Compas'2022 : Parallélisme/ Architecture / Système
MIS/UPJV - Amiens France, 5-8 juillet 2022

Conception d'architectures de FFT pour FPGA à base de modèles comportementaux

Hugues ALMORIN[1,2], Bertrand LE GAL[1], Jeremie CRENNE[1], Christophe JEGO[1] et Vincent KISSEL[2]

1 - Laboratoire IMS (CNRS UMR 5218), Bordeaux-INP, Université de Bordeaux
351 Cours de la libération, 33405 Talence, France
prenom.nom@ims-bordeaux.fr
2 - ARELIS (Groupe LGM)
Rue des Novales, 76410 Saint-Aubin-lès-Elbeuf, France
prenom.nom@arelis.com

Résumé
La transformation de Fourier est un opérateur de traitement de signal utilisé dans de nombreux domaines applicatifs (traitement radar, communications numériques, etc.) et par conséquent intégré dans des systèmes embarqués. Les délais de conception et de mise à niveau de ces systèmes matériels hétérogènes tendent à être de plus en plus courts ce qui implique le développement et l'utilisation de nouvelles méthodologies de prototypage rapide. Les outils de synthèse haut niveau pour la conception d'accélérateurs matériels sont de bons candidats pour l'intégration d'algorithmes de traitement de signaux numériques embarqués tels que la transformation de Fourier rapide (FFT). La qualité de l'exploration de l'espace architectural dépend grandement de la maîtrise de l'outil ainsi que du nombre et de la qualité des modèles comportementaux. Dans cet article, nous proposons une méthode de conception appliquée à l'opérateur FFT. Les meilleurs paramètres de quantification des données de l'algorithme sont sélectionnés à partir de contraintes applicatives. L'exploration de l'espace architectural est ensuite effectuée à partir d'un ensemble de modèles architecturaux flexibles de FFT répondant à différentes stratégies de parallélisation et d'optimisation. L'objectif principal est d'explorer différents types d'architectures parallélisées et semi-parallélisées, couvrant un large ensemble d'implémentations matérielles.

Mots-clés : FFT, FPGA, Accélération Matérielle, HLS, DSE

---

1. Introduction

La transformation de Fourier dans sa forme discrétisée est un des opérateurs de traitement de signal numérique les plus utilisés. Elle est employée dans un grand nombre d'applications incluant par exemple les systèmes de communications numériques [3], le traitement de signaux radars [7] et le traitement de la parole ou des images [26].
Son utilisation au sein de systèmes temps réels, souvent contraints en termes de performance, a nécessité le développement architectures efficaces. De nombreux travaux de recherche se sont focalisés sur la mise au point d'implantations logicielles sur des cibles de type HPC [16], multi-core (CPU) [25] et many-core (GPU) [13] mais aussi sur des cibles embarquées de type micro-

contrôleur [14]. Ils ont donné naissances à des bibliothèques renommées telles *fftw3* [9] utilisée actuellement dans de nombreux logiciels.
À l'opposée de ces implantations logicielles, d'autres travaux de recherche se sont focalisés sur la conception d'architectures matérielles efficaces en termes de débit [30], de latence [19], de complexité [30] et d'énergie [2] pour répondre aux besoins des systèmes embarqués communicants [20]. Ces architectures numériques décrites au niveau RTL à l'aide de langages HDL (p.e. : VHDL) et implantés à l'aide de circuits FPGA et ou en technologie ASIC offrent des niveaux de performance très élevés. Ces performances élevées sont obtenues aux prix d'une faible flexibilité et de longs cycles de conception. Afin de d'assister les concepteurs, des outils dédiés ont été développés afin de concevoir automatiquement les classes d'architectures les plus usuelles [10, 17, 18, 22]. Toutefois, ces outils ne considèrent pas les caractéristiques des circuits FPGA ciblés (temps de propagation, nombre de ressources) lors de leur phase de génération pour optimiser ou contraindre les architectures.
De leur côté, les outils de synthèse de haut niveau (HLS) permettent de concevoir des architectures adaptées aux circuits FPGA ciblés sous contraintes [8, 15]. Ces outils, contrairement à ceux cités précédemment, ne sont pas dédiés à un algorithme ou une architecture. Le concepteur a à sa charge la description du comportement de l'algorithme à implémenter à l'aide d'un langage de haut niveau (p.e., C, C++…). La description comportementale est ensuite analysée et transformée afin de produire des architectures matérielles (niveau RTL) adaptée aux besoins. L'apparition des plateformes Zynq [29], Alveo [28] et Agilex [12] qui facilitent la déportation des calculs d'un processeur vers un accélérateur matériel sur circuit FPGA a suscité un engouement de la communauté pour ces outils (Vitis HLS [31], Intel HLS [11], Catapult HLS [23]). Cependant, les performances et la qualité des architectures ainsi générées dépendent fortement de la pertinence des modèles comportementaux (style d'écriture, annotations, parallélisme intrinsèque, etc.) [5, 6].
Dans cet article, nous proposons un flot de conception centré sur l'utilisation des outils HLS pour estimer le bruit de calcul induit par les formats de quantifications et faciliter l'exploration architecturale des solutions matérielles possibles.
La section 2 introduit l'algorithme FFT et son parallélisme de calcul. Puis les motivations et travaux connexes sont présentés. Le flot de conception proposé et les modèles comportementaux sont décrits dans la section 3. Enfin les résultats expérimentaux sont détaillés dans la section 4.

2. Parallélisme de calcul dans l'algorithme FFT

2.1. Principe

La transformée de Fourier discrète (DFT) qui permet de transposer une séquence de N ($N \in \mathbb{N}$) échantillons $x_n = x_0, x_1, ..., x_{N-1}$ du domaine temporel dans le domaine fréquentiel se calcule à l'aide de l'équation (1).

$$X_k = \sum_{n=0}^{N-1} x_n . e^{\frac{-j2\pi kn}{N}} \tag{1}$$

La complexité algorithmique initiale de la DFT est en $\mathcal{O}(N^2)$. Des algorithmes plus efficaces appelés transformations de Fourier rapide (FFT) permettent de minimiser cette complexité. L'algorithme de Cooley-Tukey, permet par exemple de calculer la DFT avec une complexité en $\mathcal{O}(N \log N)$ [4]. Une formulation simple de cet algorithme est basée sur une décomposition récursive de la DFT sous la forme de 2 DFT sur $\frac{N}{2}$ échantillons. Cette formulation est commu-

nément utilisée pour les implantations de DFT sur des processeurs CPU, GPU et sur circuits FPGA.

2.2. Parallélisation de la transformation

Les calculs nécessaires à la réalisation d'une FFT avec la méthode de Cooley-Tukey peuvent être représentés à l'aide d'une modélisation de type flot de données [1] présentée dans la figure 1a. Classiquement, ce flot de données peut être exécuté séquentiellement à l'aide d'un seul et unique élément de calcul nommé *butterfly* (BF) (① dans la figure 1a). Cet élément de calcul consomme et produit $R = 2$ échantillons en parallèle. Le parcours séquentiel du flot de donnés est représenté par la flèche rouge dans la figure 1a. Au total, l'exécution d'une FFT nécessite $\frac{N}{R}\log_R(N)$ calculs de butterfly de taille R.
Afin d'accélérer l'exécution de la transformation sur des architectures parallèles (CPU, GPU) ou des circuits FPGA, différents niveaux de parallélismes intrinsèques à l'algorithme sont utilisables. Ces derniers sont détaillés ci-dessous :

— Le premier niveau de parallélisme ($\mathcal{P}_1$) nommé parallélisme vertical est présenté dans la figure 1b. En effet, les calculs de butterfly ($R = 2$) au sein de chaque colonne (②) du modèle sont indépendants les uns des autres. Il est donc possible de calculer N/R butterfly en parallèle et d'exploiter ce parallélisme de calcul afin de réduire la latence et augmenter le débit. La complexité matérielle va alors croître linéairement avec le nombre d'opérateurs.

— Le second niveau de parallélisme ($\mathcal{P}_2$) nommé parallélisme horizontal est présenté dans la figure 1c. Chaque colonne est dépendante de sa voisine de gauche. Afin de passer outre cette dépendance, il est possible de cascader (pipeliner) plusieurs calculs de FFT en parallèle : la colonne 1 réalise le traitement des données $X(t)$ tandis que la colonne 2 traite les échantillons de l'instant $X(t+1)$. Ainsi, il est possible d'exploiter un parallélisme de $P = [1, \log_R(N)]$. Le parallélisme horizontal augmente le coût matériel (nombre d'opérateurs et bancs mémoires) d'un facteur P tandis qu'il augmente le débit. Il est à noter que la latence de traitement reste quant à elle constante.

— Le troisième niveau de parallélisme ($\mathcal{P}_3$) est localisé au niveau de l'opérateur de calcul élémentaire. Le butterfly est l'opérateur de base 2 (Radix-2) qui traite deux données en parallèle. En modifiant la base de l'opérateur élémentaire (p.e., Radix-4, Radix-8) sa complexité peut être modulée. Cela permet d'augmenter le nombre d'entrées (figure 1d) et de simplifier certains calculs intermédiaires. Cette approche est utilisée principalement pour réduire la latence et augmenter le débit des architectures matérielles [24].

Les trois niveaux de parallélisme présentés ($\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$) peuvent être utilisés partiellement ou intégralement. Ils peuvent aussi être combinés en fonction des capacités de l'architecture supportant l'implantation de l'algorithme. Les figures 1e et 1f présentent respectivement des parallélisations partielles et mixtes ($\mathcal{P}_1 + \mathcal{P}_2$) et ($\mathcal{P}_2 + \mathcal{P}_3$) fournissant des compromis différents une fois implantées. Il est à noter que ces différentes parallélisations impactent le stockage des données en mémoire ce qui implique dans certains cas des permutations complémentaires [32] comme représentées par ③ dans la figure 1a.

2.3. État de l'art et motivations

De nombreux travaux ont traité de la conception automatique d'accélérateurs matériels pour les algorithmes FFT. En effet, en plus des travaux ciblant la conception d'architectures matérielles spécifiques, d'autres approches basées sur des outils dédiés ont été proposés. En paral-

1. Il est à noter que pour améliorer la lisibilité des figures, seules 8 données ($x_n = x_0, ..., x_7$) sont représentées tandis que 16 sont nécessaires pour une FFT avec $\log_2(N) = 4$ étages.

lèle, des études se sont concentrées sur la conception de modèles comportementaux couplés à l'utilisation d'outils de HLS pour générer des architectures sous contraintes (fréquence d'horloge, ressources disponibles, circuit FPGA ciblé, etc.). Le défi consiste à décrire le comportement de la FFT de manière à mettre en exergue de manière compréhensible par l'outil de synthèse le parallélisme de calcul à exploiter lors de l'implémentation. Ainsi différentes descriptions comportementales de l'algorithme FFT ont été proposées dans [8, 15, 27]. À ces descriptions comportementales sont associées diverses directives de synthèse qui permettent de guider l'outil dans ses choix (p.e., pour les outils Xilinx : *UNROLL* ou *ARRAY PARTITION*). Comme cela a été démontré dans [21], les performances des architectures obtenues sont comparables à des architectures développées manuellement (ex. l'IP LogiCore xFFT de Xilinx). Les premiers modèles comportementaux se focalisaient sur la mise en oeuvre d'une ou deux approches de parallélisation [1, 8, 15, 21, 27, 32]. Une synthèse de ces informations est fournie dans le tableau 1.
Afin d'améliorer l'exploration de l'espace des solutions à partir de contraintes applicatives, nous avons souhaité développer une méthodologie automatisée permettant de (1) sélectionner un ou des formats de quantifications adaptés aux contraintes applicatives et (2) identifier la ou les l'architectures matérielles adaptées aux besoins. Cette seconde étape exploite plusieurs modèles comportementaux permettant de générer un large panel d'architectures.

3. Flot de conception proposé

3.1. Organisation du flot

Afin de pouvoir identifier un ensemble d'accélérateurs matériels respectant les spécifications systèmes, la méthodologie de conception a été découpée en deux parties disjointes comme illustré par la figure 2.
La première partie est chargée de l'évaluation des performances de l'algorithme FFT en fonction des formats de quantification (virgule fixe) employés au niveau des entrées/sorties et en interne. Cette analyse des performances est réalisée à partir de modèles génériques et *bit-accurate* décrivant l'algorithme FFT ①. Ces modèles sont intégrés dans l'environnement automatisé ②. À partir de signaux fournis par l'utilisateur, associés à une contrainte de précision, la méthodologie consiste à faire varier la quantification des données d'entrée, des données internes ainsi que celle des coefficients de rotation afin identifier les solutions respectant les contraintes imposées au niveau de bruit de calcul. Le bruit de calcul est estimé en termes d'erreur quadratique logarithmique moyenne vis-à-vis de la sortie d'un modèle de référence en représentation à virgule flottante. En fonction des contraintes applicatives et des résultats obtenus, plusieurs configurations peuvent être sélectionnées pour l'étape suivante.
La seconde partie de la méthodologie est décrite dans la figure 2, étape ⑤. Ce flot automatisé permet l'exploration de l'espace des solutions architecturales à partir des formats de quantification retenus à l'issue de l'étape ②. L'exploration de l'espace des solutions est effectuée à l'aide d'un outil de synthèse de haut niveau (Vitis HLS). Différents modèles comportementaux ④ bit-accurate, décrivant des variantes architecturales d'accélérateur de FFT sont employés. Ces derniers sont génériques et flexibles. Ils permettent une configuration fine des formats de quantifications, des niveaux de parallélisme, etc. Ces modèles couplés à des directives de synthèse (③), des contraintes d'implantation (ex. fréquence d'horloge, cible technologique) aboutissent à la génération d'architectures dédiées et optimisées selon le contexte applicatif. La synthèse HLS suivie des étapes de synthèse logique et de placement-routage permet d'obtenir des métriques fiables pour comparer les différentes solutions obtenues. Un classement des solutions en fonction des contraintes exprimées par le concepteur permet à ce dernier d'identifier les solutions les plus pertinentes.

3.2. Modèles comportementaux génériques

Afin d'aboutir à une exploration la plus exhaustive possible de l'espace des solutions, plusieurs modèles comportementaux (figure 2 ④) ont été décrits en C/C++. Les spécifications de ces différents modèles comportementaux ($\mathcal{M}_x$, $\mathcal{M}_y$, $\mathcal{M}_z$, ...) ont été inspirées de l'état de l'art. Ces différents modèles exploitent spécifiquement un ou plusieurs types de parallélisation (cf. section 3.2). Ils permettent de générer des architectures matérielles qui correspondent aux différents schémas de parallélisation. Un dernier modèle, plus générique $\mathcal{M}_5$ a été décrit afin de faciliter la génération d'architectures combinant plusieurs degrés de parallélisme. L'ensemble de modèles a été annoté à l'aide de directives de synthèse (*pragma*) afin de guider l'outil de synthèse d'architecture (Vitis HLS) durant les phases d'optimisations.
La liste des modèles comportementaux qui ont été décrits afin d'explorer au mieux l'espace des solutions à l'aide d'un ou de plusieurs niveaux de parallélisme est récapitulé comme suit :

— Le modèle $\mathcal{M}_1$ ne contient aucun niveau de parallélisme ($\mathcal{P}_0$). Il permet la génération d'une architecture peu complexe. Un seul butterfly est utilisé de façon séquentielle pour réaliser tous les calculs comme décrit dans la figure 1b.

— Le modèle $\mathcal{M}_2$ exploite le parallélisme horizontal. Il a été conçu à partir des travaux décrits dans [15] et [8]. La parallélisation peut être partielle ou totale ($\mathcal{P}_2$). Dans ce modèle, $\log_2(N)/G$ tranches chacune composée de G étages sont exécutées en parallèle sur des données disjointes. Chaque groupe d'étages possède un opérateur de type butterfly. À un instant t, si le pipeline d'exécution est complet, le modèle calcule $\log_2(N)/G$ FFTs en parallèle.

— Le modèle $\mathcal{M}_3$ permet d'obtenir une architecture dans laquelle il n'y a aucune réutilisation : ($\mathcal{P}_1$) et ($\mathcal{P}_2$). Elle est extrêmement coûteuse et difficilement envisageable pour des valeurs élevées de N. Les architectures produites n'ont donc pas été exploitées dans la suite des travaux.

— Le modèle $\mathcal{M}_4$ est un modèle comportemental dérivé de [32]. Ce modèle intègre un mécanisme de permutation des données à la volée pour permettre une parallélisation verticale ($\mathcal{P}_1$).

— Le modèle $\mathcal{M}_5$ correspond à une combinaison des modèles $\mathcal{M}_2$ et $\mathcal{M}_4$. Il intègre les niveaux de parallélisation ($\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$). Ce modèle permet de générer tout type de compromis architectural au niveau de la parallélisation des calculs. Toutefois, sa description comportementale guidée par la flexibilité et et aboutit à des résultats architecturaux différents de ceux des modèles à parallélisation unique.

En plus d'une flexibilité de parallélisation, ces modèles comportementaux, supportent une configuration fine du format de données. En effet, il est possible de régler de indépendamment, le format de données d'entrée, de sortie ainsi que le format données intermédiaires et le format des coefficients de rotation. Le nombre de points N sur laquelle l'algorithme FFT est appliquée est, lui aussi, configurable avant le processus de synthèse d'architecture.

4. Résultats expérimentaux

Afin de démontrer l'intérêt de la méthodologie développée, nous avons évalué sa capacité à explorer l'espace des solutions. Dans un premier temps, nous avons évalué, l'impact du format de quantification des coefficients de rotation utilisés en interne lorsque la dynamique des données d'entrées est représenté sur 16 bits. Les performances en termes d'erreur quadratique logarithmique moyenne de la sortie par rapport à un modèle de référence, à savoir *fftw3* en mode avec un format virgule flottante double précision, sont fournies dans les figures 3a et 3b

pour $N = 256$ et $N = 1024$. La quantification des coefficients de rotation est comprise entre Q_{10} et Q_{26} ($Q_X = Q_{I.(X-I)}$ avec $I = 2$). L'impact sur la complexité matérielle et les performances temporelles (débit, latence) du format de quantification est illustré au travers des données présentes dans les figures 3c et 3d. Ces figures illustrent l'évolution de la complexité matérielle de l'architecture en pourcentage maximum d'utilisation des ressources du circuit FPGA [2] et de la fréquence maximum d'utilisation de l'accélérateur matériel en fonction de la quantification des données internes. Ces données ont été obtenues après placement et routage des architectures issues du modèle $\mathcal{M}_1$ pour une cible Kintex Ultrascale+ (xcku5p-sfvb784-2-i). On peut constater sur la figure 3c un saut dans le coût matériel correspondant à la saturation d'une ressource du circuit (c.-à-d. DSP, BRAM). Il est à noter que lors de la synthèse d'architecture, l'ensemble des architectures a été contrainte par une fréquence de fonctionnement fixée à 100 MHz.
Un deuxième jeu d'expérimentations a été réalisé afin d'illustrer les différents compromis atteignable en fonction des modèles comportementaux. Lors de cette évaluation, nous avons fixé le format de quantification des coefficients de rotation à 20 bits. Les figures 4a et 4b présentent l'évolution des caractéristiques des différentes architectures obtenues par rapport à l'architecture issue du modèle $\mathcal{M}_1$. Ces données démontrent que les modèles développées permettent l'obtention de nombreux compromis entre débit, latence et complexité matérielle. Par exemple, à complexités matérielles équivalentes, les solutions (1) et (6) optimisent respectivement la latence ou le débit.
La généricité et la flexibilité des modèles comportementaux développés permet l'obtention d'un nombre important de compromis architecturaux. Afin de démontrer cette caractéristique, des expérimentations supplémentaires ont été réalisées pour $N = 4096$. Les figures 5a et 5b illustrent l'espace couvert par les différentes architectures lorsque l'ensemble des valeurs de configuration est exploité avec une fréquence de fonctionnement contrainte à 100 MHz. La figure 6, quant à elle expose l'ensemble des solutions architecturales lorsque différentes contraintes de fréquence de fonctionnement sont spécifiées pour la synthèse d'architecture. Ces courbes illustrent l'intérêt de la méthodologie développée qui permet à l'utilisateur de sélectionner la solution la plus efficace vis-à-vis des contraintes applicatives. Il est intéressant de noter que l'étape d'exploration de l'espace des solutions prend entre 10 minutes et plus d'une heure par jeu de paramètre sur une machine équipée d'un CPU Intel Xeon E5-1620.

5. Conclusions et perspectives

Le flot de conception proposé permet à la fois (1) d'évaluer et de sélectionner des formats de quantifications selon les contraintes applicatives, et (2) de générer avec le support d'outils de synthèse d'architecture (ex. Vitis HLS) des implantations matérielles offrant différents compromis entre complexité et performance. Les architectures matérielles générées peuvent être utilisées de manière autonome, couplées avec un coeur ARM (Zynq) ou couplées avec un coeur x86 (Alveo). Les résultats expérimentaux présentés illustrent la capacité de la méthodologie à explorer finement l'espace des solutions possibles à partir de contraintes applicatives. Certaines des solutions développées ont été testées dans un système Hardware In the Loop (HIL) et implanté dans une plate-forme SDR de la société Arelis à base de Zynq 7030 (xc7z030tfbg485-2) en sortie de flot afin d'en vérifier le bon fonctionnement. Les futurs travaux porteront sur une généralisation du flot afin d'effectuer du prototypage rapide de chaînes de traitement radar et plus largement de systèmes hyperfréquences. En parallèle, l'évaluation des modèles sur des plateformes Intel FPGA via Intel HLS est prévue.

2. Les circuits FPGA sont composées d'éléments hétérogènes. Pour simplifier l'analyse des résultats, le choix de mesurer la complexité matérielle comme étant le % maximum d'utilisation des ressources (LUTs, FFs…) a été fait.

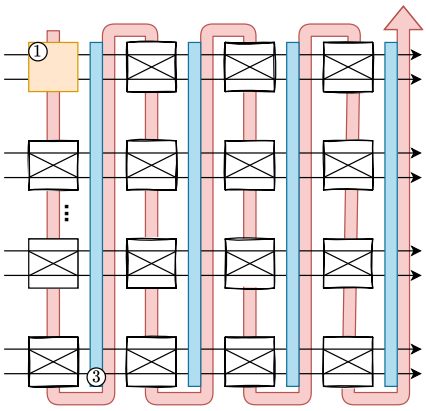
Bibliographie

1. Akkad (G.), Mansour (A.), ElHassan (B.), Roy (F. L.) et Najem (M.). – FFT radix-2 and radix-4 FPGA acceleration techniques using HLS and HDL for digital communication systems. – In *Proceedings of the International Multidisciplinary Conference on Engineering Technology (IMCET)*, 2018.
2. Chen (R.) et Prasanna (V. K.). – Energy optimizations for FPGA-based 2-D FFT architecture. – In *Proceedings of High Performance Extreme Computing Conference (HPEC)*, 2014.
3. Chuppala (H.), Chinthalapalli (S.), Aditi (V.) et Karunavathi (R. K.). – Modified Cooley-Tukey algorithm for implementation of integrated serial FFT/IFFT processor in half-duplex OFDM systems. – In *Proceedings of International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, 2017.
4. Cooley (J. W.) et Tukey (J. W.). – An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.*, vol. 19, 1965.
5. Delomier (Y.), Le Gal (B.), Crenne (J.) et Jego (C.). – Model-Based Design of Flexible and Efficient LDPC Decoders on FPGA Devices. *Journal of Signal Processing Systems, Springer*, vol. 92, n7, 2020.
6. Delomier (Y.), Le Gal (B.), Crenne (J.) et Jego (C.). – Model-based Design of Hardware SC Polar Decoders for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, n2, 2020.
7. Eugin Hyun, Sang-Dong Kim, Yeong-Hwan Ju, Jong-Hun Lee, Eung-Noh You, Jeong-Ho Park, Dong-Jin Yeom, Sang-Hyun Park et Seung-Gak Kim. – FPGA based signal processing module design and implementation for FMCW vehicle radar systems. – In *Proceedings of CIE International Conference on Radar (RADAR)*, 2011.
8. Fingeroff (M.). – *High-Level Synthesis Blue Book*. – Xlibris Corporation, 2010.
9. Frigo (M.) et Johnson (S.). – The Design and Implementation of FFTW3. *Proceedings of the IEEE*, vol. 93, n2, 2005.
10. Gisselquist (D.). – A Generic Piplined FFT Core Generator. – `https://github.com/ZipCPU/dblclockfft`, 2022. En ligne ; Dernier Accès le 2021-04-12.
11. Intel. – High-Level Synthesis Compiler - Intel® HLS Compiler. – `https://www.intel.com/content/www/fr/fr/software/programmable/quartus-prime/hls-compiler.html`. En ligne ; Dernier Accès le 2021-04-12.
12. Intel. – Intel® Agilex™ FPGA and SoC FPGA Family. – `https://www.intel.com/content/www/us/en/products/details/fpga/agilex.html`. En ligne ; Dernier Accès le 2022-04-14.
13. Jaber (M. A.), Massicotte (D.), Jaber (R. A.) et Nesmith (K.). – An Efficient Data Parallelization of the Radix-2 ^{3} (Carbon) FFT on GPU/CPU. – In *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, 2019.
14. Jacko (P.) et Kravets (O.). – Spectral Analysis by STM32 Microcontroller of the Mixed Signal. – In *Proceedings of International Conference on Modern Electrical and Energy Systems (MEES)*, 2019.
15. Kastner (R.), Matai (J.) et Neuendorffer (S.). – Parallel Programming for FPGAs. *ArXiv e-prints*, 2018.
16. Li (Z.), Jia (H.), Zhang (Y.), Chen (T.), Yuan (L.) et Vuduc (R.). – Automatic Generation of High-Performance FFT Kernels on Arm and X86 CPUs. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 31, n8, 2020.
17. Milder (P. A.), Franchetti (F.), Hoe (J. C.) et Püschel (M.). – Computer generation of hardware for linear digital signal processing transforms. *ACM Transactions on Design Automation*

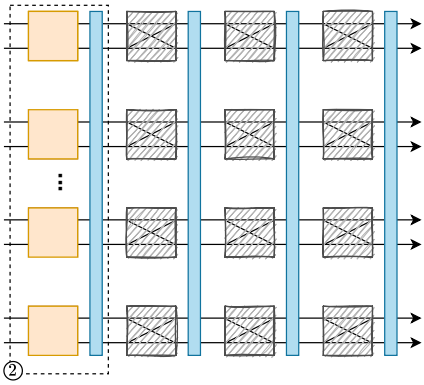
of Electronic Systems (TODAES), vol. 17, n2, 2012.

18. O'Sullivan (J.), Weiss (S.) et Rice (G.). – Automatic fft code generation for fpgas with high flexibility and human readability. – In *Proceedings of Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2011.
19. Ouerhani (Y.), Jridi (M.) et Alfalou (A.). – Area-Delay Efficient FFT Architecture Using Parallel Processing and New Memory Sharing Technique. *Journal of Circuits, Systems, and Computers, World Scientific*, vol. 21, n6, 2012.
20. Saeed (A.), Elbably (M.), Abdelfadeel (G.) et Eladawy (M.). – Efficient fpga implementation of fft/ifft processor. *International Journal of circuits, systems and signal processing*, vol. 3, n3, 2009.
21. Salaskar (A.) et Chandrachoodan (N.). – FFT/IFFT implementation using Vivado™ HLS. – In *Proceedings of 20th International Symposium on VLSI Design and Test (VDAT)*, 2016.
22. Serre (F.) et Puschel (M.). – A DSL-Based FFT Hardware Generator in Scala. – In *Proceedings of 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018.
23. Software (S. D. I.). – Catapult c++/SystemC Synthesis. – `https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/hls/c-cplus/`.
24. Sun (M.), Tian (L.) et Dai (D.). – Radix-8 FFT processor design based on FPGA. – In *Proceedings of 5th International Congress on Image and Signal Processing (ICSIP)*, 2012.
25. Takahashi (D.). – Implementation and Evaluation of Parallel FFT Using SIMD Instructions on Multi-core Processors. – In *Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA)*, 2007.
26. Takaki (S.) et Yamagishi (J.). – A deep auto-encoder based low-dimensional feature extraction from FFT spectral envelopes for statistical parametric speech synthesis. – In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.
27. Tran (M.-T.), Gautier (M.) et Casseau (E.). – On the FPGA-Based Implementation of a Flexible Waveform from a High-Level Description : Application to LTE FFT Case Study. – In Noguet (D.), Moessner (K.) et Palicot (J.) (édité par), *Proceedings of Cognitive Radio Oriented Wireless Networks (CROWN)*. Springer, 2016.
28. Xilinx. – Alveo. – `https://www.xilinx.com/products/boards-and-kits/alveo.html`. En ligne ; Dernier Accès le 2021-04-12.
29. Xilinx. – SoCs, MPSoCs and RFSoCs. – `https://www.xilinx.com/products/silicon-devices/soc.html`. En ligne ; Dernier Accès le 2021-04-12.
30. Xilinx. – Fast Fourier Transform v9.0 LogiCORE IP Product Guide. – `https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_1/pg109-xfft.pdf`, 2021. En ligne ; Dernier Accès le 2021-11-08.
31. Xilinx. – Vitis High-Level Synthesis User Guide. – `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1399-vitis-hls.pdf`, 2021. En ligne ; Dernier Accès le 2021-11-08.
32. Xu (G.), Low (T. M.), Hoe (J. C.) et Franchetti (F.). – Optimizing fft resource efficiency on fpga using high-level synthesis. – In *Proceedings of High Performance Extreme Computing (HPEC)*, 2017.

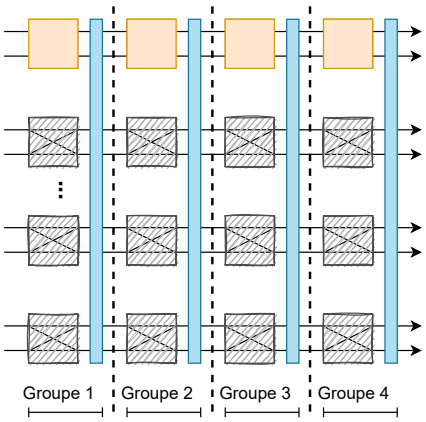
Annexes



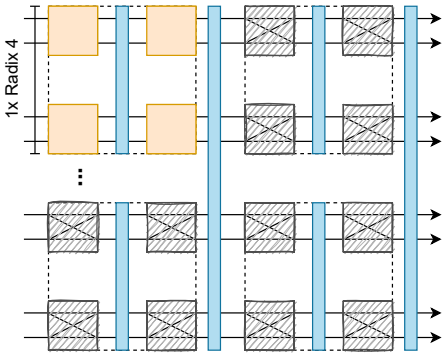
(a) Réutilisation BF, réutilisation étage, Radix-2



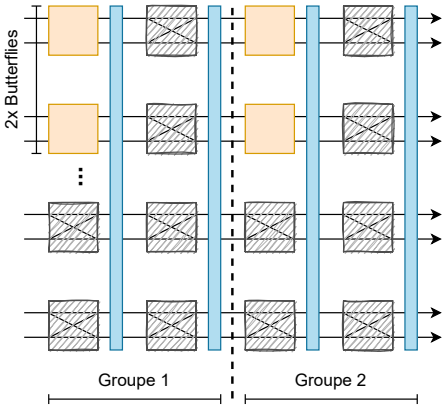
(b) Parallélisation BF, réutilisation étage, Radix-2



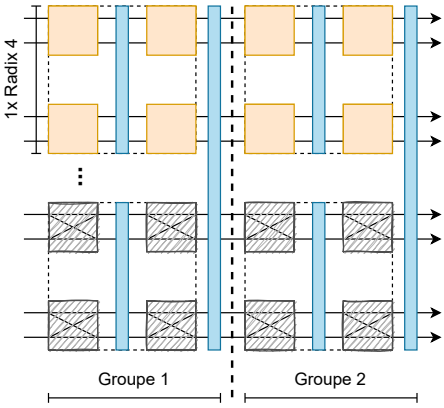
(c) Réutilisation BF, pipelining étage, Radix-2



(d) Réutilisation BF, Réutilisation étage, Radix-4



(e) Semi-parallélisation BF & étage, Radix-2



(f) Réutilisation BF, Pipelining étage, Radix-4

FIGURE 1 – Différentes stratégies de parallélisation de l'algorithme FFT sur 16 points (seuls 8 points parmi les 16 sont représentés pour une question de lisibilité).

	Année	Méthode d'implem.	Taille (N)	Radix (R)	Butterfly (B)	Etages (S)	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
Ouerhani et Al. [19]	2011	RTL	4^k	4	1	$\log_R(N)$		✓	✓
Xilinx LogiCore xFFT 9.1 [30]	2018	RTL	[8; 65536]	{2, 4}	1	$\{1; \log_2(N)\}$		✓	✓
Serre et Al. [22]	2018	Gen. RTL	2^k	$\geqslant 2$	[1; N/R]	$[1; \log_R(N)]$	✓	✓	✓
Tran et Al. [27]	2016	HLS	LTE	{2,Mixed 2/3}	1	$[1; \log_2(N)]$			✓
Salaskar et Al. [21]	2016	HLS	512	2	1	1			
Xu et Al. [32]	2017	HLS	[128; 4096]	{2, 4, 8}	{1, 2, 4}/{1, 2}/{1}	1	✓		✓
Akkad et Al. [1]	2018	HLS	8, 16	{2, 4}	1	1			✓
Modèle dédié $\mathcal{M}_1$	-	HLS	[64; 8192]	2	1	1			
Modèle dédié $\mathcal{M}_2$	-	HLS	[64; 8192]	2	1	$[1; \log_2(N)]$		✓	
Modèle dédié $\mathcal{M}_3$	-	HLS	[64; 4096]	2	N/2	$\log_2(N)$	✓	✓	
Modèle dédié $\mathcal{M}_4$	-	HLS	[64; 4096]	2	{1, 2, 4}	1	✓		
Modèle générique $\mathcal{M}_5$	-	HLS	[64; 4096]	{2, 4, 8}	{1, 2, 4}/{1, 2}/{1}	$[1; \log_R(N)]$	✓	✓	✓

TABLE 1 – Travaux connexes et caractéristiques des architectures générées

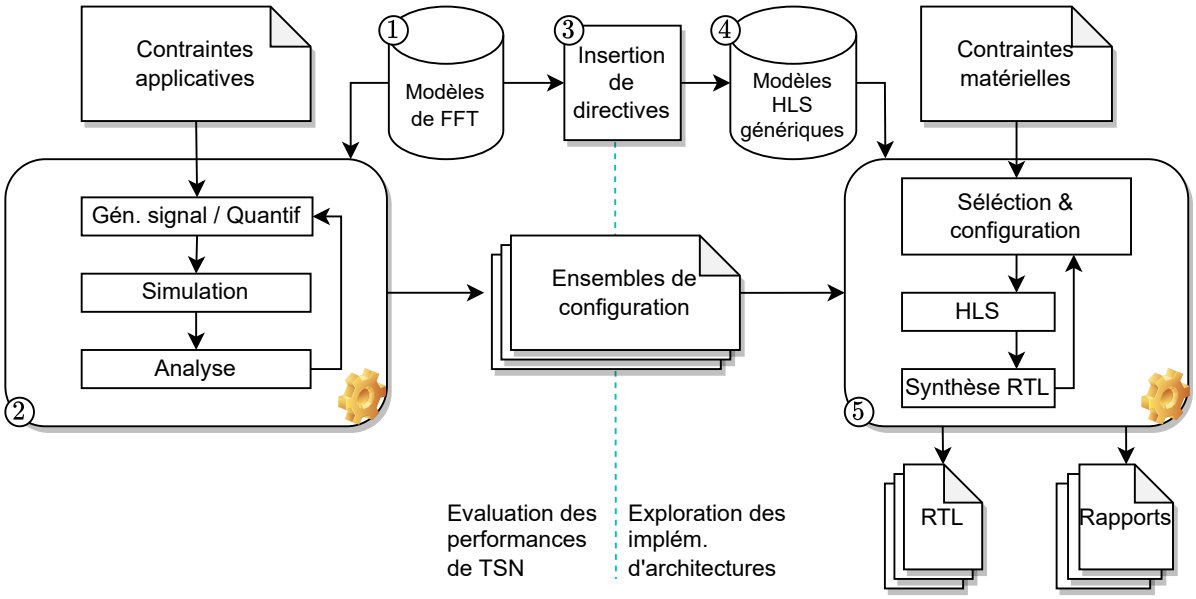
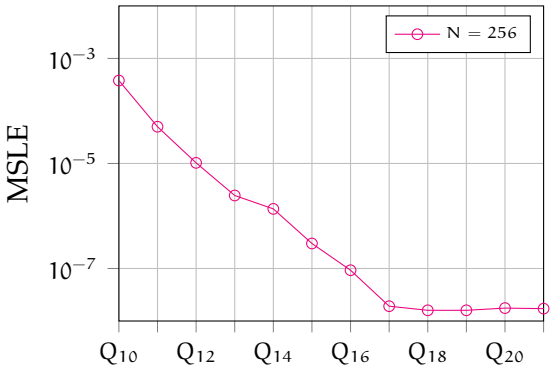
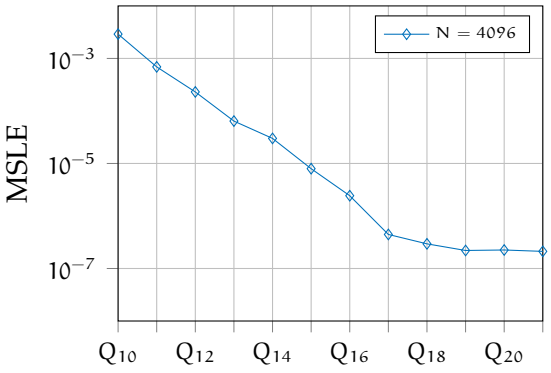


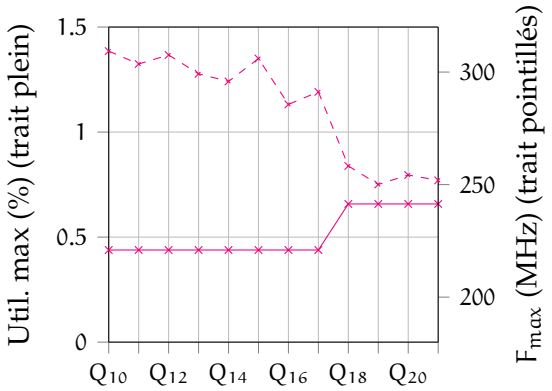
FIGURE 2 – Environnement d'exploration d'architecture



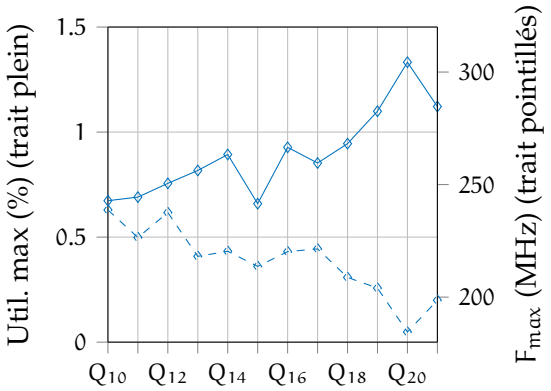
(a) Quantification pour N = 256



(b) Quantification pour N = 4096

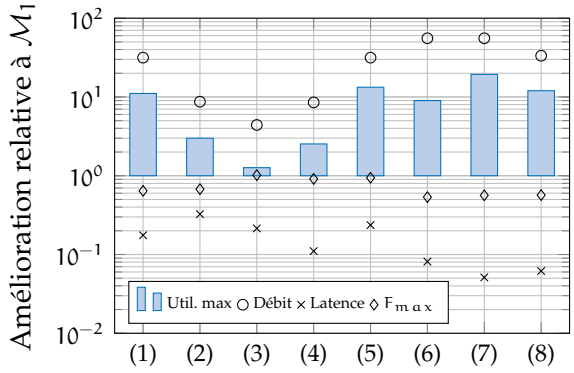


(c) Quantification pour N = 256

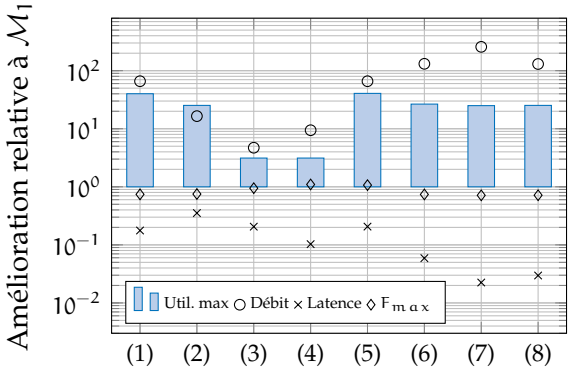


(d) Quantification pour N = 4096

FIGURE 3 – Effets du format de quantification des coefficients de rotation - Entrées codées sur 16 bits avec une augmentation de la dynamique interne (1 bit par étage).

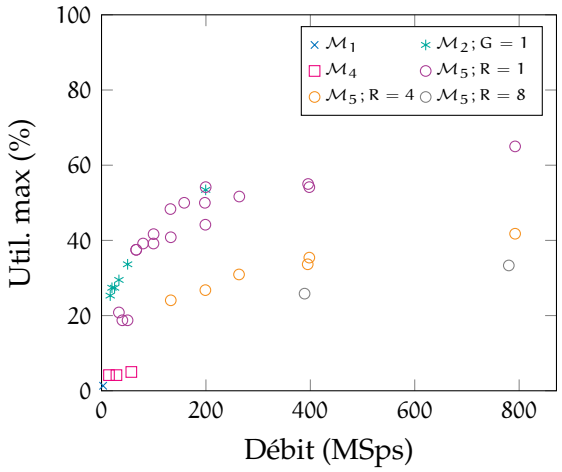


(a) Architectures pour N = 64

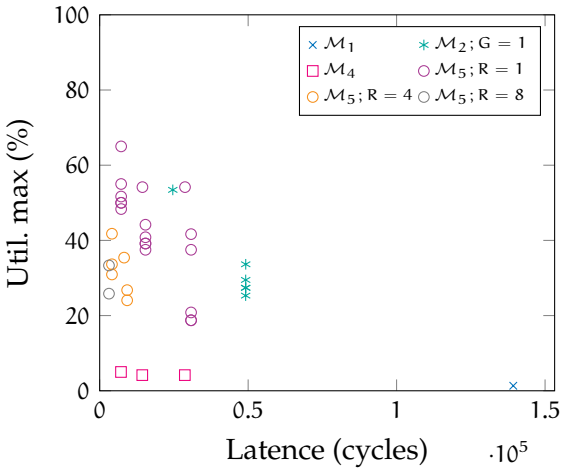


(b) Architectures pour N = 4096

FIGURE 4 – Compromis architecturaux possibles en fonction des parallélisations exploitées @100MHz - (1) $\mathcal{M}_2$; G = 1, (2) $\mathcal{M}_2$; G = 2, (3) $\mathcal{M}_4$; B = 1, (4) $\mathcal{M}_4$; B = 2, (5) $\mathcal{M}_5$; B = 1; R = 2; G = 1, (6) $\mathcal{M}_5$; B = 1; R = 4; G = 1, (7) $\mathcal{M}_5$; B = 1; R = 8; G = 1, (8) $\mathcal{M}_5$; B = 2; R = 4; G = 2



(a) Utilisation de ressource maximale en fonction du débit



(b) Utilisation de ressources maximale en fonction de la latence

FIGURE 5 – Espace des solutions exploré pour N = 4096 @100MHz ; Entrées codées sur 16bits et coefficients de rotation 20bits

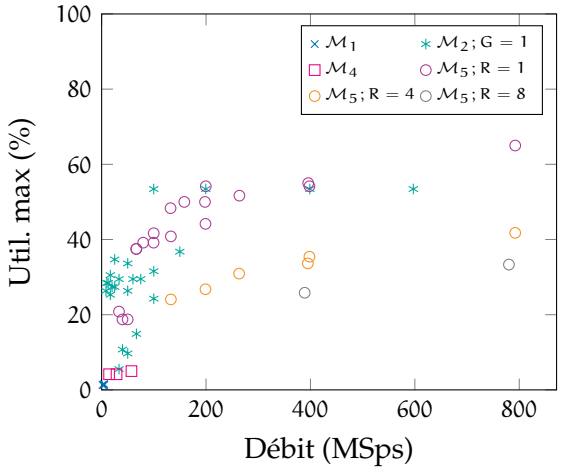


FIGURE 6 – Espace des solutions exploré pour N = 4096 @f = 50, 100, 200, 300MHz ; Entrées codées sur 16bits et coefficients de rotation 20bits