COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud

Thèse de doctorat de

L'UNIVERSITÉ DE BRETAGNE SUD

École Doctorale N° 644
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Électronique*

Par
Maël TOURRES

Enrichissement de l'ISA de cœurs de processeurs pour la couche physique des communications mobiles de 5ème génération

Thèse présentée et soutenue à Lorient, le 15/07/2024
Unité de recherche :
Laboratoire Lab-STICC UMR 6285 CNRS, Université de Bretagne Sud
Laboratoire IMS UMR 5218 CNRS, Université de Bordeaux

Rapporteurs avant soutenance :

Lilian BOSSUET	Professeur, Université de Lyon (Laboratoire Hubert Curien - CNRS)
Fakhreddine GHAFFARI	Maître de conférences, Université de CY Cergy Paris (Laboratoire ETIS -CNRS)

Composition du Jury :
Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Catherine DOUILLARD	Professeur (Lab-STICC, IMT Atlantique, Université de Bretagne-Sud, France)
Dir. de thèse :	Philippe COUSSY	Professeur des universités (Lab-STICC, Université de Bretagne-Sud, France)
Co-Enc. de thèse :	Bertrand LE GAL	Maître de conférences (IRISA,Inria Université de Rennes, France)
Co-Enc. de thèse :	Cyrille CHAVET	Maître de conférences (TIMA, Université de Grenoble Alpes, France)

Sommaire

Acronymes

ASIC Application-Specific Integrated Circuit.

ASIP Application-Specific Instruction Set Processor.

CCE Code Correcteur d'Erreurs.

DSP Digital Signal Processor.

FPGA Field-programmable gate array.

GCC GNU Cross-Compiler.

RISC Reduced instruction set computer.

SIMD Single Instruction Mulitple Data.

SISD Single Instruction Single Data.

SISO Soft Input Soft Output.

UAL Unité arithmétique et logique.

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Contexte de l'étude

Au cœur de nos sociétés modernes, les systèmes embarqués communicants sont devenus omniprésents et sont intégrés dans d'innombrables domaines d'application. La capacité à échanger rapidement, constamment, de manière sécurisée et fiable en tout lieu est un enjeu majeur ; c'est l'une des clés de voûte du développement de notre monde. Nous disposons d'un accès étendu à des millions d'appareils et de services interconnectés par le biais de technologies de communication dont l'évolution est exponentielle. La présence de ces systèmes de communication est telle qu'il semble désormais impossible pour nombre de personnes d'imaginer un quotidien sans leur utilisation, qu'il s'agisse de discussions privées via les réseaux sociaux ou encore de programmation à distance d'objets connectés divers au sein de nos habitations.

Cependant, ces derniers doivent aussi s'adapter à leur contexte d'utilisation de manière fiable et économique. Afin d'assurer que chaque système puisse s'intégrer rapidement et communiquer dans des réseaux déjà établis, il est nécessaire qu'ils se comprennent, ainsi l'utilisation de standards normalisés est une condition sine qua non. Pour ce faire, de nombreux standards de communication ont vu le jour. Si l'on se restreint à la téléphonie mobile, le standard 3G (UMTS) [1] déployé en France en 2004 a été remplacé progressivement au début des années 2010 par le standard 4G [2], puis étendu quelques années plus tard par le standard 4G LTE [3] Ces évolutions ont permis d'améliorer la fiabilité des liens et d'augmenter les débits de communication. Des gains significatifs ont depuis été obtenus grâce aux techniques employées dans le standard 5G [4] qui sera à son tour rendu obsolète dans quelques années par la future 6G [5].

Un des éléments clés, ayant permis ces évolutions successives, vient de l'étape de codage canal qui exploite la puissance des Codes Correcteurs d'Erreurs (CCE). Leur importance dans les systèmes de communications numériques est telle qu'ils ont été intégrés de manière systématique dans tous les standards de communication depuis plusieurs dé-

cennies. Fondamentaux, ils améliorent de manière notable la qualité de communication ; ils en assurent aussi la fiabilité et la sécurité en exploitant des algorithmes mathématiques poussés, basés sur les informations redondantes ajoutées dans les messages transmis. Cette redondance est ensuite exploitée par le récepteur, afin de corriger et restituer le message d'origine, même en présence de forts niveaux de bruit (*i.e.* perturbation/modification du message transmis). L'utilisation de codes correcteurs d'erreurs permet ainsi d'améliorer la robustesse du lien de communication, mais également de réduire les puissances d'émission. Cela induit toutefois une augmentation importante de la complexité calculatoire au niveau des récepteurs. La fonction de décodage est reconnue comme l'élément limitant des récepteurs, qu'ils soient matériels (ASIC/FPGA) ou logiciels (CPU/GPU). C'est pourquoi il est nécessaire pour les concepteurs d'innover et de trouver des solutions afin de concevoir des systèmes respectant les contraintes applicatives, tout en minimisant les coûts de conception, la consommation d'énergie et le temps de mise au point de tels décodeurs. Pendant très longtemps, seules les solutions matérielles étaient viables pour obtenir des performances temps réel, cependant le travail d'adaptation ou de re-conception des circuits pour chaque standard et/ou cadre applicatif induisait des temps et des coûts non-récurrents importants. L'accélération de la publication de nouveaux standards de communication couplée avec des besoins de flexibilité importants ont poussé les systèmiers à rechercher d'autres types de solutions d'implantation tels que les architectures programmables de type multi ou many-cœurs. La viabilité actuelle de ce type de solution est due à l'augmentation conséquente de la puissance de calcul induite par la multiplication des cœurs de processeurs dans un seul circuit ainsi qu'à l'introduction d'unités de calcul spécialisées (SIMD).

Ces avancées technologiques ont concouru à la mise en œuvre de récepteurs de type Software-Define Radio (SDR) et l'émergence du cloud-RAN qui permettent, en parallèle, d'exécuter de manière déportée des récepteurs hétérogènes dans des centres de calculs. La flexibilité de ces systèmes, reposant sur l'utilisation d'architectures programmables, est toutefois obtenue au prix d'une consommation énergétique au moins 100 fois supérieure à celle de solutions ASIC dédiées. Cet aspect peut s'avérer rédhibitoire alors que le déploiement de ces systèmes de réception s'accélère. Pour limiter la consommation énergétique de ces solutions logicielles, une piste consiste à augmenter l'adéquation entre les besoins applicatifs (*e.g.* les standards 4G ou 5G) et les capacités architecturales.

Organisation du manuscrit

L'objectif du chapitre 2 est de présenter le contexte de ces travaux de thèse. Les principes élémentaires des systèmes de communications numériques sont exposés, permettant de positionner le codage canal et les codes correcteurs d'erreurs au sein des chaînes de communications numériques. Ensuite, une présentation des principales familles de codes correcteurs d'erreurs (CCE) employées actuellement dans les standards de communication est fournie. Le cadre d'usage de ces familles de CCE, ainsi qu'une description synthétique de leurs algorithmes de décodage sont exposés.

Dans le chapitre 3, l'évolution dans le temps des approches d'implantation des décodeurs de codes CCE est présentée et mise en perspective. Cette partie permettra d'introduire les motivations qui ont amené à la réalisation de ces travaux de thèse, dont l'objectif consiste à proposer des extensions aux jeux d'instructions des architectures programmables, pour mieux supporter l'exécution de décodeurs CCE.

Dans le chapitre suivant, l'approche méthodologique proposée afin d'identifier les motifs d'instructions récurrents est présentée. Les contraintes de l'étude liées au ciblage d'architectures de processeurs classiques manipulant des instructions possédant 2 opérandes en entrée (et produisant une unique donnée en sortie) sont expliquées. La méthodologie développée est ensuite décrite et appliquée sur des descriptions logicielles optimisées. L'analyse des instructions les plus impactantes pour la mise en œuvre du décodage va permettre l'identification de motifs d'instructions. Ces motifs sont analysés et de nouvelles instructions dédiées sont mises au point. Chacune des familles de codes sont ensuite présentées. Afin de démontrer la pertinence des choix effectués, le chapitre 4 présente aussi une évaluation des performances des décodeurs CCE, suite à l'intégration de nos nouvelles instructions. Ces évaluations sont réalisées sur des cœurs de processeur RISC-V, dont l'Instruction Set Architecture (ISA) a été étendue avec nos nouvelles instructions matérielles. Plusieurs cœurs RISC-V ont été sélectionnés et modifiés, leurs performances sont comparées à leurs architectures de base. Cela permet d'estimer les améliorations obtenues en termes de latence, mais également les surcoûts engendrés par l'utilisation de nos kits d'instructions.

La recherche d'optimisation des performances de calcul sur les cibles retenues nous a amené à étudier la prise en compte du parallélisme de données dans les architectures cibles. Nous avons donc mis au point des extensions du jeu d'instructions capables de travailler en mode SIMD (Single Instruction Multiple Data). Ces instructions et l'évolution de leurs

performances sont décrites dans le chapitre 5.

La conception des extensions présentées dans les chapitres précédents est contrainte par l'architecture interne du cœur de processeur ciblé : cette dernière ne peut fournir que deux opérandes. Ce choix sera remis en cause dans le chapitre 6. En effet, dans de nombreuses architectures de processeurs DSP, il est possible d'accéder à 3 opérandes en parallèle, par exemple pour réaliser des opérations de type MAC. Nous proposons ainsi une évolution de nos jeux d'instructions afin d'obtenir une plus grande liberté sur la construction des extensions et donc, à terme, de meilleures performances. L'étude qui en découle est analogue à celle présentée dans les chapitres précédents, mais elle considère donc un opérande supplémentaire en entrée des instructions. Les codes sources des décodeurs de CCE sont analysés et de nouvelles extensions sont détaillées. Afin d'évaluer l'impact de ces kits, un nouveau cœur RISC-V, plus complexe, est utilisé. Les résultats d'intégration de ces nouveaux kits sur cible FPGA sont présentés et discutés.

Le dernier chapitre conclut sur les travaux présentés dans ce manuscrit de thèse. Celui-ci expose les verrous scientifiques levés et résume les résultats obtenus. Une partie est dédiée aux différentes perspectives envisagées pour poursuivre ces travaux.

Les différentes contributions de ces travaux de thèse sont résumées ci-dessous :

1. Une méthodologie d'identification, d'intégration et d'évaluation d'instructions spécifiques a été détaillée et mise en œuvre pour améliorer le support des algorithmes de décodage des CCE. À partir d'une réécriture des codes logiciels de l'état de l'art, cette dernière a permis d'identifier des extensions scalaires et vectorielles compatibles avec les contraintes architecturales des processeurs usuels, *i.e.* RISC-V, INTEL et ARM.
2. Un ensemble d'extensions a été proposé. Ces extensions composées d'instructions possédant au maximum deux opérandes d'entrée et une sortie, ont été définies spécifiquement pour améliorer les performances des algorithmes de décodage des CCE. Les familles de CCE pour lesquelles des extensions ont été proposées sont les turbo-codes, les codes LDPC binaires, les codes LDPC-NB et les codes polaires. Différents cas d'usage ont été considérés, incluant le traitement scalaire ou vectoriel des données.
3. Ces travaux ont ensuite été étendus pour des cibles plus évoluées permettant de réaliser des opérations pouvant avoir jusqu'à 3 opérandes en entrée et une taille de registres de 64 bits. Ces modifications des hypothèses architecturales ont permis de créer de nouvelles instructions dédiées. Les extensions, composées d'instructions à

2 et à 3 opérandes, fournissent des facteurs d'accélération plus importants.

4. Le prototypage des diverses extensions proposées a été mené sur des cœurs de calculs de type RISC-V. Pour ce faire, un processus permettant l'intégration de ces fonctionnalités au travers des outils de compilation (GCC) pour permettre leur usage à partir de codes logiciels C/C++ a été mis en œuvre.

   L'utilisation de cœurs RISC-V open-source a, de plus, permis le déploiement sur circuit FPGA des prototypes facilitant une validation fonctionnelle et une évaluation fine des gains apportés par l'approche proposée, mais également des surcoûts matériels induits.

Ces différentes contributions ont été valorisées au travers de publications et présentations scientifiques au niveau national et au niveau international :

- Conférence internationale avec comité de lecture :
  - M. Tourres, C. Chavet, B. Le Gal, J. Crenne and P. Coussy, "Extended RISC-V hardware architecture for future digital communication systems" 2021 IEEE 4th 5G World Forum (5GWF), 2021, pp. 224-229
- Présentation internationale :
  - M. Tourres, C. Chavet, B. Le Gal, J. Crenne and P. Coussy, "Specialized Scalar and SIMD instructions for Error Correction Codes Decoding on RISC-V processor" Spring 2022 RISC-V Week, 3-5 May 2022.
- Conférence nationale :
  - M. Tourres, C. Chavet, B. Le Gal, J. Crenne and P. Coussy, "Architecture matérielle programmable optimisée pour les systèmes de communications numériques" Conférence Francophone d'informatique en Parallélisme, Architecture et Système (COMPAS), 6-9 Juillet 2021
- Présentation nationale :
  - M. Tourres, C. Chavet, B. Le Gal, J. Crenne and P. Coussy, "Architecture programmable pour les systèmes de communications numériques" Colloque GdR SoC2, 8-10 Juin 2021.
- Revue internationale avec comité de lecture *(En cours de rédaction)*
  - M. Tourres, C. Chavet, B. Le Gal and P. Coussy, "Specialized Scalar and SIMD instructions for Error Correction Codes Decoding on RISC-V processors"

Chapitre 2

Introduction aux codes correcteurs d'erreurs

Dans la partie initiale de ce premier chapitre, deux concepts sont introduits : celui de codage canal, une technique avancée utilisée dans les systèmes de communications numériques afin de fiabiliser les transmissions, ainsi que le concept de codes correcteurs d'erreurs (CCE) et leur positionnement dans les chaînes de communications numériques. La seconde partie de ce chapitre présente les principales familles de CCE actuellement utilisées dans différents standards de communication ainsi que les algorithmes de décodage associés communément employés.

2.1 Introduction au codage canal

Durant les deux dernières décennies, la quantité de systèmes communicants présents dans notre environnement a considérablement augmenté. Cette évolution rapide a été stimulée par l'émergence de nouveaux standards de communication (3G [1], WIFI [6],

WiMAX [7], 4G-LTE [3], 5G [8] et 6G [5]), dont le but était de répondre aux nouveaux usages, tout en permettant aux infrastructures de supporter l'augmentation croissante des besoins en termes de bande passante, mais aussi de réduire la consommation énergétique. Pour atteindre ces objectifs, de nombreux travaux de recherches se sont focalisés sur (a) la proposition de nouvelles formes d'ondes, et (b) la mise au point des algorithmes de traitement du signal associés. Ces innovations ont pour vocation de proposer des niveaux de performances élevés (débits, latence, QoS, efficacité énergétique) d'un point de vue théorique, tout en garantissant la faisabilité de ces approches à l'aide de technologies grand public.

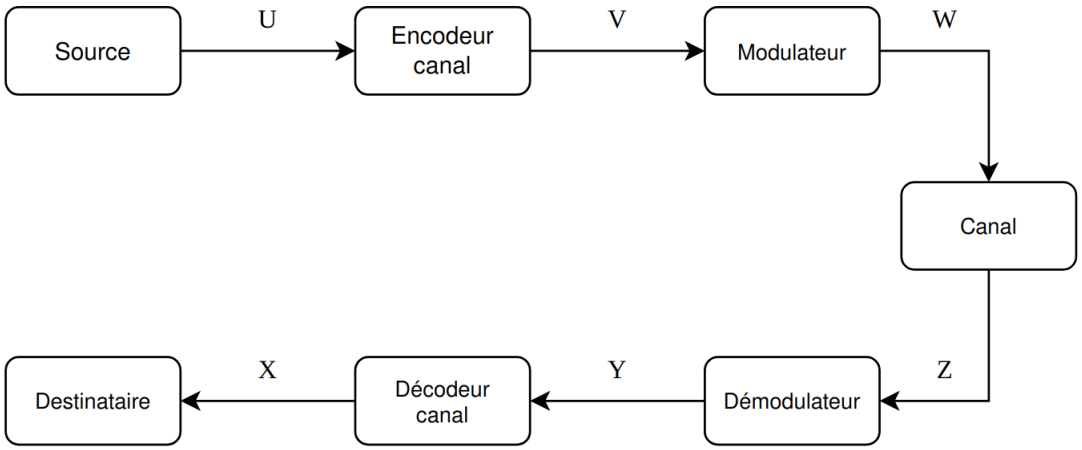

Figure 2.1 – Schéma simplifié d'une chaine de communications numérique

Les systèmes de communications numériques visent à permettre la transmission d'informations numériques d'un émetteur (la source) vers un ou plusieurs récepteurs (les destinataires). Ces systèmes peuvent être modélisés de manière simplifiée comme illustré dans la figure 2.1, représentant une chaîne de communication numérique.
Cette chaîne "gros grain", a été conceptualisée par Claude Shannon en 1948 [9]. Les principales étapes mises en œuvre lors de la transmission de données numériques d'un *émetteur* (source) vers un *récepteur* (destinataire) sont décrites dans ce schéma. L'échange d'informations entre source et récepteur est effectué au travers d'un canal de communication qui permet de modéliser les perturbations pouvant apparaître lors d'une communication réelle mettant en œuvre, par exemple, un câble, un couple d'antennes et/ou un support de stockage.
Les données transmises (**W**) à travers le canal sont dans le domaine continu, tandis que les informations provenant de la source (**U**) sont dans le domaine discret. Le *modulateur*

est responsable de la transformation du flux binaire (**V**) en une forme d'onde continue (**W**).
Dans le canal, les informations transmises (**W**) sont d'abord perturbées par (a) les imperfections des composants analogiques intégrés dans les étages de transmission et de réception Radio-Fréquence (RF) et (b) le support de transmission, comprenant les phénomènes physiques, mais également les interférences entre les différents systèmes.
Le *démodulateur* a pour but de transformer les informations provenant du canal (**Z**) dans le domaine discret (**Y**) afin de permettre dans un second temps la réception et le décodage du message transmis. Le processus de démodulation, produit en l'absence de bruit un vecteur d'information (**Y**) équivalent au vecteur d'information transmis (**V**). Toutefois, en fonction des conditions de communication, le message démodulé (**Y**) peut être différent de celui émis (**V**) impliquant la réception d'informations erronées.
Afin de corriger ces erreurs, et ainsi d'améliorer la Qualité de Service (QoS), il est judicieux d'ajouter et d'utiliser un encodeur et un décodeur de canal, respectivement dans l'émetteur et le récepteur. On parle alors de Codes Correcteurs d'Erreurs (CCE).

Il existe différents types de CCE. Dans ce chapitre, nous nous limitons à la présentation des CCE en bloc [10]. Afin de clarifier ce concept, l'encodeur de canal transforme la séquence binaire **U** composée de **K** bits en un mot de code (codeword) **V** composé de **N** bits avec $\mathbf{N} > \mathbf{K}$. La redondance $\boldsymbol{M = N - K}$ et l'information utile **K** sont liées par une relation mathématique. Le rendement d'un CCE est défini par $\boldsymbol{R = K/N}$, celui-ci est ainsi compris entre 1 et 0. Les **M** bits de redondance introduits dans **V** avant la transmission au travers du canal sont utilisés par le décodeur de canal afin d'améliorer l'estimation des **K** bits transmis et potentiellement détecter et corriger des erreurs de transmission dans **Y**. L'ajout de **M** bits de redondance augmente le nombre total de bits à transmettre via le canal et diminue l'efficacité spectrale du système de communication. Cependant, cette diminution est compensée par des gains importants en termes de qualité de service obtenus grâce à la correction des erreurs de transmission. L'utilisation de CCE améliore ainsi considérablement la qualité des transmissions, en permettant d'approcher la limite théorique de Shannon [9], comme cela est mis en exergue dans [11].

Depuis la proposition dans les années 1950 du code de Hamming [12], capable de détecter, d'identifier et de corriger un bit dans une séquence de **N** bits, de nombreux progrès ont été réalisés dans le domaine des CCE avec, par exemple, les codes de Reed-Solomon [13], les codes BCH [14], ect... Les informations binaires dites *informations dures* fournies par le processus de démodulation ont été d'abord considérées dans ces travaux. Cepen-

dant, les avancées technologiques au niveau des *Frontends* d'acquisition Radio-Fréquence (RF) ont permis d'obtenir des *informations souples* au niveau du démodulateur. Ces informations souples permettent d'avoir, en plus des décisions binaires les plus probables, des indicateurs de confiance (ex. *LLR - Log-Likelihood Ratios*) associés à chaque bit.
Cette évolution notable des systèmes de communication a permis d'améliorer, entre autres, le pouvoir de correction des algorithmes existants et de développer de nouvelles approches [10, 15] au prix d'une augmentation de la complexité calculatoire des algorithmes de décodage. Ainsi, en fonction de la famille de CCE considérée, du rendement de codage et de la complexité calculatoire des algorithmes utilisés, les différentes familles permettent de réaliser des communications numériques dans des environnements défavorables avec des rapports signal sur bruit (*SNR - Signal to Noise Ratio*) inférieurs à 0 dB [10].

La section qui suit a pour objectif de familiariser le lecteur avec les métriques de performances permettant d'évoluer le pouvoir de correction d'un CCE et ainsi proposer une mise en relief de la notion de décibels (dB) dans ce cadre.

2.2 Les métriques de performance du codage canal

Deux métriques sont principalement utilisées pour mesurer le pouvoir de correction d'un CCE : le BER, ou *Bit Error Rate* et le FER pour *Frame Error Rate*. Le premier est le ratio du nombre de bits erronés non corrigés par trames (nombre de bits erronés *vs* nombre de bits totaux transmis) et le second, de manière analogue, correspond au ratio du nombre de trames décodées par rapport aux trames envoyées.
Ces deux métriques sont obtenues à l'aide de simulations de Monte Carlo, le message émis par la source est connu et le bruit contrôlé par différentes techniques. Elles sont tracées avec pour abscisse le rapport signal à bruit (SNR).
Dans le cas des simulations pour des CCE, le SNR est le rapport de l'énergie moyenne par bit d'information émis Eb sur la densité spectrale de puissance du bruit N_0. Il est généralement calculé et représenté avec l'échelle logarithmique, donc en décibels (dB). Plus ce rapport augmente, plus l'énergie du message se "détache" du bruit, et par conséquent augmente la fiabilité d'estimation des erreurs issues du canal par le décodeur.
Ces résultats permettent de comparer l'efficacité d'un algorithme de décodage pour une famille de code donnée. À titre d'exemple, la figure 2.2 propose un comparatif entre un message sans aucune technique d'encodage *vs* un message avec l'application d'un CCE. Cette figure démontre le profil typique d'un CCE. La différence du pouvoir de correction

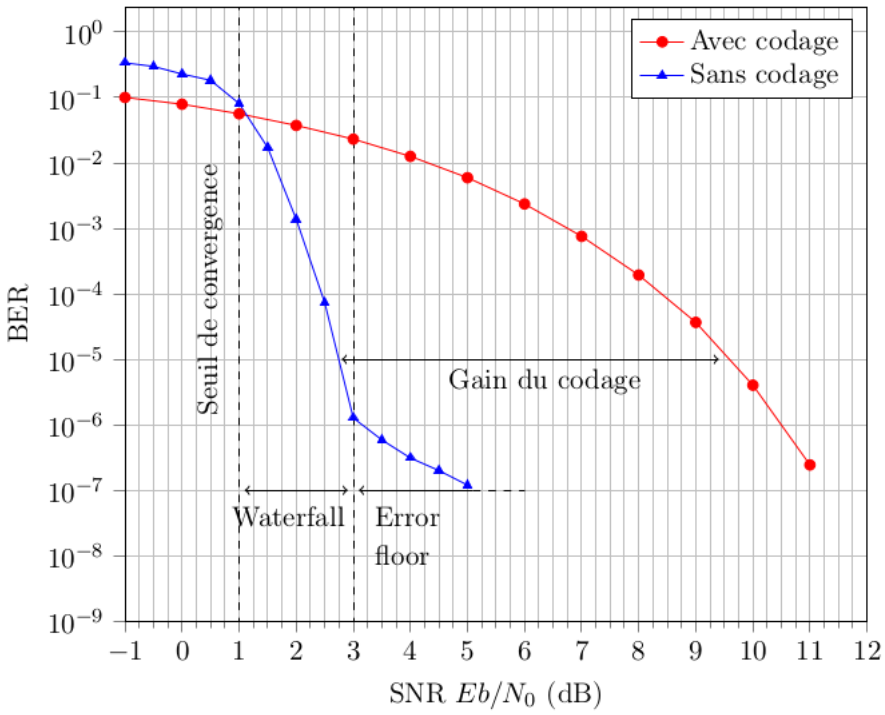

Figure 2.2 – Évaluation des performances d'un CCE

est évidente après 1 dB, qualifié de *seuil de convergence*. Cependant, en fonction du CCE utilisé et des paramètres de l'algorithme de décodage mis en œuvre, 2 phénomènes sont observables : le *Waterfall*, et le *Error floor*. Le premier est la zone dans laquelle les performances de correction du code s'améliorent rapidement, par ordre de magnitudes supérieur relativement au SNR. Concernant le second phénomène, le plancher d'erreurs est ainsi désigné, car il met en évidence la capacité de correction du code étant parvenue à un plateau lorsque le SNR a atteint une certaine valeur.

Dans la section suivante, nous présenterons de manière concise les principales familles de CCE actuellement utilisées dans les standards de communications numériques ainsi que les propriétés des algorithmes de décodage qui leur sont associées.

2.3 Les différentes familles de CCE

L'intégration des CCE dans les systèmes numériques a débuté dans les années 1950 par l'intégration des codes de Hamming [12], qui ont été les premiers CCE proposés dans la littérature. Ces derniers sont des codes correcteurs linéaires qui utilisent quelques bits

de redondance pour protéger l'information binaire, permettant ainsi de détecter et de corriger une fraction d'erreurs. Les recherches dans ce domaine se sont poursuivies et ont permis l'émergence des codes de Reed-Solomon dans les années 1960, ainsi que celle des codes convolutifs dans les années 1970. Ces deux familles améliorent fortement le pouvoir de correction des codes de Hamming au prix d'une augmentation notoire de la complexité calculatoire des algorithmes mis en œuvre pour leur décodage. Ces derniers ont été mis en application dans les premières générations de systèmes de communications numériques sans fils terrestres, mais aussi dans le domaine spatial. Toutefois, ils ont été remplacés dans les années 1990-2000 par de nouvelles familles de codes plus performantes telles que, par exemple, les turbo codes et les codes LDPC. Ces familles de codes, associées aux codes polaires et les codes LDPC non binaires, sont d'usage aujourd'hui dans des standards usuels tels que la 4G [2], et la 5G [8], ainsi que dans les standards de communications numériques par satellites [16-19]. Cet engouement autour de ces quatre familles est dû aux compromis intéressants qu'elles offrent en termes de pouvoir de correction et de complexité calculatoire.

2.3.1 Les turbo codes

Les turbo codes sont une classe de CCE découverte au début des années 1990 [10, 20]. Le développement de ces codes est fondé sur l'utilisation de deux codes convolutifs parallèles appelés codes composants, combinés par un processus d'encodage itératif. Ces derniers fonctionnent de manière complémentaire et exploitent l'interférence mutuelle pour améliorer les performances de correction d'erreurs. Les turbo codes ont permis d'obtenir des performances de décodage proches de la limite théorique de Shannon. En plus de leurs excellentes performances de décodage, la complexité calculatoire modérée de ces algorithmes et donc de leurs réalisations matérielles [21], a accéléré leur inclusion dans de nombreux standards au début des années 2000 tels que les standards 3G [1], 4G [2], et 4G LTE [3]. Ils ont aussi été utilisés dans le domaine des communications satellitaires [16, 22]. Depuis leur découverte, ces codes ont été étudiés avec attention [23] afin d'améliorer leurs niveaux de performances :

(a) en réduisant l'apparition des planchers d'erreurs [24] ;

(b) en proposant des stratégies de parallélisation des algorithmes de décodage pour monter en débit [25, 26] ;

(c) en concevant des architectures efficaces matériellement et énergétiquement pour

faciliter leur intégration dans des systèmes embarqués contraints [27-29].

Le processus de décodage associé aux turbo codes est schématisé dans la figure 2.3. De manière concise, les informations provenant du canal de communication notées ($\boldsymbol{S_n}, \boldsymbol{P_n}, \boldsymbol{P_i}$) sont traitées de manière itérative dans le récepteur. Dans un premier temps, les vecteurs d'information $\boldsymbol{S_n}$ et les données redondantes $\boldsymbol{P_n}$ sont traitées par le premier SISO (❶ figure 2.3). Cette étape produit un vecteur de données extrinsèques $\boldsymbol{E_n}$, informations qui sont ensuite entrelacées par $\boldsymbol{\Pi}$ afin de produire le vecteur $\boldsymbol{L_i}$ (❷ figure 2.3). Dans un second temps, les données $\boldsymbol{S_i}, \boldsymbol{P_i}, \boldsymbol{L_i}$ sont traitées de manière similaire par le second SISO (❸ figure 2.3). Ce traitement produit les extrinsèques $\boldsymbol{E_i}$ qui post-entrelacement deviennent $\boldsymbol{L_n}$. L'exécution séquentielle des SISO ❶ et ❸ est réalisée $\boldsymbol{i}$ fois au maximum, ou jusqu'à obtention d'un mot de code valide en sortie du décodeur $\boldsymbol{U}$. Une formulation algorithmique du processus de turbo décodage est proposée dans l'Algorithme 1 avec $\boldsymbol{\Pi}$ et $\boldsymbol{\Pi^{-1}}$ les opérations d'entrelacement et de désentrelacement [24].

Les traitements nommés SISO (Soft-Input, Soft-Output), utilisés dans la figure 2.3, sont des traitements œuvrant sur un treillis construit à partir des spécifications du turbo code [10]. Pour identifier la séquence la plus probable dans le treillis constitué de $\boldsymbol{S}$ états figure 2.4, des algorithmes de type BCJR [30] sont utilisés. L'algorithme BCJR parcourt le treillis dans les deux sens afin d'estimer le chemin le plus probable à partir de cette estimation et de calculer les $\boldsymbol{K}$ valeurs extrinsèques.

L'algorithme privilégié pour réaliser ce calcul au niveau du treillis lors de l'implantation temps réel de turbo décodeurs est l'algorithme BCJR Max-Log-MAP [31], car les opérations arithmétiques mises en œuvre sont peu complexes. Toutefois, il est à noter que ce dernier réduit légèrement les performances de décodage. Ce traitement, dont la

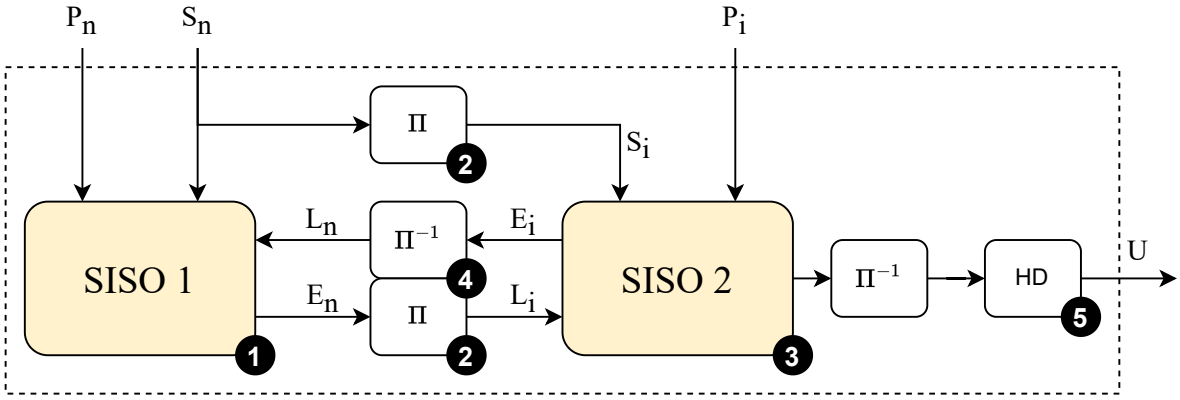

Figure 2.3 – Structure d'un décodeur de turbo code

Algorithm 1 Formulation conventionnelle de l'algorithme de décodage turbo

1 : **Step 1** : Chargement des données
2 : $S_i = \Pi(\ S_n\ )$
3 : $E_i = \text{zeros}(\ E_i\ )$

4 : **Step 2** : Décodage itératif
5 : **for all** $t = 0 \rightarrow (iter\_max - 1)$ **do**
6 : $\quad E_n = \text{SISO}(\ P_n, S_n, E_i\ )$
7 : $\quad E_n = \Pi\ (\ E_n\ )$
8 : $\quad E_i = \text{SISO}(\ P_i, S_i, E_i\ )$
9 : $\quad E_i = \Pi^{-1}(\ E_i\ )$
10 : **end for**

11 : **Step 3** : Décision dure
12 : $u_n =$ Décision dure$(\ P_n, S_n, E_i\ )$

complexité calculatoire domine dans le décodeur, est composé de boucles itératives calculant séquentiellement les métriques internes au processus $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ afin de produire $\boldsymbol{K}$ informations extrinsèques. Ces calculs impliquent des accès irréguliers en mémoire liés à la structure du treillis.

Afin de rendre le processus de décodage des turbo codes compatible avec les contraintes temps réel, différents verrous ont dû être levés [21, 23]. Tout d'abord, dans le but d'atteindre des débits compatibles avec les besoins applicatifs, différentes stratégies de parallélisation du parcours du treillis ont été proposées tout comme l'exécution parallèle des SISO [29].

Toutefois, de manière à permettre l'implantation temps réel de turbo décodeurs dans des contextes contraints en termes d'énergie et de débit [28], de nombreuses solutions ont été proposées pour simplifier les algorithmes de décodage [23] mis en œuvre et augmenter la parallélisation des calculs [25]. Ces avancées ont permis l'intégration des turbo codes dans la 3G [1], la 4G LTE [3] et potentiellement leur usage dans la 5G/6G [5, 8] ou les standards associés à l'IoT [32]. En parallèle des travaux menés autour des turbo codes, d'autres familles de codes correcteurs, tels que les codes LDPC, ont aussi émergé grâce à leur excellent pouvoir de correction.

2.3.2 La famille des codes LDPC binaires

Les codes LPDC (Low Density Parity Check) sont proposés pour la première fois par Robert G. Gallager au début des années 1960 [33]. À l'origine, malgré l'intérêt de

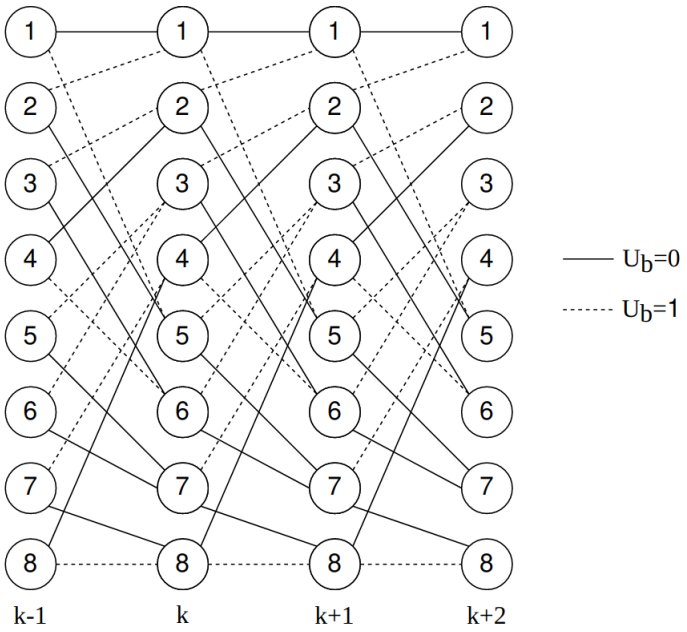

Figure 2.4 – Exemple de treillis à 8 états pour turbo code

la communauté pour ses travaux, la complexité calculatoire induite par le processus de décodage était trop importante vis-à-vis des capacités d'intégration. Il fallut attendre la fin des années 1990, et les progrès technologiques réalisés dans le domaine de l'intégration des circuits numériques pour que ces travaux soient exhumés par MacKay [34]. Comme les turbo codes, les codes LDPC possèdent un pouvoir de correction proche de la limite de Shannon [34]. Les algorithmes de décodage mis en œuvre sont aussi itératifs et possèdent des complexités calculatoires modérées [35, 36]. Toutefois, contrairement aux turbo codes cette famille de CCE qui n'était pas protégée par des brevets, a généré un engouement dans la communauté scientifique, tant sur les aspects théoriques liés à la construction des codes et leur décodage, que sur la conception de circuits numériques efficaces en termes de débit, d'énergie et de latence. Ainsi, un nombre important de travaux de recherche se sont focalisés sur la conception de codes LDPC, que l'on nomme parfois matrices de parité, efficaces [37, 38] pour la simplification des algorithmes de décodage [39] et la conception d'architectures matérielles dédiées [40]. Les avancées significatives dans ces domaines, associées aux besoins de montée en débit des systèmes de communications numériques, ont abouti à l'inclusion en 2005 des codes LDPC dans les standards WiMAX [41] et DVB-S2 [42]. Ces standardisations ont amplifié l'intérêt des communautés universitaires et industrielles pour cette famille de CCE, engendrant ensuite son adoption dans les standards DVB-S2X [19] et CCSDS [18]. Plus récemment, les codes LDPC ont aussi été

Algorithm 2 Implémentation standard du BCJR

for Toutes les frames **do**
  for $k = 0; k < K; k = k + 1$ **do**
    $\gamma^k \leftarrow calculGamma\ (l^k_{sys}, l^k_p, l^k_e)$
  end for
  $\alpha^0 \leftarrow initAlpha()$
  for $k = 0; k < K; k = k + 1$ **do**
    $\alpha^k \leftarrow calculAlpha\ (\alpha^{k-1}, \gamma^{k-1})$
  end for
  $\beta^{K-1} \leftarrow initBeta()$
  for $k = K - 2; k \geq 0; k = k - 1$ **do**
    $\beta^k \leftarrow calculBeta\ (\beta^{k+1}, \gamma^k)$
  end for
  for $k = 0; k < K, k = k + 1$ **do**
    $L^k_e \leftarrow calculExtrinsèque(\alpha^k, \beta^k, \gamma^k)$
  end for
end for

adoptés dans le standard 5G [8] pour assurer la protection des paquets de données. Les codes LDPC, sont des codes linéaires en blocs définis à l'aide de matrices de parité de faible densité nommées **H**. Ces matrices, dont un exemple est présenté dans l'équation 2.1, sont composées de **N** colonnes et de $\boldsymbol{M = N - K}$ lignes. À l'émission, un vecteur **x** de **N** bits est généré pour transmettre **K** bits d'information à l'aide d'une matrice génératrice nommée $\boldsymbol{G = H^{-1}}$. Si l'on considère qu'un vecteur $\boldsymbol{\lambda}$ composé de **N** informations bruitées est reçu après le passage des données par le canal de transmission, les **M** équations de parité issues de **H** sont utilisées à la réception pour détecter et corriger les erreurs présentes dans $\boldsymbol{\lambda}$. Les **N** colonnes et les **M** lignes de **H** représentent respectivement les nœuds variables ($\boldsymbol{V_n}$) et les nœuds de vérification de parité ($\boldsymbol{C_n}$). Pour chaque élément non nul de **H**, il existe une dépendance entre les éléments $\boldsymbol{V_n}$ et $\boldsymbol{C_n}$. Le nombre d'éléments de la ligne J et de la colonne I de **H** sont respectivement les degrés de $\boldsymbol{C_n}$ et $\boldsymbol{V_n}$, notés $\boldsymbol{D_c}(j)$ et $\boldsymbol{D_v}(i)$ pour un code LDPC($\boldsymbol{D_c}$, $\boldsymbol{D_v}$).

$$\mathbf{H} = \begin{array}{c} \\ \boldsymbol{C_0} \\ \boldsymbol{C_1} \\ \boldsymbol{C_2} \\ \boldsymbol{C_3} \\ \boldsymbol{C_4} \end{array} \begin{array}{c} \begin{array}{cccccccc} \boldsymbol{V_0} & \boldsymbol{V_1} & \boldsymbol{V_2} & \boldsymbol{V_3} & \boldsymbol{V_4} & \boldsymbol{V_5} & \boldsymbol{V_6} & \boldsymbol{V_7} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{2.1}$$

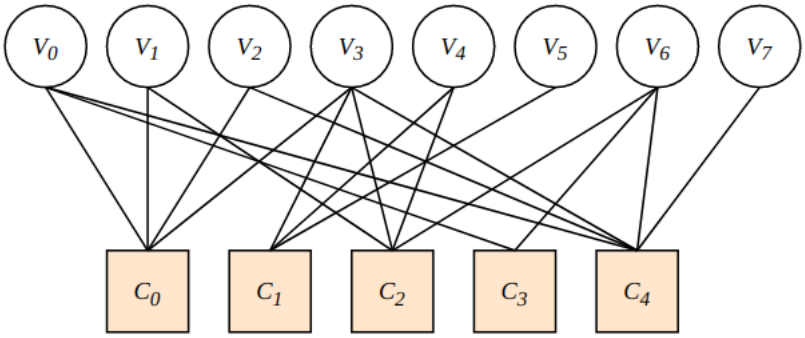

Figure 2.5 – Représentation d'un code LDPC sous forme d'un graphe de Tanner

Les matrices **H** peuvent être représentées de manière visuelle à l'aide de graphes bipartites nommés graphes de Tanner [43]. Par exemple, le graphe de Tanner représenté en figure 2.5 est équivalent au code LDPC (8,5) décrit dans l'équation 2.1.

Dans leur forme générale [34, 44], les matrices **H** sont irrégulières et non-structurées. Cependant, la conception de décodeurs efficaces pour un code LDPC non-structuré est admise par tous comme étant une tâche ardue [45]. Ainsi, pour faciliter la conception de décodeurs LDPC temps réel, qu'ils soient matériels ou logiciels, les codes LDPC Quasi Cycliques (QC-LDPC) ont été adoptés dans la majorité des standards [6, 7, 46, 47], y compris pour le standard 5G 3GPP [8]. Cette sous-famille des codes LDPC, permet de garantir un niveau de parallélisme constant durant le processus de décodage ainsi que l'absence de conflits d'accès à la mémoire. Pour procurer ces caractéristiques, les codes LDPC-QC [48] de taille $\boldsymbol{N} \times \boldsymbol{M}$ sont structurés à partir de sous matrices de taille $\boldsymbol{Z} \times \boldsymbol{Z}$.

Pour décoder les codes LDPC, les algorithmes de type passage de messages (MP, message passing) sont habituellement utilisés. L'algorithme de référence en termes de pouvoir de correction est l'algorithme somme-produit (SPA) détaillé dans [49]. Le principe général de décodage est le suivant : à partir des valeurs des informations en provenance du canal ($\boldsymbol{V_n}$), des messages $\boldsymbol{V_n} \rightarrow \boldsymbol{C_n}$ sont calculés, puis transmis aux nœuds de parité ($\boldsymbol{C_n}$). Ces nœuds de parité, en fonction de leur validation ou non, retournent des messages ($\boldsymbol{C_n} \rightarrow \boldsymbol{V_n}$) vers les nœuds $\boldsymbol{V_n}$ afin de confirmer ou d'infirmer les données reçues. Le processus de décodage est itératif, il s'arrête lorsque tous les nœuds $\boldsymbol{C_n}$ sont validés, ou bien lorsqu'un nombre maximal d'itérations a été réalisé.

La nature des opérations réalisées dans l'algorithme SPA original [49] ainsi que l'ordonnancement des calculs n'étaient pas adaptés à l'implémentation de décodeurs temps réel. Des simplifications algorithmiques permettant l'abandon des fonctions transcendantales et le passage en virgule fixe du processus de décodage ont été proposés dans [50-54].

Ces variantes algorithmiques regroupées sous la dénomination générique d'algorithmes *Min-Sum* ont ensuite bénéficié d'une modification de l'ordonnancement des calculs lors du processus de décodage, passant d'une approche par vagues en deux phases (TPMP – two phase message passing) à une approche par couches horizontales (HL – horizontal layered schedule) [55]. Cette modification de l'ordonnancement des calculs permet de réduire d'environ *2/5* la complexité mémoire et de diviser, approximativement, par deux le nombre d'itérations de décodage à iso-performances. Ces caractéristiques avantageuses ont entrainé une utilisation de la stratégie HL dans les décodeurs LDPC récents, qu'ils soient matériels [52-54, 56] ou logiciels [57-60].

L'Algorithme 3 présente la formulation d'un algorithme de décodage LDPC dans lequel l'approximation *Min-Sum* est couplée avec un ordonnancement par couche horizontale $\boldsymbol{C_n}$ [55]. Cette formalisation de l'algorithme de décodage met en évidence la structuration du processus de décodage en nids de boucles sur 3 niveaux. Différentes stratégies de parallélisation ont été proposées et mises en œuvre pour accélérer l'implantation du processus de décodage :

- les kernels internes (3.1, 3.2, 3.3) qui ont des nombres d'itérations variables en fonction du degré des nœuds $\boldsymbol{C_n}$ peuvent être déroulées ou partiellement déroulées pour certains codes.
- le kernel (3) peut être déroulé partiellement d'un facteur $\mathbf{Z}$ dans le cas de codes LDPC QC. Un déroulage partiel plus important est difficile à obtenir à cause des dépendances de données liées à la structure de la matrice $\mathbf{H}$.
- Le kernel (2) doit quant à lui itérer séquentiellement i fois à cause des dépendances de données entre les itérations.

L'exploitation de ces différents niveaux de parallélismes associés à des contraintes applicatives spécifiques, a permis la conception d'architectures numériques variées permettant d'atteindre des débits de plusieurs Gbps sur technologies FPGA [61, 62], mais aussi d'atteindre des débits similaires sur des architectures programmables [63, 64].
L'ensemble de ces solutions ont permis d'obtenir des compromis *débit vs flexibilité*, permettant l'émergence et l'usage à grande échelle des plateformes SdR [65, 66] et des systèmes de type Cloud-RAN [67, 68] plébiscités par exemple pour le déploiement de la 5G [69].

En supplément des travaux relatifs à la construction des codes LDPC et de l'amélioration des caractéristiques des décodeurs matériels et logiciels, d'autres initiatives se sont focalisées sur l'extension de cette famille aux corps de Galois non binaires ($\mathbb{GF}(\boldsymbol{q})$ avec $\boldsymbol{q} \geq 2$).

Algorithm 3 Algorithme Min-Sum basé sur un ordonnancement par couches horizontales

Kernel 1 : Initialisation
Charge la trame dans la mémoire dédiée au VN
for all $m \in \Psi, n \in \Phi(m)$ **do**[Ψ *est l'ensemble des CNs*]
    Initialisation de la mémoire message : $\mathcal{L}_{mn}^{(0)} = 0$
end for
Kernel 2 : Traitement des $Imax$ itérations de décodage
for all $i = 1 \rightarrow (Imax)$ **do**
    for all $m \in \Psi$ **do**[*Pour tout CN*]
        Kernel 3.1 : Calcul des messages entrants $\mathcal{L}_{nm}$
        for all $n \in \Phi(m)$ **do**[*VNs contribuant à CN*]
            $\mathcal{L}_{nm}^{i} = VN_n - \mathcal{L}_{mn}^{(i-1)}$
        end for
        Kernel 3.2 : Génération des messages sortants $\mathcal{L}_{mn}$
        for all $n \in \Phi(m)$ **do**
            $$sign(\mathcal{L}_{mn}^{i}) = \left[\prod_{n' \in \Phi(m)/n} sign(\mathcal{L}_{n'm}^{(i-1)}) \right]$$
            $$|\mathcal{L}_{mn}^{i}| = \left[\min_{n' \in \Phi(m)/n} |\mathcal{L}_{n'm}^{(i-1)}| \right]$$
        end for
        Kernel 3.3 : Mise à jour des VNs
        for all $n \in \Phi(m)$ **do**
            $VN_n = \mathcal{L}_{nm}^{(i)} + \mathcal{L}_{mn}^{(i)}$
        end for
    end for
end for
Kernel 4 : Décision dure sur chaque LLR de la trame
for all $n \in trame$ **do**
    $$c_n = \begin{cases} 0, & \text{if } VN_n \leq 0 \\ 1, & \text{if } VN_n > 0 \end{cases}$$
end for

2.3.3 La famille des codes LDPC non binaires (LDPC-NB)

Les codes LDPC binaires offrent des performances de décodage proches de la limite de Shannon lorsque la taille des trames est suffisamment grande ($\mathbf{N} > 1000$). Cependant, dans de nombreux domaines applicatifs, cette condition n'est pas remplie, car les systèmes échangent peu d'informations et nécessitent des latences faibles, impliquant de fait l'usage de trames courtes.
Pour améliorer les performances de décodage dans ces conditions, une extension des codes LDPC binaires au corps de Galois avec ($\mathbb{GF}(\boldsymbol{q})$ avec $\boldsymbol{q} \geq 2$ a été proposée dans [70, 71]. L'utilisation de corps de Galois non binaires permet d'augmenter la diversité et de lutter contre les cycles courts dans les codes.

L'utilisation de corps de Galois avec $\boldsymbol{q} \in \{16, 64, 256\}$ permet d'améliorer le pouvoir de correction de 1,0 dB à 1,3 dB face à des codes LDPC binaires de tailles équivalentes, comme cela est démontré dans [17, 72]. Toutefois, l'amélioration notable des performances de décodage est obtenue au prix d'une augmentation polynomiale de la complexité calculatoire [70].
Par conséquent, un nombre important de travaux tels que [73] se sont concentrés sur cette nouvelle famille de codes. Un code NB-LDPC **(N, K)** est un code en bloc linéaire défini par une matrice de parité de taille $\mathbf{M}\times\mathbf{N}$ appelée **H**, dont les éléments non-nuls appartiennent à ($\mathbb{GF}(\boldsymbol{q})$ comme cela est présenté dans l'équation 2.2 ci-dessous pour $\boldsymbol{q} = 128$.

$$\mathbf{H} = \begin{array}{c} \\ \boldsymbol{C_0} \\ \boldsymbol{C_1} \\ \boldsymbol{C_2} \\ \boldsymbol{C_3} \end{array} \begin{array}{c} \begin{array}{cccccccc} \boldsymbol{V_0} & \boldsymbol{V_1} & \boldsymbol{V_2} & \boldsymbol{V_3} & \boldsymbol{V_4} & \boldsymbol{V_5} & \boldsymbol{V_6} & \boldsymbol{V_7} \end{array} \\ \begin{pmatrix} \alpha_{00} & \alpha_{10} & \alpha_{20} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{31} & \alpha_{41} & \alpha_{51} & 0 & 0 \\ \alpha_{02} & 0 & 0 & \alpha_{32} & 0 & 0 & \alpha_{62} & 0 \\ 0 & \alpha_{13} & 0 & 0 & \alpha_{43} & 0 & 0 & \alpha_{73} \end{pmatrix} \end{array} \tag{2.2}$$

Contrairement aux codes LDPC binaires, les éléments $\boldsymbol{V_N}$ et $\boldsymbol{C_N}$ sont des symboles composés de $\boldsymbol{log_2(q)}$ bits. La modélisation sous la forme de graphe de Tanner de ces matrices met en évidence les opérations de multiplication et de division des messages $\boldsymbol{V_N}\rightarrow\boldsymbol{C_N}$ et $\boldsymbol{C_N}\rightarrow\boldsymbol{V_N}$ dans ($\mathbb{GF}$) lors de l'échange des messages, comme représenté dans la figure 2.6.

Le décodage des codes NB-LDPC s'effectue grâce à l'utilisation d'algorithmes de passage de messages équivalents à ceux employés pour le décodage des codes LDPC binaires.

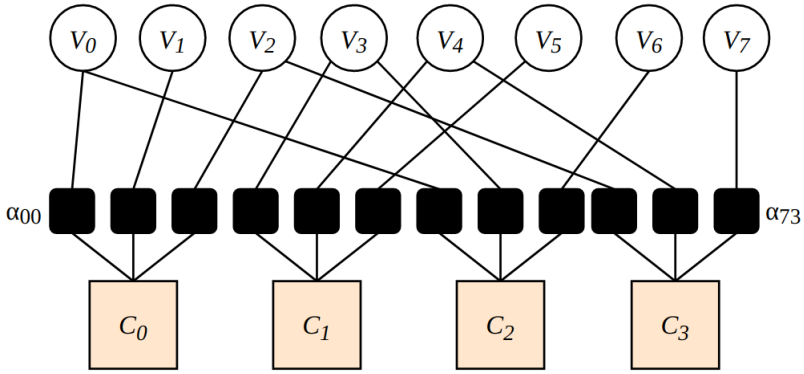

Figure 2.6 – Graphe de Tanner correspondant à la matrice LDPC-NB décrite dans l'équation 2.2.

Toutefois, les algorithmes employés sont plus complexes, principalement à cause de la nature des données échangées. En effet, les valeurs internes des nœuds $\boldsymbol{V_N}$ et $\boldsymbol{C_N}$ ne sont plus des données scalaires, mais des vecteurs composés de $\mathbf{q}$ valeurs de LLR. Il en va de même pour les messages échangés entre les différents nœuds. Les meilleures performances de correction sont obtenues en exécutant l'algorithme *Sum-Product* (SPA) [74] ou sa reformulation FFT-SPA [74] qui réduit fortement la complexité calculatoire du processus de décodage à iso-performance.

L'algorithme 4 présente une formalisation de l'algorithme de décodage *Min-Sum* basée sur un ordonnancement horizontal des calculs de parité [75]. Cet ordonnancement des calculs de parité permet d'obtenir une accélération de la convergence du processus de décodage et diminue la complexité mémoire.
L'algorithme 4 met en évidence une structure en nids de boucle, proche de celle présente dans les algorithmes des décodages des codes LDPC binaires (algorithme 3). Toutefois, la complexité calculatoire et le parallélisme de calcul présents ici sont localisés au sein des kernels qui ne manipulent plus des données scalaires, mais des vecteurs de données dont la taille est proportionnelle à $2^{\mathbb{GF}}$. L'algorithme est inspiré des travaux détaillées dans [75-78], proposant un algorithme FFT-SPA adapté à une exécution sur des cibles programmables multicœurs et many-cœurs.
Des variantes algorithmiques [79-81] supportant le codage des données en virgule fixe sur un faible nombre de bits ont été élaborées à partir de l'algorithme de décodage SPA. Ces variantes nommées *Min-Sum* (MS), *Min-Max* (MM) et *Extended Min-Sum* (EMS) ont permis par la suite le développement de décodeurs matériels dédiés [82, 83], atteignant de hauts débits avec des complexités matérielles maîtrisées. Cependant, ces niveaux élevés de performance démontrés par ces architectures dédiées ont été obtenus au prix d'une

Algorithm 4 Algorithme horizontal TPMP Min-Sum

Étape 1 : Initialisation
$p_i(x) = p(v_i = x|y{-}i)$
$m^0_{ji}(x) = 1$
▷ Effectue le processus avec $iter\_max$ itérations
for all $t = 1 \rightarrow (iter\_max)$ **do**
Étape 2 : Pour chaque C_n du code LDPC-NB
  for all $j \in C$ **do**
    ▷ Calcul les messages entrants m_{ij}
    for all $i \in \Psi(j)$ **do**
      $m^t_{ij} = p_i(x) - m^{t-1}_{ji}(x)$
      $m^t_{ij} = m^t_{ij} - min(m^t_{ij})$ // <- min des GF
      $M^t_{ij} = \Theta_{\alpha_{ij}}(m^t_{ij})$
    end for
    ▷ Noeuds parité, recherche du minimum dans le domaine $\mathbb{GF}$
    for all $i \in \Psi(j)$ **do**
      $\tilde{M}^t_{ji} = min \sum_{i' \in \Psi(j)/i} \tilde{M}^t_{i'j}$
    end for
    ▷ Calcul des messages m_{ij} sortants
    for all $i \in \Psi(j)$ **do**
      $m^t_{ji} = \Theta^{-1}_{\alpha_{ij}}(M^t_{ji})$
      $p_i(x) = p_i(x) + m^t_{ji}$
    end for
  end for
end for
Étape 3 : Décision dure des symboles
for all $i \in V$ **do**
  $\hat{v}_i = argmax(p_i(x))$
end for

réduction drastique de la flexibilité des décodeurs et nécessitent en amont de nombreuses étapes de simplification du décodeur vis-à-vis du code à traiter, étapes qui peuvent s'avérer chronophages [84].

Les gains de codage fournis par cette famille de codes pour des courtes trames, combinés à leur capacité à s'associer avec des modulations d'ordre élevé ainsi que l'émergence de solutions d'implantations efficaces, ont motivé son introduction dans plusieurs standards spatiaux CCSDS [17], BeiDou Navigation System [85] ainsi que son usage par exemple, dans le projet ANR QCSP [86, 87] pour le domaine des communications IoT.

2.3.4 La famille des codes polaires

Les codes polaires proposés en 2008 par Arikan [88] sont des codes linéaires en blocs dont la taille des blocs $\boldsymbol{N}$ évolue en 2^n avec $\boldsymbol{n}$ un nombre entier. Ces codes permettent d'atteindre théoriquement des performances proches de la capacité sur un canal gaussien lorsque $\boldsymbol{N}$ tend vers l'infini. Afin d'étudier leurs performances de décodage lorsque $\boldsymbol{N} < \infty$ et les comparer aux décodeurs développés pour les turbo codes et les codes LDPC, de nombreux travaux de recherche ont été menés durant la dernière décennie [89, 90].

L'algorithme de décodage initial nommé Successive Cancellation (SC) [91, 92] a mis en évidence la faible complexité calculatoire du processus de décodage. Cependant, l'écart de performance en termes de pouvoir de correction avec les autres familles de CCE a motivé la proposition d'algorithmes plus complexes tels que le Simplified-SC [93-95], le SC-List [96, 97], le SC-Stack [98], le SC-Flip [99, 100] et le SC-SCAN [101].

Ces derniers, au prix de complexités calculatoires plus importantes, permettent de compenser cette faiblesse et d'approcher et égaler les performances décodeurs LDPC et des turbo codes. En parallèle, différentes architectures de décodeurs décrites au niveau RTL ont été détaillées dans la littérature [91, 93, 94, 102-104] afin de montrer la viabilité de ces algorithmes de décodage. Ces diverses avancées ont abouti leur inclusion en 2017 dans le standard 5G [8], en remplacement des turbo codes, pour la protection des canaux de contrôle.

Le processus de codage côté émetteur consiste à multiplier la matrice de Kronecker [88] par un vecteur $\hat{\boldsymbol{u}}$ de $\mathbf{N}$ bits contenant $\mathbf{K}$ bits d'information et $\boldsymbol{M} = \boldsymbol{N} - \boldsymbol{K}$ bits gelés. Ces bits gelés dont les positions et les valeurs sont partagées par l'émetteur et le récepteur jouent le rôle de redondance. Le processus de décodage vise à faire successivement une estimation pour chaque valeur des bits $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$ sur la base des bits estimés précédemment $\hat{\boldsymbol{u}}_{(\mathbf{0}:\boldsymbol{i}-\mathbf{1})}$. Comme proposé dans [105], une représentation en *Factor Graph* (FG) des codes polaires peut être utilisée pour représenter le calcul de $\boldsymbol{\lambda}_{\mathbf{0}:\boldsymbol{i}}$. La figure 2.7 contient un exemple de *factor graph* pour un code de largeur $\boldsymbol{N} = \mathbf{8}$. Le décodeur estime successivement les bits $\hat{\boldsymbol{u}}$ à partir des données (LLR) provenant du canal et injectés à la droite du graphe. La nature séquentielle de l'algorithme de décodage SC liée au calcul ordonné des sorties $\hat{\boldsymbol{u}}$ implique de fortes dépendances de données entre les nœuds. Cette caractéristique limite le parallélisme exploitable et génère un parallélisme de calcul, fluctuant en fonction des étapes, comme cela est mis en évidence par la taille des rectangles bleus dans la figure 2.7. Afin de mieux mettre en évidence les dépendances de données ainsi que la variation du parallélisme de calcul, il est possible de modéliser le processus de décodage à l'aide

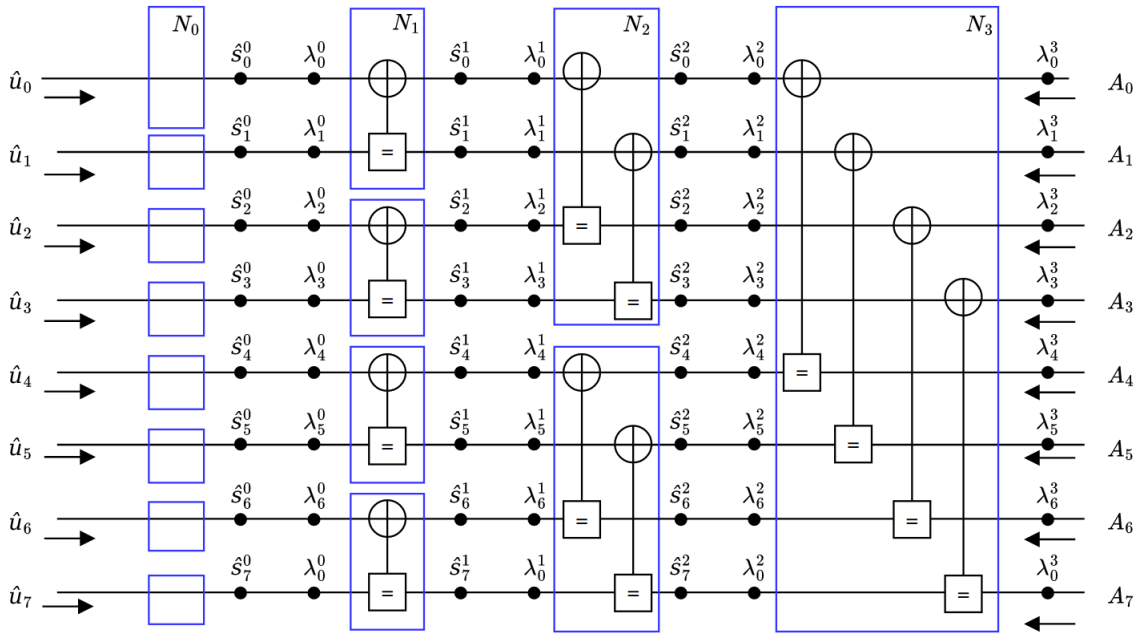

Figure 2.7 – Représentation en factor graph du décodage SC d'un code polaire de taille N=8.

d'un arbre binaire comme suggéré dans [93]. Lors du décodage d'une trame composée $\mathbf{N}$ LLRs, l'arbre binaire représenté dans la figure 2.8 est parcouru de façon récursive depuis le nœud $(\boldsymbol{N_3})$ de manière à produire l'ensemble des $\boldsymbol{N/2}$ sorties $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$. Le parcours de l'arbre binaire est opéré dans l'ordre suivant : du nœud racine vers le sous-nœud gauche puis vers le sous-nœud droit. Le nœud racine est situé au niveau $\boldsymbol{d = log(N) - 1}$ et les nœuds feuilles sont les nœuds du niveau $\boldsymbol{d = 0}$. Le nœud racine $\boldsymbol{N_3}$ reçoit $\boldsymbol{N}$ valeurs de LLRs provenant du canal et échange successivement des données avec les sous-nœuds gauches et droits du niveau $\boldsymbol{N_2}$. Les nœuds feuilles $\boldsymbol{N_0}$ sont appelés par les nœuds $\boldsymbol{N_1}$ et ils génèrent l'estimation $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$. En supposant qu'un nœud non feuille/non racine $\boldsymbol{N_j}$ reçoit $\boldsymbol{\lambda_{a:a+J-1}}$, il exécute l'Algorithme itératif décrit dans l'Algorithme 5. Les fonctions f et g mises en œuvre dans lors du processus de décodage dans la figure 2.8 et l'algorithme 5 possèdent une complexité calculatoire faible et sont définies par :

— $f(\lambda_a, \lambda_b) = sign(\lambda_a, \lambda_b).min(|\lambda_a|, |\lambda_b|)$ avec $b = a + \frac{j}{2}$

— $g(\lambda_b, \lambda_a, \hat{u}) = (-1).1 - 2.\hat{u}.\lambda_a + \lambda_b$ avec $b = a - \frac{j}{2}$ et $\hat{u} = \hat{s}_{i-\frac{J}{2}}$

Chaque nœud du graphe reçoit $\frac{\boldsymbol{N}}{\boldsymbol{2}}$ LLRs de son père et lui retourne après complétion du calcul $\boldsymbol{N'}$ sommes partielles. Ainsi, la complexité calculatoire par nœud est divisée par deux à chaque fois que l'on descend d'un niveau. Le parallélisme de calcul permettant d'accélérer l'exécution du processus de décodage évolue ainsi durant le parcours de l'arbre.

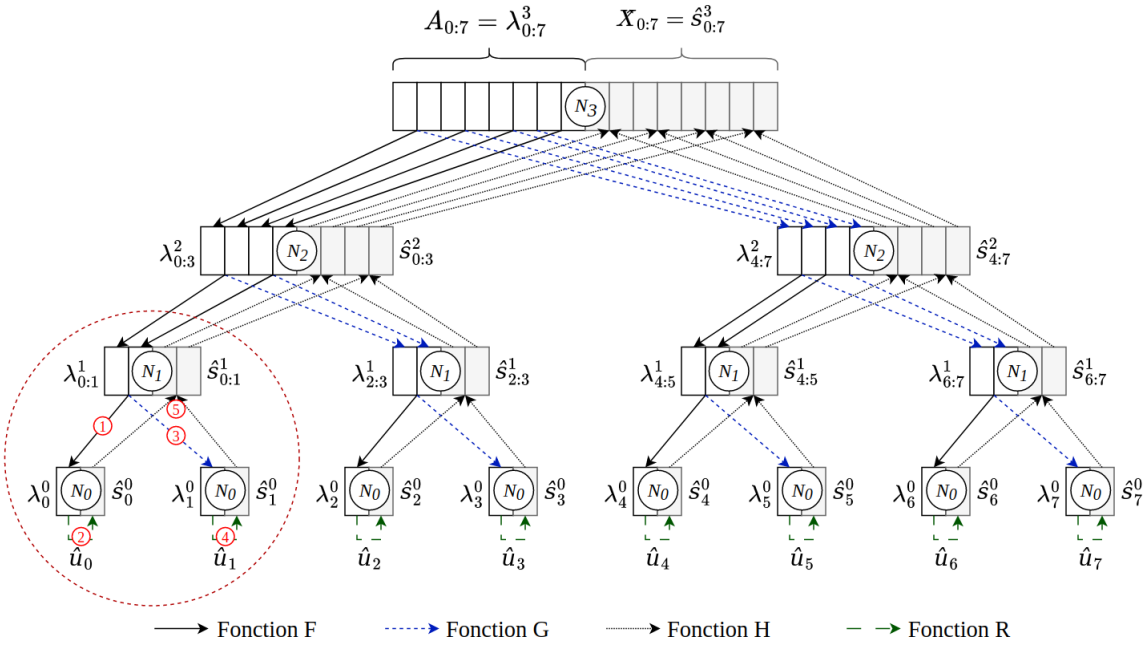

Figure 2.8 – Représentation sous forme d'arbre du processus de décodage d'un décodeur polaire SC N=8

Ainsi, l'algorithme de décodage SC est bien adapté pour les implémentations matérielles [91, 93, 94, 102-104] et logicielles [106-108], car il possède une faible complexité calculatoire qui évolue en $\mathcal{O}(\boldsymbol{N} \times log(\boldsymbol{N}))$. Cependant, le processus de décodage étant principalement séquentiel, il nécessite une traversée pré-ordonnée de l'arbre binaire [109] qui fait : (1) varier le niveau de parallélisme exploitable en fonction du niveau de l'arbre considéré et (2) empêche l'accélération de l'exécution des feuilles terminales même si plusieurs éléments de calcul sont disponibles. Afin de résoudre ce dernier point, des techniques d'élagage de l'arbre binaire ont été proposées [110]. L'algorithme SC-List [111], plus efficace du point de vue des performances de décodage, augmente fortement la complexité calculatoire à cause de la gestion d'une liste de **L** solutions probables de manière concurrente. Cela rend son implantation plus difficile tant d'un point de vue logiciel [112, 113] que matériel [111], car malgré des niveaux de parallélisation supplémentaires, des besoins de tri des données et de synchronisation des listes apparaissent.

Algorithm 5 Fonction de mise à jour du nœud $\mathcal{N}_j$

Nécessite : $\lambda^j_{a:a+J-1}$

Calcul $J/2$ fonctions f concurrentes :

1 : $\lambda_i^{j-1} = f(\lambda_i^j; \lambda^j_{i+\frac{j}{2}}), a \leq i \leq \frac{J}{2},$

Appel récursif du noeud fils gauche $\mathcal{N}_{j-1}$

2 : $\hat{s}^{j-1}_{a:a+\frac{J}{2}-1} = \mathcal{N}_j(\lambda^j_{a:a+J-1})$

Calcul $J/2$ fonctions g concurrentes :

3 : $\lambda_i^{j-1} = g(\lambda^j_{i-\frac{J}{2}} : \lambda_i^j : \hat{s}^{J-1}_{i-\frac{J}{2}}), a + \frac{j}{2} \leq i \leq J$

Appel récursif du noeud fils droit $\mathcal{N}_{j-1}$

4 : $\hat{s}^{j-1}_{a+\frac{J}{2}:a+J-1} = \mathcal{N}_j(\lambda^{j-1}_{a+\frac{J}{2}:a+J-1})$

Combine les sommes partielles des étapes 2 et 4 :

5 : $\hat{s}_i^j = \hat{s}_i^{j-1} \oplus \hat{s}^{j-1}_{i+\frac{J}{2}}, a \leq i < \frac{J}{2}$

6 : **Retourne** $\hat{s}_{a:a+J-1}$

2.4 Conclusion

Les CCE sont des éléments essentiels des systèmes de communications numériques actuels, car ils permettent d'améliorer fortement la fiabilité des liens de communication. Comme nous venons de le voir, quatre principales familles de CCE sont actuellement utilisées dans les standards de communication terrestres et spatiaux. Cette hétérogénéité provient des caractéristiques intrinsèques de chacune des familles, que cela soit en termes de complexité calculatoire, de performances de décodage, mais également de l'efficacité des implantations logicielles et matérielles de leurs algorithmes de décodage.
Même si différents travaux ont essayé d'identifier la meilleure famille de code [114-116], l'association de ces trois facteurs rend les comparaisons entre ces différentes familles impossibles en l'absence de tout contexte applicatif. Cette absence de consensus implique un travail de recherche et de développement conséquent pour proposer un large spectre de décodeurs CCEs efficaces, en fonction du contexte.

Chapitre 3

Stratégies d'implantation des décodeurs de CCE

Ce chapitre se focalise sur les architectures numériques et les systèmes utilisés pour intégrer ces différentes familles de CCE, sous contrainte temps réel dans des systèmes embarqués. Les différentes stratégies d'implantation de ces CCE sont également présentées. Ces notions permettent de positionner ces travaux de thèse relativement à l'état de l'art et de préciser les motivations dans la dernière partie de ce chapitre.

3.1 Introduction

Comme évoqué précédemment, les CCE sont présents dans la majorité des systèmes de communication et de stockage de l'information que nous utilisons. L'hétérogénéité des contextes applicatifs a abouti au développement et à la standardisation de nombreux codes différents.

L'implantation de décodeurs sous contrainte temps réel devient complexe pour les concepteurs, car différents paramètres sont à prendre en considération : complexité matérielle, débit, latence et/ou consommation énergétique. Afin d'adresser l'ensemble de ces besoins, différentes stratégies de parallélisation et d'implantation des décodeurs ont été imaginées

et déployées sur différentes technologies matérielles ASIC/FPGA. Par ailleurs, ces différentes stratégies ont été plus récemment investiguées sur des architectures programmables (CPU multicœurs et GPU). Lors de l'implantation d'un décodeur CCE, les diverses solutions proposées dans la littérature doivent être déclinées et optimisées afin de s'adapter aux paramètres algorithmiques du ou des codes sélectionnés et des contraintes applicatives aboutissant à un nombre important de solutions pour une seule famille de CCE. Deux types de paramètres contraignent fortement la conception des décodeurs :

Les paramètres algorithmiques – ce sont les caractéristiques intrinsèques du code telles que la taille **N** des trames à décoder ainsi que le rendement **R** du code. D'autres caractéristiques propres à chaque famille affectent aussi la conception des décodeurs, comme la structure de la matrice de parité pour un code LDPC ou encore les positions des bits gelés pour un code polaire. Ces paramètres varient généralement d'un standard ou d'un cas d'usage à l'autre.

Les paramètres architecturaux – ce sont les caractéristiques d'implantation. Il peut s'agir du format de quantification des données internes au décodeur, du facteur de parallélisation des calculs, du nombre d'itérations de décodage pour un algorithme itératif, etc. Ces paramètres, dérivés des contraintes du système à concevoir, peuvent, pour certains, être issus de phases de raffinement et de simulation successives. Ils ont un impact direct sur l'architecture à concevoir et, par conséquent, sur la complexité matérielle et énergétique du décodeur.

La multiplicité des cadres applicatifs nécessite donc de disposer d'un large éventail d'implantations dans l'espace des solutions. Par exemple, dans le standard 3GPP 5G [117], le standard couvre des besoins allant de décodeurs soumis à de faibles débits pour des réseaux de capteurs [118] à des décodeurs très hauts débits (quelques Gbit/s) pour des systèmes de vidéo transmission [119]. En plus de ces cas extrêmes, d'autres contextes applicatifs, tels que celui des voitures autonomes [120], nécessitent des décodeurs à très faible latence. Cette hétérogénéité s'étend à d'autres paramètres liés à la nature des systèmes : les contraintes énergétiques ne sont pas homogènes entre des systèmes de type IoT [121] et les infrastructures de type Cloud-RAN [67-69, 122] tout comme les besoins en termes de flexibilité et d'évolutivité des décodeurs intégrés dans ces systèmes.

En fonction de la nature des systèmes à concevoir, en plus des contraintes appliquées à ces derniers, différentes stratégies d'implantation sont actuellement mises en œuvre. Depuis quelques années, en parallèle de la conception d'architectures dédiées (ex. ASIC, FPGA), les implantations logicielles sur des cibles programmables (CPU multicœurs et

GPU) sont devenues viables. Cela a été rendu possible grâce à l'évolution des architectures programmables qui offrent un niveau de flexibilité élevé au détriment des performances énergétiques. L'évolution des stratégies d'implantation des décodeurs CCE ainsi que des méthodologies de conception associées sont traitées dans les sections suivantes.

3.2 Architectures dédiées

Historiquement, l'implantation de chaînes de communications numériques, et donc de décodeurs de CCE, nécessitait la conception d'architectures numériques dédiées afin de respecter les contraintes applicatives, cela tant en termes de débits que de consommation énergétique. Ces architectures dédiées étaient décrites au niveau RTL [79] dans des langages tels que le VHDL ou Verilog avant d'être synthétisées, placées et routées pour des technologies ASIC avant d'être produites en grande quantité dans des fonderies (figure 3.1). En procédant ainsi, les concepteurs s'assuraient une maîtrise totale du système implanté, leur permettant, en fonction des besoins, soit de maximiser le débit, de minimiser la consommation énergétique ou bien de trouver des compromis entre ces paramètres. Ce contrôle total sur l'implantation permettait aussi l'usage de techniques d'optimisations fines en lien avec les capacités de la technologie ASIC ciblée. Ainsi, de nombreux décodeurs CCE, toutes familles de codes confondues, ont été réalisés [123]. Malgré la complexité des méthodologies de conception et le temps nécessaire au développement et la production d'ASIC, cette approche est toujours plébiscitée pour les systèmes très contraints en termes de débits et/ou énergétiquement. Récemment, des décodeurs CCE qui atteignent des débits avoisinants 100 Gbps voire 1 Tbps ont notamment été décrits dans la littérature [124-126].

Le manque de flexibilité et d'évolutivité des architectures dédiées sur cible ASIC a motivé les concepteurs à évaluer et adopter d'autres types de technologies tels que les circuits FPGA. À l'origine, les circuits FPGA servaient à prototyper et à valider les architectures numériques avant leur intégration dans des ASIC. Cependant, leur faculté de reconfiguration permet une évolution des systèmes dans le temps (flexibilité et évolutivité) tout en évitant les délais de production imposés par le passage en fonderie. Cette transition a été rendue possible grâce à l'augmentation de leurs capacités d'intégration qui a permis la mise au point de décodeurs CCE atteignant des débits de plusieurs Gbps [61, 62, 127].

Malgré les gains substantiels en termes de temps de mise au point et de flexibilité, la conception d'architectures dédiées sur cibles ASIC ou FPGA nécessite de longs cycles de

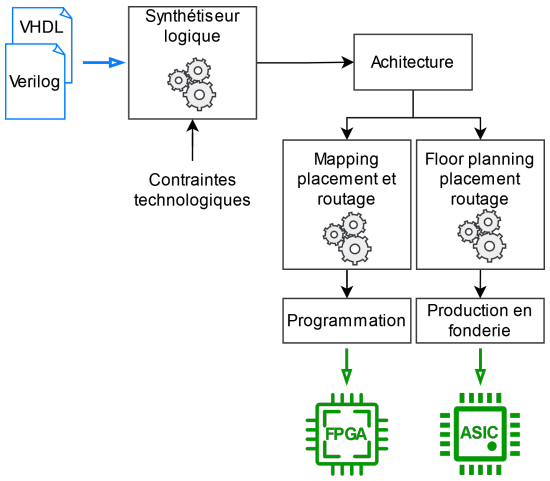

Figure 3.1 – Méthodologie de conception pour les architectures dédiées

développement. De plus, ces descriptions matérielles doivent être réécrites en fonction du débit ciblé, du format de quantification et de la taille du code (**N**). Cela implique des coûts récurrents liés aux modifications plus ou moins importantes à apporter aux architectures. Ces aspects limitent fortement l'exploration de l'espace des solutions.

Depuis une dizaine d'années, avec les progrès réalisés par les outils de synthèse de haut-niveau (HLS), dont l'arrivée est annoncée depuis des décennies [128-130], une simplification de l'exploration de l'espace des solutions et un raccourcissement des cycles de développement est possible. En effet, les outils de HLS industriels tels que Vitis HLS [131] ou Catapult-C [132] procurent des gains majeurs en termes de productivité. Ces outils permettent, comme cela est schématisé dans la figure 3.2, de concevoir automatiquement des architectures dédiées de niveau RTL à partir de descriptions comportementales exprimées dans des langages de haut-niveau tels que les langages C, C++ et SystemC. En plus de réduire le temps de conception d'architectures dédiées au niveau RTL, ils facilitent l'exploration de l'espace des solutions via l'utilisation d'annotations présentes dans le code comportemental et l'insertion de contraintes lors de la synthèse.

Les différents travaux de recherche menés depuis le milieu des années 2010 ont démontré la capacité de ces méthodologies à produire des architectures de qualité comparables à celles décrites manuellement au niveau RTL. Ce résultat est dû à l'amélioration des algorithmes de synthèse intégrés à ces outils, mais également à leur capacité à être intelligemment guidés par le concepteur. Différentes études traitant de l'implantation de décodeurs CCE pour les codes LDPC [133-135], les codes LDPC non binaires [136], les turbo codes [137] et les codes polaires [138] ont mis en évidence la maturité des méthodo-

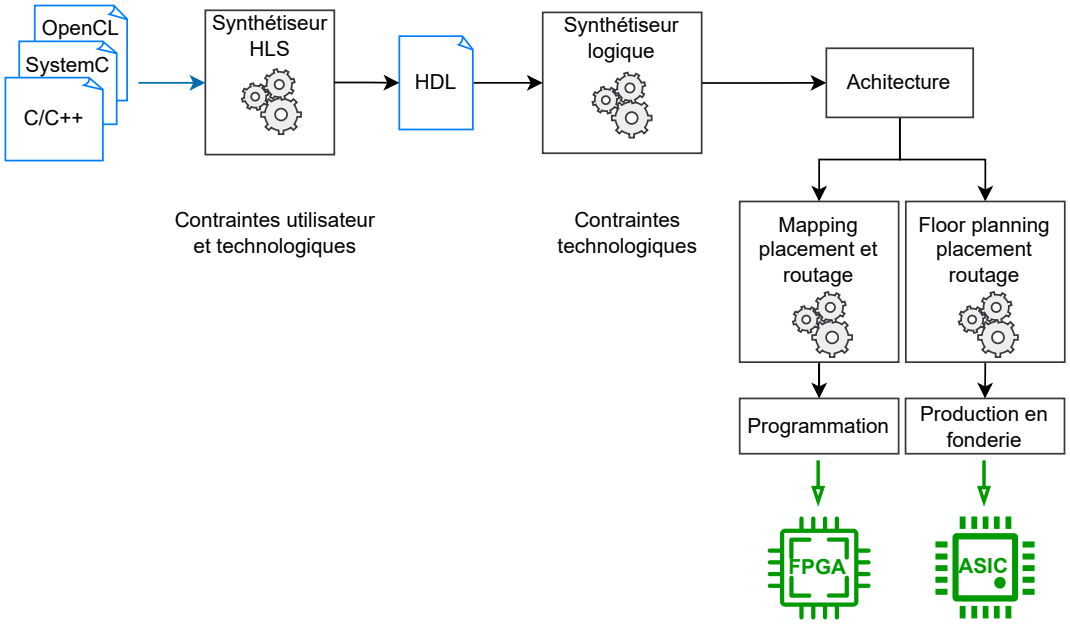

Figure 3.2 – Méthodologie de conception intégrant la synthèse de haut niveau

logies basées sur les outils HLS. Toutefois, ces études ont aussi mis en lumière l'impact de la qualité des modèles comportementaux et des directives de synthèse sur les niveaux de performance atteignables : pour atteindre des niveaux de performance équivalents à celui des architectures développées au niveau RTL, il est, par exemple, nécessaire de développer des modèles comportementaux intégrant des aspects architecturaux (ex. découpage en sous-tâches) [133-135, 138].

Ces méthodologies fondées sur les outils HLS permettent de considérablement réduire le temps de développement des décodeurs de CCE matériels. De plus, elles sont actuellement en phase avec l'écosystème du Cloud Computing. Cet écosystème est composé des servers distants, eux-mêmes constitués de plusieurs processeurs et des cartes accélératrices de type FPGA. Ainsi, l'application de méthodologies HLS dans cet écosystème permet de rapidement déployer des infrastructures de communications numériques [139-142]. Cependant, il reste difficile de produire des architectures matérielles génériques et flexibles capables, par exemple, de gérer efficacement les multiples variantes de codes LDPC présentes dans le standard 5G et/ou de supporter plusieurs familles de CCE.

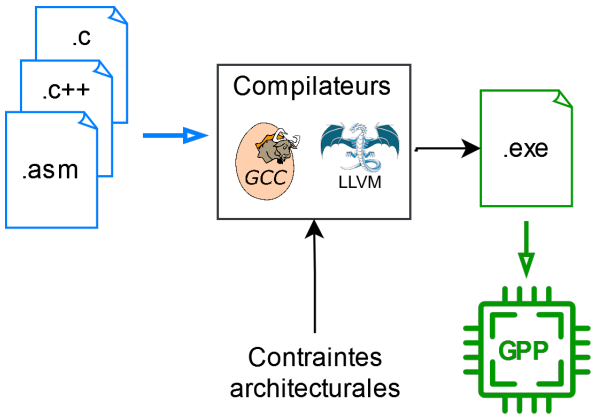

Figure 3.3 – Flot de conception pour un processeur généraliste

3.3 Architectures programmables

Afin d'apporter des niveaux de flexibilité et d'évolutivité supérieurs et répondre aux besoins énoncés par le concept de Software Defined Radio (SDR) [65], au début des années 2010, de nombreuses études se sont focalisées sur le développement de chaînes de communications numériques purement logicielles. En effet, l'exécution de programmes sur des architectures programmables de type CPU ou GPU offre, d'un point de vue système, à la fois des niveaux de flexibilité et d'évolutivité élevés ainsi qu'une simplicité de mise en œuvre face à la mise au point de solutions ASIC/FPGA.

Pour déployer tout ou partie d'un système sur une architecture programmable, il est nécessaire de décrire le comportement des algorithmes à exécuter dans un langage de programmation tel que le C/C++. Ces différentes descriptions comportementales sont dans un second temps analysées, optimisées et traduites en code machine par un compilateur tel que schématisé dans la figure 3.3. Le compilateur (ex. GCC,Clang/LLVM), durant les phases d'optimisation et de génération du code machine, est contraint par le jeu d'instructions du processeur ciblé, mais aussi par la qualité de la description logicielle. Dans le but d'améliorer les performances des applicatifs, certaines parties des descriptions comportementales peuvent être décrites en assembleur afin de profiter au mieux des capacités spécifiques des processeurs (*prefetch* des caches, instructions spécialisées, etc.). Même si ce travail de description et d'optimisation est un processus qui peut être long et complexe, ces durées sont incomparables avec celles nécessaires pour la conception d'architectures matérielles. Toutefois, au début des années 2000, les puissances de calcul offertes par les architectures programmables (CPU mono-cœur et DSP) étaient bien inférieures à celles nécessaires pour l'exécution temps réel des systèmes de communication. Cet état de fait

restreignait l'usage des CPU à l'étude des performances des systèmes de communications numériques à l'aide de simulations de type Monte Carlo.

L'apparition d'unités de calcul vectoriel telles que les extensions MMX/SSE[1] dans les cœurs de processeur généraliste et dans les DSP relança l'engouement pour ce type de solution [143-145]. Cependant, ce n'est qu'avec l'arrivée des premières architectures massivement parallèles telles que l'architecture CELL [146] produite en 2006 par Sony pour la PlayStation 3 que le concept de radio-logicielle devint envisageable [147-149]. En effet, le processeur CELL était composé d'un cœur PowerPC traditionnel qui était couplé à 8 cœurs spécialisés. Chacun de ces cœurs possédait une unité vectorielle (Single Instruction Mulitple Data, SIMD) capable d'exécuter 2 instructions par cycle d'horloge, chaque instruction traitant Q données (avec Q données $\times$largeur des données = 128 bits).

Les premiers travaux universitaires ciblant cette architecture [147-149] se sont focalisés sur la proposition de stratégies de parallélisation des algorithmes de décodage des CCE en phase avec les spécificités architecturales et le jeu d'instructions vectoriel à disposition. Ces études se sont concentrées sur des algorithmes de décodage CCE « naïfs » offrant un parallélisme de calcul régulier facilement accessible. Par exemple, dans le cadre du décodage des codes LDPC, l'ordonnancement par vague (flooding schedule) était préféré aux ordonnancements par couches (layered schedule) à cause de sa régularité et du fort parallélisme de calcul exposé. Couplées à l'utilisation de données flottantes, ces premières descriptions logicielles de décodeurs LDPC appliquées au standard WiMAX exploitant l'intégralité du processeur CELL permettaient d'atteindre des débits de quelques dizaines de Mbps.

Ces travaux permirent de démontrer la viabilité du concept de Software-Defined Radio (SDR), malgré des niveaux de performances modestes en termes de débit face aux solutions ASIC/FPGA. Ils ont ensuite été étendus aux architectures multi-cœurs et many-cœurs (GPU) dont la conception et l'usage s'étaient généralisés. Les architectures de processeurs graphiques (GP-GPU) offraient de nouvelles opportunités grâce à (1) leur paradigme de programmation SIMT (Single Instruction Multiple Threads) simplifiant grandement la description des traitements à réaliser et (2) un nombre de cœurs de calcul « élémentaires » pouvant atteindre plusieurs milliers d'unités [76][13]. Ce grand nombre de cœurs de calcul associé à des fréquences de fonctionnement variant de 1 à 2 GHz semblait un choix pertinent pour l'implantation de systèmes de communications numériques, et plus

1. Extensions multimédia du jeu d'instructions des processeurs INTEL et AMD, respectivement SIMD 64 bits et 128 bits.

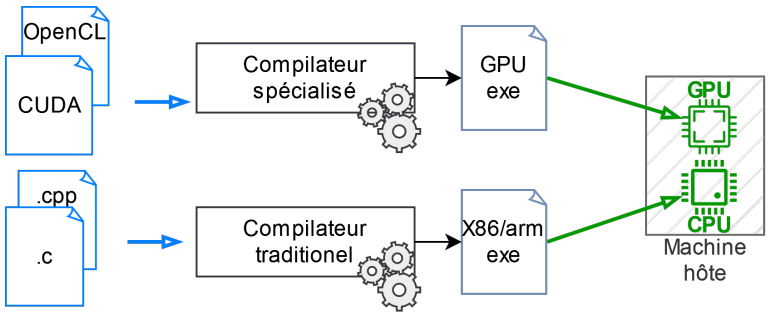

Figure 3.4 – Méthodologie de conception pour une architecture multi/many cœurs

particulièrement de décodeurs de CCE.

Différents langages de programmation ont été développés pour permettre de programmer ces circuits, parmi les plus connus, peuvent être cités OpenCL [150] et CUDA [151]. Ils permettent à la fois de décrire simplement le parallélisme de calcul et d'avoir un niveau de granularité suffisant pour adapter au mieux ce parallélisme à la plateforme d'exécution. Comme cela est illustré dans la figure 3.4, le développeur doit modéliser le comportement des fonctions à accélérer à l'aide d'un langage dédié, mais il doit également écrire le code logiciel permettant au système hôte d'effectuer les transferts de données vers/depuis le GPU et d'initier les traitements. Toutefois, même si des débits supérieurs à quelques Mbps peuvent être obtenus assez aisément [152-154], l'identification de bonnes stratégies de parallélisation associée à la conception de structures de données adaptées est nécessaire pour tirer parfaitement parti de ces architectures complexes. Cette difficulté transparaît dans de nombreux travaux menés autour du décodage des codes LDPC [63, 155]. Les premières études ciblant des GPU [154] ont mis en exergue la capacité de ces plateformes à fournir des débits de décodage supérieurs à 100 Mbps, compatibles avec les attendus de standards tels que DVB-S2 [42] et WiMAX [7]. Cependant, la nature même des algorithmes de décodage de CCE possédant un parallélisme de calcul limité (LDPC, turbo codes), irrégulier (codes polaires) ou bien la nécessité de réaliser des accès non contigus à la mémoire (LDPC et turbo codes) limite l'efficacité de ce type de solution par rapport aux capacités théoriques des architectures. Afin de saturer les centaines ou les milliers de cœurs de calcul disponibles au sein des GPU et apporter de la régularité au niveau des accès-mémoire, une solution étudiée a consisté à décoder un nombre important de trames en parallèle [156]. Cette stratégie améliore fortement le débit, mais impacte aussi négativement la latence de traitement, rendant de facto ces solutions incompatibles avec les standards de communication mobile tels que la 4G LTE [3] et la 5G 3GPP [8].

Même si les débits sont compatibles avec les besoins applicatifs, les architectures de GPU possèdent des inconvénients rédhibitoires pour bon nombre de systèmes communicants :

1. leur consommation de puissance qui dans les travaux publiés dépasse 100∼200 Watts [156],
2. le pouvoir de correction dégradé car les décodeurs exécutent 2× à ∼ 4× moins d'itérations à iso-débit que les solutions matérielles sur circuit FPGA [155].

L'ensemble de ces points explique pourquoi, à l'heure actuelle, en dehors de travaux universitaires, les GPUs ne sont pas utilisés dans les solutions industrielles [122, 157] et restent cantonnés à la simulation et l'étude de systèmes [158].

En parallèle de ces travaux relatifs autour des GPU, des études sur les architectures multicœurs ont démontré que l'évolution de ces dernières (nombre de cœurs physiques, extensions des unités SIMD) permettaient l'obtention de solutions plus efficaces. Ils peuvent offrir des débits supérieurs à 1 Gbps et des latences inférieures à 100 µs avec consommations énergétiques maîtrisées. Plusieurs caractéristiques architecturales expliquent ces résultats :

— Les jeux d'instruction SIMD présents dans les cœurs ARM et INTEL excellent dans l'exécution d'opérations entières (8 bits), ils peuvent traiter de 16 à 64 données en parallèle par cycles d'horloge.

— Les décodeurs de CCE manipulent un nombre de données limité, ce qui leur permet d'exploiter efficacement les caches de niveau L1 et L2. Cela minimise le temps d'accès aux données durant le processus de décodage.

— Les processeurs multicœurs actuels ont des fréquences de fonctionnement 2× à 5× supérieures à celles des GPU. De plus, ce sont des cœurs superscalaires pouvant réaliser plusieurs instructions par cycle d'horloge lorsque les dépendances de données et la disponibilité des unités fonctionnelles l'autorisent.

Les premières études réalisées sur des architectures x86 fournissaient des résultats bien inférieurs à ceux obtenus sur des architectures GPU [159]. Ces résultats étaient liés à des choix algorithmiques peu pertinents et des stratégies de parallélisation peu efficaces, avec par exemple, un décodage par inondation sur des données flottantes 32 bits pour le décodage des codes LDPC [160]. Une seconde vague de travaux s'inspirant des algorithmes de décodage utilisés dans les architectures de décodeurs matériels a changé la donne. Par exemple, les résultats présentés dans [63] pour le décodage de codes LDPC, obtenus sur

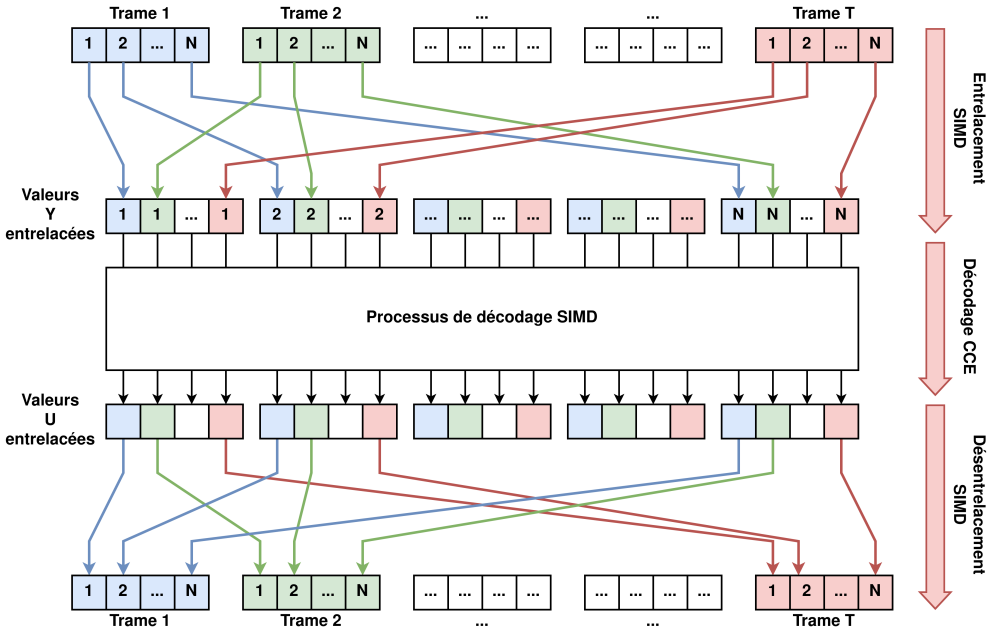

Figure 3.5 – Procédure de décodage SIMD

une architecture INTEL multicœurs issue d'un ordinateur portable, étaient de 3 à 100 fois supérieurs en termes de débits par rapport à l'état de l'art sur GPU.

Ces résultats ont motivé l'extension de ces études se focalisant sur les architectures multicœurs aux autres familles de CCE : les turbo codes [161] les codes polaires [162, 163] et les codes LDPC-NB [164]. Pour l'ensemble de ces familles, les solutions multicœurs ont démontré des niveaux de performance en termes de débit, de latence et d'énergie plus intéressants que ceux des solutions fondées sur l'utilisation de GPU. Afin de saturer les unités de calculs SIMD et obtenir une régularité des accès à la mémoire, ces décodeurs logiciels, contrairement aux décodeurs matériels qui se focalisent sur le parallélisme présent dans le processus de décodage d'une trame (intra-trame), utilisent le parallélisme présent lors du décodage de $\mathbf{Q}$ trames homogènes (inter-trames). Ainsi, chaque cœur décode $\mathbf{P}$ trames en parallèle avec 8 bits $\times\mathbf{Q}$ égal à la largeur de l'unité SIMD. La régularité des accès mémoire est obtenue en entrelaçant les trames en amont et en aval des états de décodage comme que schématisé dans la figure 3.5.

Ces étapes d'entrelacement, qui peuvent être accélérées grâce aux instructions SIMD des processeurs INTEL et ARM, consomment un temps d'exécution négligeable au regard du temps de décodage de CCE [162]. Au niveau système, chacun des $\mathbf{P}$ cœurs physiques peut décoder $\mathbf{Q}$ trames. Ces travaux de recherche ont donné naissance à la bibliothèque

aff3ct [165] qui a servi à concevoir, par exemple, un récepteur SDR pour le standard DVB-S2 [166].

La stratégie de parallélisation inter-trames produit cependant des effets indésirables qui limitent la scalabilité de l'approche et l'intérêt réel des décodeurs :

1. l'augmentation linéaire de l'empreinte mémoire avec $\mathbf{Q}$ qui provoque rapidement une saturation des caches L1/L2 lorsque $\boldsymbol{Q} \in \{16, 32, 64\}$ même pour des trames de tailles modérées. À l'exécution, les performances sont limitées par la bande passante mémoire. Ainsi le facteur d'accélération plafonne et atteint une asymptote horizontale inférieure à $\boldsymbol{P} \times \boldsymbol{Q}$ lorsque l'empreinte mémoire dépasse la taille des caches L2/L3. Au niveau du système, cette utilisation de la bande passante mémoire peut aussi impacter les performances des autres traitements présents dans le récepteur.
2. des latences de décodage (100 µs $\sim$ 10 ms) bien plus importantes que celles issues de décodeurs matériels se focalisant sur une exploitation du parallélisme intra-trame.

L'intégration des contraintes fortes sur la latence de décodage a engendré le développement de nouveaux schémas de parallélisation [161, 163, 164, 167, 168] pour garantir le respect des valeurs attendues dans les 4G LTE [3] et 5G [8]. Ces nouvelles études ont permis de maintenir des débits de décodage élevés tout en réduisant les latences de décodage à des valeurs compatibles avec les contraintes des standards ($\leq$ 1 ms). Les implantations logicielles ont été calquées sur le comportement des architectures matérielles, tant au niveau des stratégies de parallélisation des calculs que du stockage des données. La scalabilité de l'approche est ainsi améliorée, car chaque cœur utilise son unité SIMD pour accélérer le décodage d'une seule et unique trame (parallélisation intra-trame). Cette approche diminue la pression sur les caches L2/L3 et réduit a minima d'un facteur $\mathbf{Q}$ les besoins en termes de bande passante mémoire. Comme cela est mis en évidence dans [161], les débits de ces décodeurs CCE évoluent donc linéairement avec $\mathbf{P}$ et $\mathbf{Q}$.

Malgré les progrès réalisés au niveau des architectures internes des multicœurs SIMD pour augmenter leurs performances et/ou réduire leur consommation énergétique, ces dernières restent inefficaces ($\sim 10\times$ à $100\times$) lorsqu'elles sont comparées à des architectures dédiées sur FPGA [64]. Ce décalage est en partie lié à l'inefficacité du jeu d'instructions des processeurs multicœurs. En effet, leurs instructions sont génériques et peu adaptées aux opérations arithmétiques et logiques mises en œuvre dans les algorithmes de décodage des CCE. Ainsi, afin d'implanter les opérations élémentaires des algorithmes de décodage CCE, et ce, contrairement aux architectures dédiées où ces opérations peuvent être réali-

sées en un cycle d'horloge, des dizaines d'instructions peuvent être nécessaires [63].

Afin de pallier ce facteur d'inefficacité lié à la nature intrinsèque des architectures programmables, tout en maintenant un niveau de flexibilité élevé, d'autres approches architecturales ont vu le jour. Ces dernières proposent de concevoir des cœurs de processeurs spécialisés à un domaine applicatif dans le but de rapprocher les niveaux de performance des deux mondes.

3.4 Architectures programmables spécifiques

L'élaboration d'architectures dédiées de décodeurs de CCE nécessite des compétences importantes dans la conception au niveau RTL et implique des cycles de développement longs. À l'opposé, l'utilisation d'architectures programmables de type processeur superscalaire ou VLIW est plus aisée et offre de la flexibilité aux concepteurs de systèmes. Cependant, leurs unités de calcul génériques ainsi que leurs jeux d'instructions limitent l'efficacité des implantations logicielles. Pour pallier cette limitation, une solution proposée au début des années 2000 [169] et employée depuis consiste à développer des architectures de processeur spécialisées (Application Specific Instruction-set Processor – ASIP) pour un ou des domaines applicatifs. Ces cœurs de processeur, décrits à l'origine au niveau RTL, pouvaient être prototypés et déployés sur circuits FPGA ou implantés en technologie ASIC.

Les ASIP sont donc des processeurs qui intègrent des unités de calcul dédiées à un domaine applicatif. Des instructions assembleur dédiées permettent au développeur d'exploiter ces fonctionnalités au niveau logiciel. Afin de développer ces cœurs de processeurs ASIP, différentes approches méthodologiques ont été proposées dans la littérature. La première stratégie, schématisée dans la figure 3.6, consiste à concevoir intégralement l'architecture pour un domaine applicatif donné [170, 171]. La paramétrisation du processeur, de ses bus de données, de ses bancs de registres et du jeu d'instructions est réalisée de manière spécifique par rapport aux besoins applicatifs. Les propriétés architecturales du processeur sont alors formalisées à l'aide d'un langage dédié (Domain Specific Language – DSL) [172-174] ou de manière graphique [175]. Ces spécifications, une fois traitées par une suite d'outils, tels que CoWare Processor Designer [176, 177], permettent la génération de :

(a) la description architecturale du cœur de processeur au niveau RTL ;

(b) l'environnement de développement logiciel (compilateur, debugger, etc.) ;

(c) un ou des simulateurs TML/CABA couplés à des outils d'analyse.

Ce type d'approche a permis la conception de décodeurs de CCE dédiés au décodage des turbo codes [178], des codes LDPC [179] et plus récemment des codes polaires [175]. Cette garde méthodologie a permis l'obtention de niveaux de performance élevés grâce au dimensionnement au plus juste de l'architecture matérielle, cependant, elle implique des cycles de développement relativement longs.

Afin de réduire les cycles de développement et augmenter la flexibilité des architectures ASIP, une seconde approche méthodologique a émergé. Elle est basée sur la conception ou l'adaptation de cœurs de processeur existants. Cette adaptation est basée sur l'analyse logicielle des programmes à exécuter ou bien de directement depuis les fichiers binaires [180, 181]. Ces approches plus génériques et fortement automatisées, comme cela est représenté dans la figure 3.6, tentent d'extraire des motifs d'instructions récurrents dont la factorisation sous la forme d'instructions spécialisées pourrait être bénéfique [182]. Les instructions ainsi identifiées sont ensuite ajoutées à l'architecture RTL du processeur considéré, [183]. En fonction des flots de conception, la chaîne de compilation peut être mise à jour, tout comme le code logiciel décrivant l'application. Ces approches permettent aux concepteurs de facilement améliorer les caractéristiques temporelles et énergétiques de leurs systèmes. Cependant, la complexité des algorithmes de détection et de sélection des motifs nécessite l'utilisation d'heuristiques pour adresser des applications réelles. Ces heuristiques associées à l'absence de connaissance a priori du domaine applicatif abouti à des niveaux de performances limités vis-à-vis d'ASIP spécifiés « manuellement ».

Une troisième approche méthodologique basée à la fois sur la connaissance du domaine applicatif et sur la réutilisation de cœurs de processeur existants vise à offrir un compromis pertinent entre les performances et le temps de conception. Pour améliorer les caractéristiques des cœurs de processeurs existants, l'approche consiste à enrichir leurs jeux d'instructions à partir d'extensions liées à un ou des domaines applicatifs [184]. Cette stratégie, basée sur les compétences du concepteur en charge de développer les extensions, a bénéficié de la démocratisation des processeurs softcore [185]. Ces derniers, tels que les cœurs industriels NIOS II [183] ou Xtensa [183, 186-188] permettaient aux concepteurs d'intégrer aisément leurs propres extensions. L'avantage principal de cette approche décrite dans la figure 3.6 réside dans le gain de temps lié à la réutilisation des architectures RTL et de leurs écosystèmes (compilateurs, debuggers, bibliothèques logicielles, etc.). Malgré la disponibilité de frameworks plus ou moins automatisés [182, 183, 186], les capacités d'adaptation de ces approches étaient à l'origine limitées à cause de la nature propriétaire

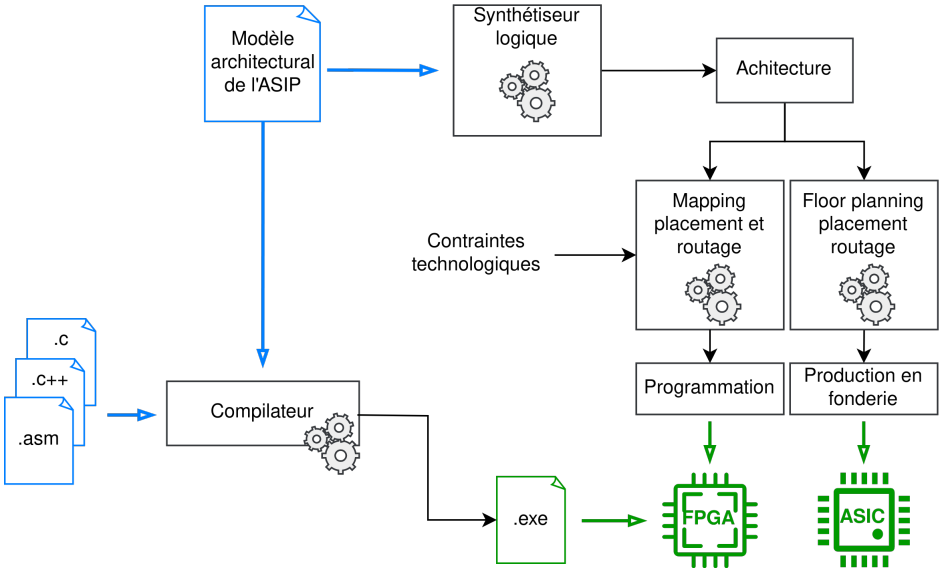

Figure 3.6 – Méthodologie de conception pour ASIP

des architectures softcore et de leurs codes RTL.

L'initiative RISC-V qui vise à proposer un jeu d'instructions open-source permettant la conception de cœurs de processeurs compatibles et qui a débuté en 2010 sous l'impulsion Berkeley, a ouvert une nouvelle voie et a stimulé cet axe de recherche [184]. En effet, de nombreux cœurs de processeurs compatibles avec ISA RISC-V ont été publiés sous diverses licences open-source [189, 190]. Ainsi, ces dernières années, de nombreux travaux de recherche ont été réalisés dans le domaine afin d'identifier et de proposer des extensions au jeu d'instructions des processeurs RISC-V [189] pour en améliorer l'efficience dans les domaines du traitement du signal, de la cryptographie [191], de la sécurité [192] et de l'intelligence artificielle (IA) [193]. Dans ces travaux, l'expertise des concepteurs a été mise à profit afin d'identifier les séquences d'instructions à intégrer sous la forme de nouvelles instructions. Ces études qui mesurent les avantages (accélération, consommation énergétique) et les inconvénients (coût, silicium, impact sur la fréquence) se basent sur les travaux réalisés dans le cadre de la conception d'architectures dédiées afin d'identifier les formats de données et patterns efficaces. Cependant, cette tâche n'est pas aisée, compte tenu des limitations imposées par le jeu d'instructions RISC-V. Par exemple, les instructions RV32I classiques sont contraintes à ne posséder que 2 opérandes en entrée et à produire qu'une seule donnée en sortie.

Cible	Technologie	Performances	Latence	Consommation énergétique	flexibilité	Coût	Effort de mise en œuvre
GPP	ASIC	+	+	++	++	−	−
SoftCore	FPGA	+	+	++	++	−	−
GPU	ASIC	++	++	+	++	+	+
ASIP	FPGA	++	+	+	+	+	+
	ASIC	++	+	+	+	+	+
Dédiée	ASIC	+ + ++	−−	−		++	+ + +
	FPGA	+ + ++	−−	−		++	+ + +

Table 3.1 – Caractérisation des différentes méthodologies de conception de systèmes embarqués

3.5 Conclusions

Dans les sous-sections précédentes, nous avons synthétisé les diverses solutions architecturales à disposition des concepteurs pour déployer tout ou partie des systèmes de communications numériques actuels. Ces solutions, en raison de leurs caractéristiques intrinsèques et des méthodologies associées, se divisent principalement en deux catégories :

— **Les architectures dédiées -** elles sont conçues spécifiquement pour un applicatif ou un domaine applicatif, elles fournissent de fait des niveaux de performance élevés au détriment de la flexibilité et du temps de conception.

— **Les architectures programmables -** elles sont génériques et flexibles par construction. Elles permettent d'adresser rapidement une large gamme de domaines applicatifs, au détriment des performances (complexité, énergie).

Les avantages et inconvénients de ces approches, ainsi que des solutions intermédiaires, sont résumées dans 3.1. Cette synthèse permet de positionner ces différentes solutions accessibles aux concepteurs de systèmes de communications numériques en fonction de critères tels que la flexibilité, l'efficacité énergétique, le temps de conception, etc.

L'ensemble des approches présentées dans le tableau 3.1 permet actuellement de concevoir des systèmes embarqués communicants adaptés aux besoins des applications de type Software-defined radio (SDR). Toutefois, en fonction de leurs caractéristiques intrinsèques, elles ne sont pas toutes interchangeables en fonction des domaines applicatifs (IoT, Cloud-RAN, etc).

Cependant, depuis plusieurs années, on observe une augmentation forte de l'utilisation de solutions purement logicielles dans la conception d'infrastructures de systèmes de communication. Cela s'explique comme cela a été évoqué par la flexibilité et l'évolutivité de ces solutions ainsi que leur passage à l'échelle (scalabilité). Néanmoins, malgré ces

avantages, ces solutions basées sur l'utilisation de multicœurs SIMD ou de GPU restent en retrait lorsque les systèmes sont contraints énergétiquement. Cette sous-optimalité est principalement liée à l'inadéquation du jeu d'instructions de ces architectures programmables vis-à-vis des besoins applicatifs. Des opérations simples à réaliser matériellement peuvent ainsi nécessiter de longues séquences d'instructions élémentaires, annulant de facto le gain apporté par leurs fréquences de fonctionnement élevées.

Afin de modifier cet état de fait, nous avons décidé dans le cadre de cette thèse de nous intéresser à la proposition d'instructions adaptées au décodage des CCE usuels. Ces instructions, pouvant être intégrées dans les ISA usuelles, visent à améliorer le support des algorithmes de décodage des CCE sur les architectures programmables. Les chapitres suivants présentent diverses extensions qui permettront d'améliorer le support des CCE usuels, mais aussi de supporter différents schémas de parallélisation dédiés à l'augmentation du débit de communication ou à la réduction de la latence.

Chapitre 4

Méthodologie de conception et d'implantation d'extensions

Ce chapitre vise à d'introduire les travaux effectués afin de proposer des jeux d'instruction spécialisés dédiés au décodage de codes correcteurs d'erreurs (CCE).

Dans un premier temps, le flot d'analyse et d'exploration mis en œuvre est introduit, ainsi que les contraintes architecturales retenues. L'objectif est la mise en place d'une approche permettant d'extraire les instructions ou blocs d'instructions qui affectent les performances des décodeurs de CCE en termes de temps d'exécution sur des cibles de type cœur de processeur.

À titre d'exemple, il est ensuite présenté quelques sous-parties des codes logiciels pour décodeurs les plus performants (en termes de débit) de l'état de l'art, pour les différentes familles de codes correcteurs, afin d'illustrer certains des motifs d'instructions sélectionnés.

La pertinence du flot d'exploration et les nouvelles instructions opérant sur des données scalaires sont ensuite validées par des expérimentations sur des circuits FPGA. Ces dernières sont réalisées grâce à l'enrichissement de cœurs de processeurs *open-source* de type RISC-V. L'analyse des résultats obtenus permet d'évaluer l'apport des extensions proposées sur le temps d'exécution des algorithmes de décodage considérés, mais également leurs impacts sur l'augmentation de la complexité matérielle.

4.1 Introduction

Cette première partie du chapitre a pour but de présenter le flot de conception utilisé dans le cadre de cette thèse. En ce sens, est introduit, pour chacun des algorithmes de décodeur de CCE considérés dans cette étude, les instructions ou séquences d'instructions ayant des impacts négatifs sur leurs performances générales. La sélection des codes logiciels C/C++ retenus, est effectuée sur la base de leur capacité à être implantés efficacement sur des cibles de type processeur généralistes. Les algorithmes retenus constituent, à ce jour, l'état de l'art pour les cibles considérées et leurs performances ont été validées par des simulations BER/FER.

L'objectif est de déterminer, au cœur des algorithmes de décodage, les instructions ou groupes d'instructions récurrents les moins efficaces/pertinentes sur architectures programmables et qui puissent avoir une implantation matérielle *simple*. La sélection de ces motifs d'instructions est basée sur les critères suivants :

— les instructions sélectionnées doivent être compatibles avec la plupart des jeux d'instructions standards ; *i.e.* deux registres d'entrée ***S***ource *rS1 et rS2* et un registre de sortie ***D***estination *rD* ;

— les nouvelles instructions doivent pouvoir s'exécuter sur un seul cycle d'horloge, sans provoquer de rupture de pipeline ;

— lorsque possible, les données internes sont codées 8 bits signés.

Ce dernier point est retenu, car il présente le meilleur compromis entre complexité d'implantation, indépendamment de la cible, et les performances de décodage.

Pour mettre en œuvre cette approche, un flot de conception a été mis au point et schématisé dans la figure 4.1. Il est important de rappeler que les codes logiciels décrivant le processus de décodage des CCE ont été optimisés afin d'exposer les séquences d'instructions qui peuvent être réduites à une seule.

4.2 Descriptif du flot de conception

Dans un premier temps (*cf.* figure 4.1), les différents codes logiciels (*a.k.a. kernels (*❶ dans la figure) sont exécutés sur les cibles sélectionnées (❼). Pour toutes les familles de CCE étudiées, nous sommes partis des meilleurs algorithmes de l'état de l'art, re-codés pour un contexte embarqué. Cette première étape permet d'extraire les instructions les plus onéreuses en temps d'exécution et qui répondent aux critères définis précédemment (❷). Cela a permis d'établir pour chaque famille de CCE une liste d'instructions à intégrer (❸).

L'ensemble de ces instructions ont, par la suite, été décrites au niveau RTL en *Verilog* ou *System-Verilog* (❹). Ensuite, leur ajout dans l'ISA des cœurs RISC-V (❼) [194] à été effectué. Naturellement, afin que les kernels décrits en langage C/C++ puissent exploiter ces nouvelles instructions (❸), il a été nécessaire de les adapter en remplaçant les sections de codes initiales par des invocations en assembleur à ces nouvelles fonctionnalités (❻). Il a été également primordial de les intégrer dans les chaînes de compilation des cœurs ciblés via GCC/Binutils (❺) pour permettre la génération de programmes exécutables. Enfin, une dernière étape (❽) a permis de valider les performances des kernels modifiés sur ces cœurs étendus, grâce à des tests réalisés en simulation ou par prototypage sur des circuits FPGA (Digilent Nexys 4 et Digilent Genesys 2). Ce sont donc ces versions incluant notre kit d'instructions qui sont alors utilisées pour les études expérimentales présentées dans ce chapitre.

La figure 4.2 montre, sur l'exemple d'un cœur de processeur à l'architecture simplifiée, mais proche d'un RISC-V CV32 [194], les principales parties concernées par l'ajout de nos extensions d'instructions. Ces nouvelles instructions proposées sont encodées selon le format de type R présenté dans 4.1, qui permet un mécanisme d'échange de données registres vers registres, relativement simple et direct. Ce type d'encodage mémorise l'*Op-Code* sur les 7 bits de poids faibles du registre d'instruction. Les adresses de registres sont passées dans les champs *rS1,rS2* pour les sources et *rD* pour la destination. Les champs *funct7* et *funct3* sont utilisés pour identifier l'instruction.

Bien que d'autres formats d'instructions existent dans la spécification RISC-V, le format de type R est suffisant pour enrichir le jeu d'instructions.

On constate ainsi qu'après l'étape de *Fetch* de l'instruction, l'étape de *Decoder* doit être mise à jour pour reconnaître les nouvelles instructions et ainsi générer des signaux de contrôle adaptés. De même, une unité de calcul spécifique à notre kit d'instructions doit

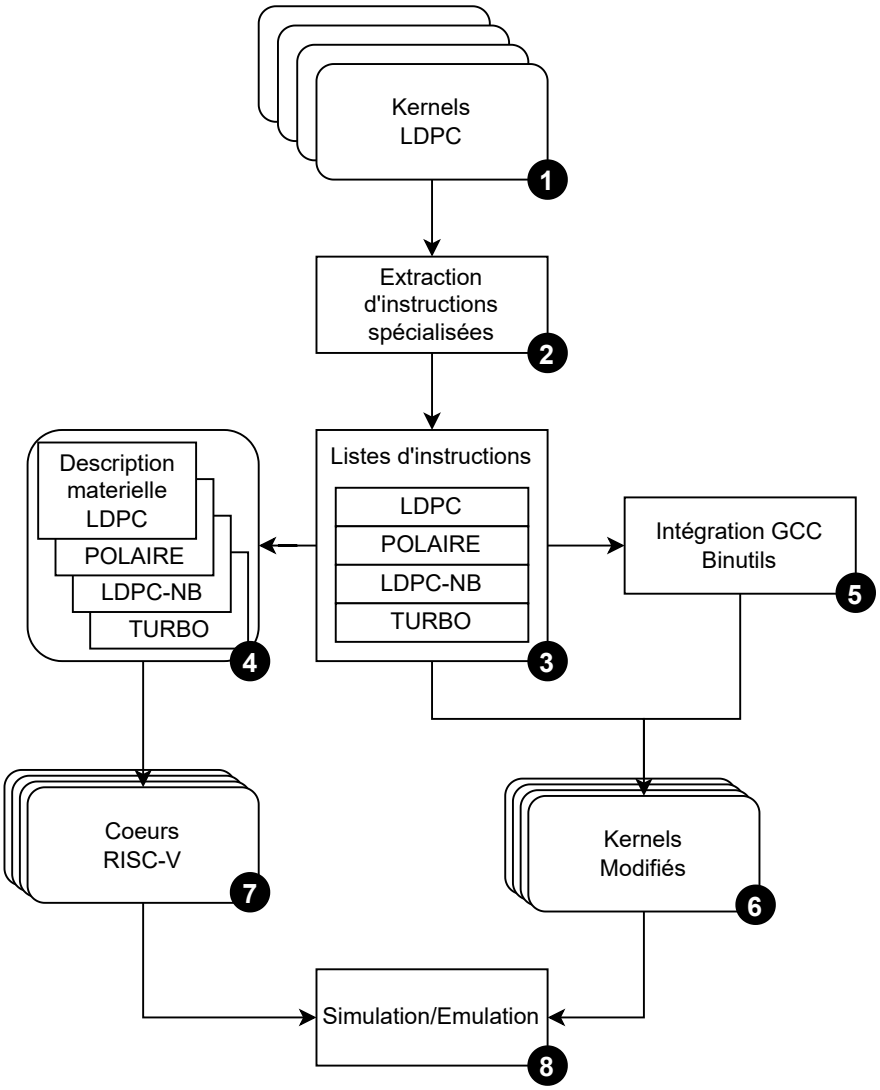

Figure 4.1 – Flot de conception global intégrant l'analyse et la sélection des instructions, l'intégration dans le flot de compilation et la mesure des performances.

Champ	**Bits**
Opcode	[6:0]
Registre destination	[11:7]
Fonction 3	[14:12]
Registre source 1	[19:15]
Registre source 2	[24:20]
Fonction 7	[31:25]

Table 4.1 – Format 32 bits standardisé pour les instructions de type R (2 registres d'entrées) dans la spécification RISC-V.

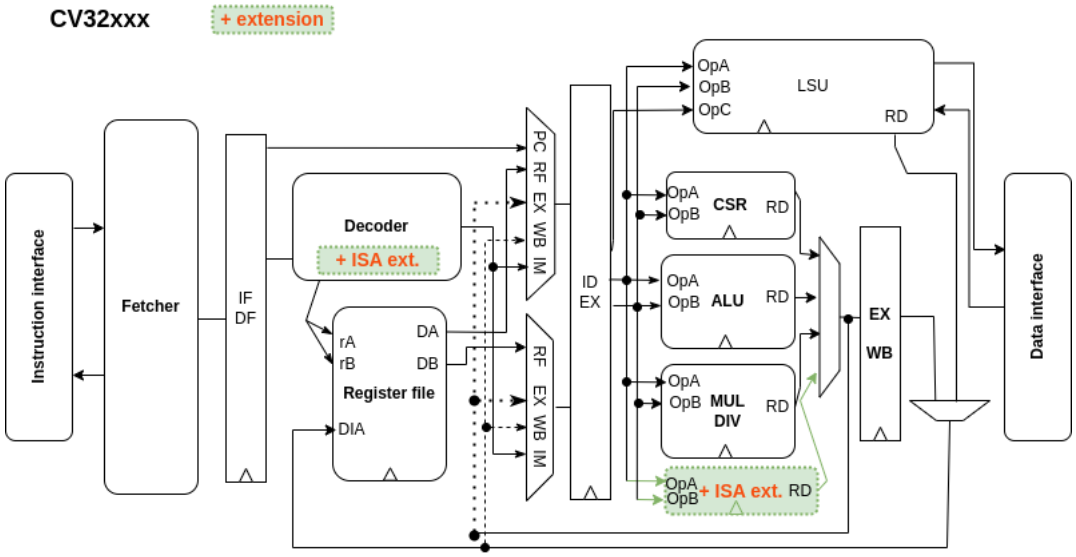

Figure 4.2 – Principe d'insertion d'instruction spécialisées dans un cœur RISC-V (schéma pédagogique du cœur [194])

être ajoutée dans l'architecture du système. Le reste de l'architecture du processeur n'est pas affecté par l'insertion de ces nouvelles instructions. Les expérimentations montrent les gains en performances que ces modifications peuvent apporter pour différents cœurs de processeur RISC-V et pour différents codes correcteurs, en fonction du parallélisme qui sera utilisé. Toutefois, ce type d'architecture avec 2 registres d'entrée et 1 registre de sortie se révèle peu efficace pour certains CCE, comme cela sera démontré par la suite.

La suite de ce chapitre présente, pour chacune des familles de CCE, les motifs d'instructions qui ont été sélectionnés.

4.2.1 Amélioration du décodage logiciel des codes LDPC

Depuis leur redécouverte, il y a une trentaine d'années, les codes LDPC ont vu la mise au point de nombreux algorithmes de décodage qui offrent à la fois de bonnes propriétés d'implantation et un pouvoir élevé en termes de correction d'erreurs. Ces derniers ont donné lieu à une multitude d'architectures matérielles tant sur cible ASIC [52] que FPGA [53].

Dans le cadre de ces travaux de thèse, l'algorithme considéré est le ***M**in-**S**um/**O**ffset **M**in-**S**um* (*MS/OMS*) offrant le meilleur compromis entre complexité d'intégration *vs* et performances de décodage. De manière analogue aux architectures matérielles [52, 53] et aux implantations logicielles sur cibles multicœurs [63], la formulation par couche hori-

Listing 4.1 – Formulation en langage C du calcul de mise à jour des noeuds de parité pour un algorihtme MS/OMS avec un ordonnancement horizontal.

```
1  for(iterations)  // boucle principale
2    for(int i = 0; i < nb_CN; i++)  // boucle Cn
3      int8_t min1 = 127, min2 = 127, signe = 0;
4      int8_t DegCn = deg_Cns[i] ;
5
6      //Parcours des Vn connectées au idx_Cn courant
7      for(int j = 0; j < DegCn; j++) // boucle Vn liés au Cn
8        int8_t contr = accuVN[ posVn[j] ] - c2v[j];
9        contr        = contr >  127 ?  127 : contr;
10       contr        = contr < -127 ? -127 : contr;
11
12       signe        = signe ^ (contr < 0);
13       int8_t abs   = contr > 0 ? contr : -contr;
14
15       if(abs < min1) min1 = abs;
16       else if(abs < min2) min2 = abs;
```

zontale des calculs de parité a été sélectionnée afin d'accélérer la vitesse de convergence du décodeur. L'algorithme de décodage est composé de plusieurs boucles imbriquées permettant le calcul de la parité des nœuds $\boldsymbol{C_n}$ et la mise à jour des nœuds de données $\boldsymbol{V_n}$. Pour ce faire, à chaque *itération* de décodage, l'algorithme va, pour chaque $\boldsymbol{C_n}$, actualiser sa valeur de parité à partir des $DegCn$ nœuds de variable qui lui sont connectés. À la suite de l'évaluation de sa parité, il va rediffuser des messages à ces mêmes $\boldsymbol{V_n}$. Par souci de clarté, un extrait de la formulation de cet algorithme de décodage est présenté ci-après dans le listing 4.1. Il s'agit d'un code en langage C présentant le calcul de la valeur interne des nœuds de parité à partir des valeurs des accumulateurs et des messages entrants. Cette description simple permet de mettre en exergue la prédominance des opérations arithmétiques et logiques simples par rapport aux accès mémoires et à l'arithmétique sur les pointeurs.

En analysant cet exemple pédagogique de l'algorithme 4.1, il est possible de se faire une idée des séquences et/ou groupes d'instructions arithmétiques et logiques pouvant être optimisées : soit par une concaténation d'opérations élémentaires, soit par un ensemble d'instructions réduisant la complexité des calculs utilisés dans les boucles. Par exemple, les calculs décrits dans les lignes *8 à 10* réalisent une soustraction signée sur 8 bits suivi d'une saturation symétrique dans l'intervalle $[127, -127]$.

Dans le meilleur des cas, cette séquence d'instructions sera exécutée en cinq cycles d'horloge sur un processeur possédant des instructions conditionnées, ou plus sur un

Listing 4.2 – Exemple pédagogique intégrant les instructions du kit ISA.

```
1  for(iterations)  // boucle principale
2    for(int i = 0; i < nb_CN; i++)  // boucle Cn
3      int8_t min1 = 127, min2 = 127, signe = 0, temp;
4      int8_t DegCn = deg_Cns[i];
5
6      //Parcours des Vn associés au Cn courant
7      for( int j = 0; j < DegCn; j++ )  // boucle Vn liés au Cn
8          int8_t contr    = i8_sub_sat127_pi8(accuVN[ posVn[j] ] - c2v[j]);
9
10         signe           = signe ^ (contr < 0);
11         int8_t abs      = i8_abs_pi8(contr);
12
13         temp            = i8_max_pi8(min1, abs);
14         min1            = i8_min_pi8(min2, temp);
15         min2            = i8_min_pi8(abs, min1);
```

cœur RISC-V qui réaliserait des branchements conditionnels. Pour réduire drastiquement ce temps d'exécution, une approche consiste à créer, par exemple, une instruction capable de réaliser cette soustraction saturée en un seul cycle d'horloge grâce à des ressources matérielles spécifiques. Cette instruction spécifique pourrait ensuite être utilisée dans une version optimisée du code C original.

Afin d'illustrer le résultat de cette transformation, on peut constater dans le listing 4.2 à la ligne *7*, la présence de cette nouvelle instruction *i8_sub8_sat127_pi8*. Celle-ci remplace les instructions des lignes *7, 8* et *9* de l'algorithme originel (listing 4.1). Cette évolution ne modifie pas le nombre d'itérations de boucle à réaliser, mais réduit la latence de chaque itération et donc minimise le temps d'exécution global.

Pour permettre l'usage de cette nouvelle instruction, il est nécessaire de l'intégrer dans le flot de compilation associé au processeur et plus particulièrement de la provisionner dans l'outil *binutils* comme cela est indiqué dans la figure 4.1. Ainsi, elle devient partie intégrante de l'ISA supportée par le processeur cible et est à la disposition des utilisateurs.

Dans le but de compléter l'intégration de cette nouvelle fonctionnalité, l'opérateur matériel associé doit être décrit au niveau RTL et inséré dans l'architecture de l'UAL du processeur cible. Afin d'illustrer ce travail, la description RTL en langage Verilog de l'opération de soustraction saturée est donné à titre d'exemple dans le listing 4.3.

L'exemple fourni dans la figure 4.2 montre que, sur une section algorithmique simple, les possibilités d'améliorations possibles offertes par l'approche proposée sont multiples. Cependant, ces instructions dédiées permettant de réaliser le calcul d'une soustraction

Listing 4.3 – Description matérielle correspondant à l'instruction *i8_sub_sat127_pi8* écrite en System Verilog.

```
//op_a : opérande a de l'UAL
//op_b : opérande b de l'UAL
//9 bits permettent de contrôler le possible overflow
logic[8:0]  ldpc_res_minus ;
assign      ldpc_res_minus = $signed(op_a[7:0])-$signed(op_b[7:0]);

always_comb begin
 unique case (operator)
     UAL_LDPC_SUB_SAT: begin
     ldpc_subsat_result  = ($signed(ldpc_res_minus) >  9'sd127)?  9'sd127:
                           ($signed(ldpc_res_minus) < -9'sd127)? -9'sd127:
                           {ldpc_res_minus[7:0]}; //8 bits
    end
end
```

Instruction	**Mnémonique**	**Implantation**
Valeur absolue sur 8 bit	i8_abs_pi8 rD,rS1	rD := ABS8(rS1)
Addition saturée à 127;-127	i8_add_sat127_pi8 rD,rS1,rS2	rD := SAT127(rS1+rS2)
Génération de masque	i8_cmpeq_pi8 rD,rS1,rS2	rD := (rS1==rS2) ?0xff:0
Maximum signé sur 8 bit	i8_max_pi8 rD,rS1,rS2	rD := MAX8s(rS1,rS2)
Minimum signé sur 8 bit	i8_min_pi8 rD,rS1,rS2	rD := MIN8s(rS1,rS2)
Valeur nouv. message	i8_invB_Aneq1_pi8 rD,rS1,rS2	rD := (rS1⩾1) ?rS2 :-rS2
Calcul du signe	i8_xorA_signB_pi8 rD,rS1,rS2	rD := rS1 xor ((rS2⩾0) ?1:0)
Soustraction saturée à 127;-127	i8_sub_sat127_pi8	rD := SAT127(rS1-rS2)

Table 4.2 – Ensemble d'instructions proposées pour améliorer le décodage de code LDPC dans un contexte de fonctionnement scalaire.

saturée, d'une valeur absolue, d'un minimum et d'un maximum (*i.e. subsat, abs, min et max*) doivent toutes être provisionnées dans l'UAL des cœurs de processeur et être capable de s'exécuter en un cycle d'horloge sans pénaliser fortement la fréquence de fonctionnement. Toutes les sections de l'algorithme ne peuvent pas être optimisées de manière analogue, c'est le cas par exemple des accès à la mémoire qui impliquent des modifications architecturales complexes en dehors de l'UAL.

Finalement, le tableau 4.2, résume les résultats de l'étude menée sur les codes LDPC. Les 8 instructions spécifiques identifiées après analyse de l'algorithme LDPC étudié y sont décrites. L'impact de ces instructions spécialisées sur les performances en termes de réduction du nombre de cycles d'horloges nécessaires pour le décodage sur un cœur RISC-V est détaillé dans la section 5.

Instruction	**Mnémonique**	**Implantation**
Addition non-signée saturée à 64 ;0	u8_addu_sat64_pu8 rD,rS1,rS2	rD := SAT64u(rS1+rS2)
Soustraction non-signée saturée à 64 ;0	u8_subu_sat64_pu8 rD,rS1,rS2	rD := SAT64u(rS1-rS2)
Minimum non-signée sur 8b	u8_minu_pu8 rD,rS1,rS2	rD := MIN8u(rS1,rS2)

Table 4.3 – Résumé des instructions spécifiques LDPC-NB.

4.2.2 Amélioration du décodage logiciel des codes LDPC non binaires

Comme introduit dans le chapitre précédent, les codes LDPC non-binaires sont une transposition des codes LDPC dans les corps de Gallois, ($\mathbb{GF}(\boldsymbol{q})$ avec $\boldsymbol{q} \geq 2$. Cette extension dans $\mathbb{GF}$ permet d'atteindre des performances bien supérieures en termes de pouvoir de correction, mais l'évolution des algorithmes de décodage engendre une augmentation considérable de la complexité calculatoire. Du fait de cette complexité, il n'existe que peu d'implantations logicielles ciblant des architectures programmables, et aucune actuellement n'implante des algorithmes manipulant des données entières [76, 164]. Ainsi, la grande majorité des implantations de décodeur LDPC non binaires sont des architectures portées sur ASIC ou FPGA [73]. L'algorithme de décodage retenu pour ces travaux de thèse est l'algorithme ***Min-Sum*** (*MS*). Son comportement est proche de celui précédemment introduit pour le décodage des codes LDPC binaires. Une matrice de parité $\boldsymbol{H}$, indiquant les connexions entre les nœuds $\boldsymbol{C_n}$ et $\boldsymbol{V_n}$ est toujours utilisée, même si les valeurs manipulées ne sont plus binaires, mais se trouvent dans un corps des Gallois. L'ordonnancement par couches horizontales est conservé afin de réduire les besoins de mémorisation et réduire le nombre d'itérations de décodage. L'algorithme de décodage est constitué de boucles imbriquées. La nature des données qui sont représentées dans les corps de Gallois apporte de la régularité au niveau des accès à la mémoire. L'algorithme ***Min-Sum*** ne requiert pas l'utilisation d'opérations complexes à implanter, telles que les multiplications. Dans les calculs de parité associés aux $\boldsymbol{C_N}$, l'utilisation d'additions et de soustractions est nécessaire. Des opérations plus "complexes" telles que de la recherche de minima et de sa position nécessitent toutefois l'utilisation de branchements conditionnels. Par rapport aux LDPC binaires, une différence forte existe au niveau de l'échange des messages entre les éléments $\boldsymbol{V_N}$ et $\boldsymbol{C_N}$, les multiplications et les divisions dans les corps de Gallois, complexes à réaliser, sont implantées à l'aide de permutations (accès mémoire indexés) dont les valeurs sont pré-calculées.

En analysant l'algorithme de décodage réécrit en langage C, dont une sous partie

Listing 4.4 – Extraits de fonctions issues de l'algorithme de décodage LDPC-NB sans instructions spécifiques.

```
1  void process_ecn(){ //calcul du noeud de parité courant
2      ...
3      for( unsigned int i =0; i<gf; j++){
4      // boucle de parcours de noeud de valeur gf avec j
5          for(unsigned int j=0; j<gf; j++){
6              int p    = i ^ j;
7              // fetch les données depuis la mémoire et addition n-signée
8              // et saturation dans [0, 63]
9              int sum  = (unsigned char)a[i] + (unsigned char)b[j];
10             int sumS = sum > 63 ? 63 : sum;
11             // r[p] prends le min
12             r[p] = (sumS < r[p]) ? sumS : r[p];
13         }
14     }
15 }
16 //update additionne et sature dans [0, 63] dans une boucle de degré GF
17 void update(int GF, unsigned char dst[GF], unsigned char src1[GF]){
18     for(int i=0; i<GF; i++) {
19         int tmp = dst[i] + src1[i];
20         dst [i] = tmp > 63 ? 63 : tmp;
21     }
22 }
```

est fournie dans le listing 4.4, nous avons procédé à l'identification et à la sélection des motifs d'instructions pouvant être optimisés. Les lignes 2 à 10 présentent l'une des boucles de calcul utilisée dans le calcul de parité. Dans cette boucle, la taille du corps du nœud courant est le nombre d'itérations nécessaires à effectuer pour la mise à jour des messages.

Dans ces deux fonctions, pour réduire ce nombre de cycles d'horloges nécessaires à la complétion des calculs, il est possible de concevoir un opérateur matériel supportant plus efficacement les traitements que les instructions présentent nativement dans l'ISA du processeur. La chaîne de compilation qui devra ensuite être mise à disposition des utilisateurs, pourra s'appuyer sur cette nouvelle instruction pour réaliser par exemple cette addition saturée en un seul cycle d'horloge. C'est ce que l'on observe dans l'algorithme mis à jour 4.6 à la ligne *11*, où l'on a remplacé l'ensemble des instructions précédentes par la nouvelle instruction *u8_addu_sat64_pu8*. D'un point de vue matériel, il est nécessaire de provisionner dans l'UAL du processeur un opérateur matériel tel que celui décrit dans le listing 4.5 pour effectuer cette addition saturée dans l'intervalle $[0, 63]$.

Le code logiciel présenté dans le listing 4.6 correspond au traitement décrit dans le listing 4.4 avec l'utilisation de cette nouvelle instruction.

Listing 4.5 – Description en System Verilog de l'opérateur associé aux instructions *u8_addu_sat64_pu8* et *u8_minu_pu8*.

```
//op_a : opérande a de l'UAL
//op_b : opérande b de l'UAL
unique case (operator_i)
    UAL_LDPCNB_MINU :
    ldpcnb_result = (op_a[7:0] >= op_b[7:0])?
                    op_b[7:0]    : op_a[7:0];

    UAL_LDPCNB_ADDU_SAT64 :
    ldpcnb_result = ((op_a[7:0] + op_b[7:0]) > 8'sd63 )?
                    8'sd63       : op_a[7:0] + op_b[7:0];
```

Listing 4.6 – Code source du décodeur bénéficiant de la présence des instructions *u8_addu_sat64_pu8* et *u8_minu_pu8*.

```
//ré-écriture direct avec l'appel aux instructions spécialisées
void process_ecn(){ //calcul du noeud de parité courant
    ...
    for( unsigned int i =0; i<gf; j++){
    // boucle de parcours de noeud de valeur gf avec j
        for(unsigned int j=0; j<gf; j++) {
            int p    = i ^ j; // simple xor
            unsigned int sumS ;
            // fetch les données depuis la mémoire et addition n-signée
            // et saturation dans [0, 63]
            sumS = u8_addu_sat64_pu8(sumS,(unsigned char)a[i],(unsigned
                char)b[j]);
            // r[p] prends le min
            r[p] = u8_minu_pu8( sumS ,r[p] );
        }
    }
}
//update additionne et sature dans [0, 63] dans une boucle de degré GF
void update(int GF, unsigned char dst[GF], unsigned char src1[GF] )
    for(int i=0; i<GF; i++){
        dst[i] = u8_minu_pu8(dst[i], src1[i]);
    }
```

Cette analyse s'est poursuivie sur l'ensemble du processus de décodage. Le tableau 4.3 regroupe l'ensemble des instructions retenues pour accélérer le décodage des LDPC-NB. Le nombre réduit d'instructions s'explique ici par la faible diversité des opérations arithmétiques utilisées dans cet algorithme de décodage et la prépondérance des accès mémoire mis en œuvre par exemple pour réaliser les opérations de multiplication et de division dans $\mathbb{GF}$.

Listing 4.7 – Description de la fonction *func_f()* utilisée dans l'algorithme *SC*, ainsi que la fonction *node()* de décodage de l'arbre.

```
//node est la fonction de parcours de l'arbre de décodage
void node(N, LLR, PS,...){
    if (N == 1) { Sommes partielles et bits gelés; }
    for(x=0; x < N/2 ; x+=1){
        (LLR+N)[x]= func_f( LLR[x],(LLR+N/2)[x] );
    }
    node(...)
    for(x=0; x < N/2 ; x+=1){
        temp = func_g( ptr_sum[x], LLR[x], (LLR+N/2)[x] );
    }
    node(...)
    for(x=0; x < N/2 ; x+=1){
        ptr_sum[x] = func_h( ptr_sum[x], ptr_sum[x+(N/2)] );
    }
  }

//fonction f de l'arbre : retourne le minimum signée entre les LLRs (la,lb)
int8_t func_f(int8_t la, int8_t lb){
    int8_t min1 = abs(la);
    int8_t min2 = abs(lb);
    if(min1 > min2) min1 = min2;
    int8_t sign = (la < 0) ^ (lb < 0);
    return (sign==0)? min1 : -min1;
}
```

4.2.3 Algorithme de décodage logiciel des codes polaires

Les codes polaires sont une famille récente de CCE [105], mais fait d'ores et déjà partie intégrante du standard 5G. Les enjeux liés à ce standard ont motivé bon nombre de travaux de recherche durant la dernière décennie. De nombreux algorithmes de décodage ont été proposés dans la littérature, comme cela a été exposé dans le chapitre précédent. Ces derniers ont donné lieu à la réalisation d'implantation matérielles dédiées, mais aussi au développement de solutions logicielles [163, 195]. Dans ces travaux, nous nous sommes focalisés sur l'un des algorithmes qui a est adapté aux implantations logicielles, le *SC*, ainsi qu'à sa variante *Fast-SC*(F-SC) [94]. Il est à noter que ces travaux pourraient par la suite être étendus à l'optimisation de décodeurs basés sur l'utilisation de l'algorithme SC-list [168].

De manière analogue aux réalisations pour les CCE précédents, l'étude menée prend pour source le code C optimisé décrivant ces deux algorithmes de décodage. Une fonction de calcul mise en œuvre dans l'algorithme *SC* est présentée dans le listing 4.7. Ce listing

Listing 4.8 – Description de l'opérateur *i8_Fx_pi8* en langage System Verilog.

```
//op_a : opérande a de l'UAL
//op_b : opérande b de l'UAL
assign sign1 = op_a[7] ^ op_b[7] ; //extraction du signe via lecture MSB[7]
assign min1 = (($signed( op_a[7:0]) >= 0) ? op_a[7:0] : -op_b[7:0] );
assign min2 = (($signed( op_b[7:0]) >= 0) ? op_b[7:0] : -op_a[7:0] );

UAL_POLAR_F: polar_result =
  {(min1 > min2)?  (sign1 == 0)? min2 : -min2 :
                   (sign1 == 0)? min1 : -min1
  };
end
```

décrit l'algorithme de décodage sous sa forme récursive ainsi qu'une fonction de calcul régulièrement exécutée. Cette fonction *func_f()* est utilisée pour calculer les données des branches de gauche lors du parcours de l'arbre binaire. Elle est composée d'une séquence d'opérations arithmétiques interdépendantes.

Il est possible de remplacer toutes les instructions de cette fonction par une unique instruction spécialisée. En effet, la fonction f : (a) ne nécessite que deux variables d'entrée et (b) ne produit qu'une seule et unique sortie, ce qui correspond aux contraintes architecturales fixées. La conception de l'opérateur matériel associé, dont une description est fournie dans le listing 4.8, doit permettre de réduire notablement le temps d'exécution du processus de décodage.

Une analyse complète de l'algorithme *SC* a permis d'identifier six motifs pouvant bénéficier de la conception d'instructions dédiées, comme cela est résumé dans le tableau 4.4. Il est à noter que la même procédure a été appliquée sur l'algorithme *Fast-SC* qui est une extension de l'algorithme *SC*. Cela a permis d'identifier cinq instructions supplémentaires comme mentionné dans le tableau 4.4.

4.2.4 Algorithme de décodage logiciel des turbo codes

Les turbo codes [127, 196] sont connus et utilisés depuis plusieurs années dans de nombreux standards, comme par exemple le 4G-LTE [3]. Lors de la conception de décodeurs associés, qu'ils soient matériels ou logiciels, l'algorithme *Max-log-MAP* (MLM) est souvent plébiscité à cause du compromis complexité / performance qu'il offre. C'est aussi, d'après les travaux de la littérature, le plus adapté pour une implantation logicielle [197], justifiant sa sélection. Comme cela a été introduit dans le chapitre précédent, les

Instruction	**Mnémonique**	**Implantation**
Addition signée saturée à 8 bit	i8_add_sat127_pi8 rD,rS1,rS2	rD := SAT127(rS1 + rS2)
Soustraction signée saturée à 8 bit	i8_sub_sat127_pi8 rD,rS1,rS2	rD := SAT127(rS1 - rS2)
Fonction F de l'arbre de décodage	i8_Fx_pi8 rD,rS1,rS2	min1 := ABS8b(rS1) min2 := ABS8b(rS2) min1 := MIN8(min1, min2) sign := (rS1 < 0)∧(rS2 < 0) rD := (sign == 0) ? min1 : -min1
Fonction R de l'arbre de décodage	i8_Rx_pi8 rD,rS1,rS2	rD := (rS2 == 1) ?0 : (rS1 < 0) ?1 : 0

Table 4.4 – Résumé des instructions spécifiques au décodage des codes polaires (SC et F-SC) dans un contexte scalaire.

Listing 4.9 – Code source du décodeur bénéficiant de la présence de l'instruction *i8_Fx_pi8*.

```
//Nouvelle fonction f de l'arbre fait appel à l'instruction i8_Fx_pi8()
int8_t func_f(int8_t la , int8_t lb){
    return i8_Fx_pi8(la,lb);
}
//le décodage de l'arbre reste inchangé
void node(N, LLR, PS,...){
    if (N == 1) { Sommes partielles et bits gelés ;}
    for(x=0; x < N/2 ; x+=1)
        (LLR+N)[x]= func_f( LLR[x], (LLR+N/2)[x] );
    node(...)
    for(x=0; x < N/2 ; x+=1)
        temp = func_g( ptr_sum[x], LLR[x], (LLR+N/2)[x] );
    node(...)
    for(x=0; x < N/2 ; x+=1)
        ptr_sum[x] = func_h( ptr_sum[x], ptr_sum[x+(N/2)] );
  }
```

algorithmes de turbo décodage opèrent sur des treillis sur lesquels ils calculent divers métriques de branches. Cela se traduit dans la formulation algorithmique par la présence de boucles dans lesquelles, pour chaque état du treillis, différentes valeurs sont mises à jour. Contrairement aux autres familles de CCE, ces métriques codées en virgule fixe ne sont pas saturées. Ainsi, dans la version scalaire de l'algorithme de décodage, peu de motifs d'instructions répondent aux contraintes énoncées. Parmi les motifs restants, le facteur de pénalisation des valeurs des extrinsèques présenté dans le listing 4.10 est un candidat.

En effet, l'application d'un coefficient *3/4* sur une valeur entière signée ne peut pas être réalisée à l'aide d'un décalage à droite et d'une simple soustraction. Cette optimisation

Listing 4.10 – Extrait minimal de la fonctions scalaire *scale* de algorithme de décodage turbo code.

```
int8_t scale_ext(int8_t llr_ext){
    return val = (i_ext[k] * 3) / 4 ;
}
```

Listing 4.11 – Exemple de l'instruction *scale* spécifique dans l'UAL en System Verilog.

```
//op_a : opérande a de l'UAL
assign tb_sign = ($signed(op_a) >= 0 )? 1 : 0 ;
assign tb_in_A = (tb_sign) ? op_a[7:0] : -op_a[7:0] ;
assign tb_in_B = tb_in_A >> 2 ;
assign tb_in_C = tb_in_A - tb_in_B ;

UAL_TURBO_SCALE : begin
    turbo_scale_result = (tb_sign) ? tb_in_C : -tb_in_C ;
end
```

provoque une asymétrie liée au codage en complément à deux des données signées. Ainsi, cette mise à l'échelle des extrinsèques peut s'avérer onéreuse en termes de temps d'exécution. Cette suite d'instructions, onéreuse en cycles d'horloge, peut être efficacement réalisée d'un point de vue matériel tel que cela est présenté dans le listing 4.11.

L'absence de saturation dans ces algorithmes de décodage, ainsi que les contraintes architecturales (2 opérandes d'entrée) limitent le nombre de motifs pouvant bénéficier d'une implantation matérielle. Ainsi, pour cet algorithme, uniquement 2 instructions sont identifiées comme pertinentes à l'amélioration de la latence générale du décodeur tel que cela est résumé dans le tableau 4.5.

4.2.5 Synthèse des extensions proposées

Dans cette partie, une synthèse des différentes instructions qui ont été retenues est proposée. Ces instructions, listées dans le tableau 4.6 ont été conçues pour travailler sur

Instruction	**Mnémonique**	**Implantation**
Maximum non-signée sur 8b	i8_max_pi8 rD,rS1,rS2	rD := MAX8u(rS1,rS2)
Application coefficient 0.75	i8_scale_pi8 rD,rS1	Sign := (rS1 $\geqslant$0)? 1:0
		A := ABS8b(rS1)
		B := A$\gg$2
		C := A-B
		rD := (Sign)? -C : C

Table 4.5 – Résumé des instructions spécifiques au décodage des turbo codes

Mnémonique proposé	**LDPC** OMS	**LDPC-NB** MS	**polaire** F-SC/SC	**turbo code** MLM
i8_abs_pi8 rD,rS1	✓	-	-	-
i8_add_sat127_pi8 rD,rS1,rS2	✓	-	✓	-
i8_cmpeq_pi8 rD,rS1,rS2	✓	-	-	-
i8_max_pi8 rD,rS1,rS2	✓	-	-	✓
i8_min_pi8 rD,rS1,rS2	✓	-	-	-
i8_invB_Aneq1_pi8 rD,rS1,rS2	✓	-	-	-
i8_xorA_signB_pi8 rD,rS1,rS2	✓	-	-	-
i8_sub_sat127_pi8 rD,rS1,rS2	✓	-	✓	-
u8_addu_sat64_pu8 rD,rS1,rS2	-	✓	-	-
u8_subu_sat64_pu8 rD,rS1,rS2	-	✓	-	-
u8_minu_pu8	-	✓	-	-
i8_scale_pi8	-	-	-	✓
i8_Fx_pi8 rD,rS1,rS2	-	-	✓	-
i8_Rx_pi8 rD,rS1,rS2	-	-	✓	-
Total	8	3	4	2

Table 4.6 – Récapitulatif des instructions proposées pour l'amélioration du décodage des CCE.

des données scalaires codées sur 8 bits. Ce format de représentation limite les pertes de performances lors du décodage de codes CCE par rapport au format flottant et permet d'imaginer leur extension sous la forme d'instructions SIMD par la suite. Toutefois, il est possible de les transposer sur des formats de quantification supérieurs, c.-à-d.. 32 bits. Le tableau 4.6 montre que le décodage des codes LDPC peut bénéficier de 8 instructions spécialisées, tandis que les algorithmes de décodage des codes LDPC-NB et des codes polaires (SC et F-SC) n'en possèdent que 4. Les améliorations sont plus limitées pour le turbo décodage où seules 2 instructions ont été identifiées. Il est intéressant de noter qu'il existe un sous-ensemble d'instructions commun à tout ou partie des codes CCE. Ainsi, afin d'accélérer l'ensemble des décodeurs, seules 17 instructions sont nécessaires. Afin de tester et d'évaluer l'efficacité et la pertinence des instructions proposées, le flot décrit dans la figure 4.1 est mis en place. Nous allons commencer par détailler celui-ci, puis nous présenterons les cœurs RISC-V retenus dans le cadre de ces travaux de thèse avant de présenter les résultats expérimentaux.

4.3 Évaluation de l'impact de l'ajout d'instructions spécialisées sur cœurs RISC-V

Les premiers travaux autour du RISC-V ont généré une importante émulation au sein des communautés scientifiques et industrielles, menant au développement de nombreux cœurs. À ce jour, de multiples cœurs RISC-V sont disponibles sous des licences *open-source* à des fins d'évaluation et/ou d'expérimentations. Les implantations varient tant au niveau du jeu d'instructions que des caractéristiques intrinsèques de l'architecture (nombre d'étages de pipeline, complexité de l'UAL, taille des mémoires caches…). Dans certains cœurs des mécanismes plus complexes, tel que la capacité d'exécution dans le désordre (*i.e. Out-of-Order*) et/ou l'utilisation de front-end super-scalaire ont été intégrés. Toutefois, chaque cœur est potentiellement extensible et modifiable pour permettre l'ajout d'instructions dédiées. Cependant, leurs complexités architecturales et la qualité de leurs descriptions sources peuvent parfois rendre complexe, voir compromettre, ce type d'opération. Dans le cadre de ces travaux et après une vaste étude préliminaire, quatre cœurs RISC-V ont été sélectionnés : **PicoRV32**[198], **RISCY**[199], **IBEX**[200, 201] et **SCR1**[202].

Cette sélection est basée sur les différences architecturales, les performances générales et la complexité matérielle respective des cœurs. L'objectif principal étant d'évaluer de la façon la plus objective possible l'apport et l'impact des différentes instructions ajoutées avec notre approche. Les quatre architectures RISC-V sélectionnées supportent le jeu d'instruction RV32-IMC (I : integer, M : multiplication et division, C : instructions compressées). Comme cela est indiqué dans le tableau 4.7, le nombre d'étages de pipeline mis en œuvre varie entre 1 et 4.

Les caractéristiques principales des quatre cœurs sélectionnés sont présentées ci-dessous :

— **PicoRV32**[198] est un cœur à faible complexité. Il a été conçu pour supporter des applications simples. Son implantation matérielle est légère et permet d'atteindre des fréquences de fonctionnement élevées (>300 MHz) sur des cibles de type FPGA. Cependant, il ne possède qu'un seul étage de pipeline avec exécution dans l'ordre.

— **RISCY**[199] est un cœur RISC-V performant initialement développé par les équipes du MIT et depuis maintenu par le projet *PULP*. Ce cœur implémente un jeu d'instructions spécialisées visant à l'optimisation des opérations arithmétiques régulières. Son architecture cible principalement des applications DSP à faible consommation

	Pipelines	CoreMark/MHz	Complexité (LUTs)	Sources
PicoRV32	1	0.4	∼1 000	Verilog
IBEX	2	2.47	∼2 500	Sys. Verilog
SCR1	3	2.43	∼4 000	Sys. Verilog
RISCY	4	3.19	∼10 000	Sys. Verilog

Table 4.7 – Comparatifs de cœurs de processeurs RISC-V sélectionnés

énergétique. Il possède 4 étages de pipeline avec exécution dans l'ordre [203].

— **IBEX**[200, 201] initialement connu comme le *micro-RISCY*, ce cœur offre un compromis entre performance et complexité matérielle. Il implémente 2 étages de pipeline avec exécution dans l'ordre dans sa configuration de base. Similairement au **RISCY**, son développement a commencé au sein du projet PULP avant d'être cédé au groupe *lowRISCV*.

— **SCR1**[202] est un cœur performant qui atteint un score MIPS (Million Instructions Per Second) élevé tout en conservant une complexité matérielle modérée. Son architecture utilise trois étages de pipeline avec exécution dans l'ordre dans sa configuration de base.

Comme mentionné plus haut, ces quatre cœurs RISC-V ont été sélectionnés afin de proposer une évaluation impartiale de nos extensions des ISA de ces processeurs. En effet, les différentes caractéristiques architecturales permettent une analyse des bénéfices et des coûts dans différents contextes. Les résultats obtenus sont présentés et analysés dans la prochaine section. Il faut noter, et ce, afin de garantir l'homogénéité des tests et de simplifier les comparaisons futures, que le jeu d'instruction des différents cœurs a été restreint à la spécification RV32I. Dans cette section, les performances de l'ensemble des instructions spécialisées identifiées et déployées sur les cœurs RISC-V sont présentées et évaluées. Comme décrit dans les sections précédentes, les versions logicielles originales des CCE considérés ont été exécutées sur chacun des cœurs sans modification, établissant ainsi les performances de référence. Ensuite, les codes sources modifiés, intégrant l'usage des instructions spécifiques, ont également été exécutés dans le but de mesurer la différence entre les temps d'exécution. De même, l'impact en termes de complexité matérielle de l'ajout de ces nouvelles instructions dans les cœurs RISC-V, ainsi que l'évolution de leur fréquence de fonctionnement maximale ont été étudiés. À cette fin, les versions originales et enrichies des cœurs ciblés ont été prototypées sur des FPGA *Xilinx Artix-7* et *Kintex-7*.

Pour mener à bien ces expérimentations, les programmes décrivant le comportement des décodeurs ont été écrits en langage C visant une exécution en mode bare-metal. Afin de

compiler l'ensemble de ces codes et de produire les binaires exécutés par les cœurs RISC-V, GCC 12.2 a été utilisé. Le compilateur est configuré afin de générer un exécutable ne comportant que des instructions RV32I et avec le flag d'optimisation -O3.

Un ensemble d'expérimentations a été conduit afin de mesurer les gains apportés par l'utilisation d'instructions spécialisées opérant sur des données scalaires codées sur 8 bits (SISD). Ces expérimentations ont été réalisées sur chacune des cibles présentées précédemment.

Un sous-ensemble des résultats de mesure concernant les temps d'exécution et les débits des différents décodeurs est présenté dans le tableau 4.8. Les valeurs fournies dans ce tableau présentent pour chacun des cœurs, le nombre de cycles d'horloge nécessaires pour l'exécution du code logiciel original (sans instruction dédiées), noté ❶*Baseline* ainsi que le nombre de cycles d'horloge nécessaires à la version exploitant les nouvelles instructions (❷*SISD ISA 2R*). La différence entre ces deux mesures est indiquée sur les lignes *Gain* ❶→❷, le gain apporté par la version enrichie est exprimé en pourcentage. Il est important de noter que le gain en pourcentage est la métrique la plus cohérente au vu des différences d'architectures des cœurs de tests. Les débits sont calculés en fonction de la fréquence maximale obtenue après placement et routage par l'outil Vivado sur cible FPGA.

L'amélioration des performances pour l'algorithme de décodage LDPC est significative pour l'ensemble des cœurs considérés. La réduction du nombre de cycles d'horloges varie de 36% dans le cas le plus défavorable (**IBEX**) et atteint $\approx 40\%$ avec le cœur **SCR1**. Ces gains sont liés à la forte diminution du nombre d'instructions dans les kernels de calcul dont la complexité était prépondérante jusque-là.

Le tableau 4.9 permet une lecture plus fine des instructions impliquées lors de l'exécution des décodeurs LDPC. Ces données font apparaître le type et nombre d'instructions exécutées entre les 2 versions du même algorithme, sur le cœur **PicoRV32**. Les instructions de contrôle (*i.e.* branchements conditionnels ou sauts), ainsi que les opérations arithmétiques et logiques sont en nette diminution. Les gains sont d'environ 83% pour les premières et 60% pour les secondes). L'analyse de la ligne *Total* permet d'observer la réduction du nombre d'instructions réellement exécutées par le cœur **PicoRV32** pour les deux versions du décodeur LDPC. Avec une réduction avoisinant 40% d'instructions exécutées, l'efficacité des choix réalisés est avérée.

Si l'on s'intéresse au décodage des codes polaires par l'algorithme *SC*, dont les valeurs sont résumées dans le tableau 4.11, les gains mesurés ont une plus grande variabilité. Les gains les plus faibles sont obtenus pour les expérimentations réalisées sur le cœur **SCR1**

			LDPC OMS	SC	polaire F-SC	LDPC-NB MS	turbo MLM
PicoRV32	❶ Baseline	cycles	217172	1333324	676218	3476435	2503511
	❷ SISD	cycles	131853	753144	332693	2875953	2133442
	Gain (❶→❷)	cycles	85319	580180	343525	600482	370069
	Gain (❶→❷)	(%)	39.3%	43.5%	50.8%	17.3%	14.8%
	Débit ❶	Kbits/s	429	2106	4153	252	1695
	Débit ❷	Kbits/s	659	3357	7599	305	1947
	Accélération (❶→❷)		1.5×	1.6×	1.8×	1.2×	1.1×
IBEX	❶ Baseline	cycles	84966	516441	261090	1390238	1018910
	❷ SISD	cycles	54340	288594	138544	1086873	782778
	Gain (❶→❷)	cycles	30626	227847	122546	303365	236132
	Gain (❶→❷)	(%)	36.0%	44.1%	46.9%	21.8%	23.2%
	Débit ❶	Kbits/s	317	1571	3107	18	1204
	Débit ❷	Kbits/s	515	2942	6129	24	1615
	Accélération (❶→❷)		1.6×	1.9×	2.0×	1.3×	1.3×
SCR1	❶ Baseline	cycles	76775	397790	218120	1310272	927495
	❷ SISD	cycles	47995	274793	125037	1081939	758988
	Gain (❶→❷)	cycles	28780	122997	93083	228333	168507
	Gain (❶→❷)	(%)	37.5%	30.9%	42.7%	17.4%	18.2%
	Débit ❶	Kbits/s	362	2107	3842	20	1366
	Débit ❷	Kbits/s	586	3021	6639	24	1661
	Accélération (❶→❷)		1.6×	1.4×	1.7×	1.2×	1.2×
RISCY	❶ Baseline	cycles	77678	348107	192935	1103396	890712
	❷ SISD	cycles	48096	228581	113843	878938	674609
	Gain (❶→❷)	cycles	29582	119526	79092	224458	216103
	Gain (❶→❷)	(%)	38.1%	34.3%	41.0%	20.3%	24.3%
	Débit ❶	Kbits/s	105	706	1274	7	417
	Débit ❷	Kbits/s	170	1075	2159	9	551
	Accélération (❶→❷)		1.6×	1.5×	1.7×	1.3×	1.3×

Table 4.8 – Comparaison des performances des cœurs usant des instructions à 2 entrées, configuration 8 bits SISD avec la fréquence de fonctionnement maximale pour chaque implantation.

qui voient se réduire de 30.9% le temps de calcul. Les gains les plus importants sont quand à eux obtenus sur le cœur **PicoRV32** ou le temps d'exécution est réduit jusqu'à 43.5%. La version *Fast-SC* de l'algorithme de décodage bénéficie d'une amélioration variant de 50.8% à 40.9% sur l'ensemble des cœurs RISC-V testés. Ce différentiel, par rapport à

Type d'instruction	décodeur baseline	décodeur modifié	différence
Contrôle	8729	1493	-82.9%
Arithm. et logique	44689	17837	-60.1%
Instructions dédiées	0	9360	>100%
Accès mémoire	8912	9229	+3.6%
Total	62332	37921	-39.2%

Table 4.9 – Nombre d'instructions exécutées lors du décodage de codes LDPC sur le cœur **PicoRV32** classées en fonction de leur type.

Type d'instruction	décodeur baseline	décodeur modifié	différence
Contrôle	37238	10517	-71.8%
Arithm. et logique	116527	37887	-67.5%
Instructions dédiées	0	6704	>100%
Accès mémoire	38515	31865	-17.3%
Total	192280	86973	-54.7%

Table 4.10 – Nombre d'instructions exécutées lors du décodage de codes polaires sur le cœur **PicoRV32** (algorithme *Fast-SC*), classées en fonction de leur type.

l'algorithme SC, s'explique par l'élagage qui évite de descendre trop profondément dans l'arbre de décodage où le ratio entre le nombre de calculs sur le nombre d'accès mémoire et d'instructions de contrôle est faible. Ces explications sont confortées par les données expérimentales exposées dans le tableau 4.10. Ces données précisent le nombre d'instructions de chaque type nécessaire à l'exécution de l'algorithme de décodage sur les 2 cœurs **PicoRV32**.

Si l'on s'intéresse aux autres CCE présentés dans le tableau 4.8, on peut noter que les gains en termes de temps d'exécution pour l'algorithme de décodage des codes LDPC-NB n'est pas aussi importante en comparaison des autres codes. Ces derniers avoisinent 22% sur le cœur **IBEX** et seulement 17.3% sur le cœur **PicoRV32**. La cause de ces gains plus faibles est liée au nombre important d'accès à la mémoire. En effet, le nombre d'accès mémoire est à peu près équivalent au nombre de calculs à réaliser, cela limite le facteur d'accélération.

Enfin, pour conclure, les facteurs d'accélération atteint par les turbo décodeurs, tels qu'indiqué dans le tableau 4.8 varient de 14.8% sur le cœur **PicoRV32** tandis qu'ils atteignent 24.2% sur le cœur **RISCY**. Ces faibles gains s'expliquent par le faible nombre d'instructions spécifiques identifiées.

	BASE	LDPC	POLAIRE		LDPC-NB	TURBO
		OMS	SC	Fast-SC	MS	Max-Log MAP
Nb. Insn. Cst.		8	4	4	3	2
PicoRV32	978 LUTs	+34%	+12.3%	+12.3%	+12.5%	+12.6%
	567 FFs	+3%	+1.6%	+1.6%	+1.2%	+0.9%
	343 MHz	-6.7%	-9.9%	-9.9%	0.0%	-2.1%
IBEX	2446 LUTs	+9.1%	+3.1%	+3.1%	+4%	+4.2%
	883 FFs	+0.0%	+0.0%	+0.0%	+0.0%	+0.0%
	99 MHz	+3.8%	+4.7%	+4.7%	+4.1%	+3.1%
SCR1	3986 LUTs	+5.3%	+3.0%	+3.0%	+3.3%	+1.6%
	2399 FFs	0.0%	0.0%	0.0%	0.0%	0.0%
	102 MHz	+1.0%	-0.9%	-0.9%	-0.4%	-0.5%
RISCY	10302 LUTs	+1.8%	+2.1%	+0.7%	+0.8%	+0.6%
	3033 FFs	0.0%	0.0%	0.0%	0.0%	0.0%
	30 MHz	0.0%	0.0%	0.0%	0.0%	0.0%

Table 4.11 – Surcoût matériel de l'ajout d'instructions spécialisées (en LUT)

Ces résultats expérimentaux sur les cœurs RISC-V démontrent que l'enrichissement proposé améliore notablement, dans tous les cas analysés, l'implantation logicielle des algorithmes de décodage. Cependant, ces gains sont non homogènes démontrant, de ce fait, l'intérêt d'évaluer ce travail sur plusieurs cœurs.

L'extension du jeu d'instructions des cœurs et l'adjonction d'un ensemble d'opérateurs matériels spécifiques dans le chemin de données modifie les caractéristiques post-synthèse. Plus spécifiquement, modifier la logique interne et l'architecture d'un cœur de processeur impacte la complexité matérielle et potentiellement la fréquence maximale de fonctionnement. Pour analyser les pénalités engendrées par les instructions ajoutées pour le support des décodeurs de CCE, différentes expérimentations ont été menées. Les résultats obtenus sont reportés dans le tableau 4.11. Les données présentées correspondent aux valeurs obtenues après les étapes de placement et de routage.

Les données présentées dans la table 4.11 et les figures 4.3 montrent que le surcoût introduit par nos instructions dépend fortement du cœur RISC-V considéré. Même si la complexité matérielle des parties opératives placées dans l'UAL est constante, le surcoût engendré par la modification du décodeur d'instructions et des barrières de registres du pipeline lui est variable en fonction des cœurs. Ces différences peuvent être de plus exacerbées par les scripts de synthèse fournis avec les différents cœurs qui ne sont pas tous identiques en termes d'objectif d'optimisation (vitesse versus complexité). Ainsi, l'intégration des instructions proposées pour le décodage des codes LDPC augmente le coût de

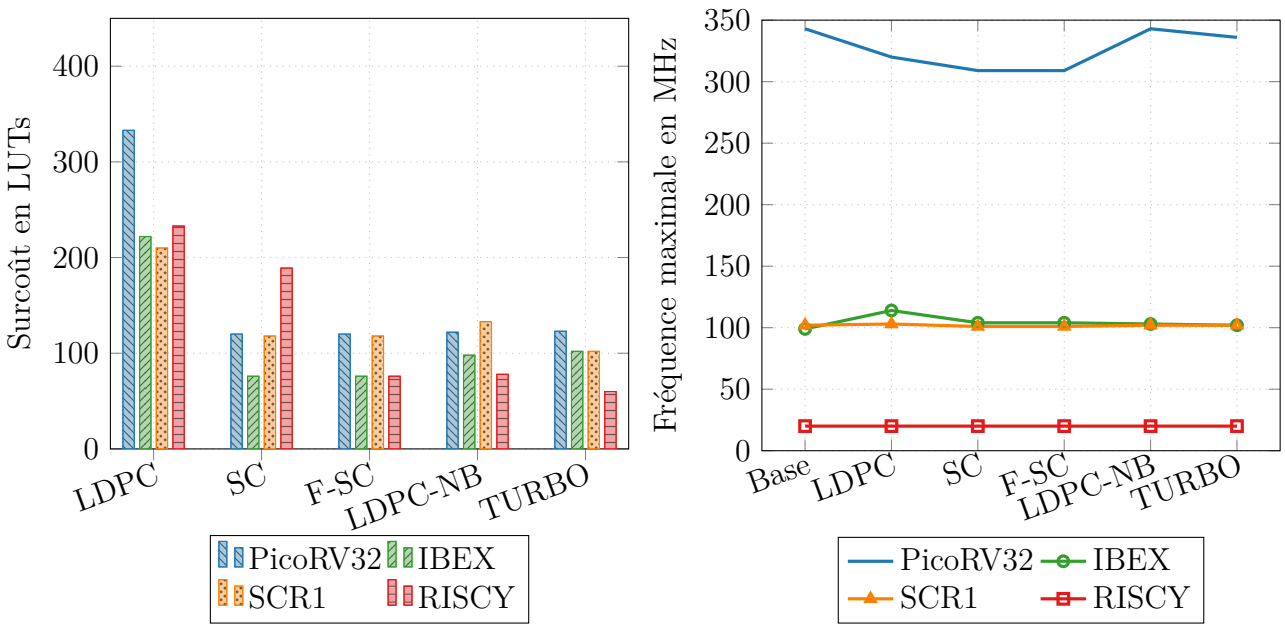

Figure 4.3 – Impact de l'ajout des instructions spécialisées : surcoût en LUT et fréquence de fonctionnement.

l'architecture **IBEX** d'environ +9.1% en termes de LUTs, tandis que pour le cœur **PicoRV32** plus compact, l'augmentation est de 34%. Si l'on considère les autres extensions pour lesquelles le nombre d'instructions ajoutées est moins important, les surcoûts sont plus faibles. Cependant, l'analyse fine de ces différents résultats, et le positionnement du coût des extensions les unes par rapport aux autres n'est pas triviale, à cause des choix opérés par Vivado lors de la synthèse logique et du placement/routage.

Ces observations sont confirmées par l'analyse de l'évolution des fréquences maximales de fonctionnement. Dans la figure 4.3, la fréquence de fonctionnement maximale pour le **PicoRV32** est affectée par l'intégration des différents jeux d'instructions. Ce cœur subit une augmentation de la durée de son chemin critique de $\approx 7\%$ (LDPC) à $\approx 10\%$ (polaire) en fonction des instructions qui lui sont ajoutées. A l'opposé, sur le cœur **RISCY**, la durée du chemin critique n'est pas impactée car ce dernier ne se situe pas dans l'étage d'exécution. Les expérimentations sur les cœurs **IBEX** et **SCR1** démontrent quant à elles l'imprévisibilité des résultats de synthèse logique liée aux choix réalisés par les outils. Ainsi l'on observe une augmentation de la fréquence de fonctionnement de $\approx 3\%$ à $\approx 5\%$ est obtenue pour le cœur **IBEX** tandis que pour le **SCR1**, la fréquence varie de -1% à $+1\%$.

Les résultats présentés dans cette partie mettent en avant l'intérêt de l'approche SISD. En effet, l'impact sur les caractéristiques matérielles des cœurs reste bien inférieur aux

gains en termes de temps d'exécution présentés dans le tableau 4.8. Il est important de considérer qu'un travail supplémentaire d'optimisation des opérateurs matériels ajoutés pourrait permettre de réduire leur coût en partageant par exemple certaines des ressources mises en œuvre dans l'UAL traditionnelle et celles nécessaires pour les nouvelles instructions.

4.4 Conclusion

Dans ce chapitre, nous avons présenté la méthodologie d'identification et de conception d'instructions spécialisées pour des algorithmes de décodage des CCE. Les instructions proposées ont été conçues sous forme kits d'instructions suffisamment agnostiques de l'architecture cible, pour pouvoir être intégrées dans tous types de cœur de processeurs.

Pour démontrer l'intérêt de l'approche et la pertinence de ces instructions retenues, nous avons porté nos kits sur plusieurs cœurs RISC-V. Les expérimentations menées permettent, évidemment d'observer que ces instructions spécialisées à des traitements spécifiques (ici les CCE) dans des cœurs, réduisent fortement les temps d'exécution des applicatifs pour lesquels ils ont été conçus.

Toutefois, la nature d'exécution de ces décodeurs de CCE permet d'utiliser des schémas de parallélisation de type inter-trames et intra-trame afin de considérablement améliorer les performances en termes de débit et de latence. Ces techniques ainsi que les instructions spécialisées associées de type SIMD sont présentées dans le chapitre suivant.

Chapitre 5

Extensions SIMD pour l'accélération des décodeurs de CCE

Dans ce chapitre, l'amélioration des instructions scalaires opérant sur des données codées sur 8 bits vers des instructions de type SIMD afin d'exploiter l'architecture 32 bits des processeurs est détaillée. L'objectif est alors de permettre une montée en débit des décodeurs logiciels grâce aux stratégies de parallélisation utilisées dans la littérature. De manière analogue à l'intégration des instructions scalaires, les résultats issus du prototypage sur cible FPGA des processeurs RISC-V enrichis évaluent l'impact des propositions sur le débit et la complexité.

Il est à noter que dans ce chapitre, comme pour le précédent, nous étudierons des architectures à 2 registres d'entrée et un registre de sortie. Ce format correspond au format standard des jeux d'instructions des processeurs x86 (Intel, AMD), RISC et dérivés (ARM) et RISC-V. Une étude exploitant des architectures levant en partie cette restriction sera présentée dans le chapitre suivant.

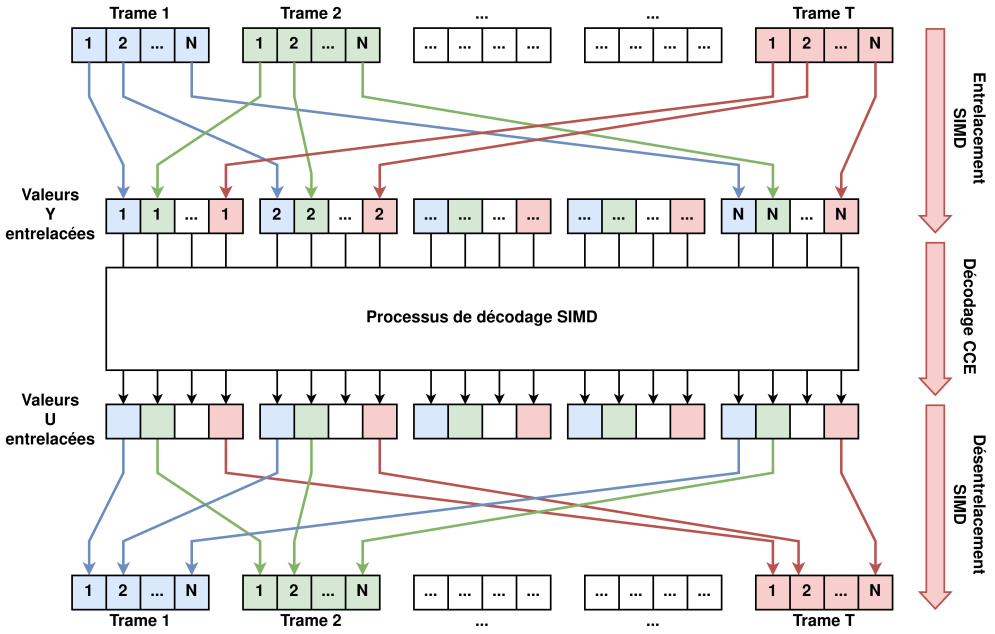

Figure 5.1 – Approche de la parallélisation inter-trames incluant les étages d'entrelacement dans l'ordre et inverse.

5.1 Introduction

L'étude et les expériences précédentes se sont focalisées sur des instructions de type SISD. Toutefois, dans la littérature, les décodeurs logiciels tirent parti des fonctionnalités SIMD des architectures multicœurs actuelles pour atteindre des niveaux de performances élevés. Ainsi, l'extension des instructions scalaires sur 8 bits vers des vecteurs de $Q \times 8$ bits est étudié et évalué dans cette section. Comme cela a été introduit dans le chapitre précédent, les décodeurs de CCE manipulent des données sur 8 bits. L'architecture interne des processeurs modernes, de leur chemin de données à leurs registres, sont sur 32 bits, voir 64 bits. Ainsi il est théoriquement possible sur des architectures 32 bits de traiter jusqu'à quatre données en parallèle en un même cycle d'horloge si les formulations des algorithmes le permettent. Le schéma de parallélisation inter-trames consiste à répliquer les mêmes opérations de décodage sur plusieurs trames indépendantes afin de mettre en exergue un parallélisme de calcul facilement accessible. Cela permet d'augmenter le débit des architectures au prix de faibles modifications dans les codes algorithmiques. Cette parallélisation inter-trames a été utilisée à l'origine dans des travaux relatifs à l'accélération des décodeurs CCE sur des architectures multicœurs [60, 197, 204]. Toutefois, afin de faciliter les accès mémoires et maximiser l'utilisation les unités de calcul SIMD, les don-

nées doivent être arrangées correctement dans la mémoire. Un entrelacement des données contenues dans les différentes trames est réalisé en amont du processus de décodage, tel que cela est schématisé dans la figure 5.1. Pour maintenir le comportement du système, à la fin de chaque décodage, l'opération inverse est exécutée. Comme cela a été rapporté dans [60], le temps d'exécution des ces étapes d'entrelacement des données est négligeable par rapport au temps d'exécution des processus de décodage ces CCE.

À l'opposée de cette stratégie de parallélisation inter-trames opérant sur des jeux de données différents, la parallélisation intra-trame vise à exploiter le parallélisme de calcul interne à l'algorithme de décodage, tel que cela est mis en œuvre dans l'intégralité des décodeurs de CCE matériels. Cette approche permet de réduire fortement la complexité mémoire, car le processus de décodage ne mémorise et ne traite qu'une trame à la fois. Cela permet aussi de réduire la latence de traitement. Cependant, d'un point de vue logiciel, la formalisation de tels décodeurs est plus complexes à réaliser que pour l'approche inter-trames.

Dans tous les cas, comme cela a été démontré dans la littérature, les formulations algorithmiques permettant de tirer parti des spécificités architecturales tel que les unités SIMD sont plus complexes à décrire. De plus, les facteurs d'accélération n'évoluent pas toujours linéairement avec la taille des unités SIMD.

Pour pouvoir utiliser l'ensemble de ces stratégies de parallélisation, il est nécessaire d'étendre le jeu d'instructions afin pour pouvoir gérer par exemple les opérations conditionnelles. En effet, lors des traitements SIMD, les mêmes traitements sont appliqués à l'ensemble des données, rendant de fait impossible l'utilisation de structure de type if permettant d'inverser par exemple le signe d'une donnée en fonction d'un calcul de parité. Pour palier à cela, des instructions spécifiques ont été ajoutées à la liste des opérations scalaires dédiées déjà identifiées. La liste exhaustive des extensions nécessaires aux descriptions SIMD est fournie dans le tableau 5.1. Ce tableau indique pour chaque instruction son usage en fonction des stratégies de parallélisation. La description des traitements réalisés par ces instructions a été placé en annexe pour des raisons de lisibilité. L'ensemble des descriptions logicielles des décodeurs CCE ont été retravaillées afin d'exploiter ces instructions et offrir des stratégies de parallélisation inter et intra-trame. Un résumé du nombre d'instructions par famille de code et schéma de parallélisation est fourni dans le tableau 5.2. Dans la suite de ce chapitre, les résultats obtenus sont présentés.

CCE	**Mnemonic**	**SISD** (int8)	**SIMD Inter** (int8x4)	**SIMD Intra** (int8x4)
LDPC	i8_abs_pi8	✓	✓	✓
	i8_add_sat127_pi8	✓	✓	✓
	i8_invB_Aneq1_pi8	✓	✓	
	i8_xorA_signB_pi8	✓	✓	
	i8_cmpeq_pi8	✓	✓	✓
	i8_max_pi8	✓	✓	✓
	i8_min_pi8	✓	✓	
	i8_sub_sat127_pi8	✓	✓	✓
	i8_sign_pi8		✓	
	i8_cpysign_pi8			✓
	i8_signextract_pi8			✓
polaire	i8_add_sat127_pi8	✓	✓	✓
	i8_sub_sat127_pi8	✓	✓	✓
	i8_Fx_pi8	✓	✓	✓
	i8_Rx_pi8	✓	✓	✓
	i8_clrA_Bneq0_pi8		✓	✓
	i8_setMask_Aeq1		✓	✓
+ F-SC	i16_add_accA_loB_pi8		✓	
	i16_add_accA_hiB_pi8		✓	
	i8_sign_pi16		✓	
	i8_Hxor_pi8			✓
	i8_Hadd_pi8			✓
LDPC-NB	u8_minu_pu8	✓	✓	✓
	u8_addu_sat64_pu8	✓	✓	✓
	u8_subu_sat64_pu8	✓	✓	✓
	u8_cmpge_pu8		✓	
	u8_Aandb_lo8B_pu8		✓	
	u8_maxu_pu8		✓	
	u8_hminu_pu8			✓
turbo	i8_max_pi8	✓	✓	✓
	i8_scale_pi8	✓	✓	
	i8_add_pi8		✓	✓
	i8_sub_pi8		✓	✓
	i8_add_srl_pi8		✓	
	i8_srl1_pi8		✓	
	i8_sign_pi8		✓	
	i8_shuffle_pi8			✓
	i8_srl_pi8			✓
	i8_min_pi8			✓

Table 5.1 – Liste des instructions spécifiques avec Mnemonic et compabilité SISD/SIMD.

5.2 Expérimentations menées sur des décodeurs SIMD inter-trames

L'évaluation conduite s'est d'abord portée sur le schéma de parallélisation inter-trames. Ce choix est lié au fait que les modifications à apporter aux descriptions logicielles sont moins conséquentes que de basculer vers un schéma de parallélisation intra-trame. Les cœurs de processeur RISC-V étudiés étant des cœurs 32 bits, il était ainsi naturel de sélectionner un format SIMD de type *int8_t* $\times 4$ sachant que les décodeurs manipulent

CCE	**SISD** (int8)	**SIMD Inter** (int8x4)	**SIMD Intra** (int8x4)
LDPC	8	9	7
polaire	4	6	6
polaire F-SC	4	9	8
LDPC-NB	3	6	4
turbo	2	7	6

Table 5.2 – Synthèse du nombre d'instructions identifiées par famille de CCE.

naturellement des données sur 8 bits. Ce format SIMD permet le traitement de 4 données en parallèle par cycle d'horloge, et par voie de conséquence autorise 4 trames à être décodées de manière simultanée. Les opérations appliquées sur les 4 trames étant identiques, il est alors possible d'assurer une utilisation maximale des ressources de calcul.

Les résultats expérimentaux concernant le temps d'exécution des différents décodeurs sont reportés dans la table 5.3 et 5.4 et 5.5. Ces résultats sont comparés à ceux obtenus précédemment lorsque les instructions spécialisées manipulaient des instructions scalaires (SISD) (❷ *SISD*) et également avec les décodeurs de base sans l'aide d'instructions spécialisées ❶ *Baseline*. Les données présentées montrent qu'en moyenne, les instructions SIMD ajoutées permettent de réduire de 75% le temps de décodage des bits par rapport à la solution incorporant nos instructions SISD. Les débits sont également considérablement améliorés lorsqu'ils sont comparés avec la baseline et les instructions SISD. Ce constat est cohérent avec la stratégie de parallélisation employée, même s'il est difficile d'expliquer précisément les raisons des légères disparités entre les versions d'un même décodeur sur les différentes architectures.

Si l'on analyse dans un premier temps les codes CCE et que l'on regarde les décodeurs LDPC binaires et non binaires, le problème ne se pose pas. En effet, les gains avoisinent 75% et les débits sont améliorés d'un facteur supérieur à 6× sur toutes les architectures. Cela s'explique par le fait que ces algorithmes sont composés de nids de boucles et sont donc réguliers dans leur flot d'exécution.

Les codes polaires utilisent la récursivité durant le décodage, engendrant des appels de fonction non accélérés par les instructions SIMD introduites. En conséquence, les gains observés sont moindres, variant de ≈ 66% (**IBEX**, **RISCY**) à 70% (**SCR1**) en réduction de cycles/bit et de 4.4× à 5.3× en débit. Les codes polaires avec l'algorithme Fast-SC présentent des gains moindres, variant de 63% à ≈ 59% en cycles/bit également, et les

			LDPC OMS	SC	**polaire** F-SC	**LDPC-NB** MS	**turbo** MLM
PicoRV32	Cycles/Bit	SISD	484	91	40	11 234	202
		SIMD	-76.9%	-68.2%	-62.3%	-72.5%	-70.8%
IBEX	Cycles/Bit	SISD	199	35	16	4 245	63
		SIMD	-76.1%	-65.9%	-58.9%	-72.2%	-70%
SCR1	Cycles/Bit	SISD	172	34	15	4 226	61.3
		SIMD	-76.7%	-70.6%	-63.1%	-72.6%	-70.5%
RISCY	Cycles/Bit	SISD	176	28	13	948	54
		SIMD	-76.4%	-65.6%	-58.8%	-72.4%	-70%

Table 5.3 – Réduction du nombre de cycles d'horloge nécessaire au décodage d'un bit grâce à l'usage des instructions SIMD dédiées - Stratégie de parallélisation inter-trames.

débits varient de 4.1× à 4.9×. En effet, dans cet algorithme, la simplification de séquences d'instructions est plus ardue du fait de nœuds très spécifiques (*REP, SPC*). Ces nœuds comportent des opérations qui ne sont pas réductibles à des instructions spécialisées. Par exemple, les nœuds *SPC* nécessitent une recherche de minima avec leur position associée dans leurs versions inter-trames. Ces opérations conditionnelles sont plus simples à approcher avec une donnée unique, mais deviennent plus complexes à implanter avec 4 données distinctes.

Les turbo codes présentent une réduction des cycles nécessaires au décodage d'un bit de donnée en moyenne de 70%, conformément aux résultats attendus avec des débits variant de 3.8× à 4.5×.

En conclusion, d'un point de vue applicatif on peut noter qu'indépendamment de la famille de code testée, les versions inter-trames SIMD déployées sur les cœurs offrent systématiquement une réduction significative de $\approx 70\%$ en nombre de cycles nécessaires au décodage d'un bit et des débits augmentant d'un facteur de 5× en moyenne.

L'utilisation d'instructions de type SIMD nécessite l'utilisation d'opérateurs matériels plus complexes. En effet, 4 données différentes doivent subir le même traitement à chaque cycle d'horloge, impliquant une augmentation en théorie d'un facteur 4× de leur complexité matérielle. De plus, afin de décrire les algorithmes de décodage en utilisant une parallélisation inter-trames, de nouvelles instructions ont dû être introduites, augmentant la complexité globale des extensions proposées. La figure 5.6 fournie des informations concernant l'augmentation relative de la complexité matérielle des cœurs RISC-V, tandis que la figure 5.2 offre une représentation graphique des surcoûts absolus et la fréquence de fonctionnement maximale des différentes extensions. Ces chiffres ont été obtenus de

			LDPC OMS	SC	**polaire** F-SC	**LDPCNB** MS	**turbo** MLM
PicoRV32	❶ Baseline	cycles	217172	1333324	676218	3476435	2503511
	❷ SISD	cycles	131853	753144	332693	2875953	2133442
	❸ SIMD INTER	cycles	122462	916029	502106	3166507	2488956
	Gain (❶→❷)	cycles	85319	580180	343525	600482	370069
	Gain (❶→❷)	(%)	39.3%	43.5%	50.8%	17.3%	14.8%
	Gain (❶→❸)	cycles	94710	417295	174112	309928	14555
	Gain (❶→❸)	(%)	43.6%	31.3%	25.7%	8.9%	0.6%
	Débit ❶	Kbits/s	429	2106	4153	25	1696
	Débit ❷	Kbits/s	659	3357	7600	31	1948
	Débit ❸	Kbits/s	2657	11253	20477	102	6367
	Accélération (❶→❷)	8 bits	1.5×	1.6×	1.8×	1.2×	1.1×
	Accélération (❶→❸)	32 bits	6.2×	5.3×	4.9×	4.0×	3.8×
IBEX	❶ Baseline	cycles	84966	516441	261090	1390238	1018910
	❷ SISD	cycles	54340	288594	138544	1086873	782778
	❸ SIMD INTER	cycles	52025	394196	227587	1206494	938735
	Gain (❶→❷)	cycles	30626	227847	122546	303365	236132
	Gain (❶→❷)	(%)	36.0%	44.1%	46.9%	21.8%	23.2%
	Gain (❶→❸)	cycles	32941	122245	33503	183744	80175
	Gain (❶→❸)	(%)	38.8%	23.7%	12.8%	13.2%	7.9%
	Débit ❶	Kbits/s	317	1571	3107	18	1204
	Débit ❷	Kbits/s	515	2942	6129	24	1615
	Débit ❸	Kbits/s	2121	8333	14617	86	5402
	Accélération (❶→❷)	8 bits	1.6×	1.9×	2.0×	1.3×	1.3×
	Accélération (❶→❸)	32 bits	6.7×	5.3×	4.7×	4.7×	4.5×

Table 5.4 – Comparaison des performances des cœurs **IBEX** et **PicoRV32** usant d'instructions à 2 entrées, configuration SIMD inter-trame avec la fréquence de fonctionnement maximale par implantation.

manière similaire à ceux présentés dans la partie expérimentale relative aux instructions scalaires (SISD). De manière prévisible, l'augmentation de la complexité matérielle est plus marquée que lors des expérimentations pour les instructions scalaires. L'introduction de ces nouvelles instructions dans le cœur le moins complexe (**PicoRV32**) aboutie à une augmentation de la complexité matérielle de $\approx 50\%$ pour chacun des jeux d'extensions proposés. Ces valeurs sont liées à la faible empreinte "silicium" de sa version originale. Pour les cœurs RISC-V plus complexes, tels que les cœurs **IBEX** et **SCR1**, l'augmentation relative de la complexité est plus limitée, elle varie de +9.4% à +13.7% pour ce premier et de +7.6% à +10.1% pour le second. L'augmentation la moins marquée est observable pour le cœur le plus complexe, le **RISCY** pour lequel la variation est seulement de $\approx 4\%$ au maximum.

L'introduction des nouvelles instructions impacte aussi la fréquence de fonctionne-

			LDPC OMS	SC	**polaire** F-SC	**LDPCNB** MS	**turbo** MLM
SCR1	❶ Baseline	cycles	76775	397790	218120	1310272	927495
	❷ SISD	cycles	47995	274793	125037	1081939	758988
	❸ SIMD INTER	cycles	44000	343892	184528	1187911	897056
	Gain (❶→❷)	cycles	28780	122997	93083	228333	168507
	Gain (❶→❷)	(%)	37.5%	30.9%	42.7%	17.4%	18.2%
	Gain (❶→❸)	cycles	32775	53898	33592	122361	30439
	Gain (❶→❸)	(%)	42.7%	13.5%	15.4%	9.3%	3.3%
	Débit ❶	Kbits/s	362	2107	3842	20	1366
	Débit ❷	Kbits/s	586	3021	6639	24	1661
	Débit ❸	Kbits/s	2507	9735	18014	86	5487
	Accélération (❶→❷)	8 bits	1.6	1.4	1.7	1.2	1.2
	Accélération (❶→❸)	32 bits	6.9	4.6	4.7	4.3	4.0
RISCY	❶ Baseline	cycles	77678	348107	192935	1103396	890712
	❷ SISD	cycles	48096	228581	113843	878938	674609
	❸ SIMD INTER	cycles	45323	314875	187693	970983	810365
	Gain (❶→❷)	cycles	29582	119526	79092	224458	216103
	Gain (❶→❷)	(%)	38.1%	34.3%	41.0%	20.3%	24.3%
	Gain (❶→❸)	cycles	32355	33232	5242	132413	80347
	Gain (❶→❸)	(%)	41.7%	9.5%	2.7%	12.0%	9.0%
	Débit ❶	Kbits/s	105	706	1274	7	417
	Débit ❷	Kbits/s	170	1075	2159	9	551
	Débit ❸	Kbits/s	720	3122	5237	32	1834
	Accélération (❶→❷)	8 bits	1.6×	1.5×	1.7×	1.3×	1.3×
	Accélération (❶→❸)	32 bits	6.9×	4.4×	4.1×	4.5×	4.4×

Table 5.5 – Comparaison des performances des cœurs **SCR1** et **RISCY** usant d'instructions à 2 entrées, configuration SIMD inter-trame avec la fréquence de fonctionnement maximale par implantation.

ment maximale des différents cœurs, tel que le démontrent les données rapportées dans le tableau 5.6 et la figure 5.2. Le cœur le plus impacté est le **PicoRV32** avec une baisse moyenne de $\approx 10\%$ de sa fréquence de fonctionnement, ce qui est $1.5\times$ plus important que précédemment. Pour le cœur **IBEX**, l'impact est quand à lui plus faible ou équivalent en fonction de l'extension considérée, tandis que pour le cœur **RISCY**, aucun impact n'a été observé. En effet, ce cœur est limité en fréquence par d'autres composants de son architecture. L'ensemble de ces résultats démontrent que malgré l'augmentation de la complexité matérielle des architectures et la baisse de la fréquence de fonctionnement, les instructions SIMD proposées permettent d'améliorer notablement les performances globales des décodeurs logiciels déployés sur les cœurs RISC-V.

	BASE	LDPC OMS	POLAIRE SC	POLAIRE Fast-SC	LDPC-NB MS	TURBO Max-Log MAP
Nb. Insn Cst.		9	6	9	6	7
PicoRV32	978 LUTs	+44.7%	+41.8%	+48.0%	+43.3%	+37.7%
	567 FFs	+3.4%	+2.3%	+3.0%	+2.3%	2.6%
	303 MHz	-12.8%	-8.2%	-8.5%	-7.8%	-6.7%
IBEX	2 446 LUTs	+12.2%	+12.4%	+13.1%	+13.7%	+9.4%
	883 FFs	0.0%	0.0%	0.0%	0.0%	0%
	100 MHz	+2.4%	+4.7%	+1.2%	+2.9%	+3.4%
SCR1	3 986 LUTS	+10.1%	+8.5%	+8.5%	+9.8%	+7.6%
	2 399 FFs	+0.1%	0.0%	0.0%	0.0%	0.0%
	102MHz	-0.9%	-0.1%	-0.1%	-2.8%	-2.9%
RISCY	10 302 LUTs	+3.1%	+3.8%	+4.1%	+4%	+3.2%
	3 033 FFs	0%	0%	0%	0%	0%
	30 MHz	0%	0%	0%	0%	0%

Table 5.6 – Impact des instructions SIMD dédiées sur la complexité matérielle et la fréquence de fonctionnement des cœurs RISC-V - Stratégie de parallélisation inter-trames

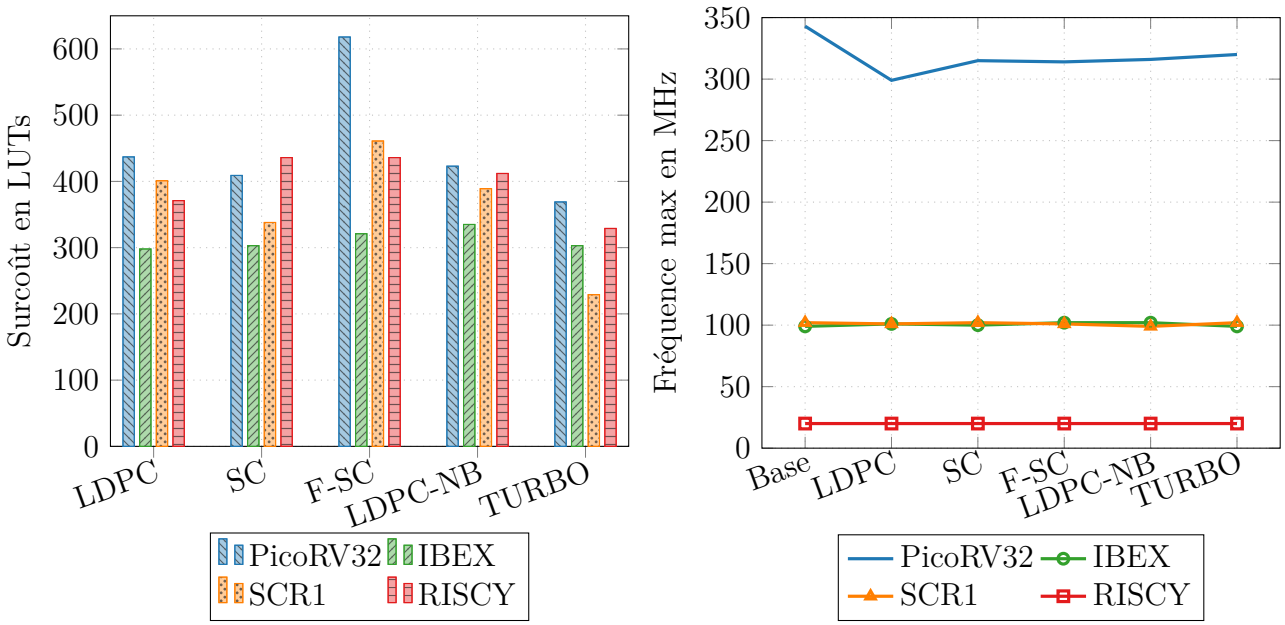

Figure 5.2 – Impact de l'ajout des instructions spécialisées : surcoût en LUTs et fréquence de fonctionnement.

5.3 Expérimentations menées sur des décodeurs SIMD intra-trame

Cette dernière section présente les gains obtenus lorsque les extensions SIMD sont utilisées pour accélérer des décodeurs utilisant une stratégie de parallélisation intra-trames. Par opposition à la parallélisation inter-trames, l'approche intra-trame utilise le parallélisme interne à l'algorithme pour minimiser la latence de décodage. La mise en œuvre de cette stratégie de parallélisation nécessite toutefois l'usage d'instructions SIMD différentes de celles employées pour réaliser de l'inter-trames comme cela est détaillé dans le tableau 5.1. À partir des instructions identifiées et intégrées dans les cœurs de processeur RISC-V, nous avons développé des versions spécifiques des codes sources des décodeurs logiciels dans des formes proches de celles décrites dans [161, 166, 167]. Ces décodeurs ont été évalués avec les cœurs RISC-V déployés sur circuit FPGA. Les temps d'exécution et les débits maximum ainsi obtenus ont ensuite été comparés. Les données issues de ces expérimentations sont résumées dans le tableau 5.7 et 5.8.

Les résultats expérimentaux mettent tout d'abord en évidence une accélération des décodeurs spécialisés dans le décodage des codes polaires et des turbo codes. Les gains en termes de temps d'exécution sont pour ces familles de codes compris dans l'intervalle de $\approx$ 58% à 68%. Ces gains sont calculés par rapport aux versions scalaires ❷ *SISD* développées précédemment. Les gains relatifs aux versions de base ❶*Baseline* sont également présentés. Concernant les décodeurs LDPC, les limitations architecturales des cœurs RISC-V employés réduisent drastiquement l'intérêt de ce type d'implantation.

En effet, le processus de décodage LDPC réalise des accès 32 bits non alignés à la mémoire. Ce type d'accès, relativement atypique, n'est pas supporté par la majorité des cœurs RISCV testés, à l'exception du cœur **IBEX**.

Pour les décodeurs LDPC non binaires, l'accélération est faible en comparaison de son implantation inter-trames, avec une réduction du temps d'exécution de $\approx$ 5% à 6.8%. Ces gains limités sont liés à la nature de l'algorithme avec un schéma intra-trames, car il est difficile d'extraire des instructions spécialisées opérant sur 32 bits. À titre d'exemple, les fonctions de calcul utilisées pour le décodage des nœuds de parités sont dépendants d'accès en mémoire ordonnés pour chaque donnée de 8b. Les fonctions de conversion entre le domaine naturel et le domaine des $\mathbb{GF}$ sont également problématiques. Ainsi, sauf pour certaines sections du décodeur, extraire des instructions compatibles avec un alignement des données en mémoire bénéfique à l'exécution intra-trame est impossible. Si l'on s'in-

			SC	Polaire FSC	LDPCNB MS	Turbo MLM
PicoRV32	❶ Baseline	cycles	1333324	676218	3476435	2503511
	❷ SISD	cycles	753144	332693	2875953	2133442
	❸ SIMD INTRA	cycles	327502	133244	2713146	788086
	Gain❶→❷	cycles	580180	343525	600482	370069
	Gain❶→❷	%	43.5%	50.8%	17.3%	14.8%
	Gain❶→❸	cycles	1005822	542974	763289	1715425
	Gain❶→❸	%	75.4%	80.3%	22.0%	68.5%
	Débit❶	Kbits/s	2106	4153	25.2	1695
	Débit❶	Kbits/s	3357	7599	30.5	1947
	Débit❶	Kbits/s	7797	18704	25	5049
	Accel❶→❷	8bits	1.6	1.8	1.2	1.1
	Accel❶→❸	32bits	3.7	4.5	1.0	3.0
IBEX	❶ Baseline	cycles	516441	261090	1390238	1018910
	❷ SISD	cycles	288594	138544	1086873	782778
	❸ SIMD INTRA	cycles	46379	126879	1025576	286696
	Gain❶→❷	cycles	227847	122546	303365	236132
	Gain❶→❷	%	44.1%	46.9%	21.8%	23.2%
	Gain❶→❸	cycles	470062	134211	364662	732214
	Gain❶→❸	%	91.0%	51.4%	26.2%	71.9%
	Débit❶	Kbits/s	1571	3107	18	1204
	Débit❶	Kbits/s	2942	6129	24	1615
	Débit❶	Kbits/s	6711	13560	25	4571
	Accel❶→❷	8bits	1.9	2.0	1.3	1.3
	Accel❶→❸	32bits	4.3	4.4	1.4	3.8

Table 5.7 – Comparaison des performances des cœurs usant d'instructions à 2 entrées, configuration SIMD intra-trame avec la fréquence de fonctionnement maximale par implantation.

téresse maintenant à l'évolution de la complexité matérielle des cœurs enrichis, dont les valeurs sont fournies dans le tableau 5.9, on peut remarquer que les extensions nécessaires pour assurer une parallélisation intra-trame ont un coût équivalent à celles utilisées pour une parallélisation inter-trame. Les variations en termes de complexité matérielle se situent dans l'intervalle $[-3\% ; +8\%]$ en fonction des cœurs et des CCE. Les disparités dans ces gains et dans ces pertes sont difficilement explicables et semblent être uniquement liées aux choix réalisés par les outils de synthèse logique. Par exemple, pour l'extension dédiée au décodage *Fast-SC*, elle est $\approx 5\%$ moins onéreuse sur le cœur **PicoRV32** tandis que dans le même temps, elle est $\approx 7\%$ plus complexe sur le cœur **IBEX**. Ces observations liées à l'augmentation de la complexité matérielle entre ces deux types d'extension se retrouvent lorsque l'on analyse l'impact que peuvent avoir les extensions sur la fréquence maximale de fonctionnement des cœurs. Les ordres de grandeur sont conservés malgré quelques disparités.

			SC	Polaire FSC	LDPCNB MS	Turbo MLM
SCR1	❶ Baseline	cycles	397790	218120	1310272	927495
	❷ SISD	cycles	274793	125037	1081939	758988
	❸ SIMD INTRA	cycles	115954	49161	1018811	283452
	Gain❶→❷	cycles	122997	93083	228333	168507
	Gain❶→❷	%	30.9%	42.7%	17.4%	18.2%
	Gain❶→❸	cycles	281836	168959	291461	644043
	Gain❶→❸	%	70.9%	77.5%	22.2%	69.4%
	Débit❶	Kbits/s	2107	3842	20	1366
	Débit❶	Kbits/s	3020	6638	24	1660
	Débit❶	Kbits/s	7218	17391	26	4334
	Accel❶→❷	8bits	1.4	1.7	1.2	1.2
	Accel❶→❸	32bits	3.4	4.5	1.3	3.2
RISCY	❶ Baseline	cycles	348107	192935	1103396	890712
	❷ SISD	cycles	228581	113843	878938	674609
	❸ SIMD INTRA	cycles	95375	44752	832577	216187
	Gain❶→❷	cycles	119526	79092	224458	216103
	Gain❶→❷	%	34.3%	41.0%	20.3%	24.3%
	Gain❶→❸	cycles	252732	148183	270819	674525
	Gain❶→❸	%	72.6%	76.8%	24.5%	75.7%
	Débit❶	Kbits/s	706	1274	7	417
	Débit❶	Kbits/s	1075	2159	9	551
	Débit❶	Kbits/s	2577	5492	9	1719
	Accel ❶→❷	8bits	1.5	1.7	1.3	1.3
	Accel ❶→❸	32bits	3.6	4.3	1.3	4.1

Table 5.8 – Comparaison des performances des cœurs **SCR1** et **RISCY** usant d'instructions à 2 entrées, configuration SIMD intra-trame avec la fréquence de fonctionnement maximale par implantation.

L'ensemble de ces résultats expérimentaux mettent en évidence que les extensions dédiées à la parallélisation intra-trame offrent, pour les turbo codes et les codes polaires, des facteurs d'accélération substantiels ($> 50\%$ du temps d'exécution) avec un impact limité sur les caractéristiques des cœurs si l'on omet le cœur **PicoRV32**.

5.4 Synthèse de la partie expérimentale

Cette dernière sous-section se propose de synthétiser les différents résultats présentés précédemment afin de conclure sur l'intérêt des extensions proposées. Le tableau 5.10 récapitule les données issues des différentes expérimentations. Les gains minimums, maximums et moyens en termes de réduction de cycles d'horloges nécessaires pour décoder 1 bit d'information sont fournis pour les 4 architectures testées. Il est important de rappeler que (a) les gains pour les versions SISD sont relatifs aux codes logiciels sans instruction

	BASE	LDPC	POLAIRE		LDPC-NB	TURBO
		OMS	SC	Fast-SC	MS	Max.Log-Map
Nb. Insn Cst.		7	6	8	4	6
PicoRV32	978 LUTs	+44.4%	+41.9%	+43.3%	+37.2%	+54.3%
	567 FFs	+3.7 %	+2.3%	+2.6%	+1.4%	+2.6%
	303 MHz	-9.2 %	-9.0%	-7.8%	-21.6%	-6.3%
IBEX	2 446 LUTs	+11.0%	+16.3%	+20.5%	8.9%	+22.4%
	883 FFs	0.0%	0.0%	0.0%	0.0%	+0.8%
	100 MHz	+3.4%	+5.0%	-2.2%	+3.0%	+6.9%
SCR1	3 986 LUTs	+10.1%	+8.5%	+11.6%	+5.2%	+11.0%
	2 399 FFs	0.0%	0.0%	0.0%	0.0%	0.0%
	102 MHz	-2.6%	-0.1%	-0.8%	-0.7%	-3.0%
RISCY	10 302 LUTs	+4.6%	+3.8%	+4.5%	+3.4%	+5%
	3 033 FFs	0.0%	0.0%	0.0%	0.0%	0.0%
	30 MHz	0.0%	0.0%	0.0%	0.0%	0.0%

Table 5.9 – Impact des instructions SIMD dédiées sur la complexité matérielle et la fréquence de fonctionnement des cœurs RISC-V - Stratégie de parallélisation intra-trame

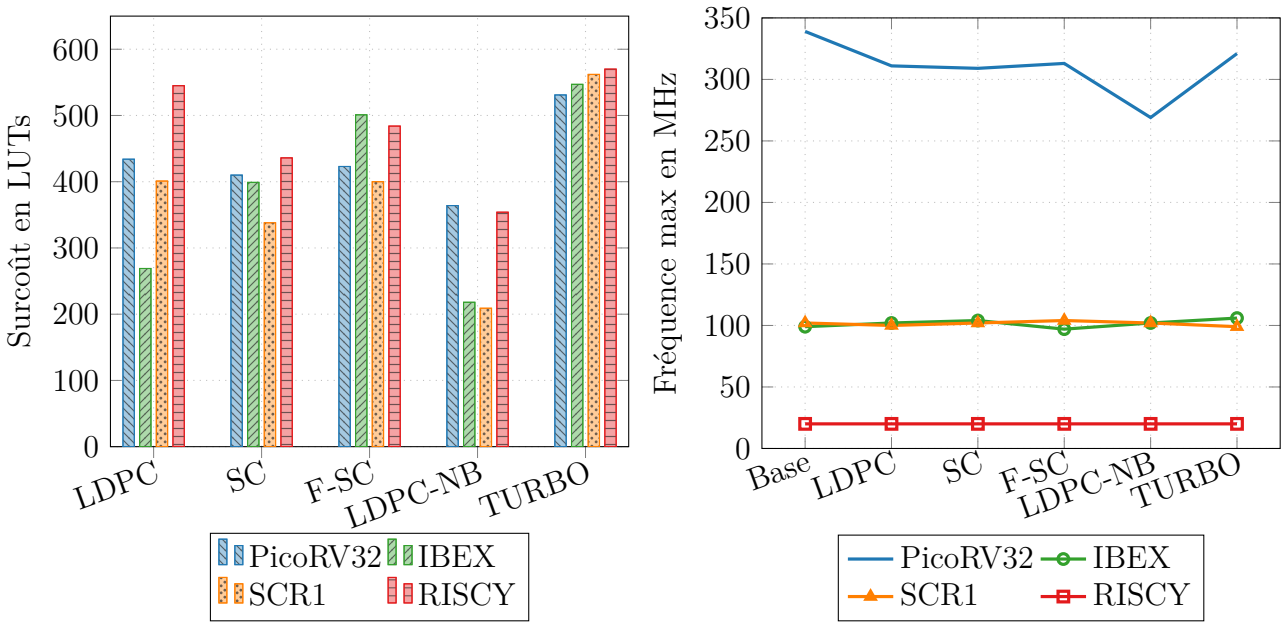

Figure 5.3 – Impact de l'ajout des instructions SIMD intra-trame : surcoût en LUTs et fréquence de fonctionnement

spécialisée, tandis que les gains inter/intra-trame ont été calculés par rapport aux versions SISD ; (b) pour les les versions inter-trames, le gain en terme de temps d'exécution permet une augmentation du débit des décodeurs, mais ne réduit pas la latence de décodage par rapport aux implantations SISD, tandis qu'avec la parallélisation intra-trame,

		Réduction (cycles/bit en %)			**Surcoût matériel** (LUTs en %)		
CCE	Implantation	min	moy	max	min	moy	max
LDPC	SISD	36.0%	37.7%	39.3%	1.8%	12.5%	34.0%
OMS	INTER	76.1%	76.6%	77.1%	3.1%	17.5%	44.7%
	INTRA	14.7%	14.7%	14.7%	4.6%	17.5%	44.4%
polaire	SISD	30.9%	38.2%	44.1%	2.1%	5.1%	12.3%
SC	INTER	65.6%	67.4%	69.6%	3.8%	16.6%	41.8%
	INTRA	56.0%	57.2%	58.3%	3.8%	17.6%	41.9%
polaire	SISD	41.0%	45.4%	50.8%	2.1%	5.1%	12.3%
F-SC	INTER	58.8%	60.8%	63.1%	4.1%	19.2%	48.0%
	INTRA	57.7%	59.8%	60.7%	4.5%	24.6%	63.2%
LDPC NB	SISD	17.3%	19.2%	21.8%	0.8%	5.1%	12.5%
MS	INTER	72.2%	72.4%	72.6%	4.0%	17.7%	43.3%
	INTRA	5.3%	5.6%	5.8%	3.4%	13.7%	37.2%
turbo	SISD	14.8%	20.1%	24.3%	0.6%	4.7%	12.6%
Max-Log-Map	INTER	70.0%	70.3%	70.8%	3.2%	14.5%	37.7%
	INTRA	62.7%	64.3%	68.0%	5.0%	23.2%	54.3%

Table 5.10 – Impacts des instructions sélectionnées en termes de réduction de cycles d'horloge par bit décodé et le surcoût en LUT

l'augmentation du débit s'accompagne d'une réduction de la latence du même ordre de grandeur. Des colonnes similaires fournissent les métriques relatives aux surcoûts en LUT, cependant toutes ces valeurs doivent se lire relativement aux cœurs RISC-V originaux.

Ce tableau met en relief les gains obtenus pour chacune des familles de codes correcteurs d'erreurs considérés. Il est à noter que les gains sont substantiels en SISD lorsque l'on observe le temps d'exécution. Cependant, tous les algorithmes de décodage n'en bénéficient pas de manière équivalente. Par exemple, les turbo codes, avec seulement 2 instructions spécialisées possibles, ne gagnent que $\approx$ 15 20%. Le constat est similaire pour le décodage des codes LDPC-NB, qui avec uniquement 3 instructions spécialisées ne voit son temps d'exécution réduit que de $\approx$ 20%. En fonction des cœurs RISC-V considérés, le surcoût matériel induit par l'introduction de ces nouvelles instructions est bien inférieur aux gains apportés : de $\approx 2\times$ à $\approx 8\times$.

Les versions inter-trames (INTER) qui traitent en interne 4 données de 8 bits en parallèle, réduisent le temps de décodage des bits d'information de 58% et 77% par rapport aux décodeurs SISD utilisant des instructions spécifiques. Les gains ainsi obtenus permettent d'augmenter le débit binaire des décodeurs. Avec cette stratégie de parallélisation et aux instructions spécifiques ajoutées, l'ordre de grandeur des gains observés est

similaire pour les 4 cœurs RISC-V et les différents algorithmes de décodage. L'augmentation de la complexité matérielle des cœurs de processeur RISC-V est plus conséquente que lors des expérimentations SISD, cependant les jeux d'instructions étant différents, l'augmentation moyenne du nombre de LUT post-implantation varie de 1.5× à 4×. À l'exception du cœur **PicoRV32** pour lequel l'augmentation de la complexité matérielle atteint environ 50%, pour les autres cœurs cette augmentation est beaucoup plus faible (ex. $\leq 4\%$ pour le cœur **RISCY**).

Les expérimentations ciblant un schéma de parallélisation intra-trame pour les algorithmes de décodage offrent des résultats plus mitigés. L'intérêt de ce schéma de parallélisation est lié à la réduction de la latence de décodage. Il s'inspire des stratégies mises en œuvre dans les architectures matérielles. D'un point de vue théorique, les gains mesurés devraient être similaires à ceux obtenus pour les expérimentations INTER car 4 données 8 bits sont ici aussi traitées en parallèle. Cependant, comme le démontrent les résultats rapportés dans le tableau 5.10, les facteurs d'accélération obtenus pour les codes LDPC et LDPC-NB sont bien inférieurs à ceux obtenus par leurs homologues INTER. Ce constat est lié aux formulations des algorithmes de décodage qui limitent l'exploitation du parallélisme et impactent négativement sur la conception des instructions spécialisées. Les résultats sont meilleurs pour les autres familles de CCE utilisées, pour lesquelles des réductions de 57% à 64% du temps d'exécution sont mesurées. Le coût matériel des instructions spécialisées rajoutées dans ce contexte s'avère, en outre plus onéreux que pour la parallélisation inter-trames (jusqu'à 10%).

Pour conclure, l'ensemble de ces résultats expérimentaux démontrent l'intérêt des extensions proposées pour l'accélération du décodage des CCE actuels. Afin d'améliorer les performances ainsi obtenues, différentes pistes d'amélioration existent, tel que cela sera détaillé dans le prochain chapitre.

5.5 Conclusion

Ce chapitre permet de conclure sur l'efficacité de l'intégration d'instructions de type SIMD, pour l'accélération des CCE lorsque appliqués à leurs algorithmes de décodage. En effet, avec les résultats obtenus par les expérimentations menées, les débits augmentent en concordance avec la réduction de la latence des décodeurs étudiés.

On retrouve là une philosophie qui a abouti par exemple à l'intégration d'extensions SSEx, AVX sur processeur X86_64. Toutefois, cette amélioration des performances en-

gendre un surcoût matériel dont l'impact varie en fonction de la complexité initiale du cœur tel que cela est mis en évidence dans le tableau 5.10. Afin de minimiser cette augmentation de la complexité matérielle, il serait possible de pousser encore plus loin l'intégration de nos extensions dans les architectures cibles. Cela pourrait se faire en réutilisant par exemple tout ou partie des ressources de l'ALU, mais on perdrait alors le principe de *généricité*, sur lequel repose une grande partie de l'intérêt de la solution étudiée.

Cette première étude a été volontairement limitée à l'utilisation de jeux d'instructions à deux registres sources. Cette caractéristique a été initialement maintenue pour permettre une plus forte adaptabilité des instructions identifiées pour des ISA de cœur des calculs généralistes. Cependant, il est intéressant d'étendre cette première étude, avec des architectures proposant l'utilisation d'instruction à trois registres sources. Cette nouvelle approche est présentée dans le chapitre suivant ; nous y analyserons les avantages et les inconvénients des extensions que nous avons proposées.

Chapitre 6

Extensions spécialisées avec 3 opérandes

Ce chapitre étend les travaux présentés dans le chapitre précédent, utilisant des opérations classiques à deux opérandes en entrées. Comme cela a été mentionné, ce choix, parfaitement valide au demeurant, reste la principale limitation bridant la conception d'instructions adaptées au décodage de CCE. Dans cette partie, nous évaluons les gains qu'il est possible d'atteindre en levant cette contrainte architecturale. En effet, des cœurs RISC-V plus complexes permettent l'accès à 3 données pour exécuter, par exemple, des opérations de type MAC (Multiply And Accumulate). La première partie du chapitre expose les différentes caractéristiques architecturales de ces cœurs pouvant opérer sur trois registres d'entrée simultanément. Nous présentons également le cœur RISC-V sélectionné et la stratégie mise en œuvre pour insérer nos instructions dans l'ISA RISC-V.

Dans un second temps, nous présentons les différentes extensions mises au point et nous mettons en valeur les différences avec les résultats obtenus dans le chapitre précédent. Pour terminer, une évaluation de l'impact de ces nouvelles extensions est réalisée. Les gains en termes de temps d'exécution sont exposés et mis en perspective des surcoûts matériels engendrés. Ces expérimentations, comparables à celles réalisées pour les instructions à 2 registres, sont étendues pour les configurations SIMD sur 64 bits.

Contents

6.1 Modèle d'architecture ciblé

Les cœurs de processeurs généralistes sont en principe conçus de manière à pouvoir délivrer 2 opérandes par cycle d'horloge à l'unité arithmétique et logique (UAL). Ce choix de conception est intrinsèquement lié à l'ISA des processeurs de type RISC où les instructions exécutées manipulent au plus 2 deux opérandes provenant des registres et/ou une valeur immédiate contenue dans un champ de l'instruction. Cependant, certains cœurs de processeurs pensés pour exécuter plus efficacement des applications de traitement du signal possèdent un troisième port de lecture pour la file de registres, afin de permettre l'exécution d'instructions de type MAC (Multiply And Accumulate). L'accès à une troisième opérande permet l'intégration d'opérations plus complexes au sein de l'UAL, dans le but de réduire le nombre d'instructions à réaliser, et par voie de conséquence, de diminuer le temps d'exécution des applicatifs ainsi que la taille de leur binaire. Ces nouvelles opportunités ont toutefois un impact négatif sur la complexité matérielle de l'architecture, car elles nécessitent l'ajout d'un réseau de multiplexage supplémentaire dans la file de registres ou bien une duplication des bancs mémoire dans le but de délivrer cette donnée supplémentaire à l'UAL. De plus, un format d'instruction supplémentaire doit être géré dans le décodeur d'instructions pour autoriser l'usage de cet troisième opérande. L'augmentation de la complexité matérielle induite et le faible usage d'une troisième opérande dans l'UAL pour un bon nombre d'applications généralistes, limitent l'intégration de cette approche la majorité des cœurs RISC-V.

Dans le cadre de l'étude présentée dans les chapitres précédents, nous avons identifié et intégré des instructions possédant au maximum deux opérandes en entrée, notée dans ce chapitre **2R**. Cette contrainte architecturale a limité certains choix de conception et n'a offert que peu d'opportunités pour exploiter au mieux les codes LDPC-NB et les turbo codes. Dans ce chapitre, cette contrainte est allégée en considérant que l'UAL, et donc les instructions, sont en mesure de manipuler trois données/registres (*rS1, rS2, rS3*). La contrainte sur la sortie de l'UAL, elle reste inchangée, elle ne peut produire au maximum qu'une seule donnée (*rD*).

Dans cette section, nous présentons une évolution des extensions conçues afin d'accélérer l'exécution des algorithmes de décodage des codes correcteurs d'erreurs. L'utilisation

Champ	**Bits**
Opcode	[6:0]
Registre destination	[11:7]
Fonction 3	[14:12]
Registre source 1	[19:15]
Registre source 2	[24:20]
Fonction 2	[26:25]
Registre source 3	[31:27]

Table 6.1 – Format 32 bits standardisé pour les instructions de type R4 (3 registres d'entrées) dans les spécifications RISC-V : RV-FDQ

d'une troisième opérande en entrée a nécessité une remise à plat des extensions proposées précédemment, pour aboutir à une solution efficace : certaines instructions à deux entrées ont été fusionnées, et d'autres ont aussi été créées et/ou modifiées afin de maintenir un flot d'exécution cohérent.

Dans la spécification de base de l'ISA RISC-V, il n'existe pas de format standardisé permettant d'encoder directement 3 registres sources dans une instruction de type RV32/64I. Afin de rester compatible avec cette ISA et ainsi de continuer à bénéficier des environnements de compilation, nous avons décidé d'utiliser le format proposé dans les extensions RV-FDQ (simple, double et quadruple précision en virgule flottante). Ces dernières proposent un format 32 bits spécifique pour les opérations de type FMADD (Fused Multiplication-Accumulate) tel que cela est décrit dans le tableau 6.1.

Le détournement de ce format existant pour les opérations flottantes permet de réduire le surcoût matériel lié à l'introduction de nos extensions dans le décodeur d'instructions. Afin de maintenir la compatibilité des cœurs avec l'extension F, nous avons sélectionné des *opcodes* prévus pour l'ajout d'instructions spécialisées. Ces derniers sont utilisés uniquement pour des instructions spécialisées additionnelles à deux ou trois opérandes et ont pour valeur $\times 0b, \times 3b, \times 5b, \times 7b$, conformément à la spécification RISC-V. Il est à noter que les instructions manipulant seulement un ou deux registres utilisent comme précédemment le type R.

Les cœurs de RISC-V que nous avions auparavant sélectionnés possèdent des architectures internes simples et ne supportent pas par défaut les extensions F, D ou Q qui nécessitent une file de registres à 3 sorties pour exécuter les instructions de type FMA. En conséquence, leurs décodeurs d'instructions n'étaient pas prévus pour décoder les instructions de type R et de générer les signaux de contrôle adéquats. En conséquence, pour

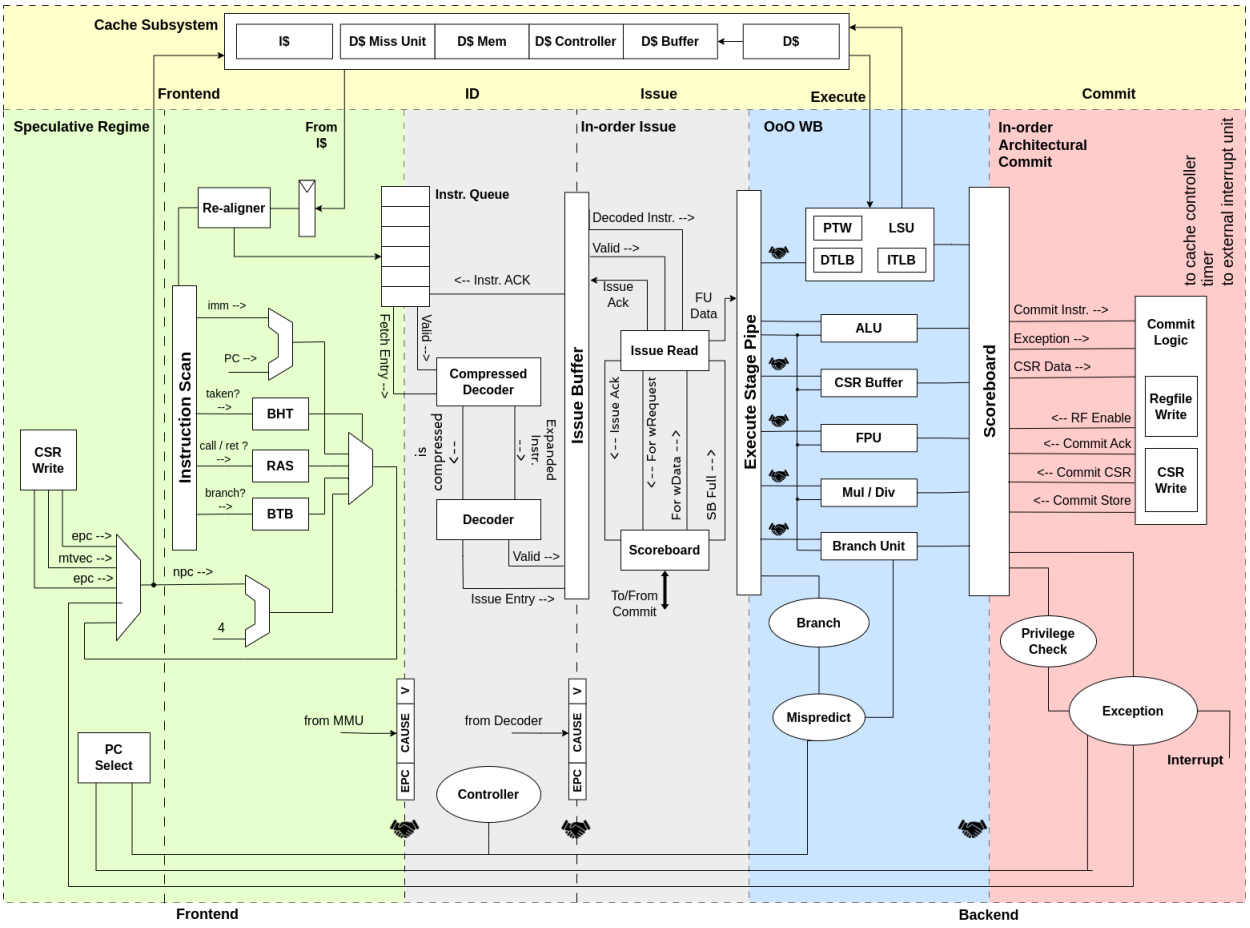

Figure 6.1 – Schéma de l'architecture du cœur CVA6.

éviter des temps d'ingénierie chronophages, nous avons décidé de changer de cible architecturale. Parmi les cœurs RISC-V *high-end* supportant ces extensions, nous pouvons citer par exemple les cœurs BOOM [205], CVA6 [206], SHAKTI [207] et ROCKET [208]. Ces derniers, en plus de couvrir un nombre important d'extensions de l'ISA possèdent aussi un mode de fonctionnement 64 bits.

Les SoC dans lesquels ils sont intégrés possèdent des mémoires caches ainsi que bien d'autres périphériques leur permettant de faire démarrer un noyau Linux. Pour des raisons techniques, nous avons sélectionné le cœur CVA6 développé par l'Open Hardware Group. Ce cœur, développé par ETH Zürich et poussé dans des projets étudiants par Thales, possède pour principal avantage d'être déployable sur les cartes de prototypage Digilent Genesys 2 que nous avions à disposition.

Le cœur CVA6 possède une architecture interne composée de 6 étages de pipeline,

Système évalué	**CVA6 cœur**	**CVA6 cœur**	**CVA6 cœur**	**CVA6 SoC**	**CVA6 SoC**	**CVA6 SoC**
Chemin de données	64 bits	64 bits	64 bits	64 bits	64 bits	64 bits
Entrées dans l'UAL	2	3	3	2	3	3
ISA	RV64I	RV64I	RV64I	RV64I	RV64I	RV64I
Extensions RV	MAC	MAC	MACFD	MAC	MAC	MACFD
LUT	34576	34695	46615	59495	59624	71551
Flip-Flop	19806	19929	23954	41870	41993	46006
Blocs DSP	16	16	27	16	16	27
Blocs RAM	72	72	72	87	87	87

Table 6.2 – Complexités matérielles des cœurs CVA6 avec SoC sur cible Kintex 7 lorsque 2 ou 3 opérandes sont accessibles dans l'UAL. Résultats post-placement routage avec fréquence de fonctionnement fixée à 50 MHz.

tel que cela est schématisé dans la figure 6.1. Il implante l'ISA de base RV64I, supporte nativement les extensions M/C/F et A de la spécification RISC-V et intègre un mécanisme de prédiction de branchement efficace. En interne, en fonction de la configuration du cœur, ce dernier peut opérer sur des données codées sur 32 bits ou 64 bits. Ce dernier format de représentation des données, choisi avant la synthèse logique du cœur, permet d'appliquer un facteur de parallélisation deux fois supérieur lors des modes de fonctionnement SIMD. Toutefois, il est à noter que tout comme les cœurs RISC-V précédemment étudiés, le cœur CVA6 ne gère pas les accès mémoire aux données non-alignées. Les informations fournies dans le tableau 6.2 permettent de quantifier la complexité du cœur CVA6 ainsi que du SoC dans lequel il est inclus. Les résultats présentés sont issus des rapports post-placement et routage pour un FPGA de type Xilinx Kintex 7 lorsqu'une fréquence de fonctionnement de 50 MHz est fixée.

6.2 Évolution des extensions

Dans cette partie, nous allons succinctement présenter les évolutions des extensions précédemment proposées. Pour des raisons de lisibilité et de redondance, les descriptifs complets des instructions sont reportés en annexes.

L'utilisation de trois opérandes d'entrée permet de concevoir des instructions plus complexes en chaînant, par exemple, plus de ressources matérielles. Afin d'identifier les nouvelles instructions pour chaque famille de codes correcteurs d'erreurs, nous sommes repartis des spécifications algorithmiques et des listings assembleurs issus de nos descriptions logicielles initiales. Ceci, pour ne pas être influencé par des choix et des décisions

	Mnémonique	Mode SISD		Mode SIMD inter-trames	
		2R	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_abs_pi8	✓	✓	✓	✓
	i8_add_sat127_pi8	✓	✓	✓	✓
	i8_min_pi8	✓	✓	✓	✓
	i8_sub_sat127_pi8	✓	✓	✓	✓
	i8_cmpeq_pi8	✓		✓	
	i8_invB_Aneq1_pi8	✓		✓	
	i8_max_pi8	✓		✓	
	i8_xorA_signB_pi8	✓		✓	
	i8_sign_pi8			✓	✓
Instr à 3 reg.	i8_invC_xorA_signB_pi8		✓		✓
	i8_minC_maxAB_pi8		✓		✓
	i8_blendC_cmpeqAB_pi8		✓		✓
Total (2R|3R)		8	7(4|3)	9	8(5|3)

Table 6.3 – Instructions proposées pour l'accélération du décodage des codes LDPC en utilisant 2 ou 3 opérandes.

issues de la première phase de nos travaux.

Il résulte de cette analyse qu'en plus des instructions manipulant trois opérandes d'entrées, notées **3R**, il est nécessaire de conserver des instructions à une ou deux opérandes. En effet, pour certaines opérations complexes qui pourraient être factorisées, certaines valeurs intermédiaires doivent aussi être stockées en mémoire pour une utilisation ultérieure. C'est le cas, par exemple, du calcul de la valeur absolue (*i8_abs_pi8*), et des additions/soustractions saturées (*i8_add_sat127_pi8* et *i8_sub_sat127_pi8*). L'extension du jeu d'instructions que nous avons ainsi développé, est une combinaison entre un sous-ensemble des instructions précédemment identifiées et de nouvelles instructions. Les instructions nouvellement définies, ainsi que celles issues du travail présenté dans le chapitre 3 sont résumées pour les codes LDPC dans le tableau 6.3. Dans ce tableau, les colonnes notées **2R** font référence aux extensions proposées dans le chapitre précédent, tandis que les colonnes labellisées **3R** décrivent les extensions nécessaires ici (qu'elles possèdent 2 ou 3 opérandes).

On observe pour les codes LDPC que trois nouvelles instructions à 3 registres ont été proposées pour le mode d'exécution scalaire. Ces dernières, décrites dans le tableau 6.4, sont utilisées de la manière suivante :

— *i8_invC_xorA_signB_pi8* : Permet de simplifier le calcul de parité en couplant le calcul du signe du LLR et la mise à jour de la valeur de parité.

Instruction	Mnémonique	Implantation
Maximum puis minium	i8_minC_maxAB_pi8 rD,rS1,rS2,rS3	rD := MIN8(MAX8(rS1,rS2),rS3)
Calcul du minimum et mise à jour	i8_blendC_cmpeqAB_pi8 rD,rS1,rS2,rS3	mask := (rS1==rS2) ?0xff:0x00 min_t := rS1 & ∼mask min_u := rS3 & mask rD := min_t |min_u
Génération du message sortant du noeud de parité	i8_invC_xorA_signB_pi8 rD,rS1,rS2,rS3	rD := rS1 ∧(rS2 ⩾0) ?rS3 : -rS3

Table 6.4 – Spécification des nouvelles instructions à 3 opérandes pour le décodage des codes LDPC.

Instruction	Mnémonique	Implantation
Fonction G de l'arbre de décodage SC	i8_Gx_pi8 rD,rS1,rS2,rS3	Addsat := sat127rS1 + rS2 Subsat := sat127rS2 - rS1 rD :=(rS3 == 0) ? Addsat : Subsat

Table 6.5 – Spécification de l'instruction pour la fonction **G** pour le décodage des codes polaires.

— *i8_minC_maxAB_pi8* : Réalise la mise à jour de la valeur du *min2* de l'algorihtme Min-Sum en une seule instruction.

— *i8_blendC_cmpeqAB_pi8* : Met en œuvre un masque logique pour sélectionner la valeur de *min1* ou de *min2* lors de la génération du LLR en sortie.

La mise au point de ces trois instructions spécialisées pour le mode scalaire a permis de remplacer quatre instructions précédemment déployées (**2R**). Il est à noter que ces instructions à 3 opérandes permettent aussi de supprimer l'usage d'instructions arithmétiques et logiques RV32I qui étaient mises en œuvre dans les cœurs de calcul [1]. Pour les architectures SIMD inter-trames, les résultats obtenus sont similaires à ceux obtenus précédemment.

Contrairement au cas des codes LDPC où les nouvelles instructions **3R** sont une combinaison des instructions précédemment identifiées, pour les codes polaires l'extension à 3 opérandes a aussi permis d'optimiser la fonction **G** dont l'équation est fournie dans le chapitre 3. En effet, celle-ci ne pouvait être accélérée auparavant. La possibilité d'utiliser 3 opérandes en entrée de l'UAL permet d'inclure l'ensemble des traitements à réaliser au sein d'une seule et unique instruction, tel que cela est indiqué dans le tableau 6.5.

L'analyse des données présentées dans le tableau 6.6 permet de noter que l'utilisation de cette nouvelle instruction permet de se passer des instructions *i8_setMask_Aeq1_pi8*, *i8_add_sat127_pi8* et *i8_sub_sat127_pi8* et ainsi de réduire le nombre de modifications

1. Ces dernières n'apparaissent pas dans le tableau

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_setMask_Aeq1_pi8			✓		✓	
	i8_add_sat127_pi8	✓		✓		✓	
	i8_sub_sat127_pi8	✓		✓		✓	
	i8_clrA_Bneq0_pi8	✓	✓	✓	✓	✓	✓
	i8_Rx_pi8	✓	✓	✓	✓	✓	✓
	i8_Fx_pi8	✓	✓	✓	✓	✓	✓
Instr. à 3 reg.	i8_Gx_pi8		✓		✓		✓
Total (2R|3R)		6	4(3|1)	6	4(3|1)	6	4(3|1)
Extension pour l'algorithme Fast-SC(F-SC)							
Instr. à 2 reg.	i16_add_accA_loB_pi8			✓	✓		
	i16_add_accA_hiB_pi8			✓	✓		
	i8_sign_pi16			✓	✓		
	i8_Hxor_pi8					✓	✓
	i8_Hadd_pi8					✓	✓
Total (2R|3R)		6	4(3|1)	9	7(6|1)	9	6(5|1)

Table 6.6 – Instructions proposées pour l'accélération du décodage des codes polaires (algorithmes SC et F-SC) en utilisant 2 ou 3 opérandes.

Instruction	**Mnémonique**	**Implantation**
Minimum entre C et addition saturée de A et B	u8_min_addsat_pu8	rD := Min(Sat63(rS1+rS2), rS3)
Comparaison égale ou supérieure de A et B retourne les 8 LSB de C	u8_cmpge_AndB_lo8C_pu8	rD := (rS1 >= rS2) ? rS3[7:0]:0x00

Table 6.7 – Nouvelles instructions spécifiques pour les codes LDPC-NB avec 3 registres sources.

à apporter au décodeur d'instructions. Afin d'accélérer le décodage des codes polaires utilisant l'algorithme SC, seules 4 instructions sont maintenant requises. Cette diminution d'un facteur 1.5× du nombre d'instructions spécialisées, par rapport à l'approche **2R** va permettre de réduire significativement le nombre d'instructions à exécuter. Le tableau 6.6, met en évidence que les instructions présentées afin d'améliorer les opérations d'élagage de leur côté, ne bénéficient pas de ce 3$^{\text{ème}}$ opérande. Ainsi, l'accélération de l'élagage est toujours supportée par 5 instructions **2R**. En procédant à une analyse similaire à celle que nous venons de décrire, un nouveau jeu d'instruction est ainsi identifié et reporté dans le tableau 6.7 pour la famille de code LDPC-NB. Celle-ci consiste en l'assemblage de plusieurs instructions spécifiques proposées précédemment et de nouvelles.

Toutefois, du fait de la complexité accrue de l'algorithme de décodage (en logiciel),

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	u8_addu_sat64_pu8	✓	✓	✓	✓	✓	✓
	u8_minu_pu8	✓	✓	✓	✓	✓	
	u8_subu_sat64_pu8	✓	✓	✓	✓	✓	✓
	u8_maxu_pu8			✓	✓		
	u8_cmpge_pu8			✓			
	u8_hminu_pu8					✓	✓
	u8_Aandb_lo8B_pu8			✓			
Instr. à 3 reg.	u8_min_addsat_pu8		✓		✓		✓
	u8_cmpge_AandB_lo8C_pu8				✓		
Total (2R|3R)		3	4(3|1)	5	6(4|2)	4	4(3|1)

Table 6.8 – Instructions proposées pour l'accélération du décodage des codes LDPC-NB en utilisant 2 ou 3 opérandes.

il est difficile d'extraire des instructions spécifiques pertinentes et compatibles avec les contraintes voulues. En effet, le nombre d'accès mémoires et la gestion des pointeurs mémoire complexifient l'analyse du code. Les instructions pertinentes identifiées sont les suivantes :

— *u8_min_addsat_pu8* : permet une concaténation des instructions d'additions saturées, puis génère le minimum entre ce résultat et la troisième entrée.

— *u8_cmpge_AandB_lo8C_pu8* : effectue une comparaison entre A et B puis renvoie les 8 premiers bits de C ou bien 0 selon le résultat de la comparaison.

Comme pour les codes LDPC, l'utilisation de ces nouvelles instructions vont théoriquement aboutir à une réduction du nombre total d'instructions exécutées dans l'algorithme. Une différence importante entre les versions inter et intra-trame est l'utilisation d'un minimum horizontal. Une majorité des instructions à 2 registres sources de notre extension sont conservées. Les nouvelles instructions à 3 registres permettent, quand à elles, d'améliorer la latence dans la fonction de calcul des nœuds de parité, à l'aide de la fonction *u8_min_addsat_pu8*. Enfin, l'instruction *cmpge_AandB_lo8C_pu8* permet de simplifier l'étape de recherche de l'index de la valeur minimale du vecteur de données inter-trames.

Enfin, en ce qui concerne les turbo codes, les résultats sont grandement améliorés grâce à l'utilisation d'une architecture **3R**. En effet, cette famille de codes bénéficie grandement des instructions à 3 registres sources, comme montré dans 6.10. Par exemple, les calculs nécessaires au parcours d'un treillis LTE de l'algorithme BCJR font appel à des opéra-

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_max_pi8	✓	✓	✓	✓	✓	✓
	i8_scale_pi8	✓	✓	✓	✓		
	i8_sign_pi8			✓	✓		
	i8_add_pi8			✓	✓	✓	✓
	i8_sub_pi8			✓	✓	✓	✓
	i8_add_srl1_pi8			✓	✓		
	i8_srl1_pi8			✓	✓		
	i8_shuffle_pi8					✓	✓
	i8_add_div2_pi8					✓	✓
	i8_sub_div2_pi8					✓	✓
	i8_sat_sub_pi8					✓	✓
	i8_scale_add_pi8					✓	✓
Instr. à 3 reg.	i8_accumax_pi8		✓		✓		
	i8_maxpm_pi8		✓		✓		✓
	i8_accupp_pi8		✓		✓		✓
	i8_accump_pi8		✓		✓		✓
Total (2R|3R)		2	6(2|4)	7	11(7|4)	8	11(8|3)

Table 6.9 – Instructions proposées pour l'accélération du décodage des turbo codes en utilisant 2 ou 3 opérandes.

Instruction	**Mnémonique**	**Implantation**
Addition accumulée	i8_accupp_pi8 rD,rS1,rS2,rS3	rD := rS1 + rS2 + rS3
Addition et soustraction accumulée	i8_accump_pi8 rD,rS1,rS2,rS3	rD := rS1 - rS2 + rS3
Maximum accumulée	i8_accumax_pi8 rD,rS1,rS2,rS3	rD := MAX8(MAX8(rS1+rS2),rS3)
Maximum entre A+B et C-B	i8_maxpm_pi8 rD,rS1,rS2,rS3	rD := MAX8(rS1 + rS2 , rS3 - rS2)

Table 6.10 – Nouvelle Instruction spécifique pour les turbo codes avec 3 registres sources.

tions utilisant 3 données. Il s'agit de la concaténation d'opérations simples : addition, soustraction et maximum. On retrouve ces nouvelles instructions dans la table ci-dessous 6.9.

NOTA BENE : Les instructions *i8_add_div2_pi8*, *i8_sub_div2_pi8*, *i8_sat_sub_pi8* et *i8_scale_add_pi8* ne figurent pas dans les chapitres précédents.

En effet, celles-ci ont été identifiées après l'écriture de ces chapitres, et conservent le type **2R** (ou 2 registres sources). Puisque leur application est totalement intégrée à ce chapitre, elles sont ainsi présentées ici. Il est important de noter la forte comptabilité entre les instructions **3R** pour les différents modes de parallélisation, ce qui permet de diminuer d'autant la complexité du jeu d'instructions dédié. La latence finale est également optimisée grâce au nombre d'instructions spécialisées mises en œuvre. La version scalaire

SISD bénéficie de 2 instructions, tandis que les architectures **3R**, ont à disposition entre 6 et 11 instructions spécialisées. Les résultats de leur implantation matérielle sont présentés dans les sections suivantes.

6.3 Impact des nouvelles extensions de l'ISA

Afin de mesurer l'impact de ces nouvelles extensions, plusieurs campagnes de tests ont été effectuées. Deux métriques ont été mesurées pour évaluer les performances de nos propositions :

— Le nombre de cycles d'horloges nécessaires pour exécuter les algorithmes de décodage.

— L'augmentation de la complexité matérielle du cœur CVA6 et l'impact sur sa fréquence de fonctionnement.

Ces évaluations ont été réalisées en sélectionnant la configuration de base du cœur CVA6 (RV64I). L'évaluation du temps d'exécution des différents algorithmes de décodage a été obtenu par simulation. Celles-ci ont été réalisées avec l'aide de l'outil *verilator* (version 4.110) qui, à partir de la description en SystemVerilog du cœur CVA6, permet de réaliser des simulations *Bit Accurate* et *Cycle Accurate*. Cette stratégie, a été privilégiée pour éviter des temps de synthèse logique prohibitifs. En effet, il faut compter environ une heure pour réaliser une synthèse et le placement routage du SoC contenant le cœur CVA6.

6.3.1 Évaluation des extensions scalaires

Cette section présente les résultats obtenus pour les décodeurs de CCE étudiés lorsqu'ils opèrent sur des données scalaires, i.e. sans parallélisation SIMD. Les instructions spécialisées traitent des données de 8 bits. Le tableau 6.11 présente une synthèse des résultats obtenus pour l'ensemble des algorithmes de décodage évalués. Les résultats rapportés ici sont homogènes : l'ensemble des travaux présentés dans les chapitres 4 et 5 ont été rejoués pour l'architecture CVA6. Dans le tableau 6.11, sont indiqués les temps nécessaires à l'exécution des versions basiques des décodeurs (❶, Baseline), des versions étendues précédentes (❷, **ISA 2R**) et des versions présentées dans ce chapitre (❸, **ISA 3R**). Dans ce tableau, sont aussi indiqués les gains relatifs entre les différentes versions (❶→❸, ❷→❸) afin de simplifier l'analyse.

		LDPC OMS	SC	**polaire** F-SC	**LDPC-NB** MS	**turbo** MLM
❶ Baseline	cycles	95955	454637	239292	1715625	1585822
❷ SISD ISA 2R	cycles	66013	271609	125008	1357945	938893
❸ SISD ISA 3R	cycles	56514	257901	114207	1086453	819757
Gain (❶→❸)	cycles	39441	196736	125085	629172	766065
Gain (❶→❸)	(%)	41.1%	43.2%	52.2%	36.6%	48.3%
Gain (❷→❸)	cycles	9499	13708	10801	271492	119136
Gain (❷→❸)	(%)	14.3%	5.0%	8.6%	19.9%	12.6%
Débit ❶	Kbits/s	141	900	1711	7	390
Débit ❷	Kbits/s	206	1508	3276	9	659
Débit ❸	Kbits/s	240	1588	3586	12	755
Accélération ❶ → ❸		1.7×	1.8×	2.1×	1.6×	1.9×

Table 6.11 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration 8 bits SISD avec une fréquence de fonctionnement de 50 MHz.

Si l'on s'intéresse tout d'abord aux décodeurs LDPC, on peut noter une réduction du nombre de cycles d'horloge d'environ 14% entre la version précédente (ISA 2R ❷) et la version actuelle (ISA **3R** ❸). Ces gains sont liés aux instructions à 3 opérandes d'entrée que nous avons développées. Par rapport à la version d'origine du décodeur, les 7 instructions spécialisées que nous avons ajoutées ont permis de réduire le temps d'exécution de 41% sur le cœur CVA6.

Pour les codes polaires, les gains sont plus limités lorsque l'on considère l'algorithme de décodage SC. En effet, le passage de la version ❷ à ❸ n'engendre qu'une modeste réduction de 5% du temps d'exécution. Cette dernière n'ajoute qu'une seule instruction depuis la version ❷ avec l'instruction *i8_Gx_pi8* appliquant la fonction **G** de l'arbre de décodage. Cela permet de s'affranchir de l'utilisation des instructions d'addition et soustraction saturées ainsi que de la sélection conditionnelle. Toutefois, lors de la compilation, le compilateur GCC introduit des instructions supplémentaires de conversions entre les formats int8_t ↔ int32_t lors de l'utilisation de l'instruction **i8_Gx_i8** à 3 entrées. La pénalité liée à l'ajout de ces instructions d'extension de signe des opérandes est moins présente pour le décodeur élaguant l'arbre de décodage (F-SC). Ainsi, une réduction de 8.6% est mesurée entre les versions ❷ et ❸. L'accélération atteint même 52.2% lorsque l'on compare la version ❶ et ❸.

Les facteurs d'accélérations mesurés pour le décodage des codes LDPC-NB sont plus

importants. En effet, entre les versions ❷ et ❸ l'accélération avoisine les 20%. Ce facteur d'accélération est issu de l'utilisation de la nouvelle instruction *u8_min_addsat_pu8* qui est mise en œuvre à la fois dans la fonction de mise à jour des messages des nœuds de variables vers les nœuds de parités ainsi que dans le calcul des nœuds de parités. Ce gain important s'explique par la faible accélération issue des instructions proposées dans ❷. L'accélération obtenue par rapport à la solution initiale ❶ est de $\approx 37\%$.

Enfin, si l'on concentre notre attention sur les turbo décodeurs, une amélioration de 48% est obtenue par la configuration ❸ par rapport à la configuration initiale ❶. Les 4 instructions (ainsi que les instructions **2R**) ajoutées dans ❸ par rapport à la configuration ❷ ont permis une réduction supplémentaire de $\approx 12\%$. Les gains obtenus sont limités à ces niveaux à cause des accès mémoire (*load*, *store* et arithmétique des pointeurs) qui deviennent prépondérants par rapport aux instructions arithmétiques.

Les dernières lignes du tableau 6.11 présentent le débit normalisé des décodeurs, pour une fréquence de fonctionnement à 50 MHz. Cette fréquence est liée à l'implantation matérielle du cœur CVA6 sur le FPGA Kintex-7 présent sur la carte Digilent Genesys 2. Il est à noter que les instructions ajoutées dans le cœur CVA6 n'ont pas d'impact sur la fréquence maximale de fonctionnement. Les résultats présentés mettent en exergue que les configurations ❸ des décodeurs offrent des gains substantiels grâce aux instructions spécialisées proposées.

Afin d'estimer le coût matériel associé à l'implantation des ces instructions dans le cœur testé, les UAL correspondant à chaque configuration sont isolées et comparées à la version de l'UAL standard du cœur. Les surcoûts, exprimés en LUT, sont présentés dans la figure 6.2.

La figure 6.2 présente ainsi les résultats obtenus. Pour les codes LDPC, la configuration **2R** implante 8 instructions, soit 1 instruction supplémentaire par rapport à la configuration **3R** avec 7 instructions. Toutefois, le coût varie de presque 50% entre ces 2 versions. CE gain provient de la simplification du multiplexeur de sortie de l'UAL. Dans le cas des codes polaires, les algorithmes SC et Fast-SC (F-SC) partagent les mêmes implantations d'instructions en matériel pour les configurations **2R** et **3R**, ainsi les résultats observés sont similaires. L'implantation de 4 (**3R**) au lieu de 6 instructions (**3R**), permet de réduire l'empreinte matérielle associée de $\approx 30\%$. Cette observation est également valable avec les codes LDPC-NB. Enfin, pour les turbo codes, 6 instructions sont implantées pour la configuration (**3R**), contre seulement 2 (**2R**). On constate ainsi que le coût matériel augmente fortement et est plus que doublé (137%). Il est à noter que toutes ces instructions

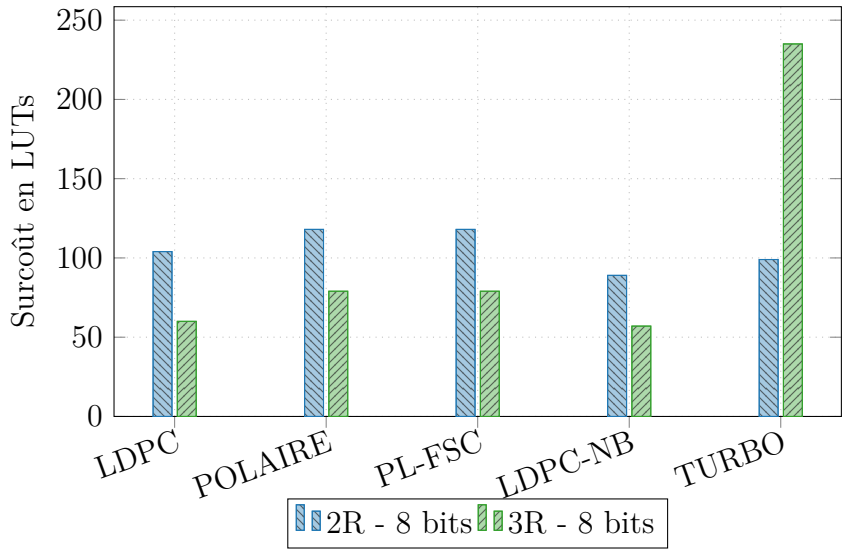

Figure 6.2 – Surcôt des instructions spécialisées SISD dans l'UAL du cœur CVA6.

sont implantées sans conséquences sur la fréquence de fonctionnement du cœur.

6.3.2 Évaluation des extensions SIMD inter-trames

Dans les expérimentations précédentes, nous nous sommes focalisés sur les décodeurs manipulant des données scalaires. Nous allons maintenant étudier l'impact de ces nouvelles extensions lors de l'exécution de décodeurs bénéficiant d'une parallélisation inter-trames. Le CVA6 étant un cœur 64 bits, nous avons décidé de tester deux formats de parallélisation SIMD : 32 et 64 bits. Le premier traite quatre informations codées sur 8 bits chacune (**int8_t×4**), tandis que le second étend ce traitement à 8 données (**int8_t×8**). Les résultats de ces expérimentations sont résumés dans le tableau 6.12.

Si l'on s'intéresse dans un premier temps aux performances des décodeurs LDPC manipulant des données codées sur **int8_t×4**, on observe une réduction de 12% lors du passage ❷→❸. Cette observation est cohérente par rapport aux résultats des expérimentations SISD (tableau 6.11). Pour la version manipulant des données **int8_t×4**, le gain mesuré est plus faible avec 9.2%. Ceci est causé par le temps d'entrelacement des données en amont et en aval de l'étape de décodage, qui consomme 2 fois plus de cycles d'horloge. Toutefois, comme le nombre de données manipulées est 2 fois plus élevé, avec un nombre de cycles d'horloge à peu près équivalent, le débit final est ainsi doublé. Concernant les autres codes correcteurs, lorsqu'ils manipulent des données codées sur 32 bits, on remarque que les gains liés au passage de ❷→❸ sont plus importants que ceux relevés

inter-trames			**LDPC** OMS	SC	**polaire** F-SC	**LDPC-NB** MS	**turbo** MLM
int8	❶ Baseline	cycles	95955	454637	239292	1715625	1585822
int8_t×4 (32 bits)	❷ ISA 2R	cycles	61443	373854	215509	1292853	1051247
	❸ ISA 3R	cycles	54027	292678	167095	969990	804471
	Gain (❶→❸)	cycles	41928	161959	72197	745635	781351
	Gain (❶→❸)	(%)	43.7%	35.6%	30.2%	43.5%	49.2%
	Gain (❷→❸)	cycles	7416	81176	48414	322863	246776
	Gain (❷→❸)	(%)	12.0%	21.7%	22.4%	25.0%	23.4%
	Débit ❶	Kbits/s	141	900	1711	7	390
	Débit ❷	Kbits/s	885	4382	7602	39	2356
	Débit ❸	Kbits/s	1006	5598	9805	52	3078
int8_t×8 (64 bits)	❹ ISA 2R	cycles	57898	388220	243988	1359186	1207724
	❺ ISA 3R	cycles	52546	335032	213560	1015731	1035964
	Gain (❶→❺)	cycles	43409	119605	25732	699894	549858
	Gain (❶→❺)	(%)	45.2%	26.3%	10.8%	40.8%	34.6%
	Gain (❷→❺)	cycles	5352	53188	30428	343455	171760
	Gain (❷→❺)	(%)	9.2%	13.7%	12.4%	25.7%	14.2%
	Débit ❶	Kbits/s	141	900	1711	7	390
	Débit ❹	Kbits/s	1879	8440	13430	75	4101
	Débit ❺	Kbits/s	2070	9780	15343	100	4781
Accélération							
	❶ → ❸	32 bits	7.1×	6.2×	5.7×	7.1×	7.9×
	❶ → ❺	64 bits	14.6×	10.9×	9.0×	13.5×	12.2×
	❸ → ❺	32 → 64 bits	2.1×	1.7×	1.6×	1.9×	1.6×

Table 6.12 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration SIMD inter-trames avec une fréquence de fonctionnement fixée à 50 MHz.

lors des expérimentations en SISD. Ils varient de 20% à 26% en fonction des familles de code, montrant l'intérêt de ces extensions d'instructions. Des gains moins marqués sont observés pour les décodeurs manipulant des données de type **int8_t×8**. En effet, pour les données **int8_t×8**, GCC introduit fréquemment des instructions d'extension de signe (32 bits → 64 bits) qui sont inutiles dans nos cas d'usage. Ces dernières impactent négativement le temps d'exécution des décodeurs. L'utilisation de nos instructions à 3 opérandes en mode **int8_t×8** permet de supprimer des données temporaires, et donc une partie de ces extensions de signe associées, augmentant de facto le facteur d'accélération.

L'ajout des extensions 32 bits et 64 bits apporte des gains notables en termes d'accélération du temps de calcul. Toutefois, l'implantation des opérateurs matériels au sein de l'UAL du cœur CVA6 engendre aussi une augmentation de la complexité matérielle.

La figure 6.3 présente le surcoût effectif de ces extensions (**3R**) lorsqu'elles sont insérées dans le cœur CVA6. Il est à noter que, comme précédemment, l'ajout de ces extensions en matériel n'a pas eu d'influence sur la fréquence de fonctionnement maximale du cœur après placement et routage par l'outil Vivado.

Si l'on s'intéresse dans un premier temps aux extensions proposées pour les décodeurs LDPC, on observe que les surcoûts matériels sont équivalents (5% supérieur) en mode **2R** et **3R**. De fait, même si plusieurs instructions **2R** ont été fusionnées pour produire des instructions **3R**, une instruction **2R** supplémentaire a été ajoutée. Le surcoût matériel pour les codes polaires est quant à lui plus faible (4%) que celui de l'extension **2R**. L'implantation des codes polaire avec l'algorithme FAST-SC mobilise 3 instructions supplémentaires en mode **2R** et **3R** afin d'accélérer les nœuds de l'arbre élagué. Cette différence permet un début d'explication pour la différence de coût $\approx 25\%$ observée en comparaison avec les codes polaires simples. Toutefois, il y a aussi une différence entre les 2 modes précédemment observés. Pour les codes LDPC-NB et les turbo codes, l'ajout de 2 et 4 instructions respectivement dans l'extension **3R** ont engendré une augmentation respective de $\approx 2\%$ et $\approx 77\%$ du coût en LUT.

Ces coûts matériels, qui peuvent sembler importants, sont dans un premier temps à mettre en perspective par rapport au coût matériel du cœur CVA6 (tableau 6.2) qui nécessite 34576 LUT dans sa configuration 64 bits. Ainsi, l'augmentation de la complexité du cœur est comprise entre 1% et 3% en fonction de l'extension considérée. Ces surcoûts peuvent être considérés comme marginaux lorsqu'on les compare aux facteurs d'accélération de $9\times$ à $14\times$ obtenus (tableau 6.12).

6.3.3 Évaluation des extensions SIMD intra-trame

Cette dernière section est consacrée à l'étude des extensions proposées pour le second schéma de parallélisation SIMD. Une partie des instructions est commune avec celles mises en œuvre pour la parallélisation inter-trames. Toutefois, d'autres instructions impliquant des réductions horizontales ont été nécessaires. L'objectif de ce schéma est de réduire la latence de traitement des décodeurs. Pour y parvenir, il est nécessaire d'exploiter un parallélisme calculatoire variable et/ou plus faible car les descriptions algorithmiques, malgré le fait qu'elles soient plus complexes, sont moins favorables.

Les performances temporelles des décodeurs ainsi optimisés à l'aide des extensions présentées dans ce chapitre sont exposées dans le tableau 6.13, à l'exception des décodeurs LDPC qui à cause de leur nécessité de réaliser des accès non alignés en mémoire n'ont pu

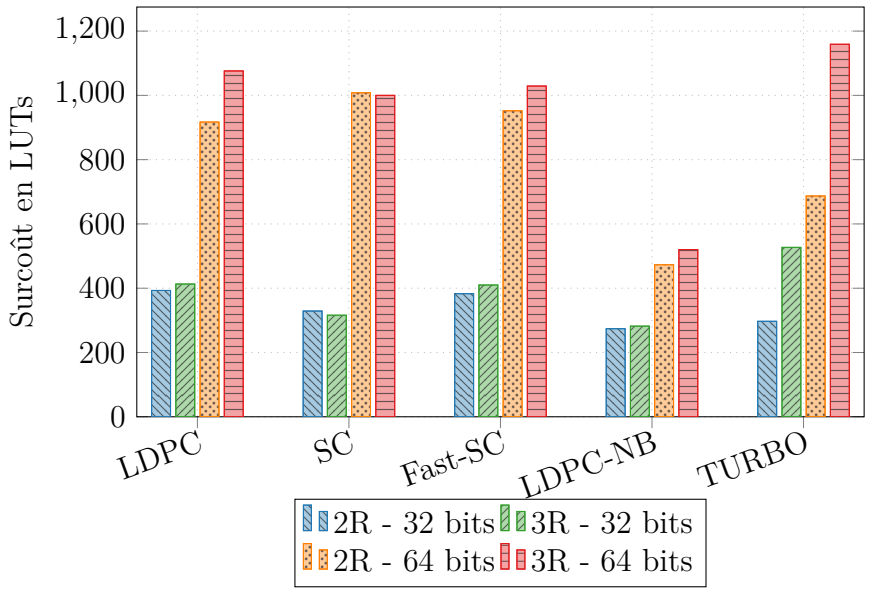

Figure 6.3 – Surcoût matériel induit par les extensions nécessaires à la parallélisation inter-trames en configuration 32 bits et 64 bits.

être évalués.

Les données issues du tableau 6.13 comparent les performances obtenues par les nouvelles extensions à **3R** par rapport à celles du chapitre précédent. Si l'on s'intéresse dans un premier temps au décodage des codes polaires et plus précisément au décodeur SC, on remarque que le schéma de parallélisation SIMD en 32 bits manipulant 4 données en parallèle permet de réduire de 76% le temps d'exécution par rapport à l'approche naïve (❶, Baseline). Par rapport à la version **2R**, l'utilisation de **3R** apporte un gain supplémentaire de 23%. Ces ordres de grandeur se retrouvent pour les expérimentations réalisées pour l'algorithme F-SC.

Les gains obtenus pour le décodage des turbo codes sont plus conséquents. Par rapport à la version naïve, le temps d'exécution de la version **3R** est réduit de 90%. Les nouvelles extensions mises au point grâce à la 3$^{\text{ème}}$ opérande ont permis de réduire de 45% le temps d'exécution de la version **2R**.

Les gains observés pour les décodeurs LDPC-NB atteignent 27% et 27.1% lorsque comparés au décodeur de base. Théoriquement, l'utilisation de 8 données concurrentes en parallèle (**int8_t×8**) devrait améliorer significativement la latence de décodage par rapport à 4 données (**int8_t×4**), contrairement à ce qui est observé. Ces résultats sont principalement liés à l'inadéquation de la stratégie de parallélisation vis-à-vis des opérations mises en oeuvre dans l'algorithme de décodage, et du format des données par

intra-trame				**polaire**	**LDPC-NB**	**turbo**
			SC	F-SC	MS	MLM
int8	❶ Baseline	cycles	454637	239292	1715625	1585822
int8_t×4	❷ ISA 2R	cycles	139961	55031	1270968	269167
(32 bits)	❸ ISA 3R	cycles	108701	42684	927114	148194
	Gain (❶→❸)	cycles	345936	196608	788511	1437628
	Gain (❶→❸)	(%)	76.0%	82.1%	46.0%	90.6%
	Gain (❷→❸)	cycles	31260	12347	343854	120973
	Gain (❷→❸)	(%)	22.3%	22.4%	27.0%	44.9%
	Débit ❶	Kbits/s	900	1711	7	390
	Débit ❷	Kbits/s	2926	7443	10	2300
	Débit ❸	Kbits/s	3768	9596	14	4178
int8_t×8	❹ ISA 2R	cycles	102298	24172	1264971	139404
(64 bits)	❺ ISA 3R	cycles	80894	20660	921355	67483
	Gain (❶→❺)	cycles	373743	218632	794270	1518339
	Gain (❶→❺)	(%)	82.2%	91.3%	46.3%	95.7%
	Gain (❹→❺)	cycles	21404	3512	343616	71921
	Gain (❹→❺)	(%)	20.9%	14.3%	27.1%	51.5%
	Débit ❶	Kbits/s	900	1711	7	390
	Débit ❹	Kbits/s	4003	16945	10	4441
	Débit ❺	Kbits/s	5063	19825	14	8175
Accélération						
	❶ → ❸	32 bits	4.2×	5.6×	1.9×	10.7×
	❶ → ❺	64 bits	5.6×	11.6×	1.9×	23.7×
	❹ → ❺	32 → 64 bits	1.3×	2.1×	1×	2.2×

Table 6.13 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration SIMD intra-trame avec une fréquence de fonctionnement fixée à 50 MHz.

rapport à la largeur de l'unité SIMD. En effet, le format $\mathbb{GF}$=16 nécessiterait des instructions SIMD sur 16×8 bits pour accélérer les multiplications et les divisions dans $\mathbb{GF}$.

Les facteurs d'accélération entre les versions originales des décodeurs et des implantations (❶ → ❸), présentés dans le tableau 6.13 soulignent ici aussi l'intérêt des extensions proposées.

Lorsque l'on s'intéresse à l'extension de la taille des données manipulées par nos extensions SIMD de 32 bits à 64 bits on observe une accélération variant d'un facteur 1× → 2×. Des facteurs d'accélération de ≈ 2× sont obtenus pour certains codes, tandis que pour d'autres les accélérations sont moins favorables, voir bien inférieures à celles obtenues dans le cas de la parallélisation inter-trames. Cela s'explique par un parallélisme de calcul plus difficilement exploitable en mode intra-trame. Par exemple, concernant le décodage

SC des codes polaires, le parallélisme de calcul varie lors du décodage. Ainsi, au niveau des feuilles de l'arbre, seules des opérations scalaires sont réalisées. En effet, durant une partie du processus de décodage, le doublement des unités de calculs est sans conséquence, limitant de fait l'accélération globale. Le facteur d'accélération du décodeur LDPC-NB est bien inférieur aux attentes. En effet, les accès non réguliers à la mémoire lors des calculs des CNs induisent des pénalités qui annulent les gains liés à nos extensions.

L'impact des extensions proposées pour l'accélération des décodeurs intra-trame sur la complexité matérielle a aussi été évaluée. La figure 6.4 présente le surcoût des extensions en termes de LUT issues des rapports post-placement et routage. Contrairement aux expérimentations réalisées en mode inter-trames, on peut observer une plus grande disparité entre les surcoûts des extensions **2R** et **3R**. Cela est lié au fait que la possibilité d'utiliser 3 opérandes a permis de concevoir plus d'instructions spécialisées. C'est particulièrement le cas, par exemple, pour les turbo codes où l'utilisation d'une troisième opérande a permis de créer 4 instructions spécialisées supplémentaires. Cela explique le doublement du coût de l'extension **3R** par rapport à son pendant **2R**. Toutefois, même si cette observation devrait également se vérifier en mode 64 bits, l'outil Vivado réalise des optimisations durant les étapes post-placement et routage qui tempèrent cette tendance. Un phénomène équivalent est observé pour l'extension dédiée aux code LDPC-NB où l'extension **2R** est plus onéreuse que la version **3R** en configuration 32 bits. Enfin, si l'on compare le surcoût des extensions dédiées au mode intra-trame, par rapport à celles pour le mode inter-trames, on note qu'il est du même ordre de grandeur, à l'exception des instructions pour les turbo codes ; ces dernières sont environ 20% plus onéreuses. Comme nous l'avons précédemment évoqué, ces coûts matériels qui peuvent sembler importants ne représentent que de 1% à 3% de la complexité du CVA6.

6.4 Conclusion

Dans ce chapitre, nous avons présenté de nouvelles extensions permettant d'accélérer les algorithmes de décodage des CCE. Contrairement aux précédentes extensions, ces dernières bénéficient d'un accès possible à un troisième opérande. Cette opportunité a permis de proposer de nouvelles instructions en vue de minimiser le temps d'exécution des algorithmes. Les différentes extensions ont été déclinées sous plusieurs formes afin de permettre d'évaluer à la fois un traitement scalaire, et SIMD, des données. De plus, grâce à l'utilisation d'un cœur RISC-V plus complexe possédant un chemin de données 64 bits,

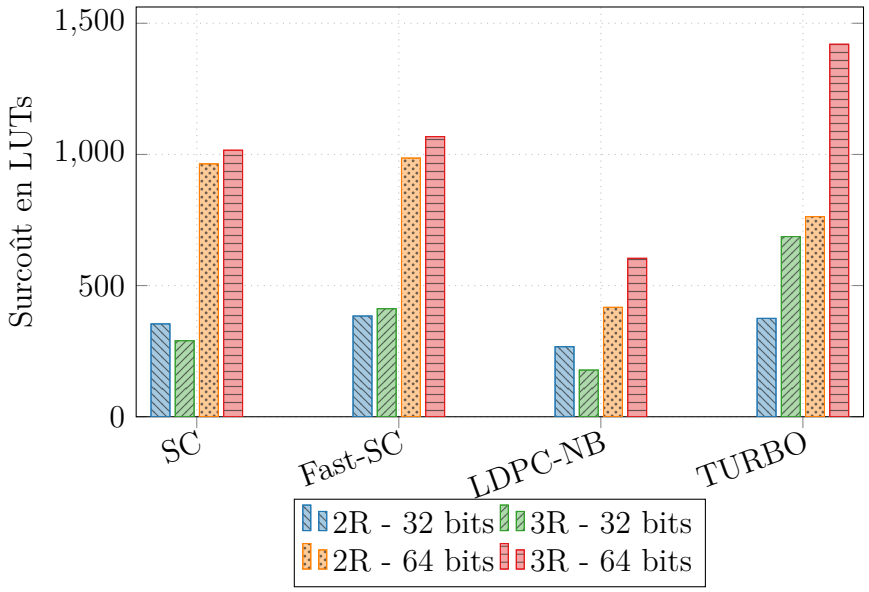

Figure 6.4 – Surcoût matériel induit par les extensions nécessaires à la parallélisation intra-trame en configuration 32 bits et 64 bits.

nous avons pu étudier les options de passage à l'échelle des extensions proposées.

Le tableau 6.14 propose une synthèse des facteurs d'accélération qui ont été mesurés pour les schémas de parallélisation SIMD. Les facteurs d'accélérations qui sont rapportés ont été calculés par rapports aux codes logiciels n'exploitant que le jeu d'instructions de base des cœurs RISC-V. Comme le met en évidence ce tableau, l'ensemble des extensions proposées offre des facteurs d'accélération élevés tandis que le nombre d'instructions ajouté à l'ISA est relativement faible. Par exemple, pour les codes polaires, le passage de 3 opérandes d'entrées à permis de concevoir 1 instruction qui permet d'atteindre une accélération avoisinant 10.7× (64 bits, mode inter-trames). Des facteurs d'accélération plus important ont été obtenus pour le décodage des turbo codes (23.5×) au prix d'un nombre d'instructions spécialisées supérieur (11 instructions en mode 64 bits pour une parallélisation intra-trame). L'utilisation d'un troisième opérande d'entrée pour les opérations spécialisées a permis, en moyenne, d'augmenter le facteur d'accélération de 1.32× (32%) par rapport aux extensions **2R**. Le choix de la stratégie de parallélisation impacte fortement sur les niveaux de performance atteignables. En effet, alors que la stratégie intra-trame permet d'atteindre un facteur d'accélération de 25.5× pour les turbo codes, elle ne permet d'obtenir que 1.7× pour le décodage des codes LDPC-NB où l'approche inter-trames offre 13.5×.

D'un point de vue matériel, les extensions **3R** ont des complexités matérielles com-

CCE		**2R**	**3R**	**2R**	**3R**
SIMD		**int8_t×4**	**int8_t×4**	**int8_t×8**	**int8_t×8**
Bits		**32**	**32**	**64**	**64**
LDPC	inter-trames	885 Kbps	1006 Kbps	1879 Kbps	2070 Kbps
OMS		(**6.2**×)	(7.1×)	(**13.2**×)	(14.6×)
polaire	inter-trames	4382 Kbps	5506 Kbps	8440 Kbps	9633 Kbps
SC		(4.9×)	(6.2×)	(9.4×)	(10.8×)
	intra-trame	2926 Kbps	3768 Kbps	4003 Kbps	5063 Kbps
		(3.2×)	(4.2×)	(4.4×)	(5.6×)
polaire	inter-trames	7602 Kbps	9625 Kbps	13430 Kbps	15212 Kbps
Fast-SC		(4.4×)	(5.7×)	(7.8×)	(9×)
	intra-trame	7443 Kbps	9596 Kbps	16495 Kbps	19825 Kbps
		(4.3×)	(5.6×)	(9.9×)	(11.6×)
LDPC NB	inter-trames	39 Kbps	50 Kbps	75 Kbps	94 Kbps
MS		(5.3×)	(7.0×)	(10.1×)	(13.5×)
	intra-trame	10 Kbps	13 Kbps	10 Kbps	14 Kbps
		(1.3×)	(1.8×)	(1.3×)	(1.8×)
TURBO	inter-trames	2356 Kbps	3078 Kbps	4101 Kbps	4781 Kbps
MLM		(6.0×)	(7.9×)	(10.5×)	(12.2×)
	intra-trame	2300 Kbps	4178 Kbps	4441 Kbps	8175 Kbps
		(5.9×)	(**10.7**×)	(11.4×)	(**23.5**×)

Table 6.14 – Accélération observé des débits par rapport à la Baseline.

parables aux extensions **2R**. En effet, le coût lié à l'ajout de nouvelles instructions est en partie compensée par les gains réalisés par la fusion d'autres instructions. La fusion d'instructions engendrant par exemple une réduction du coût du multiplexeur sélectionnant les résultats en sortie de l'UAL. Les résultats basés sur l'utilisation du cœur CVA6, obtenus post-placement routage, mettent en évidence le faible impact des extensions sur le coût global du cœur. En effet, ces dernières induisent une augmentation limitée de 1% à 3% du coût du cœur en termes de LUT. Enfin, pour toutes les expérimentations que nous avons réalisées, aucune pénalité concernant la fréquence maximale de fonctionnement du cœur n'a été observée.

Chapitre 7

Conclusion et Perspectives

Conclusion

Les travaux de recherches effectués durant cette thèse se sont concentrés sur la spécification et la conception d'instructions spécifiques permettant d'accélérer le décodage de différentes familles de CCE. Un prototypage matériel de ces extensions sur circuit FPGA a permis de caractériser l'augmentation de la complexité matérielle induite et de la mettre en perspective les facteurs d'accélérations observés.

Le déploiement des standards 4G, 5G et ultérieurs devraient permettre d'interconnecter des milliards de systèmes embarqués communicants. Ces systèmes développés sous diverses contraintes liées aux domaines applicatifs des dispositifs devront évoluer pour être inter-opérable. Au sein de ces standards de communication, le codage canal et plus particulièrement le décodage des CCE dans les récepteurs présentent simultanément une forte complexité calculatoire et une grande variabilité en fonction des standards. C'est la raison pour laquelle cette thèse adresse l'optimisation de ces algorithmes de décodage qui garantissent la fiabilité des communications numériques, en corrigeant les erreurs éventuelles dues au canal de communication. Les principales familles de codes correcteurs d'erreurs, ainsi que leurs algorithmes de décodage sont abordées dans le chapitre 2.

Le chapitre 3 s'intéresse aux stratégies d'implantation mises en œuvre pour intégrer ces algorithmes dans les systèmes communicants. Depuis une décennie, pour répondre aux besoins de flexibilité des systèmes de communications numériques, les solutions traditionnelles fondées sur l'utilisation d'ASIC spécialisés, ou bien de FPGA est en décroissance, au profit de l'utilisation d'architectures programmables telles que des CPU multicœurs ou des GPU. Ces solutions procurent des temps de développement raccourcis et une plus grande flexibilité et adaptabilité grâce aux langages de programmation de type C/C++. Cependant, cette flexibilité est obtenue au prix de performances en retrait face aux solutions ASIC/FPGA. Ces différences proviennent de leurs jeux d'instructions génériques. Ceux-ci, peu adaptés au contexte applicatif du décodage des CCE, induisent des débits

réduits et une consommation énergétique plus élevée.

Afin d'améliorer cet état de fait, dans le chapitre suivant, la méthodologie que nous avons utilisée dans le but d'identifier et de développer différentes extensions aux jeux d'instructions des cœurs de processeur est présenté. L'analyse de codes sources logiciels optimisés issus de la littérature et d'une analyse fine des séquences d'instructions produites après compilation logicielle, ont permis d'identifier des motifs d'instructions compatibles avec la conception de nouvelles instructions. Ainsi, des instructions spécifiques aux différents CCE étudiés ont été proposées dans le cadre d'un traitement scalaire des données. L'impact de ces extensions tant en termes de réduction du temps d'exécution que d'augmentation de la complexité matérielle a été évaluée à l'aide de 4 cœurs RISC-V. Ces expérimentations menées sur FPGA ont permis de conclure à l'intérêt de l'approche proposée avec des accélérations variant de 5.3% à 77.1% en fonction des cœurs et des codes, pour une augmentation de la complexité matérielle de 0.8% à 63.2%. Dans le chapitre 5, nous avons étendu ces extensions afin de permettre un traitement parallèle des données (SIMD) lors du processus décodage. La modification des descriptions algorithmiques, couplées aux nouvelles instructions ont permis d'augmenter les débits de facteurs **1.2**× à **4.1**×, en fonction de la stratégie de parallélisation employée et du code correcteurs d'erreurs considéré.

Ces premiers travaux ont mis en évidence les forts potentiels d'accélérations accessibles. Toutefois, le choix des instructions composant les extensions est contraint par l'accès à uniquement 2 opérandes d'entrée dans l'UAL. Ainsi, dans le chapitre suivant, une seconde étude exploitant un cœur RISC-V plus complexe, offrant l'accès à une UAL à trois registres a été réalisée. Ces nouvelles opportunités architecturales ont permis la conception d'extensions plus performantes, tout en étant plus compactes. Le prototypage de ces nouvelles extensions a permis de mettre en exergue une amélioration des facteurs d'accélération pour le décodage des CCE. Ces derniers peuvent atteindre jusqu'à (13.2×) pour une parallélisation inter-trames et (23.5×) pour une parallélisation intra-trame en mode 64 bits, par rapport aux descriptions logicielles initiales sans extensions. Pour le cœur CVA6 étudié, l'augmentation matérielle induite par les extensions vis-à-vis du coût du cœur se limite à 1% à 3% (pour chaque extension). Ce rapport entre l'augmentation de la complexité matérielle et les facteurs d'accélération démontrent la pertinence des choix réalisés.

Perspectives

Les travaux précédemment présentés ouvrent de nouvelles perspectives de recherche sur différents axes et à court, moyen et long terme en fonction de leur complexité. À court terme, différentes actions qui n'ont pas pu être menées durant cette thèse pourraient être réalisées :

— Une description RTL plus efficace des opérateurs pourrait être envisagée. Dans notre étude, aucune factorisation matérielle des ressources mises en œuvre dans nos extensions n'a été réalisée manuellement. Cette tâche d'optimisation a été laissée à l'outil Vivado. Certains résultats expérimentaux démontrent que l'outil ne factorise pas les ressources au niveau RTL comme cela pourrait être attendu. Ce travail pourrait être conduit afin d'améliorer le coût de chacune des extensions et/ou d'une méta-extension regroupant les instructions pour l'ensemble des familles de codes ECC.

— Une étude précise des performances énergétiques liées à l'utilisation de nos extensions pourrait être menée. Des estimations préliminaires via l'outil XPower de Xilinx ont montré un faible impact de nos extensions sur la consommation de puissance sur les architectures déployées sur FPGA. Les facteurs d'accélération mesurés laissent présager des gains énergétiques conséquents, il serait donc intéressant de les caractériser finement.

À moyen terme, d'autres travaux pourraient être envisagés afin de généraliser l'approche proposée dans ce manuscrit :

— une remise en cause des algorithmes de décodage CCE étudiés dans ces travaux. Par exemple, l'algorithme de décodage Extended Min-Sum pourrait être utilisé pour les codes LDPC-NB afin d'améliorer le débit des décodeurs. Les études réalisées autour des algorithmes de décodage polaire SC et F-SC pourraient servir de base pour proposer des extensions pour les différentes variantes de l'algorithme SC-List qui est plus efficace concernant le pouvoir de correction d'erreurs.

  De manière analogue, d'un point de vue matériel, les choix réalisés quand à l'intégration des extensions dans l'UAL pourraient être remis en cause. En effet, d'autres stratégies d'intégration des instructions dans les cœurs pourraient être envisagées : l'utilisation du format d'instruction P pour les instructions vectorielles (SIMD) et de leur file de registre générique pouvant atteindre plusieurs centaines de bits, mais aussi l'utilisation de l'interface de couplage CV-X-IF [209] pour plus de généricité.

À plus long terme, il serait judicieux de faciliter les étapes d'ingénierie qui ont été chronophages lors de ces travaux. En effet, la spécification d'une méthodologie de conception unifiée permettant de (1) modifier les codes logiciels à partir des extensions proposées et (2) de générer les extensions matérielles au niveau RTL permettrait une exploration plus rapide et plus efficace de l'espace de conception. Pour ce faire, il serait possible de réutiliser en partie des travaux de la littérature [182] mais en utilisant une approche plus guidée par le concepteur. À partir de ces travaux, il serait possible d'imaginer des solutions multi-cœurs pour la conception de systèmes de communications avec des cœurs hétérogènes spécialisés.

Annexes

Liste des instructions spécialisées

Synopsis :
Instruction : **i8_abs_pi8** *rD rS1*
Type : R
Code : LDPC
Description :
Retourne la valeur absolue de rS1
Opération :

```
//scalaire
rD[7:0]:=(rS1[7:0] >0 )? rS1[7:0] : -rS1[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := abs8(rS1[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8_add_sat127_pi8** *rD rS1,rS2*
Type : R
Code : LDPC, polaire SC/F-SC
Description :
Addition signée entre rS1 et rS2 puis saturation à +/- 127.
Opération :

```
//scalaire
rD[7:0]:=Sat127( rS1[7:0]+rS2[7:0] )

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := sat127(rS1[i+7:i] + rS2[i+7:i] )
ENDFOR
```

Synopsis :

Instruction : **i8__invB__Aneq1__pi8** *rD rS1,rS2*
Type : R
Code : LDPC
Description :
Retourne rS2 ou - rS2 en fonction du signe de rS1 : si rS1 >=1 : retourne rS2 sinon retourne -rS2.
Opération :

```
//scalaire
rD[7:0]:=(rS1[7:0] >= 1)?rS2[7:0]:-rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := (rS1[7:0] >= 1)?rS2[7:0]:-rS2[7:0]
ENDFOR
```

Synopsis :
Instruction : **i8__xorA__signB__pi8** *rD rS1,rS2*
Type : R
Code : LDPC
Description :
Récupère le signe de rS2 (1:positif et 0:négatif) , rS1 est déja un signe, puis effectue un ou exclusif entre rS1 et rS2 et retourne le resultat.
Opération :

```
//scalaire
rD[7:0]:=rS1[7:0] xor ((rS2[7:0]>=0)?1:0)

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[7:0] xor ((rS2[7:0]>=0)?1:0)
ENDFOR
```

Synopsis :
Instruction : **i8__cmpeq__pi8** *rD rS1,rS2*
Type : R
Code : LDPC
Description :
Comparaison signée entre rS1 et rS2, si rS1 est différent de rS2 retourne 0x00, sinon retourne 0xFF
Opération :

```
//scalaire
rD[7:0]:=(rS1[7:0]==rS2[7:0])?0xff:0x00

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[7:0]:=(rS1[7:0]==rS2[7:0])?0xff:0x00
ENDFOR
```

Synopsis :
Instruction : **i8__max__pi8** *rD rS1,rS2*
Type : R
Code : LDPC,turbo
Description :
Retourne le maximum signé entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= ( rS1[7:0] > rS2[7:0] )? rS1[7:0] :rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := max8(rS1[i+7:i], rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **i8__min__pi8** *rD rS1,rS2*
Type : R
Code : LDPC,turbo
Description :
Retourne le minimum signé entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= ( rS1[7:0] < rS2[7:0] )? rS1[7:0] :rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := min8(rS1[i+7:i] , rS2[i+7:i] )
ENDFOR
```

Synopsis :

Instruction : **i8__sub__sat127__pi8** *rD rS1,rS2*
Type : R
Code : LDPC, polaire SC/F-SC
Description :
Soustraction signée entre rS1 et rS2 puis saturation à +/- 127.
Opération :

```
//scalaire
rD[7:0]:=Saturate127(rS1[7:0]-rS2[7:0])

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := sat127(rS1[i+7:i] - rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **i8__sign__pi8** *rD rS1*
Type : R
Code : LDPC, turbo
Description :
Comparaison signée de rS1 avec 0 ; si inf. à 0 retourne 1 sinon 0 ; signe négatif vaut 1 , signe positif vaut 0 .
Opération :

```
//scalaire
 rD[7:0] := (rS1[7:0]< 0) ? 0x01:0x00)

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := (rS1[i+7:i]< 0) ? 0x01:0x00)
ENDFOR
```

Synopsis :
Instruction : **i8__invC__xorA__signB__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : LDPC
Description :
Ou Exclusif entre rS1 et le signe de rS2, retourne rS3 si résultat vaut 1 sinon -rS3
Opération :

```
//scalaire
rD :=rS1[7:0] ^ (rS2[7:0] > 0) ? rS3[7:0] : -rS3[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD :=rS1[7:0] ^ (rS2[7:0] > 0) ? rS3[7:0] : -rS3[7:0]
ENDFOR
```

Synopsis :
Instruction : **i8__minC__maxAB__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : LDPC
Description :
Maximum entre rS1 et rS2, puis compare et retrourne le min entre le résultat et rS3
Opération :

```
//scalaire
rD := MIN8(MAX8(rS1[7:0], rS2[7:0] ),rS3[7:0] )

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD := MIN8(MAX8(rS1[7:0], rS2[7:0] ),rS3[7:0] )
ENDFOR
```

Synopsis :
Instruction : **i8__blendC__cmpeqAB__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : LDPC
Description :
Génère un masque selon rS1 et rS2 puis utilise ce masque pour trier et retourner les minimums
Opération :

```
//scalaire
Mask := (rS1[7:0] == rS2[7:0])? 0XFF:0x00
min_t := rS1[7:0] &~Mask
min_u := rS2[7:0] &Mask
rD := min_t | min_u

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD :=opération SISD
ENDFOR
```

Synopsis :
Instruction : **i8__Fx__pi8** *rD rS1,rS2*
Type : R
Code : Polaire SC et F-SC
Description :
Application de la fonction F de l'arbre de décodage. Retourne le signe et le minimum entre rS1 et rS2.
Opération :

```
//scalaire
min1 = abs(rS1[7:0] )
min2 = abs(rS2[7:0] )
min1 = min( min1 , min2 )
sign = (rS1[7:0]  <0) ^ (rS2[7:0]  <0)
rD[7:0]  = (sign == 0 )? min1 : -min1

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := f8(rS1[i+7:i], rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **i8__Rx__pi8** *rD rS1,rS2*
Type : R
Code : Polaire SC et F-SC
Description :
Application de la fonction R de l'arbre de décodage, calcul des feuilles de niveau 0 en fonction du vecteur de bit gelées (rS2). Retourne si rS2 eq. 1 sinon si rS1 est inf. 0 retroune 1 sinon retourne 0 .
Opération :

```
//scalaire
rD[7:0] := (rS2[7:0] == 1 )? 0x00 : (rS1[7:0] < 0 ) ? 0x01 :0x00

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := r8(rS1[i+7:i], rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **i8__clrA__Bneq0__pi8** *rD rS1,rS2*
Type : R
Code : Polaire SC
Description :

rS2 vecteur de bit gelés, rS1 Sommes Partielles : Décodage individuel des sommes partielles pour chaque element de la trame Std : Comparaison à 0 de rS2, si valide, retourne rS1 sinon 0 .
Opération :

```
//scalaire
rD[7:0] := (rS2[7:0] == 0 )? rS1[7:0] : 0x00

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := (rS2[i+7:i]==0x00)?  rS1[i+7:i] : 0x00)
ENDFOR
```

Synopsis :
Instruction : **i8_setMask_Aeq1** *rD rS1,rS2*
Type : R
Code : Polaire SC et F-SC
Description :
Utilisé pour la fonction g de l'arbre de décodage. Génère un masque en fonction de rS1. rS1 est le vecteur de signes. Comparaison de rS1 avec 1, si valide, retourne 0xFF sinon 0.
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := (rS1[i+7:i]  ==1) ? 0xff : 0x00
ENDFOR
```

Synopsis :
Instruction : **i8_Gx_pi8** *rD rS1,rS2,rS3*
Type : R4
Code : Polaire SC et F-SC
Description :
Application de la fonction G de l'arbre de décodage. Retourne l'addition saturé de rS1 et rS2 ou alors rS2-rS1.
Opération :

```
//scalaire
Addsat := sat127(rS1+rS2)
subsat := sat127(rS2-rS1)
RD := (rS3==0)?Addsat: subsat

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := g8(rS1[i+7:i], rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **i16__add__accA__loB__pi8** *rD rS1,rS2*
Type : R
Code : Polaire F-SC
Description :
Noeud REP polaire Fast-SC. Additonne les elements 1 et 2 de rS2 dans les upper et lower bit de rS1
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
RD[31:16] = rS1[31:16] + rS2[15:8]
RD[15: 0] = rS1[15: 0] + rS2[7:0]
```

Synopsis :
Instruction : **i16__add__accA__hiB__pi8** *rD rS1,rS2*
Type : R
Code : Polaire F-SC
Description :
Noeud REP polaire Fast-SC. Additonne les elements 3 et 4 de rS2 dans les upper et lower bit de rS1
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
rD[31:16] = rS1[31:16] + rS2[31:24]
rD[15: 0] = rS1[15: 0] + rS2[23:16]
```

Synopsis :
Instruction : **i8__sign__pi16** *rD rS1,rS2*

Type : R
Code : Polaire F-SC
Description :
Noeud REP polaire Fast-SC. Compare les upper et lower bits de rS1 et rS2 et retourne 1 si négatif sinon retourne 0 si positif
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
rD[V3] = ( rS2[31:16]) < 0 ) ? 1 : 0
rD[V2] = ( rS2[15:0 ] ) < 0 ) ? 1 : 0
rD[V1] = ( rS1[31:16]) < 0 ) ? 1 : 0
rD[V0] = ( rS1[15:0] ) < 0 ) ? 1 : 0
```

Synopsis :
Instruction : **i8__Hxor__pi8** *rD rS1,rS2*
Type : R
Code : Polaire F-SC
Description :
Noeud SPC polaire Fast-SC. INTRA-trames. Retourne le résultat d'un Ou exclusif concaténé entre tout les elements de rS2 puis rS1
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
rD = rS1 ^( rS2[7:0] ^ rS2[15:8] ^ rS2[23:16] ^ rS2[31:24] )
```

Synopsis :
Instruction : **i8__Hadd__pi8** *rD rS1,rS2*
Type : R
Code : Polaire F-SC
Description :
Noeud REP polaire Fast-SC INTRA-trames. Retourne le résultat d'un AND concaténé entre tout les elements de rS2 puis rS1
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
rD = rS1 + (rS2[7:0]+rS2[15:8]+rS2[23:16]+rS2[31:24] )
```

Synopsis :
Instruction : **u8__minu__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Retourne le minimum non-signé entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= ( rS1[7:0] < rS2[7:0] )? rS1[7:0] :rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := min8u(rS1[i+7:i] , rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **u8__addu__sat64__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Retourne l'addition saturée à +64/0 entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0] :=  saturate64( rS1 - rS2 )

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := saturate64( rS1[i+7:i] - rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **u8__subu__sat64__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Retourne la soustraction saturée à +64/0 entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0] :=  saturate64( rS1 - rS2 )

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := saturate64( rS1[i+7:i] - rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **u8__cmpge__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Part de la fonction vectorisé de recherche de minima. Comparaison signée entre rS1 et rS2 pour générer un masque en fonction du min. Si rS1 sup. ou égal à rS2 , retourne xFF, sinon retourne 0.
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := ( rS1[i+7:i] >= rS2[i+7:i] ) ? 0xff:0x00
ENDFOR
```

Synopsis :
Instruction : **u8__Aandb__lo8B__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Part de la fonction vectorisé de recherche de minima. Applique un et logique entre chaque element de rS1 avec le premier element 8b de rS1
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := ( rS1[i+7:i] & rS2[7:0] )
ENDFOR
```

Synopsis :
Instruction : **u8__maxu__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Maximum non signé entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= ( rS1[7:0] > rS2[7:0] )? rS1[7:0] :rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[i+7:i]:= ( rS1[i+7:i] > rS2[i+7:i] )? rS1[i+7:i] :rS2[i+7:i]
ENDFOR
```

Synopsis :
Instruction : **u8__hminu__pu8** *rD rS1,rS2*
Type : R
Code : LDPC-NB
Description :
Recherche du minimum horizontal Renvoie le minimum des 4 elements de 8b de rS1, et le compare avec rS2, pour renovyer le min absolu.
Opération :

```
//scalaire
simd only

//SIMD de degré N
FOR j := 0 to N
i := j*8
rD[i+7:i] := ( rS1[i+7:i] & rS2[i+7:i] )
ENDFOR
```

Synopsis :
Instruction : **u8__min__addsat__pu8** *rD rS1,rS2,rS3*
Type : R4
Code : LDPC-NB
Description :
Minimum entre rS3 et addition saturée de rS1 et rS2
Opération :

```
//scalaire
rD[7:0] := Min(Sat63(rS1[7:0]+rS2[7:0]), rS3[7:0])

//SIMD de degré N
FOR j := 0 to N
i := j*8
rD[i+7:i] := Min(Sat63(rS1[i+7:i]+rS2[i+7:i]), rS3[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **u8__cmpge__AndB__lo8C__pu8** *rD rS1,rS2,rS3*
Type : R4
Code : LDPC-NB
Description :
Comparaison égale ou supérieure de rS1 et rS2, retourne les 8 LSB de rS3
Opération :

```
//scalaire
rD[7:0] :=(rS1[7:0] >=rS2[7:0] )? rS3[7:0]:0x00

//SIMD de degré N
FOR j := 0 to N
i := j*8
rD[[i+7:i]] :=(rS1[i+7:i] >=rS2[i+7:i] )? rS3[i+7:i]:0x00
ENDFOR
```

Synopsis :
Instruction : **i8__scale__pi8** *rD rS1*
Type : R
Code : turbo
Description :
Retourne rS1*0.75 /coefficient 0.75
Opération :

```
//scalaire
( rS1 > 0 )? 1:0
A  = abs(rS1)
B  = A>>2
C  = A-B
rD := (Sign==0)?-C:C

//SIMD de degré N
FOR j := 0 to N
i := j*8
rD[i+7:i] := scale(rS1[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8__add__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Addition signée entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]+rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] + rS2[i+7:i]
ENDFOR
```

Synopsis :
Instruction : **i8__sub__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Soustraction signée entre rS1 et rS2
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]-rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] - rS2[i+7:i]
ENDFOR
```

Synopsis :
Instruction : **i8_add_srl_pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Addition entre rS1 et rS2 puis shift logique droit de 1
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]+rS2[7:0] >>1

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] + rS2[i+7:i] >>1
ENDFOR
```

Synopsis :
Instruction : **i8_srl1_pi8** *rD rS1*
Type : R
Code : turbo
Description :
Shift logique droit de 1 de rS1
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]>>1

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] >>1
ENDFOR
```

Synopsis :

Instruction : **i8__shuffle__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Shuffle, selection des elements à retourner de rS1 avec l'index contenu dans rS2
Opération :

```
//scalaire
SIMD uniquement

//SIMD de degré N
FOR j := 0 to N
I := j*8
        IF rS2[j+7] == 1
              rD[j+7:i] := 0
        ELSE
              index[3:0] := rS2[j+3:j]
              RD[j+7:i] := rS1[index*8+7:index*8]
        FI
ENDFOR
```

Synopsis :
Instruction : **i8__add__div2__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Addition entre rS1 et rS2 puis shift logique droit de 1 clone de add_srl
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]+rS2[7:0] >>1

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] + rS2[i+7:i] >>1
ENDFOR
```

Synopsis :
Instruction : **i8__sub__div2__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Soustraction entre rS1 et rS2 puis shift logique droit de 1
Opération :

```
//scalaire
rD[7:0]:= rS1[7:0]-rS2[7:0] >>1

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] - rS2[i+7:i] >>1
ENDFOR
```

Synopsis :
Instruction : **i8__sat__sub__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
sature rS1 puis le soustraction entre le résultat et rS2
Opération :

```
//scalaire
rD[7:0]:= sat63(rS1[7:0])-rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[i+7:i]:= sat63(rS1[i+7:i])-rS2[i+7:i]
ENDFOR
```

Synopsis :
Instruction : **i8__scale__add__pi8** *rD rS1,rS2*
Type : R
Code : turbo
Description :
Effectue l'instruction de scale à 0.75 sur rS1 puis additionne le résultat avec rS2
Opération :

```
//scalaire
RD[7:0] = scale0.75(rS1[7:0]) + rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[i+7:i]:= scale(rS1[i+7:i]) + rS2[i+7:i]
ENDFOR
```

Synopsis :

Instruction : **i8__accumax__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : turbo
Description :
Maximum accumulée de rS1, rS2 et rS3
Opération :

```
//scalaire
rD[7:0] := MAX8( rS1[7:0], rS2[7:0], rS3[7:0])

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] :=MAX8( rS1[i+7:i], rS2[i+7:i], rS3[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8__maxpm__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : turbo
Description :
Maximum entre rS1+rS2 et rS3-rS2
Opération :

```
//scalaire
rD[7:0] := MAX8( rS1[7:0] + rS2[7:0], rS3[7:0] –rS2[7:0])

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] :=MAX8( rS1[i+7:i] + rS2[i+7:i], rS3[i+7:i] –rS2[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8__accupp__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : turbo
Description :
Addition accumulée de rS1, rS2 et rS3
Opération :

```
//scalaire
rD[7:0] := rS1[7:0] + rS2[7:0] + rS3[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] + rS2[i+7:i] + rS3[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8__accump__pi8** *rD rS1,rS2,rS3*
Type : R4
Code : turbo
Description :
Addition et soustraction accumulée
Opération :

```
//scalaire
rD[7:0] := rS1[7:0] - rS2[7:0] + rS3[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] - rS2[i+7:i] + rS3[i+7:i])
ENDFOR
```

Synopsis :
Instruction : **i8__srl__pi8** *rD rS1,rS2,rS3*
Type : R
Code : turbo
Description :
Shift logique droit signé de rS1 avec rS2
Opération :

```
//scalaire
rD[7:0] := rS1[7:0] >>rS2[7:0]

//SIMD de degré N
FOR j := 0 to N
 i := j*8
 rD[i+7:i] := rS1[i+7:i] >> rS1[i+7:i]
ENDFOR
```

Synopsis :

Instruction : **i8__cpysign__pi8** *rD rS1,rS2,rS3*
Type : R
Code : ldpc
Description :
Renvoie rS2 basé sur le signe de rS1
Opération :

```
//scalaire
rD[7:0] = (rS1[7:0] ) < 0 )? -rS2[7:0] : (rS1[7:0] ) >= 0 )? rS2[7:0] : 8'h00

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[i+7:i] = (rS1[i+7:i] ) < 0 )? -rS2[i+7:i] : (rS1[i+7:i] ) >= 0 )? rS2[i+7:i] : 8'h00
ENDFOR
```

Synopsis :
Instruction : **i8__storeu__pi8** *rD rS1,rS2,rS3*
Type : R
Code : ldpc
Description :
Xor signe rS1 avec combinaison de rS2
Opération :

```
//scalaire
RD[7:0]={rS1[7] ,1'b0} ^( (rS2[0]==0)? 2'b01 : 2'b11 ) ;

//SIMD de degré N
FOR j := 0 to N
 i := j*8
rD[i+7:i] ={rS1[i+7] ,1'b0} ^( (rS2[0]==0)? 2'b01 : 2'b11 ) ;
ENDFOR
```

Bibliographie

[1] 3GPP, « Overview of 3GPP Release 99 », rapp. tech., 1999. adresse : https://www.3gpp.org/specifications-technologies/releases/release-1999.

[2] 3GPP, « Overview of 3GPP Release 8 V0.3.3 », rapp. tech., 2014. adresse : https://www.3gpp.org/specifications-technologies/releases/release-8.

[3] F. Krasniqi, A. Maraj et E. Blaka, « Performance analysis of mobile 4G/LTE networks », in *2018 South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference (SEEDA-CECNSM)*, 2018, pp. 1-5.

[4] S. Baek, D. Kim, M. Tesanovic et A. Agiwal, « 3GPP New Radio Release 16 : Evolution of 5G for Industrial Internet of Things », *IEEE Communications Magazine*, Vol. 59, pp. 41-47, 2021.

[5] W. Jiang, B. Han, M. A. Habibi et H. D. Schotten, « The Road Towards 6G : A Comprehensive Survey », *IEEE Open Journal of the Communications Society*, Vol. 2, pp. 334-366, 2021.

[6] *IEEE Standard for Information technology - Part 11 : Wireless LAN Medium Access Control and Physical Layer Specifications*, IEEE Std 802.11ah, 2016, pp. 1-594.

[7] *IEEE Standard for Local and metropolitan area networks - Part 16 : Air Interface for Broadband Wireless Access Systems*, IEEE Std 802.16-2009, 2009, pp. 1-2080.

[8] 3GPP, « Technical specification ; 5G ; NR ; Multiplexing and channel coding (3GPP TS 38.212 version 15.2.0 Release 15) », juill. 2018.

[9] C. E. Shannon, « A mathematical theory of communication », *The Bell System Technical Journal*, Vol. 27, no. 3, pp. 379-423, juill. 1948.

[10] C. Berrou, *Codes and Turbo Codes*, C. Berrou, éd. Springer Paris, jan. 2010, Vol. 1, pp. 400.

[11] C. Berrou, A. Glavieux et P. Thitimajshima, « Near Shannon limit error-correcting coding and decoding : Turbo-codes. », in *Proceedings of the IEEE International Conference on Communications 1993*, Vol. 2, Geneva, Switzerland : IEEE, mai 1993, pp. 1064-1070.

[12] R. W. Hamming, « Error detecting and error correcting codes », *The Bell System Technical Journal*, Vol. 29, no. 2, pp. 147-160, avr. 1950.

[13] I. S. Reed et G. Solomon, « Polynomial Codes Over Certain Finite Fields », *Journal of the Society for Industrial and Applied Mathematics*, Vol. 8, no. 2, pp. 300-304, juin 1960.

[14] R. Bose et D. Ray-Chaudhuri, « On a class of error correcting binary group codes », *Information and Control*, Vol. 3, pp. 68-79, avr. 1960.

[15] J. Proakis, *Digital Communications* (Electrical engineering series), 4e éd. McGraw-Hill, 2001, isbn : 9780072321111.

[16] *Digital Video Broadcasting (DVB) ; Interaction channel for Satellite Distribution Systems.*

[17] Consultative Committee for Space Data Systems (CCSDS), *CCSDS 231.1-O-1 - Orange Book - Short block length LDPC codes for tc synchronization and channel coding*, Washington, DC, USA, avr. 2015.

[18] Consultative Committee for Space Data Systems (CCSDS), *CCSDS 230.2-G-1 Green Book - Next Generation Uplink - Informational report*, Washington, DC, USA.

[19] D. D. A83-2, « Digital video broadcasting (DVB) - part II : S2-extensions (DVB-s2x) march », rapp. tech., 2014.

[20] C. Berrou, A. Glavieux et P. Thitimajshima, « Near Shannon limit error-correcting coding and decoding : Turbo-codes. », in *Proceedings of the IEEE International Conference on Communications 1993*, Vol. 2, Geneva, Switzerland : IEEE, mai 1993, pp. 1064-1070 vol.2.

[21] C. Berrou, « The ten-year-old turbo codes are entering into service », *IEEE Communications Magazine*, Vol. 41, no. 8, pp. 110-116, août 2003.

[22] Consultative Committee for Space Data Systems (CCSDS), *CCSDS 101.0-B-4 - Blue Book - Recommendation for space data system standards - Telemetry channel coding*, mai 1999.

[23] M. F. Brejza, L. Li, R. G. Maunder, B. M. Al-Hashimi, C. Berrou et L. Hanzo, « 20 Years of Turbo Coding and Energy-Aware Design Guidelines for Energy-Constrained Wireless Applications », *IEEE Communications Surveys & Tutorials*, Vol. 18, no. 1, pp. 8-28, 2016.

[24] R. Garzón-Bohórquez, C. A. Nour et C. Douillard, « Improving Turbo Codes for 5G with parity puncture-constrained interleavers », in *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Brest, France, sept. 2016, pp. 151-155.

[25] O. Muller, A. Baghdadi et M. Jezequel, « Parallelism Efficiency in Convolutional Turbo Decoding », *EURASIP Journal on Advances in Signal Processing*, pp. 11, 2010.

[26] S. Weithoffer, R. Klaimi, C. A. Nour, N. Wehn et C. Douillard, « Fully Pipelined Iteration Unrolled Decoders the Road to TB/S Turbo Decoding », in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, mai 2020, pp. 5115-5119.

[27] A. Li, P. Hailes, R. G. Maunder, B. M. Al-Hashimi et L. Hanzo, « 1.5 Gbit/s FPGA Implementation of a Fully-Parallel Turbo Decoder Designed for Mission-Critical Machine-Type Communication Applications », *IEEE Access*, Vol. 4, pp. 5452-5473, août 2016.

[28] L. Li, R. G. Maunder, B. M. Al-Hashimi et L. Hanzo, « A Low-Complexity Turbo Decoder Architecture for Energy-Efficient Wireless Sensor Networks », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 21, no. 1, pp. 14-22, jan. 2013.

[29] O. Muller, A. Baghdadi et M. Jezequel, « From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 17, no. 1, pp. 92-102, jan. 2009.

[30] L. Bahl, J. Cocke, F. Jelinek et J. Raviv, « Optimal decoding of linear codes for minimizing symbol error rate (Corresp.) », *IEEE Transactions on Information Theory*, Vol. 20, no. 2, pp. 284-287, mars 1974.

[31] P. Robertson, P. Hoeher et E. Villebrun, « Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding », *European Transactions on Telecommunications*, Vol. 8, no. 2, pp. 119-125, 1997.

[32] R. Garzón-Bohórquez, C. Abdel Nour et C. Douillard, « Protograph-Based Interleavers for Punctured Turbo Codes », *IEEE Transactions on Communications*, Vol. 66, no. 5, pp. 1833-1844, mai 2018.

[33] R. Gallager, « Low-Density Parity-Check Codes », *IRE Transactions on Information Theory*, Vol. 8, no. 1, pp. 21-28, jan. 1962.

[34] D. J. MacKay et R. M. Neal, « Near Shannon limit performance of low density parity check codes », *Electronics letters*, Vol. 33, no. 6, pp. 457-458, 1997.

[35] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi et L. Hanzo, « A Survey of FPGA-Based LDPC Decoders », *IEEE Communications Surveys Tutorials*, Vol. 18, no. 2, pp. 1098-1122, déc. 2016.

[36] A. K. Panigrahi et A. Panda, « Design of LDPC Decoder Using FPGA : Review of Flexibility », *International Journal of Engineering Science*, Vol. 1, pp. 2278-4721, nov. 2012.

[37] G. Liva, S. Song, L. Lan, Y. Zhang, S. Lin et W. E. Ryan, « Design of LDPC Codes : A Survey and New Results », *Journal of communications software and systems*, Vol. 2, pp. 191, 2017.

[38] D. G. M. Mitchell, M. Lentmaier, A. E. Pusane et D. J. Costello, « Randomly Punctured LDPC Codes », *IEEE Journal on Selected Areas in Communications*, Vol. 34, no. 2, pp. 408-421, fév. 2016.

[39] M. Fossorier, M. Mihaljevic et H. Imai, « Reduced complexity iterative decoding of low-density parity check codes based on belief propagation », *IEEE Transactions on Communications*, Vol. 47, no. 5, pp. 673-680, 1999.

[40] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi et L. Hanzo, « A Survey of FPGA-Based LDPC Decoders », *IEEE Communications Surveys Tutorials*, Vol. 18, no. 2, pp. 1098-1122, déc. 2016.

[41] IEEE Std 802.11ah, *IEEE Standard for Information technology - Part 11 : Wireless LAN Medium Access Control and Physical Layer Specifications*, 2016, pp. 1-594.

[42] European Telecommunications Standards Institute, « EN 302 307 V1. 1.1, Digital video broadcasting (DVB) ; second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broad-band satellite applications. », rapp. tech.

[43] R. Tanner, « A recursive approach to low complexity codes », *IEEE Transactions on Information Theory*, Vol. 27, no. 5, pp. 533-547, 1981.

[44] R. Gallager, « Low-Density Parity-Check Codes », *IRE Transactions on Information Theory*, Vol. 8, no. 1, pp. 21-28, jan. 1962.

[45] B. Le Gal, C. Jego et C. Leroux, « A Flexible NISC-Based LDPC Decoder », *IEEE Transactions on Signal Processing*, Vol. 62, no. 10, pp. 2469-2479, mai 2014.

[46] *IEEE Local and Metropolitan Area Networks - Specific Requirements - Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std P802.11ad, 2007.

[47] verizon, *V. Wireless, Verizon 5G TF ; Air Interface Working Group ; Verizon 5th Generation Radio Access ; Multiplexing and channel coding (Release 1), TS V5G.212 V1.2*, 2016.

[48] M. Fossorier, « Quasicyclic low-density parity-check codes from circulant permutation matrices », *IEEE Transactions on Information Theory*, Vol. 50, no. 8, pp. 1788-1793, août 2004.

[49] R. Gallager, « Low-Density Parity-Check Codes », *IRE Transactions on Information Theory*, Vol. 8, no. 1, pp. 21-28, jan. 1962.

[50] X. Wu, Y. Song, M. Jiang et C. Zhao, « Adaptive-Normalized/Offset Min-Sum Algorithm », *IEEE Communications Letters*, Vol. 14, no. 7, pp. 667-669, juill. 2010.

[51] C. Marchand, « Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards », thèse de doct., Université de Bretagne Sud, Lorient, France, déc. 2010.

[52] R. Maunder, *Survey of ASIC implementations of LDPC decoders*, 2016. adresse : https://eprints.soton.ac.uk/399259/.

[53] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi et L. Hanzo, « A Survey of FPGA-Based LDPC Decoders », *IEEE Communications Surveys Tutorials*, Vol. 18, no. 2, pp. 1098-1122, déc. 2016.

[54] T. Nguyen-Ly et al., « FPGA design of high throughput LDPC decoder based on imprecise Offset Min-Sum decoding », in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, Grenoble, France : IEEE, juin 2015, pp. 1-4.

[55] D. Hocevar, « A reduced complexity decoder architecture via layered decoding of LDPC codes », in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS)*, Austin, TX, USA, oct. 2004.

[56] C. Marchand et E. Boutillon, « LDPC decoder architecture for DVB-S2 and DVB-S2X standards », in *IEEE Workshop on Signal Processing Systems (SIPS)*, Hangzhou, China, oct. 2015.

[57] B. L. Gal, C. Jego et J. Crenne, « A High Throughput Efficient Approach for Decoding LDPC Codes onto GPU Devices », *IEEE Embedded Systems Letters*, Vol. 6, no. 2, pp. 29-32, juin 2014.

[58] B. Le Gal et C. Jego, « Low-latency software LDPC decoders », in *Proceedings of the IEEE International Workshop on Signal Processing Systems (SIPS)*, Lorient, France, oct. 2017.

[59] G. Wang, M. Wu, B. Yin et J. R. Cavallaro, « High throughput low latency LDPC decoding on GPU for SDR systems », in *2013 IEEE Global Conference on Signal and Information Processing*, Austin, TX, USA : IEEE, déc. 2013, pp. 1258-1261.

[60] B. Le Gal et C. Jego, « High-Throughput Multi-Core LDPC Decoders Based on x86 Processor », *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, no. 5, pp. 1373-1386, mai 2016.

[61] V. Pignoly, B. Le Gal, C. Jego et B. Gadat, « High data rate and flexible hardware QC-LDPC decoder for satellite optical communications », in *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, Hong Kong, China, déc. 2018, pp. 1-5.

[62] C. Marchand et E. Boutillon, « LDPC decoder architecture for DVB-S2 and DVB-S2X standards », in *IEEE Workshop on Signal Processing Systems (SIPS)*, Hangzhou, China, oct. 2015.

[63] B. Le Gal et C. Jego, « High-Throughput Multi-Core LDPC Decoders Based on x86 Processor », *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, no. 5, pp. 1373-1386, mai 2016.

[64] V. Pignoly, B. Le Gal, C. Jégo, B. Gadat et L. Barthe, « Fair comparison of hardware and software LDPC decoder implementations for SDR space links », in *Proceedings of the IEEE International Conference on Electronics Circuits and Systems Conference (ICECS)*, Glasgow, UK, nov. 2020.

[65] E. Grayver, *Implementing Software Defined Radio*. Springer, 2013.

[66] R. Akeela et B. Dezfouli, « Software-defined Radios : Architecture, state-of-the-art, and challenges », *Computer Communications*, Vol. 128, pp. 106-125, 2018.

[67] Fujitsu Inc., « The Benefits of Cloud-RAN Architecture in Mobile Network Expansion », Fujitsu Network Communications Inc., rapp. tech., 2015.

[68] A. Checko et al., « Cloud RAN for Mobile Networks - A Technology Overview », *IEEE Communications Surveys Tutorials*, Vol. 17, no. 1, pp. 405-426, 2015.

[69] D. Wubben et al., « Benefits and Impact of Cloud Computing on 5G Signal Processing : Flexible centralization through Cloud-RAN », *IEEE Signal Processing Magazine*, Vol. 31, no. 6, pp. 35-44, nov. 2014.

[70] M. Davey et D. MacKay, « Low-density parity check codes over GF(q) », *IEEE Communications Letters*, Vol. 2, no. 6, pp. 165-167, juin 1998.

[71] L. Barnault et D. Declercq, « Fast decoding algorithm for LDPC over GF(2q) », in *Proceedings 2003 IEEE Information Theory Workshop (Cat. No.03EX674)*, Paris, France, mars 2003, pp. 70-73.

[72] L. Dolecek, D. Divsalar, Y. Sun et B. Amiri, « Non-Binary Protograph-Based LDPC Codes : Enumerators, Analysis, and Designs », *IEEE Transactions on Information Theory*, Vol. 60, no. 7, pp. 3913-3941, juill. 2014.

[73] O. Ferraz et al., « A Survey on High-Throughput Non-Binary LDPC Decoders : ASIC, FPGA, and GPU Architectures », *IEEE Communications Surveys & Tutorials*, Vol. 24, no. 1, pp. 524-556, 2022.

[74] R. A. Carrasco et M. Johnston, *Non-binary Error Control Coding for Wireless Communications and Data Storage*. John Wiley & Sons, jan. 2008, isbn : 978-0-470-51819-9.

[75] B. Le Gal et C. Jego, « High-Throughput FFT-SPA Decoder Implementation for Non-Binary LDPC Codes on x86 Multicore Processors », *Journal of Signal Processing Systems*, Vol. 92, pp. 37-53, mars 2020.

[76] J. Andrade, G. Falcao, V. Silva et K. Kasai, « FFT-SPA non-binary LDPC decoding on GPU », in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 5099-5103.

[77] J. Andrade, G. Falcao et V. Silva, « Optimized Fast Walsh–Hadamard Transform on GPUs for non-binary LDPC decoding », *Parallel Computing*, Vol. 40, no. 9, pp. 449-453, 2014.

[78] H. T. Pham, S. Ajaz et H. Lee, « Parallel block-layered nonbinary QC-LDPC decoding on GPU », in *2015 IEEE Workshop on Signal Processing Systems (SIPS)*, 2015, pp. 1-6.

[79] L. Barnault et D. Declercq, « Fast decoding algorithm for LDPC over GF(2/sup q/) », in *Proceedings 2003 IEEE Information Theory Workshop (Cat. No.03EX674)*, Paris, France : IEEE, mars 2003, pp. 70-73.

[80] D. Declercq et M. Fossorier, « Decoding Algorithms for Nonbinary LDPC Codes Over GF(q) », *IEEE Transactions on Communications*, Vol. 55, no. 4, pp. 633-643, 2007.

[81] V. Savin, « Min-Max decoding for non binary LDPC codes », in *2008 IEEE International Symposium on Information Theory*, juill. 2008, pp. 960-964.

[82] E. Boutillon, L. Conde-Canencia et A. Al Ghouwayel, « Design of a GF(64)-LDPC Decoder Based on the EMS Algorithm », *IEEE Transactions on Circuits and Systems I : Regular Papers*, Vol. 60, no. 10, pp. 2644-2656, oct. 2013.

[83] A. Voicila, F. Verdier, D. Declercq, M. Fossorier et P. Urard, « Architecture of a low-complexity non-binary LDPC decoder for high order fields », in *2007 International Symposium on Communications and Information Technologies*, oct. 2007, pp. 1201-1206.

[84] H. Harb, « Design of ultra high throughput rate NB-LDPC decoder », Thèse, Université de Bretagne Sud ; Université Libanaise, nov. 2018. adresse : https://theses.hal.science/tel-02136786.

[85] *BeiDou Navigation Satellite System, Signal In Space,Interface Control Document, Open Service Signal B1C (Version 1.)*

[86] C. Monière, « Real-Time implementation of Quasi-Cyclic Short Packet Receiver », Thèse, Universite de Bretagne-Sud, jan. 2023. adresse : https://hal.science/tel-04088743.

[87] K. Saied, « Quasi-Cyclic Short Packet (QCSP) Transmission for IoT », thèse de doct., Université Bretagne Sud, 2022.

[88] E. Arikan, « Channel Polarization : A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memory less Channels », *IEEE Transactions on Information Theory*, Vol. 55, no. 7, pp. 3051-3073, juill. 2009.

[89] K. Niu, P. Zhang, J. Dai, Z. Si et C. Dong, « A golden decade of polar codes : From basic principle to 5G applications », *China Communications*, Vol. 20, no. 2, pp. 94-121, fév. 2023.

[90] S. A. Hashemi, « Fast, Flexible, and Area-Efficient Decoders for Polar Codes », thèse de doct., déc. 2018.

[91] A. Raymond et W. Gross, « A Scalable Successive-Cancellation Decoder for Polar Codes », *Signal Processing, IEEE Transactions on*, Vol. 62, pp. 5339-5347, oct. 2014.

[92] K. Niu, K. Chen, J. Lin et Q. T. Zhang, « Polar codes : Primary concepts and practical decoding algorithms », *IEEE Communications Magazine*, Vol. 52, no. 7, pp. 192-203, juill. 2014.

[93] A. Alamdar-Yazdi et F. R. Kschischang, « A Simplified Successive-Cancellation Decoder for Polar Codes », *IEEE Communications Letters*, Vol. 15, no. 12, pp. 1378-1380, déc. 2011.

[94] G. Sarkis, P. Giard, A. Vardy, C. Thibeault et W. J. Gross, « Fast Polar Decoders : Algorithm and Implementation », *IEEE Journal on Selected Areas in Communications*, Vol. 32, no. 5, pp. 946-957, mai 2014.

[95] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault et W. J. Gross, « Fast Low-Complexity Decoders for Low-Rate Polar Codes », *Journal of Signal Processing Systems*, Vol. 90, no. 5, pp. 675-685, août 2016. adresse : https://doi.org/10.1007%2Fs11265-016-1173-y.

[96] I. Tal et A. Vardy, « List Decoding of Polar Codes », *IEEE Transactions on Information Theory*, Vol. 61, no. 5, pp. 2213-2226, mai 2015.

[97] K. Niu et K. Chen, « CRC-Aided Decoding of Polar Codes », *IEEE Communications Letters*, Vol. 16, no. 10, pp. 1668-1671, oct. 2012.

[98] W. Song et al., « Efficient Successive Cancellation Stack Decoder for Polar Codes », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, no. 11, pp. 2608-2619, 2019.

[99] O. Afisiadis, A. Balatsoukas-Stimming et A. Burg, « A low-complexity improved successive cancellation decoder for polar codes », in *2014 48th Asilomar Conference on Signals, Systems and Computers*, nov. 2014, pp. 2116-2120.

[100] Y. Zhou, J. Lin et Z. Wang, « Improved Fast-SSC-Flip Decoding of Polar Codes », *IEEE Communications Letters*, Vol. 23, no. 6, pp. 950-953, juin 2019.

[101] S. Lee, J. Park, I.-M. Kim et J. Heo, « Flexible soft-output decoding of polar codes », *EURASIP Journal on Wireless Communications and Networking*, Vol. 2021, no. 1, pp. 170, août 2021, issn : 1687–1499. adresse : https://doi.org/10.1186/s13638-021-02042-x.

[102] A. J. Raymond et W. J. Gross, « Scalable successive-cancellation hardware decoder for polar codes », in *2013 IEEE Global Conference on Signal and Information Processing*, déc. 2013, pp. 1282-1285.

[103] B. Le Gal, C. Leroux et C. Jego, « A scalable 3-phase polar decoder », in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, mai 2016, pp. 417-420.

[104] O. Dizdar et E. Arikan, « A High-Throughput Energy-Efficient Implementation of Successive Cancellation Decoder for Polar Codes Using Combinational Logic », *IEEE Transactions on Circuits and Systems I : Regular Papers*, Vol. 63, no. 3, pp. 436-447, mars 2016.

[105] E. Arikan, « Channel Polarization : A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memory less Channels », *IEEE Transactions on Information Theory*, Vol. 55, no. 7, pp. 3051-3073, juill. 2009.

[106] A. Cassagne, B. Le Gal, C. Leroux, O. Aumage et D. Barthou, « An Efficient, Portable and Generic Library for Successive Cancellation Decoding of Polar Codes », sept. 2015.

[107] B. Le Gal, C. Leroux et C. Jego, « Multi-Gb/s Software Decoding of Polar Codes », *IEEE Transactions on Signal Processing*, Vol. 63, no. 2, pp. 349-359, jan. 2015.

[108] P. Giard, G. Sarkis, C. Leroux, C. Thibeault et W. Gross, « Low-Latency Software Polar Decoders », *Journal of Signal Processing Systems*, Vol. 90, mai 2018.

[109] C. Leroux, I. Tal, A. Vardy et W. J. Gross, « Hardware architectures for successive cancellation decoding of polar codes », in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mai 2011, pp. 1665-1668.

[110] M. Hanif et M. Ardakani, « Fast Successive-Cancellation Decoding of Polar Codes : Identification and Decoding of New Nodes », *IEEE Communications Letters*, Vol. 21, no. 11, pp. 2360-2363, nov. 2017.

[111] Y. Delomier, « Conception et prototypage de décodeurs de codes correcteurs d'erreurs à partir de modèles comportementaux », Theses, Université de Bordeaux, juin 2020. adresse : https://theses.hal.science/tel-02935204.

[112] M. Léonardon, A. Cassagne, C. Leroux, C. Jégo, L.-P. Hamelin et Y. Savaria, « Fast and Flexible Software Polar List Decoders », *Journal of Signal Processing Systems*, Vol. 91, no. 8, pp. 937-952, août 2019.

[113] Z. Liu, R. Liu et H. Zhang, « High-Throughput Adaptive List Decoding Architecture for Polar Codes on GPU », *IEEE Transactions on Signal Processing*, Vol. 70, pp. 878-889, 2022.

[114] B. Tahir, S. Schwarz et M. Rupp, « BER comparison between Convolutional, Turbo, LDPC, and Polar codes », in *2017 24th International Conference on Telecommunications (ICT)*, Limassol, Cyprus, mai 2017, pp. 1-7.

[115] A. Balatsoukas-Stimming, P. Giard et A. Burg, « Comparison of Polar Decoders with Existing Low-Density Parity-Check and Turbo Decoders », in *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, San Francisco, CA, USA, avr. 2017, pp. 1-6.

[116] M. C. Coşkun et al., « Efficient error-correcting codes in the short blocklength regime », *Physical Communication*, Vol. 34, pp. 66-79, 2019.

[117] « Technical specification ; 5G ; NR ; Multiplexing and channel coding (3GPP TS 38.212 version 15.2.0 Release 15), 3GPP, July 2018. », rapp. tech.

[118] P. Lynggaard et K. E. Skouby, « Deploying 5G-Technologies in Smart City and Smart Home Wireless Sensor Networks with Interferences », Vol. 81, no. 4, pp. 1399-1413, avr. 2015.

[119] N.-S. Vo, T. Q. Duong, H. D. Tuan et A. Kortun, « Optimal Video Streaming in Dense 5G Networks With D2D Communications », *IEEE Access*, Vol. 6, pp. 209-223, 2018.

[120] C. Campolo, A. Molinaro, A. Iera et F. Menichella, « 5G Network Slicing for Vehicle-to-Everything Services », *IEEE Wireless Communications*, Vol. 24, no. 6, pp. 38-45, déc. 2017.

[121] G. M. C. X. Mavromoustakis et J. M. Batalla, *Internet of Things (IoT) in 5G Mobile Technologies*. Springer, 2016.

[122] INTEL, « FlexRAN™ Reference Architecture for Wireless Access », rapp. tech. adresse : https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/tools/flexran.html.

[123] S. Shao et al., « Survey of Turbo, LDPC, and Polar Decoder ASIC Implementations », *IEEE Communications Surveys & Tutorials*, Vol. 21, no. 3, pp. 2309-2333, 2019.

[124] V. H. S. Le, « Design of next-generation Tb/s turbo codes », Theses, Ecole nationale supérieure Mines-Télécom Atlantique, mars 2021. adresse : https://theses.hal.science/tel-03617216.

[125] A. Süral, E. G. Sezer, E. Kolağasioğlu, V. Derudder et K. Bertrand, « Tb/s polar successive cancellation decoder 16nm ASIC implementation », *arXiv preprint arXiv:2009.09388*, 2020.

[126] R. Ghanaatian et al., « A 588-Gb/s LDPC Decoder Based on Finite-Alphabet Message Passing », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 26, no. 2, pp. 329-340, fév. 2018.

[127] S. Weithoffer, C. A. Nour, N. Wehn, C. Douillard et C. Berrou, « 25 Years of Turbo Codes : From Mb/s to beyond 100 Gb/s », in *Proceedings of the IEEE International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Hong Kong, China, déc. 2018, pp. 1-6.

[128] G. Martin et G. Smith, « High-Level Synthesis : Past, Present, and Future », *IEEE Design & Test of Computers*, Vol. 26, no. 4, pp. 18-25, juill. 2009.

[129] A. Parker et al., « The CMU Design Automation System - An Example of Automated Data Path Design », jan. 1979, pp. 73-80.

[130] Springer, *High-Level Synthesis : From Algorithm to Digital Circuit*. Springer, 2008.

[131] AMD-XILINX, « Vitis Software Platform Development Environment », rapp. tech. adresse : https://www.xilinx.com/products/design-tools/vitis.html.

[132] MENTOR, « CATAPULT-C », rapp. tech., 2024. adresse : https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/hls/c-cplus/.

[133] G. Choi, K.-B. Park et K.-S. Chung, « Optimization of FPGA-Based LDPC Decoder Using High-Level Synthesis », in *Proceedings of the 4th International Conference on Communication and Information Processing*, sér. ICCIP '18, Qingdao, China : Association for Computing Machinery, 2018, pp. 256-259.

[134] Y. Delomier, B. Le Gal, J. Crenne et C. Jego, « Model-based design of flexible and efficient LDPC decoders on FPGA devices », *Journal of Signal Processing Systems*, Vol. 92, pp. 727-745, 2020.

[135] Y.-F. Zhang, L. Sun et Q. Cao, « TLP-LDPC : Three-Level Parallel FPGA Architecture for Fast Prototyping of LDPC Decoder Using High-Level Synthesis », *Journal of Computer Science and Technology*, Vol. 37, pp. 1290-1306, 2022.

[136] J. Andrade et al., « Design Space Exploration of LDPC Decoders Using High-Level Synthesis », *IEEE Access*, Vol. 5, pp. 14600-14615, 2017.

[137] W. Stirk et J. Goeders, « Implementation and Design Space Exploration of a Turbo Decoder in High-Level Synthesis », in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico : IEEE, déc. 2019, pp. 1-5.

[138] Y. Delomier, B. L. Gal, J. Crenne et C. Jego, « Model-Based Design of Hardware SC Polar Decoders for FPGAs », *ACM Trans. Reconfigurable Technol. Syst.*, Vol. 13, no. 2, mai 2020.

[139] AMD, *xilinx-demonstrating-industry-first-5g-o-ran-solutions-at-mwc*, 2022. adresse : https://community.amd.com/t5/adaptive-computing/xilinx-demonstrating-industry-first-5g-o-ran-solutions-at-mwc/ba-p/559372.

[140] counterpointresearch.com, *Open RAN Networks – Layer 1 Acceleration Strategy Will Be Key To Operator Success*, 2023. adresse : https://www.counterpointresearch.com/insights/open-ran-networks-layer-1-acceleration-strategy-will-be-key-to-operator-success/.

[141] I. ComSoc, *Intel FlexRAN^{TM} gets boost from AT&T ; faces competition from Marvel, Qualcomm, and EdgeQ for Open RAN silicon*, 2022. adresse : https://techblog.comsoc.org/2022/02/25/intel-flexran-gets-boost-from-at-faces-competition-from-marvel-qualcomm-and-edgeq-for-open-ran-silicon/.

[142] S. Pinneterre, S. Chiotakis, M. Paolino et D. Raho, « vFPGAmanager : A Virtualization Framework for Orchestrated FPGA Accelerator Sharing in 5G Cloud Environments », in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2018, pp. 1-5.

[143] Y. Lin, « Realizing Software Defined Radio – A Study in Designing Mobile Supercomputers », thèse de doct., University of Michigan, 2008.

[144] S.-M. Tseng, Y.-C. Kuo, Y.-C. Ku et Y.-T. Hsu, « Software Viterbi Decoder with SSE4 Parallel Processing Instructions for Software DVB-T Receiver », in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, août 2009, pp. 102-105.

[145] Intel Inc., « mmx-app-viterbi-decoding-140701 », rapp. tech.

[146] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe et T. Yamazaki, « Synergistic Processing in Cell's Multicore Architecture », *IEEE Micro*, Vol. 26, no. 2, pp. 10-24, mars 2006.

[147] G. Falcao, V. Silva, L. Sousa et J. Marinho, « High coded data rate and multi-codeword WiMAX LDPC decoding on Cell/BE », *Electronics Letters*, Vol. 44, pp. 1415-1416, déc. 2008.

[148] J. Zhao, M. Zhao, H. Yang, J. Chen, X. Chen et J. Wang, « High Performance LDPC Decoder on CELL BE for WiMAX System », in *2011 Third International Conference on Communications and Mobile Computing*, Qingdao, China : IEEE, mars 2011, pp. 278-281.

[149] H. Guo, J. Zhao, J. Chen, X. Chen et J. Wang, « High performance turbo decoder on CELL BE for WiMAX system », in *2009 International Conference on Wireless Communications & Signal Processing*, Nanjing, China, nov. 2009, pp. 1-5.

[150] Khronos Group, « OpenCL™ (Open Computing Language), Open Standard for Parallel Programming of Heterogeneous Systems », rapp. tech. adresse : https://www.khronos.org/opencl/.

[151] NVIDIA®, « CUDA® (Compute Unified Device Architecture) », rapp. tech. adresse : https://developer.nvidia.com/cuda-toolkit.

[152] G. Wang, M. Wu, B. Yin et J. R. Cavallaro, « High throughput low latency LDPC decoding on GPU for SDR systems », in *2013 IEEE Global Conference on Signal and Information Processing*, 2013, pp. 1258-1261.

[153] G. Wang, M. Wu, Y. Sun et J. R. Cavallaro, « A massively parallel implementation of QC-LDPC decoder on GPU », in *2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, 2011, pp. 82-85.

[154] G. Falcao, J. Andrade, V. Silva et L. Sousa, « GPU-based DVB-S2 LDPC decoder with high throughput and fast error floor detection », English, *Electronics Letters*, Vol. 47, pp. 542-543(1), 9 avr. 2011. adresse : https://digital-library.theiet.org/content/journals/10.1049/el.2011.0201.

[155] J. Andrade, G. Falcao, V. Silva et L. Sousa, « A Survey on Programmable LDPC Decoders », *IEEE Access*, Vol. 4, pp. 6704-6718, 2016.

[156] B. L. Gal, C. Jego et J. Crenne, « A High Throughput Efficient Approach for Decoding LDPC Codes onto GPU Devices », *IEEE Embedded Systems Letters*, Vol. 6, no. 2, pp. 29-32, juin 2014.

[157] F. Kaltenberger, G. d. Souza, R. Knopp et H. Wang, « The OpenAirInterface 5G New Radio Implementation : Current Status and Roadmap », in *Proceedings of the International ITG Workshop on Smart Antennas (WSA)*, Vienna, Austria, avr. 2019, pp. 1-5.

[158] K. Qiu, S. Zheng, L. Zhang, C. Lou et X. Yang, « DeepSIG : A Hybrid Heterogeneous Deep Learning Framework for Radio Signal Classification », *IEEE Transactions on Wireless Communications*, pp. 1-1, 2023.

[159] G. Falcao, L. Sousa et V. Silva, « Massively LDPC Decoding on Multicore Architectures », *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, no. 2, pp. 309-322, 2011.

[160] C.-C. Chang, Y.-L. Chang, M.-Y. Huang et B. Huang, « Accelerating Regular LDPC Code Decoders on GPUs », *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, Vol. 4, no. 3, pp. 653-659, 2011.

[161] A. Cassagne, T. Tonnellier, C. Leroux, B. Le Gal, O. Aumage et D. Barthou, « Beyond Gbps Turbo decoder on multi-core CPUs », in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2016, pp. 136-140.

[162] B. Le Gal, C. Leroux et C. Jego, « Multi-Gb/s software decoding of Polar Codes », *IEEE Transactions on Signal Processing (TSP)*, Vol. 63, no. 2, pp. 349-359, jan. 2015.

[163] P. Giard, G. Sarkis, C. Leroux, C. Thibeault et W. J. Gross, « Low-Latency Software Polar Decoders », *Journal of Signal Processing Systems*, Vol. 90, no. 5, pp. 761-775, juill. 2016.

[164] B. Le Gal et C. Jego, « High-Throughput FFT-SPA Decoder Implementation for Non-Binary LDPC Codes on x86 Multicore Processors », *Journal of Signal Processing Systems*, Vol. 92, pp. 37-53, mars 2020.

[165] A. Cassagne et al., « AFF3CT : A Fast Forward Error Correction Toolbox ! », *Elsevier SoftwareX*, Vol. 10, pp. 100345, oct. 2019.

[166] A. Cassagne et al., « A Flexible and Portable Real-time DVB-S2 Transceiver using Multicore and SIMD CPUs », in *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1-5.

[167] B. Le Gal et C. Jego, « Low-latency software LDPC decoders », in *Proceedings of the IEEE International Workshop on Signal Processing Systems (SIPS)*, Lorient, France, oct. 2017.

[168] M. Léonardon, A. Cassagne, C. Leroux, C. Jégo, L.-P. Hamelin et Y. Savaria, « Fast and Flexible Software Polar List Decoders », *Journal of Signal Processing Systems*, Vol. 91, no. 8, pp. 937-952, août 2019.

[169] M. Jain, M. Balakrishnan et A. Kumar, « ASIP design methodologies : survey and issues », in *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, jan. 2001, pp. 76-81.

[170] K. Keutzer, S. Malik et A. Newton, « From ASIC to ASIP : the next design discontinuity », in *Proceedings. IEEE International Conference on Computer Design : VLSI in Computers and Processors*, Freiburg, Germany : IEEE, sept. 2002, pp. 84-90.

[171] D. Liu, « ASIP (Application Specific Instruction-set Processors) design », in *2009 IEEE 8th International Conference on ASIC*, Changsha,China : IEEE, oct. 2009, pp. 16-16.

[172] M. Rashid, L. Apvrille et R. Pacalet, « Evaluation of ASIPS design with LISATek », Vol. 5114, juill. 2008, pp. 177-186, isbn : 978-3-540-70549-9.

[173] Synopsys, *ASIP Designer : Design Tool for Application- Specific Instruction-Set Processors*. adresse : https://riscv.or.jp/wp-content/uploads/asip-designer-ds.pdf.

[174] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen et H. Meyr, « A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA », in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, nov. 2001, pp. 625-630.

[175] M. Léonardon, C. Leroux, P. Jääskeläinen, C. Jégo et Y. Savaria, « Transport Triggered Polar Decoders », in *Proceedings of the IEEE International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Vol. 2018, Hong Kong, China, déc. 2018, pp. 1-5.

[176] J. D. Franz, « An evaluation of CoWare Inc.'s Processor Designer tool suite for the design of embedded processors », thèse de doct., Baylor University, 2008.

[177] C. Schmidt-Knorreck et al., « Application-specific instruction-set processor for digital front-end processing in wireless communications », in *ACROPOLIS 2011, 1st Annual Workshop on Advanced coexistence technologies for radio resource usage optimisation, 10 4-5, 2011, Barcelona, Spain*, EURECOM, éd., Barcelona, 2011.

[178] P. Murugappa, A. Baghdadi et M. Jézéquel, « Parameterized area-efficient multi-standard turbo decoder », in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, mars 2013, pp. 109-114.

[179] M. Alles, T. Vogt et N. Wehn, « FlexiChaP : A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding », in *Proceedings of the International Symposium on Turbo Codes and Related Topics (ISTC)*, Lausanne, Switzerland, sept. 2008, pp. 84-89.

[180] N. Paulino, J. C. Ferreira et J. M. P. Cardoso, « Improving Performance and Energy Consumption in Embedded Systems via Binary Acceleration : A Survey », *ACM Comput. Surv.*, Vol. 53, no. 1, fév. 2020, issn : 0360-0300.

[181] P. Plagwitz, F.-J. Streit, A. Becher, S. Wildermann et J. Teich, « Compiler-Based High-Level Synthesis of Application-Specific Processors on FPGAs », in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, déc. 2019, pp. 1-8.

[182] K. Martin, « Génération automatique d'extensions de jeux d'instructions de processeurs », thèse de doct., 2010. adresse : http://www.theses.fr/2010REN1S076.

[183] ALTERA, *AN 188:Custom Instructions for the Nios Embedded Processor*, https://cse.unl.edu/ witty/class/embedded/document/extension.pdf, sept. 2002.

[184] C. Galuzzi et K. Bertels, « The Instruction-Set Extension Problem : A Survey », *ACM Trans. Reconfigurable Technol. Syst.*, Vol. 4, no. 2, mai 2011.

[185] J. G. Tong, I. D. L. Anderson et M. A. S. Khalid, « Soft-Core Processors for Embedded Systems », *2006 International Conference on Microelectronics*, pp. 170-173, 2006.

[186] R. Gonzalez, « Xtensa : a configurable and extensible processor », *IEEE Micro*, Vol. 20, no. 2, pp. 60-70, mars 2000.

[187] *Xtensa® Instruction Set Architecture (ISA) Reference Manual*, 2010. adresse : https://0x04.net/ ~mwk/doc/xtensa.pdf.

[188] A. Ingole et V. Agarwal, « Instruction Set Design for Elementary Set in Tensilica Xtensa », in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kanpur, India : IEEE, juin 2019, pp. 1-5.

[189] E. Cui, T. Li et Q. Wei, « RISC-V Instruction Set Architecture Extensions : A Survey », *IEEE Access*, Vol. 11, pp. 24696-24711, 2023.

[190] C. Heinz, Y. Lavan, J. Hofmann et A. Koch, « A Catalog and In-Hardware Evaluation of Open-Source Drop-In Compatible RISC-V Softcore Processors », déc. 2019, pp. 1-8.

[191] B. Marshall, G. R. . Newell, D. Page, M.-J. O. Saarinen et C. & Wolf, « The design of scalar AES Instruction Set Extensions for RISC-V », in *Transactions on Cryptographic Hardware and Embedded Systems (CHES)*, sept. 2021.

[192] A. De, A. Basu, S. Ghosh et T. Jaeger, « FIXER : Flow Integrity Extensions for Embedded RISC-V », in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy : IEEE, mars 2019, pp. 348-353.

[193] R. Porter, S. Morgan et M. Biglari-Abhari, « Extending a Soft-Core RISC-V Processor to Accelerate CNN Inference », in *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*, déc. 2019, pp. 694-697.

[194] OpenHW Group CORE-V CV32E40P RISC-V IP, *https://github.com/openhwgroup/cv32e40p*, mars 2024.

[195] M. Léonardon, A. Cassagne, C. Leroux, C. Jégo, L.-P. Hamelin et Y. Savaria, « Fast and Flexible Software Polar List Decoders », *Journal of Signal Processing Systems*, Vol. 91, no. 8, pp. 937-952, août 2019.

[196] C. Berrou, A. Glavieux et P. Thitimajshima, « Near Shannon limit error-correcting coding and decoding : Turbo-codes », in *Proceedings of the IEEE International Conference on Communications (ICC)*, Geneva, Switzerland, mai 1993, pp. 1064-1070.

[197] B. Le Gal et C. Jego, « Low-latency and high-throughput software turbo-decoders on multi-core architectures », *Annals of Telecommunications, Springer*, Vol. 75, pp. 27-42, fév. 2020.

[198] PicoRV32 RISC-V Core, *https://github.com/cliffordwolf/picorv32*, mars 2021.

[199] M. Gautschi et al., « Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, no. 10, pp. 2700-2713, fév. 2017.

[200] Ibex RISC-V Core, *https://github.com/lowRISC/ibex*, mars 2021.

[201] D. Schiavone et al., « Slow and steady wins the race ? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications », in *Proceedings of the International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Thessaloniki, Greece, sept. 2017, pp. 1-8.

[202] SCR1 RISC-V Core, *https://github.com/syntacore/scr1*, mars 2021.

[203] RISCY RISC-V Core, *https://github.com/pulp-platform/pulpissimo*, avr. 2021.

[204] A. Cassagne, T. Tonnellier, C. Leroux, B. Le Gal, O. Aumage et D. Barthou, « Beyond Gbps Turbo decoder on multi-core CPUs », in *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Brest, France, sept. 2016, pp. 136-140.

[205] J. Zhao, B. Korpan, A. Gonzalez et K. Asanovic, « SonicBOOM : The 3rd Generation Berkeley Out-of-Order Machine », mai 2020.

[206] F. Zaruba et L. Benini, « The Cost of Application-Class Processing : Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, no. 11, pp. 2629-2640, nov. 2019.

[207] S. Processors, *Shakti Processors*. adresse : https://shakti.org.in/processors.html.

[208] K. Asanović et al., « The Rocket Chip Generator », rapp. tech. no. UCB/EECS-2016-17, avr. 2016.

[209] OpenHWGroup, *CV-X-IF Interface and Coprocessor*, 2024. adresse : https://docs.openhwgroup.org/projects/cva6-user-manual/01_cva6_user/CVX_Interface_Coprocessor.html.

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud

Titre : Enrichissement de l'ISA de cœurs de processeurs pour la couche physique des communications mobiles de 5ème génération

Mot clés : 4G, 5G, ASIP, SIMD, RISC-V, CCE

Résumé : L'évolution des systèmes IoT défini de nouvelles contraintes sur les standards de communications, à l'image de la 5G. Celle-ci a pour objectif d'assurer la compatibilité entre des millions d'objets connectés hétérogènes et utilise les codes correcteurs d'erreurs (CCE) pour assurer la fiabilité et la qualité des informations. Toutefois, les contraintes applicatives liées à de multiples scénarios possibles exigent de ces systèmes d'être modulaires et adaptables. Ces travaux proposent la définition et l'étude approfondie d'instructions spécialisées pour étendre l'architecture de processeurs. Cette approche permet d'exploiter la souplesse de programmation des cœurs matériels afin de résoudre ces questions d'adaptabilité et lever ces verrous scientifiques. Ces jeux d'instructions (ISA) dédiées sont conçues afin de fonctionner sur les processeurs courants (x86, ARM), mais également elles sont prototypées sur cinq cibles de type RISC-V possédant des architectures variées, en 32 et 64 bits. Ces extensions sont étudiées avec des instructions classiques à 2 registres sources, mais l'étude porte également sur des versions à 3 registres sources. Plusieurs modes de parallélisme de données sont proposés (inter/intra trame), pour des décodeurs scalaires et vectoriels paramétrables. Les résultats montrent une réduction en latence avec une augmentation en débit pour des décodeurs de CCE utilisés par la 5G avec les codes LDPC et Polaire, les codes LDPC non-binaire (Satellites) et turbo-codes (4G). Les impacts matériels engendrés par l'implantation de ces instructions sont faibles et maîtrisés relativement au cœur ciblé. L'ensemble de ces résultats ont été mesurés sur FPGA et par simulation.

Title: ISA enrichement of processor cores for the physical layer for the 5th generation mobiles communications

Keywords: 4G, 5G, ASIP, SIMD, RISC-V, ECC, FEC

Abstract: The evolution of IoT systems has defined new constraints on communication standards, such as 5G. The aim of 5G is to ensure compatibility between millions of heterogeneous connected objects and to use error correction codes (ECC) to ensure the reliability and quality of information. However, the application constraints related to multiple possible scenarios require these systems to be modular and adaptable. This work proposes the definition and in-depth study of specialized instructions to extend the architecture of processors. This approach allows to exploit the flexibility of hardware cores programming in order to solve these adaptability issues and overcome these scientific challenges. These dedicated ISA (Instruction Set Architecture) are designed to work on common processors (x86, ARM), but they are also prototyped on five RISC-V targets with varying architectures, in 32 and 64 bits. These extensions are studied with classic 2-source register instructions, but the study also covers versions with 3 source registers. Several data parallelism modes are proposed (inter/in-

tra frame), for scalable and vectorizable parametric decoders. The results show a reduction in latency with an increase in throughput for ECC decoders used by 5G with LDPC and Polar codes, non-binary LDPC codes (Satellites) and turbo-codes (4G). The hardware impacts generated by the implementation of these instructions are low and controlled relatively to the targeted core. All these results have been measured on FPGA and by simulation.