Compas'2023 : Parallélisme/ Architecture / Système
LISTIC/USMB - Annecy France, 4-7 juillet 2023

Conception d'architectures de traitement radar FMCW sur FPGA à base de modèles comportementaux

Hugues ALMORIN[1,2], Bertrand LE GAL[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
Depuis plusieurs années, des travaux de recherche ont permis de démontrer l'intérêt des outils de synthèse haut niveau (HLS) pour concevoir des accélérateurs matériels à implémenter sur ASIC/FPGA. Cela permet de réduire notablement les temps d'intégration d'algorithmes de traitement de signal et facilite, à l'aide de descriptions comportementales appropriées, l'exploration de l'espace des solutions architecturales. Dans cet article, nous expérimentons la capacité de ces outils à concevoir des chaines complètes de traitement du signal. Le cadre applicatif sélectionné est relatif aux systèmes radar de type FMCW. Nous positionnons aussi les performances atteintes sur circuit FPGA vis-à-vis d'implémentations de ces systèmes sur multicoeurs SIMD.

Mots-clés : conception à base de modèles, synthèse d'architectures, HLS, FPGA, traitement radar, FMCW, multicoeurs SIMD.

---

1. Introduction

Les systèmes radar (*RAdio Detection And Ranging*) sont utilisés dans de nombreux domaines et se démocratisent dans les produits destinés au grand public. La complexité calculatoire maitrisée ainsi que la simplicité de fonctionnement des radars à ondes continues modulées en fréquence (*Frequency Modulated Continuous Wave*, FMCW) en font un choix privilégié. À partir des données reçues de l'étage de réception analogique, numérisées et descendues en bande de base, l'objectif d'un système de traitement radar est de permettre l'observation de la position et la vitesse d'objets détectés. Le contexte applicatif est celui des radars embarqués dans des aéronefs ou dans des automobiles, pour lesquels les traitements numériques doivent s'effectuer en temps réel, et ce, avec des contraintes matérielles et énergétiques fortes.
Pendant plusieurs décennies, ces traitements en bande de base dont la complexité calculatoire est élevée étaient exclusivement effectués par des circuits numériques dédiés (ASIC) [14, 22]. Cependant, l'évolution des technologies couplée à des marchés à faible volume ont permis aux circuits FPGA de s'imposer dans de multiples domaines, et cela, grâce à des temps de conception réduits et à leur capacité à être mis à jour [4, 16, 18, 19, 28]. En parallèle, les innovations réalisées dans le domaine des architectures programmables logiciellement, multicoeurs (DSP

& CPU) ou *many*-coeurs (GPU) en font aussi des cibles architecturales exploitables pour des systèmes radar. En effet, ces derniers offrent des niveaux de flexibilité supérieurs tout comme des puissances de calcul élevées via l'utilisation de différents paradigmes de programmation parallèle (ex. SIMD, MIMD, SIMT). Cependant, ces performances sont obtenues au détriment de la consommation énergétique. Des bibliothèques logicielles telles que *fftw* ou *LAPACK* facilitent la mise au point de systèmes radars performants [8, 24, 25, 27, 31]. Cependant, des systèmes temps réel peuvent être rapidement limités dans leurs usages lorsque des informations provenant de multiples canaux sont considérées ou lorsque la consommation énergétique est la contrainte majeure.

Afin de rendre les circuits FPGA compétitifs vis-à-vis des circuits programmables en termes de flexibilité et de l'exploitation du parallélisme, des outils de synthèse haut niveau [1, 2, 10, 20, 29] (*High-Level Synthesis*, HLS) ont été proposés et permettent de simplifier les étapes de conception d'architectures numériques dédiées. En effet, ces outils, à partir des comportements décrits par l'utilisateur dans un langage de haut niveau (C, C++, SystemC, ...), génèrent des descriptions architecturales au niveau RTL en VHDL ou en Verilog adaptée aux contraintes applicatives (p. ex. fréquence d'horloge, débit) ainsi qu'aux ressources du circuit ciblé. D'un point de vue industriel, les différentes étapes de simulation présentes dans ces outils, autant avant qu'après synthèse HLS, permettent de raccourcir les phases de vérifications, qu'elles soient au niveau *bit-accurate* ou bien *cycle-accurate*. Toutefois, la qualité des architectures générées dépend fortement de la qualité des modèles comportementaux ainsi que de la maitrise des outils HLS. Cela a pu engendrer des différences marquées entre les performances d'IP conçues directement au niveau RTL et celles produites par les outils de synthèse HLS, avec des écarts atteignant parfois un facteur 100 [7, 11].

Ainsi, de nombreux travaux se sont donc concentrés sur la conception et sur l'optimisation d'implémentation d'algorithmes de traitement de signal [30] et de l'image [21] ou de communications numériques [11]. Cependant, peu de travaux ont concerné l'implémentation et l'évaluation de système de traitement temps réel complet. Dans cet article, nous nous intéressons à évaluer (1) la pertinence d'outils HLS pour concevoir des systèmes réels à partir d'une description comportementale unique, et (2) notre capacité à décrire de tels modèles efficacement. Le cadre de cette étude est une chaine de traitement de signaux radar FMCW.

Le reste de l'article est organisé de la manière suivante : la section 2 présente les algorithmes mis en uvre dans une chaine de traitement radar FMCW ainsi que les travaux connexes. Puis, la section 3 présente les stratégies de parallélisation et les directives déployées en vue de guider les outils de synthèse d'architecture. Les résultats expérimentaux sont présentés dans la section 4 et comparés à des implémentations sur CPU.

2. Principe du radar FMCW

Les radars FMCW qui permettent d'estimer, par exemple, la position et la vitesse des objets en mouvement, utilisent des formes d'ondes composées de signaux modulés en fréquence. Les séquences émises sont composées de M *chirps*, éléments dont la fréquence évolue dans le temps (rampe, triangle, sinusoïde, ...), composés chacun de N échantillons complexes (I/Q), comme illustré par la figure 1. Afin de pouvoir estimer les distances et les vitesses d'objets à partir des signaux réfléchis, il est nécessaire de traiter ces signaux reçus. Le *frontend* d'une chaine de traitement radar FMCW typique est schématisé dans la figure 2.

Le signal émis par le radar est mélangé avec le signal reçu. Puis, ce signal est filtré par un filtre passe bas pour ne garder que le signal de battement qui est un signal basse fréquence. Dans un second temps, cette séquence est mise en forme à l'aide de différents filtres et amplifica-

teurs à faible bruit, avant, enfin, sa conversion analogique-numérique. Ce *frontend* analogique est associé à une étape de traitement numérique de l'information qui permet d'établir une carte distance-Doppler. Cette dernière permet de facilement visualiser les objets présents dans l'environnement ainsi que leurs vitesses de déplacement. Afin d'obtenir cette carte distance-Doppler, des algorithmes usuels de traitement de signal sont nécessaires.
La figure 3 représente une chaine de traitements usuelle. Dans cet exemple, le système de traitement numérique possède un canal d'émission et de réception. Le système émet des *chirps* en forme de dents de scie. L'entrée du récepteur est un flux de données I/Q, données temporisées et structurées sous la forme de M paquets contenant chacun N échantillons complexes, provenant de la partie analogique du radar (Fig. 3). En sortie de la chaine de traitement, les données sont restituées sous la forme de N paquets de taille M constituant la carte distance-Doppler. Afin de produire cette carte, différents traitements sont appliqués sur les données d'entrée. De manière synthétique, les opérations suivantes sont appliquées séquentiellement :

- Le signal complexe est tout d'abord mis en forme lors de l'étape ① à l'aide d'une opération de filtrage associée à une pondération par une fenêtre d'apodisation. En général, le filtrage est réalisé par un filtre passe haut (FIR) de profondeur L_1 et l'apodisation utilise du fenêtrage classique (ex. Hann ou Hamming). Les complexités calculatoires sont respectivement de $\mathcal{O}(N \times L_1)$ et $\mathcal{O}(N)$.
- Le signal mis en forme est ensuite transposé dans le domaine fréquentiel lors de l'étape ② via l'utilisation d'une FFT. Cette dernière est appliquée M fois sur des paquets de N échantillons. Ce passage en représentation fréquentielle permet d'extraire les fréquences de battement. La complexité calculatoire de cette étape évolue de la manière suivante : $\mathcal{O}(N \log N)$.
- Afin de poursuivre les traitements, il est nécessaire durant l'étape ③ d'entrelacer / de transposer les M *burst* de N données sous la forme de N *burst* de M données. Cette opération ne peut être faite qu'après la réception des $N \times M$ valeurs. Cela requiert donc une mise en mémoire de l'ensemble des données avant de poursuivre les traitements.
- Les données transposées sont ensuite remises en forme au cours de l'étape ④. La nature des traitements appliqués est similaire à ceux décrits durant l'étape ①, mais la complexité calculatoire est alors $\mathcal{O}(M \times L_2)$ et $\mathcal{O}(M)$.
- La dernière étape ⑤ consiste à passer les paquets de M échantillons dans le domaine fréquentiel à l'aide d'une seconde FFT de taille M. Cette transformation permet d'extraire les fréquences Doppler du signal reçu. La complexité calculatoire résultante est ici en $\mathcal{O}(M \log M)$.

Cette chaine de traitement radar classique peut se décliner de nombreuses façons en fonction des domaines applicatifs et des contraintes du système. Par exemple, la structure et le choix des éléments de filtrage et de pondération peut changer en fonction de la nature des signaux et des objets à détecter. Il en va de même pour les valeurs des paramètres N et M. Ainsi, même si la chaine de traitement reste toujours similaire, des modifications des spécifications peuvent nécessiter un redéveloppement complet d'un accélérateur matériel au niveau RTL. C'est la principale raison pour laquelle nous évaluons dans cet article l'intérêt et les limitations des approches basées sur l'utilisation de modèles comportementaux qui doivent permettre d'offrir la flexibilité nécessaire.

3. Conception d'un système de traitement numérique de signal radar à partir de modèles comportementaux

L'utilisation d'outils de HLS pour concevoir des architectures numériques sous contraintes nécessite tout comme l'usage efficace de circuits GPU [12, 17, 23] l'écriture de modèles comportementaux pertinents. Dans un premier temps, nous allons décrire ici les modèles comportementaux génériques développés afin de guider l'outil de HLS, AMD-Xilinx Vitis HLS 2022.2 dans le cadre de cette étude, dans l'exploitation du parallélisme de calcul intrinsèque au système de traitement radar. Ce modèle se doit d'être le plus générique possible au niveau de ses performances en termes de débit/latence et également en termes de précision de traitement effectué. C'est à ce niveau que le parallélisme de calcul grain fin est mis en exergue tout comme les directives permettant de contraindre l'outil afin de permettre une exploration future de l'espace des solutions architecturales.
Pour pouvoir décrire et implémenter le système décrit précédemment, nous avons décidé de structurer notre modélisation de manière hiérarchique selon 3 niveaux :

1. Le niveau algorithmique - Les modèles algorithmiques sont décrits en langage C++. L'utilisation de templates permet de garder une certaine généricité au niveau du format de données (float, fixed), ainsi qu'au niveau de la forme d'onde (N et M). Les directives HLS telles que *UNROLL* et *PIPELINE* ou *ARRAY_PARTITION* permettent de contraindre et d'orienter le logiciel HLS afin de synthétiser l'architecture matérielle souhaitée. En fonction du type d'architecture souhaité, l'utilisation de plusieurs descriptions formulées différemment peut s'avérer judicieuse [1]. Certaines parties de la description telles que les tableaux contenants les valeurs des fenêtres d'apodisation ou les coefficients de filtres FIR sont pré-calculés et autogénérées.
2. Le niveau tâches - Le deuxième niveau permet de spécifier les acteurs du système sous la forme de tâches autonomes. Les fonctions C++ sont encapsulées seules ou regroupées au sein de tâches (*hls::task*) afin de spécifier un comportement séquentiel ou parallèle de ces opérations. Le regroupement de tâches associé à l'utilisation de la directive INLINE permet à l'outil de synthèse de mutualiser les ressources matérielles lorsque cela est possible.
3. Le niveau système - Le troisième niveau permet de décrire les tâches inclues dans le traitement système ainsi que la manière dont elles communiquent entre elles. Cela permet décrire un parallélisme de tâches élevé ou bien de favoriser la réutilisation des ressources matérielles en fonction des contraintes applicatives. Un exemple de traitement FMCW mono-canal est présenté dans la figure 4. Il est composé de cinq sous modules indépendants traitant des données en flux continu. Il a donc été décidé de les interconnecter à l'aide de FIFO (*First In, First Out*) spécifiées dans le modèle à l'aide du type générique *hls::stream*. L'ensemble des acteurs étant indépendants à ce niveau, la directive HLS *DATAFLOW* a aussi été employée pour imposer une décentralisation du contrôle du système.

Le découpage en 3 niveaux d'abstraction permet la modélisation de modèles dérivés directement de la figure 8, afin de réduire par exemple la latence de traitement. Cette organisation du système peut être par exemple obtenue en dupliquant les taches positionnées après l'entrelaceur comme cela est schématisé dans la Figure 9. A l'opposé pour les systèmes dans lesquels

1. La conception d'un modèle générique et flexible décrivant l'algorithme de FFT efficacement a été une tâche complexe ayant donné lieu à une communication antérieure [6].

les performances de traitement d'une voie sont supérieures aux contraintes, il est possible de décrire des architectures réutilisant temporellement et spatialement les ressources comme cela est schématisé dans les figures 10, 11 et 12. Ces architectures peuvent être obtenues à partir des paramètres extraits des templates et des directives de parallélisation dans le modèle au niveau système.

4. Résultats expérimentaux

Afin d'étudier la pertinence de l'approche pour concevoir des systèmes de traitement radar, nous avons évalué les performances de nos modèles en testant les architectures générées sur un circuit FPGA de type Zynq UltraScale+. Dans un premier temps, plusieurs cas d'usage ont été étudiés pour comparer ces architectures générées avec celles produites en interne par la société ARELIS. Ensuite, nous avons positionné les architectures produites grâce à l'approche présentée par rapport à des solutions logicielles, plus flexibles mais moins efficaces d'un point de vue énergétique.

4.1. Performances absolues

Les architectures de niveau RTL ont été générées à l'aide de l'outil Vitis HLS 2022.2 et déployées dans un premier temps sur un circuit *zu4ev*, un MPSoC FPGA de chez AMD-Xilinx. Une partie de ces différents systèmes a été validée en conditions réelles sur carte pour vérifier la validité des architectures produites ainsi que les niveaux de performance estimés lors de la synthèse d'architecture. Pour ce faire, les architectures ont été insérées au sein du système de test modélisé dans la Figure 5. Ce système pilote des accès DMA via le coeur ARM afin de (1) fournir un ensemble de données générées par une machine hôte en entrée de l'architecture et (2) de récupérer ses sorties afin de valider fonctionnellement le système. Ce banc de test permet également d'avoir une mesure fine des temps de traitement, y compris les temps de transfert DMA.

Les caractéristiques d'une partie des différentes architectures par l'approche proposée sont présentées dans les Figures 6 et 7. Différentes stratégies de parallélisation issues des Figures 8 à 12 ont été employées. Les résultats rapportés ont été obtenus pour des chaînes de traitement radar possédant $N \in \{512, 1024, 2048\}$ et $M \in \{64, 32, 16\}$ et $Q = \{1, 5\}$ pour démontrer la flexibilité de l'approche. Comme le démontre les Figures 6 et 7, un nombre important de compromis architecturaux peuvent être obtenus tant au niveau de la complexité, du débit que de la latence de traitement. Certaines des solutions architecturales obtenues sont comparables à $\pm 20\%$ près aux solutions architecturales développées chez ARELIS pour des performances similaires.

4.2. Performances relatives

Afin de positionner les solutions produites à partir de la méthodologie à base d'outils de synthèse HLS et de modèles comportementaux, nous avons implanté ces mêmes chaînes de traitement radar sur des architectures multi-coeurs SIMD. Les chaînes radar ont été réécrites en C++ pour pouvoir dans un premier temps bénéficier des capacités d'auto-vectorisation de GCC et LLVM. Ensuite, pour exécuter efficacement les calculs de la FFT, les bibliothèques open-source implantant l'algorithme de Cooley-Tukey [9] adaptées au calcul parallèle sur cible multi-coeurs SIMD ont été évaluées. Les bibliothèques FFTW3 *fftw3* [13] et *PFFFT* [26] ont été sélectionnées, respectivement pour les plateformes INTEL (AVX512 [15]) et ARM (NEON [3]). Enfin, l'opération d'entrelacement a été décrite spécifiquement à l'aide d'intrinsèques SIMD afin de la rendre efficace. L'absence, à notre connaissance, de bibliothèques SIMD optimisées pour le traitement FFT de données en virgule fixe (16b/32b) nous a contraint à modéliser ces chaînes de traitement

en manipulant des données flottantes.
Cette chaîne de traitement radar optimisée pour des cibles multicoeurs SIMD a été déployée sur trois plateformes distinctes :

- Un ordinateur portable intégrant un coeur Intel i7-11800H (8 coeurs physiques, AVX512, @2.3 GHz),
- Une plateforme Nvidia Jetson AGX Xavier (8 coeurs ARM, NEON @ 2.3 GHz),
- Le coeur ARM Cortex-A53 inclus dans une carte d'évaluation basée sur un SoC ZCU4EV (2 coeurs ARM NEON, @ 1.2 GHz).

Les résultats en termes de débit, de latence et de consommation d'énergie que nous avons obtenus sont récapitulés dans le Tableau 1. Ce tableau intègre aussi les résultats de plusieurs architectures implantées sur différents circuits FPGA (cf. section précédente). Les valeurs de débit et de latence rapportées intègrent les temps de transfert entre la mémoire DDR du système et l'IP via les DMA.
Les résultats obtenus sur la cible INTEL i7-11800H et la Nvidia Jetson AGX Xavier montrent des niveaux de performance élevés, dépassant les 100 MSps. Cependant, lorsque plusieurs voies sont traitées en parallèle, une diminution des performances est observée, principalement à cause des limites thermiques imposées par le matériel. Ces dernières imposent une baisse de la fréquence d'horloge des coeurs lors d'une charge de travail importante (*Thermal throttle*).
Les accélérateurs matériels issus de notre approche offrent des débits inférieurs à ceux des implantations logicielles. Cependant, le FPGA cible de l'étude a une complexité matérielle limitée. Malgré cela des débits supérieurs à 100 MSps ont été obtenus avec une consommation d'énergie < 1 Watt rendant l'approche pertinente pour les systèmes embarqués contraints. Il est à noter que l'utilisation de circuits FPGA plus complexes permettrait de déployer plusieurs chaînes de traitement en parallèle ce qui augmenterait le débit.

5. Conclusion

Dans cet article, nous avons évalué la capacité d'un outil de HLS à générer des systèmes complets de traitement radar. Les outils de HLS associés aux modèles comportementaux que nous avons développés nous ont permis de générer des architectures matérielles compétitives par rapport à des implémentations CPU lorsque l'aspect énergétique est pris en considération. Le coût matériel des architectures matérielles ainsi conçues, est cependant 20 à 30% plus complexes que celle développées au niveau RTL. Il est à noter que les codes sources permettant de simuler et synthétiser ces travaux sont disponibles en open-source [5].

Bibliographie

1. Catapult High-Level Synthesis and Verification. – `https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/hls/c-cplus/`, 2022.
2. Intel High Level Synthesis Compiler. – `https://www.intel.fr/content/www/fr/fr/software/programmable/quartus-prime/hls-compiler.html`, 2022.
3. Neon Overview. – `https://developer.arm.com/Architectures/Neon`, 2023.
4. Al-Qudsi (B.), Joram (N.), Strobel (A.) et Ellinger (F.). – Zoom fft for precise spectrum calculation in fmcw radar using fpga. – In *Proceedings of the Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, pp. 337–340, 2013.
5. Almorin (H.) et Le Gal (B.). – Generic FMCW Model Git repository. – `https://github.com/Bisuketto/FMCW_Model`, 2023.
6. Almorin (H.), Le Gal (B.), Crenne (J.), Jego (C.) et Kissel (V.). – High-throughput FFT archi-

tectures using HLS tools. – In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2022.

7. Andrade (J.), George (N.), Karras (K.), Novo (D.), Pratas (F.), Sousa (L.), Ienne (P.), Falcao (G.) et Silva (V.). – Design space exploration of ldpc decoders using high-level synthesis. *IEEE Access*, vol. 5, 2017, pp. 14600–14615.
8. Bordat (A.), Dobias (P.), Kernec (J. L.), Guyard (D.) et Romain (O.). – Gpu based implementation for the pre-processing of radar-based human activity recognition. – In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pp. 593–598, 2022.
9. Cooley (J. W.) et Tukey (J. W.). – An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.*, vol. 19, 1965.
10. Coussy (P.), Chavet (C.), Bomel (P.), Heller (D.), Senn (E.) et Martin (E.). – *GAUT : A High-Level Synthesis Tool for DSP Applications*, pp. 147–169. – Dordrecht, Springer Netherlands, 2008.
11. Delomier (Y.), Le Gal (B.), Crenne (J.) et Jego (C.). – Model-based design of efficient ldpc decoder architectures. – In *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, pp. 1–5, 2018.
12. Fialka (O.) et Cadik (M.). – FFT and convolution performance in image filtering on gpu. – In *Proceedings of the International Conference on Information Visualisation*, 2006.
13. Frigo (M.) et Johnson (S.). – The Design and Implementation of FFTW3. *Proceedings of the IEEE*, vol. 93, n2, 2005.
14. Guo (M.), Zhao (D.), Wu (Q.), Wu (J.), Li (D.) et Zhang (P.). – An integrated real-time fmcw radar baseband processor in 40-nm cmos. *IEEE Access*, vol. 11, 2023, pp. 36041–36051.
15. Intel. – Intel Advanced Vector Extensions 512. – `https://www.intel.fr/content/www/fr/fr/architecture-and-technology/avx-512-overview.html`, 2023.
16. Joram (N.), Al-Qudsi (B.), Wagner (J.), Strobel (A.) et Ellinger (F.). – Design of a multi-band fmcw radar module. – In *2013 10th Workshop on Positioning, Navigation and Communication (WPNC)*, pp. 1–6, 2013.
17. Köpcke (B.), Steuwer (M.) et Gorlatch (S.). – Generating efficient fft gpu code with lift. – In *Proceedings of the 8th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing (FHPNC 2019)*, p. 113, 2019.
18. Lestari (A.), Patriadi (D. D.), Putri (I. H.), Harnawan (B.), Winarko (O. D.), Sediono (W.) et Titasari (M. A. K.). – Fpga-based sdr implementation for fmcw maritime surveillance radar. – In *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, pp. 15–20, 2017.
19. Li (X.) et Liu (G.). – Design and implementation of 77g radar system. – In *Proceedings of the International Conference on Electronic Information and Communication Technology (ICEICT)*, 2022.
20. Microchip. – SmartHLS Compiler Software. – `https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/smarthls-compiler`, 2022.
21. Milló n (R.), Frati (E.) et Rucci (E.). – A comparative study between HLS and HDL on SoC for image processing applications. *Elektron*, vol. 4, n2, dec 2020, pp. 100–106.
22. Mitomo (T.), Ono (N.), Hoshino (H.), Yoshihara (Y.), Watanabe (O.) et Seto (I.). – A 77 ghz 90 nm cmos transceiver for fmcw radar applications. *IEEE Journal of Solid-State Circuits*, 2010.
23. Nvidia. – cuFFT Library User's Guide. – `https://docs.nvidia.com/pdf/CUFFT_Library.pdf`, 2020.
24. Otten (M.), Vlothuizen (W.), Spreeuw (H.) et Varbanescu (A.). – Real-time processing of multi-channel sar data with gpus. – In *Proc. of the European Radar Conference (EuRAD)*, 2016.

25. Perdana (R. S.), Sitohang (B.) et Suksmono (A. B.). – Radar signal processing in parallel on gpu : Case study dual polarization fmcw weather radar. – In *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 657–661, 2019.
26. Pommier (J.). – Pffft, a pretty fast fourier transform. – `https://bitbucket.org/jpommier/pffft/src/master/`, novembre 2011.
27. Radecki (K.), Samczyski (P.), Kulpa (K.) et Drozdowicz (J.). – A real-time unfocused sar processor based on a portable cuda gpu. – In *Proceedings of the European Radar Conference (EuRAD)*, 2015.
28. Shehata (M. G.), Ahmed (F. M.), Salem (S.) et Zakaria (H.). – Design and implementation of lfmcw radar signal processor for slowly moving target detection using fpga. – In *2020 12th International Conference on Electrical Engineering (ICEENG)*, pp. 241–248, 2020.
29. Xilinx. – Vitis High-Level Synthesis User Guide. – `https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction`, décembre 2022.
30. Xu (G.), Low (T. M.), Hoe (J. C.) et Franchetti (F.). – Optimizing fft resource efficiency on fpga using high-level synthesis. – In *Proceedings of HPEC*, 2017.
31. Zhijun (Y.), Xiangfei (N.), Wenyi (X.), Xiaowei (N.) et Weiming (T.). – Real time imaging processing of ground-based sar based on multicore dsp. – In *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 1–5, 2017.

Annexes

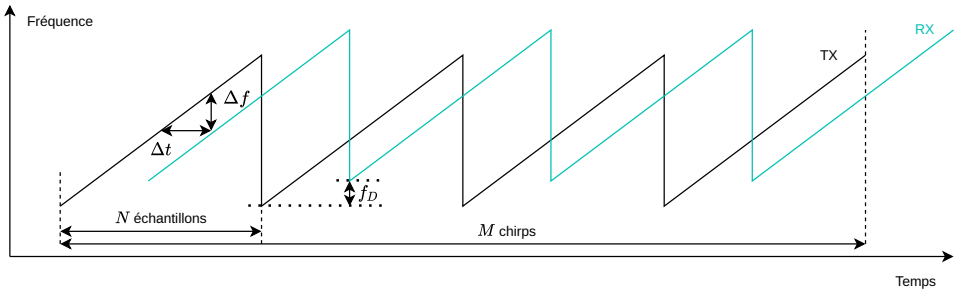

FIGURE 1 – Représentation des formes d'ondes en dents de scie émises et reçues dans les systèmes radars de type FMCW [1]

1. Les décalages temporels et fréquentiels sont volontairement exagérés
2. Puissances mesurées sauf sur AMD Xilinx où il s'agit d'une estimation

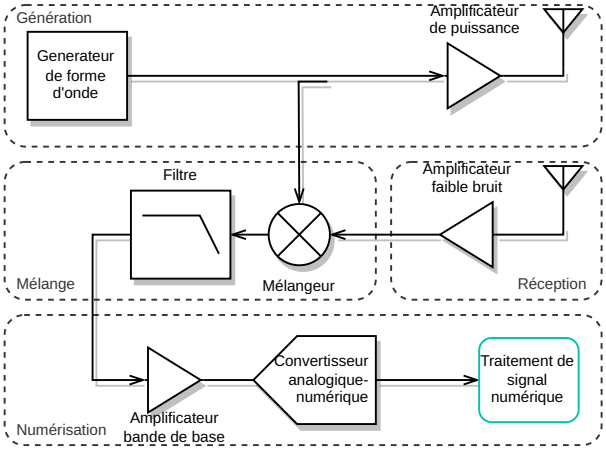

FIGURE 2 – Système radar FMCW de base

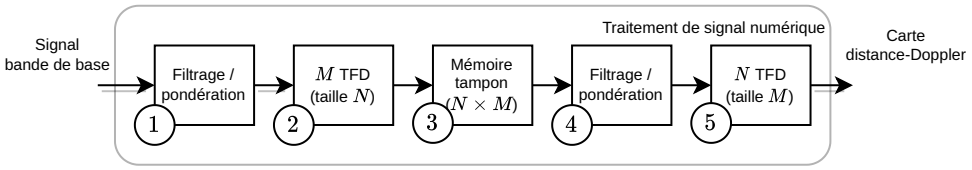

FIGURE 3 – Chaîne de traitement de signal numérique adaptée aux radars FMCW

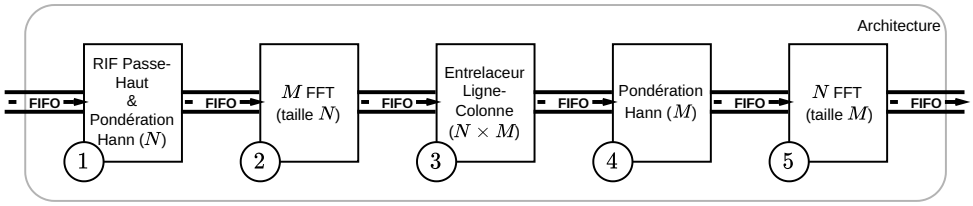

FIGURE 4 – Découpe d'architecture pour un traitement FMCW monocanal.

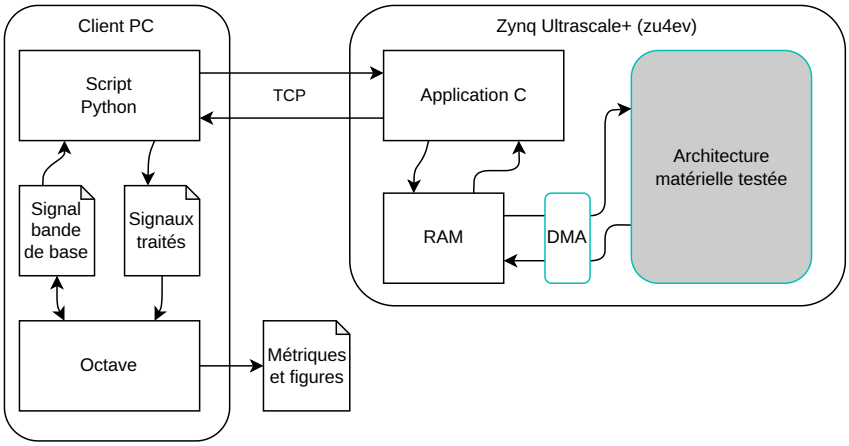

FIGURE 5 – Environnement de test pour les architectures matérielles

TABLE 1 – Comparaison des caractéristiques des implémentations logicielles et matérielles d'une chaîne de traitement radar lorsque $N = 2048$, $M = 16$ et $P \in \{1, 5\}$.

Plateforme	Implémentation	Canaux traités	Puissance [2] (Watts)	Largeur des données traitables	Latence (µs)	Débit (MSps)
Nvidia Jetson AGX Xavier	pffft	1 5	2 12	128b 128b	330 366	99 447
Intel i7-11800H	fftw3	1 5	16.5 46.5	512b 512b	121 175	271 934
Zynq UltraScale+ ARM A53	pffft pffft	1 5	≈ 0.5 –	128b 128b	2608 –	12 –
Zynq UltraScale+ ZU4EV @100MHz	Arch 1 HP Arch 4 HP	1 5	< 1 < 1	N/A N/A	604 1932	84.2 80
Zynq UltraScale+ ZU4EV @150MHz	Arch 1 HP Arch 4 HP	1 5	< 1 < 1	N/A N/A	413 1288	120 120

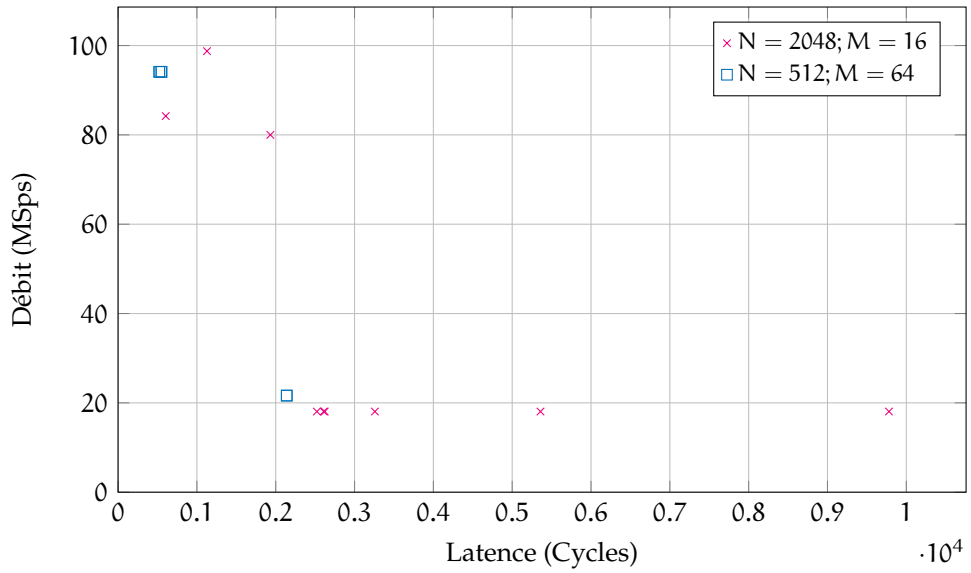

FIGURE 6 – Compromis débit-latence de l'espace des solutions architecturales exploré @100MHz

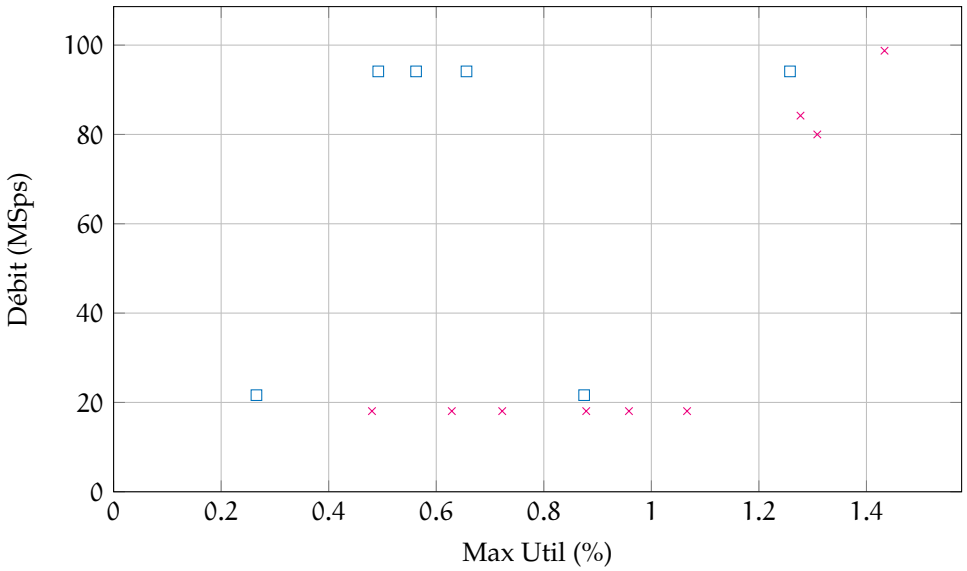

FIGURE 7 – Compromis débit-utilisation matérielle de l'espace des solutions arctitecturales exploré @100MHz

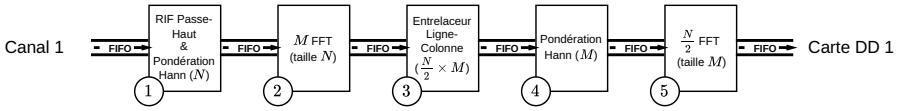

FIGURE 8 – Architecture 1 - Système de traitement mono-canal avec parallélisme de tâches

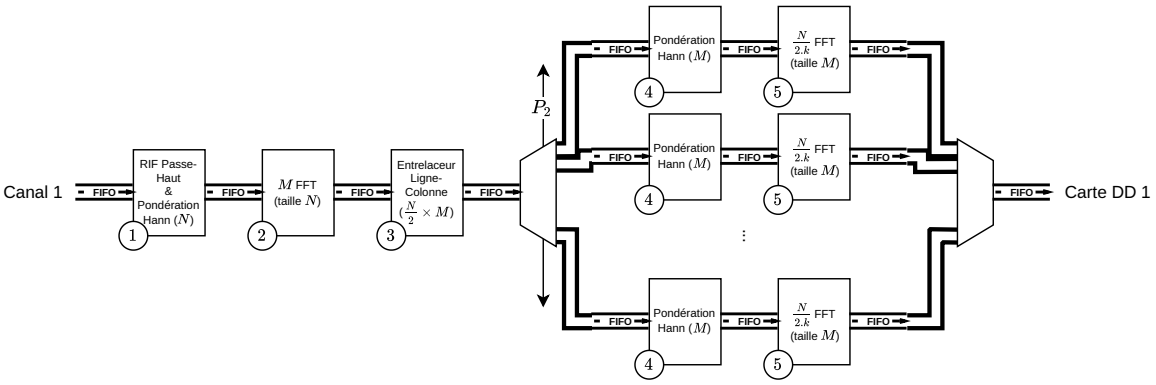

FIGURE 9 – Architecture 2 - Système de traitement mono-canal avec parallélisme de tâches et duplication des accélérateurs

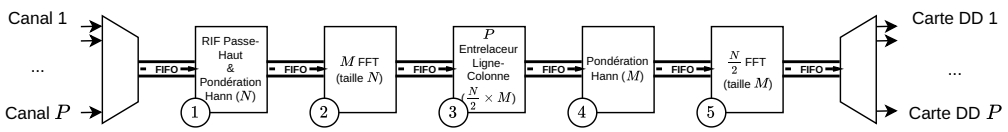

FIGURE 10 – Architecture 3 - Système de traitement multi-canaux avec réutilisation temporelle des accélérateurs matériels et parallélisme de tâches

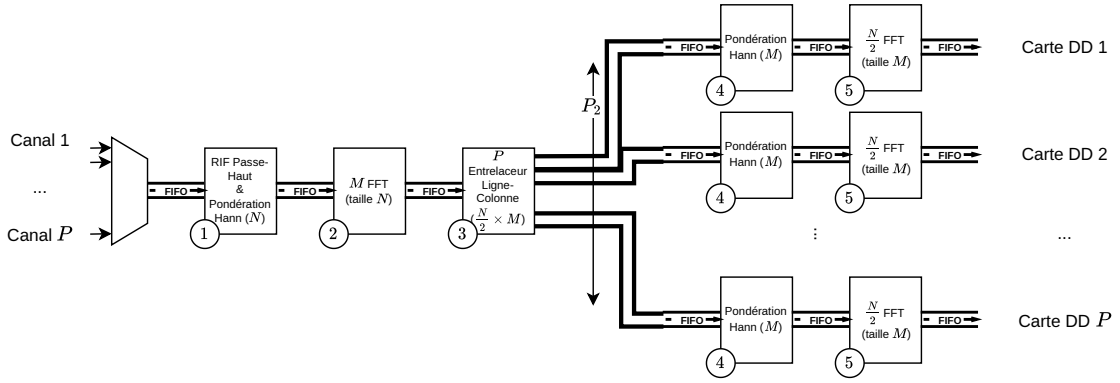

FIGURE 11 – Architecture 4 - Système de traitement multi-canaux avec parallélisme de tâches, réutilisation temporelle et duplication des accélérateurs

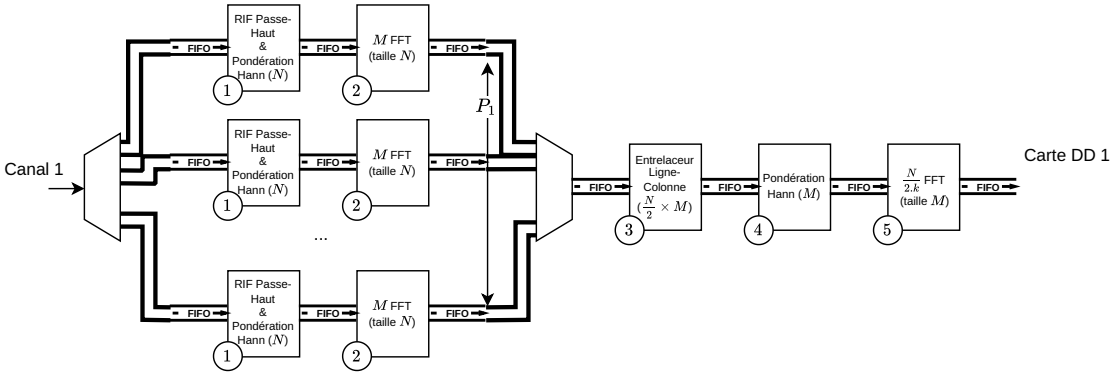

FIGURE 12 – Architecture 5 - Système de traitement multi-canaux avec parallélisme de tâches, réutilisation temporelle et duplication des accélérateurs