivement de l'entrée vers la sortie pour une architecture DIF. Ainsi, N-1 espaces mémoires sont nécessaires au total. Pour former une transformée de taille N quelconque, il suffit de répéter les blocs de base de l'architecture et d'ajuster le contrôleur en conséquence.

L'algorithme exposé possède l'entrée en ordre naturel et la sortie en ordre des bits renversé. On peut assez facilement adapter l'algorithme DIF en DIT et pour obtenir l'entrée en ordre des bits renversé et la sortie en ordre naturel. La chaîne de blocs connectés en série est propice au contrôle de la largeur des mots binaires à chaque étape.

2.2.2.2.2 Latence

La latence du R2^{2}SDF est de N cycles en plus des étages de pipeline supplémentaires qu'on retrouve dans les blocs. Par exemple, une multiplication complexe doit être effectuée en plus d'un cycle pour obtenir une fréquence d'horloge plus élevée.

2.2.3 Autres architectures

Il existe de nombreuses autres approches pour concevoir un module matériel de FFT/IFFT. Ces architectures peuvent difficilement faire partie des catégories réutilisation ou pipelinée. Dans cette section nous en citons quelques exemples:

- Une approche hybride est présentée par Hidalgo, Lopez, Arguello, & Zapata (1999): une structure en rangées et en colonnes de processeurs à géométrie fixe est utilisée. Le processeur est un papillon associé à un module de commutation et à de la mémoire locale dont la taille dépend du nombre de processeurs employés pour bâtir l'architecture. Une telle architecture peut être utilisée pour accepter en entrée plus d'un échantillon par cycle ou encore pour faire une transformée de N points en moins de N cycles. Tous les mécanismes d'adressage et de contrôle du flot de données sont décrits dans l'article. Cette méthode semble bien adaptée pour l'implémentation d'un module de FFT à taille fixe dans un ASIC. En revanche, la distribution de la mémoire dans chaque processeur n'est pas avantageuse pour l'implémentation dans un FPGA où des blocs de mémoire sont disponibles. De plus, avec une telle architecture, concevoir un module avec la taille paramétrable est très complexe au niveau de la génération des signaux de contrôle et d'adresse.

- Une architecture systolique utilisant des unités de traitement disposées en rangées et en colonnes est proposée par Dick (1996). Dans cette architecture, le multiplicateur est remplacé par un module effectuant une forte rotation de phase (π, $\pm \pi/2$) suivie

d'un module CORDIC (COordinate Rotation Digital Computer) qui fait une rotation de phase plus fine par un mécanisme itératif. Le CORDIC provoque un gain qui doit être compensé ultérieurement par une mise à l'échelle. Une telle architecture est économe en ressources logiques, mais occasionne une latence plus importante que l'emploi de multiplicateurs. Le contrôle d'une telle architecture est aussi complexe.

- Dans l'article de Wang, Lam, Tsui, Cheng, & Mow (2002), il est proposé d'utiliser la représentation des nombres logarithmique au lieu du complément à deux. Cela a l'avantage de transformer les multiplications en simples additions. Les additions et soustractions pour leur part se font par une addition, une soustraction et un accès à une table de conversion (LUT). Une étape de conversion en arithmétique logarithmique est nécessaire avant la FFT et une étape anti-log après, pour retrouver la notation point fixe du départ. Une telle architecture nécessiterait moins de transistors dans une implémentation ASIC, qu'une implémentation point fixe équivalente. En revanche, il est moins avantageux d'éviter l'opération multiplication dans une implémentation FPGA où des multiplicateurs dédiés d'assez grande taille sont présents.

2.3 Choix de l'architecture pour l'implémentation

Les caractéristiques des deux architectures envisagées sont résumées par le tableau III et les figures 12 et 13. Le nombre d'additionneurs devient supérieur pour l'architecture pipelinée R2^{2}SDF à partir d'une transformée sur 32 points. Par contre, le point de rencontre des deux courbes pour les multiplicateurs est moins évident puisque l'architecture à réutilisation du papillon croît par palier et que l'architecture pipelinée augmente régulièrement avec la taille. Dans la majorité des tailles illustrées, l'architecture pipelinée est meilleure ou équivalent à l'architecture à réutilisation. De plus, les ressources mémoires sont toujours moindres pour l'architecture pipelinée.

Tableau III

Comparaison de la complexité matérielle et de la latence pour les architectures à réutilisation du papillon et pipelinée

Architecture	**Réutilisation du papillon radix-4**			**Pipelinée R2^{2}SDF**	
	$N \leq 256$	$N = 2^{10}$	$2^{10} < N \leq 2^{16}$	N puis. de 2	N puis. de 4
Additionneurs	22	38	44	$5\log_2 N - 1$	$5\log_2 N - 2$
Multiplicateurs	12	12	24	$2\log_2 N - 2$	$2\log_2 N - 4$
Mémoire*	$3N$	$4N$		N-1	N-1
Latence	$N + N/4\log_4 N$			N	N

*N'inclut pas la mémoire pour les facteurs de phase et les registres de pipeline présents dans tous les cas.

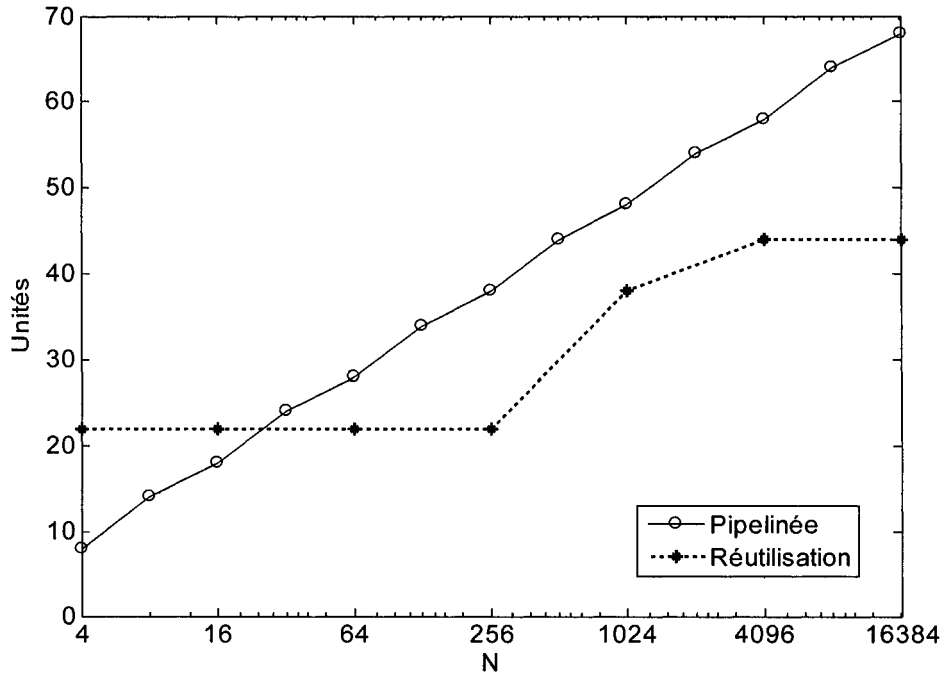


Figure 12 Utilisation des additionneurs

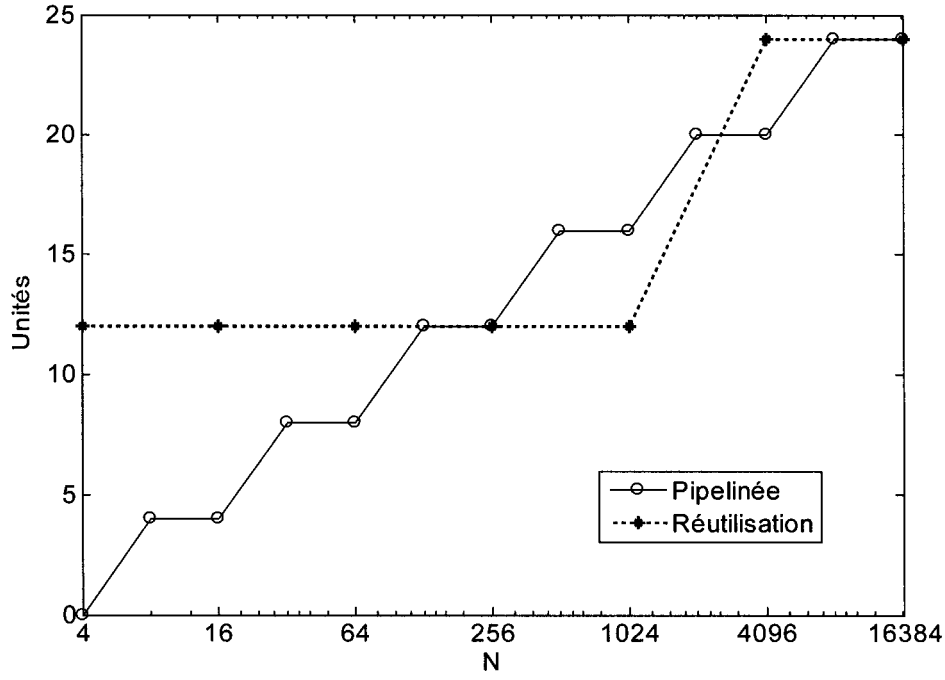


Figure 13 Utilisation des multiplicateurs

L'architecture choisie pour l'implémentation du module configurable de transformée rapide de Fourier est l'architecture pipelinée R2^{2}SDF. Les résultats quantitatifs orientent le choix de la meilleure architecture, mais ils ne sont pas suffisant pour prendre la décision. Les principales raisons qualitatives menant à ce choix sont les suivantes:

- Les blocs formant l'architecture sont réutilisables, ce qui facilite le design. De plus, la croissance est régulière (logarithmique) avec l'augmentation de la taille.
- Le traitement des données sériel permet de minimiser la latence et l'utilisation des ressources de mémoire.
- Le contrôle de l'architecture pipelinée est moins complexe que celui de l'architecture à réutilisation du papillon, surtout pour les tailles où v est impair. Le contrôle n'a pas besoin d'être global, il peut facilement être distribué à même chaîne de traitement.
- La configuration de la largeur des mots binaires pour avoir un contrôle sur la précision des résultats est possible. Dans le cas où la mémoire serait utilisée sur place

(*In-place)*, il ne serait pas possibles de changer la largeur progressivement au cours d'une transformée.

Ainsi, l'architecture pipelinée de type R2^{2}SDF répond le mieux aux spécifications de reconfigurabilité et de ressources.

CHAPITRE 3

ARCHITECTURE ET CONSTRUCTION HIÉRARCHIQUE

3.1 Architecture et chemin de données

L'algorithme du Radix-2^2 détaillé à la section 2.2.2.2 a été adapté pour être implémenté dans une architecture matérielle. Pour faciliter la compréhension du chemin de données, le schéma de cet algorithme est repris ici à la figure 14. Dans la suite de ce travail, on nommera le module reconfigurable de FFT/IFFT: R2^{2}PC, pour Radix-2^2 Parameterizable Core.

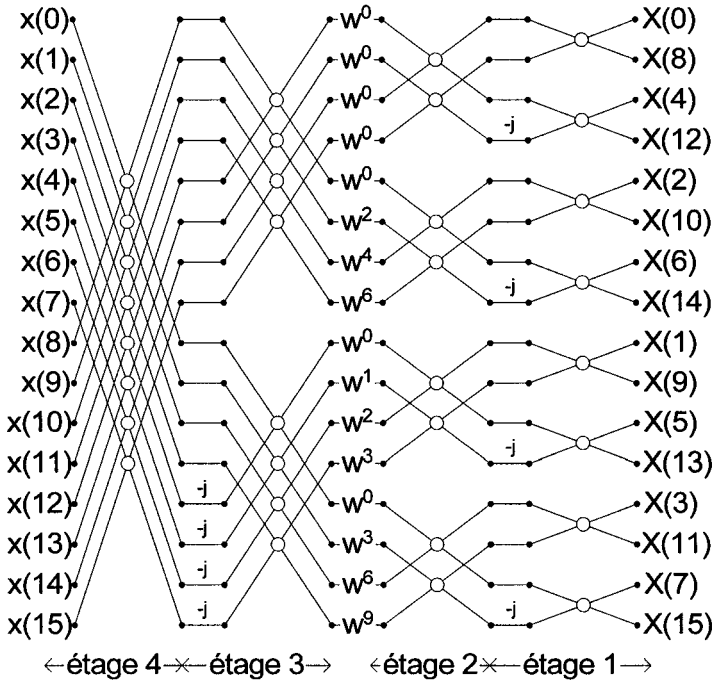


Figure 14 Algorithme de la FFT Radix2^2 DIF pour N=16 (v=4)

Sur le schéma bloc du R2^{2}PC, présenté à la figure 15, on remarque que les blocs sont interconnectés de manière sérielle. Pour effectuer une transformée de N points, seulement v papillons radix-2 sont requis, où $v = \log_2 N$. Chaque papillon est accompagné d'une ligne à délai en rétroaction qui permet la synchronisation des données. La combinaison du papillon et du délai en rétroaction sera nommé étage et on y attribue un numéro prenant la valeur de v à 1, numéroté respectivement de l'entrée vers la sortie de la transformée. Entre chaque groupe de deux étages, on retrouve un multiplicateur complexe accompagné d'une table de facteurs de phase. Rappelons que le multiplicateur n'est pas requis après l'étage 1. Le contrôle du R2^{2}PC est effectué par différents contrôleurs qui s'échangent un signal d'activation en provenance de l'entrée. Le contrôleur se trouvant à l'entrée reçoit un signal `startIn`, qui initialise ses sorties, et il contrôle le ou les premiers étages et le multiplicateur associé. Il émet aussi un signal d'activation `startOut` au contrôleur suivant. Ce signal est retardé, par rapport au signal `startIn`, du nombre d'étages de pipeline contenus dans les modules contrôlés. De cette façon, l'activation se produit au bon moment dans chaque contrôleur, jusqu'à la sortie du module.

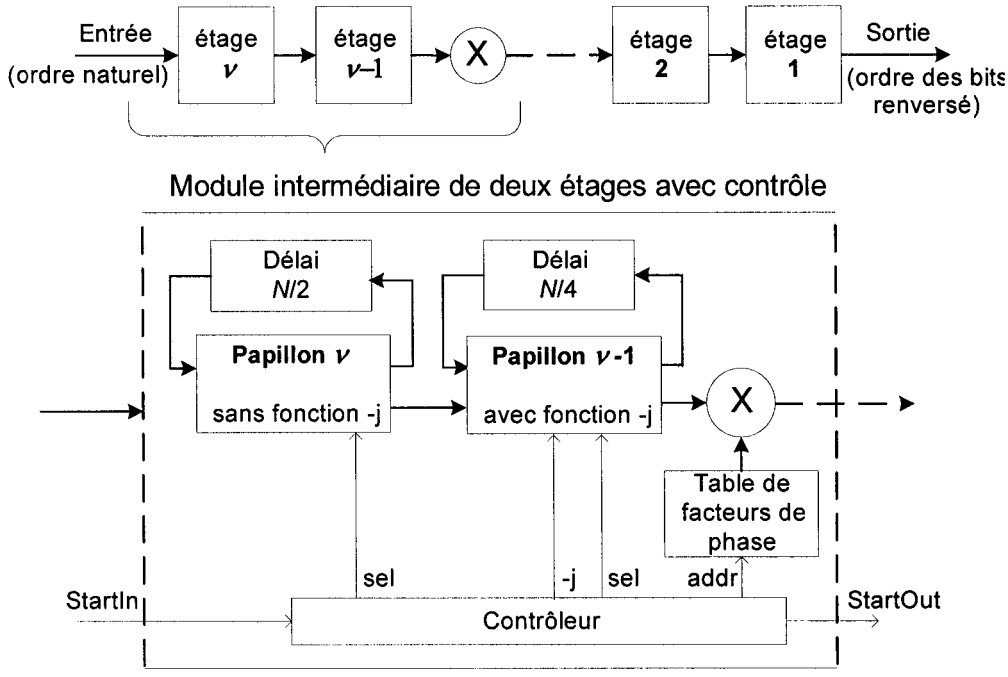


Figure 15 Schéma bloc du module R2^{2}PC

Chaque étage du design effectue le traitement correspondant à une colonne de l'algorithme. La configuration détaillée ci-dessous correspond au cas standard, les autres cas de fonctionnement en découlent et leurs particularités sont décrites à la section 3.2. Il s'agit d'une FFT de taille N puissance de quatre, ayant l'entrée en ordre naturel et la sortie en ordre des bits renversé. Le chemin emprunté par les données est décrit par les étapes suivantes :

- Les échantillons temporels ordonnés entrent en série dans le papillon de l'étage v. Les $N/2$ premiers échantillons d'une séquence, de l'indice 0 à $N/2$-1, passent au travers du papillon sans aucun traitement et sont envoyés dans la ligne à délai associée au papillon v. Cette dernière crée un délai de $N/2$ données, soit de 2^{v-1}.
- Ensuite, le mode de fonctionnement du papillon commute, du mode de transmission au mode de traitement. Le papillon effectue des opérations mathématiques, addition et soustraction, entre le prochain échantillon soit $x(N/2)$ et l'échantillon qui sort du délai

en rétroaction soit $x(0)$. Sur la figure 14, il s'agit du traitement fait par le 1er papillon (cercle) de la colonne de gauche. Le résultat de l'addition, branche du haut du cercle est envoyé à l'étage suivant v-1, tandis que le résultat de la soustraction, branche du bas, est transmis dans la ligne à délai en rétroaction.

- Le mode de traitement est maintenu jusqu'à ce que le calcul soit fait avec les échantillons $x(N/2\text{-}1)$ et $x(N\text{-}1)$, le papillon se remet alors en mode de transmission des données sans faire de traitement. Pendant que l'échantillon $x(0)$ d'une nouvelle séquence est envoyé dans la ligne à délai, le premier résultat de soustraction mémorisé dans la ligne à délai est envoyé vers l'étage suivant et ainsi de suite.
- À l'entrée de l'étage v-1, les échantillons arrivent en série dans l'ordre correspondant au haut vers le bas de la colonne v-1 illustrée à la figure 14. À cet étage, la commutation entre les deux modes de fonctionnement du papillon se produit deux fois plus souvent qu'à l'étage précédent pour permettre aux bonnes données de se rencontrer. Ainsi à l'étage v-1, le délai en rétroaction à une taille de 2^{v-2}. Si une multiplication triviale par -j doit être effectuée, le calcul est inclus à l'intérieur du papillon de l'étage impair. Les données de cet étage sortent aussi dans le même ordre qu'une colonne de l'algorithme et elles sont transmises en série à un multiplicateur complexe.
- Un seul multiplicateur complexe permet d'effectuer N multiplications avec les facteurs de phase. Un multiplicateur complexe est ainsi nécessaire seulement à tous les deux étages, sauf à la suite de l'étage 1. On considère le facteur de phase W^0 (correspondant à la valeur 1 + 0j) nécessitant le traitement d'une vraie multiplication complexe (voir section 3.3.3.3).
- Le groupe de deux étages suivant un multiplicateur ont le même chemin de données que les deux premiers étages. En revanche, ils ont des délais toujours de la demi-taille de celle de l'étage précédant. Jusqu'à ce qu'elle soit de un pour l'étage 1, l'étage de sortie. La commutation entre le mode de transmission et le mode de traitement se produit à tous les $2^{\text{no. étage-1}}$ cycles, permettant ainsi de remplir le délai associé.

3.2 Particularité des différentes configurations

Selon ses besoins, l'utilisateur du R2^{2}PC peut choisir différents paramètres. Ceux-ci ont une influence directe sur l'architecture matérielle qui sera implémentée et sur les interconnections du chemin de données. Parmi ces paramètres on retrouve la taille de la transformée, son type (directe ou inverse) et l'ordonnancement de l'entrée.

3.2.1 Cas d'une FFT avec *N* puissance de quatre

Le chemin de donnée pour une transformée de Fourier directe de taille puissance de deux paire à l'entrée en ordre naturel est celui décrit à la section 3.1. En effet, avec N étant une puissance de quatre, on retrouve un nombre pair d'étages. Les données traversent donc des groupes formés de deux étages suivis d'un multiplicateur.

3.2.2 Cas d'une FFT avec *N* puissance de deux mais non de quatre

Pour les transformées avec N une puissance de deux, mais non de quatre, on retrouve un nombre impair d'étages. Dans le cas d'une transformée à l'entrée en ordre naturel, l'étage v est particulier. Il s'agit d'un étage radix-2 régulier, c'est-à-dire un papillon accompagné d'une ligne à délai de taille 2^{v-1} et suivi par un multiplicateur complexe. La figure 16a) illustre ce cas.

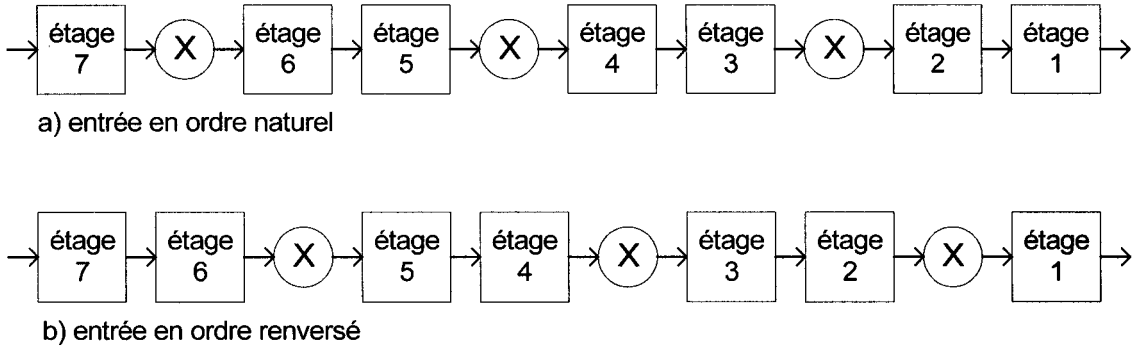


Figure 16 Structure d'une FFT de 128 points

Dans le cas où l'entrée est en ordre binaire renversé, le papillon radix-2 solitaire se retrouve à l'étage 1, comme illustré par la figure 16b). Cet étage n'est pas suivi d'un multiplicateur puisqu'il se situe en position de sortie. Peu importe l'ordonnancement, on retrouve donc (v-1)/2 multiplicateurs complexes pour un v impair au lieu de v/2 -1 pour un v pair.

3.2.3 Cas d'une IFFT

Pour effectuer une transformée de Fourier inverse, on doit ajouter au traitement standard de la FFT l'opération conjugué complexe avant le premier étage et après le dernier. Habituellement pour faire une IFFT au lieu d'une FFT, on prend le conjugué complexe du facteur de phase. L'équation (3.1) prouve que prendre le conjugué complexe à l'entrée et à la sortie revient au même. Le calcul du conjugué complexe correspond au changement de signe de la partie imaginaire. Au point de vue de l'implémentation, un additionneur est nécessaire à cette opération vu la représentation des nombres en complément à deux. La division par N devant la sommation dans la formule de l'IDFT (eq. 1.4) n'est pas implicitement incluse dans le R2^{2}PC. En revanche, elle peut être effectuée par une mise à l'échelle adéquate.

$$X(k)=\left[\sum_{n=0}^{N-1} x(n)^* \mathrm{e}^{-j\frac{2\pi}{N}nk}\right]^*$$
$$X(k)=\sum_{n=0}^{N-1} (x(n)^*)^* (\mathrm{e}^{-j\frac{2\pi}{N}nk})^* \qquad (3.1)$$
$$X(k)=\sum_{n=0}^{N-1} x(n)\mathrm{e}^{j\frac{2\pi}{N}nk}$$

3.2.4 Cas d'une entrée en ordre des bits renversé

Le chemin de données exposé à la section 3.1, en est un du type *Differentiation In Frequency* (DIF), où l'étage d'entrée fait le calcul entre des données en ordre naturel. Les délais en rétroaction de taille décroissante de l'entrée vers la sortie servent à distancer les points traités. En effet, le premier couple de données traité est $x(0)$ avec $x(N/2)$. Ce type d'architecture DIF se remarque par une partie additive suivie de la multiplication. Pour pouvoir réaliser le même traitement avec une entrée en ordre des bits renversé, l'architecture doit être du type *Differentiation In Time* (DIT). La figure 17 illustrant l'algorithme DIT montre que le chemin de données est l'inverse de celui présenté à la figure 3.1 pour la FFT DIF. Dans cette architecture, les lignes à délai en rétroaction sont de taille croissante de l'entrée vers la sortie; la taille est de 1 pour l'étage v. Avec l'ordonnancement renversé, on observe que les deux premiers points qui entrent dans la transformée sont x(0) et x(N/2). Ils sont immédiatement traités ensemble. L'algorithme DIF effectue aussi le premier calcul avec ces deux premiers points, mais suite à un délai. L'architecture de base, soit la séquence de deux papillons suivie d'un multiplicateur complexe, se retrouve aussi dans le cas d'une entrée en ordre binaire renversé. Pour une taille donnée, on retrouve le même nombre de multiplicateurs et la même latence peu importe le type d'ordonnancement de l'entrée.

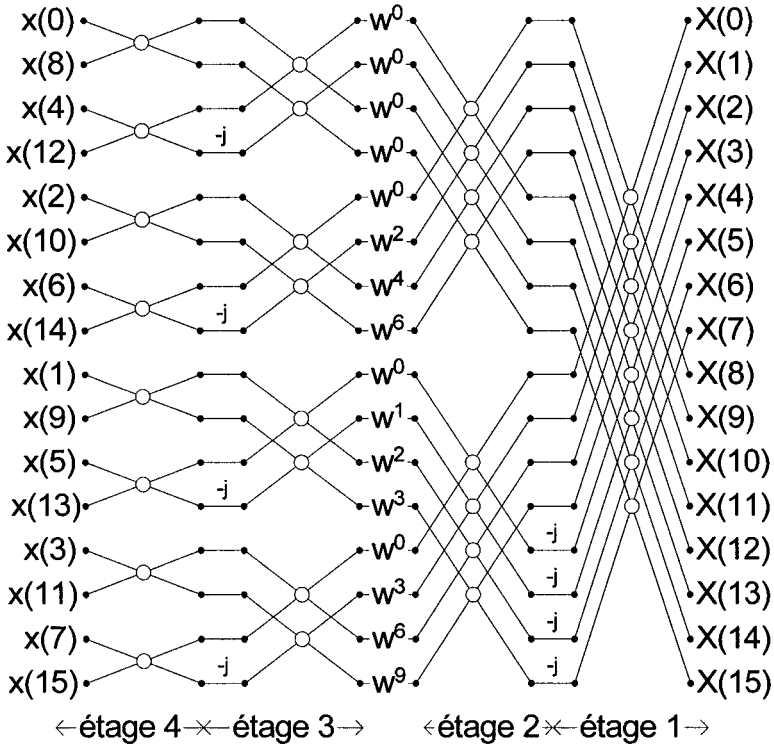


Figure 17 Algorithme de la FFT Radix2^2 DIT, entrée on ordre binaire renversé et sortie en ordre naturel, pour N=16

3.3 Construction hiérarchique

Une approche de conception hiérarchique descendante a été employée. Le R2^2PC est construit en VHDL sur trois niveaux. D'abord, on retrouve le niveau supérieur qui instancie, selon les paramètres fournis, successivement les étages simples ou doubles de la transformée. Ensuite, le niveau intermédiaire a uniquement une architecture structurelle qui permet les interconnections des sous-modules et l'assignation des différents paramètres de configuration à ceux-ci. Enfin, on retrouve au plus bas niveau, les sous-modules réutilisables qui possèdent la logique nécessaire au calcul et au contrôle. Au travers des sections suivantes, les différents modules sont décrits et illustrés.

3.3.1 Niveau supérieur

Le niveau hiérarchique supérieur, qui se nomme R2^{2}PC, est celui qui établit l'interface avec l'usager. C'est à ce niveau que tous les paramètres de configuration peuvent être choisis à l'aide de *generics*. La taille de la transformée dicte le nombre d'étages à instancier et l'ordonnancement de l'entrée établit la manière de configurer les étages (ceux-ci se trouvant au niveau hiérarchique intermédiaire). C'est une boucle *for-generate* qui s'exécute pour construire la transformée de *v* étages.

De nombreuses fonctions s'activent à la compilation avant la synthèse et permettent de fixer adéquatement les largeurs de mot binaire ainsi que les paramètres *generics* pour chaque bloc de niveau inférieur créé. Si une IFFT est requise, l'entrée et la sortie du module sont connectées à des blocs de conjugué complexe.

Fichier : *r22pc.vhd*

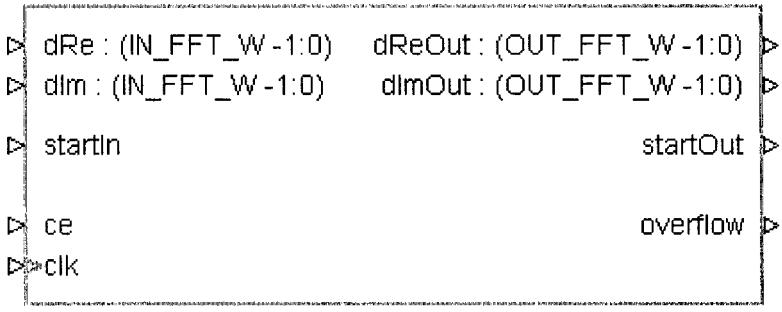


Figure 18 Symbole du R2^{2}PC

Generics:

IN_FFT_W	positive	Largeur binaire de l'entrée du R2^{2}PC.
OUT_FFT_W	positive	Largeur binaire de la sortie du R2^{2}PC.
TF_W	positive	Largeur binaire des facteurs de phase.
LOG2N	positive	Indication de la taille de la transformée, nombre d'étages nécessaires, soit $v = \log_2 N$.
SCALING	positive	Mise à l'échelle correspondant au nombre de divisions par deux effectuées, valeur entre un et v.
INVERSE	boolean	Vrai pour une transformée inverse et faux pour une transformée directe.
BIT_REV_IN	boolean	Ordonnancement de l'entrée, vrai pour l'ordre binaire renversé et faux pour l'entrée en ordre naturel.
W	width_typ	Largeur binaire des bus inter-étages suivant les multiplicateurs. Le type *width_typ* est une liste d'entiers positifs défini dans le package *define_width.vhd.*
MULT_STYLE	string	Type de ressources pour l'implémentation des multiplicateurs, "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.
MULT_ROUND_W	positive	Largeur binaire du bus de données précédent les multiplicateurs complexes, paramètre fixé à 18 dans le cas d'un multiplicateur dédié de Xilinx.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Indique le début d'une transformée ou d'une suite de transformées.
dRe	Partie réelle du vecteur d'entrée.
dIm	Partie imaginaire du vecteur d'entrée.

Sorties:

startOut	Indique la fin de la latence suite à un signal *startIn.*

`overflow`	Indique un débordement dans les calculs et donc une séquence non valide.
`dReOut`	Partie réelle du vecteur de sortie.
`dImOut`	Partie imaginaire du vecteur de sortie.

3.3.1.1 Exemple de configuration

Cette section donne un exemple de l'assignation des paramètres de configuration, *generics,* pour effectuer une IFFT sur 256 points avec une mise à l'échelle de 128. L'entrée est de type renversé et a 10 bits de large. La largeur des bus internes de données varie progressivement de l'entrée vers la sortie. Elle est fixée entre chaque groupe de deux étages après le multiplicateur à 12, 14 et 16 bits. La figure 19 illustre de manière simplifiée l'architecture qui sera instanciée. La sortie est fixée à 17 bits et les facteurs de phase à 16 bits.

```
entity r22pc is
   generic(
      IN_FFT_W      : positive := 10;
      OUT_FFT_W     : positive := 17;
      LOG2N         : positive := 8;
      SCALING       : positive := 7;
      MULT_STYLE    : string   := "block_mult";
      MULT_ROUND_W  : positive := 18;
      INVERSE       : boolean  := true;
      TF_W          : positive := 16;
      BIT_REV_IN    : boolean  := true;
      W             : width_typ := (10,12,14,16,17)
   );
```

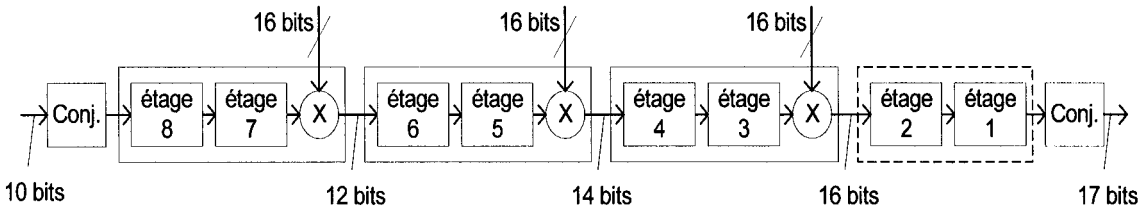


Figure 19 Architecture simplifiée d'une IFFT de 256 points, avec la largeur des bus de donnée croissante

Pour l'assignation de `W`, le type `width_typ` doit être défini correctement dans le *package define_width.vhd.*

```
PACKAGE define_width IS
   type width_typ is array (X downto 0) of integer;
END define_width;
```

La variable **X** correspond au nombre de bus de données pouvant être ajustés. Par exemple, pour une FFT/IFFT de 256 points, $v = 8$, il y a quatre groupes de deux étages d'instanciés pour créer le module, donc on trouve trois bus internes plus le bus d'entrée et de bus de sortie. Le total est de cinq bus configurables, comme illustré par la figure 19. La variable **X** du *package* doit être fixée à quatre `(4 downto 0)` dans le cas de v=8. En général, la variable **X** prend la valeur de l'entier supérieur du calcul $\lceil v/2 \rceil$.

3.3.2 Modules intermédiaires

Le niveau intermédiaire contient deux modules configurables et sérialisables formant les étages de la transformée. Le premier module *two_stages* contient un groupe de deux étages suivi ou non d'un multiplicateur complexe et un contrôleur. Pour des transformées avec v pair, N puissance de 4, seul ce module est nécessaire. Le groupe de deux étages placé en sortie de l'architecture ne contient pas le multiplicateur. Les rectangles en tireté des figures 15 et 19 correspondent au module *two_stages*.

Le second module, *one_stage,* contient un étage avec un contrôleur et peut aussi posséder un multiplicateur complexe. Pour construire le R2^{2}PC, il est utilisé en combinaison avec le module *two_stages* dans le cas où *v* est impair. On le retrouve accompagné du multiplicateur à l'entrée de l'architecture pour les transformées ordonnées et sans multiplicateur en sortie pour les transformées en ordre des bits renversé.

L'architecture VHDL qui les décrit les deux modules en est une de type structurelle. Le code de ceux-ci est très semblable. Les blocs du niveau hiérarchique inférieur, appelés sous-modules, sont interconnectés et paramétrés selon leur position et leurs configurations. Ces configurations sont calculées au niveau supérieur. Par exemple, il n'y aura pas de multiplicateur complexe si l'étage est le dernier de la transformée. Ou encore, la taille du délai en rétroaction sera fixée en fonction du type d'ordonnancement et de la position de cette ligne à délai.

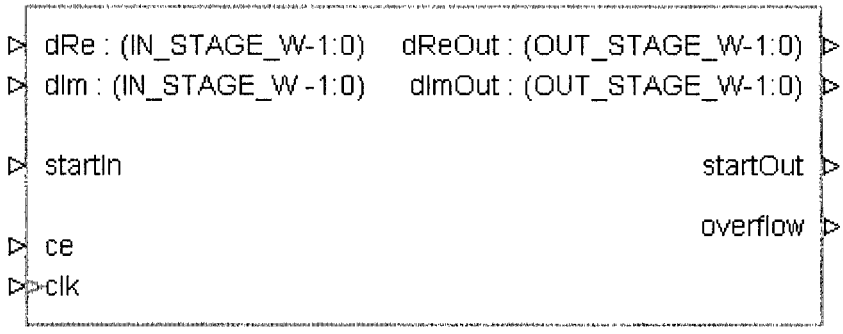


Figure 20 Symbole du module *two_stages*

Fichier: *two_stages.vhd*

Generics:

IN_STAGE_W	positive	Largeur binaire de l'entrée du bloc.
OUT_STAGE_W	positive	Largeur binaire de la sortie du bloc.
TF_W	positive	Largeur binaire des facteurs de phase.

ADDR_STAGE	positive	Valeur calculée par une fonction à partir de la position des étages dans la transformée et de l'ordonnancement. Elle permet d'assigner la largeur du compteur dans le contrôleur et la largeur du bus d'adresse pour la table des facteurs de phase. Si l'ordonnancement est naturel, elle correspond au plus élevé des deux numéros d'étage. Si l'ordre est renversé, c'est une valeur paire partant à quatre pour le double étage d'entrée et augmentant par pas de deux pour chaque double étage suivant. Cette progression est causée par l'algorithme DIT employé dans ce cas.
LATENCY	positive	Latence nécessaire en cycle entre l'arrivée d'une donnée en entrée du bloc et la sortie du résultat correspondant. Ce paramètre est utilisé par le contrôleur. On compte un cycle pour chaque papillon plus la latence du multiplicateur complexe, typiquement quatre cycles, s'il est présent. S'il n'est pas présent, il est remplacé par un bloc d'arrondi qui demande un cycle. La latence résultante est donc de 3 ou 6 cycles.
DELAY_B	positive	Taille du délai en rétroaction pour le deuxième étage, l'étage B. Cette valeur est issue d'un calcul fait par une fonction.
BIT_REV_IN	boolean	Ordonnancement de l'entrée, vrai pour l'ordre des bits renversé et faux pour l'entrée ordonnée.
S_A	integer	Mise à l'échelle du premier étage, l'étage A. La valeur "1" correspond à une division par deux et la valeur "0" indique qu'aucune mise à l'échelle ne doit être faite.
S_B	integer	Mise à l'échelle du deuxième étage, l'étage B.
ROM_ATTRIB	string	Type d'implémentation de la table des facteurs de phase soit "*select_rom*" pour de la mémoire ROM distribuée dans les slices, soit "*block_rom*" pour l'utilisation d'un Block Ram ou soit "*logic*" pour la conversion en slices de l'équation logique unissant les adresses et les données.
MULT_STYLE	string	Type de ressources pour l'implémentation des multiplicateurs: "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.

MULT_ROUND_W	positive	Largeur binaire du bus de données précédent les multiplicateurs complexes, ce paramètre est fixé à 18 dans le cas d'un multiplicateur dédié de Xilinx.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Remise à zéro du contrôleur et des pointeurs si nécessaire.
dRe	Partie réelle du vecteur d'entrée.
dIm	Partie imaginaire du vecteur d'entrée.

Sorties:

startOut	Indique la fin de la latence suite à un signal *startIn.*
overflow	Indique un débordement dans les calculs et donc une séquence non valide.
dReOut	Partie réelle du vecteur de sortie.
dImOut	Partie imaginaire du vecteur de sortie.

Fichier: *one_stage.vhd*

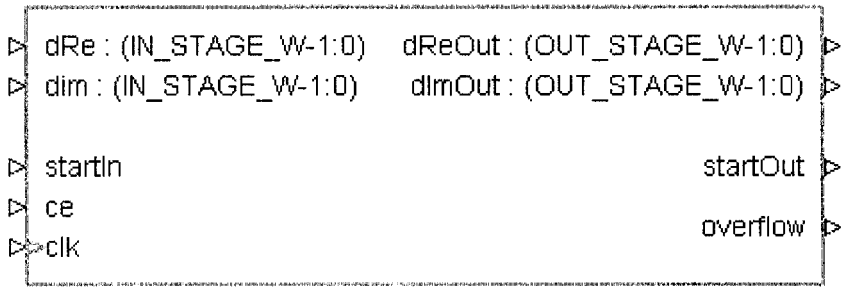


Figure 21 Symbole du module *one_stage*

Generics:

IN_STAGE_W	positive	Largeur binaire de l'entrée du bloc.
OUT_STAGE_W	positive	Largeur binaire de la sortie du bloc.

TF_W	positive	Largeur binaire des facteurs de phase.
ADDR_STAGE	positive	Valeur calculée par une fonction à partir de la position de l'étages dans la transformée et de l'ordonnancement. Elle correspond au numéro de l'étage, valeur impaire. Elle permet d'assigner la largeur du compteur dans le contrôleur et la largeur du bus d'adresse pour la table des facteurs de phase, dans le cas de l'entrée ordonnée.
LATENCY	positive	Latence nécessaire en cycle entre l'arrivée d'une donnée en entrée du bloc et la sortie du résultat correspondant. Ce paramètre est utilisé par le contrôleur. On compte un cycle pour le papillon plus la latence du multiplicateur complexe, typiquement de quatre cycles, s'il est présent. S'il n'est pas présent, il est remplacé par un bloc d'arrondi qui demande un cycle. La latence résultante est donc de 2 ou 5 cycles.
DELAY_B	positive	Taille du délai en rétroaction pour l'étage. Cette valeur est issue d'un calcul fait par une fonction.
BIT_REV_IN	boolean	Ordonnancement de l'entrée: vrai pour l'ordre des bits renversée et faux pour l'entrée ordonnée.
S	integer	Mise à l'échelle de étage: la valeur "1" correspond à une division par deux et la valeur "0" indique qu'aucune mise à l'échelle ne doit être faite.
ROM_ATTRIB	string	Type d'implémentation de la table des facteurs de phase soit "*select_rom*" pour de la mémoire ROM distribuée dans les slices, soit "*block_rom*" pour l'utilisation d'un BRAM ou soit "*logic*" pour la conversion en slices de l'équation logique unissant les adresses et les données.
MULT_STYLE	string	Type de ressource pour l'implémentation des multiplicateurs: "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.
MULT_ROUND_W	positive	Largeur binaire du bus de données précédent les multiplicateurs complexes. Ce paramètre est fixé à 18 dans le cas d'un multiplicateur dédié de Xilinx.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*

`startIn`	Remise à zéro du contrôleur et des pointeurs si nécessaire.
`dRe`	Partie réelle du vecteur d'entrée.
`dIm`	Partie imaginaire du vecteur d'entrée.

Sorties:

`startOut`	Indique la fin de la latence suite à un signal *startIn.*
`overflow`	Indique un débordement dans les calculs et donc une séquence non valide.
`dReOut`	Partie réelle du vecteur de sortie.
`dImOut`	Partie imaginaire du vecteur de sortie.

3.3.3 Sous-modules

3.3.3.1 Papillon radix-2

Ce bloc correspond à un papillon radix-2 complexe avec deux modes de fonctionnement : le mode de transmission et le mode de traitement. Il contient quatre additionneurs et/ou soustracteurs permettant de faire l'arithmétique nécessaire aux deux branches d'un papillon, comme illustré par la figure 22. Selon le mode de fonctionnement requis, un signal de sélection `ctrlMux` connecté au multiplexeur de sortie permet de choisir entre transmettre les données d'entrée inchangées ou le résultat du calcul. Ce multiplexeur fonctionne avec l'horloge et son signal d'activation `ce`. Un cycle de latence est introduit par le sous-module papillon.

Pour les deuxièmes papillons des groupes de deux étages, un mécanisme permet de faire la multiplication par -j de la branche provenant de l'étage précédent. Cette opération se fait en inversant les parties réelle et imaginaire et en changeant le signe de l'addition et de la soustraction pour la partie imaginaire. Il est facile de modifier un additionneur en soustracteur, puisqu'en représentation complément à deux, un soustracteur consiste en en additionneur précédé d'inverseurs sur l'entrée à soustraire et dont le signal de retenue

vaut "1" ($a-b=a+\bar{b}+1$). Seulement les papillons concernés par cette multiplication triviale occasionnelle de -j implémenteront cette fonctionnalité lors de la synthèse.

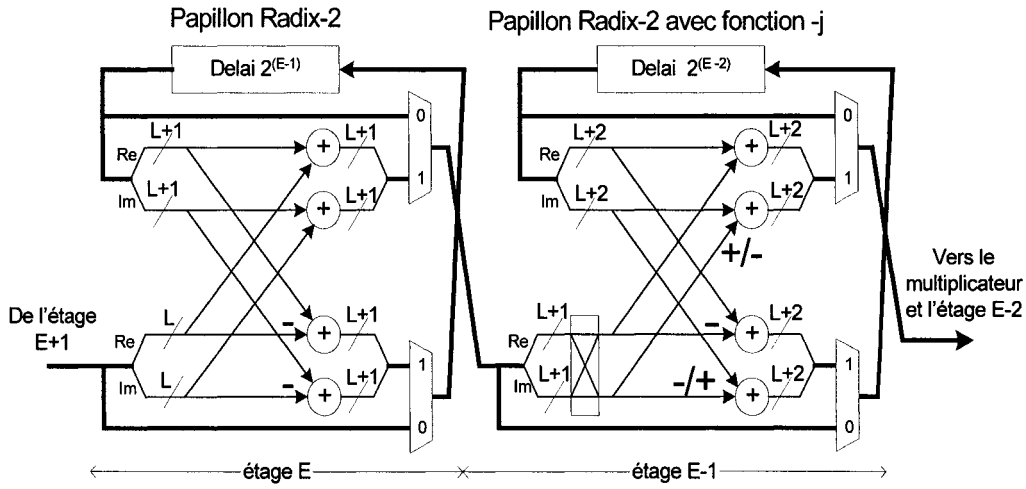


Figure 22 Structure interne des papillons de deux étages consécutifs avec une mise à l'échelle

Une mise à l'échelle pour éviter les débordements est faite dans ce sous-module. Si la mise à l'échelle requise est inférieure à la valeur de *N,* les derniers papillons en sortie n'implémenteront pas de mise à l'échelle. Elle se fait progressivement à raison d'une division par deux par papillon, et ce, de l'entrée vers la sortie du R2^{2}PC. Le paramètre S indique qu'une mise à l'échelle doit être faite. Ainsi, lorsqu'il prend la valeur "1", une division par deux sera effectuée. Lorsqu'elle est implémentée, on retrouve une extension de signe de un bit (MSB) des mots binaires provenant de l'étage précédent. Les additionneurs ainsi que les sorties des étages sont ajustés à ces nouvelles dimensions. Pour deux étages avec la mise à l'échelle, on passera de L bits à L+1 bits puis à L+2 bits comme illustré par la figure 22. La réduction de la largeur des mots binaires se produit uniquement dans le multiplicateur complexe. Comme un papillon qui implémente la mise à l'échelle ne peut pas faire de débordement, il ne possède pas de mécanisme de détection du débordement. En revanche, un débordement peut se produire et ne pas être détecté dans un papillon sans mécanisme de mise à l'échelle.

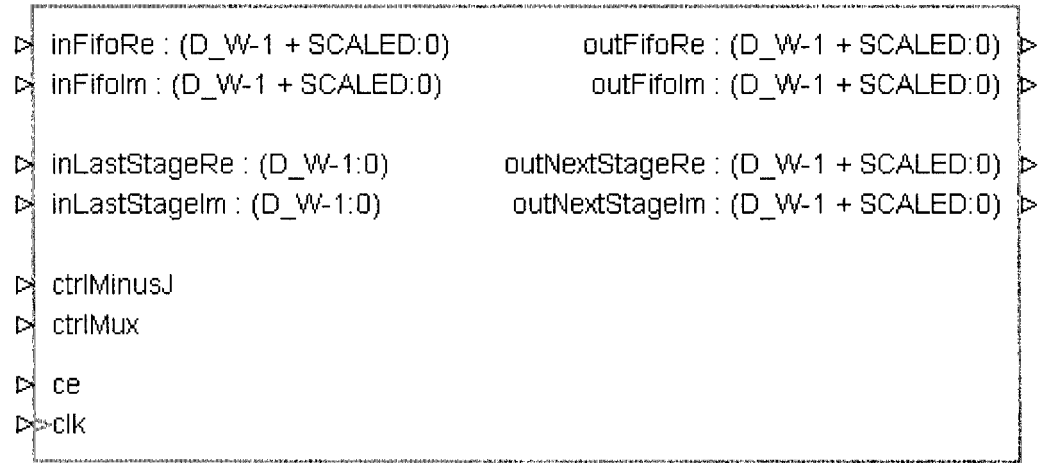


Figure 23 Symbole du sous-module papillon

Fichier: *bfly_r2.vhd*

Generics:

D_W	positive	Largeur binaire de l'entrée du bloc.
SCALED	integer	Mise à l'échelle de l'étage: la valeur "1" correspond à une division par deux et la valeur "0" indique qu'aucune mise à l'échelle ne doit être faite.
EN_MINUS_J	boolean	Activation de la fonctionnalité -j pour le papillon: "1" indique que -j doit être implémenté et la valeur "0" indique que cette fonctionnalité ne doit pas être présente.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
ctrlMux	Signal de contrôle du multiplexeur de sortie, permet de commuter entre les deux modes de fonctionnement du papillon.

`ctrlMinusJ`	Signal de contrôle activant l'opération de -j sur l'entrée provenant de l'étage précédent.
`inLastStageRe`	Partie réelle du vecteur d'entrée provenant de l'étage précédent.
`inLastStageIm`	Partie imaginaire du vecteur d'entrée provenant de l'étage précédent.
`inFifoRe`	Partie réelle du vecteur d'entrée provenant du délai en rétroaction.
`inFifoIm`	Partie imaginaire du vecteur d'entrée provenant du délai en rétroaction.

Sorties:

`outFifoRe`	Partie réelle du vecteur de sortie vers le délai en rétroaction.
`outFifoIm`	Partie imaginaire du vecteur de sortie vers le délai en rétroaction.
`outNextStageRe`	Partie réelle du vecteur de sortie vers l'étage suivant.
`outNextStageIm`	Partie imaginaire du vecteur de sortie vers l'étage suivant.

3.3.3.2 Délai en rétroaction

Un délai en rétroaction accompagne chaque papillon pour former un étage. La taille de cette ligne à délai dépend de l'étage où il se trouve et de l'ordonnancement. La donnée d'entrée, provenant d'une sortie du papillon, est mémorisée pour un nombre de cycles correspondant à la taille du délai et ressort pour entrer de nouveau dans le papillon, d'où la rétroaction. Ce délai peut être implémenté avec des registres à décalage pour des délais jusqu'à 32 cycles ou à l'aide de mémoire (BRAM) agissant en FIFO pour les plus grandes tailles. Dans le cas de l'utilisation d'une mémoire, un pointeur adresse une case mémoire où l'ancienne valeur est lue en même temps que le nouveau contenu y est écrit dans le même cycle. À chaque cycle le pointeur est incrémenté.

Fichier: *stage_fifo.vhd*

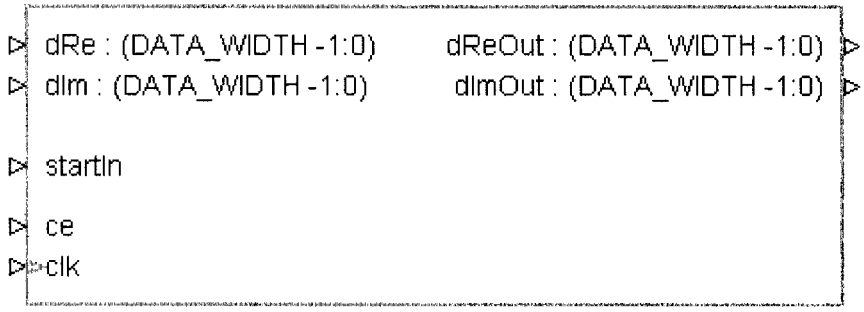


Figure 24 Symbole du sous-module de délai en rétroaction

Generics:

DATA_WIDTH	positive	Largeur binaire de l'entrée et de la sortie du bloc.
FIFO_SIZE	positive	Taille du délai (puissance de deux).

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Remise à zéro du pointeur si le délai est implémenté sous forme de mémoire FIFO, sinon ce signal est inutilisé.
dRe	Partie réelle du vecteur d'entrée provenant du papillon.
dIm	Partie imaginaire du vecteur d'entrée provenant du papillon.

Sorties:

dReOut	Partie réelle du vecteur de sortie vers le papillon.
dImOut	Partie imaginaire du vecteur de sortie vers le papillon.

3.3.3.3 Multiplicateur complexe

Le multiplicateur complexe effectue la rotation de phase des données. Il fait une multiplication complexe avec les données d'entrée et les facteurs de phase (W^{nk}) qui proviennent d'une table. Le multiplicateur illustré par la figure 26 est constitué de quatre

multiplicateurs réels et de deux additionneurs réels séparés par des registres de pipeline pour augmenter la vitesse d'opération. On retrouve un registre avant l'étage de multiplication, un dans chaque multiplicateur (dans le cas des multiplicateurs dédiés de Xilinx, BMULT), un après les multiplications et un dernier suite aux additionneurs. Le multiplicateur complexe introduit donc une latence de quatre cycles. Bien que ce pipelinage s'avère adéquat, des registres supplémentaires peuvent aussi être ajoutés.

Pour les plates-formes Virtex-II et Virtex-II Pro de Xilinx, il peut être avantageux d'utiliser les BMULT qui ont une taille maximale d'entrée de 18 bits par 18 bits et qui peuvent contenir un étage de pipeline. En revanche, pour pouvoir utiliser un seul multiplicateur par multiplication, la taille des entrées doit absolument être limitée à 18 bits. Conséquemment, on retrouve à l'entrée des données un bloc pour faire l'arrondi et limiter la largeur des mots binaires. Il est paramétré par R_W qui est en général fixé à 18. Si la donnée est moins large que 18 bits, l'arrondissement n'est pas requis et il n'est donc pas implémenté. Dans ce dernier cas, la donnée est acheminée au multiplicateur dans la partie la moins significative des 18 bits et une extension du signe est effectuée.

Plus du quart des multiplications sont effectuées avec W^0 qui correspond à la valeur +1+0j. Cette valeur équivaut à ne faire aucun traitement sur la donnée. Faire transiter les données qui doivent rester inchangées pose un problème. En effet, une multiplication exacte par un n'est pas possible. Vu la représentation des nombres en complément à deux où la plus grande valeur représentable est $1-2^{-(TF\_W+1)}$, avec `TF_W` la largeur binaire de la partie réelle ou imaginaire du facteur de phase. En effectuant les multiplications par W^0 avec cette valeur, une erreur systématique serait introduite dans plus du quart des points. Pour contrer cet effet indésirable, on aurait pu faire un chemin de données contenant quatre étages de registres et contournant le multiplicateur dans le cas des multiplications par W^0. Ce chemin devrait être choisi à l'aide de multiplexeurs, ce qui implique plus de ressources et un contrôle plus complexe. Au lieu de cela, on fera transiter toutes les données par le multiplicateur complexe. Pour éviter l'erreur systématique, on peut

ajouter un bit supplémentaire en position MSB à tous les facteurs de phase. La plage dynamique devient alors]2,-2] au lieu de]1,-1] et la valeur +1+0j devient maintenant représentable. Encore une fois, cette solution n'a pas été retenue, puisqu'elle nécessite une augmentation non négligeable des ressources de mémoire pour les facteurs de phase.

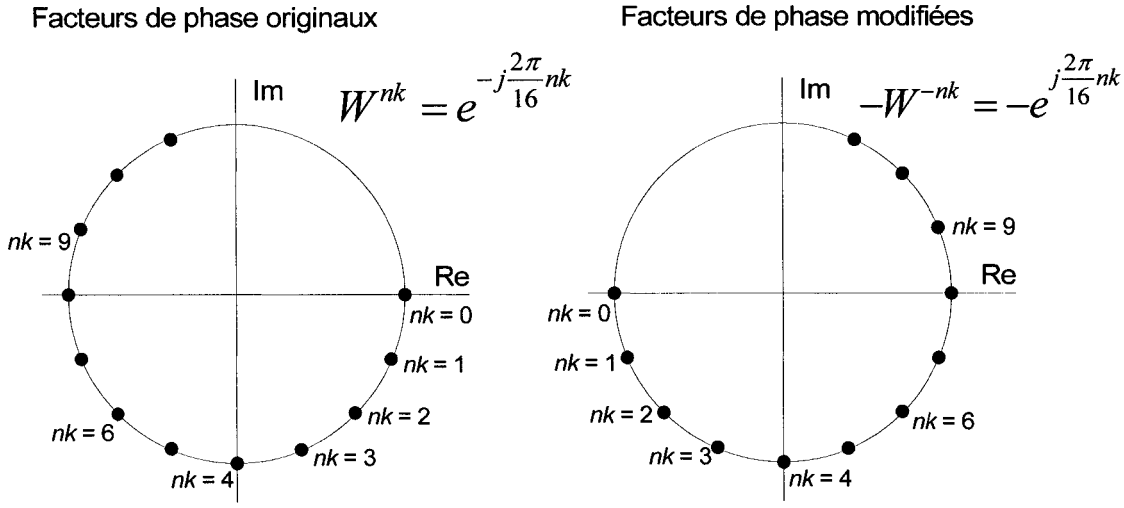


Figure 25 Modification des facteurs de phase pour N=16

La méthode qui a été retenue pour éviter l'erreur systématique est inspirée de (Elterich & Stammler, 1988). Celle-ci porte sur une architecture radix-2, mais peut facilement être adaptée à notre architecture. Pour ce faire, les facteurs de phase W^{nk} sont remplacés par leur conjugué complexe négatif, $-W^{-nk}$ (complément à deux de la partie réelle). On prend aussi le conjugué de la donnée, soit le complément à deux de la partie imaginaire. Conséquemment, les multiplications par le facteur de phase +1 deviennent des multiplications par -1, représentables en complément à deux. Ainsi, pour passer d'un facteur de phase à l'autre, au lieu de faire une rotation dans le sens horaire sur le cercle unitaire comme c'est le cas pour l'algorithme de base, les rotations se font maintenant dans le sens anti-horaire, comme illustré par la figure 25. Pour obtenir un résultat valide, l'architecture traditionnelle du multiplicateur complexe a légèrement dû être modifiée. En effet, un multiplicateur complexe est constitué de quatre produits *AC, AD, BC* et *BD*

qu'on assemble selon l'équation 3.2. Les quatre entrées du multiplicateur étant habituellement *A, B, C, D.*

$$(A+Bj)(C+Dj)=(AC-BD)+(AD+BC)\,j \tag{3.2}$$

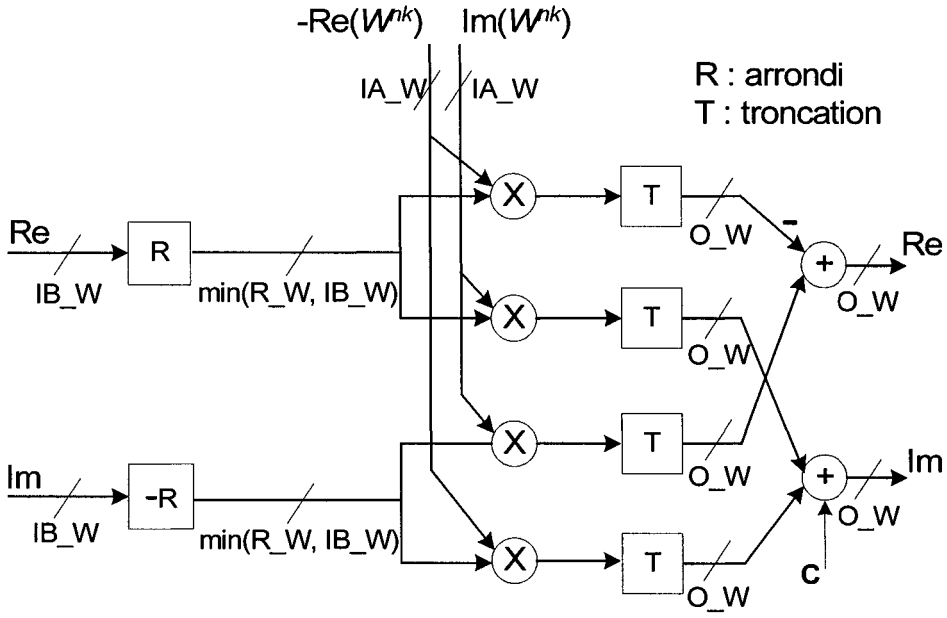


Figure 26 Schéma du multiplicateur complexe avec largeur des bus

Dans le cas qui nous concerne, les facteurs de phase deviennent $-W^{-nk}$ soit -Re(W^{nk}) + Im(W^{nk}). Reprenons l'exemple précédent, mais cette fois-ci avec *-A, +B, +C* et *-D* de disponible. On calcul d'abord les produits et on obtient *-AC*, *AD*, *BC* et *-BD*. Ensuite, on les assemble avec les additionneurs/soustracteurs pour obtenir ((*-BD*) - (*-AC*)) et (*AD* + *BC*). On obtient ainsi le même résultat que pour le multiplicateur complexe traditionnel. Pour l'implémentation, les facteurs de phase mémorisés dans les tables sous forme $-W^{-nk}$ ne prennent pas de mémoire supplémentaire, ni de fonction d'inversion. Pour ce qui du calcul du conjugué complexe, il est effectué entre le bloc d'arrondi et l'étage de registre à l'entrée du multiplicateur. Le multiplicateur complexe modifié est illustré par la figure 26.

Le contrôle de la largeur binaire se produit dans le multiplicateur complexe. La figure 26 illustre la progression des largeurs des mots. La multiplication du facteur de phase sur `IA_W` bits par la donnée sur `IB_W` bits donne un résultat sur `IA_W+IB_W` bits, pour un `IB_W` inférieur ou égal à `R_W`, largeur de l'arrondi. Si les résultats étaient conservés à cette largeur, trop de ressources seraient utilisées, on doit donc procéder à une réduction de la largeur binaire. Pour ce faire, la troncation à `O_W` bits du résultat de chaque multiplication est effectuée. Cette troncation introduit un biais avec une moyenne de $-2^{-(O\_W)}$ et après l'addition de deux nombres biaisés, la moyenne passe à $-2^{-(O\_W-1)}$. Pour compenser cette moyenne non nulle, le bit de retenue dans l'additionneur est fixé à "1" dans le cas où l'une des opérandes aurait dû être arrondi à la hausse. La retenue est générée par un OU logique entre les deux 1^{er} bits qui ont été tronqués suite à la multiplication d'une partie imaginaire avec une partie réelle. Ces bits se trouvent en position `IA_W + IB_W - O_W - 2` à la sortie des multiplicateurs. Dans le cas de la soustraction, la moyenne du biais est annulée dans le calcul. Si un débordement se produit dans l'additionneur ou le soustracteur, il sera détecté et le signal `overflow` s'activera. Ce signal indique que tous les résultats de la transformée courante sont invalides. L'augmentation de la mise à l'échelle permet d'éviter les débordements pour un type de signal donné, en revanche pour les mêmes tailles binaires la précision des résultats se trouve réduite. Il y a un compromis à faire quant à l'ajustement de la mise à l'échelle versus la probabilité de débordement.

Fichier: *mult_complex_rxt_conjb.vhd*

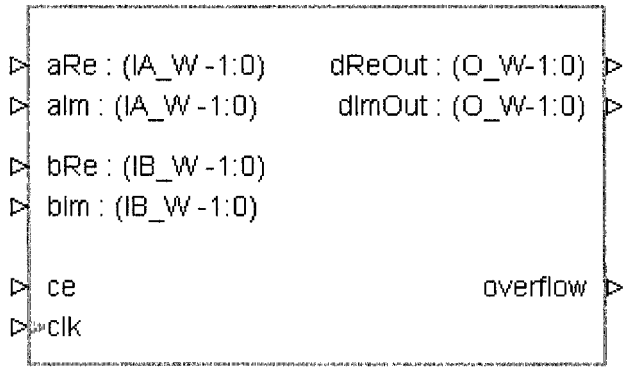


Figure 27 Symbole du multiplicateur complexe

Generics:

IA_W	positive	Largeur binaire de l'entrée A du bloc, entrée des facteurs de phase.
	positive	Largeur binaire de l'entrée B du bloc, entrée des données.
O_W	positive	Largeur binaire de la sortie des données.
R_W	positive	Largeur binaire du bus de données précédent les multiplicateurs complexes, paramètre fixé à 18 dans le cas d'un BMULT de Xilinx.
STYLE	string	Type de ressource pour l'implémentation des multiplicateurs: "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.
PIPELINE_IN	positive	Nombre de registres précédant les multiplicateurs.
PIPELINE_OUT	positive	Nombre de registres suivant les multiplicateurs, avant les additionneurs.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
aRe	Partie réelle du facteur de phase.
aIm	Partie imaginaire du facteur de phase.
bRe	Partie réelle du vecteur de données provenant d'un papillon.

`bIm`	Partie imaginaire du vecteur de données provenant d'un papillon.

Sorties:

`dReOut`	Partie réelle du vecteur de sortie.
`dImOut`	Partie imaginaire du vecteur de sortie.
`overflow`	Indique un débordement dans les calculs.

3.3.3.4 Table de facteurs de phase

Les facteurs de phase sont contenus dans une table générée par un script Matlab. Le script produit du code VHDL selon la taille de la table et la largeur des mots binaires. Trois types d'implémentation sont possibles selon la taille de la mémoire. Pour les mémoires de grande taille (64 mots complexes ou plus), on utilise les ressources RAM dédiées (BRAM) employées en mémoire de type ROM. Pour les tables de plus petite taille, les facteurs de phase peuvent être contenus dans les *slices* de deux façons, soit assemblés en mémoire ROM ou soit reliés en fonctions logiques unissant l'adresse aux données. Peu importe le type d'implémentation, le facteur de phase est présent sur le bus de donnée au cycle d'horloge suivant la lecture de l'adresse. Les facteurs de phase contiennent une certaine redondance puisqu'ils représentent les coordonnées des points sur le cercle unitaire. Dans une table, les mêmes facteurs de phase peuvent revenir à plusieurs reprises, ainsi que leur inverse et leur conjugué. Des techniques ont été mises au point pour éliminer l'information superflue et donc minimiser la taille des facteurs de phase (Hasan & Arslan, 2002b). Ces techniques ne sont pas utilisées dans le R2^{2}PC. D'une part, elles demandent l'ajout de logique de calcul à chaque table de facteurs de phase et requièrent des contrôleurs plus complexes. D'autre part pour les grandes tables où le gain en ressource serait le plus substantiel, même avec l'emploi de techniques éliminant la redondance l'usage de BRAM serait encore nécessaire.

Fichier: *coeff_gen.vhd*

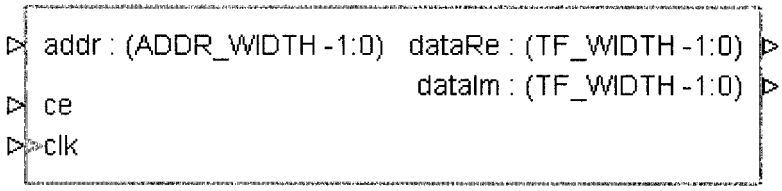


Figure 28 Symbole du sous-module de table des facteurs de phase

Generics:

`ADDR_WIDTH`	positive	Largeur de l'adresse pour atteindre un facteur de la table, correspond à l'entier supérieur de $\log_2$(nombre de facteur de phase).
`TF_WIDTH`	positive	Largeur binaire des facteurs de phase.
`ROM_ATTRIBUTE`	string	Type d'implémentation de la table soit "*select_rom*" pour de la mémoire ROM distribuée dans les slices, soit "*block_rom*" pour l'utilisation d'un BRAM ou soit "*logic*" pour la conversion en slices de l'équation logique unissant l'adresse et les données.

Entrées:

`clk`	Horloge.
`ce`	Signal d'activation de l'horloge *clock enable.*
`addr`	Adresse pour atteindre un facteur de phase.

Sorties:

`dataRe`	Partie réelle du facteur de phase.
`dataIm`	Partie imaginaire du facteur de phase.

3.3.3.5 Arrondi complexe

Le sous-module d'arrondi complexe se situe à la fin de toute transformée pour ajuster la largeur binaire des mots en sortie. Il remplace le multiplicateur complexe et contribue à un cycle de latence. Si la largeur de sortie requise `O_W` est la même que la largeur de son entrée `I_W`, il n'y a pas d'arrondi à effectuer. Par contre, si `O_W` est plus petit que `I_W`, l'arrondi complexe est fait en additionnant le bit en position `I_W-O_W-1` avec le vecteur d'entré tronqué après le bit en position `I_W-O_W`. Si un débordement se produit, le signal `overflow` devient actif.

Fichier: *round_complex.vhd*

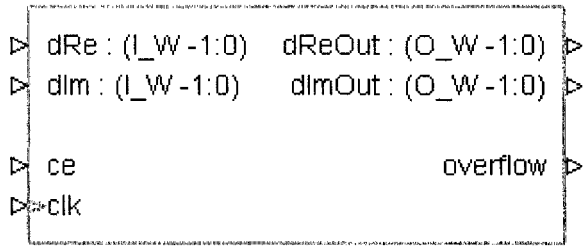


Figure 29 Symbole du sous-module d'arrondi

Generics:

`I_W`	positive	Largeur binaire de l'entrée du bloc.
`O_W`	positive	Largeur binaire de la sortie du bloc.

Entrées:

`clk`	Horloge
`ce`	Signal d'activation de l'horloge *clock enable.*
`dRe`	Partie réelle du vecteur d'entrée.
`dIm`	Partie imaginaire du vecteur d'entrée.

Sorties:

`dReOut`	Partie réelle du vecteur de sortie.
`dImOut`	Partie imaginaire du vecteur de sortie.
`overflow`	Indique un débordement dans les calculs.

3.3.3.6 Conjugué complexe

Le sous-module de conjugué complexe précède et suit toute IFFT et n'est pas présent dans le cas d'une FFT. Il fait l'inversion du signe de la partie imaginaire et contient donc des inverseurs suivis d'un additionneur. Ce bloc contribue à un cycle de latence en raison de la présence de bascules de pipeline.

Fichier: *complex_conjugate.vhd*

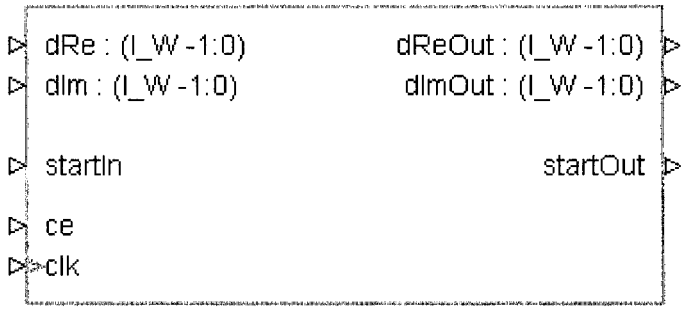


Figure 30 Symbole du sous_module de conjugué complexe

Generics:

`I_W`	positive	Largeur binaire de l'entrée du bloc.
`O_W`	positive	Largeur binaire de la sortie du bloc.

Entrées:

`clk`	Horloge
`ce`	Signal d'activation de l'horloge *clock enable.*
`dRe`	Partie réelle du vecteur d'entrée.
`dIm`	Partie imaginaire du vecteur d'entrée.

Sorties:

`dReOut`	Partie réelle du vecteur de sortie.
`dImOut`	Partie imaginaire du vecteur de sortie.
`overflow`	Indique un débordement dans les calculs.

3.3.3.7 Contrôleurs

Le R2^{2}PC ne possède pas un contrôleur global, mais plutôt plusieurs contrôleurs locaux qui échangent de l'information de synchronisation entre eux. On retrouve deux types de contrôleurs, un pour les modules de deux étages (*two_stages*) et un pour l'étage simple (*one_stage*). Les deux types de contrôleurs sont très semblables et sont principalement composés d'un compteur qui est remis à zéro par le signal `strartIn`. La largeur de ce compteur correspond à la largeur de l'adresse de la table de facteurs de phase contrôlée, puisque la valeur du compte sert d'adresse. Il incrémente à chaque cycle d'horloge où le signal `ce` est actif. Le compteur contrôle aussi les signaux de sélection des multiplexeurs commutant le mode de fonctionnement du ou des papillons et le signal permettant l'opération -j. Par exemple, dans le cas d'une FFT en ordre naturel, le multiplicateur suivant les étages v et v-1 a besoin de N facteurs de phase, donc d'un contrôleur avec un compteur de v bits. Le papillon de l'étage v doit changer de mode de fonctionnement à tous les 2^{v-1} cycles, la fréquence de commutation du bit le plus significatif du contrôleur. Des tampons permettent de synchroniser les signaux de contrôle avec leur destination. On retrouve aussi une ligne à délai équivalente à la latence des modules contrôlés pour activer le contrôleur suivant par le signal `startOut`.

Fichier: *stages_controller.vhd*

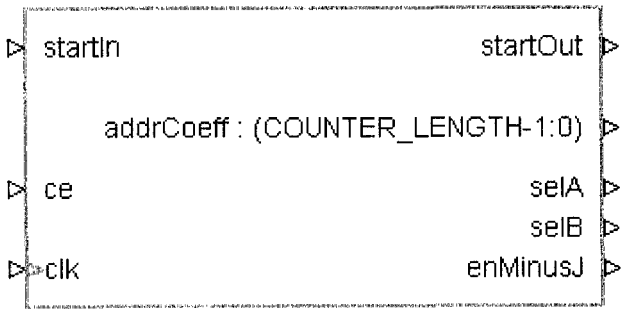


Figure 31 Symbole du sous-module contrôleur de deux étages

Generics:

COUNTER_LENGTH	positive	Largeur binaire du contrôleur, correspond à l'adresse de la table de facteurs de phase contrôlée.
CONTROLLER_LATENCY	positive	Latence requise pour la progression des données au travers des blocs contrôlés.
BIT_REV_IN	boolean	Ordonnancement de l'entrée: vrai pour l'ordre des bits renversé et faux pour l'entrée ordonnée.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Signal de remise à zéro du contrôleur.

Sorties:

startOut	Signal de remise à zéro des étages suivants.
addrCoeff	Adresse de la table de facteur de phase contrôlée.
enMinusJ	Signal contrôlant l'opération -j pour le papillon B: "1" correspond à l'opération -j et "0" à ne pas effectuer l'opération.

selA	Signal de sélection du mode de fonctionnement du papillon A: "1" correspond au mode traitement mathématique et "0" à laisser passer les données inchangées.
selB	Signal de sélection du mode de fonctionnement du papillon B: "1" correspond au mode traitement mathématique et "0" à laisser passer les données inchangées.

Fichier: *odd_stage_controller.vhd*

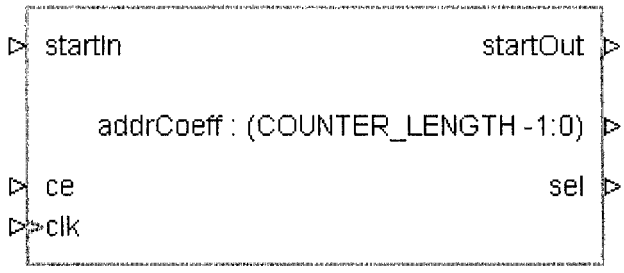


Figure 32 Symbole du sous-module contrôleur d'un seul étage

Generics:

COUNTER_LENGTH	positive	Largeur binaire du contrôleur, correspond à l'adresse de la table de facteurs de phase contrôlée.
CONTROLLER_LATENCY	positive	Latence requise pour la progression des données au travers des blocs contrôlés.
BIT_REV_IN	boolean	Ordonnancement de l'entrée, vrai pour l'ordre des bits renversé et faux pour l'entrée ordonnée.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Signal de remise à zéro du contrôleur.

Sorties:

`startOut`	Signal de remise à zéro des étages suivants.
`addrCoeff`	Adresse de la table de facteur de phase contrôlée.
`sel`	Signal de sélection du mode de fonctionnement du papillon de l'étage contrôlé: "1" correspond au mode de traitement mathématique et "0" à laisser passer les données inchangées.

3.4 Conclusion

En bref, une approche hiérarchique sur trois niveaux a permis de construire le module de FFT reconfigurable R2^{2}PC. Les sous-modules du niveau inférieur ont été conçus avec un souci de la précision des résultats et du maximum de paramétrisation. La configuration du module se fait au niveau supérieur du code VHDL par l'intermédiaire de *generics* et les paramètre se propagent jusqu'aux sous-modules. Les résultats obtenus suite à la simulation du code sont présentés au chapitre suivants.

CHAPITRE 4

CARACTÉRISATION DU BRUIT DE QUANTIFICATION ET DES RESSOURCES

Voyons maintenant les performances du R2^{2}PC programmé en VHDL synthétisable. Ce chapitre présente d'abord la définition des métriques permettant de caractériser le module de FFT. Puis, la structure des bancs d'essai ainsi que les outils de simulation, de synthèse, de placement et de routage sont exposés. Par la suite, on retrouve des sections illustrant les effets de la variation d'un paramètre de configuration. Certains cas de vérification permettent de faire ressortir les influences de paramètres combinées sur la précision des résultats et la complexité du design.

4.1 Méthodologie de vérification

Le R2^{2}PC a été caractérisé selon deux aspects principaux soit la précision des résultats de la transformée et les ressources nécessaires à sa mise en oeuvre dans une puce programmable.

4.1.1 Étude du bruit de quantification

La métrique employée pour caractériser le bruit de quantification est le rapport de puissance entre le signal et le bruit de quantification (SQNR pour *Signal to Quantization Noise Ratio*). Le SQNR utilise les signaux temporels et il est habituellement exprimé en décibel.

4.1.1.1 Structure de l'environnement de vérification

La structure de l'environnement de vérification pour déterminer le SQNR est illustrée par la figure 33. Les logiciels Modelsim (version se 5.7e) pour la simulation et Matlab pour la génération des vecteurs de vérification et pour l'analyse des résultats ont été employés.

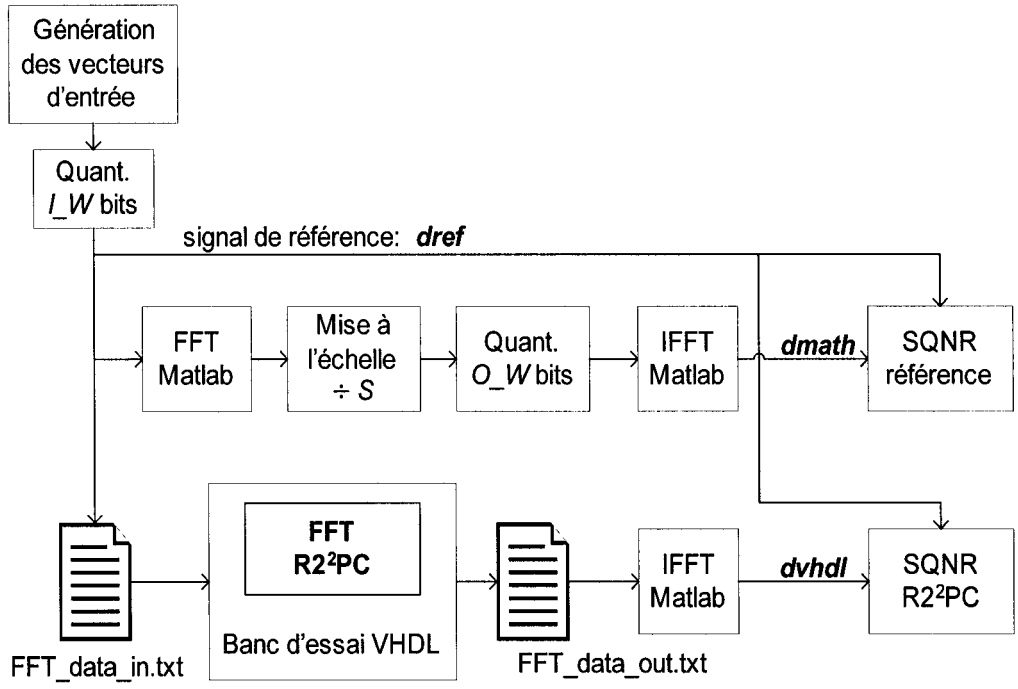


Figure 33 Structure du banc d'essai

Les étapes permettant d'obtenir le SQNR sont les suivantes :

- Génération des vecteurs de vérification par un script Matlab. La séquence d'échantillons d'entrée *dref* est inscrite dans un fichier texte (`FFT_data_in.txt`) sous forme d'entiers signés respectant la plage dynamique de représentation des nombres sur *I_W* bits. La nature des vecteurs d'entrée peut être de deux types:

- Type I: une sinusoïde pour les parties réelles et imaginaire dont les fréquences distinctes sont choisies aléatoirement pour chaque vecteur. On s'attend à un spectre complexe ayant deux raies principales.
- Type II: un vecteur de données aléatoires complexes suivant une distribution uniforme centrée autour de zéro. Le spectre de ce signal possède de l'énergie en chaque point.

Le nombre de vecteurs pour chaque type est un paramètre à sélectionner.

- Simulation du R2^{2}PC par un banc d'essai en VHDL avec l'utilisation de Modelsim. Le banc d'essai VHDL lit le fichier d'entrée, impose les entrées au R2^{2}PC, recueille ses sorties et inscrit les résultats dans un autre fichier texte (`FFT_data_out.txt`).
- Analyse des résultats par un script Matlab. Le signal de référence temporel *dref* correspond aux vecteurs d'entrée quantifiés sur *I_W* bits. Le SQNR de référence est obtenu en appliquant une FFT logicielle en point flottant au signal de référence, puis en appliquant une mise à l'échelle et une quantification sur *O_W* bits au résultat en fréquence afin de simuler la représentation point fixe. Ensuite, ce signal est ramené dans le domaine temporel par une IFFT logicielle. On obtient ainsi le signal *dmath* et le calcul de l'équation 4.1 est effectué. Le résultat de la simulation VHDL est aussi ramené dans le domaine du temps par une IFFT mathématique, signal *dvhdl*, et il est comparé au signal de référence selon l'équation 4.2. Les résultats de SQNR qui seront présentés dans les sections suivantes correspondent à une moyenne du SQNR pour tous les vecteurs de vérification d'un même type.

$$SQNR\,ref = \frac{\sum_{n=0}^{N-1} |dref(n)|^2}{\sum_{n=0}^{N-1} |dref(n) - dmath(n)|^2} \tag{4.1}$$

$$SQNR\,R2^2\,PC = \frac{\sum_{n=0}^{N-1} |dref(n)|^2}{\sum_{n=0}^{N-1} |dref(n) - dvhdl(n)|^2} \tag{4.2}$$

4.1.1.2 Aperçu de la théorie du bruit de quantification

L'étude du bruit de quantification provoqué par l'opération FFT sur des nombres en représentation point fixe est très complexe. Certaines analyses (Knight & Kaiser, 1979; Proakis & Manolakis, 1996; Rabiner & Gold, 1975; Tran-Thong & Liu, 1976) ont été faites pour des algorithmes de FFT précis et en posant des hypothèses sur les méthodes utilisées pour limiter les largeurs binaires. Les différents auteurs utilisent leurs propre nomenclature et métriques pour évaluer la borne du bruit de quantification. Certaines études donnent une borne supérieure de l'erreur, mais en pratique une erreur beaucoup plus faible est observée. Une étude approfondie ne sera pas faite dans cette section, on présente seulement certaines considérations.

Malgré le nombre réduit d'opérations présentes dans la FFT, le bruit de quantification est sensiblement le même que celui de la DFT correspondante (Proakis & Manolakis, 1996). Pour une FFT radix-2 avec la largeur des bus fixée à b bits et une mise à l'échelle de $N=2^v$, Proakis & Manolakis (1996) et Rabiner & Gold (1975) présentent des résultats théoriques similaires. L'expression du SQNR est présentée à l'équation 4.3, où C est une constante dépendante du signal d'entrée et de la manière dont la réduction binaire s'effectue. L'étude théorique du bruit de quantification pour l'algorithme radix-2^2 ne se trouve pas dans la littérature. On ne retrouve pas non plus d'analyses d'architectures qui permettent un accroissement binaire à la suite d'une mise à l'échelle et où la largeur est réduite seulement dans les multiplicateurs à une valeur paramétrée.

$$SQNR_{radix-2} = C\,2^{2b-v} \tag{4.3}$$

Les sources de bruit de quantification sont les suivantes :

- Arithmétique en point fixe et réduction de la largeur binaire
  La représentation d'un signal en complément à deux sur b bits introduit une erreur de quantification. Le rapport de puissance du signal au bruit de quantification (SQNR) est détaillé dans Proakis & Manolakis (1996) pour le cas où le signal est un sinus d'amplitude maximale. On trouve les équations 4.4 et 4.5 respectivement pour le rapport en puissance et en décibel. Des résultats similaires peuvent être trouvés pour des signaux occupant toute la plage dynamique. Ainsi, on gagne environ 6 dB pour chaque bit supplémentaire utilisé.

$$SQNR_{sinus\ quantifié} = \frac{3}{2} 2^{2b} \tag{4.4}$$

$$SQNR(dB) = 10 \log_{10} SQNR = 1{,}76 + 6{,}02\, b \tag{4.5}$$

  Les échantillons entrant dans une FFT possèdent déjà une quantification. Pour conserver toute la précision avec l'arithmétique en complément à deux, additionner ou soustraire deux valeurs sur b bits résulte en un mot sur b+1 bits et multiplier deux valeurs donne un résultat de la somme des largeurs binaires des opérandes moins un bit. Garder la précision maximale impliquerait beaucoup de ressources matérielles et serait inapplicable dans le cas du traitement par processeur. La largeur des mots binaires doit donc être limitée par l'arrondi ou la troncation (arrondi à la baisse), ce qui introduit une erreur. Vu la construction par étage de l'algorithme de la FFT, le bruit de quantification provenant des calculs d'un étage se propage aux étages suivants. Ainsi, chaque étage ajoute du bruit et l'ampleur de ce bruit dépend de la manière dont la réduction binaire est effectuée (Meyer, 1989).

- La précision des facteurs de phase

  Les facteurs de phase, $W_N^{nk} = e^{-j2\pi nk/N}$, correspondent à des coordonnées sur le cercle unitaire complexe. Plus le nombre de bits utilisés pour les représenter sera grand, plus ils seront précis et plus les résultats des multiplications le seront aussi. En revanche, selon Weinstein (1969) la quantification des données domine sur l'effet des facteurs de phase à précision finie.

- Les décalages à droite

  La mise à l'échelle pour éviter les débordements se fait par des décalages à droite d'un bit à la fois ce qui correspond à division par deux. L'équation 4.6 prouve qu'un décalage par étage d'un algorithme radix-2 suffit pour ne pas déborder de la plage dynamique (Proakis & Manolakis, 1996; Rabiner & Gold, 1975). Les indices i et j indiquent les points entrant et sortant du papillon radix-2. On peut donc effectuer la mise à l'échelle progressivement avec un décalage à droite des données avant chaque opération papillon, donc au total de v bits. Pour le radix-4 ou radix-2^2, le pire cas de mise à l'échelle est un décalage de v+1 bits à cause de l'arithmétique mis en jeux. Lors d'une division par deux, on peut soit arrondir le LSB ou augmenter la largeur binaire d'un bit. Cette dernière stratégie est celle adoptée pour le R2^2PC.

$$\begin{aligned} \max[|X_n(i)|,\ |X_n(j)|] &\leqslant \max[|X_{n+1}(i)|,\ |X_{n+1}(j)|] \\ &\leqslant 2\ \max[|X_n(i)|,\ |X_n(j)|] \end{aligned} \tag{4.6}$$

  Vu la relation de Parceval (éq. 2.1), le carré moyen du signal en fréquence est N fois le carré moyen du signal dans le temps. Ceci implique que l'on doit faire une division de $1/\sqrt{N}$ ou $2^{v/2}$ pour avoir conservation de puissance entre le domaine fréquentiel et le domaine temporel. Selon la nature du signal d'entrée, la mise à l'échelle maximale de 2^v n'a pas nécessairement besoin d'être effectuée, mais une mise à l'échelle supérieure à $2^{v/2}$ est souvent requise pour respecter la plage dynamique. Ce paramètre a un effet important sur le SQNR puisque chaque division par deux de

l'amplitude a pour effet de diminuer par un facteur quatre la puissance du signal, environ 6 dB. Ainsi, avec une mise à l'échelle de $S = 2^s$ on abaisse la puissance du signal en sortie à 2^{v-2s}. On retrouve ce terme dans l'équation 4.7 qui résume les effets combinés qui agissent sur le rapport signal à bruit. Le bruit de quantification qui se propage d'étage en étage est largement affecté par les largeurs binaires employées, mais son amplitude est réduite par la mise à l'échelle progressive. Les termes C et B de l'équation 4.7 représentent ces effets, mais ils ne seront pas davantage analysés dans le présent travail.

$$SQNR = C\,2^{B+v-2s} \tag{4.7}$$

4.1.2 Estimation des ressources requises

La plate-forme choisie pour tester l'implémentation est une puce programmable de la famille VirtexII Pro de Xilinx (Xilinx. 2004). Ces FPGA possèdent des ressources logiques sous forme d'unités de base nommées *slices*. Chaque *slice* contient deux générateurs de fonction à quatre entrées, une chaîne de retenue, des multiplexeurs et deux registres. Chaque générateur de fonction peut être programmé soit en une table de conversion (LUT, Look Up Table) à quatre entrées, soit en 16 bits de mémoire ou soit comme un registre à décalage de 16 bits. De plus, le FPGA possède des ressources dédiées telles que des multiplicateurs ayant une largeur d'entrée de 18 bits (BMULT). Une bascule peut être activée environ au centre de la logique de multiplication lorsque l'option « pipeline » est sélectionnée. Cette option permet d'accélérer la fréquence d'horloge et introduit un cycle de latence. De la mémoire embarquée est aussi disponible sous forme de *Block RAM* (BRAM) de taille 18 kbit. Ces blocs sont configurables en mode simple ou double ports et utilisent des mots d'entrée de 1, 2, 4, 18 ou 36 bits. Par exemple, on peut avoir une RAM 18x1024 ou 36x512.

Les résultats en terme de ressources requises pour l'implémentation dans ces puces seront exposés par le nombre de *slices*, de blocs de mémoire RAM (BRAM) et de multiplicateurs dédiés (BMULT). Des processeurs PowerPC sont aussi présents dans lesVirtexII Pro, mais ils ne sont pas utilisés par le $R2^2PC$.

Pour l'étape de la synthèse du code VHDL, le logiciel Synplify Pro 7.3.3 de Synplicity a été employé. Aucun paramètre dans l'interface usager de Symplify n'a été sélectionné parmi les options disponibles : *pipelining, retiming, ressources sharing, modular desing, FSM compiler.* Ces optimisations n'occasionnent pas de modifications au design synthétisé, mais augmentent le temps requis pour effectuer cette étape. L'attribut de Synplify *pipelining* permet d'activer la bascule dans les BMULT et il est présent à même le code VHDL du multiplicateur complexe.

Pour la plupart des scénarios de vérification, le FPGA choisi est le XC2VP20-7. Le suffixe -7 indique la performance sur le plan de la vitesse. En synthétisant avec un suffixe différent, on obtient une contrainte d'horloge plus restrictive. Lorsqu'on choisit un autre FPGA de la famille VirtexII Pro suffisamment grand pour contenir le design, on obtient les mêmes besoins en ressources. La seule contrainte imposée concerne la période de l'horloge, celle-ci est fixée à 5 ns (200 Mhz) dans le cas d'une puce -7. Le rapport de synthèse nous donne les ressources sous forme du nombre de LUT, de registres, de BMULT et de BRAM.

Finalement, les étapes de placement et de routage ont été effectuées avec l'outil de Xilinx ISE v5.2.03i avec les options suivantes sélectionnées:

- `Optimization Strategy : speed`
- `Perform Timing-Driven Packing : yes`
- `Place & Route Effort Level : high`

Le nombre de *slices* occupés lorsque le placement est effectuée est plus élevé que la moitié du plus grand nombre entre les LUT et les registres obtenus à la synthèse. Un *slice* comporte deux LUT et deux registres, mais l'outil de placement ne prend pas toutes les ressources disponibles dans chaque *slice* parce qu'il a suffisamment d'espace pour s'étendre dans la puce. On peut donc affirmer que la quantité de *slices* présenté est pire cas de ressources requises. En réalité, moins de *slices* seraient nécessaires dans le cas de l'utilisation du design dans un FPGA plus rempli.

4.1.3 Fréquence d'horloge

Les outils utiliés pour l'estimation des ressources requises permettent d'estimer la fréquence maximale de fonctionnement du R2^{2}PC. Elle correspond à la fréquence de l'horloge unique qui dépend du taux d'entrée des échantillons dans le module. La contrainte imposée sur la période de l'horloge est de 5 ns (200 Mhz) dans le cas d'une puce -7. Après le placement routage le rapport, de contrainte de synchronisation (*Timing*) nous indique que le chemin critique se trouve au niveau des multiplicateurs dédiés. La contrainte de 200 MHz dans un -7 a été respectée pour toutes les tailles de FFT.

Aucun placement manuel n'a été effectué puisque le R2^{2}PC doit être utilisable sans manipulation particulière par l'usager. Par contre, de meilleures performances de fréquence d'horloge ou de ressources pourraient êtres obtenues en contraignant le design dans des régions et en plaçant les blocs embarqués à l'aide d'un outil spécialisé.

4.2 Variation de la taille de la transformée

La première série de vérifications consiste à faire varier la taille de la transformée pour une largeur binaire d'entrée et de sortie fixée à 10 bits. Cette série permet de voir l'effet général du paramètre N sur la précision des résultats et sur les ressources nécessaires à

l'implémentation. Les effets de l'ordonnancement de l'entrée, du type de transformée et de la nature des vecteurs de test sont aussi traités. Pour tous les essais présentés dans cette section, la largeur des facteurs de phase a été fixée à 12 bits, ce qui permet de négliger son effet sur le bruit (section 4.5).

4.2.1 Scénarios de vérification

Série A:

- Taille (`LOG2N`) :	*v* de 3 à 12
- Largeur en entrée (`IN_FFT_W`) :	10
- Largeur en sortie (`OUT_FFT_W`) :	10
- Largeur des facteurs (`TF_W`) :	12
- Largeur interne (`W`) :	valeur constante de 10, 11 et 12 bits, exemple (10, 12,...12, 10)
- Mise à l'échelle (`SCALING`) :	*v*
- Ordonnancement (`BIT_REV_IN`) :	les deux, naturel et renversé
- Type (`INVERSE`) :	directe pour tous les essais et inverse pour certains
- Multiplicateurs (`MULT_STYLE`) :	"*block_mult*"
- Arrondi (`MULT_ROUND_W`) :	18

- 100 vecteurs de type I ayant un amplitude de $\sqrt{2}/2$
- 100 vecteurs de type II ayant un amplitude de $\sqrt{2}/2$

Note : avec une amplitude de $\sqrt{2}/2$ pour la sinusoïde de chaque composante complexe, on obtient une norme maximale théorique de 1.

4.2.2 Résultats de SQNR

4.2.2.1 Effet du type de vecteur d'entrée

À la figure 34, on constate que les résultats de simulation suivent la tendance des résultats du modèle mathématique de référence. La même pente peut être observée sur les courbes de référence et simulées. L'écart séparant ces courbes sera détaillé davantage avec la figure 35.

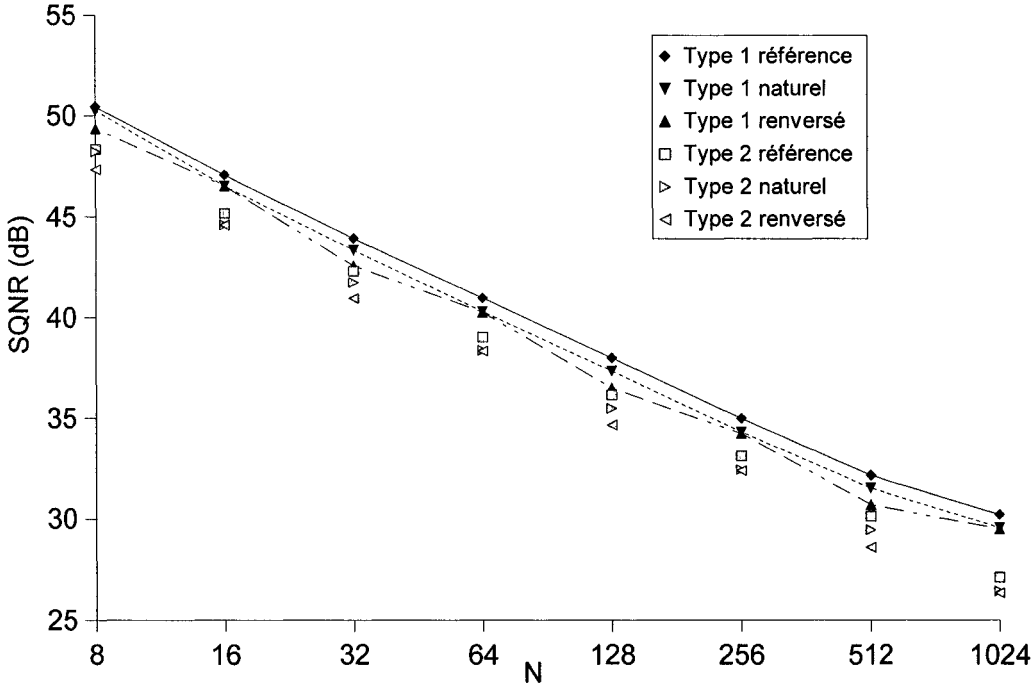


Figure 34 SQNR pour différents vecteurs d'entrée appliqués à des FFT avec des bus de 10 bits en fonction de la taille

La différence entre les courbes de SQNR issues de la vérification faite avec les vecteurs de type I et de type II est attribuable à la différence de puissance de ces vecteurs. Pour les sinusoïdes du type I, la puissance moyenne est de 1/2 par point tandis que celle des points aléatoires du type II est de 1/3 en moyenne. On obtient ainsi un écart théorique moyen de 1,8 dB entre les deux types. Cet écart est aussi observé graphiquement entre

les courbes. La nature du signal traité par le $R2^2PC$ influence la valeur du SQNR, mais elle n'influence pas le comportement général des résultats. Pour cette raison, on utilisera soit les vecteurs de type I ou de type II dans les séries subséquentes, les deux types n'étant pas nécessaires.

4.2.2.2 Effet de la taille

Avec les largeurs de mots fixées, la taille de la transformée a un effet important sur la précision atteignable. Plus la taille de la transformée grandit, plus la précision diminue. La diminution du SQNR que l'on retrouve sur les figures 34, 35 et 36 est de l'ordre de 3 dB à chaque fois que la taille double. Cet effet est attribuable à la mise à l'échelle qui provoque une diminution de puissance de 6 dB du signal en sortie pour chaque division par deux (preuve à la section 4.6) et par l'opération FFT qui, à l'inverse, provoque une augmentation de puissance d'un facteur N, donc $3v$ dB (expliqué à la section 4.1.1.2). Les cas couverts ont une mise à l'échelle de N, c'est-à-dire v divisions par 2. Ainsi, la puissance du signal est réduite de $3v$–6s dB (-$3v$ dB si v=s) par rapport à la puissance à l'entrée. Le SQNR théorique calculé selon l'équation 4.5 est d'environ 62 dB en considérant l'entrée sur 10 bits. Si on calcule 62,0 dB - $3v$ dB, on trouve une courbe située à environ 2 dB au dessus de la courbe de référence pour les vecteurs de type I. On peut donc établir que l'ordre de grandeur des résultats de SQNR obtenu est en accord avec la théorie.

4.2.2.3 Effet de l'ordonnancement

Sur le graphique 34, on observe que le SQNR de la transformée à l'entrée ordonnée a une courbe plus lisse que celui de la transformée en ordre des bits renversé. Cet effet est mis davantage en évidence par le graphique 35. On remarque que quand N est une puissance de quatre, le SQNR est sensiblement le même pour les deux ordonnancements. En revanche, pour un N d'une puissance de deux mais non de quatre, on observe une perte

de performance pour l'architecture renversée par rapport à la version ordonnée. Dans le cas d'une largeur interne de 10 bits, cet écart est inférieur à 2 dB et, pour 11 bits de largeur interne, il est de moins de 1 dB. Cet effet peut être attribuable aux étapes de calcul et de mise à l'échelle qui ne s'exécutent pas dans le même ordre pour les deux architectures.

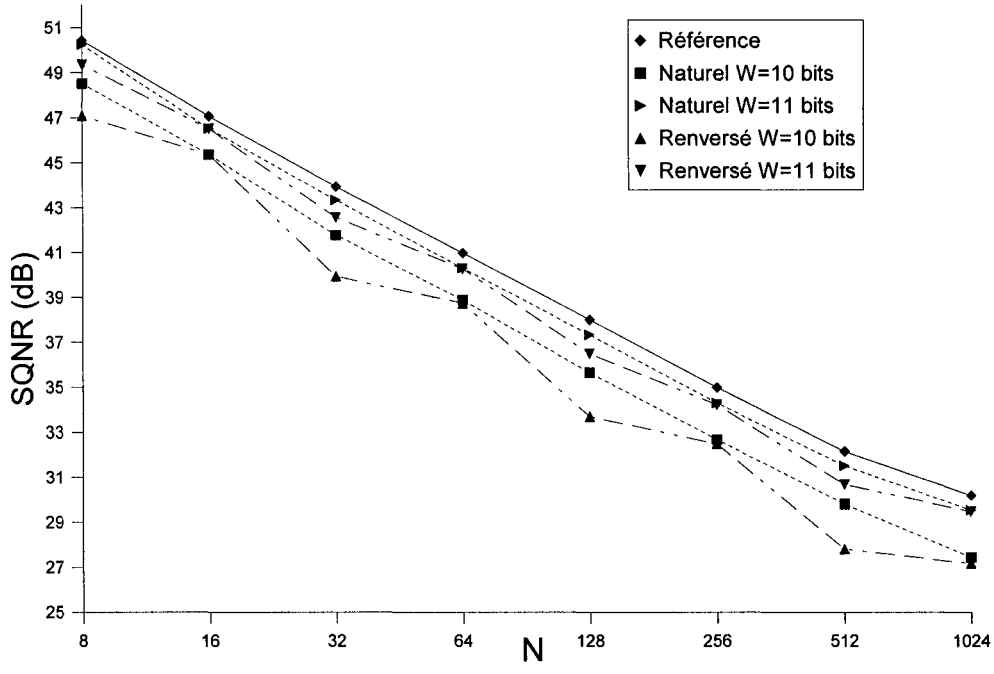


Figure 35 SQNR de FFT avec l'entrée en ordre naturel et en ordre renversé en fonction de la taille

4.2.2.4 Effet de la largeur binaire interne

Le graphique 36 nous montre l'influence de la largeur interne maintenue fixe sur la précision des résultats. Si on maintient une largeur de 10 bits suivant les multiplicateurs complexes, on se retrouve avec une perte de performance qui se chiffre en moyenne à 2,2 dB par rapport à la courbe de référence. Cet écart moyen est réduit à 0,58 dB et à 0,17 dB respectivement pour une largeur interne de 11 et 12 bits. Aller au-delà de deux

bits supplémentaires (> 12 bits) ne serait pas avantageux pour le SQNR et utiliserait des ressources inutilement. Cette augmentation de la précision en fonction de la largeur binaire est attribuable au fait que plus de bits sont conservés pour les calculs. Les opérations d'arrondi introduisent ainsi moins d'erreurs dans les résultats. Cet effet est aussi traité par les cas de test des sections 4.3 et 4.6. La croissance en terme de ressources est exposée dans la prochaine section.

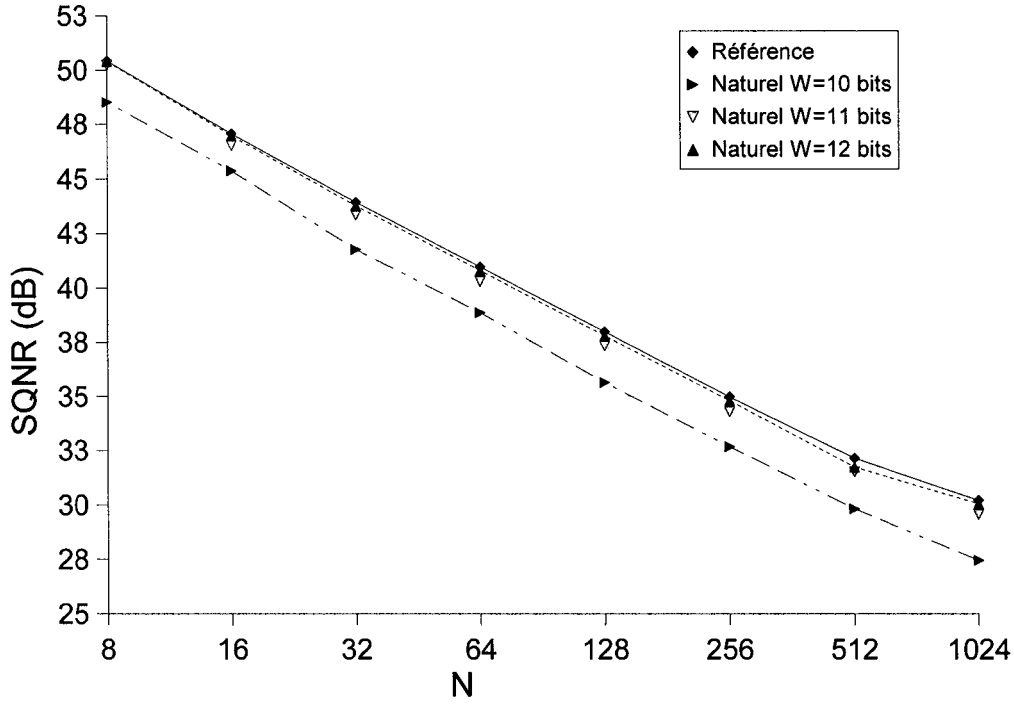


Figure 36 SQNR de FFT de 10 bits d'entrée/sortie avec la largeur interne constante en fonction de la taille

4.2.2.5 Cas de l'IFFT

Les résultats du SQNR pour la transformée rapide de Fourier inverse (IFFT) sont les mêmes que ceux obtenus pour la FFT. Les opérations de conjugué complexe n'introduisent pas d'erreurs de quantification supplémentaires. Les tests de SQNR qui suivent portent donc seulement sur la FFT, les résultats de IFFT étant équivalents.

4.2.3 Ressources nécessaires

Dans l'architecture du R2^{2}PC, un étage (papillon et délais en rétroaction) s'ajoute chaque fois que la taille de la transformée double et un multiplicateur complexe accompagné d'une table de facteurs de phase s'ajoutent lorsque la taille quadruple (au passage d'une puissance de quatre à la puissance de deux suivante). On s'attend à ce que plus N est grand, plus de ressources soient nécessaires et c'est ce qu'on observe sur le graphique 37. La croissance en nombre de *slices* est logarithmique. Comme prévu, $2\log_2(N)$ -2 ou $2\log_2(N)$ -4 multiplicateurs sont requis selon que v soit impair ou pair. L'utilisation des multiplicateurs dédiés croît avec des bonds de quatre BMULT, ce qui équivaut à un multiplicateur complexe. Pour ce qui est des blocs de mémoire RAM, leur nombre dépend des paramètres d'implémentation. Ceux-ci ne sont pas nécessaires pour les transformées de petite taille ($N < 256$), la mémoire étant alors implémentée dans les *slices*.

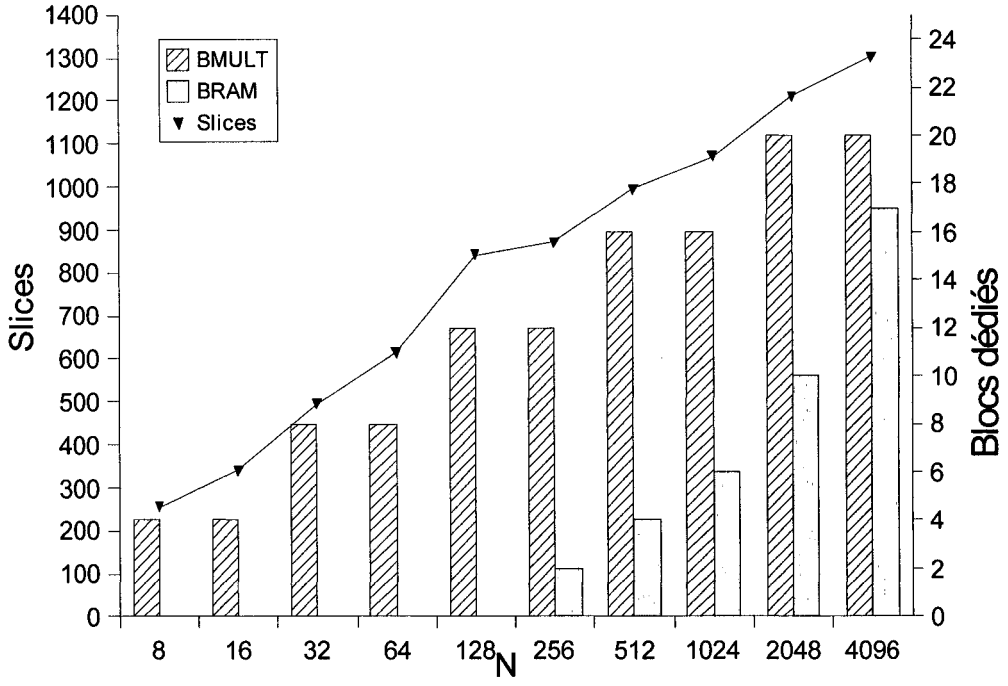


Figure 37 Ressources nécessaires à des FFT à l'entrée ordonnée avec 10 bits de large pour les bus de données et 12 bits pour les facteurs de phase

Le point de la courbe du nombre de *slices* pour N =128 rompt la linéarité du graphique en étant presque au même niveau que celui pour N=256. Ce cas particulier est dû au fait que deux BRAM sont utilisés dans le cas où N=256 alors qu'aucun n'est utilisé pour N=128. On pourrait ramener le point de N=128 sur la droite en utilisant un BRAM pour stocker la plus grande table de facteurs de phase (64 facteurs) et/ou la plus longue ligne à délais (64 données). Les BRAM ont une taille maximale de 18 kbit. Pour notre application, les configurations intéressantes sont 1024 mots de 18 bits ou 512 mots de 36 bits. Cette dernière configuration est intéressante, car on peut généralement juxtaposer les parties réelles et imaginaires d'une donnée dans 36 bits. Voici la configuration mémoire de certains des cas illustrées :

- N=256 : 1 de 192 x 24 bits pour une table de facteurs de phase

1 de 128 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction

- *N*=512: 1 de 256 x 24 bits pour une table de facteurs de phase
  1 de 192 x 24 bits pour une table de facteurs de phase
  1 de 128 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction
  1 256 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction
- *N*=1024: 1 de 256 x 24 bits pour une table de facteurs de phase
  2 de 768 x 12 bits pour une table de facteurs de phase
  1 de 128 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction
  1 de 256 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction
  1 de 512 x 21 ou 22 bits en mode FIFO pour un délai en rétroaction

Selon la taille du module et en connaissant son architecture interne, il est assez simple de prédire combien de BRAM sont utilisés et quels sont leur rôle.

La figure 38 compare les ressources nécessaires en terme de *slices* pour la FFT et l'IFFT à l'entrée en ordre naturel et celle en ordre des bits renversé. L'IFFT demande plus de *slices* que la FFT parce que son implémentation requiert deux additionneurs supplémentaires, ce qui représente environ 25 *slices* pour le cas illustré. On remarque aussi que les transformées en ordre renversé requièrent plus de ressources que les transformées en ordre naturel. Même si le total de mots dans les délais en rétroaction reste inchangé, peu importe l'ordonnancement, les largeurs de mots contenues dans les délais en rétroaction augmentent de l'entrée vers la sortie. Prenons par exemple l'étage d'entrée d'un groupe de deux étages, l'étage Y. On y entre sur 10 bits et on en ressort sur 11 bits, une conséquence de la mise à l'échelle. La largeur binaire de la ligne à délai associée à l'étage Y est de 11 bits et elle contient des mots de 22 bits. L'étage suivant (Y-1) passe de 11 à 12 bits, ce qui donne un délai en rétroaction avec des mots sur 24 bits. Si l'entrée est de type ordonnée la taille du délai associé à l'étage Y-1 est de 2^{Y-1} et elle contient de mots de 24 bits de largeur. Par contre, si l'entrée est de type renversé la taille du délais est de 2^{Y+1} par 24 bits.

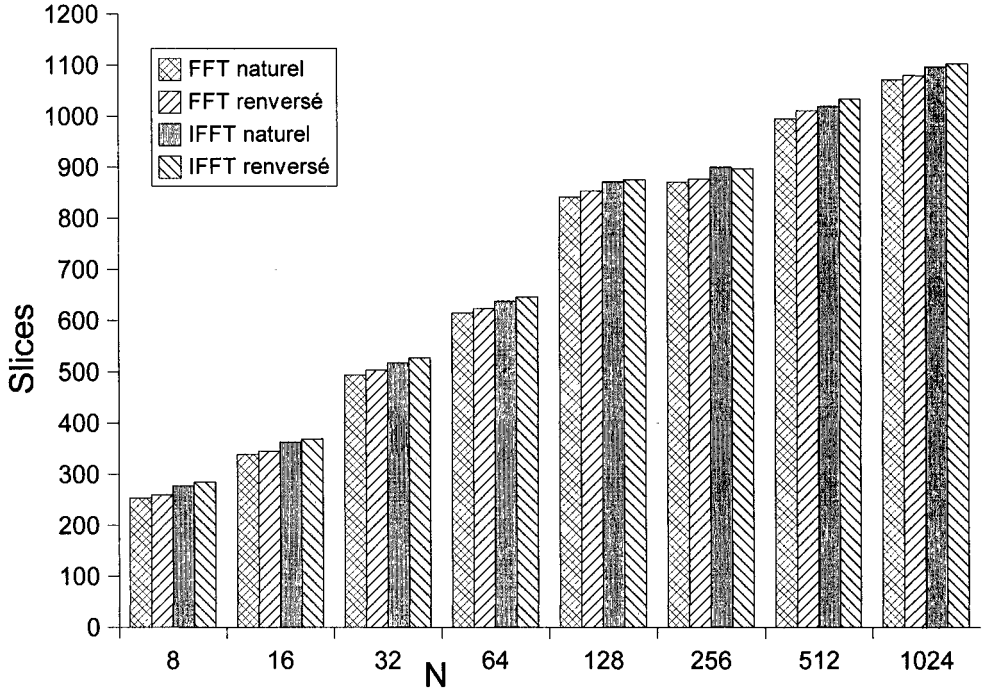


Figure 38 Nombre de *slices* nécessaires pour des FFT et IFFT de différentes tailles avec les deux types d'ordonnancement

Conséquemment, la variation du nombre de *slices* utilisés provient du fait que dans le cas renversé, les étages ayant des tailles binaires supérieures sont aussi ceux ayant les plus longues lignes à délai dans un groupe de deux étages. L'utilisation de BRAM n'est pas sujet à cet effet, puisqu'ils ne sont en général pas utilisés à leur pleine largeur, soit de 36 bits pour 512 mots. En bref, l'écart maximal observé pour les différentes configurations d'une transformée de taille donnée et de largeur de bus constante est de l'ordre de 50 *slices*, ce qui est relativement peu.

L'utilisation d'un bit de plus pour les bus à l'interne s'accompagne d'une augmentation du nombre de *slices*. Le tableau IV présente les résultats pour des tailles internes de 10 et 11 bits pour lesquelles le SQNR a été exposé à la figure 36. On trouve qu'un bit de plus cause une augmentation de 4% à 8 % de l'utilisation des *slices*.

Tableau IV

Ressources nécessaires pour des FFT de 10 bits en entrée et en sortie et de 10 ou 11 bits de largeur interne

N	**Nombre de *slices***	
	10 bits à l'interne	11 bits à l'interne
8	254	265
16	339	351
32	494	530
64	615	646
128	842	900
256	871	926
512	995	1068
1024	1071	1144

4.3 Variation de la largeur binaire en sortie

Les cas de vérification exposés dans cette section permettent de voir l'effet de l'accroissement de la largeur binaire entre l'entrée et la sortie. La série **B** concerne une FFT de 16 points avec entrée à largeur fixe de 5 bits. Deux paramètres seront variés : la largeur en sortie de 5 à 12 bits et la largeur binaire interne après le multiplicateur complexe. Puisqu'on retrouve un seul multiplicateur dans cette configuration, on peut étudier l'effet de la variation de la largeur interne sur la précision des résultats.

La série **C** quant à elle concerne une FFT de N=256, où sa largeur en entrée est fixée à 10 bits et sa largeur en sortie varie de 10 à 22 bits. Les largeurs internes paramétrées par `OUT_FFT_W` ou plus simplement `O_W` sont dans tous les cas fixées à un bit de plus que la largeur de sortie. On retrouve des largeurs binaires précédent certains multiplicateurs à des valeurs de `O_W` + 3, plus un bit pour la largeur interne et plus deux bits pour la mise à l'échelle. Selon la valeur de `O_W`, la largeur à l'entrée du

multiplicateur peut dépasser 18 bits où elle excède la taille des multiplicateurs dédiés de Xilinx. Si l'on veut se limiter à l'utilisation des multiplicateurs dédiés, on doit fixer MULT_ROUND_W à 18. D'autre part, pour voir l'effet de cet arrondi sur la précision, on peut le fixer de manière à ce qu'il n'y ait pas d'arrondi précédant le multiplicateur.

4.3.1 Scénarios de vérification

Série B

- Taille (LOG2N) : 4
- Largeur en entrée (IN_FFT_W) : 5
- Largeur en sortie (OUT_FFT_W) : 5, 6, 8, 10 et 12
- Largeur des facteurs (TF_W) : 10 (effet non dominant sur le bruit)
- Largeur interne (W) : largeur binaire interne fixée de OUT_FFT_W-2 à OUT_FFT_W+2
- Mise à l'échelle (SCALING) : 4
- Ordonnancement (BIT_REV_IN) : les deux, naturel et renversé
- Type (INVERSE) : directe
- Multiplicateurs (MULT_STYLE) : "*block_mult*"
- Arrondi (MULT_ROUND_W) : 18
- 100 vecteurs de type I ayant un amplitude de 0,8
- 100 vecteurs de type II d'amplitude de]-1,1[

Série C

- Taille (LOG2N) : 8
- Largeur en entrée (IN_FFT_W) : 10
- Largeur en sortie (OUT_FFT_W) : 10 à 22 avec des incréments de 2 bits
- Largeur des facteurs (TF_W) : 16 ou 18 (effet non dominant sur le bruit)
- Largeur interne (W) : Largeur binaire interne fixée à OUT_FFT_W+1
- Mise à l'échelle (SCALING) : 7
- Ordonnancement (BIT_REV_IN) : les deux, naturel et renversé

- Type (INVERSE) : directe
- Multiplicateurs (MULT_STYLE): "*block_mult*"
- Arrondi (MULT_ROUND_W): 18 à 24
- 100 vecteurs de type I ayant une amplitude $\sqrt{2}/2$
- 100 vecteurs de type II d'amplitude de]-1,1[

4.3.2 Résultats de SQNR

Les résultats de la série **B** sont illustrés par la figure 39. Les points en forme de losange représentent le SQNR de référence pour chaque taille en sortie. On remarque qu'environ 6 dB de précision sont gagnés pour chaque augmentation d'un bit de la largeur en sortie. En accordant plus de bits pour les calculs et en sortie, le poids des erreurs devient plus faible.

On observe aussi que la valeur de référence est pratiquement atteinte lorsque la largeur interne dépasse la largeur de la sortie. Les erreurs dues à la représentation binaire de taille limitée sont alors grandement réduites. Lorsque la largeur interne est la même que celle à la sortie, on trouve une perte moyenne de 1,6 dB par rapport à la référence pour les cas testés. Si on utilise 2 bits de moins à l'interne qu'à la sortie, l'effet est plus important et l'écart devient de 8,2 dB, 7,1 dB et 6,8 dB respectivement pour les largeurs en sortie de 8, 10 et 12 bits. Un phénomène semblable est aussi observé avec la série **A** de la section 4.2. La largeur interne est un facteur important à considérer et pour plus de précision, elle doit être fixée à une valeur égale ou supérieure à la largeur de sortie.

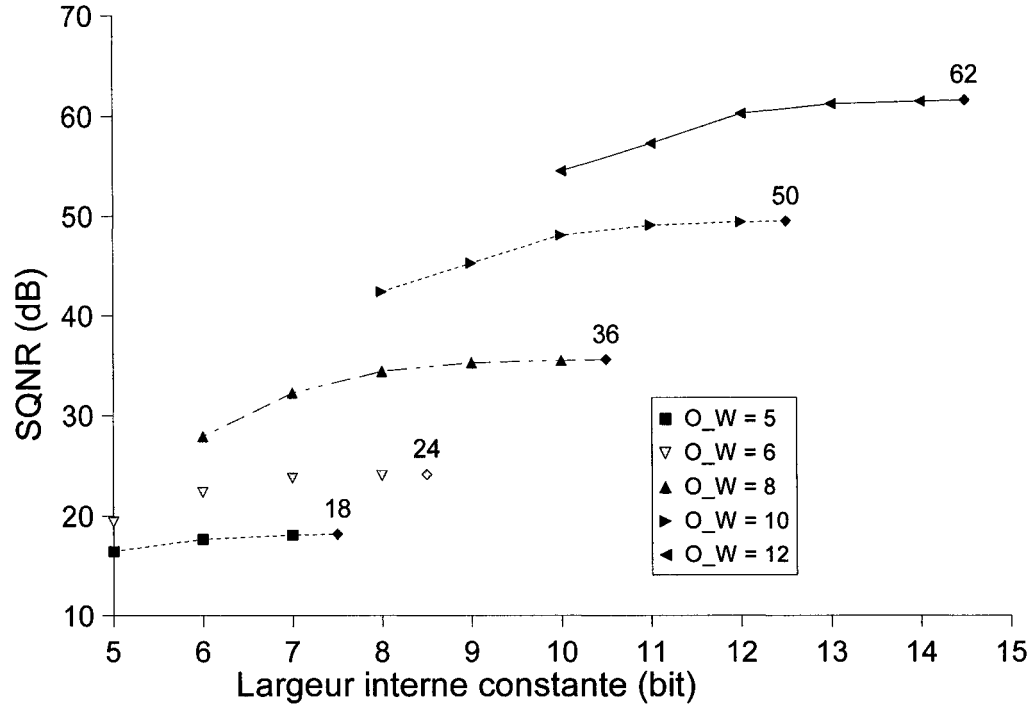


Figure 39 Effet de la largeur du bus interne dans les cas de FFT avec N=16, une entrée sur 5 bits et différentes largeurs en sortie

Les résultats de la série **C** sont présentés à la figure 40. La droite de référence a une pente de 6 dB par bit de sortie supplémentaire, ce qui est en accord avec la série **B**. La largeur interne fixée à un bit de plus que la largeur de sortie permet d'obtenir des droites de simulation avec seulement 2,2 dB d'écart moyen à la référence et ce jusqu'à ce que la largeur de sortie atteigne 16 bits. Dans ce dernier cas (`O_W`=16), la largeur avant les multiplicateurs est de 19 bits et elle dépasse la largeur maximale du multiplicateur dédié. Par conséquent, le mot est arrondi à 18 bits avant le multiplicateur ce qui introduit une erreur de quantification. En fixant en simulation `MULT_ROUND_W` à des valeurs supérieures à 18, on peut éliminer cette erreur. L'écart à la référence est de 2,2 dB sans l'arrondi et de 2,5 dB avec l'arrondi. Cette petite différence de 0,3 dB n'est pas visible sur le graphique 40. En revanche, plus la largeur pré-multiplicateur augmente au-delà de 18 bits, plus l'effet de l'arrondi se fait sentir. Il en vient à faire plafonner la courbe du SQNR à environ 89 dB, tandis que la courbe sans arrondi continue de suivre la courbe de

référence. En limitant la largeur binaire interne à 18, les bits supplémentaires employés ne sont plus significatifs et ne contribuent pas à augmenter le SQNR.

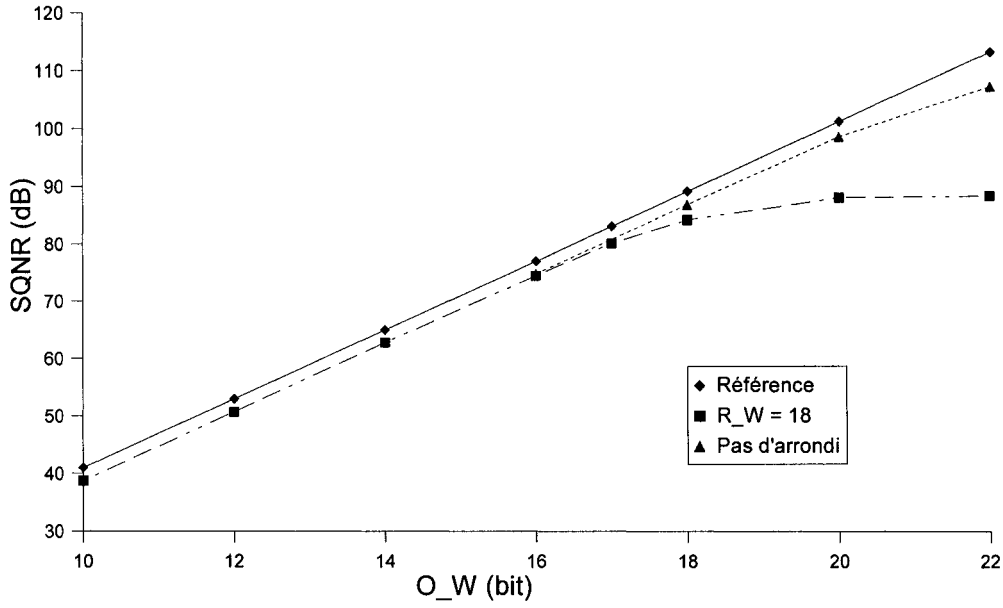


Figure 40 Effet de la largeur binaire de sortie pour une entrée de 10 bits et N=256

4.3.3 Ressources nécessaires

L'usage des ressources pour certains cas de l'esssai **C** est illustré par le graphique 41. Dans tous les cas, `MULT_ROUND_W` a été fixé à 18 bits, ainsi 12 BMULT sont employés. Pour les cas où `O_W` est fixé à 10, 12 et 14 bits, on obtient que la FFT renversée requière plus de *slices* que celle ordonnée à cause de la construction par étage du R2^{2}PC. Les deux FFT utilisent un BRAM comme FIFO pour la ligne à délais de taille 128 et un autre pour contenir la plus grande table de facteurs de phase, soit 192 mots complexes.

L'ordre des courbes d'usage des *slices* s'inverse pour `O_W` valant 17 et 20 bits. C'est une conséquence de l'usage accru de BRAM pour la transformée en ordre renversé. En effet, 2 BRAM sont utilisés pour la FFT en ordre naturel alors que 5 BRAM le sont pour la FFT en ordre renversé. Pour ce dernier type de FFT un BRAM est employé pour contenir la plus grande table de facteur de phase et quatre sont utilisés comme FIFO pour les délais en rétroaction. L'outil de synthèse utilise deux BRAM pour implémenter le délai de taille 128 par 38 ou 44 bits et deux autres pour le délai 64 par 38 ou 44 bits, respectivement selon que la sortie est de 17 ou 20 bits. Pour ces deux derniers cas, un choix différent de ressources aurait pu être fait en utilisant moins de BRAM, mais plus de *slices*. Ceci permettrait à la courbe de la FFT renversé de rester au-dessus de celle de la FFT ordonnée.

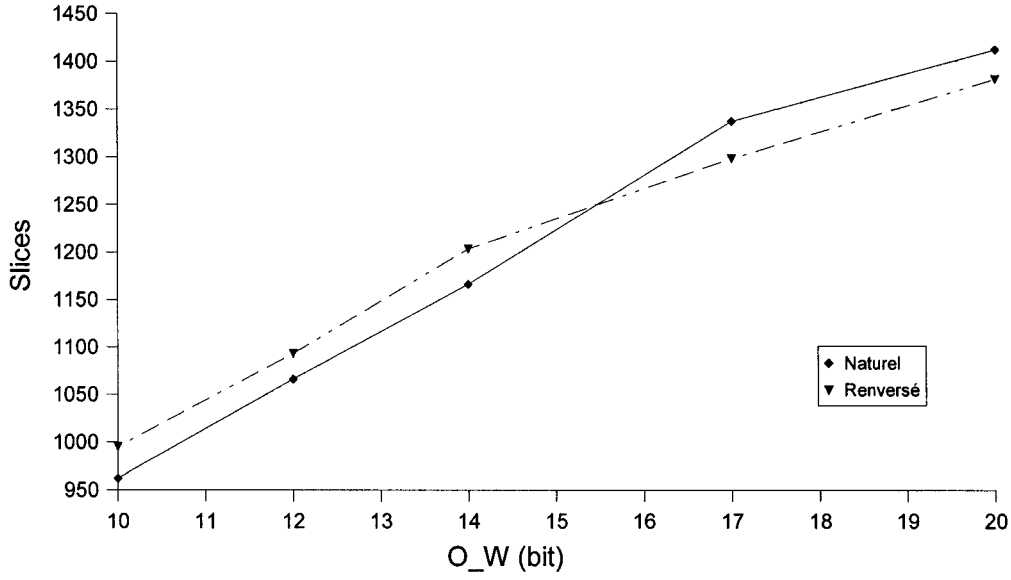


Figure 41 *Slices* nécessaires selon l'ordonnancement et la largeur en sortie

Outre les particularités mentionnées précédemment, on remarque que plus la taille en sortie est grande, plus l'usage de ressources augmente. Avec N=256 et une largeur interne d'un bit supérieur à `O_W`, on trouve un gain approximatif de 50 *slices* pour chaque bit supplémentaire requis pour le port de sortie et les bus l'internes.

Tableau V

Ressources requises par des FFT de 256 points avec 10 bits en entrée et 20 bits en sortie en fonction du paramètre d'arrondi et du style des multiplicateurs

MULT_ ROUND_W	Mult_ style	Ordre naturel			Ordre renversé		
		slice	BMULT	BRAM	*slice*	BMULT	BRAM
18	block_mult	1412	12	2	1381	12	5
23	block_mult	1527	20	2	1527	20	5
23	logic	3694	0	2	3611	0	5

Dans la section précédente, il a été fait mention de l'arrondi à 18 bits qui limite la précision. Voyons maintenant l'effet de ne pas faire l'arrondi sur l'usage des ressources. Le tableau V nous montre trois cas d'implémentation possibles pour une FFT de N=256 ayant 10 bits en entrée, 20 bits en sortie, une largeur interne post-multiplicateur de 21 bits avec une largeur pré-multiplicateur qui peut aller jusqu'à 23 bits. Le premier cas force l'arrondi à 18 bits pour 8 des 12 BMULT utilisés. Dans le deuxième cas, aucun arrondi n'est fait avant les multiplicateurs et l'usage de multiplicateurs dédiés est forcé par l'attribut « *block_mult* ». Par conséquent, 20 multiplicateurs dédiés sont utilisés, soit un premier groupe de quatre entre les étages sept et six où la largeur pré-multiplicateur est de 12 bits et deux groupes de huit multiplicateurs dont la largeur d'une entrée est de 23 bits. L'usage des *slices* augmente vu l'usage de registres plus larges pour implémenter le pipeline du module multiplicateur complexe. Le troisième cas implémente les multiplicateurs sous forme de logique, donc en *slices*. On remarque un usage de *slices* de plus du double de celui nécessaire aux autres implémentations. Les multiplicateurs dédiés permettent donc d'économiser une portion non négligeable des ressources logiques. On peut faire le calcul approximatif suivant pour estimer les ressources nécessaires aux 12 multiplicateurs de 23 par 16 bits: (23x16)/2 *slices* x 12 = 2208 *slices*. Ce calcul ne tient pas compte de l'augmentation de la largeur des registres du

pipeline. À la lumière de ces résultats, on se rend compte que des compromis doivent être fait entre l'usage des *slices* et de ressources dédiées.

4.4 Variation progressive de la largeur interne

Les scénarios de vérification précédents, où la largeur binaire interne a été maintenue fixe, nous ont montré que plus on conserve un grand nombre de bits, plus la FFT est précise. La présente vérification porte sur l'accroissement graduel de la largeur interne entre chaque module de niveau hiérarchique intermédiaire, c'est-à-dire suite aux multiplicateurs. Les largeurs internes variables permettent de réduire les ressources requises au prix d'une perte de performance du SQNR. Cet effet a été testé pour des FFT de 256 points avec 10 bits à l'entrée, 17 bits à la sortie et trois largeurs internes de paramétrées.

4.4.1 Scénarios de vérification

Série D

- Taille (`LOG2N`) : 8
- Largeur en entrée (`IN_FFT_W`) : 10
- Largeur en sortie (`OUT_FFT_W`) : 17
- Largeur des facteurs (`TF_W`) : 16 (effet non dominant sur le bruit);
- Largeur interne (`W`) : augmente graduellement de l'entrée vers la sortie
- Mise à l'échelle (`SCALING`) : 7
- Ordonnancement (`BIT_REV_IN`): les deux, naturel et renversé
- Type (`INVERSE`) : directe
- Multiplicateurs (`MULT_STYLE`): "*block_mult*"
- Arrondi (`MULT_ROUND_W`): 18
- 100 vecteurs de type I ayant une amplitude de $\sqrt{2}/2$

4.4.2 Résultats de SQNR et ressources

Le SQNR de référence est de 83,1 dB pour une FFT de 256 points avec une entrée de 10 bits et une sortie de 17 bits. Le tableau VI illustre les résultats obtenus pour les différents cas couverts. Les trois premiers cas ont des largeurs internes variables et les deux derniers des largeurs internes fixes. Dans tous les cas, 12 BMULT sont employés. On remarque que plus les largeurs internes sont petites, moins la transformée est précise. Pour le premier cas, l'écart à la référence est de plus de 20 dB et le SQNR obtenu est similaire à celui obtenu avec seulement 14 bits de sortie et 15 bits à l'interne à la série C.

L'augmentation de la précision s'accompagne aussi d'une augmentation du nombre de *slices.* La logique supplémentaire est assez faible lorsqu'on considère l'importante augmentation du SQNR obtenue. Selon l'utilisation prévue de la FFT et les ressources disponibles, l'utilisateur du $R2^2PC$ est en mesure de faire le bon choix de configuration. Par contre, le seul moyen d'atteindre des performances s'approchant du SQNR de référence est de limiter au maximum la quantification en fixant les bus internes à la même taille ou à un bit supérieur à la largeur de sortie.

Tableau VI

SQNR et ressources pour de FFT de 256 points avec les largeurs binaires internes augmentant progressivement

Variation de la largeur de l'entrée vers la sortie	**Ordre naturel**			**Ordre renversé**		
	SQNR (réf. 83,1)	*slice*	BRAM	SQNR (réf. 83,1)	*slice*	BRAM
(10, 12, 14, 16, 17)	62,2	1117	2	62,8	1093	3
(10, 13, 15, 17, 17)	68,0	1153	2	68,5	1146	3
(10, 14, 16, 17, 17)	72,8	1179	2	73,0	1177	3
(10, 17, 17, 17, 17)	76,8	1288	2	76,5	1254	3
(10, 18, 18, 18, 17)	80,2	1337	2	79,7	1298	5

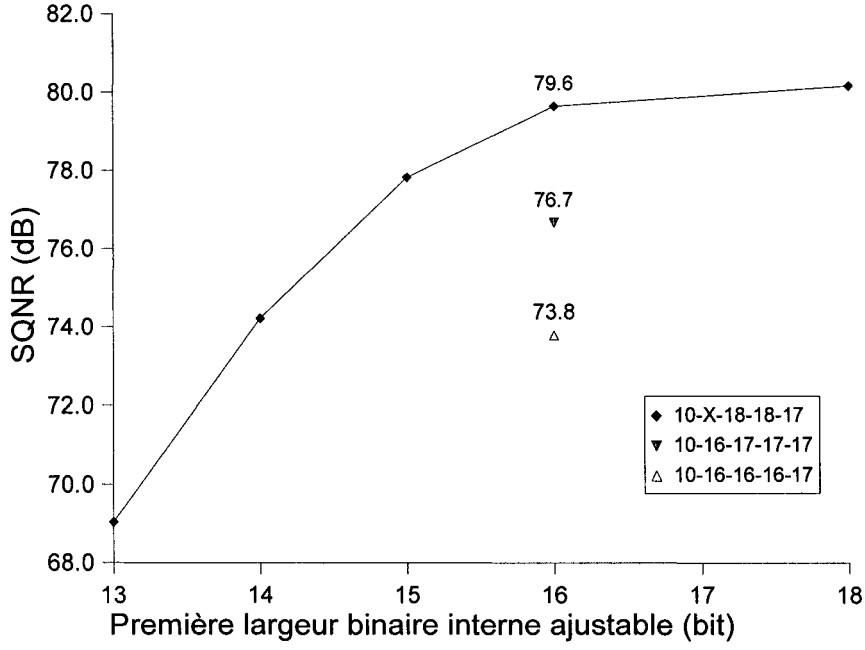


Figure 42 SQNR d'une FFT de 256 points avec 10 bits en entrée, 17 bits en sortie en fonction de certaines largeurs binaires internes ajustables.

Pour voir l'effet seul de la largeur à la suite du premier multiplicateur, on a fait varier cette largeur en maintenant les autres largeurs internes à 18 bits pour réduire leur influence. Les résultats obtenus en terme de SQNR sont illustrés par la courbe de la figure 42. On remarque le gain de performance provenant de l'augmentation de la première largeur interne paramétrable est important lorsqu'on passe de 13 à 14 bits ou de 14 à 15 bits, cependant le gain est faible au-delà de 16 bits comme première largeur interne pour une largeur de sortie de 17 bits. Sur la figure 42, l'effet des autres largeurs internes est aussi illustré par le cas particulier où la première largeur est fixée à 16 bits et les deux suivantes sont fixées à 16, 17 et 18 bits. On remarque un gain d'environ 3 dB pour chaque augmentation d'un bit de ces largeurs, ce qui n'est donc pas négligeable. La variation (10, 15, 18, 18, 17) donne de meilleures performances que celle avec (10, 16, 17, 17, 17). Le bruit de quantification provient de l'effet combiné de tous les arrondis effectués au cours des calculs.

4.5 Variation de la largeur des facteurs de phase

La largeur des facteurs de phase est aussi un paramètre important du R2^{2}PC. Dans chacun des tests précédents, on a attribué à ce paramètre une valeur constante. Pour tester son effet, on a fait varier sa valeur pour des FFT de différentes tailles et avec des mots de diverses largeurs.

4.5.1 Scénarios de vérification

Série E

- Taille (`LOG2N`) : 4, 6, 8, et 10
- Largeur en entrée (`IN_FFT_W`) : 16
- Largeur en sortie (`OUT_FFT_W`) : 16
- Largeur des facteurs (`TF_W`) : 10 à 18 par incrément de 2 bits
- Largeur interne (`W`) : valeur constante de 16

- Mise à l'échelle (SCALING) : *v*
- Ordonnancement (BIT_REV_IN) : naturel
- Type (INVERSE) : directe
- Multiplicateurs (MULT_STYLE) : "*block_mult*"
- Arrondi (MULT_ROUND_W) : 18
- 100 vecteurs de type I ayant une amplitude de $\sqrt{2}/2$

Série F

- Taille (LOG2N) : 10
- Largeur en entrée (IN_FFT_W) : 10, 14 et 16
- Largeur en sortie (OUT_FFT_W) : 10, 14 et 16
- Largeur des facteurs (TF_W) : 8 à 16 bits par incrément de 2 bits
- Largeur interne (W) : fixée à une valeur constante de 10, 14 et 16 bits
- Mise à l'échelle (SCALING) : 10
- Ordonnancement (BIT_REV_IN) : naturel
- Type (INVERSE) : directe
- Multiplicateurs (MULT_STYLE) : "*block_mult*"
- Arrondi (MULT_ROUND_W) : 18
- 100 vecteurs de type II ayant une amplitude de $\sqrt{2}/2$

4.5.2 Résultats de SQNR et ressources

La figure 43 illustre les résultats de la série **E**. Les symboles en bordure du graphique à droite représentent le SQNR de référence associé à chaque courbe. Il y a saturation des performances pour une largeur de facteurs de phase relativement faible (14 à 16 bits) pour des transformées configurées avec 16 bits pour tous les bus de données. La taille de la transformée influence la hauteur du plateau, mais n'a pas d'impact sur le comportement associé à la largeur des facteurs de phase. L'écart que l'on retrouve entre le plateau et la référence est attribuable à la quantification et à l'arrondi des données au travers des calculs. La quantification des facteurs de phase n'a pas d'effet dominant sur

le bruit de quantification causé par les erreurs d'arrondi, ce qui est en accord avec Weinstein (1969). Par contre, elle peut dominer le bruit si les facteurs sont trop quantifiés. Par exemple, avec `TF_W` = 10 bits, le SQNR subit une dégradation de importante, jusqu'a 10 dB.

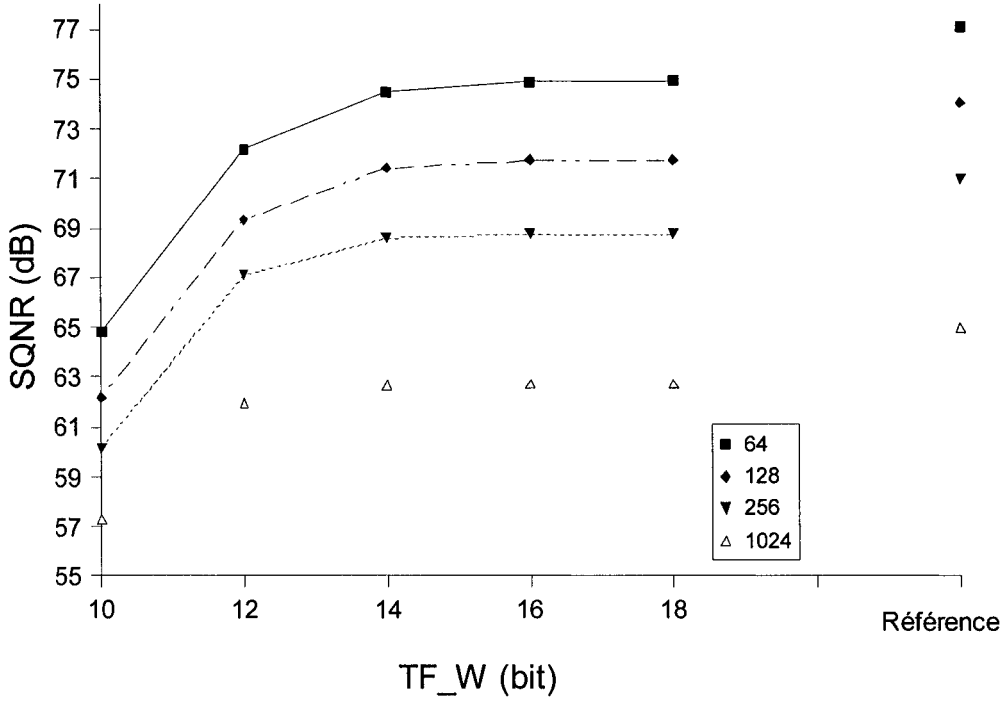


Figure 43 SQNR pour des FFT de différentes tailles avec des bus de 16 bits en fonction de la largeur des facteurs de phase

Voyons maintenant l'effet de la modification de la largeur des facteurs de phase sur les ressources (figure 44). On peut voir que le nombre de *slices* utilisés a augmenté de moins de 5 % lors du passage de 10 à 18 bits pour les facteurs de phase. Cette faible augmentation de ressources peut être attribuée à l'usage des BRAM qui reste le même et que ceux-ci contiennent les grandes tables de facteurs de phase.

Dans le cas où N=1024 (série **F**), en plus de faire varier la largeur des facteurs de phase, on a varié la largeur des bus de données en fixant une même valeur pour les interfaces et

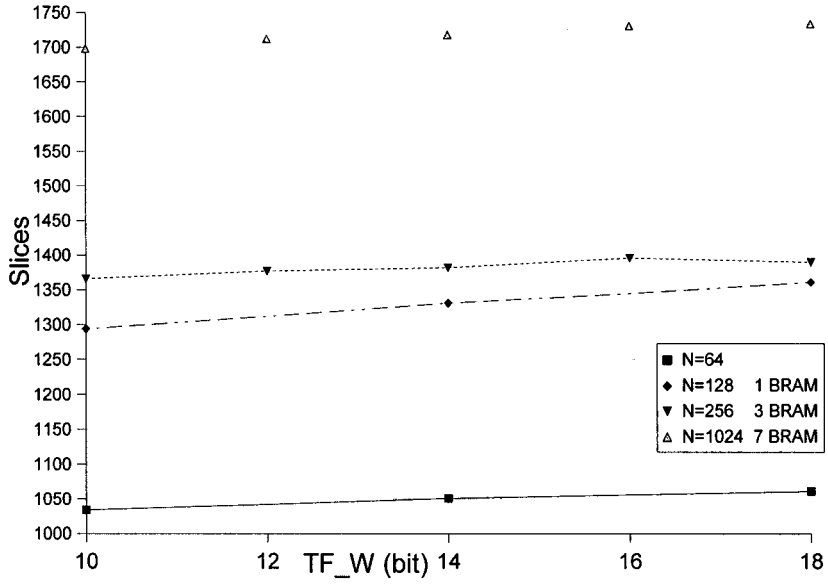


Figure 44 *Slices* pour des FFT de différentes tailles avec des bus de 16 bits en fonction de la largeur des facteurs de phase

les bus internes pour chaque cas de test. Les résultats de SQNR sont illustrés à la figure 45. On peut remarquer que `TF_W`=10 est une valeur trop faible dans le cas où l'on utilise des bus de 16 bits, mais qu'elle est suffisante pour des bus de 10 bits. En prenant des facteurs de phase de la même largeur que les données, leur influence sur le bruit de quantification n'est pas significative. Il est à noter que cette affirmation est aussi validée par la série **C** de la section 4.3. Pour les FFT sans arrondi devant le multiplicateur et ayant en sortie de 18 et 20 bits, les simulations ont été faites avec des largeurs de facteurs de phase de 16 et 18 bits. Au tableau VII, on peut voir que l'écart de performance est faible pour une sortie sur 18 bits, mais qu'il s'intensifie pour une sortie sur 20 bits.

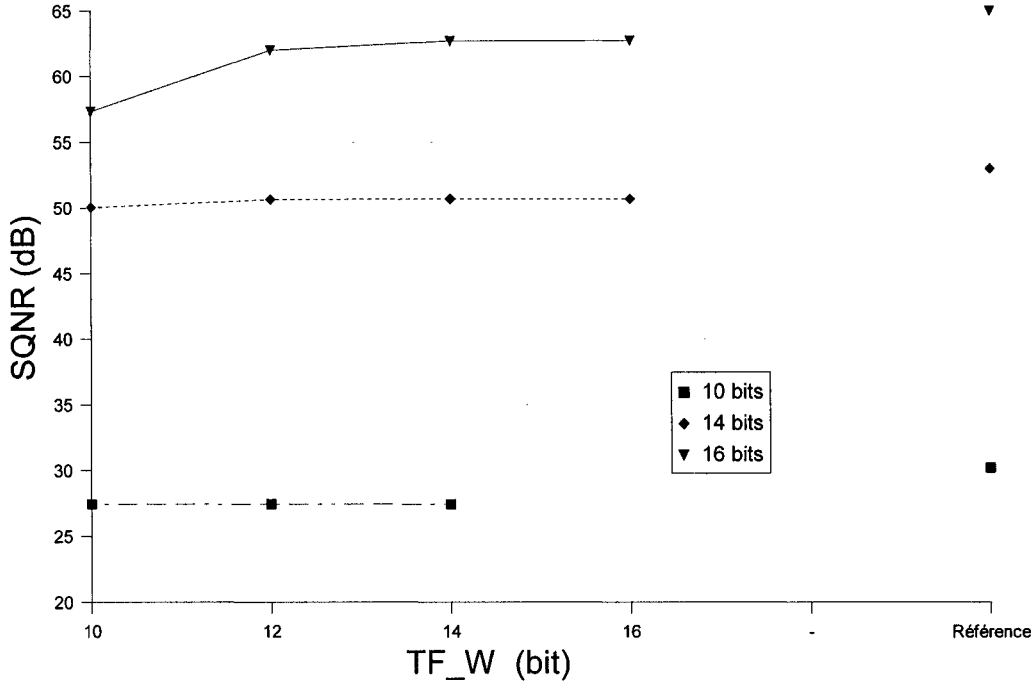


Figure 45 SQNR pour des FFT de 1024 points avec des bus et des facteurs de phase de différentes largeurs

Tableau VII

Effet de la largeur de la sortie et des facteurs de phase sur le SQNR pour des FFT de 256 points de 10 bits en entrée

Largeur en sortie	MULT_ ROUND_W	TF_W	SQNR R2²PC	SQNR référence
18	21	16	86,3	89,1
		18	86,9	
20	23	16	93,7	101,2
		18	98,6	

4.6 Effet de la mise à l'échelle

Afin de vérifier l'effet de la mise à l'échelle sur la précision des résultats, on utilise des vecteurs d'entrées de type II. Ces derniers sont constitués de données aléatoires d'amplitude maximale de $\sqrt{2}/2$ pour la partie réelle et la partie imaginaire. La puissance moyenne par point de la transformée est de 1/3. Bien entendu, le pire cas ne s'est pas produit et aucun débordement n'a corrompu les résultats. Des transformées de taille N = 256, 512 et 1024 points ont été simulées avec des mises à l'échelle variant de N à la plus faible puissance de deux ne causant pas de débordement. Encore une fois, la largeur des bus de données internes a été variée pour atteindre plus de précision.

4.6.1 Scénarios de vérification

Série G

- Taille (`LOG2N`) : 8, 9 et 10
- Largeur en entrée (`IN_FFT_W`) : 10
- Largeur en sortie (`OUT_FFT_W`) : 10
- Largeur des facteurs (`TF_W`) : 12 (effet non dominant sur le bruit);
- Largeur interne (`W`) : valeur constante de 10, 11 et 12 bits
- Mise à l'échelle (`SCALING`) : v à v-4
- Ordonnancement (`BIT_REV_IN`) : les deux, naturel et renversé
- Type (`INVERSE`) : directe
- Multiplicateurs (`MULT_STYLE`) : "*block_mult*"
- Arrondi (`MULT_ROUND_W`) : 18
- 100 vecteurs de type II ayant une amplitude de $\sqrt{2}/2$

4.6.2 Résultats de SQNR

L'effet de la mise à l'échelle peut être observé sur le graphique 46. Les courbes de référence perdent 6 dB de SQNR par division par deux effectuée. Ceci est dû au fait

qu'une division par deux de l'amplitude occasionne une diminution d'un facteur quatre de la puissance du signal. Le SQNR pour les courbes de simulation augmentent à mesure que la mise à l'échelle diminue, mais avec une pente plus faible que 6 dB par un facteur deux. Cet écart à la référence a pour cause la mise à l'échelle qui affecte aussi le bruit de quantification dû aux arrondis. Le modèle de référence effectue une DFT en point flottant et la sortie est arrondie pour une certaine largeur binaire, alors que le bruit de quantification issu de l'arithmétique en point fixe n'est pas présent. Pour le $R2^2PC$, le bruit de quantification se propage d'un étage à l'autre. Dans les étages où il y a une mise à l'échelle, il se fait diviser par deux, mais dans les étages qui ne nécessitent pas de mise à l'échelle, il se propage plus fortement et sa puissance augmente. Pour obtenir des résultats précis, il est avantageux d'effectuer seulement la mise à l'échelle dans le but d'éviter les débordements. Pour les différentes tailles de FFT, on remarque un écart d'environ 3 dB entre les courbes comme ce qui a été observé avec la série **A**.

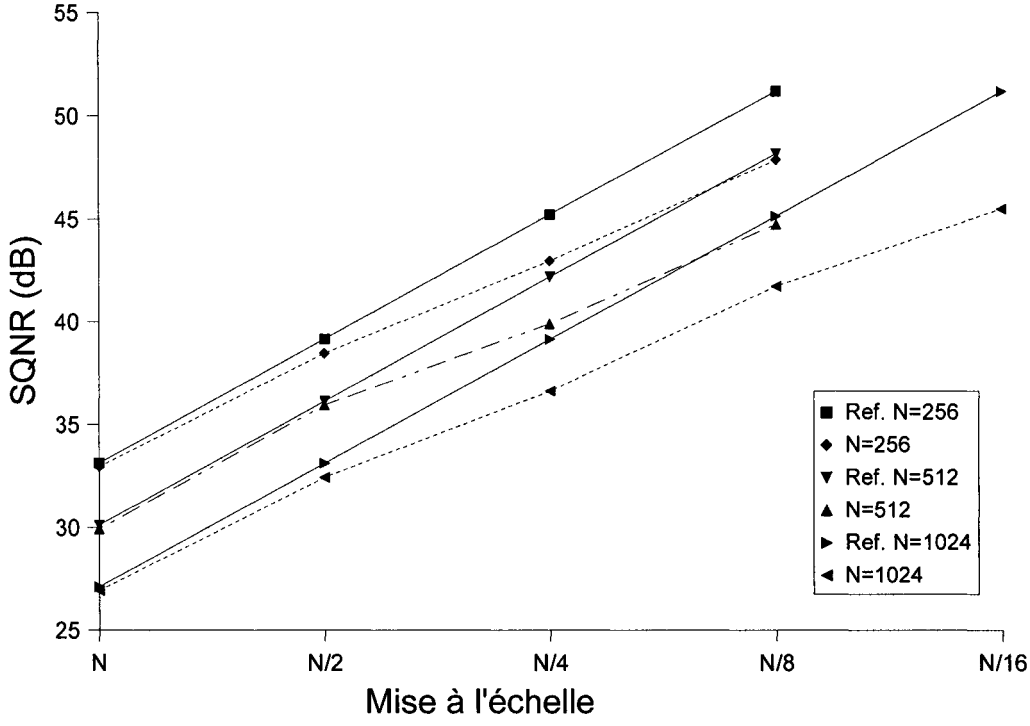


Figure 46 Effet de la mise à l'échelle pour des FFT de différentes tailles avec une entrée aléatoire

Les courbes de la figure 46 ont été simulées avec la largeur interne des bus de données fixée à 12 bits. Les cas de largeur interne de 10 et 11 bits pour une FFT de 1024 points sont présentés à la figure 47. Comme précédemment, on retrouve le phénomène d'augmentation du SQNR avec la diminution de la mise à l'échelle. L'écart à la référence s'accroît avec la diminution de la mise à l'échelle et il est plus prononcé pour une largeur binaire interne plus faible. En effet, l'erreur de quantification qui se propage affecte les bits moins significatifs du signal et sa puissance dépend du poids de ces bits. Lorsque moins de bits sont utilisés, la puissance du bruit augmente pour une même puissance du signal. Cet effet se fait sentir davantage lorsque la mise à l'échelle est faible. En conséquence, pour une mise à l'échelle inférieure à N, la largeur interne devient un paramètre important pour la précision et elle doit être idéalement supérieure à la largeur du port de sortie.

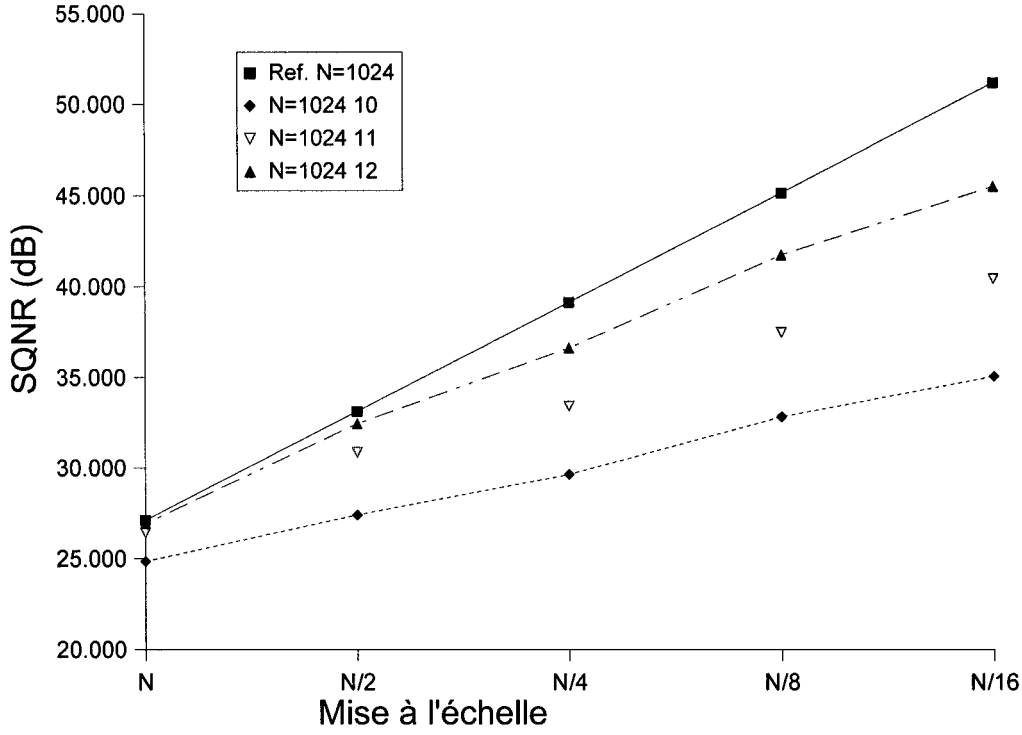


Figure 47 SQNR d'une FFT de 1024 points et de 10 bits d'interface en fonction de la mise à l'échelle et de la largeur interne

4.6.3 Ressources nécessaires

Pour chaque étage du R2^{2}PC qui n'implémente pas de décalage à droite, un bit de moins est nécessaire pour les additionneurs et un bit est sauvé dans le délai en rétroaction. Le premier étage affecté par une mise à l'échelle inférieure à N est celui en sortie. Dans le cas d'une transformée en ordre naturel, les étages affectés ont des courts délais de 1, 2, 4... respectivement de la sortie vers l'entrée. L'effet en terme de réduction des ressources est donc faible. Pour ce qui est des transformées en ordre renversé, les étages concernés ont les plus longs délais contenues dans des BRAM. À la suite du placement et routage, on observe une réduction des ressources de l'ordre de 0 à 15 *slices* pour un facteur deux de mise à l'échelle. Le choix de la mise à l'échelle adéquate a un effet important sur la précision des résultats, toutefois elle influence très peu la complexité matérielle.

4.7 Conclusion

À la suite de toutes ces vérifications pour caractériser le R2^{2}PC en terme de ressource et de précision, on peut relever les points suivants pour nous aider dans le choix des bons paramètres :

- La nature du signal d'entrée n'a pas d'influence importante sur le comportement du SQNR, mais elle influence le besoin de mise à l'échelle.
- Les résultats en terme de SQNR pour l'IFFT sont les mêmes que ceux obtenus pour la FFT. L'IFFT demande légèrement plus de *slices*.
- La transformée en ordre des bits renversé est moins précise que la transformée naturel pour les puissances de deux qui ne sont pas des puissances de quatre.
- Pour une même configuration de largeurs binaires, une augmentation d'un facteur deux de la taille de la transformée cause 3 dB de moins pour le SQNR.
- Les besoins en ressources augmentent de manière logarithmique avec la taille et l'usage de BMULT fait des bonds de 4 unités.
- La mise à l'échelle a pour effet de diminuer le SQNR d'environ 6 dB par facteur deux, lorsqu'elle est non nécessaire pour empêcher le débordement. Elle influence très peu les ressources.
- L'effet des facteurs de phase sur le SQNR peut être négligeable. Ils doivent être de la même largeur que la plus grande largeur interne pour les données.
- Pour augmenter la précision, on peut fixer la largeur de la sortie plus grande que l'entrée. Ainsi, pour chaque augmentation de un bit en sortie, on gagne environ 6 dB de SQNR. Cette relation est valide pour une augmentation de quelques bits. En revanche, la largeur maximale de la sortie qui donne un gain de performance dépend de la largeur à l'entrée, de la taille de la transformée et de la mise à l'échelle.
- La largeur interne est un facteur important à considérer et pour plus de précision, elle doit idéalement être fixée à une valeur égale ou supérieure à la largeur de sortie. Pour une mise à l'échelle de N, un bit supérieur donne un résultat précis, mais pour une mise à l'échelle plus faible, utiliser plus de bits peut améliorer les performances.

L'accroissement d'un bit des largeurs internes augmente l'usage des *slices* d'un faible pourcentage (3% à 8%).

Les objectifs de concevoir, de valider et de caractériser un module de FFT reconfigurable qui demande relativement peu de ressources matérielle et qui peut traiter un flot continu de données sérielle ont été atteints.

CHAPITRE 5

COMPARAISON AVEC DES *CORES* COMERCIAUX

5.1 Aperçu des *cores* disponibles

Dans la suite de ce travail, le terme *core* sera utilisé pour représenté les modules de FFT/IFFT sont offerts sur le marché. La plupart des *cores* sont conçus pour une implémentation dans un FPGA, nous nous limiterons ici au survol de ceux-ci. Dans cette première section, les *Logicores* de Xilinx ont volontairement été exclus, ils seront traités à la section suivante. La majorité des *cores* commerciaux de FFT sont habituellement de taille fixe et possèdent très peu de paramètres de configuration. Dans le tableau VIII, on trouve un aperçu de ce qui est disponible sur le marché pour des FFT de 1024 points, la taille la plus répandue pour les *cores*. Un tableau semblable pourrait être fait pour pratiquement toutes les puissances de deux, jusqu'à 8192, toutefois ces tableaux n'apporteraient rien de plus à la discussion.

Tableau VIII

Aperçu des *cores* commerciaux de FFT de 1024 points

Fournisseur / Nom du *core*	**FPGA cible et Ressources**	**Cycles d'horloge par transformée**	**Commentaires**
HDL Design House [a] HMC-FFT1K-R4 Note: les parenthèses correspondent aux tampons E/S facultatifs.	Virtex2 Slices: 1800 BRAM: 6+(8) BMULT: 9	(1024) + 1024+16 + (1024)	• E/S complexes de 16 bits • Entrée en ordre naturel avec le tampon d'entrée, entrée en ordre renversé si le tampon d'entrée n'est pas présent. • FFT/IFFT

Tableau VIII (suite)

Fournisseur / Nom du *core*	**FPGA cible et Ressources**	**Cycles d'horloge par transformée**	**Commentaires**
Comit Systems [b] 1024 Point FFT Core 1.0	Virtex2 Slices: 1536 BRAM: 6 BMULT: oui	6200 (approx.)	• E/S complexes de 16 bits • E/S en ordre naturel
Pantek [c] GateFlow IP-Core 4954-401	Virtex2 Pro-6 LUT: 12 702 FF: 11 410 BRAM: 40 BMULT: 64	256 1,83 μs @140 MHz	• Quatre E/S complexes de 16 bits par cycles. • Fenêtre de Hanning au choix, recouvrement possible de 50% ou 75%. • E/S en ordre naturel.
SiWorks [d] SCPFFT Parallel N point FFT/IFFT	Virtex2 une seule entrée Slices: 5384 BRAM: 7 BMULT: 14	1024 + 11 10,24 μs @100 MHz	Le cas illustré correspond à N=1024 avec une seule entrée de 16 bits complexe. • Possibilité de une à 16 entrées de données en parallèles. • E/S de largeur ajustable. • N ajustable. • FFT/IFFT.

Tableau VIII (suite)

Fournisseur / Nom du *core*	**FPGA cible et Ressources**	**Cycles d'horloge par transformée**	**Commentaires**
RF Engines [e] Vectis PFFT	Virtex2 Pro-6 Slices: 2833 BRAM: 12 BMULT: 8	5216 Fonctionne à 222.3 Mhz, mais il y a l'entrelacement des parties réelle et imaginaires, la fréquence des échantillons complexes n'est que de la moitiée.	Le cas illustré correspond à N=1024 avec l'entrée à 12 bits, les facteurs de phase et la sortie à 17 bits, un tampon à l'entrée et une fonction d'ordonnancement à la sortie. • N possible jusqu'à 128K choisi en commandant le *core*. • E/S et facteur de phase de largeur choisi en commandant le *core*. • FFT/IFFT.
Dillon Engineering [f] DE FFT Core	Virtex2 Slices: 4408 BRAM: 48 BMULT: 32	2048	Le cas illustré correspond à N=1024 avec une seule entrée complexe de 18 bits. • Possibilité de une à quatre entrées de données en parallèles. • E/S de largeur ajustable. • N ajustable. • FFT/IFFT. • Fenêtre de Hanning intégrée. • E/S en ordre naturel. • Représentation des nombres en point flottant disponible.

[a] http://www.hdl-dh.com
[b] http://www.comit.com
[c] http://www.pentek.com
[d] http://www.siworks.com
[e] http://www.rfel.com
[f] http://www.dilloneng.com

Toutes les données présentées dans le tableau VIII proviennent des fiches techniques des produits. L'implémentation diffère beaucoup d'une FFT à l'autre, par exemple on

retrouve des papillons radix-2, radix-4 interconnectés en parallèle, en série ou selon une architecture mixte. De plus, chaque fournisseur a une manière différente de présenter son produit et les performances de celui-ci. Il devient donc difficile de faire une comparaison précise d'un design à l'autre. Les deux premiers *cores* exposés ont une utilisation des ressources semblable à celle du R2^{2}PC, qui est de 1590 slices, 7 BRAM et 16 BMULT pour des largeurs de mots binaires de 16 bits. Celui de HDL Design House traite un flot continu de données comme le R2^{2}PC. En revanche, il ne possède pratiquement pas de reconfigurabilité, les seuls choix offerts portent sur l'utilisation de tampons d'entrée et de sortie facultatifs et sur la possibilité de faire une IFFT. Le *core* de Comit Systems reçoit les 1024 échantillons en rafale et produit les résultats aussi en rafale, puisqu'une FFT sur 1024 points prend environ 6200 cycles. Il est le seul *core* à ne pas pouvoir traiter une séquence continue d'échantillons. De plus, la fiche technique de ce dernier spécifie que des multiplicateurs dédiés sont utilisés, mais il n'est pas fait mention de leur nombre.

Tous les autres *cores* exposés peuvent fonctionner avec un flot continu de données, même que certains peuvent être configurés avec des entrées multiports permettant de traiter une transformée de 1024 points en moins de 1024 cycles. Pour des fins de comparaison, ces derniers ont été configurés pour n'utiliser qu'un seul port. L'utilisation de plusieurs chemins de données en parallèle occasionne une importante augmentation des ressources afin d'accélérer le traitement. En effet, le *core* de Pentek ayant une entrée de quatre ports, reçoit et produit quatre échantillons par cycle d'horloge. Avec deux registres et deux LUT présents dans chaque *slice*, il utilise sûrement plus de 6500 *slices* ($\approx$ max(LUT, FF)/2) et demande 104 ressources embarquées pour un traitement quatre fois plus rapide que le R2^{2}PC. Les ressources nécessaires dépassent d'environ quatre fois celles du R2^{2}PC. Pour les *cores* de SiWorks et de RF Engines, l'utilisation des ressources du FPGA est largement supérieure à celle nécessaire au R2^{2}PC.

Les fiches techniques des produits ne font jamais mention de la précision des résultats. En revanche, la majorité des fournisseurs offrent des modèles de simulation Matlab ou

C++ ayant le même comportement binaire que le *core*. Ces modèles n'ont pu être testés, car ils ne sont disponibles qu'à l'achat du module. Un mécanisme de mise à l'échelle et de détection du débordement est inclus dans la majorité des produits.

5.2 Test des *Logicores* de Xilinx

Le fabricant de FPGA Xilinx offre tout genre de *core* dans sa librairie *Logicore* accessible à l'aide de l'outil CORE Generator. Ce sont aussi ces modules qui sont instanciés dans le FPGA lorsque le *toolbox* de Simulink, Xilinx System Generator, est utilisé pour la conception. Parmi les L*ogicores* on retrouve plusieurs modules de FFT, certains sont reconfigurables et d'autres sont à paramètres fixes. Cette section porte sur les ressources matérielles requises par ces *cores* optimisés pour les Virtex et Virtex-II ainsi que sur la précision des résultats.

Le fabricant de FPGA Altera propose aussi des *cores* de FFT dont le plus récent se nomme FFT MegaCore Function 2.0 (Altera, 2004). Il est réputé très rapide, demandant peu de puissance et il possède plusieurs paramètres configurables. En revanche, dans les sections suivantes, nous limiterons notre analyse aux produits de Xilinx, puisque nous avons des données quantitatives permettant de les comparer au $R2^2PC$.

5.2.1 Présentation des *Logicores* de FFT et des ressources utilisées

La majorité des *Logicores* de FFT permettent de réaliser au choix des FFT ou IFFT de taille fixe. Dans le cas où la taille est une puissance de quatre, les tailles 16, 64, 256 et 1024 sont disponibles sous le nom de produit `vfft`*N*`v2`. Une architecture à réutilisation du papillon radix-4 est employée et la largeur de l'entrée, de la sortie et des facteurs de phase est fixée à 16 bits. Pour les transformées avec N = 64, 256 ou 1024, trois configurations de mémoire sont disponibles. On utilise soit un ou deux tampons de mémoire de N échantillons pour avoir des entrées et sorties par rafale ou trois tampons

pour avoir des entrées et sorties continues. La configuration utilisant trois tampons est celle qui sera retenue dans le tableau IX et est illustrée à la figure 48. Le flot continu d'échantillons doit être au rythme d'un par cycle pour N=16, d'un par 3 cycles pour N= 64 ou 256 et d'un par 4 cycles pour N=1024. Les tampons permettent la parallélisation et la resérialisation des données. Ces étapes ajoutent de la latence et sont non nécessaires dans le cas du R2^{2}PC. Les échantillons des entrées et des sorties utilisent l'ordonnancement naturel. Ces *cores* sont conçus pour être implémentés dans la gamme de FPGA Virtex qui ne possède pas de multiplicateurs BMULT, c'est pourquoi cette ressource n'est pas utilisée.

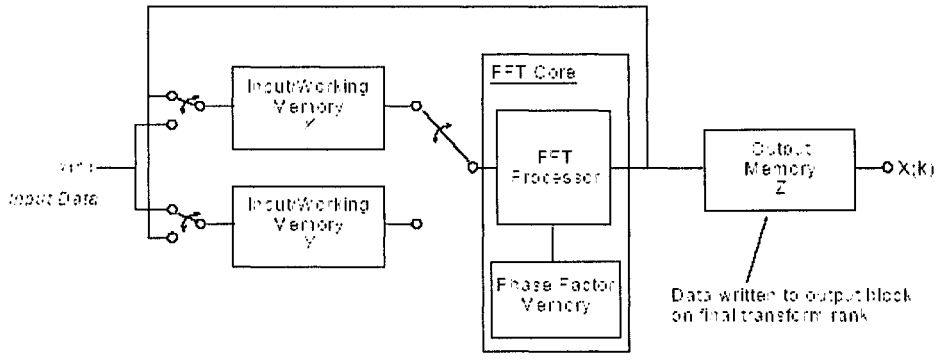


Figure 48 Schéma bloc du *Logicore* `vfft`*N*`v2` utilisant trois tampons de mémoire (tiré de la fiche technique(Xilinx, 2004))

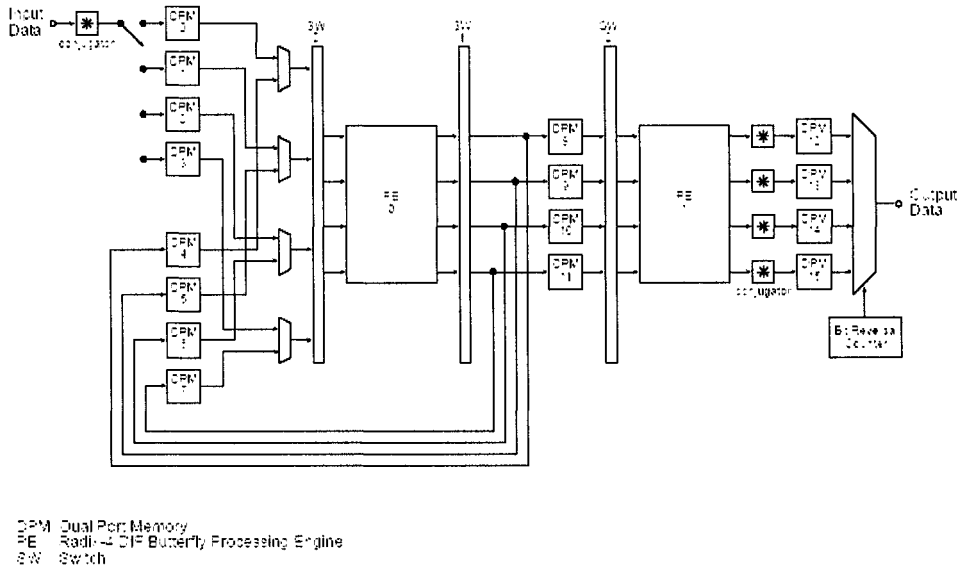


Figure 49 Schéma bloc du *Logicore* `xfft1024`(tiré de la fiche technique (Xilinx, 2004))

La librairie *Logicore* offre aussi un module configurable au niveau de la taille: le `xfft1024` (Szedo, Yang, & Dick, 2001). Lorsque la structure de la figure 49 est implémentée, l'utilisateur a le choix de faire une FFT ou une IFFT sur 64, 256 ou 1024 points. L'architecture de type réutilisation utilise 16 mémoires à double ports réparties en quatre banques et deux processeurs radix-4 (PE sur la figure 49) dont un seul contient trois multiplicateurs complexes. Les entrées, les sorties et les facteurs de phase ont tous des largeurs binaires de 16 bits. L'ordonnancement en entrée et en sortie est de type naturel. L'usager peut configurer la mise à l'échelle qui est effectuée progressivement à chaque passage dans les processeurs.

Présentons maintenant un autre module pour lequel *s*eul N=32 existe comme taille de transformée. Le *Logicore* `vfft32` utilise une architecture parallèle de type radix-2. Des paramètres de configuration permettent de choisir la largeur de l'entrée et de la sortie entre 2 et 31 bits et des facteurs de phase entre 4 et 31 bits. Ce *core* peut utiliser au choix les multiplicateurs embarqués ou des multiplicateurs utilisant la logique des *slices*.

Avec la version 6.1i de Xilinx CORE Generator, un *Logicore* de FFT possédant beaucoup de reconfigurabilité a été mis à la disposition des concepteurs pour FPGA. On le retrouve sous le nom `Fast Fourier Transform v2.1` ou `xfft` et il se veut le remplaçant des autres *Logicores* de FFT. Ce module reconfigurable peut avoir trois architectures possibles répondant aux différents compromis temps/ressources.

- La première configuration permet des entrées et sorties continues à raison d'une donnée par cycle. L'architecture utilise deux papillons de type radix-4, comme illustré par la figure 50. Des tailles de 64 à 8192 peuvent être effectuées.
- La deuxième architecture utilise un seul papillon radix-4. Elle reçoit ses échantillons d'entrée en rafale et produit sa sortie de la même manière. Plus de temps est donc nécessaire pour effectuer une transformée, mais moins de ressources sont requises. Des tailles de 64 à 16 384 peuvent être effectuées par cette architecture. L'usager a le choix entre deux représentations des nombres: soit le point fixe ou soit le point flottant par bloc (un seul exposant pour N résultats).
- La troisième architecture utilise un seul papillon radix-2 pour un minimum de ressources. Elle permet des transformées de 16 à 1024 points et elle traite les données en rafale.

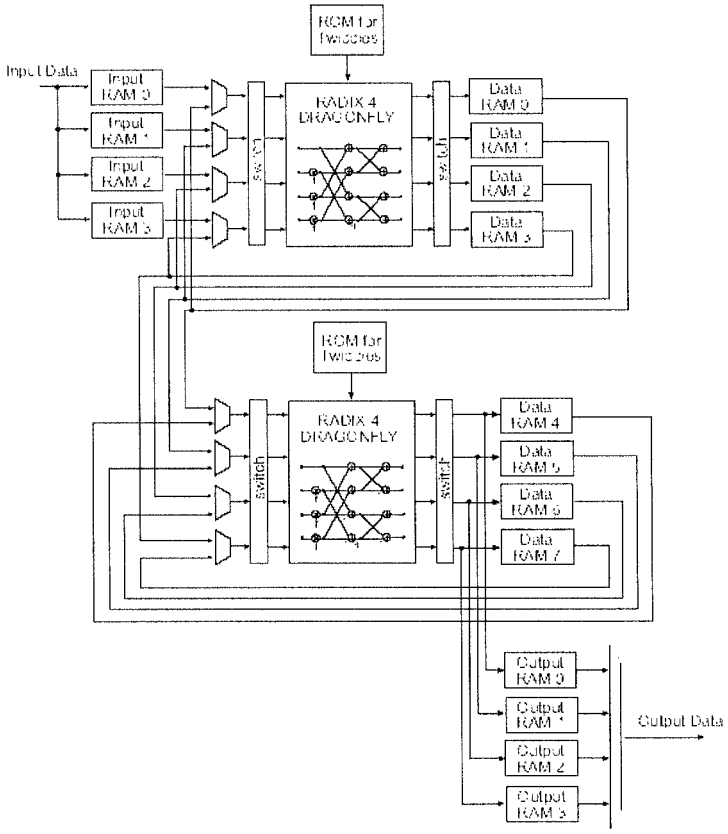


Figure 50 Schéma bloc du *Logicore* `xfft` traitant un flot continu de données (tiré de la fiche technique (Xilinx, 2004))

L'architecture du `xfft` qui sera utilisée aux fins de comparaison avec le R2^{2}PC, est la première architecture, celle qui traite un flot continu de données. Les paramètres de configuration disponibles sont les suivants:

- taille $N = 2^v$, v= 6 à 13;
- largeur des facteurs de phase 8, 12, 16, 20 ou 24 bits;
- largeur des données 8, 12, 16, 20 ou 24 bits;
- arithmétique sans mise à l'échelle (largeur de bus croissante) ou avec mise à l'échelle (largeur de bus fixe);
- arrondi convergent ou troncation à la suite du papillon;
- choix à l'exécution entre FFT ou IFFT;

- choix à l'exécution de la taille de la transformée, les tailles inférieures à la taille de base sont possibles.
- choix à l'exécution de la séquence de mise à l'échelle (division par 0, 2, 4 ou 8) à chaque passage dans le papillon;
- utilisation de BRAM ou de RAM distribuée.

L'usager a le choix d'utiliser l'arrondi convergent ou la troncation pour maintenir la largeur binaire fixe, puisque la mémoire est utilisée sur place. L'arrondi convergent consiste à alterner l'arrondi à la hausse et à la baisse quand le point milieu est rencontré. La troncation, arrondi systématiquement à la baisse, occasionne un biais, mais ne nécessite pas de ressources. Un biais est aussi introduit dans le cas d'une addition suivie d'une mise à l'échelle par deux et d'un arrondi standard, vu que le point milieu a la probabilité de se produire la moitié du temps (Kabal & Sayar, 1986 ; Meyer, 1989). Dans le cas où un mot doit être réduit de plusieurs bits, la différence entre l'arrondi convergent et l'arrondi standard est négligeable.

Tableau IX

Comparaison des *Logicores* de FFT versus le R2²PC utilisant des bus de 16 bits

N	*Core*	*Slices*	BMULT	BRAM	Latence (cycle)
16	R2²PC	536	4	0	25
	vfft16v2	1355	0	0	82
32	R2²PC	797	8	0	46
	vfft32	2066	3	0	96
64	R2²PC	990	8	0	79
	vfft64v2	1161*	0	3	192
	xfft T	2480	9	16	ND
	xfft R	3120	9	16	ND
256	R2²PC	1304	12	3	277
	vfft256v2	1616*	0	3	768
	xfft T	2472	9	19	483
	xfft R	3112	9	19	485
1024	R2²PC	1590	16	7	1051
	vfft1024v2	1869*	0	6	4096
	xfft1024	2763	12	22	2816
	xfft T	2512	9	19	1827
	xfft R	3152	9	19	1829
2048	R2²PC	1663	20	11	2080
	xfft T	3172	18	22	ND
	xfft R	3812	18	22	ND
4096	R2²PC	1779	20	21	4129
	xfft T	3192	18	38	ND
	xfft R	3832	18	38	ND

ND: Valeur non disponible
* Valeur provenant de la fiche technique, elle exclut les *slices* pour deux multiplexeurs 2 à 1 de 16 bits.

Le tableau IX résume les différentes ressources utilisées par les *Logicores* et les compare à des configurations semblables du R2^{2}PC. Pour toutes les tailles de FFT illustrées, on remarque que le R2^{2}PC utilise moins de *slices* que les *Logicores*. Comparons d'abord la série des `vfft`*N*`v`. Ces *cores* utilisent des quantités semblables de BRAM et sont avantagés par leur absence de BMULT. Cependant, ils sont désavantagés par leur plus grande latence et par le rythme plus lent d'entrée des données (voir section 5.2.1). Ces deux points sont probablement interreliés. En effet, avec une fréquence d'arrivée des données plus faible que la fréquence d'horloge, le même multiplicateur peut être réutilisé pour traiter plus d'un échantillon permettant ainsi de diminuer les ressources. Il est aussi possible qu'aucun multiplicateur ne soit employé dans cette architecture, mais qu'un module de type CORDIC se charge de faire la rotation de phase.

En ce qui concerne le `xfft1024`, 74% plus de *slices* sont requis et 22 BRAM au lieu de sept sont utilisés. Cette grande différence est due au choix initial de l'architecture de FFT. Celle du `xfft1024` en est une de type réutilisation nécessitant quatre tampons de taille *N* à quatre ports, comparativement à *N* espaces mémoire répartis dans des lignes à délai pour le R2^{2}PC. La différence en BMULT est plus faible et elle est à l'avantage du `xfft1024` par quatre multiplicateurs. Pour une architecture à réutilisation du papillon radix-4, la taille 1024 points est un cas particulier où un seul des deux papillons (`PE0` sur la figure 49) requière trois multiplicateurs complexes.

Comparons maintenant le `xfft` traitant un flot continu de données avec le R2^{2}PC. La flexibilité au niveau de la taille du `xfft` ne s'accompagne pas d'une augmentation régulière des besoins en ressources. Cet aspect est mis en relief par la figure 51. On remarque que de *N*=64 à *N*=1024, sensiblement la même quantité de ressources est employée. Le besoin en *slices* est largement supérieur à celui requis par le R2^{2}PC. En revanche, on fait usage de seulement neuf BMULT par papillon radix-4. Chacun des trois multiplicateurs complexes requis est constitué d'uniquement trois multiplicateurs dédiés. Pour ce faire, les entrées complexes du multiplicateur A+Bj et C+Dj sont assemblées selon l'équation 5.1. Cette technique requière trois multiplicateurs et cinq

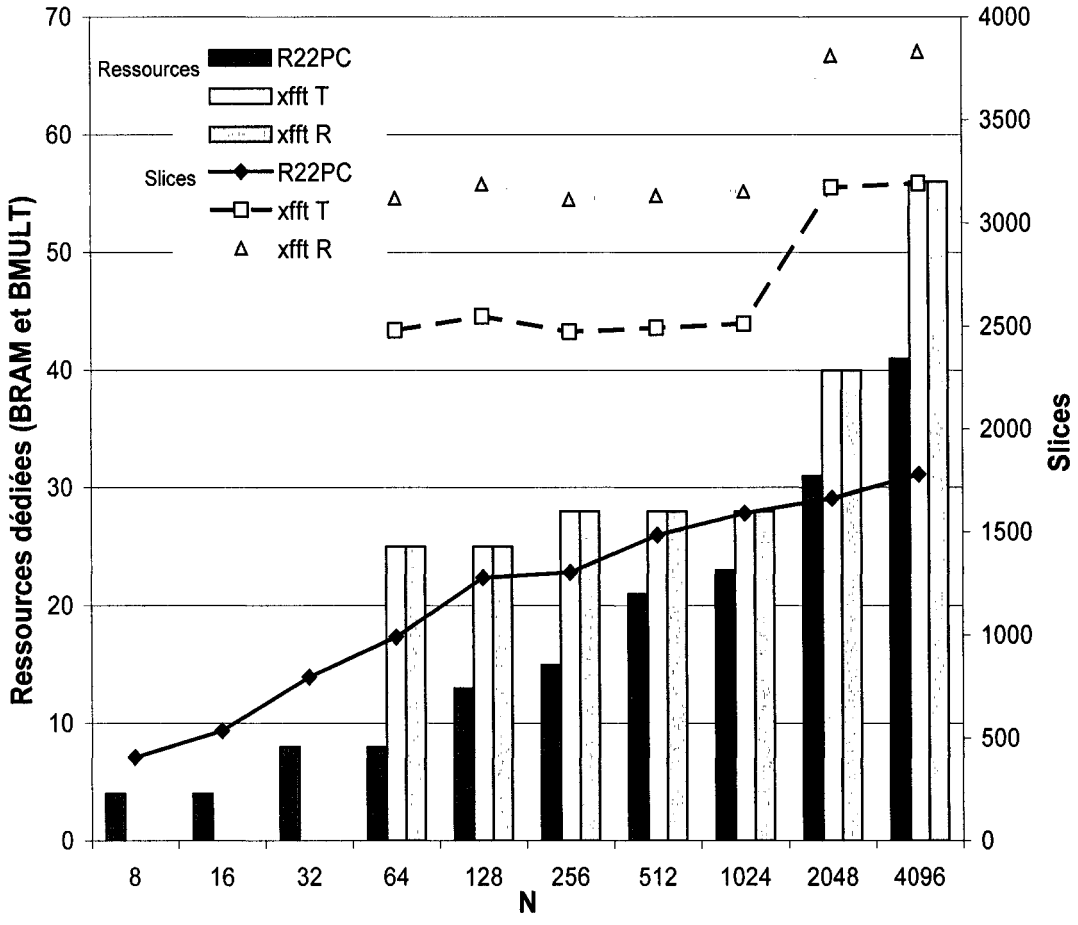


Figure 51 Comparaison des ressources FPGA requises par le R2^{2}PC et par le `xfft` pour différentes tailles de FFT ayant des largeurs binaires de 16 bits

$$\begin{aligned} &\text{Partie réelle}: AC-BD \\ &\text{Partie imaginaire}: (A+B)(C+D)-AC-BD \end{aligned} \tag{5.1}$$

Pour les tailles supérieures à 1024, deux papillons utilisant des multiplicateurs complexes sont requis, d'où les 18 BMULT nécessaires. Cette valeur se rapproche des 20 BMULT nécessaires au R2^{2}PC pour les mêmes cas. Pour toutes les tailles, le nombre

Pour les tailles supérieures à 1024, deux papillons utilisant des multiplicateurs complexes sont requis, d'où les 18 BMULT nécessaires. Cette valeur se rapproche des 20 BMULT nécessaires au R2^{2}PC pour les mêmes cas. Pour toutes les tailles, le nombre total de ressources dédiées (BRAM+BMULT) est supérieur dans le cas `xfft`. L'écart le plus faible en nombre de ressources dédiées se trouve de sept ressources pour N=1024.

La latence du `xfft` n'est pas spécifiée dans la fiche technique, il est seulement précisé qu'un délai est causé par les tampons de mémoire, le traitement et le pipeline. C'est pourquoi le tableau IX présente seulement la latence pour N=256 et N=1024, les deux seuls cas simulés. On peut déduire que le délai entre l'activation du signal *start* et la montée du signal *done* correspondant est de $1{,}75N + 35$ cycles pour la transformée avec troncation, comparativement à N plus quelques cycles pour le R2^{2}PC. Le choix entre l'arrondi convergent et la troncation a un impact important sur le nombre de *slices*. On utilise 640 *slices* supplémentaires lorsque l'arrondi convergent est effectué et 2 cycles de latence sont aussi ajoutés. Le gain en performance occasionné par ce choix est détaillé à la section 5.2.2.

5.2.2 Précision des résultats

Une vérification comparative a été effectué au sujet du bruit de quantification. Les fiches techniques des `vfft`*N*`v2` n'indiquent pas de performance en terme de SQNR, mais un modèle VHDL comportemental est disponible et permet de simuler ces *cores*. Le *core* `xfft` contient un aperçu des performances de précision des résultats suite au test nommé *slot noise test*. Dans cette section, seules quelques simulations ont été effectuées, mais elles sont suffisantes pour déduire la tendance générale. D'abord, présentons les performances des *Logicores* face au même banc d'essai qui a servi à vérifier le R2^{2}PC et qui est présenté à la section 4.1.1.1. Ensuite, exposons les performances du R2^{2}PC soumis au *slot noise test*.

5.2.2.1 SQNR

La vérification du SQNR avec 100 vecteurs sinusoïdes complexes de différentes fréquences échantillonnées en N points a été utilisée. Tout comme pour le R2^{2}PC, une mise à l'échelle de N est employée pour le `xfft`. Elle est effectuée par une mise à l'échelle de quatre à chaque passage dans le papillon radix-4, séquence 2-2-2-2-2. Les résultats obtenus pour les modules avec des entrées, sorties et facteurs de phase de 16 bits sont exposés au tableau X. On remarque que l'arrondi convergent donne des résultats à moins de 2 dB de la référence et performe mieux que la troncation d'environ 7 dB. Le R2^{2}PC a un SQNR de moins de 1 dB inférieur au `xfft` avec arrondi. La méthode de réduction de la largeur binaire à tous les deux étages employée pour le R2^{2}PC se compare avantageusement à de la simple troncation et s'approche d'une méthode d'arrondi systématique. À partir du résultat pour N=1024, on peut déduire que les `vfft`N`v2` utilisent la méthode de troncation.

Tableau X

Comparaison des résultats de SQNR du R2^{2}PC avec les Logicores pour des largeurs binaires de 16 bits

N	Module	SQNR (dB)	SQNR Référence (dB)
256	R2^{2}PC	68.75	71,00
	`xfft T`	62,38	
	`xfft R`	69,66	
1024	R2^{2}PC	62,72	64,97
	`vfft1024v2`	56,02	
	`xfft T`	56,22	
	`xfft R`	63,56	

D'autres vérifications ont été réalisées sur le `xfft` pour voir l'influence de certains paramètres sur la précision des résultats. Les différents résultats sont résumés dans le tableau VIII. Le séquencement de la mise à l'échelle peut être effectué en spécifiant le nombre de bits à décaler à gauche à chaque passage dans le papillon radix-4, pour un maximum de trois bits. Ainsi, la séquence 2-2-2-2-2 indique qu'une division par quatre doit être effectuée à chaque passage. Ce paramètre a un effet important sur le SQNR. En effet, le pire cas de séquencement est d'effectuer tous les décalages au début, soit 3-3-3-1-0. Le résultat obtenu dans le dernier cas (55 dB) est de 1 dB inférieur à celui obtenu avec la troncation. La fiche technique propose la séquence 2-3-2-2-2 pour s'assurer qu'aucun débordement ne se produise. Ici, nous avons testé 2-3-2-2-1 pour maintenir la mise à l'échelle totale à la valeur *N*. Avec cette séquence, nous avons varié la largeur des facteurs de phase. On remarque au tableau VIII, que ce paramètre n'a pas un effet dominant sur le bruit de quantification et qu'un plafond est atteint pour une certaine largeur. Le même phénomène se produit aussi pour le R2^{2}PC

Tableau XI

Effet de la variation de la taille des facteurs de phase et de la séquence de mise à l'échelle pour le `xfft` de 1024 points avec arrondi

Largeur des facteurs de phase (bit)	**Séquence de mise à l'échelle**	**SQNR (dB)**
12	2-3-2-2-1	60,60
16	2-2-2-2-2	63,65
	2-3-2-2-1	61,14
	3-3-3-1-0	55,02
20	2-3-2-2-1	61.14

5.2.2.2 Slot noise test

La fiche technique du `xfft` donne certains résultats de *slot noise test* sous forme de figures montrant le spectre de puissance en fonction du numéro du point de la FFT. Le *slot noise test* n'est que très brièvement décrit, mais on tentera quand même de le recréer dans cette section. Le test appliqué au R2^{2}PC est illustré par le schéma bloc de la figure 52. D'abord, on génère du bruit gaussien complexe à moyenne nulle que l'on fait passer par une DFT pour en obtenir le spectre sur N points. Ce spectre contient de la puissance à toutes les fréquences. Ensuite, on force une section de ce spectre à zéro (50 points pour N=1024). Après être retourné dans le domaine temporel par une IDFT, on quantifie le signal sur un certain nombre de bits de manière à ce qu'il occupe la plage dynamique au maximum. Ce signal sert d'entrée à la FFT de référence de Matlab et au *core* testé, dans ce cas ci le R2^{2}PC. Vu la quantification, le spectre est différent de zéro au fond de la fente. Enfin, on moyenne les puissances en chaque point des spectres de sortie des deux transformées pour 100 vecteurs aléatoires distincts. On peut ainsi faire tracer des figures de même nature que celles provenant de la fiche technique du `xfft`. L'utilisation de la FFT de Matlab permet d'obtenir la plage dynamique à l'entrée du *core*. Les tests

effectués sur le R2^{2}PC permettent de déterminer la plage dynamique après les étapes de quantification et de mise à l'échelle.

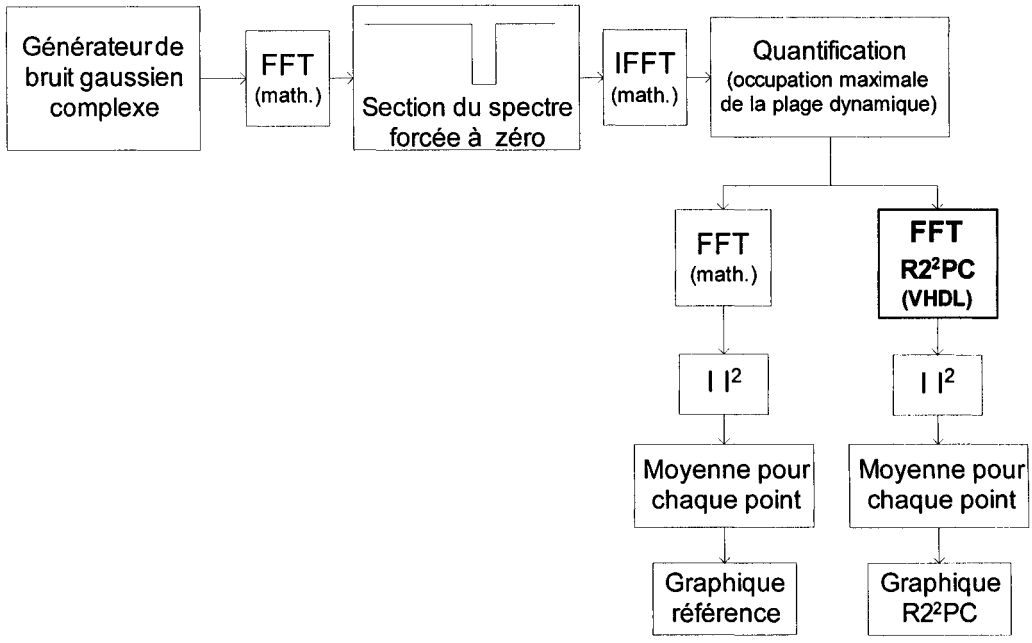
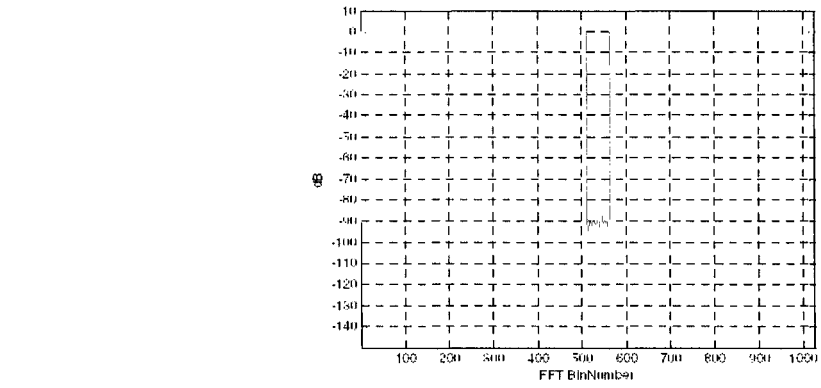
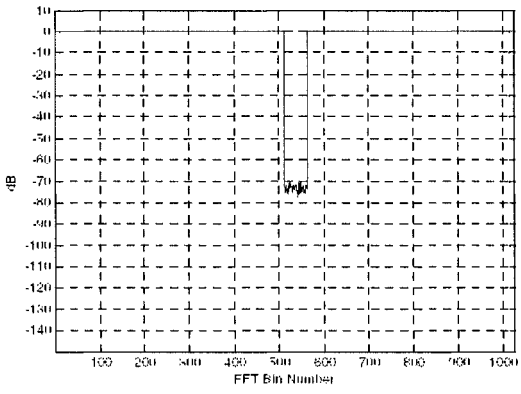


Figure 52 Diagramme bloc du *slot noise test*

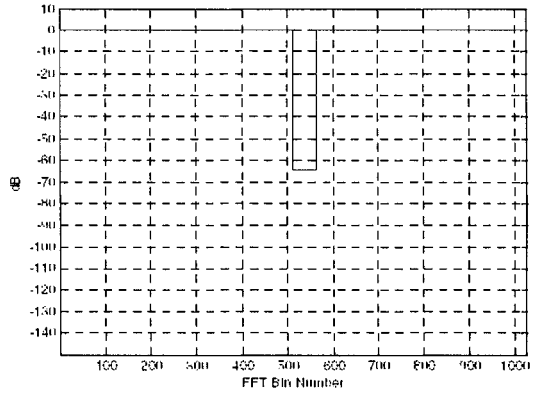
Dans la fiche technique, les résultats pour le `xfft` configuré avec N=1024, des largeurs à 16 bits et avec l'arrondi convergent sont fournis. Sur les graphiques de la figure 53, on peut estimer que la profondeur de la fente du rapport de puissance théorique est à environ de -92 dB, celle du *core* utilisant le point flottant par bloc aux environs de -72 dB et celle du *core* avec mise à l'échelle de $1/N$ à -64 dB.



a) Input Data - 16 bits



b) Block Floating Point Arithmetic



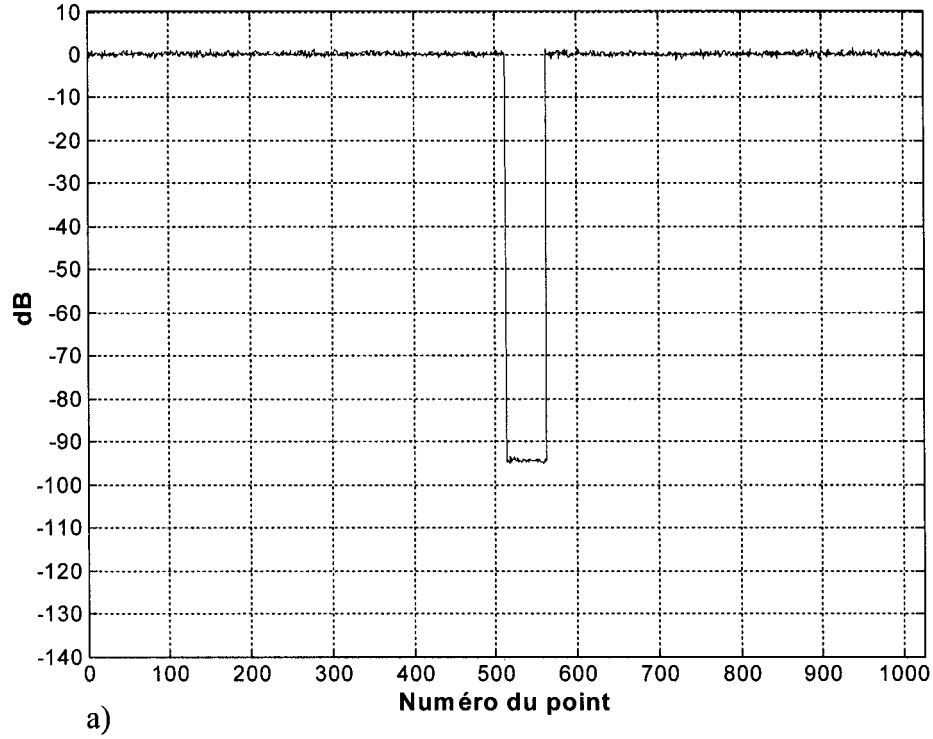
c) Scaled (scaling of 1/N) Arithmetic

Figure 53 Résultats graphiques du *slot noise test* pour le `xfft` de 1024 points et de 16 bits: **a)** fente théorique, **b)** fente du *core* utilisant la représentation des nombres en point flottant par bloc (mise à l'échelle adéquate) et **c)** fente du *core* utilisant le point fixe avec une mise à l'échelle de 1/*N*.

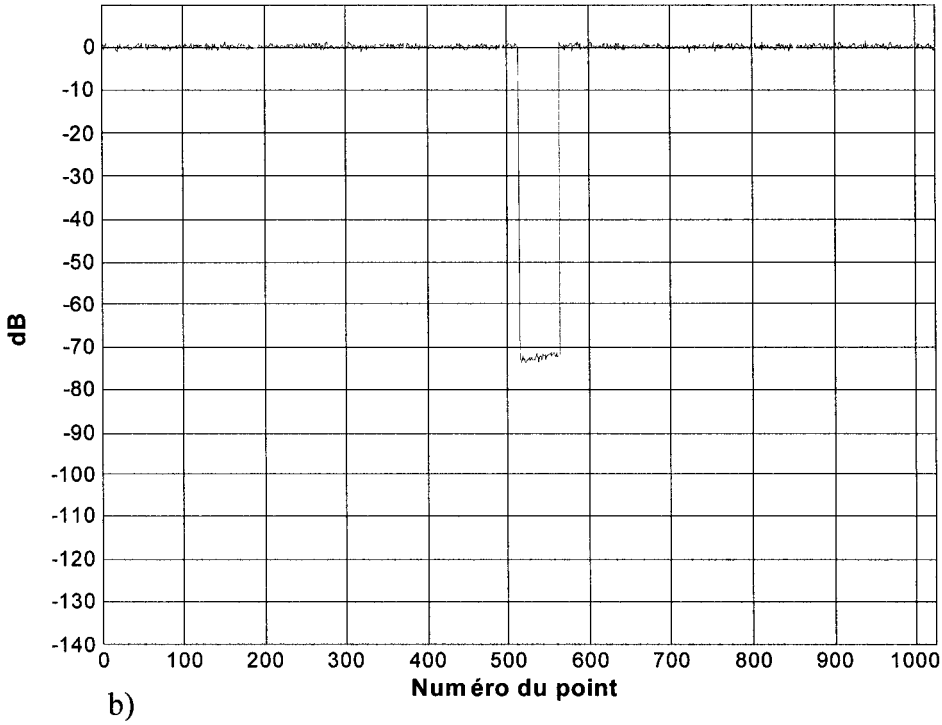
Le calcul de la puissance moyenne en chaque point a été fait et ces valeurs ont été converties en dB pour obtenir les résultats graphiques du R2^{2}PC. Empiriquement, la mise à l'échelle maximale qui ne provoque pas de débordement du R2^{2}PC est de 128 pour un groupe de 100 vecteurs aléatoires. Si les vecteurs d'entrée sont de nature semblable à ceux utilisés pour le `xfft`, mettre la mise à l'échelle au minimum peut être comparé avec l'utilisation de la représentation des nombres en point flottant par bloc.

La figure 54 nous montre certains résultats obtenus avec le R2^{2}PC. Ces derniers sont graphiquement semblables à ceux du `xfft`. La profondeur moyenne de la fente issue de la FFT de référence est de -94 dB et celle du *core* R2^{2}PC avec une mise à l'échelle de 128 de -72 dB. L'écart entre la profondeur des fentes est de 22 dB pour le R2^{2}PC comparativement à environ 20 dB pour le `xfft`. Pour ce cas de *slot noise test* le R2^{2}PC semble légèrement mieux performer que le `xfft` ce qui est en désaccord avec la vérification du SQNR. En revanche, plus de série de vecteurs ont pu être employés et leur nature diffère peut-être. Le fait que les les fentes des graphiques de référence ne soient pas au même niveau, réduit la validité des résultats.

Le *slot noise test* effectué est difficile à interpréter en ce qui concerne l'effet de la mise à l'échelle. Pour le scénario où une mise à l'échelle de 128 est amplement suffisante, en faire une de 512 provoque une fente d'environ -65 dB et avec une division par 1024 on obtient une fente de moyenne -72 dB. Cette dernière valeur est la même que celle trouvée pour 128. En revanche, l'allure graphique des deux tests diffère beaucoup. En effet, vu la trop forte mise à l'échelle du dernier cas, la fente se retrouve pour la majorité des points à la valeur zéro, valeur exacte. La puissance en ces points devient donc aussi zéro qui une fois mis en dB devient moins l'infini et est non représentable graphiquement. Ceci explique probablement la droite tracée au fond de la fente de la figure 53 c) qui provient de la fiche technique, mais n'explique pas nécessairement la profondeur de cette fente.


10
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
-130
-140
dB
0
100
200
300
400
500
600
700
800
900
1000
Numéro du point

a)


0
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
-130
-140
dB
0
100
200
300
400
500
600
700
800
900
1000
Numéro du point

b)

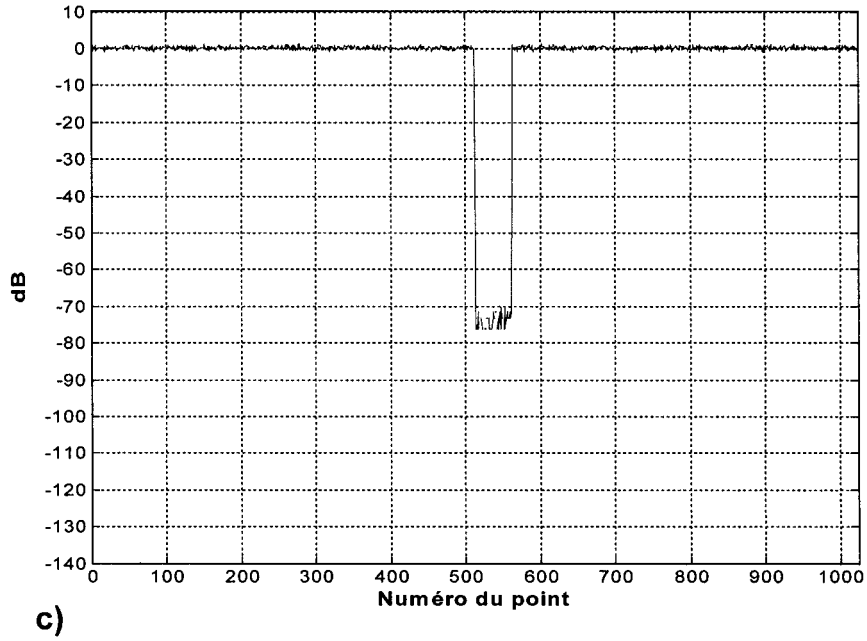


Figure 54 Résultats graphiques du *slot noise test* pour le R2^{2}PC de 1024 points et de 16 bits: **a)** fente de référence, **b)** fente avec mise à l'échelle de 128 (mise à l'échelle adéquate) et **c)** fente avec mise à l'échelle de 1024.

5.3 Conclusion

Finalement, le R2^{2}PC demande en général moins de ressources FPGA et possède plus de reconfigurabilité que la majorité des *cores* de FFT/IFFT disponibles commercialement. L'atout principal du R2^{2}PC vient de son architecture pipelinée qui donne lieu à une croissance régulière du besoin en ressource en fonction de la taille N et qui permet un contrôle de la précision des résultats à chaque étage. La polyvalente du R2^{2}PC en fait un candidat de choix pour de nombreuses applications de traitement numérique du signal.

CHAPITRE 6

CONCEPTION D'UN FILTRE DANS LE DOMAINE FRÉQUENTIEL

Au travers des sections précédentes, la conception et la caractérisation d'un module reconfigurable de FFT/IFFT ont été présentées. Ce chapitre porte sur l'utilisation du module R2^{2}PC pour la réalisation d'un filtre dans le domaine fréquentiel. Le filtrage dans le domaine fréquentiel est une application intéressante de la FFT/IFFT puisqu'il a dans certain cas une complexité moindre que celle d'un filtre traditionnel dans le temps. Le filtre configurable dans le domaine fréquentiel a été réalisé puis caractérisé.

6.1 Architecture d'un filtre dans le domaine fréquentiel

Un filtre numérique effectue une convolution discrète dans le domaine temporel (Proakis & Manolakis, 1996; Rabiner & Gold, 1975). L'équation de la convolution discrète est donnée en 6.1 avec $x(n)$ la séquence d'entrée, $h(n)$ la réponse impulsionnelle du filtre (les coefficients) et $y(n)$ le signal filtré dans le temps.

$$y(n)=x(n)*h(n)=\sum_{m=-\infty}^{\infty} h(m)x(n-m) \tag{6.1}$$

$$Y(k)=X(k)H(k) \tag{6.2}$$

Une convolution dans le temps correspond à une multiplication dans le domaine fréquentiel, comme exprimé par l'équation 6.2. En effet, $X(k)$, $Y(k)$ et $H(k)$ sont respectivement les représentations en fréquence de $x(n)$, $y(n)$ et $h(n)$ avec k l'indice fréquentiel et n l'indice temporel. Lorsque la réponse impulsionnelle est finie, le filtrage numérique est habituellement implémenté par un filtre FIR (Finite Impulse Response).

Ce filtre réalise l'opération mathématique de convolution dans le domaine temporel. L'architecture et les ressources nécessaires à ce filtre sont détaillées à la section 6.2.1.

Vu la dualité temps-fréquence, il peut être intéressant d'envisager pour l'implémentation des filtres (FDF). Pour réaliser le traitement en fréquence, la séquence $x(n)$ subit le traitement d'une DFT pour devenir $X(k)$. Point à point, $X(k)$ est multipliée par les coefficients dans le domaine fréquentiel $H(k)$ pour obtenir $Y(k)$. Cette dernière séquence est à son tour est ramenée dans le domaine temporel par une IDFT. Pour avoir une complexité mathématique réduite, le traitement des DFT et IDFT doit être effectué par des FFT et IFFT, comme illustré sur la figure 55. Lorsque la séquence en entrée du filtre est très longue, généralement infinie, on aurait besoin de FFT et de IFFT infinies, ce qui n'est pas pratiquement réalisable. Le traitement doit donc être effectué sur la séquence découpée en blocs de données. Le produit en fréquence de deux séquences finies correspond à une convolution circulaire (Proakis & Manolakis, 1996; Rabiner & Gold, 1975) qui est différente de la convolution linéaire. En revanche, une portion des résultats issus de la convolution circulaire est la même que celle de la convolution linéaire et permet de réaliser le filtrage adéquatement.

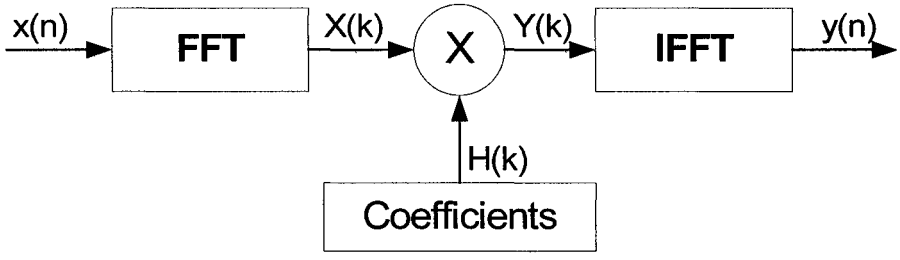


Figure 55 Schéma bloc d'un filtre dans le domaine fréquentiel

Deux méthodes de sectionnement par bloc existent pour réaliser un FDF. Il s'agit de la méthode *overlap-save* et de la méthode *overlap-add (Proakis & Manolakis, 1996; Rabiner & Gold, 1975; Shynk, 1992)*. Ces deux méthodes sont expliquées dans les sections suivantes, avec M défini comme le nombre de coefficients du filtre, N la taille

des FFT/IFFT utilisées et L la taille des blocs de données traitées. En posant $N = L+M-1$, on peut fixer N et L en connaissant la taille M du filtre équivalent dans le temps. Pour obtenir N coefficients fréquentiels, on calcule la DFT sur les M coefficients temporels suivis de L-1 zéros. La procédure est la même pour les deux méthodes de sectionnement.

6.1.1 Méthode *overlap-save*

Le filtrage utilisant la méthode *overlap-save* est illustré à la figure 56. Pour traiter chaque bloc (B# sur la figure 56) de L points, on fait la FFT sur les M-1 derniers échantillons du bloc précédent concaténés avec L nouveaux échantillons. Suite à l'IFFT, on élimine les M-1 derniers points de chaque bloc puisqu'ils contiennent du recouvrement spectral. Ensuite, on concatène les blocs de L données valides afin de former la séquence correspondant à une convolution linéaire. On doit débuter une longue séquence à filtrer avec M-1 zéro.

Pour réaliser un filtre avec la complexité matérielle la plus faible, on prend M-1 égal à une valeur puissance de deux et on choisit $L = M$-1. Ainsi, la FFT et la IFFT auront une taille de $N = 2L$, d'où le chevauchement de 50 %. Ce cas particulier implique que le taux de traitement dans les modules arithmétiques devra être du double du taux d'arrivée des échantillons, f_{in}, de la séquence à filtrer. En effet, chaque bloc de L échantillons passera deux fois dans le FDF.

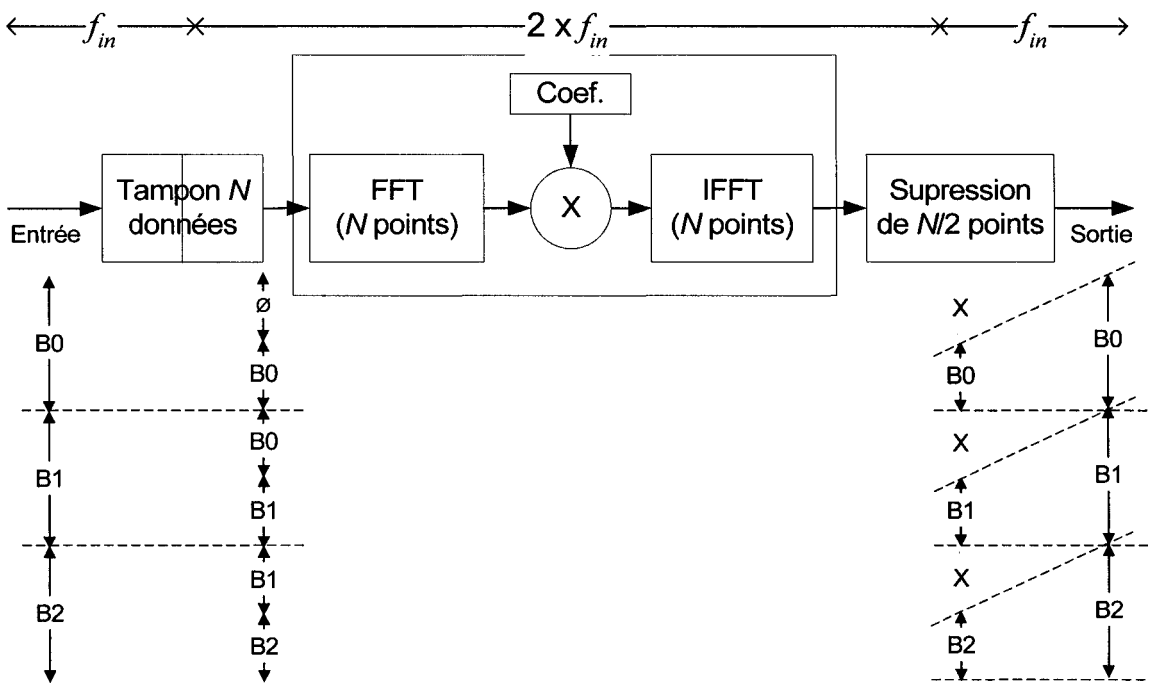


Figure 56 Traitement par bloc utilisant la méthode *overlap-save* avec chevauchement de 50 %

6.1.2 Méthode *overlap-add*

Pour la méthode *overlap-add* (figure 57), on découpe la séquence d'échantillons en blocs de L points auxquels on ajoute une séquence de M-1 zéro. Puis la FFT, la multiplication et la IFFT sont effectuées. Pour obtenir la séquence filtrée, on somme les M-1 derniers points d'un bloc avec les M-1 premiers points du bloc suivant et ainsi de suite. Cette méthode de sectionnement nécessite aussi un traitement plus rapide que le taux d'entrée des échantillons.

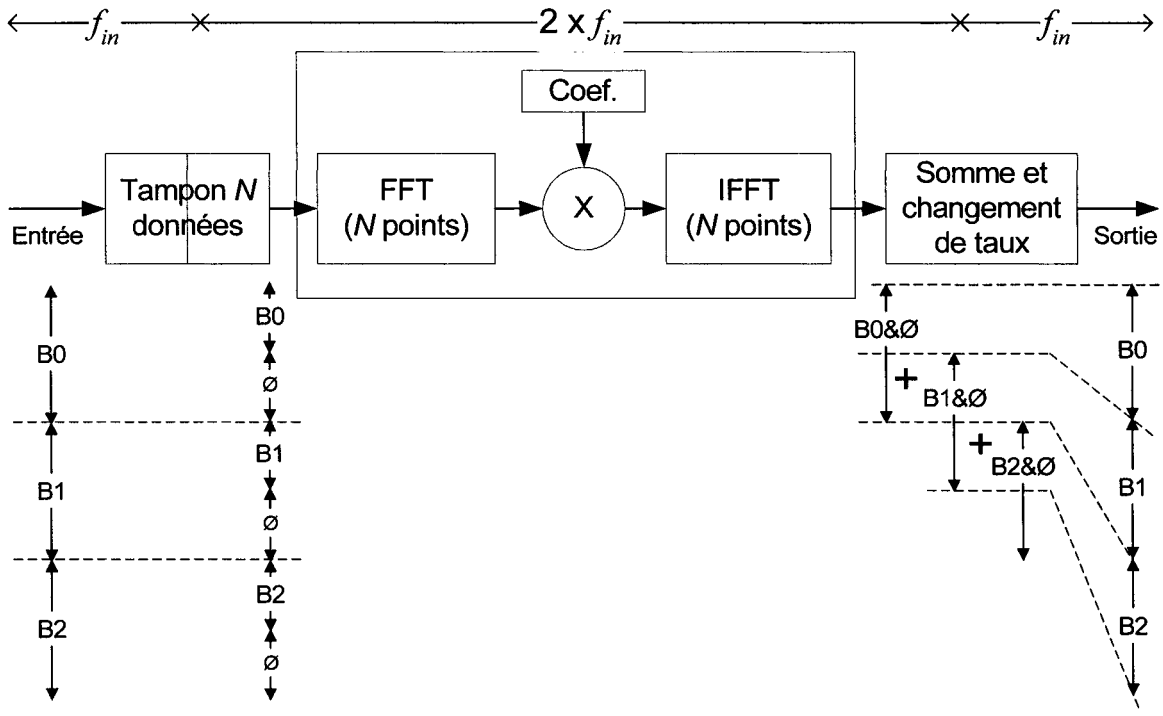


Figure 57 Traitement par bloc utilisant la méthode *overlap-add* avec chevauchement de 50 %

6.2 Comparaison de complexité entre le FIR et le FDF

La complexité des deux designs est évaluée en terme d'unités mathématiques, tel que les additionneurs et les multiplicateurs, et de mémoire requise au fonctionnement du filtre. Le terme additionneur sera utilisé pour décrire autant un additionneur qu'un soustracteur, de manière à ne pas alourdir le texte. De plus, une fois convertie en *slices* la complexité d'un soustracteur en représentation complément à deux est la même que celle d'un additionneur.

Dans les deux cas d'implémentation, les données à filtrer arrivent sous forme de flot sériel. L'analyse est valide dans le cas d'un filtre à l'entrée complexe et aux coefficients complexes. Les deux architectures ont théoriquement une différence dans le taux de traitement des données, vu le sectionnement en blocs nécessaire dans le cas du FDF.

Pour obtenir une comparaison juste, l'implémentation standard du FIR sera modifiée pour que ses multiplicateurs fonctionnent au même taux que ceux du FDF, soit deux fois plus vite que le taux d'arrivé des données en entrée.

6.2.1 Ressources nécessaires au FIR

Si l'on considère l'architecture transversale d'un FIR (figure 58), un multiplicateur-accumulateur (MAC) est nécessaire pour chacun des coefficients (Proakis & Manolakis, 1996). Un MAC complexe équivaut à quatre multiplicateurs et à quatre additionneurs (deux pour le multiplicateur, deux pour l'accumulateur). Pour faire fonctionner ces modules deux fois plus rapidement que le taux d'arrivé des échantillons, donc au même taux que les multiplicateurs du FDF, on doit modifier cette architecture habituelle. Pour ce faire, on considère qu'un seul MAC est nécessaire pour effectuer le traitement de deux étages et que des multiplexeurs acheminent les données et les coefficients correspondants à chaque cycle. En première approximation, la complexité en terme d'unité logique de cette architecture modifiée est avantagée d'un facteur deux par rapport à son implémentation traditionnelle. Le FIR nécessite donc $2M$ additionneurs et $2M$ multiplicateurs, au lieu de $4M$ additionneurs et $4M$ multiplicateurs.

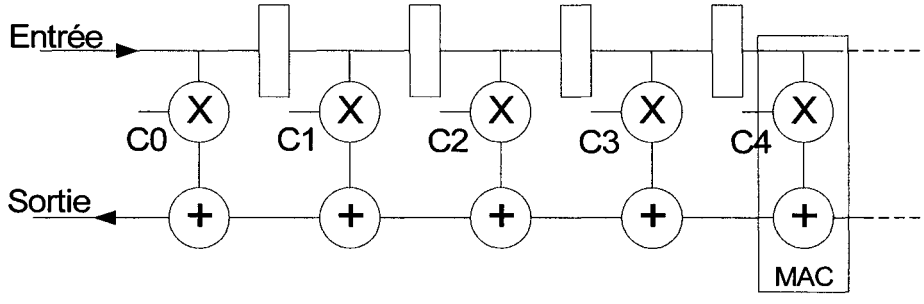


Figure 58 Architecture transversale directe d'un FIR

La seule mémoire nécessaire au FIR est celle pour les M coefficients. On ne considère pas les éléments de délais entre les étages ni les registres de pipeline qui servent à

accélérer la fréquence de traitement. La latence qu'on définit comme le nombre de cycles entre l'arrivée d'une donnée et la sortie du résultat correspondant, est de M cycles plus le délai nécessaire au pipeline du filtre.

6.2.2 Ressources nécessaires au FDF

Dans cette analyse, on considère que la méthode *overlap-save* est utilisée avec un chevauchement de 50 %. Des valeur de chevauchement inférieure à 50% nécessiteraient des transformées de taille supérieure à $2L$, donc une complexité plus grande avec une fréquence pour le traitement des données plus faible que $2f_{in}$. Si la fréquence de traitement n'est pas une contrainte majeure, 50 % est le choix optimal pour le chevauchement.

Le filtre dans le domaine fréquentiel est composé de deux modules R2^{2}PC de taille N=2L séparés par un multiplicateur complexe. Comme la FFT est configurée pour recevoir les échantillons temporels en ordre naturel, sa sortie est en ordre des bits renversé. Le produit avec la réponse impulsionnelle se fait donc avec les coefficients en ordre des bits renversé. Les résultats sont alors transmis au module de IFFT qui est configuré pour recevoir l'ordre renversé et sortir les données temporelles en ordre naturel.

Comme détaillé au chapitre 3, chaque R2^{2}PC requière un multiplicateur complexe pour chaque groupe de deux étages (cas N puissance de quatre, v pair) et un multiplicateur supplémentaire si le nombre d'étages est impair (cas N puissance de deux, v impair). Le multiplicateur n'est pas requis à la suite du dernier étage. Chaque papillon contient deux additionneurs complexes, ce qui correspond à quatre réels. Si v est pair, on aura 2v-4 multiplicateurs et 5v-2 additionneurs. Pour v impair, 2v-2 multiplicateurs et 5v-1 additionneurs sont requis. Pour réaliser une IFFT, deux additionneurs supplémentaires

sont nécessaires pour réaliser les opérations de complément à deux. Le tableau XII résume les unités de traitement mathématiques nécessaires à la réalisation du FDF.

On doit prévoir $4N$ cases mémoires pour les facteurs de phase et les lignes à délais de différentes tailles pour les modules de FFT et IFFT. Pour la méthode d'*overlap-save*, c'est $3N/2$ cases mémoire qui sont requises et N autres pour stocker les coefficients. Vu les possibilités de paramétrisation du R2^{2}PC et du FDF, tous ces emplacements mémoire peuvent avoir des largeurs binaires différentes. Afin de simplifier l'analyse, on ne considère pas l'impact de ces largeurs non constantes. Le total est donc de $13N/2$ cases mémoire, excluant les registres du pipeline.

Tableau XII

Ressources en unités mathématiques nécessaires au FDF utilisant le module R2^{2}PC

Bloc de traitement	**Additionneurs**		**Multiplicateurs**	
	v impair	v pair	v impair	v pair
FFT	$5v$-1	$5v$ -2	$2v$ -2	$2v$ -4
Multiplicateur	2	2	4	4
IFFT	$5v$ +1	$5v$	$2v$ -2	$2v$ -4
Total	$10v$+ 2	$10v$	$4v$	$4v$-4

Dans l'évaluation de la latence, le délai introduit par les registres de pipeline ne sera pas considéré puisqu'il est difficile à évaluer et qu'il est faible par rapport à la latence due au traitement. La latence est calculée en terme de cycle d'arrivée des données, donc à la fréquence f_{in}. La figure 59 détaille les étapes du FDF qui introduisent de la latence. Lorsqu'une donnée arrive au filtre, elle est mémorisée dans le tampon d'entrée et $L/2$ cycles s'écoulent avant qu'elle soit transmise une première fois dans le module de FFT. La FFT et la IFFT traitent les données au double de la fréquence d'arrivée des

échantillons. Le traitement requis par chaque transformée de N points prend L cycles. On peut considérer un seul cycle de latence pour le pipeline du multiplicateur intermédiaire et la mémoire de sortie n'introduit pas de latence supplémentaire. Comme les L=M-1 premiers points qui sortent de la IFFT sont supprimés et les L points suivants sont ramenés au même taux que l'arrivée des échantillons, un total de $5L/2$+1 cycles sont nécessaires pour obtenir un résultat.

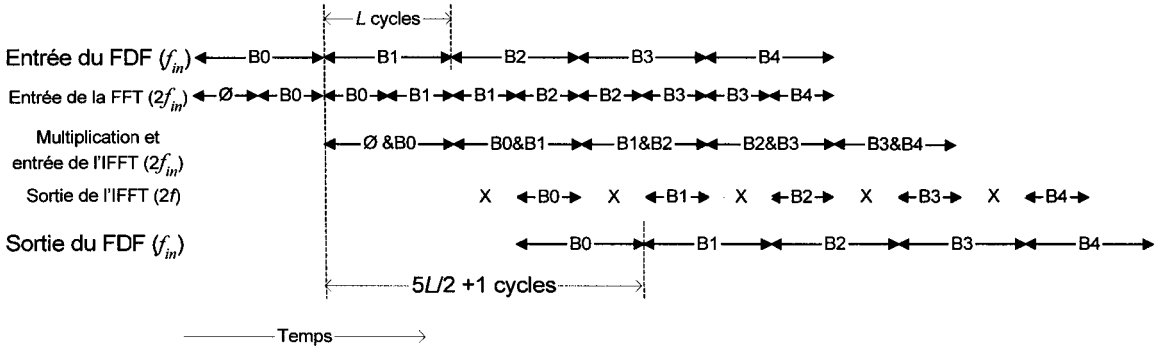


Figure 59 Détails de la latence du FDF untilsant la méthode *overlap-save* et un chevauchement de 50%

6.2.3 Comparaison

Le tableau XIII et la figure 60 exposent les caractéristiques des deux architectures traitées précédemment. Pour aider à la comparaison, on a remplacé v par son expression correspondante en terme de la M pour un chevauchement de 50%. On trouve donc v = $\log_2[2(M\text{-}1)]$. Les points de rencontre en terme de multiplicateurs et d'additionneurs sont respectivement à 8 et à 32 coefficients, soit des tailles de filtre relativement faibles. La croissance logarithmique du FDF l'avantage par rapport au FIR qui a une croissance linéaire. Ainsi, pour des filtres assez longs qui doivent travailler à une fréquence d'horloge ne permettant pas de réutiliser les étages du FIR standard pour plusieurs coefficients, l'option de l'architecture de type FDF utilisant le R2^{2}PC est avantageuse, même si ses besoins en mémoire sont supérieurs.

Tableau XIII

Comparaison du FIR et du FDF pour les ressources et la latence

	FIR	**FDF**	
	M-tap modifié	*N* puissance de 2	*N* puissance de 4
additionneurs	$2M$	$10\log_2[2(M-1)] + 2$	$10\log_2[2(M-1)]$
multiplicateurs	$2M$	$4\log_2[2(M-1)]$	$4\log_2[2(M-1)]-4$
mémoire*	M	$13(M-1)$	
latence	M	$5(M-1)/2+1$	

*Une comparaison directe n'est pas possible vu les différentes natures de mémoire

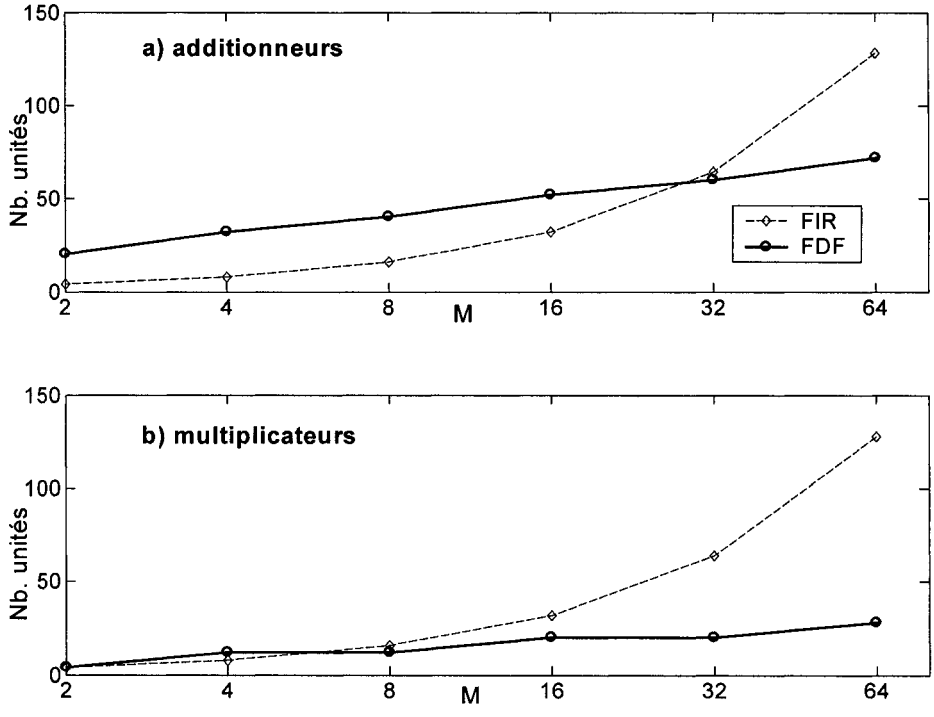


Figure 60 Ressources requises pour les deux types de filtre en fonction du nombre de coefficients: a) additionneurs, b) multiplicateurs

6.3 Construction du filtre dans le domaine fréquentiel

Le FDF configurable a été programmé en VHDL synthétisable sur deux niveaux hiérarchiques. D'une part, on retrouve le niveau supérieur qui permet l'interconnexion entre les sous-modules paramétrés. D'autre part, le plus bas niveau regroupe les sous-modules réutilisables qui possèdent la logique nécessaire au calcul et au contrôle. Au travers des sections à venir, les différents modules sont décrits et illustrés.

6.3.1 Niveau supérieur

Le niveau hiérarchique supérieur établit l'interface avec l'usager par les paramètres de configuration choisis à l'aide de *generics*. Son architecture de type structurelle effectue les interconnexions entre les différents sous-modules et transmet les paramètres. La taille du filtre ainsi que de nombreuses largeurs binaires sont configurables. Parmi les sous-modules, on retrouve le $R2^2PC$ qui est utilisé pour réaliser la FFT et l'IFFT. On atteint ainsi notre but de réutiliser des modules configurables existants. Une seule horloge est utilisée pour tous les sous-modules les échantillons temporels ne doivent pas arriver à un rythme supérieur à un par deux cycles d'horloge.

Fichier: *freq_filter_syn.vhd*

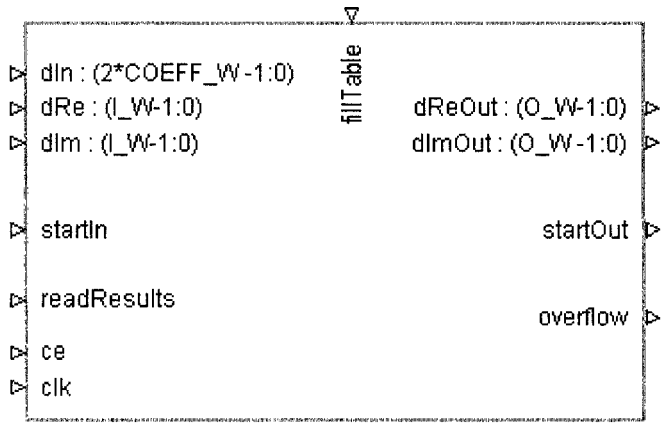


Figure 61 Symbole du filtre dans le domaine fréquentiel

Generics:

I_W	positive	Largeur binaire de l'entrée du FDF.
I_F_W	positive	Largeur binaire dans le domaine fréquentiel à l'entrée du multiplicateur, donc à la sortie de la FFT.
O_F_W	positive	Largeur binaire dans le domaine fréquentiel à la sortie du multiplicateur, donc à l'entrée de la IFFT.
O_W	positive	Largeur binaire à la sortie du FDF.
TF_W	positive	Largeur binaire des facteurs de phase.
COEFF_W	positive	Largeur binaire des coefficients du filtre.
W_FFT	width_typ	Largeur binaire des bus inter-étages dans la FFT. Le type *width_typ* est une liste d'entiers positifs définit dans le package *define_width.vhd.*
W_IFFT	width_typ	Largeur binaire des bus inter-étages dans la IFFT.
LOG2N	positive	Indication de la taille des transformées: nombre d'étages nécessaires, soit $v = \log_2 N$.
S_FFT	positive	Mise à l'échelle correspondant aux divisions par deux effectuées dans la FFT: valeur entre un et v.
S_IFFT	positive	Mise à l'échelle correspondant aux divisions par deux effectuées dans la IFFT: valeur entre un et v.

MULT_STYLE	string	Type de ressources pour l'implémentation des multiplicateurs: "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.
MULT_ROUND_W	positive	Largeur binaire du bus de données précédent les multiplicateurs complexes, paramètre fixé à 18 dans le cas d'un multiplicateur dédié de Xilinx.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
startIn	Indique le début d'une séquence à filtrer.
fillTable	Mode de mise à jour des coefficients: la valeur "1" active l'écriture des coefficients dans la table.
readResults	Signal de lecture des résultats suite au filtrage d'un bloc.
dRe	Partie réelle du vecteur d'entrée.
dIm	Partie imaginaire du vecteur d'entrée.
dIn	Bus de donnée pour l'écriture des coefficients.

Sorties:

startOut	Indique la fin de la latence suite à un signal *startIn.*
overflow	Indique un débordement dans les calculs et donc une séquence non valide.
dReOut	Partie réelle du vecteur de sortie.
dImOut	Partie imaginaire du vecteur de sortie.

6.3.2 Sous-modules

6.3.2.1 FFT et IFFT

Le R2^{2}PC sert à mettre en oeuvre la FFT en ordre naturel et l'IFFT en ordre des bits renversé. Ce module est expliqué et caractérisé en détail dans les chapitres précédents. Il contient trois niveaux hiérarchiques et plusieurs paramètres de configuration.

Fichier : *r22pc.vhd*

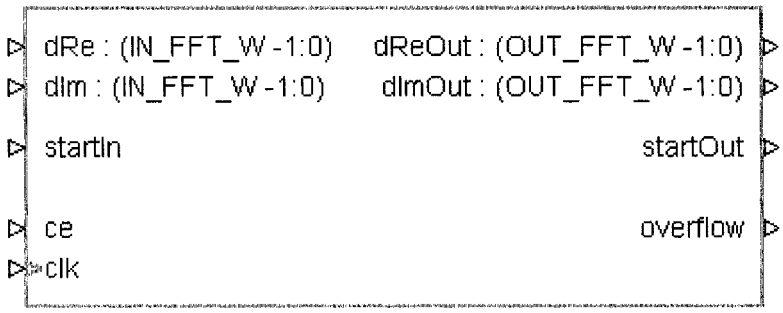


Figure 62 Symbole du R2^{2}PC

6.3.2.2 Multiplicateur complexe

Le multiplicateur complexe est utilisé pour faire le produit avec les coefficients de la réponse impulsionnelle contenus dans une table. Il est très semblable à celui utilisé dans le module R2^{2}PC. Sur la figure 63, on peut voir qu'il s'agit d'un multiplicateur complexe standard utilisant quatre multiplicateurs et deux additionneurs dont les largeurs binaires sont configurables. Il ne possède pas la particularité de conjuguer ses entrées comme c'était le cas pour celui du R2^{2}PC. Pour plus de détails sur ce sous-module, on peut se référer à la section 3.3.3.3.

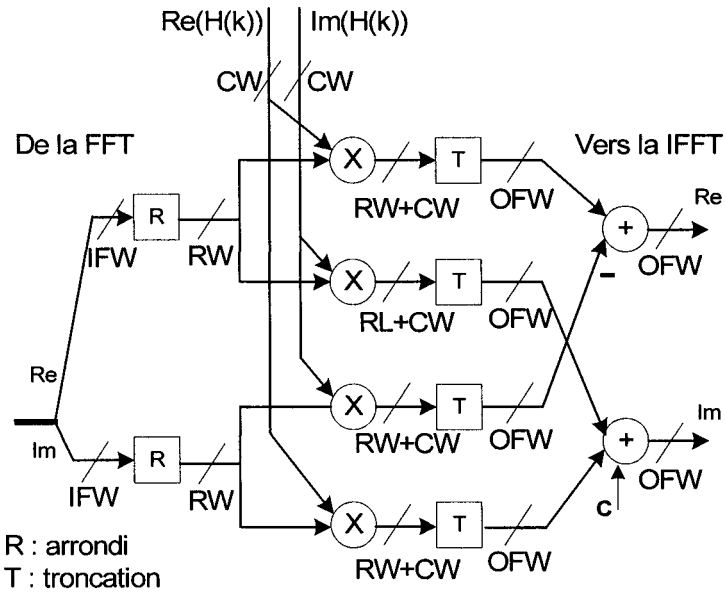


Figure 63 Schéma du multiplicateur complexe

Fichier: *mult_complex_rxt.vhd*

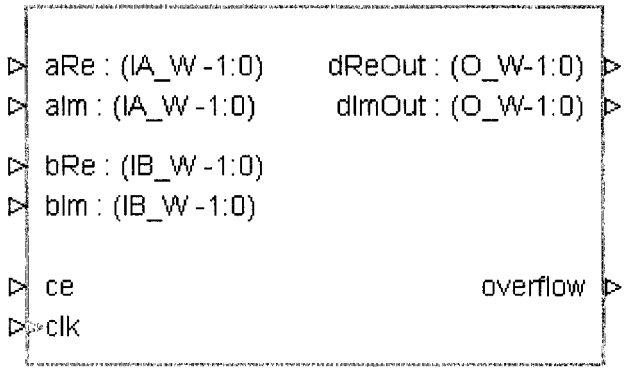


Figure 64 Symbole du multiplicateur complexe

Generics:

IA_W	positive	Largeur binaire de l'entrée A du bloc: entrée des coefficients.
	positive	Largeur binaire de l'entrée B du bloc: entrée des données.
O_W	positive	Largeur binaire de la sortie des données.
R_W	positive	Largeur binaire du bus de données précédant les multiplicateurs complexes, paramètre fixé à 18 dans le cas d'un BMULT de Xilinx.
STYLE	string	Type de ressources pour l'implémentation des multiplicateurs: "*block_mult*" pour un multiplicateur dédié et "*logic*" pour un multiplicateur en logique.
PIPELINE_IN	positive	Nombre de registres précédant les multiplicateurs.
PIPELINE_OUT	positive	Nombre de registres suivant les multiplicateurs, avant les additionneurs.

Entrées:

clk	Horloge.
ce	Signal d'activation de l'horloge *clock enable.*
aRe	Partie réelle du facteur de phase.
aIm	Partie imaginaire du facteur de phase.
bRe	Partie réelle du vecteur de données provenant d'un papillon.
bIm	Partie imaginaire du vecteur de données provenant d'un papillon.

Sorties:

dReOut	Partie réelle du vecteur de sortie.
dImOut	Partie imaginaire du vecteur de sortie.
overflow	Indique un débordement dans les calculs.

6.3.2.3 Table de coefficients

La table de coefficients contient les N coefficients de la réponse impulsionnelle provenant de la DFT sur M coefficients temporels. Cette table est construite avec un adressage en ordre des bits renversé. Les étapes de calcul des coefficients et d'ordonnancement ne sont pas incluses dans ce module. La table est essentiellement une mémoire qui a un signal d'adresse, un port pour l'écriture de nouveaux coefficients et un port pour la lecture. Pendant la mise à jour des coefficients, les résultats de sortie du filtre ne sont pas valides. Le mécanisme pour écrire dans la mémoire est très simple. On pourrait joindre un pont (*wrapper*) à cette interface pour utiliser une interface standardisée plus complexe.

Fichier: *filter_coef.vhd*

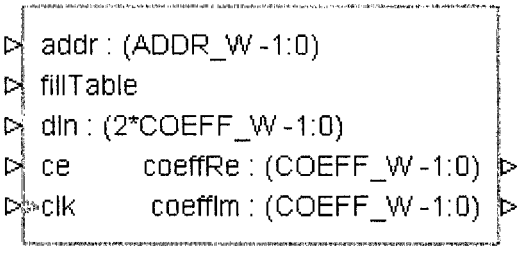


Figure 65 Symbole de la table des coefficients

Generics:

`ADDR_W`	positive	Largeur de l'adresse pour atteindre un coefficient dans la table: correspond à $\log_2(N)$.
`COEFF_W`	positive	Largeur binaire des coefficients.

Entrées:

`clk`	Horloge.
`ce`	Signal d'activation de l'horloge *clock enable.*

`addr`	Adresse pour atteindre un coefficient.
`fillTable`	Signal de mise à jour des coefficients: "1" active l'écriture des coefficients dans la table, "0" permet leur lecture.
`dIn`	Bus de donnée pour l'écriture des coefficients dans la table.

Sorties:

`coeffRe`	Partie réelle du coefficient.
`coeffIm`	Partie imaginaire du coefficient.

6.3.2.4 Tampon d'entrée

Le tampon d'entrée a pour rôle de chevaucher les blocs de la séquence d'entrée. Il est composé d'une mémoire à double port de N=2L cases. Un des ports sert à l'écriture tandis que l'autre est utilisé pour la lecture. Les échantillons de la séquence à filtrer sont écrits séquentiellement à la fréquence d'entrée f_{in}, et ils sont lus à $2f_{in}$. Ce module sert au changement de taux de fonctionnement. Des compteurs servent de pointeurs sur les données lues et écrites. Le pointeur de lecture a la particularité de boucler deux fois sur un bloc de L données avant d'adresser le prochain bloc de L données. Ce module possède un signal de remise à zéro permettant d'amorcer une nouvelle séquence à filtrer.

Fichier: *overlap_save.vhd*

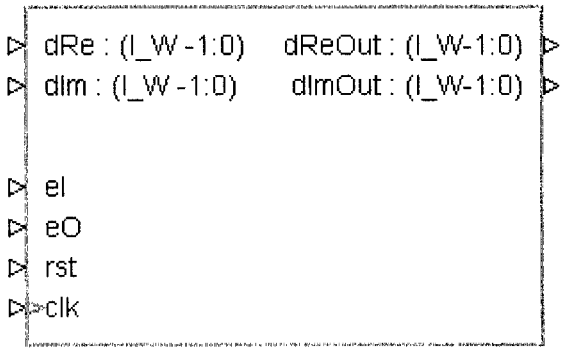


Figure 66 Symbole du tampon d'entrée permettant l'*overlap-save*

Generics:

`LOG2N`	positive	Indique la taille du tampon $\log_2(N)$.
`I_W`	positive	Largeur binaire des données.

Entrées:

`clk`	Horloge.
`eI`	Signal d'activation de l'horloge pour l'écriture dans le tampon de mémoire.
`eO`	Signal d'activation de l'horloge pour la lecture du tampon de mémoire.
`rst`	Remise à zéro des pointeurs d'adresse.
`dRe`	Partie réelle de la séquence à filtrer.
`dIm`	Partie imaginaire de la séquence à filtrer.

Sorties:

`dReOut`	Partie réelle à la sortie du tampon.
`dImOut`	Partie imaginaire à la sortie du tampon.

6.3.2.5 Tampon de sortie

Le tampon de sortie consiste en une mémoire à double port de $L = N/2$ cases. Il élimine les L premiers points de chaque bloc qui sort de la IFFT à la fréquence $2f_{in}$ et il mémorise les L données suivantes. Le port de sortie permet la lecture des échantillons filtrés et le retour à la fréquence f_{in}. Ce module possède un signal de remise à zéro qui s'active au début d'une nouvelle séquence.

Fichier: *discard_buffer.vhd*

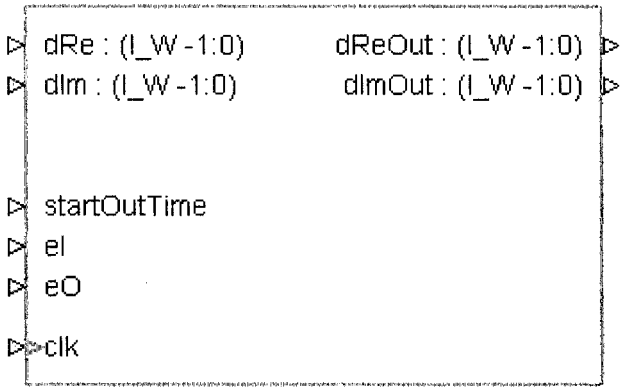


Figure 67 Symbole du tampon de sortie

Generics:

LOG2N	positive	Indique la taille du tampon: $\log_2(N)$.
I_W	positive	Largeur binaire des données.

Entrées:

clk	Horloge.
eI	Signal d'activation de l'horloge pour l'écriture dans le tampon de mémoire.
eO	Signal d'activation de l'horloge pour la lecture du tampon de mémoire.

`StartOutTime`	Indique le début d'une séquence et cause une remise à zéro des compteurs d'adresse.
`dRe`	Partie réelle de la donnée.
`dIm`	Partie imaginaire de la donnée.

Sorties:

`dReOut`	Partie réelle à la sortie du tampon.
`dImOut`	Partie imaginaire à la sortie du tampon.

6.3.2.6 Contrôleurs

La plupart du contrôle du FDF est effectué à même les sous-modules. Leur interconnexion en série permet l'échange des signaux de synchronisation. On retrouve deux sous-modules qui ont des fonctionnalités de contrôle. Le premier, *overlap_save_ctrl,* sert à envoyer les signaux d'activation de lecture et d'écriture au tampon d'entrée ainsi que le signal d'activation `ceF` aux sous-modules effectuant le traitement du signal à $2f_{in}$. Le second, *coef_ctrl,* contrôle l'adresse de la table des coefficients et il indique à la IFFT le début d'une nouvelle séquence.

Fichier: *overlap_save_ctrl.vhd*

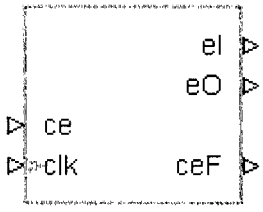


Figure 68 Symbole du contrôleur à l'entrée

Entrées:

`clk`	Horloge.
`ce`	Signal d'activation de l'horloge *clock enable* à la fréquence f_{in}.

Sorties:

`eI`	Signal d'activation de l'horloge pour l'écriture dans le tampon à l'entrée.
`eO`	Signal d'activation de l'horloge pour la lecture du tampon à l'entrée.
`ceF`	Signal d'activation de l'horloge *clock enable* pour les modules fonctionnant à $2f_{in}$.

Fichier: *coeff_ctrl.vhd*

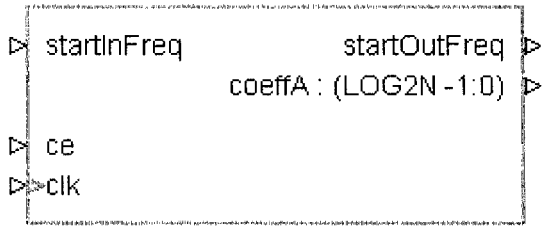


Figure 69 Symbole du contrôleur de la table des coefficients

Generics:

`LOG2N`	positive	Taille du tampon: $\log_2(N)$.
`LATENCY`	positive	Latence pour effectuer la multiplication complexe dans le domaine fréquentiel: typiquement 4 cycles.

Entrées:

`clk`	Horloge.
`ce`	Signal d'activation de l'horloge à $2f_{in}$.
`startInFreq`	Indique l'arrivée d'une nouvelle séquence dans le domaine fréquentiel.

Sorties:

`startOutFreq`	Indique la fin de la latence suite à un signal *`startInFreq`*.
`coeffA`	Adresse pour accéder aux coefficients de la table.

6.4 Performances obtenues

6.4.1 Environnement de vérification

La complexité de réalisation du filtre dans le domaine fréquentiel est donnée en terme de ressources FPGA requises. Les logiciels employés et leur configuration sont les mêmes que ceux décrits à la section 4.1.2.

L'environnement de vérification employé pour déterminer la précision des résultats du FDF est semblable à celui décrit à la section 4.1.1.1 pour le R2^{2}PC et il est illustré à la figure 70. Comme précédemment, le rapport de puissance d'un signal de référence sur la différence entre le signal obtenu et la référence est utilisé comme mesure pour déterminer et comparer la précision des résultats. On utilise ici la même terminologie qu'au chapitre 4: SQNR pour *Signal to Quantization Noise Ratio*. Dans le contexte du filtrage, il aurait peut-être été plus adéquat de renomé la métrique signal à distortion, puisque la différence entre les signaux ne provient pas uniquement du bruit de quantification. Trois modèles mathématiques sont utilisés en plus de la simulation du FDF. Le logiciel Matlab est employé pour traiter les signaux et en faire l'analyse.

Un script sert à la génération et à la quantification du signal d'entrée commun à tous les modèles. Ce signal est une séquence de points complexes de distribution uniforme dans l'intervalle]-1,1[. Ces points sont quantifiés de manière à ce qu'ils puissent être représentés en complément à deux sur IW bits et ils sont mémorisés dans un fichier.

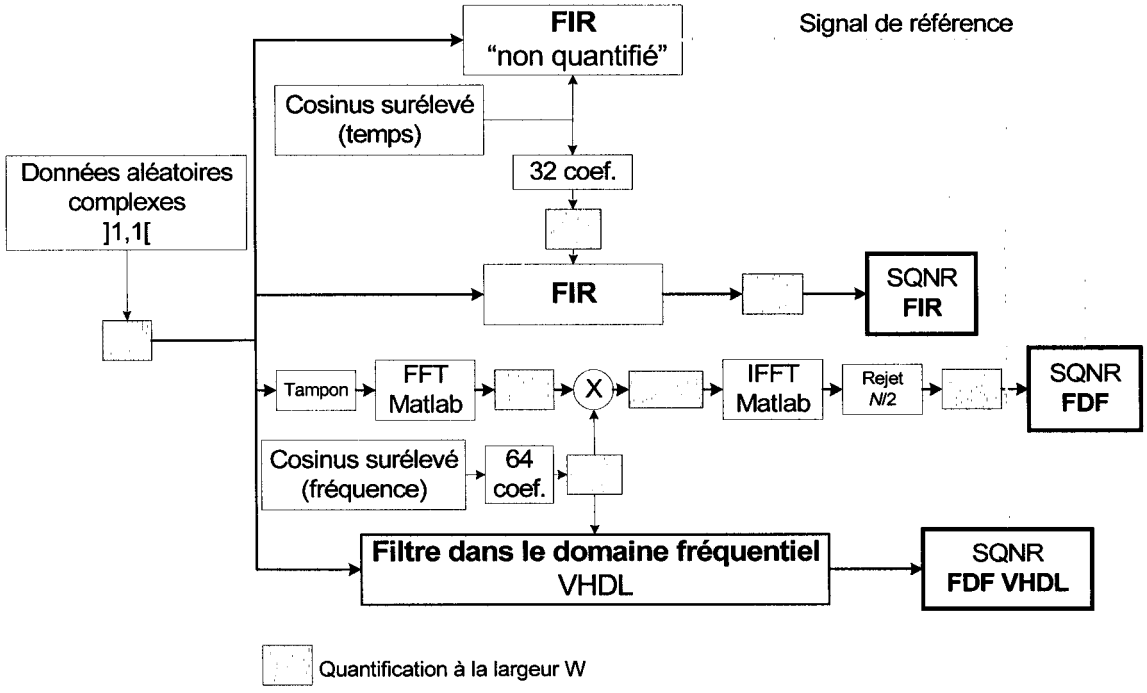


Figure 70 Environnement de vérification du filtre dans le domaine fréquentiel

Le premier modèle consiste en une convolution de la séquence d'entrée avec un grand nombre de coefficients temporels non quantifiés. Ce filtre simule une réponse impulsionnelle infinie, c'est pourquoi on appelle ce modèle FIR non quantifié et sa sortie signal de référence.

Le second modèle correspond à un filtre FIR de M coefficients quantifiés sur CW bits et dont la sortie est sur OW bits. La mesure de précision de ce module est SQNR_FIR et elle est obtenue en effectuant le rapport entre la puissance du signal de référence avec la puissance du bruit de quantification présent à la sortie du FIR quantifié.

Le troisième modèle est un FDF utilisant la méthode *overlap-save*. Il utilise des FFT/IFFT de N points et une multiplication réalisée en notation points flottants. Des quantificateurs entourent chaque bloc pour simuler le fonctionnement du FDF en point

fixe. Les valeurs de quantification en fréquence pré et post multiplicateur, IFW et OFW, peuvent être variées. La précision de ce modèle par rapport au signal de référence sera nommé SQNR_FDF.

Le filtre programmé en VHDL est vérifié par l'intermédiaire d'un banc d'essai aussi en VHDL. Des fichiers textes servent à contenir la séquence d'entrée et les coefficients ainsi qu'à recueillir le signal filtré. Le rapport du signal au bruit de quantification de ce filtre est nommée SQNR_VHDL et est calculé par un script.

Les coefficients de tous les filtres sont générés à partir des équations de la réponse impulsionnelle désirée. L'équation temporelle est échantillonnée 4096 fois pour le FIR non quantifié et M fois pour le FIR quantifié. Les FDF utilisent l'équation en fréquence échantillonnée N fois et quantifiée sur des mots de CW bits. Cette équation correspond à la transformée de Fourier de la fonction de transfert temporelle. Tous les coefficients sont générés par un script Matlab.

6.4.1.1 Filtre en cosinus surélevé

La réponse impulsionnelle choisie pour vérifier le FDF est un filtre en cosinus surélevé (Proakis, 2001). Ce filtre passe-bas est utilisé en télécommunication comme filtre de mise en forme. Le filtre a été construit en fréquence et est exprimé par l'équation 6.3, avec f la fréquence et T la période. Le filtre est plat dans la bande passante, il a la forme d'une demi-période de cosinus dans la bande de transition et rejette tout dans la bande coupée. Dans le temps, l'équation 6.4 représente la réponse impulsionnelle du filtre dans le temps. Le filtre en cosinus surélevé est paramétré par le facteur de mise en forme α (*rolloff factor*) qui prend des valeurs entre 0 et 1. Ce facteur indique l'excès de bande passante, puisque la largeur de bande occupée par le filtre est de $(1+\alpha)/(2T)$. Plus α est faible, plus la bande de transition est abrupte et plus la réponse dans le domaine temporel

s'approche de la fonction sinus cardinal. Les valeurs de α communément employées dans les systèmes de télécommunication se situent entre 0,22 et 0,35.

$$CS(f)=\begin{cases} T & ,\ |f|\leqslant\frac{(1-\alpha)}{2T} \\ \frac{T}{2}\left[1+\cos\left[\frac{\pi T}{\alpha}\left(|f|-\frac{(1-\alpha)}{2T}\right)\right]\right] & ,\ \frac{(1-\alpha)}{2T}\leqslant|f|\leqslant\frac{(1-\alpha)}{2T} \\ 0 & ,\ |f|>\frac{(1+\alpha)}{2T} \end{cases} \tag{6.3}$$

$$cs(t)=\frac{\sin\left(\frac{\pi t}{T}\right)}{\left(\frac{\pi t}{T}\right)}\frac{\cos\left(\frac{\pi \alpha t}{T}\right)}{\left(1-\frac{4\,\alpha^2 t^2}{T^2}\right)} \tag{6.4}$$

6.4.2 Ressources requises et vitesse d'opération

Les ressources nécessaires au FDF pour une implémentation dans un FPGA VirtexII Pro de Xilinx dépendent en grande partie du R2^{2}PC. On sait que l'usage des *slices* de ce sous-module croit de manière logarithmique avec l'augmentation de sa taille. Les autres sous-modules contiennent des multiplicateurs et des mémoires dédiées, mais peu de logique sous forme de *slices*. Comme, le FDF possède un nombre élevé de configurations, il est impossible de toutes les synthétiser. Au tableau XIV, on présente les résultats de synthèse d'un cas de paramétrisation. La largeur des mots en entrée du filtre est de 5 bits, la sortie est de 12 bits, les coefficients ont 12 bits et les mots en fréquence (IFW et OFW) ont 14 bits de large. La FFT voit sa taille interne progressivement augmenter de 5 à 14 bits. La taille interne de la IFFT reste stable à 14 bits, puis elle est arrondie en sortie à 12 bits. Les deux transformées utilisent des facteurs de phase de 14 bits.

Tableau XIV

Ressources nécessaires à un FDF de 5 bits en entrée, 12 bits en sortie et avec des coefficients sur 12 bits

M	*N*	*Slices*	**BRAM**	**BMULT**
33	64	1668	0	20
129	256	2141	6	28
513	1024	2677	16	36

Comme prévu, les ressources nécessaires au FDF sont loin d'augmenter de manière linéaire avec la l'augmentation de la taille comme ce serait le cas pour un FIR. Le nombre de BMULT utilisé dans le filtre peut être prédit par l'équation 6.5.

$$\text{nombre de BMULT} = 8\left\lceil \frac{v}{2} \right\rceil - 4 \tag{6.5}$$

L'usage des BRAM peut aussi être assez facilement expliqué. Par exemple dans le cas où N = 1024, six BRAM sont requis dans chaque transformée. De plus, deux servent de table des coefficients, un sert tampon d'entrée et un autre de tampon en sortie, ce qui donne un total de 16. Comparons le FDF avec le filtre FIR complexe équivalent de M=513 décrit à la section 6.2.1. Ce dernier nécessiterait plus de 1000 BMULT, ce qui est de loin supérieur aux 52 ressources dédiées employées pour le FDF. Dans ce contexte, l'usage accru de mémoire du FDF par rapport au FIR n'est pas dominant.

Une vitesse d'opération possible de plus de 200 MHz a été calculée après l'étape de placement et de routage pour chaque configuration du tableau XIV. Le chemin critique se trouve dans les multiplicateurs. Vu le chevauchement de 50 %, une séquence où les échantillons arrivent à 100 MHz pourrait être filtrée avec cette architecture.

6.4.3 Précision des résultats

Vu le grand nombre de paramètres pouvant être variés, cette section n'est pas une étude complète du comportement du FDF face au bruit de quantification. Seuls quelques scénario de vérification ont été sélectionnés pour donner un aperçu des performances. L'environnement de vérification présenté est assez polyvalent et pourrait être employé pour simuler une application différente.

6.4.3.1 Variation du facteur de mise en forme et de la largeur interne

Le filtre modélisé et simulé est celui présenté à la section 6.4.2, pour M=33 et N=64. La configuration de 5 bits en entrée et de 12 bits pour la sortie et pour les coefficients s'apparente à celle du filtre d'un transmetteur. On a utilisé les coefficients d'un filtre en cosinus surélevé. Le facteur de mise en forme varie de 0,1 à 0,4. On a aussi fait varier ensemble les valeurs de OFW et de IFW pour les FDF. La séquence d'entrée utilisée est de 1000 fois la taille du filtre. Les paramètres de mise à l'échelle de la FFT et de l'IFFT ont été ajustés de manière à ne pas causer de débordement.

La figure 71 synthétise l'ensemble des résultats obtenus. La courbe du FIR se situe entre les courbes du FDF avec 12 bits et 16 bits pour OFW et IFW. En effet, avec une paramétrisation adéquate du FDF, il est possible d'obtenir des performances supérieures à celles du FIR. Par exemple, pour α=0,22, on obtient un SQNR de 55 dB pour le FDF et de 48 dB pour le FIR.

La courbe SQNR_FIR nous montre que lorsque la transition de la bande passante à la bande coupée se fait abruptement, les performances du FIR s'en trouvent dégradées. Pour un α faible, la réponse impulsionnelle limitée à 33 coefficients est insuffisante. Cet effet se fait moins sentir dans le domaine fréquentiel puisqu'une longue réponse temporelle

devient courte en fréquence. En revanche, on remarque pour le FDF une baisse du SQNR pour un α faible. On peut attribuer cet effet à la résolution fréquentielle limitée par la taille des transformées de Fourier. Pour expliquer ce dernier point, d'avantage de simulations ont été réalisés à la section .

Les résultats de simulation du FDF programmé en VHDL, représentés par les courbes en pointillé, suivent la même tendance que ceux provenant du modèle mathématique. Le modèle est donc fiable et le filtre programmé fonctionne comme prévu. L'écart entre les courbes peut être attribué à la quantification distribuée tout au long des calculs faits en notation point fixe pour le FDF synthétisable.

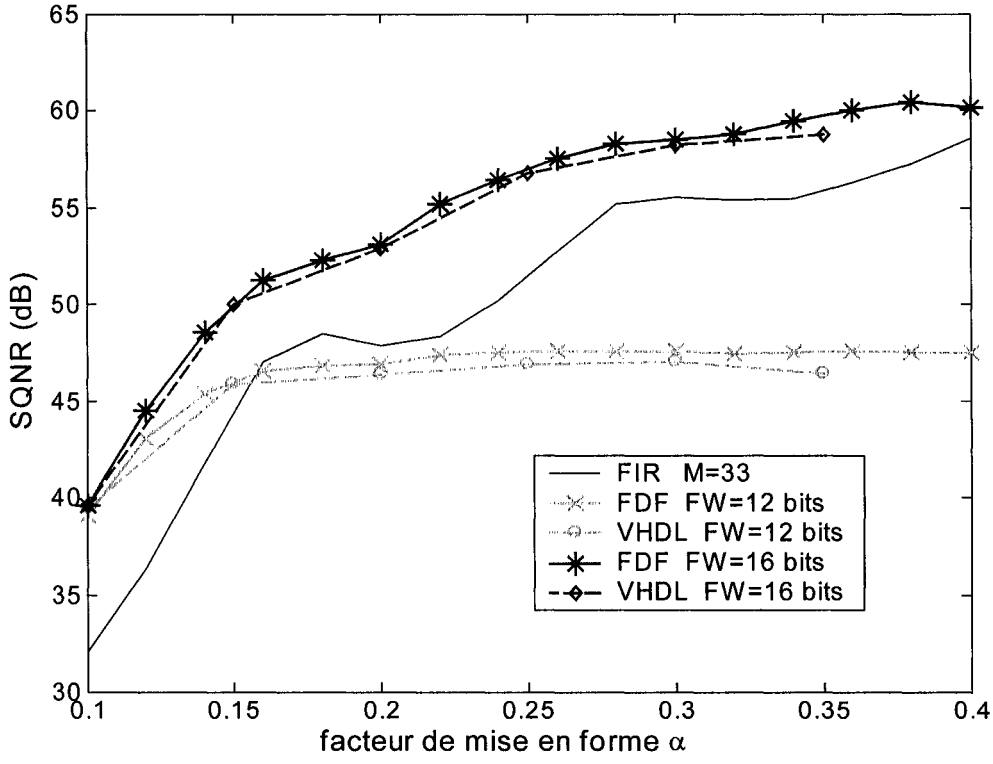


Figure 71 Influence du facteur de mise en forme sur la précision des résultats du filtre en cosinus surélevé de 5 bits en entrée, 12 bits en sortie et avec des coefficients sur 12 bits

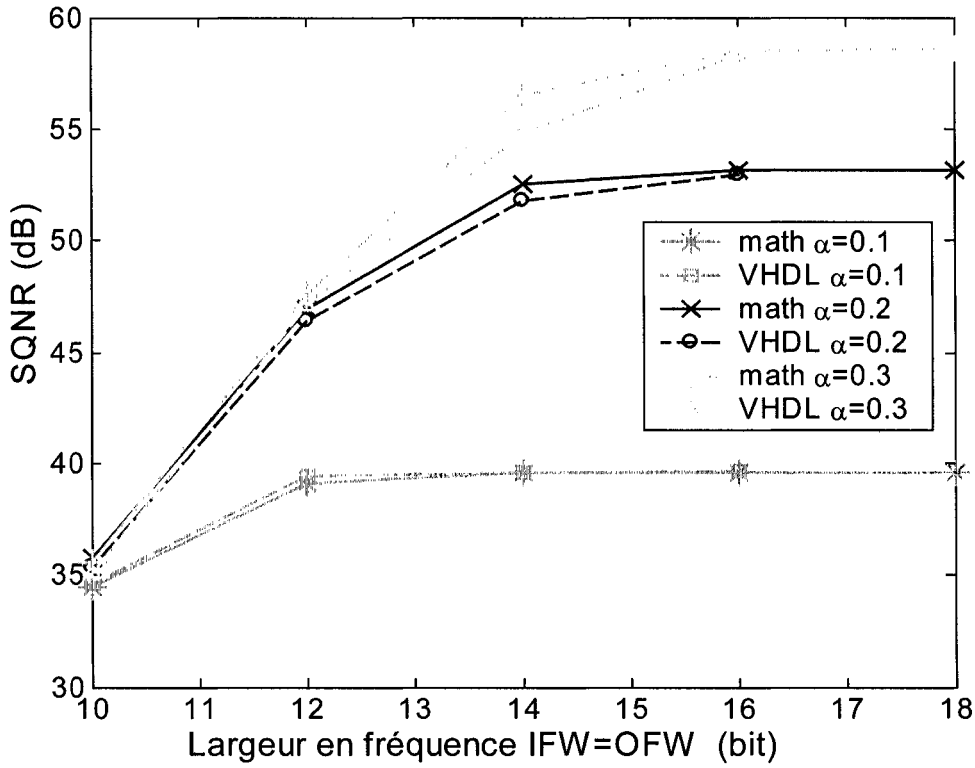


Figure 72 Influence de la largeur en fréquence sur la précision des résultats de filtres en cosinus surélevé de 5 bits en entrée, 12 bits en sortie et avec des coefficients sur 12 bits

La figure 72, met en relief l'importance de bien paramétrer les largeurs internes en fréquence OFW et IFW. Le SQNR atteint un plateau pour une largeur binaire de quelques bits de plus que celle utilisée pour quantifier les coefficients. Les cas suivants portant sur le SQNR ne présentent pas de résultats issus des simulations du FDF en VHDL. On se fiera aux performances du modèle mathématique pour réduire le temps de simulation.

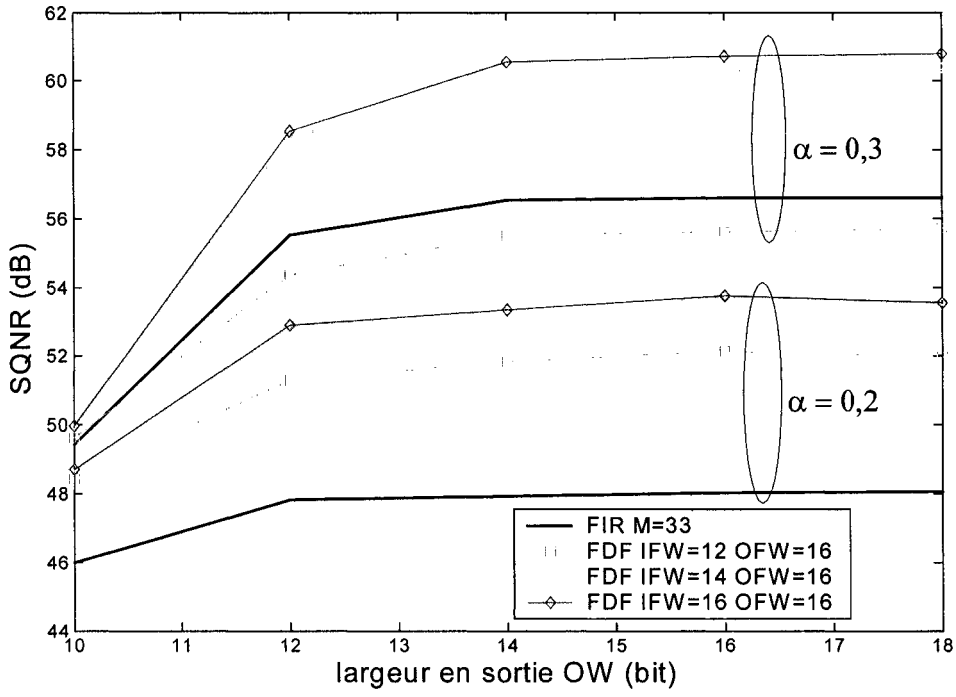


Figure 73 Influence de la largeur en sortie et des largeurs en fréquence sur la précision des résultats de filtres en cosinus surélevé de 5 bits en entrée et 12 bits de coefficients

6.4.3.2 Variation des largeurs binaires

Le scénario précédent a présenté la largeur IFW variant conjointement avec OFW. Ces deux paramètres sont indépendants et peuvent être variés séparément. La figure 73 présente différents cas de paramétrisation de IFW et OFW pour une largeur de sortie OW variant de 12 à 18 bits. On observe qu'il y a plafonnement des performances de toutes les courbes avec l'augmentation de OW. Dans la configuration présentée, 14 bits suffisent. Pour ce qui est des largeurs internes, en ajustant IFW à deux bits inférieurs à OFL (IFW=14 et OFW=16), on obtient un SQNR similaire à celui obtenu lorsqu'ils sont égaux (IFW=OFW= 16) pour α=0,2. Avec un α de 0,3, les deux courbes ont un écart inférieur à 1 dB.

Pour la configuration où IW=5, IFW=16, OFW=16 et OW=12, on a fait varier la largeur des coefficients CW de 10 à 16 bits. On remarque sur la figure 74 que les performances du FDF sont supérieures à celles du FIR et on observe le même comportement pour les deux filtres: un plafonnement du SQNR est atteint avec CW=12. Lorsque la quantification est trop forte (10 bits), le SQNR varie de manière saccadée avec le facteur de mise en forme.

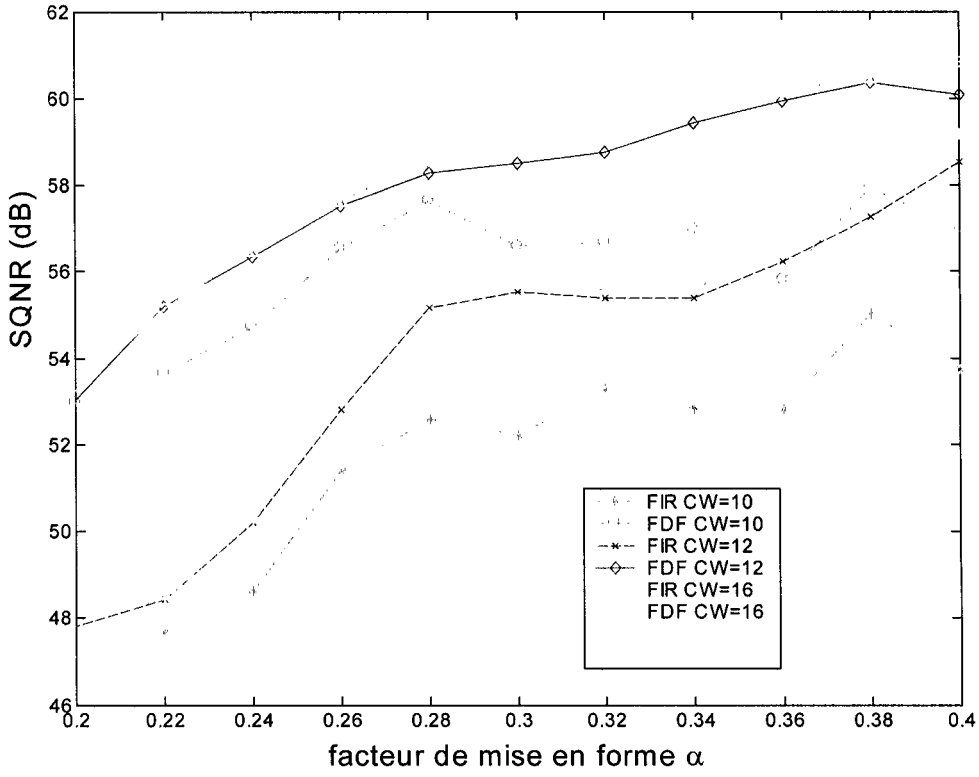


Figure 74 Influence de la largeur des coefficients sur la précision des résultats pour des filtres en cosinus surélevé réalisés dans le temps et en fréquence

6.4.3.3 Variation du rapport *M/N*

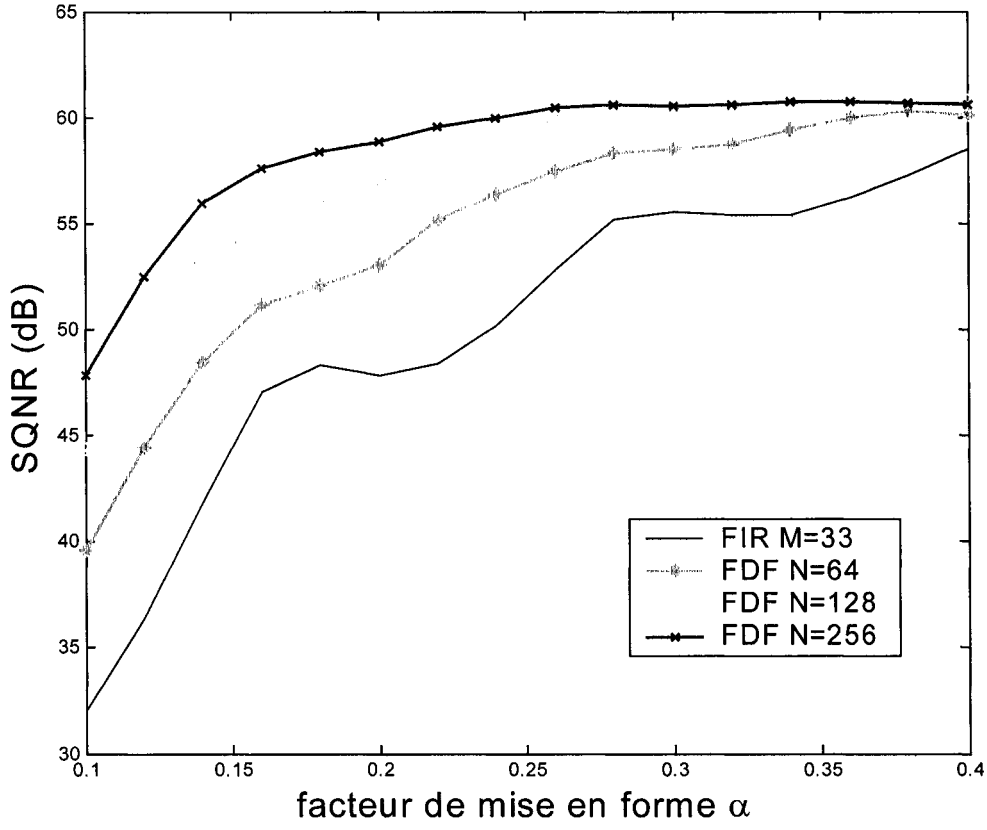


Figure 75 Influence de la taille de la FFT sur la précision des résultats des filtres en cosinus surélevé de 5 bits en entrée, 12 bits en sortie et avec des coefficients sur 12 bits

Pour une même taille de filtre temporel, M=33 dans ce cas ci, on peut faire varier la taille des blocs traités par le filtre fréquentiel, donc la taille des transformées N. Plus N est grand, plus de ressources sont nécessaires à l'implémentation du filtre. Vu la méthode *overlap-save* employée, en augmentant la taille N pour un M fixe le pourcentage de chevauchement diminue et la fréquence de traitement se rapproche de la fréquence d'arrivée des échantillons. On a vérifié la même configuration que précédemment (IW=5, IFW=16, OFW=16, OW=12 et CW=12) pour des transformées de 64, 128 et 256 points. Les résultats de ces filtres en cosinus surélevé sont présentés à la figure 75. On

observe un gain de SQNR avec l'augmentation de la taille *N*. Ce gain de performance est plus important lorsque α est faible. En effet, une plus grande taille implique une plus fine résolution fréquentielle et permet de réaliser des filtres avec des transitions abruptes en fréquence.

6.5 Conclusion

En résumé, l'architecture et la conception d'un module configurable de filtre dans le domaine fréquentiel a été présenté. Ce type de filtre est un bon exemple d'utilisation du module R2^{2}PC. De plus, on a étudié les performances du FDF pour un cas typique de filtre passe-bas. De meilleures performances ont été obtenues avec le FDF comparativement à un FIR paramétré de la même manière. De plus, le FDF requiert moins de ressources pour implémenter des filtres relativement longs. Pour atteindre un certain SQNR, il est important de bien paramétrer les largeurs binaires dans le temps et en fréquence. La méthodologie de vérification présentée et les scripts utilisés sont applicables à tout type de filtre et permettent de déterminer la bonne configuration du FDF.

CONCLUSION

La transformée rapide de Fourier est un outil de traitement du signal très fréquemment employé. Elle a été maintes fois réalisée autant sous forme logicielle que matérielle. Par contre, ses implémentations matérielles manquent de reconfigurabilité et requièrent souvent une grande quantité de mémoire. Le but de ce travail est de concevoir un module réutilisable de FFT/IFFT pour une mise en oeuvre dans une puce programmable.

Le chapitre 2 a permis de présenter les spécifications requises. Le flot sériel de données complexes doit être traité avec le minimum de ressources tout en maximisant la fréquence d'opération. La reconfigurabilité se fait avant la synthèse du module par l'intermédiaire de paramètres de configuration. Ces derniers permettent de choisir la taille de la transformée, les largeurs binaires des interfaces et à l'interne. On a aussi le choix du type de transformée, directe ou inverse, et de l'ordonnancement, naturel ou renversé. L'étude des architectures existantes, principalement les architectures parallèles et pipelinées, nous a permis de déterminer celle qui répond le mieux aux besoins. Le choix s'est arrêté sur l'architecture *Radix-2^2 Single-path Delay Feedback* (R2^2SDF). Les principaux avantages de cette architecture sont la croissance régulière des besoins en ressources en fonction de la taille, le peu de mémoire requis, la faible latence ainsi que la facilité de contrôle et de paramétrisation.

Une approche hiérarchique sur trois niveaux a permis de construire le module de FFT/IFF nommé R2^2PC, pour *Radix-2^2 Parameterizable Core.* Au chapitre 3, on retrouve la description des sous-modules qui une fois interconnectés en série forment le module de transformée de Fourier. La configuration du R2^2PC programmé en VHDL se fait avant l'étape de la synthèse par l'intermédiaire de *generics*. Le mandant de concevoir un module qui répond aux spécifications du chapitre 2 a donc été rempli.

On a procédé ensuite aux étapes de synthèse, de placement et de routage afin de déterminer les ressources requises à l'implémentation du R2^{2}PC dans un FPGA. De nombreuses simulations ont aussi permis d'établir les effets des différents paramètres sur la précision des résultats. La mesure employée pour la précision est le rapport du signal au bruit de quantification, SQNR. Les principales conclusions des tests sont les suivantes:

- Les besoins en ressources augmentent de manière logarithmique avec la taille de la transformée et l'usage de multiplicateurs dédiés augmente par des bonds de 4 unités.
- Les résultats du SQNR pour l'IFFT sont les mêmes que ceux obtenus pour la FFT. L'IFFT demande légèrement plus de *slices*.
- Pour une même configuration de largeurs binaires, une augmentation d'un facteur deux de la taille de la transformée cause 3 dB de moins pour le SQNR.
- La mise à l'échelle a pour effet de diminuer le SQNR d'environ 6 dB par facteur deux non nécessaire pour empêcher le débordement. Elle influence très peu les ressources.
- L'effet des facteurs de phase sur le SQNR est négligeable s'ils sont de la même largeur que la plus grande largeur interne pour les données.
- La largeur interne est un facteur important à considérer. Pour plus de précision, elle doit idéalement être fixée à une valeur égale ou supérieure à la largeur de sortie.

Au chapitre 5, on a comparé le R2^{2}PC avec des *cores* disponibles commercialement. On ne retrouve pas de module possédant autant de reconfigurablité que le R2^{2}PC et nécessitant aussi peu de ressources. Le *Logicore* `XFFT` de Xilinx est le *core* se rapprochant le plus du R2^{2}PC sous l'aspect de la reconfigurabilité, mais il est construit avec une architecture complètement différente. La croissance des besoins en ressources ne se fait pas régulièrement avec l'augmentation de la taille de la transformée. Pour toutes les tailles, le R2^{2}PC utilise moins de *slice* et de ressources dédiées du FPGA que le `XFFT`. Pour ce qui est de la précision, les performances du R2^{2}PC se situent plus près de celles du `XFFT` avec arrondi convergent que de celles avec troncation. La conception du module reconfigurable de FFT/IFFT est une innovation en soit.

Le R2^{2}PC est flexible et polyvalent, ce qui en fait un excellent choix pour de nombreuses applications. Le chapitre 6 donne un exemple d'utilisation de deux R2^{2}PC pour la réalisation d'un filtre configurable dans le domaine fréquentiel. Le filtre dans le domaine fréquentiel, FDF, a une complexité moindre qu'un FIR pour des filtres de plus de 8 ou 32 coefficients selon que l'on considère les multiplicateurs ou les additionneurs. La configuration des largeurs binaires influence grandement la précision des résultats. Un meilleur rapport signal à bruit de quantification peut être obtenu avec un FDF comparativement à un FIR paramétré de manière similaire, surtout si la réponse en fréquence est abrupte.

Les avenues de réutilisation du R2^{2}PC sont nombreuses. Dans le domaine des télécommunications sans-fils à haut débit, des techniques de traitement numérique du signal faisant appel à la FFT sont maintenant possibles, vu la capacité et les performances grandissantes de l'électronique numérique. Par exemple, pour un canal sans ligne de vue directe, la problématique des trajets multiples devient dominante. Comme perspective de recherche et de développement, on présente deux outils utilisant la FFT qui adressent cette problématique et dans lesquels il serait souhaitable d'utiliser le R2^{2}PC.

D'une part, on peut concevoir au récepteur un égaliseur de type bloc LMS (*Least mean square*) pour compenser les effets du canal (Skynk, 1992). Ce type d'égaliseur, illustré par la figure 76, est composé d'un filtre adaptatif dans le domaine fréquentiel (FDAF) utilisant la méthode *overlap-save*. On peut voir que cinq blocs de FFT ou de IFFT sont nécessaires, deux pour le filtrage et trois pour la mise à jour des coefficients.

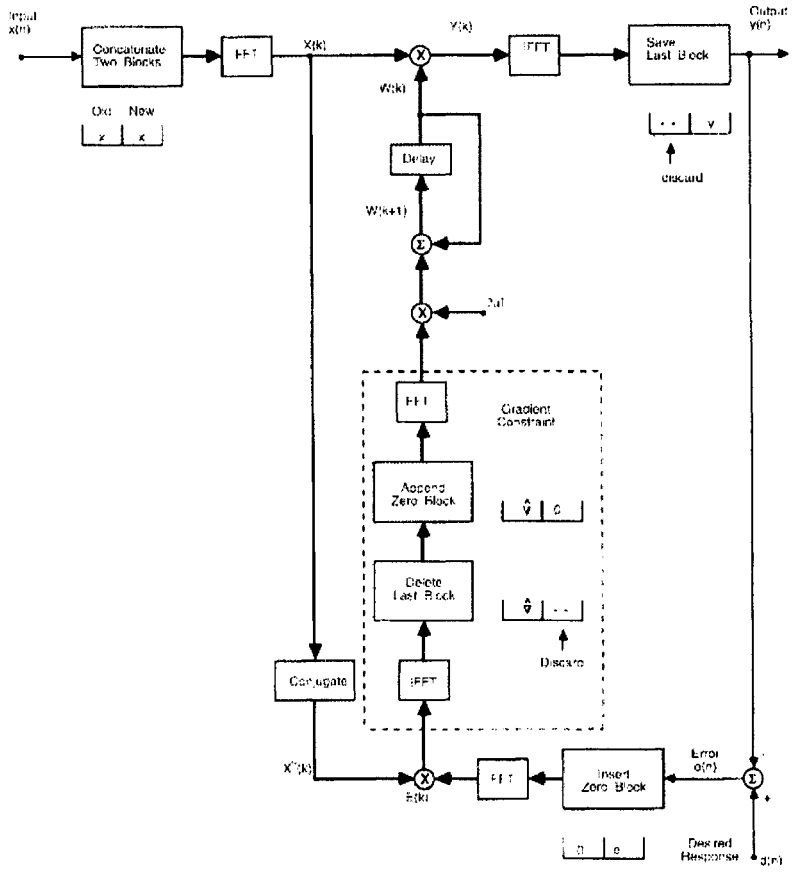


Figure 76 Schéma d'un égaliseur utilisant un filtre adaptatif dans le domaine fréquentiel utilisant la méthode overlap-save
(Tiré de (Skynk, 1992))

D'autre part, la technique OFDM pour Orthogonal Frequency Division multiplexing est de plus en plus répandue pour résoudre le problème des chemins multiples dans les transmissions numériques. La figure 77 illustre un système de communication utilisant l'OFDM. L'idée à la base de l'OFDM est de découper le spectre disponible en plusieurs sous-porteuses orthogonales se chevauchant partiellement. Chaque sous-porteuse transmet un signal numérique modulé en phase à faible débit, mais l'ensemble permet d'atteindre un taux de transfert important. Pour passer des symboles xQAM dans le domaine fréquentiel au symbole temporel OFDM, une IFFT est employée. On introduit aussi une période de garde temporelle au début du symbole pour éviter les interférences potentielles dues aux chemins multiples dans le canal. À la réception, une fois la période

de garde retirée, les symboles xQAM peuvent être obtenus en effectuant une FFT sur le symbole OFDM échantillonné.

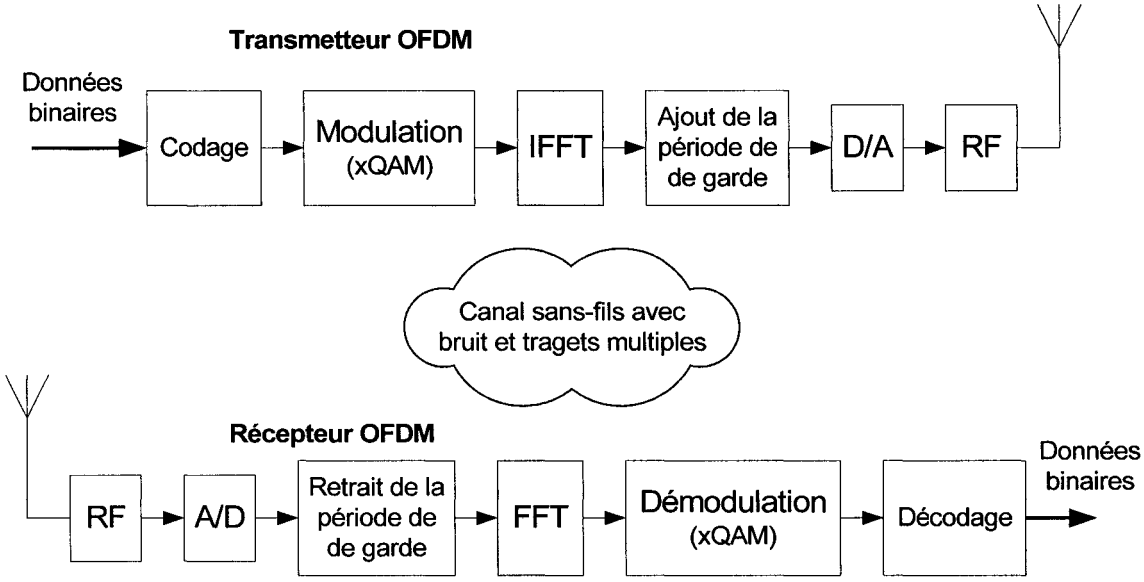


Figure 77 Schéma bloc d'un système de communication utilisant l'OFDM

Un nombre grandissant de standards de télécommunication utilise cette technique. Parmi ceux-ci, on retrouve la diffusion audionumérique DAB (Digital Audio Broadcast), les réseaux locaux Wi-Fi ou WLAN (IEEE 802.11) en Amérique et HiperLAN en Europe et plus récemment, les réseaux métropolitains WirelessMAN (IEEE 802.16) et HiperMAN. Le module reconfigurable de FFT/IFFT R2^{2}PC a un avenir prometteur dans le domaine des télécommunications sans-fils à large bande.

BIBLIOGRAPHIE

Altera Corporation.[En ligne]. www.altera.com (Consulté le 20 sept. 2004).

Bayart, F. (2004). *BibM@th*, [En ligne].www.bibmath.net (consulté le 10 juillet 2004).

Bi, G., & Jones, E. V. (1989). A pipelined FFT processor for word-sequential data. *IEEE Transactions on Acoustics, Speech, and Signal Processing,* 37(12), 1982-1985.

Bidet, E., Castelain, D., Joanblanq, C., & Senn, P. (1995). A fast single-chip implementation of 8192 complex point FFT. *IEEE Journal of Solid-State Circuits,* 30 (3), 300-305.

Boyer, C. B., & Merzbach, U. C. (1991). *A history of mathematics* (2nd ed.). New York, N.Y.: J. Wiley and Sons.

Cooley, J. W. (1992). How the FFT gained acceptance. *Signal Processing Magazine, IEEE,* 9(1), 10-13.

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation,* 19(90), 297-301.

Danielson, G. C., & Lanczos, C. (1942). Some Improvements in Practical Fourier Analysis and Their application to X-ray Scattering Form Liquids. *J. Franklin Inst.,* 233, 365-380, 435-352.

Despain, A. M. (1974). Fourier transform computer using CORDIC iterations. *IEEE Transactions on Computer,* C-23(10), 993-1001.

Dongarra, J., & Sullivan, F. (2000). Guest Editors Introduction to the top 10 algorithms. *Computing in Science & Engineering,* 2(1), 22-23.

Elterich, A., & Stammler, W. (1988, 1988). *Error analysis and resulting structural improvements for fixed point FFTs.* Presented at the International Conference on Acoustics, Speech, and Signal Processing, (ICASSP-88), 3, 1419-1422.

Hasan, M., & Arslan, T. (2002a). *A coefficient memory addressing scheme for VLSI implementation of FFT processors.* Presented at the IEEE International Symposium on Circuits and Systems, (ISCAS 2002), 4, 850-853.

Hasan, M., & Arslan, T. (2002b, 2002). *FFT coefficient memory reduction technique for OFDM applications.* Presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '02). 1, 1085-1088.

He, S., & Torkelson, M. (1996). *A new approach to pipeline FFT processor.* Presented at the The 10th International Parallel Processing Symposium, (IPPS '96), 766-770.

He, S., & Torkelson, M. (1998). *Designing pipeline FFT processor for OFDM (de) modulation.* Presented at the URSI International Symposium on Signals, Systems, and Electronics, (ISSSE 98), 257-262.

Heideman, M., Johnson, D., & Burrus, C. (1984). Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine,* 1(4), 14-21.

Hidalgo, J. A., Lopez, J., Arguello, F., & Zapata, E. L. (1999). Area-efficient architecture for Fast Fourier transform. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* 46(2), 187-193.

Johnson, L. G. (1992). Conflict free memory addressing for dedicated FFT hardware. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* 39(5), 312-316.

Kabal, P., & Sayar, B. (1986). *Performance of fixed-point FFT's: Rounding and scaling considerations.* Presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '86), 11, 221-224.

Knight, W., & Kaiser, R. (1979). A simple fixed-point error bound for the fast Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing,* 27(6), 615-620.

Kreyszig, E. (1999). *Advanced engineering mathematics* (8th ed.). New York, N.Y.: J. Wiley and Sons.

Ma, Y., & Wanhammar, L. (2000). A hardware efficient control of memory addressing for high-performance FFT processors. *IEEE Transactions on Signal Processing,,* 48(3), 917-921.

Mankiewicz, R. (2001). *L'histoire des mathématiques*. Paris: Seuil.

Maor, E. (2002). *Trigonometric Delights*. Princeton, N.J.: Prinston University Press.

Meyer, R. (1989). *Error analysis and comparison of FFT implementation structures.* Presented at the International Conference on Acoustics, Speech, and Signal Processing, (ICASSP-89), 888-891 vol.882.

Oppenheim, A. V., Schafer, R. W., & Buck, J. R. (1999). *Discrete-time signal processing* (2nd ed.). Upper Saddle River, N.J.: Prentice-Hall.

Proakis, J. G. (2001). *Digital communications* (4th ed.). New York, N.Y.: McGraw-Hill.

Proakis, J. G., & Manolakis, D. G. (1996). *Digital signal processing : principles, algorithms, and applications* (3rd ed.). Upper Saddle River, N.J.: Prentice-Hall.

Rabiner, L. R., & Gold, B. (1975). *Theory and application of digital signal processing.* Englewood Cliffs, N.J.: Prentice-Hall.

Shynk, J. J. (1992). Frequency-domain and multirate adaptive filtering. *IEEE Signal Processing Magazine,* 9(1), 14-37.

Son, B. S., Jo, B. G., Sunwoo, M. H., & Kim, Y. S. (2002). *A high-speed FFT processor for OFDM systems.* Presented at the IEEE International Symposium on Circuits and Systems, (ISCAS 2002). 3, 281-284.

Swartzlander, E. E., Jain, V. K., & Hikawa, H. (1992). A radix 8 wafer scale FFT processor. *Journal of VLSI Signal Processing,* 4(2,3), 165-176.

Swartzlander, E. E., Young, W. K. W., & Joseph, S. J. (1984). A radix 4 delay commutator for fast Fourier transform processor implementation. *IEEE Journal of Solid-State Circuits,* 19(5), 702-709.

Szedo, G., Yang, V., & Dick, C. (2001). *High-performance FFT processing using reconfigurable logic.* Presented at the Asilomar Conference on Signals, Systems and Computers, 2, 1353-1356.

Tran-Thong, & Liu, B. (1976). Fixed-point fast Fourier transform error analysis. *IEEE Trans. on Acoustics, Speech, and Signal Processing,* 24(6), 563-573.

Vergara, M., Strum, M., Eberle, W., & Gyselinckx, B. (1998). *A 195K FFT/s (256-points) high performance FFT/IFFT processor for OFDM applications.* Presented at the SBT/IEEE International Telecommunications Symposium, (ITS '98), 1, 273-278.

Wang, Y., Lam, H. M., Tsui, C.-Y., Cheng, R. S., & Mow, W. H. (2002). *Low complexity OFDM receiver using Log-FFT for coded OFDM system.* Presented at the IEEE International Symposium on Circuits and Systems, (ISCAS 2002). 3, 445-448.

Weinstein, C. J. (1969). *Quantization Effects in Digital Filters.* MIT Lincoln Laboratory TR-468.

Wold, E. H., & Despain, A. M. (1984). Pipeline and parallel-pipeline FFT processors for VLSI implementation. *IEEE Transactions on Computer,* C-33(5), 414-426.

Xilinx, Inc. [En ligne]. www.xilinx.com (Consulté le 10 sept. 2004).