Implémentations logicielles et matérielles efficientes d'une chaîne de communications QCSP

Camille Monière, Bertrand Le Gal, Emmanuel Boutillon

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

Implémentations logicielles et matérielles efficientes d'une chaîne de communications QCSP *

Camille MONIÈRE, Bertrand LE GAL, Emmanuel BOUTILLON

Lab-STICC, CNRS UMR 6285
Université Bretagne Sud
56100 Lorient - France
nom.prénom@univ-ubs.fr

IMS, CNRS UMR 5218,
Bordeaux-INP,
33400 Talence - France
nom.prénom@ims-bordeaux.fr

Résumé
Dans les communications sans-fil classiques, des préambules sont utilisés pour aider le récepteur à détecter et synchroniser les trames. Cependant, ceux-ci entraînent une surconsommation de bande passante et de ressources non négligeables pour les paquets courts. Récemment, un nouveau type de trame sans préambule appelé Quasi Cyclic Small Packet (QCSP) a été proposé. Les travaux actuels étudient la mise en œuvre de la chaîne de communication associée. Dans cet article, seule le processus de complexité calculatoire la plus élevée, la détection coté récepteur, est détaillé. Différents niveaux de parallélisme et stratégies d'implémentation sont détaillés en logiciel (CPU) et matériel (FPGA). Un comparatif des performances atteignables est réalisé sur des cibles multicœurs et FPGA.

Mots-clés : Multicœurs, FPGA, Synthèse d'architectures, HLS, CCSK, Paquets courts

1 Introduction

La transmission de quantités restreintes de données au sein d'un réseau de communication non supervisé est un véritable défi, particulièrement pour les dispositifs composants l'internet des objets (IoT). En effet, ceux-ci communiquent classiquement grâce à des trames composées d'une charge utile précédée par un préambule standardisé utilisé à des fins de détection et de synchronisation coté récepteur [8, 16, 19]. Malheureusement, pour les paquets courts, le ratio taille de charge utile par taille du préambule tend à l'équilibre [7]. Les préambules permettent certes une simplification des récepteurs, mais l'énergie qu'ils nécessitent leur est exclusive et est par conséquent perdue pour le reste de la communication [15]. Des approches sans préambule existent [3, 4, 20]. Cependant, elles n'ont prouvés leur efficacité que dans des environnements peu bruités. Une forme d'onde sans préambule, Quasi Cyclic Small Packet (QCSP) [14,17] a été récemment proposée. Elle est efficace même dans des environnements très bruités (SNRs ≈ -10 dB) grâce à une modulation Cyclic Code Shift Keying (CCSK) couplée avec un codage canal de type non binaire. En plus de ces excellentes performances, la complexité calculatoire du côté de l'émetteur est faible faisant de cette forme d'onde une solution adaptée au contexte de l'IoT dont les dispositifs IoT fonctionnent souvent sur batterie. Toutefois, cette faible complexité coté émetteur est obtenue aux dépens de la complexité du récepteur pour lequel les étapes de détection et de synchronisation sont plus complexes à cause de l'absence de préambule.
Dans [14], le principe de fenêtres glissantes (*time sliding windows*) pour la détection de trames QCSP est introduit et comparé avec la méthode classique par FFT. Les premiers résultats de

*. Travaux financés par l'ANR, projet ANR-19-CE25-0013-01.

l'implémentation logicielle sont également donnés pour montrer la pertinence de la méthode. Cet article résume et traduit le prolongement de ce travail reporté dans [13], en se concentrant sur l'étude approfondie des implémentations en temps réel possibles du détecteur QCSP. La section 2 présente d'abord le modèle du système. La section 3 compare les variantes algorithmiques actuelles tandis que les approches de parallélisation sont détaillées dans la section 4. Les résultats d'implémentation du détecteur sur une cible CPU et une cible FPGA sont rapportés dans la section 5. Enfin, les conclusions et les perspectives sont présentées dans la section 6.

2 Modélisation du Système

Cette section passe en revue le modèle et la méthode de détection présentés dans [14]. Le système de communication est représenté sur la Fig. 1. Le message envoyé par l'émetteur (K symboles de p bits, chacun appartenant à $[0, 1, \ldots, q-1]$, $q = 2^p$) est d'abord codé à l'aide d'un encodeur Low Density Parity Check Non-Binaire (LDPC-NB) en un mot de code $\mathbf{C}$ (N symboles de p bits). La CCSK est ensuite appliquée aux symboles, les transformant en décalages circulaires d'une séquence de bruit binaire pseudo-aléatoire $\mathbf{P}_0$ (le k^e décalage est noté $\mathbf{P}_k$) [6]. La trame CCSK résultante est modulée BPSK pour devenir la trame QCSP $\mathbf{F}$ [17], qui passe dans un filtre en cosinus surélevé, avant d'être envoyée à un dispositif RF pour transmission.
Coté récepteur, les données reçus de la partie analogiques sont sur-échantillonnées d'un facteur $\mathcal{O}$ et filtrés par le même filtre. Ensuite, $\mathcal{O}$ détecteurs fonctionnent en parallèle pour permettre une future décimation. Chacun d'eux calcule un score grâce à la CCSK, comme décrit dans [18] et résumé dans la sous-section suivante. Lors d'une détection, les données associées à l'hypothèse de sur-échantillonnage ayant obtenu le score le plus élevé sont synchronisées, éliminant toutes imprécisions de fréquence, de temps ou de phase. Des rapports de vraisemblance logarithmique (LLR) sont produits à partir de la trame synchronisée résultante pendant la démodulation CCSK [2]. Enfin, ces LLRs sont transmis à un décodeur LDPC-NB qui corrige et décode le message reçu.
Cet article se concentre sur l'implémentation de l'étape de détection de trames sous contrainte temps réel (en rouge sur la Fig. 1).
La détection basée sur la CCSK consiste principalement à comparer une fonction de score à un seuil U_0. Le score est calculé à partir des $N \times q$ derniers échantillons reçus (soit la longueur d'une trame) au temps n, divisés de manière égale en N sous-vecteurs $\mathbf{Y}_n$ (vecteur de $y(n-i)$ pour $i \in [0, 1, \ldots, q-1]$) de q échantillons. Si la valeur du score dépasse le seuil U_0, on considère qu'une nouvelle trame est arrivée.
La fonction de score S_n^ω correspond à la sortie d'un filtre, cette sortie étant maximale pour une trame arrivée au temps n avec un décalage de fréquence $f = \frac{\omega}{2\pi q}$. La première étape consiste à atténuer le décalage de fréquence en multipliant terme à terme (opérateur $\odot$) $\mathbf{Y}_n$ avec le vecteur de rotation $\Gamma^\omega = \{1, e^{-j\frac{\omega}{q}}, e^{-j\frac{2\omega}{q}}, \ldots, e^{-j\frac{(q-1)\omega}{q}}\}$ (c.-à-d. une sinusoïde complexe pure de fréquence $-f$). Un vecteur de corrélation $\mathbf{L}_n^\omega = (\mathbf{Y}_n \odot \Gamma^\omega) \star \mathbf{P}_0$ est alors calculé. Ce vecteur est similaire à une tentative de démodulation CCSK. Le maximum absolu normalisé de $\mathbf{L}_n^\omega$ (désignée par M_n^ω) est pris comme indicateur du succès de la démodulation. S_n^ω résulte de l'accumulation de M_n^ω sur la durée d'une trame. Comme le score résultant est comparé à un seuil U_0, et à condition d'adapter le seuil U, les racines carrées impliquées dans le calcul des valeurs absolues [14] peuvent être supprimées, simplifiant ainsi la fonction de score. Par conséquent, M_n^ω devient :

$$M_n^\omega = \frac{\max\{|L_n^\omega(i)|^2, i = 0, 1, \ldots q-1\}}{\sum_{i=0}^{q-1} |y(n-i)|^2}, \tag{1}$$

pour une fonction de score résultante définie par :

$$S_n^\omega = S_{n-q}^\omega + M_n^\omega - M_{n-Nq}^\omega. \quad (2)$$

S_n^ω étant le score associé aux $N \times q$ derniers échantillons, il doit être calculé au moins tous les q échantillons reçus, mais peut être calculé pour chaque échantillon. Cela conduit à l'introduction du paramètre p_Δ, puissance de 2 indiquant le nombre de scores calculés tous les q échantillons pour une rotation ω (donc $p_\Delta = 1, 2, 4, \ldots, q$). Il convient de noter que cela réduit également l'utilisation de la mémoire, puisque seules $N \times p_\Delta$ valeurs de M_n^ω sont nécessaires. Enfin, pour garantir la tolérance aux erreurs de fréquence, plusieurs scores S_n^ω pour différentes valeurs de ω sont calculés en parallèle. Le nombre d'hypothèses de rotation testées en parallèle de $-\pi$ à π est noté p_ω, allant de 1 à théoriquement tout entier naturel, mais limité à 8 en pratique, le gain de performance étant minime au regard du coût résultant de valeurs plus élevées [14]. En effet, pour $p_\omega = 8$, $p_\Delta = q = 64$ et un sur-échantillonnage $\mathcal{O} = 8$, il est déjà nécessaire de réaliser 64 millions de corrélations de taille $q = 64$ par seconde pour une fréquence d'échantillonnage de 8 Ms/s, soit pour un flux entrant de données de $\simeq 15$ Mio/s pour des données sur 16 bits.
L'architecture globale qui permet de calculer un score S_n^ω est donnée dans la Fig. 2, et est appelée Unité de Calcul de Score (UCS). Ces architectures sont présentées dans la section suivante.

3 Unité de Calcul de Score (UCS)

La méthode originale de calcul de corrélation [18] ($\star\mathbf{P}_0$ dans la figure 2) est représentée dans la figure 3. Cette méthode, certes éprouvée, n'est pas adaptée au traitement continu des échantillons, en raison de sa complexité calculatoire. En effet, maximiser les performances revient à traiter chaque échantillon (c.-à-d. $p_\Delta = q$), les FFTs et IFFTs de taille q devant être recalculées. Diminuer p_Δ réduit la complexité de calcul, aux dépens des performances de détection. Dans ce cas, une nouvelle corrélation est produite toutes les $\frac{q}{p_\Delta}$ données reçues.
Une manière efficace pour calculer autrement ces corrélations est présentée dans [14]. Cette approche calcule la corrélation dans le domaine temporel, en utilisant :

$$L_n^\omega(k) = L_{n-1}^\omega(k-1) + p_k d_n^\omega, \quad (3)$$

avec $d_n^\omega = (y(n) - y(n-q)e^{j\frac{\omega}{q}})e^{-jn\frac{\omega}{q}}$ et $p_k = P_k(q-1)$, pour $k = 0, 1, \ldots, q-1$. Cette nouvelle méthode, schématisée dans la figure 4, utilise le résultat de la corrélation précédente pour calculer la suivante, créant une fenêtre d'accumulation glissante (*time sliding window*).
La complexité calculatoire de chacune des méthodes est indiquée dans le tab. 1 en termes d'arithmétique/stockage flottants. Étant chacune directement proportionnelles à p_ω, ce paramètre n'est pas pris en compte dans le tab. 1.
Différentes optimisations peuvent être appliquées afin d'améliorer les propriétés de ces méthodes. En effet, la mémoire des *packers* des UCS fréquentielles peut être distribuée ou partagée. La première UCS stockant déjà les q derniers échantillons, la deuxième n'a besoin que de $\frac{q}{2}$ échantillons supplémentaires, les 2 suivantes que de $\frac{q}{4}$ en plus, etc. Pour la méthode temporelle, la dernière multiplication peut être remplacée par une simple négation conditionnelle, $\mathbf{P}_0$ étant composé de $\{-1 + 1\}$. Cela réduit d'un facteur q le nombre de multiplications complexes à réaliser (cette méthode est signalée comme Time Sliding* dans le tab. 1).
Selon les valeurs indiquées dans le tab. 1, lorsque $p_\Delta = q$ (meilleures performances de détection), la variante temporelle présente les complexités les plus faibles. Cependant, avec la variante fréquentielle, p_Δ peut être réduit, entraînant une réduction de complexité rentable au regard de l'impact sur les performances de détection. L'effet de la valeur p_Δ est détaillée dans [14]. Ainsi, il n'est pas possible, à ce stade, de déterminer quelle méthode est la plus avantageuse d'autant plus que cela peut varier en fonction de la cible CPU / FPGA.

4 Stratégies de Parallélisation Appliquées à la Détection

La tâche de détection est la plus exigeante du récepteur. En effet, tous les échantillons reçus doivent être traités en continus. Pour atteindre des performances temps-réel pour une fréquence d'échantillonnage de quelques millions d'échantillons par seconde, l'étape de détection doit être parallélisée afin de bénéficier des propriétés des processeurs multicœurs ou des FPGAs. L'augmentation des performances de traitement des dispositifs multicœurs, associée à des modèles de programmation clef en main [5,12], a rendu possible la mise en œuvre de prototypes ou de systèmes de communication réels en logiciel. Une implémentation logicielle peut ne pas atteindre le débit et l'efficacité énergétique des solutions ASIC/FPGA, mais elle offre une flexibilité, une évolutivité et un temps de prototypage inégalés. Néanmoins, l'obtention de performances élevées est un défi et nécessite des efforts de parallélisation des algorithmes.
Plusieurs niveaux de parallélisme ont été identifiés. Les parties suivantes présentent les niveaux de parallélisme inhérents au système, puis détaille les stratégies utilisées pour les cibles logicielles et matérielles.

4.1 Parallélisme inhérent au système

Trois niveaux de parallélisme gros grains existent dans l'algorithme de détection. À partir de l'observation des figures 3 et 4, deux niveaux de parallélisme sont intuitifs : le parallélisme Δ lié à p_Δ, et le parallélisme ω lié à p_ω. Cependant, les valeurs de p_Δ $(1, 2, 4, \ \dots \ , q)$ et de p_ω $(1, 2, \ \dots \ , 8)$ impactent directement sur les performances de détection et sur la complexité du système. Dans tous les cas, les calculs effectués dans les $p_\Delta \times p_\omega$ branches sont indépendants, ce qui permet une évaluation totalement concurrente. Un troisième parallélisme à gros grain résulte du sur-échantillonnage du signal. Il est lié au facteur de sur-échantillonnage $\mathcal{O}$. Il conduit à l'exécution possible de $\mathcal{O}$ détecteurs en parallèle. Ces trois niveaux de parallélisme permettent de multiples configurations. En effet, la tâche de détection globale peut être considérée comme une tâche unique ou peut être divisée en $\mathcal{O} \times p_\Delta \times p_\omega$ sous-tâches parallèles.

4.2 Parallélisation logicielle et matérielle

Les architectures multicœurs de type INTEL x86 actuellement deux approches de parallélisation distinctes, à savoir le multithreading (MT) et le SIMD (Single Instruction Multiple Data).
Tout d'abord, pour tirer parti du MT, l'application doit être découpée en sous-tâches indépendantes pouvant être exécutées en parallèle. Cette approche est efficace pour le parallélisme à gros grain. C'est le cas pour le parallélisme $\mathcal{O}$, pour Δ dans les UCS fréquentielles et semble l'être pour le parallélisme ω. Cependant, comme souligné plus tard, des sous-tâches peu complexes bénéficient moins de ce type de parallélisation (en raison des temps de démarrage/jonction des threads). L'exploitation de la vectorisation SIMD nécessite des calculs arithmétiques identiques appliqués à plusieurs données simultanément. Il s'applique naturellement au parallélisme Δ des UCS temporelles. En effet, chaque branche de corrélation peut être considérée comme un calcul parallèle. Les UCS fréquentielles peuvent également bénéficier automatiquement de la vectorisation SIMD grâce à la bibliothèque FFTW3 [10].
Un FPGA offre plus d'opportunités de parallélisation que les multicœurs. En effet, les éléments de traitement sont conçus spécifiquement pour la tâche. Cela permet de gérer le parallélisme de bas niveau en utilisant des architectures séquentielles, semi-parallèles ou entièrement parallèles. Ensuite, à un niveau plus élevé, un ensemble hétérogène d'opérateurs personnalisés peut être alloué pour gérer des sous-tâches concurrentes. Cependant, un parallélisme massif par duplication des opérateurs est fortement limité par les ressources disponibles au sein du FPGA. Différents compromis sur les architectures des UCS ont été conçus. Dans toutes ces solutions, $\mathcal{O} \times p_\omega$ sous-systèmes parallèles sont alloués pour des raisons de débit, comme indiqué dans

la section suivante.

5 Résultats d'Implémentation

Le projet de recherche dans lequel s'inscrivent ces travaux vise à implémenter un système complet de communication QCSP. Cet article se concentre sur la première étape de la réception, la tâche de détection. Les démonstrateurs résultants doivent valider la forme d'onde QCSP et produire des estimations de la complexité d'implémentation.
Le système de réception a été implémenté sur cibles CPU et FPGA. Deux cibles distinctes ont permis d'évaluer les performances en débit et en latence du récepteur : un CPU Intel Xeon et un FPGA Xilinx Kintex 7 (xc7k410tffg900-1), soit celui présent dans l'USRP X310 d'Ettus, utilisé comme station SDR. Dans les deux cas, le $//_{\mathcal{O}}$ est traité en utilisant $\mathcal{O}$ détecteurs en parallèle, aussi les résultats rapportés correspondent à un unique détecteur.

5.1 Détecteur logiciel

Le système de communication présenté dans la figure 1 a été décrit en langage C++14. Chaque variante de UCS présentée (fréquentielle et temporelle) a été implémentée pour plusieurs valeurs de p_Δ et p_ω, et en deux configurations pour la temporelle. La machine utilisée pour les tests comporte un processeur Intel Xeon Gold 6148 double socket. Chaque processeur Xeon est composé de 20 cœurs physiques qui partagent une mémoire cache L3 de 28160 Ko. La fréquence de travail normale des processeurs est de $2,60$ GHz, mais peut atteindre $3,70$ GHz grâce à la fonction turbo-boost tant que la contrainte de dissipation thermique est respectée. Dans cette description du système, les données manipulées sont en flottant 32 bits pour éviter les problèmes de précision liés à l'arithmétique en virgule fixe.
La stratégie de parallélisation décrite dans la section précédente a été appliquée aux codes sources C++. Les UCS fréquentielles utilisent la bibliothèque optimisée FFTW3 [10]. Cette bibliothèque exploite en interne toutes les propriétés du CPU (MT, SIMD, et autres). $//_\Delta$ est implémenté en utilisant l'API OpenMP, avec l'approche par FFT à mémoire partagée décrite dans la section 3. OpenMP a également été envisagée pour $//_\omega$, mais malgré tous nos efforts, cela n'a jamais abouti à une accélération.
La variante temporelle est implémentée uniquement par du code C/C++ dédié. Ce code a été décrit afin de tirer profit de la fonction d'auto-vectorisation de GCC. Toutefois, certaines parties spécifiques du code ont été finement ajustées avec des instructions SIMD intrinsèques.
Les débits mesurés pour les différentes configurations sont indiqués dans le tab. 2. Le débit et la latence ont été mesurés via l'API C++14 Chrono. Dans le tab. 2, la FFT MT utilise p_Δ threads. Les performances en termes de débit sont impactées par la méthode de corrélation appliquée et les paramètres algorithmiques sélectionnés.
Le détecteur FFT (monothread) est le plus lent, atteignant au maximum $2,2$ MChips/s, chutant jusqu'à $0,64$ MChips/s pour $p_\Delta = 16$ et $p_\omega = 8$. En revanche, le débit monte jusqu'à $5,1$ MChips/s en utilisant p_Δ cœurs de processeur physique. Il est possible de traiter jusqu'à 160 kbits/s d'informations, comme le montre le tab. 2 (MT FFT). Le MT accélère donc $2,3\times$ les débits, loin derrière le facteur p_Δ. Cet écart de performance est dû à la surcharge induite par le démarrage et la synchronisation des threads OpenMP.
L'approche temporelle, qui fournit les meilleures performances de détection grâce à $p_\Delta = q$, atteint un débit de détection de $1,4$ MChips/s sans optimisation. Son débit monte jusqu'à $3,3$ MChips/s lorsqu'elle est vectorisée, soit presque le débit de la FFT MT malgré l'utilisation d'un seul cœur de processeur, soit au moins 16 fois moins de ressources CPU.
La méthode de détection basée sur la FFT MT offre le débit le plus élevé (5,1 MChips/s) lorsqu'un grand nombre de cœurs est disponible. Cependant, le couple (p_Δ, p_ω) a un impact sur

les performances de détection. Par conséquent, l'approche temporelle offrant de meilleures performances de détection pour un débit élevé est la meilleure solution. Ces débits de réception, qui atteignent quelques MChips/s, sont déjà similaires à ceux requis, par exemple, dans le domaine des réseaux sans fil à faible débit (LRWN) [1]. Cela fait du récepteur logiciel une solution viable pour l'évaluation de la modulation QCSP.

5.2 Détecteur matériel

Les codes sources du détecteur C++ ont été réécrits en langage C à l'aide du sous-ensemble de synthèse C autorisé [9] pris en charge par Xilinx Vitis HLS 2020 [21]. Les modèles C, pour les deux approches de détection, sont configurables à l'aide des mots-clés *#define* et *#pragma*. Cela permet de tester diverses variantes architecturales en fonction de p_ω ou p_Δ. Ces modèles C sont synthétisés à l'aide de Vivado HLS pour le FPGA Xilinx Kintex 7 présent dans l'USRP X310.

Deux variantes principales ont été développées pour implémenter le détecteur à base de FFT. Elles visent à maximiser le débit en dépit des coûts, en instanciant p_ω sous-chaînes de traitement en parallèle. Elles exploitent aussi des pipelines de tâches et le traitement parallèle. Les 2 variantes se distinguent par la manière de représenter l'information. La première, en virgule flottante, demande trop de ressources pour la cible choisie. La seconde variante utilise donc des données en virgule fixe, avec une quantification conservatrice, pour ne pas impacter les performances de détection. Les modèles de FFT utilisés sont issus des travaux présentés dans [11]. Le détecteur fréquentiel flottant atteint $2,1$ MChips/s avant l'étape de placement / routage, soit le même débit que son pendant logiciel monocœur. Sa complexité matérielle élevée découle de la complexité de l'arithmétique flottante. La version du même décodeur en virgule atteint un débit de $2,9$ MChips/s, sans dépasser les ressources FPGA disponibles.

Le détecteur temporel a également été décrit en utilisant des approches de pipeline et d'exécution de tâches parallèles. Grâce à la simplification précédemment mentionnée, la complexité du $//_\Delta$ est nettement inférieure. Il en résulte un débit $3\times$ supérieur au plus élevé atteint par les décodeurs à base de FFT, malgré l'utilisation d'une arithmétique flottante. Son débit est $1,8\times$ supérieur à celui du meilleur détecteur logiciel, FFT MT pour $\mathbf{p_\Delta = 8}$, alors que le détecteur en temporel utilise $\mathbf{p_\Delta = 64}$, garantissant des performances de détection bien plus élevées. Malheureusement, pour $p_\omega = 8$, il demande trop de ressources. Pour résoudre ce problème et améliorer encore l'efficacité de l'architecture, une quantification en virgule fixe est prévue.

6 Conclusion

Cet article démontre que la tâche de détection QCSP peut être implémentée en temps réel aussi bien sur une cible multicœurs que sur FPGA. Cela permet la mise en œuvre de systèmes de communication fiables pour l'échange de paquets courts sans préambule à faible SNR. Les implémentations résultantes ont atteint des débits de quelques dizaines de kbits/s à quelques centaines de kbits/s, compatibles avec le standard LPWAN, sur des cibles multicœurs et FPGA. Elles sont particulièrement adaptées à un scénario de réseau de capteurs sans fil où des nœuds bas de gamme envoient des données à un concentrateur plus complexe. Dans tous les scénarios étudiés, l'approche à base de corrélation temporelle fournie les meilleures performances en termes de débit. Son implantation en virgule fixe sur FPGA ainsi qu'une étude de la consommation énergétique des différentes implantations font parties des travaux actuellement en cours de réalisation.

Bibliographie

1. IEEE Std 802.15.4-2020, IEEE Standard for Low-Rate Wireless Networks. 2020.
2. Abassi (O.), Conde-Canencia (L.), Mansour (M.) et Boutillon (E.). – Non-binary low-density parity-check coded cyclic code-shift keying. – In *Proceedings of WCNC*, Shanghai, China, April 2013.
3. Azari (A.) et al. – Grant-Free Radio Access for Short-Packet Communications over 5G Networks. – In *Proceedings of GLOBECOM*, 2017.
4. Bloessl (B.) et Dressler (F.). – mSync : Physical Layer Frame Synchronization without Preamble Symbols. *IEEE Trans. on Mobile Computing*, vol. 17, n10, 2018.
5. Checko (A.) et al. – Cloud RAN for mobile networks - a technology overview. *IEEE Communications Surveys & Tutorials*, vol. 17, n1, 2015.
6. Dillard (G.) et al. – Cyclic code shift keying : A low probability of intercept communication technique. *IEEE Trans. on Aerospace and Electronic Systems*, vol. 39, n3, 2003.
7. Durisi (G.) et al. – Toward Massive, Ultrareliable, and Low-Latency Wireless Communication With Short Packets. *Proceedings of the IEEE*, vol. 104, n9, 2016.
8. Ferre (G.) et Giremus (A.). – LoRa Physical Layer Principle and Performance Analysis. – In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 65–68, Bordeaux, décembre 2018. IEEE.
9. Fingeroff (M.). – *High-Level Synthesis Blue Book*. – Xlibris Corporation, 2010.
10. Frigo (M.) et Johnson (S.). – The Design and Implementation of FFTW3. *Proceedings of the IEEE*, vol. 93, n2, 2005.
11. Kastner (R.), Matai (J.) et Neuendorffer (S.). – Parallel Programming for FPGAs. *ArXiv e-prints*, 2018.
12. Mavromoustakis (C.), Mastorakis (G.) et Dobre (C.) (édité par). – *Advances in Mobile Cloud Computing and Big Data in the 5G Era*. – 2017, *Studies in Big Data*.
13. Monière (C.), Le Gal (B.) et Boutillon (E.). – Efficient software and hardware implementations of a QCSP communication system. – In *Proceedings of DASIP (To be published)*, 2022.
14. Monière (C.), Saied (K.), Le Gal (B.) et Boutillon (E.). – Time sliding window for the detection of CCSK frames. – In *Proceedings of SiPS*. IEEE, 2021.
15. Polyanskiy (Y.). – Asynchronous Communication : Exact Synchronization, Universality, and Dispersion. *IEEE Trans. on Information Theory*, vol. 59, n3, March 2013.
16. Raychowdhury (A.) et Pramanik (A.). – Survey on LoRa Technology : Solution for Internet of Things. – In Thampi (S. M.), Trajkovic (L.), Mitra (S.), Nagabhushan (P.), El-Alfy (E.-S. M.), Bojkovic (Z.) et Mishra (D.) (édité par), *Intelligent Systems, Technologies and Applications*, *Advances in Intelligent Systems and Computing*, Advances in Intelligent Systems and Computing, pp. 259–271, Singapore, 2020. Springer.
17. Saied (K.), Ghouwayel (A.) et Boutillon (E.). – Time-synchronization of CCSK short frames. – In *Proceedings of WiMob*, 2021.
18. Saied (K.), Ghouwayel (A. C. A.) et Boutillon (E.). – Quasi Cyclic Short Packet for asynchronous preamble-less transmission in very low SNRs. *Preprint HAL*, 2020.
19. Tapparel (J.), Afisiadis (O.), Mayoraz (P.), Balatsoukas-Stimming (A.) et Burg (A.). – An Open-Source LoRa Physical Layer Prototype on GNU Radio. – In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, mai 2020.
20. Walk (P.) et al. – MOCZ for Blind Short-Packet Communication : Practical Aspects. *IEEE Trans. on Wireless Communications*, vol. 19, n10, October 2020.
21. Xilinx. – *Vitis High-Level Synthesis User Guide UG1399 (v2021.1)*, June 2021.

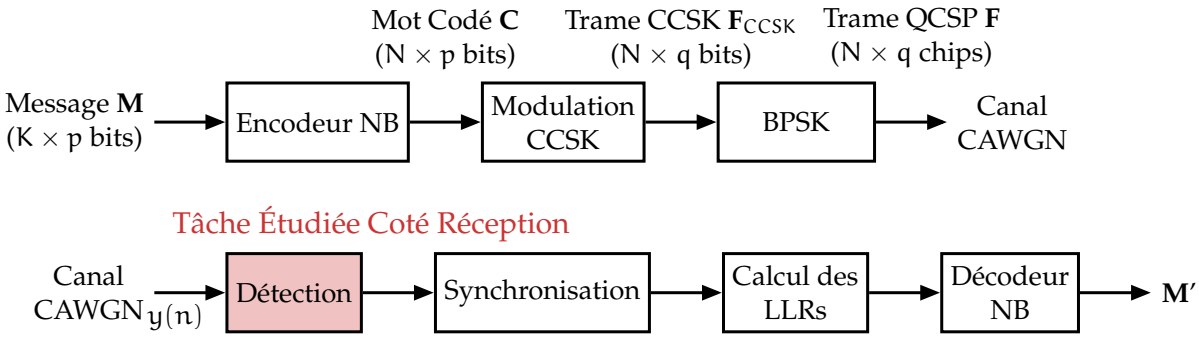

FIGURE 1 – Modèle d'une chaîne de communication QCSP

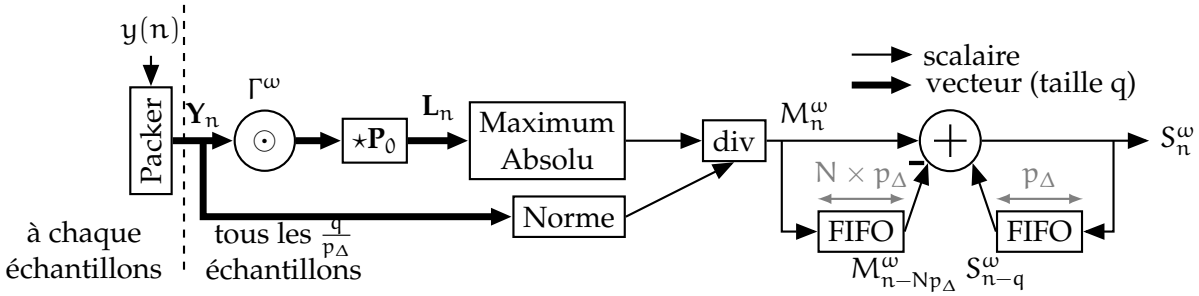

FIGURE 2 – Unité élémentaire de calcul de score pour une rotation ω donnée.

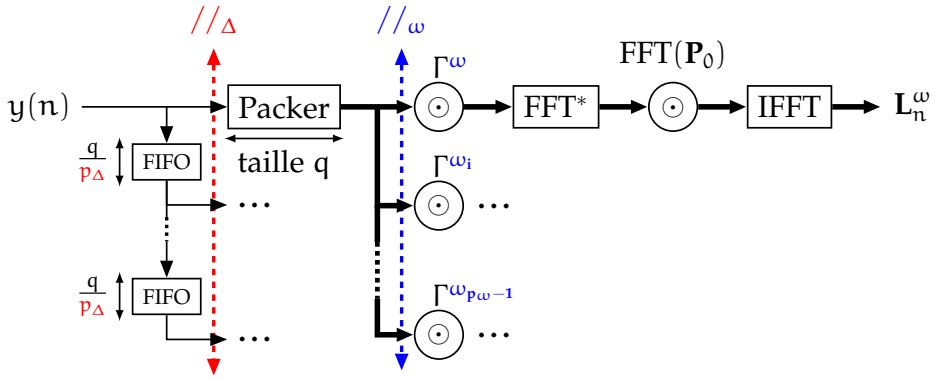

FIGURE 3 – Parallélisme Δ et ω au sein des UCS fréquentielles

.

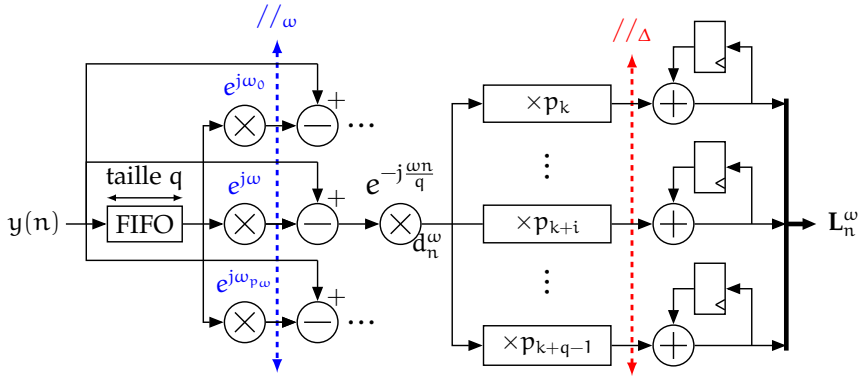

FIGURE 4 – Parallélisme Δ et ω au sein des UCS temporelles

.

TABLE 1 – Comparaison des complexités en fonction de p_Δ ($p_\omega = 1$)

Méthode	p_Δ	Add.	Mult.	Mémoire
FFT	$[1, q[$	$2p_\Delta q \log_2(q)$	$p_\Delta q(\log_2(q)+2)$	$(\frac{p_\Delta - 1}{p_\Delta} + p_\Delta)q$
(mémoire distr.)	q	$2q^2\log_2(q)$	$q^2(\log_2(q)+2)$	q^2+q+1
FFT	$[1, q[$	*idem*	*idem*	$\sum_{i=0}^{\log_2(p_\Delta)} \frac{q}{2^i}$
(mémoire part.)	q	*idem*	*idem*	$2q-1$
Time Sliding	q	$q(1+q)$	q^2	$2q$
Time Sliding*	q	$q(1+q)$	q	$2q$

TABLE 2 – Débit chip, bit et latence pour différents paramètres, pour $N = 60$, $q = 64$ et un rendement effectif $R_{eff} = \frac{1}{32}$.

Méthode	p_Δ	p_ω	Débit Chip (MChips/s)	Débit Bit (kbits/s)	Pire Latence (ms)	Meilleure Latence (ms)
FFT	8	4	2.2	69	3.5	1.7
		8	1.3	41	5.8	2.9
	16	4	1.1	34	6.8	3.4
		8	0.68	21	11	5.7
MT FFT	8	4	5.1	160	1.5	0.77
		8	4.1	130	1.9	0.93
	16	4	3.8	120	2	1
		8	3.1	97	2.5	1.2
Time Sliding	64	4	2.4	74	3.2	1.6
		8	1.4	43	5.4	2.7
Time Sliding (optimisé)	64	4	3.3	100	2.4	1.2
		8	2.0	63	3.7	1.9

TABLE 3 – Performances des implémentations matérielles de détecteur

Méthode	p_Δ	p_ω	Débit Chip (MChips/s)	Débit Bit (kbits/s)	Ressources FF	LUT	BRAM	DSP
FFT float	8	4	2,1	65	ressources insuffisantes			
FFT 16b fixe	8	4	2.6	81	25726	43206	950	380
		8	2,6	81	51452	94468	1900	760
	16	4	2.6	81	51452	94468	1900	760
		8	2,6	81	102904	188936	3800	1520
Time Sliding float	64	4	9,1	284	144064	80836	79	383
		8	9.1	284	ressources insuffisantes			