Conception et prototypage de décodeurs de codes correcteurs d'erreurs à partir de modèles comportementaux

Yann Delomier

université de BORDEAUX

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX

École Doctorale n° 209 : Sciences de l'Ingénieur
Spécialité : Électronique

par **Yann Delomier**

Conception et prototypage de décodeurs de codes correcteurs d'erreurs à partir de modèles comportementaux

Directeur de thèse : Christophe Jego
Co-encadrants de thèse : Jérémie Crenne
Bertrand Le Gal

préparée au Laboratoire IMS

soutenue le 10 Juin 2020

Jury :

Dominique Dallet	- Professeur des Universités	- Bordeaux INP	*Président*
Emmanuel Casseau	- Professeur des Universités	- Université de Rennes 1	*Rapporteur*
Laura Conde-Canencia	- Maître de Conférences HdR	- Université de Bretagne Sud	*Rapporteur*
Lilian Bossuet	- Professeur des Universités	- Université Jean Monnet	*Examinateur*
Lionel Lacassagne	- Professeur des Universités	- Sorbonne Université	*Examinateur*
Jérémie Crenne	- Maître de Conférences	- Bordeaux INP	*Co-encadrant*
Bertrand Le Gal	- Maître de Conférences	- Bordeaux INP	*Co-encadrant*
Christophe Jégo	- Professeur des Universités	- Bordeaux INP	*Directeur*

Thèse réalisée au

Laboratoire de l'Intégration du Matériau au Système (IMS)
de Bordeaux, au sein de l'équipe CSN du groupe Conception.

Université de Bordeaux, Laboratoire IMS
UMR 5218 CNRS - Bordeaux INP
351 Cours de la Libération
Bâtiment A31
33405 Talence Cedex
FRANCE

Résumé

Les communications numériques sont omniprésentes dans les objets communicants de la vie courante . L'évolution des standards de communications, la diminution des délais de mise sur le marché et l'hétérogénéité des cadres applicatifs complexifient les défis à relever par les concepteurs de circuits numériques. Les technologies mobiles de cinquième génération (5G) sont une illustration des enjeux actuels. Dans ce contexte, le développement de circuits numériques pour l'implantation de décodeurs de codes correcteurs d'erreurs s'avère particulièrement difficile. La synthèse haut niveau (HLS) est une des méthodologies de conception qui permet le prototypage rapide d'architectures numériques. Cette méthodologie est basée sur l'utilisation de descriptions comportementales pour générer des architectures matérielles. Cependant, le développement de modèles comportementaux efficaces est primordial pour la génération d'architectures performantes. Les travaux présentés dans le cadre de cette thèse ont pour thème la définition de modèles comportementaux efficaces pour la génération d'architectures de décodeurs de codes correcteurs d'erreurs pour les codes LDPC et les codes polaires. Ces deux familles de codes correcteurs d'erreurs sont celles qui ont été adoptées dans le standard 5G. Les modèles comportementaux développés se doivent d'allier flexibilité, rapidité de prototypage et efficacité.

Une première contribution significative des travaux de thèse est la proposition de deux modèles comportementaux permettant la génération d'architectures matérielles efficaces pour le décodage de codes LDPC. Ces modèles sont génériques et associés à une méthodologie flexible. Ils favorisent l'exploration de l'espace des solutions architecturales. Ainsi une multitude de compromis entre le débit, la latence et la complexité matérielle est obtenue. En outre, cette contribution constitue une avancée significative vis-à-vis de l'état de l'art concernant la génération automatique d'architectures de décodeurs LDPC. Enfin les performances atteintes par les architectures synthétisées sont similaires à celles d'architectures conçues manuellement à l'aide d'un flot de conception traditionnel.

Une deuxième contribution des travaux de thèse est la proposition d'un premier modèle

comportemental favorisant la génération d'architectures matérielles de décodeurs de codes polaires à l'aide d'un flot de synthèse de haut niveau. Ce modèle générique permet lui aussi une exploration efficace de l'espace des solutions architecturales. Il est à noter que les performances des décodeurs polaires synthétisés sont similaires à celles des architectures de décodage rapportés dans l'état de l'art.

Une troisième contribution des travaux de thèse concerne le développement d'un modèle comportemental de décodeur de codes polaires implantant un algorithme "à Liste", à savoir l'algorithme de décodage par annulation successive à liste. Cet algorithme de décodage permet d'obtenir de meilleures performances de décodage au prix d'un surcoût calculatoire important. Ce surcoût se répercute sur la complexité matérielle de l'architecture de décodage. Il est à souligner que le modèle comportemental proposé est le premier modèle pour des décodeurs de codes polaires basés sur un algorithme "à Liste".

Mots clefs : Synthèse de Haut Niveau, Modèles, Architectures Numériques, Circuits FPGA, Codes Correcteurs d'Erreurs, Codes LDPC, Codes Polaires, Annulation Successive, Annulation Successive à Liste

Abstract

Digital communications are ubiquitous in the communicating objects of everyday life. Evolving communication standards, shorter time-to-market, and heterogeneous applications make the design for digital circuit more challenging. Fifth generation (5G) mobile technologies are an illustration of the current and future challenges. In this context, the design of digital architectures for the implementation of error-correcting code decoders will often turn out to be especially difficult. High Level Synthesis (HLS) is one of the computer-aided design (CAO) methodologies that facilitates the fast prototyping of digital architectures. This methodology is based on behavioral descriptions to generate hardware architectures. However, the design of efficient behavioral models is essential for the generation of high-performance architectures. The results presented in this thesis focus on the definition of efficient behavioral models for the generation of error-correcting code decoder architectures dedicated tp LDPC codes and polar codes. These two families of error-correcting codes are the ones adopted in the 5G standard. The proposed behavioural models have to combine flexibility, fast prototyping and efficiency.

A first significant contribution of the research thesis is the proposal of two behavioural models that enables the generation of efficient hardware architectures for the decoding of LDPC codes. These models are generic. They are associated with a flexible methodology. They make the space exploration of architectural solutions easier. Thus, a variety of trade-offs between throughput, latency and hardware complexity are obtained. Furthermore, this contribution represents a significant advance in the state of the art regarding the automatic generation of LDPC decoder architectures. Finally, the performances that are achieved by generated architectures are similar to that of handwritten architectures with an usual CAO methodology.

A second contribution of this research thesis is the proposal of a first behavioural model dedicated to the generation of hardware architectures of polar code decoders with a high-level synthesis methodology. This generic model also enables an efficient exploration of the architectural solution space. It should be noted that the performance of synthesized

polar decoders is similar to that of state-of-the-art polar decoding architectures.

A third contribution of the research thesis concerns the definition of a polar decoder behavioural model that is based on a "list" algorithm, known as successive cancellation list decoding algorithm. This decoding algorithm enables to achieve higher decoding performance at the cost of a significant computational overhead. This additional cost can also be observed on the hardware complexity of the resulting decoding architecture. It should be emphasized that the proposed behavioural model is the first model for polar code decoders based on a "list" algorithm.

Key words : High Level Synthesis, Models, Digital Architectures, FPGA circuits, Error Correction Codes, LDPC Codes, Polar Codes, Successive Cancellation, List Successive Cancellation.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

$5G$	5th generation of cellular mobile communications
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
$ALUT$	Adaptive Look-Up Table
API	Application Program Interface
$ASIC$	Application Specific Integrated Circuit
$ASIP$	Application Specific Instruction set Processor
$AVX2$	Advanced Vector Extensions 2
$AWGN$	Additive White Gaussian Noise
$BCJR$	Algorithme de décodage MAP nommé d'après ses inventeurs : Bahl, Cocke, Jelinek et Raviv
BER	Bit Error Rate
$BI-AWGN$	Binary Input - AWGN
$BPSK$	Binary Phase-Shift Keying
$BRAM$	Block Random Access Memory
$CA-SCL$	CRC-Aided Successive Cancellation List
CN	Check Node
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
$CUDA$	Compute Unified Device Architecture
DSP	Digital Signal Processor
ECC	Error Correction Codes
FB	Frozen Bit
FER	Frame Error Rate
FF	Flip-Flop
$FIFO$	First In, First Out
$FPGA$	Field-Programmable Gate Array
FSM	Finite State Machine
GPP	General Purpose Processor

GPU	Graphics Processing Unit
HDL	Hardware Description Language
HLS	High Level Synthesis
IoT	Internet of Things
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
LUT	Look-Up Table
MAP	Maximum A Posteriori
PAR	Place and route
PE	Processing Element
PU	Processing Unit
QC	Quasi-Cyclic
QC – LDPC	Quasi-Cyclic LDPC
RAM	Random Access Memory
RAW	Read After Write
REP	REPetition
RTL	Register Transfer Level
SDR	Software Defined Radio
SC	Successive Cancellation
SCL	Successive Cancellation List
SIMD	Single Instruction Multiple Data
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SPA	Sum-Product Algorithm
SPC	Single Parity Check
SSC	Simplified Successive Cancellation
TPMP	Two-Phase Message Passing
TTM	Time To Market
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VN	Variable Node

Introduction

Les travaux développés dans le cadre de cette thèse ont pour thème la génération d'architectures matérielles sous contraintes à l'aide d'outils de synthèse de haut niveau. Le domaine applicatif retenu dans le cadre de ces travaux est celui des communications numériques et plus particulièrement celui des codes correcteurs d'erreurs. Les contributions concernent le développement de modèles architecturaux comportementaux dédiés à la génération d'architectures matérielles efficaces pour le décodage de codes correcteurs d'erreurs. Dans cette partie introductive, la motivation de ces travaux est exposée tout comme la méthodologie employée. Dans un second temps, la structure du manuscrit est détaillée et les différentes contributions liées à ces travaux de thèse sont résumés.

Motivations des travaux de thèse

L'évolution des architectures des systèmes embarqués associée à l'augmentation de la complexité des applications complique la tâche des concepteurs de circuits et de systèmes numériques. Or, les délais de conception toujours plus courts obligent les concepteurs à employer des méthodologies de conception automatisée, moins chronophages au prix d'une diminution des performances. Cet état de fait atteint son paroxysme dans le domaine des communications numériques et plus précisément pour le futur marché lié à la technologie 5G. C'est dans ce contexte que s'inscrivent les travaux de cette thèse.

Les codes correcteurs d'erreurs et plus particulièrement les décodeurs associés sont le plus souvent les éléments les plus complexes d'une chaîne de communications numériques. En conséquence, ces travaux de thèse s'intéressent à la conception d'architectures matérielles de décodeurs de codes correcteurs d'erreurs pour la téléphonie mobile de $5^{\text{ème}}$ génération. L'approche employée dans le cadre de ces travaux se base sur l'utilisation d'outils de synthèse de haut niveau (HLS) afin de réduire le temps de conception et ainsi augmenter l'exploration de l'espace des solutions architecturales. En effet la récente évolution des outils HLS permet d'envisager la conception d'architectures matérielles efficaces dans un

délai significativement inférieur par rapport aux approches de conception traditionnelles. Des travaux récents ont permis de démontrer la viabilité de cette approche pour la conception de décodeurs de codes LDPC [1]. Cependant le niveau de performances des architectures obtenues est nettement en deçà de l'état de l'art. Dans le cadre de cette thèse, nous proposons une approche différente, basée sur le développement de modèles comportementaux intégrant des aspects architecturaux. Ces modèles spécialement conçus pour la génération d'architectures matérielles via un flot de conception HLS fournissent des niveaux de performance proches des architectures développées manuellement au niveau RTL (Register Transfer Level). Les travaux présentés dans ce manuscrit visent à concevoir des décodeurs matériels permettant le décodage de codes LDPC et de codes polaires afin d'éprouver la méthodologie employée. Ces deux types de codes correcteurs d'erreurs ont été retenus car ils ont été sélectionnés dans le standard 5G.

Structure du manuscrit

Le chapitre 1 présente le contexte des travaux de thèse puis développe les problématiques associées. Tout d'abord, une introduction détaillant l'évolution des systèmes électroniques et leurs complexités explique les enjeux auxquels font face les concepteurs de tels systèmes. Différentes méthodologies de conception de circuits numériques sont présentées et comparées. Dans un deuxième temps, le domaine d'applicatif dans lequel cette thèse s'inscrit ainsi que la méthodologie de conception proposée sont présentés. Les systèmes de communications numériques sont introduites, et plus particulièrement les enjeux liés à l'implémentation d'algorithmes de décodage de codes correcteurs d'erreurs. Cela amène à la principale motivation de ces travaux de thèse qui consiste à développer une approche basée sur des modèles comportementaux. Cette approche exploite la capacité actuelle des outils de synthèse de haut niveau (HLS) pour réaliser l'implantation de décodeurs matériels dédiés aux codes correcteurs d'erreurs.

Dans le chapitre 2, la méthodologie de conception retenue est appliquée à la conception de décodeurs de codes LDPC. Deux modèles comportementaux génériques sont proposés et analysés. Différents paramètres algorithmiques permettent de configurer les décodeurs afin de répondre aux différents contextes applicatifs. Le premier modèle conçu se base sur une tentative de réutilisation des travaux menés dans le cadre de l'implantation de décodeurs LDPC sur des processeurs et des GPU. Suite à l'analyse des performances de ce premier modèle et l'identification de ses limitations, un second modèle palliant les inconvénients et les limitations du premier modèle a été développé. Les modèles proposés permettent une exploration efficace de l'espace de conception pour les décodeurs LDPC. Une comparaison avec les travaux issus de la littérature démontre que notre second modèle comportemental

permet d'atteindre des performances similaires aux décodeurs matériels décrits au niveau RTL.

Le décodage de codes LDPC repose sur des algorithmes itératifs possédant un parallélisme calculatoire élevé. Les résultats donnés tout au long du deuxième chapitre montrent que la méthodologie basée sur l'utilisation d'outils HLS est pertinente pour ce type d'algorithme de décodage. Afin d'évaluer cette méthodologie basée sur l'utilisation de modèles comportementaux, nous testons l'approche proposée sur une autre famille de codes correcteurs d'erreurs : les codes polaires. Ainsi, le chapitre 3 détaille la conception de modèles comportementaux pour le décodage de codes polaires basé sur un algorithme à annulation successive (SC). Cet algorithme de décodage se distingue des algorithmes de propagation de croyance utilisés pour le décodage des codes LDPC par un parallélisme de calcul irrégulier et un comportement relativement séquentiel. Dans ce chapitre, un modèle générique permettant d'implanter différentes optimisations algorithmiques et architecturales pour l'algorithme SC est détaillé. La grande diversité des optimisations existantes pour l'algorithme SC implique qu'un grand nombre d'éléments doit être paramétrables. Ainsi la méthodologie basée sur l'utilisation de modèles comportementaux permet une vaste exploration de l'espace de conception architectural. Cela aboutit à la génération de multiples compromis entre complexité et performances. La qualité des architectures générées est comparée aux architectures de la littérature décrites au niveau RTL. Les limitations de la méthodologie sont enfin abordées et explicitées.

La pertinence de la méthodologie basée sur l'utilisation de modèles comportementaux a été démontrée dans les chapitres 2 et 3 pour des algorithmes de décodages différents du point de vue de leur structure algorithmique. Il existe cependant une variante de l'algorithme de décodage SC qui implique des défis architecturaux supplémentaires : l'algorithme de décodage SC à liste (SCL). Dans le chapitre 4 un modèle de décodeur de codes polaires utilisant l'algorithme SCL est détaillé. En effet la performance des décodeurs utilisant l'algorithme SCL dépend fortement de l'efficacité de certaines opérations réalisant des traitements complexes tels que des opérations de tri. Il est donc pertinent d'évaluer la capacité de l'approche à générer des éléments de calcul complexes et efficaces du point de vue de la latence. Un modèle générique est proposé, pouvant là aussi permettre une large exploration de l'espace de conception architecturale. Les architectures générées sont évaluées et montrent la pertinence de l'approche retenue.

Contributions des travaux de thèse

Les différentes contributions de ces travaux de thèse se résument comme suit :

1. La proposition de modèles comportementaux génériques permettant la génération d'architectures matérielles de décodeurs de codes LDPC à travers l'utilisation d'un outil de synthèse de haut niveau (HLS). Ces modèles sont configurables grâce à des paramètres architecturaux et algorithmiques qui autorisent une vaste exploration de l'espace de conception. Les modèles proposés offrent une nette amélioration par rapport à l'état de l'art pour la génération d'architectures de décodeurs LDPC à l'aide d'outils de synthèse de haut niveau. Les architectures matérielles produites ont des niveaux de performances proches des architectures dédiées de la littérature. Cette contribution est développée dans le chapitre 2.
2. La proposition d'architectures fiables de canal additif à bruit blanc gaussien (AWGN) à l'aide d'outils de synthèse de haut niveau. Cette contribution est décrite dans le chapitre 2. Elle est utile pour le prototypage de chaînes de communications numériques sur circuit FPGA.
3. La proposition d'un modèle comportemental générique permettant la génération d'architectures matérielles de décodeurs SC pour les codes polaires. Ce modèle est paramétrable et les architectures générées intègrent des optimisations dynamiques à l'exécution. Cette approche constitue le premier modèle comportemental de décodeur SC de la littérature. Les résultats d'implémentations des architectures matérielles générées sont comparables à celles d'architectures matérielles dédiées.
4. La proposition d'un modèle comportemental générique permettant la génération d'architectures matérielles de décodeurs de type SCL pour les codes polaires. Ce modèle hautement paramétrable permet une exploration de l'espace de conception. De plus une méthode de tri adaptée à l'algorithme de décodage SCL est présentée.

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

— Conférence nationale avec comité de lecture :

- Y.Delomier, B. Le Gal, J. Crenne et C. Jégo, "Génération d'architectures de décodeur LDPC à l'aide d'un outil de synthèse de haut niveau", Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS), 2018

— Conférences internationales avec comité de lecture :

- Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "Fast Design of Reliable, Flexible and High-Speed AWGN architectures with High Level Synthesis" in IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018

  - Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "From multicore LDPC decoder implementations to FPGA decoder architectures : a case study", in IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018
  - Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "Model-based Design of Efficient LDPC Decoder Architectures", IEEE International Symposium on Turbo Codes & Iterative Information Processing (ISTC), 2018
  - Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "Generation of Efficient Self-adaptive Hardware Polar Decoders using High-Level Synthesis", in IEEE International Workshop on Signal Processing Systems (SiPS), 2019

— Revue internationale avec comité de lecture :

  - Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "Model-based Design of Flexible and Efficient LDPC Decoders on FPGA Devices", in Springer, the Journal of Signal Processing Systems (JSPS), January 2020, acceptée.

— Revues internationales avec comité de lecture (soumis) :

  - Y.Delomier, B. Le Gal, J. Crenne and C. Jégo, "Model-based design of hardware SC polar decoders for FPGAs", in ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2020, révision majeure.
  - B. Le Gal, Y.Delomier, C. Leroux and C. Jégo, "Low-latency Sorter Architecture for Polar Codes Successive-Cancellation-List Decoding", in IEEE Transactions on Circuits and Systems II : Express Briefs (TCAS-II), 2020, soumission.

1 Contexte et problématiques

Le but de ce chapitre 1 est de sensibiliser le lecteur aux problématiques auxquelles font face les concepteurs de circuits numériques. Plus particulièrement les défis que représente la conception de décodeurs de codes correcteurs d'erreurs dans les chaînes de communications numériques.

Dans la première section les différentes méthodologies de conception de circuits numériques sont présentées aux travers d'exemples ludiques. La deuxième section présente les chaînes de communications numériques ainsi que les codes correcteurs d'erreurs. L'approche de conception d'architectures numériques utilisée dans cette thèse est détaillé en fin de chapitre.

1.1 L'implémentation des systèmes électroniques

1.1.1 L'évolution de la complexité des systèmes

Les systèmes embarqués, et plus particulièrement les systèmes embarqués communicants sont omniprésents dans notre vie depuis maintenant deux décennies. Ces derniers sont en effet présents dans les objets du quotidien i.e. les téléphones, les consoles de jeu, les box des fournisseurs d'accès à internet, les systèmes automobiles, etc. Les systèmes électroniques embarqués ne cessent d'évoluer tant par l'augmentation de la complexité des fonctions qu'ils implémentent que par la diversification des domaines applicatifs.

Un exemple assez emblématique de l'évolution de ces systèmes embarqués est la téléphonie mobile. Le premier téléphone mobile commercialisé a été conçu par Motorola et mis sur le marché 1976 [2]. Ce téléphone ne réalisait que des fonctions d'émission et de réception d'appels téléphoniques. Ce téléphone de première génération ("1G") n'était qu'un système de radio bidirectionnelle, et était par conséquent principalement analogique. A cette époque le marché des téléphones mobiles était peu concurrentiel et le nombre d'utilisateurs était très limité (environ 143000 utilisateurs au niveau mondial).

Environ 15 ans plus tard, en 1991, la téléphonie mobile de deuxième génération (2G) [3] fut déployée sur le marché. De nouveaux téléphones sont alors proposés. En plus des fonctions réalisées par leurs prédécesseurs, ces derniers intègrent de nouvelles fonctionnalités comme l'envoi de messages textuels. Ils se composent dorénavant d'un système de communications numériques. Les systèmes électroniques qui composent ces téléphones de nouvelle génération mélangent technologies analogiques et numériques. Le marché en très forte croissance est dopé par l'adhésion du grand publique. Il dépasse les 11 millions d'utilisateurs dans le monde. La baisse du prix des forfaits téléphonique liée à l'arrivée de nouveaux opérateurs permet de pérenniser le marché, voir à pousser les constructeurs de téléphones à innover rapidement afin de se démarquer de la concurrence.

Le lancement de la téléphonie mobile de $3^{\text{ème}}$ génération (3G) [4] a lieu 10 ans plus tard au début des années 2000. Cette troisième génération de téléphones mobiles révolutionne les usages en offrant l'accès à internet, aux appels vidéo, au transfert de données sans fil, etc... L'évolution des techniques de codage du standard 3G associée à une évolution des infrastructures de communication ont permis l'adoption par le grand publique de ces technologies. En parallèle de l'évolution des réseaux, les systèmes embarqués au sein des téléphones mobiles se sont fortement complexifiés afin de répondre à ces nouveaux besoins. Les parties numériques en charge du traitement de l'information et de l'exécution des tâches sont devenues prédominantes. Ainsi le marché en pleine expansion compte

jusqu'à 700 millions d'utilisateurs. La forte concurrence entre les fabricants de téléphones implique des cycles d'innovation et de développement toujours plus courts.

Poussée par le marché, la 4ème génération de téléphonie mobile (4G) [5] voit le jour au début des années 2010. Dorénavant le réseau téléphonique permet des échanges de données à plus de 100 Mbps grâce à des techniques de communication sans fils plus évoluées. Les nouveaux téléphones compatibles 4G embarquent un système d'exploitation et peuvent réaliser de nombreuses fonctionnalités. Les systèmes électroniques embarqués dans les téléphones sont très hétérogènes. Ils combinent un processeur généraliste (GPP), un processeur graphique (GPU), des accélérateurs dédiés, des composant analogiques... On considère qu'en 2011, le marché a atteint 6 milliards d'utilisateurs.

En quelques décennies, l'architecture des téléphones portables est passée de systèmes analogiques capables uniquement de traiter la voix à un système multimédia intégrant diverses fonctionnalités complexes. En plus des différents composants analogiques, capteurs et périphériques, les téléphones embarquent aujourd'hui un ou plusieurs systèmes sur puce (SoC) complexes. Ces SoCs sont composés d'au moins un processeur généraliste, un processeur graphique, de divers contrôleurs de périphérique ainsi qu'une multitude d'accélérateurs matériels dédiés.

Le marché concurrentiel et l'évolution des technologies imposent des délais de mise sur le marché de plus en plus courts. Il s'agit d'une contrainte majeure pour les concepteurs matériels et les développeurs logiciels. C'est pourquoi, les méthodologies de conception ont du et doivent encore évoluer afin de permettre l'intégration d'applications de plus en plus complexes dans un temps toujours plus restreint.

1.1.2 Les cibles technologiques

Actuellement, plusieurs solutions technologiques permettent au concepteur d'implémenter une fonction numérique. Ces solutions d'implémentation se différencient tant au niveau du matériel adressé que par les méthodologies de conception associées. Bien entendu chaque solution a des avantages, des limitations et des inconvénients.

Dans le domaine des serveurs de calcul, les différentes solutions technologiques (ASIC, FPGA, GPU) sont bien représentées [6, 7, 8] et les choix d'implémentation des différentes fonctions ont variés ces dernières décennies. En effet à l'origine les serveurs de calculs exploitaient pour des questions de performance des circuits intégrés dédiés (ASIC) [6]. Ces circuits offrent de hauts niveaux de performances calculatoires pour un coût énergétique relativement faible. En revanche, le temps de conception relativement important des

circuits dédiés, associé à une flexibilité quasi nulle représentent actuellement un frein à leur utilisation.

Les progrès réalisés dans le domaine des architectures numériques programmables a changé la donne au début des années 2000 [7]. En effet les architectures multi/many coeurs tel que les processeur généraliste (GPP) ou les processeurs graphiques (GPU) ont vu leurs capacités et leurs efficacités calculatoires grandement augmenter. De nos jours, les architectures programmables sont capables d'atteindre des puissances de calculs proches de celles des ASIC pour certains domaines applicatifs. De plus, leur facilité de programmation en font des solutions appréciées des concepteurs. Toutefois, leur efficacité énergétique est préjudiciable dans bon nombre de systèmes embarqués qui sont, par définition, contraints au niveau de la consommation d'énergie et la de dissipation thermique.

En parallèle des évolutions dans le domaine des architectures programmable, des progrès substantiels ont été réalisés au niveau des circuits logiques programmables (FPGA). Ces derniers dont les premières réalisations datent du milieu des années 80 font actuellement leurs apparitions dans de nombreux domaines applicatifs tels que dans les serveurs de calcul [8] ainsi que dans bon nombre de systèmes embarqués. Les FPGAs sont plus efficaces énergétiquement que les GPUs et plus flexibles que les ASICs. De plus, les performances calculatoires qu'ils permettent d'atteindre sont très attrayantes. Le rachat récent d'Altera par Intel démontre que l'utilisation conjointe de GPP et de circuits FPGA est une évolution majeure pour les futurs systèmes électroniques. Ces architectures conjointes se généralisent dans bon nombre de domaines embarqués mais aussi dans le domaine des serveurs de calculs [8].

1.1.3 Les méthodologies de conception

Concevoir un système embarqué exécutant une ou plusieurs fonctionnalités est une tâche complexe et chronophage. Pour implanter une unique fonctionnalité, il existe de multiples solutions présentant toutes des compromis différents entre efficacité, temps de développement et flexibilité. Le choix de la technologie d'intégration (ASIC, FPGA, GPP ou GPU) et de la méthodologie de conception associée est souvent dicté par les contraintes applicatives. Dans cette section, les principales méthodologies de conception et leurs architectures cibles sont présentées puis comparées.

Les architectures dédiées

La conception d'architectures dédiées est la solution permettant d'obtenir les implémentations les plus efficaces au niveau du coût matériel, des performances et de la consommation

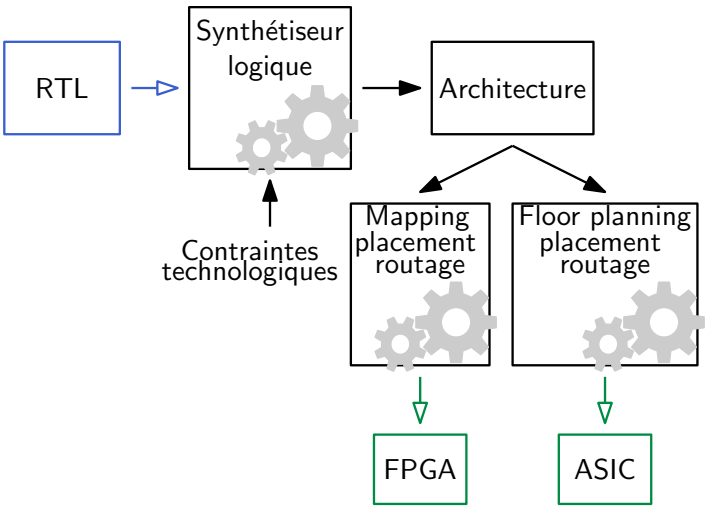

FIGURE 1.1 – Méthodologie de conception pour les architectures dédiées

d'énergie. Les architectures dédiées peuvent être intégrées dans des circuits **FPGA** ou implantées sous forme d'**ASIC**. La méthodologie de conception pour ces deux types de solutions est similaire. Cette dernière est illustrée par la Figure 1.1

A partir des spécifications de la fonction à concevoir, le concepteur imagine une architecture numérique. Afin d'implémenter cette architecture sur FPGA ou ASIC, le concepteur doit la décrire au niveau **RTL**. Cette architecture est en général décrite à l'aide d'un langage de description matérielle (HDL) tels que le langage **VHDL** ou le langage **Verilog**. Ensuite, la description RTL de l'architecture est synthétisée par un outil de **synthèse logique** tels que Quartus [9], Vivado [10] ou Design Compiler [11]. L'outil de synthèse logique fournit une description sous la forme de **Netlist** qui prend en compte des contraintes provenant de la cible FPGA ou de la technologie ASIC. Puis, une étape de **placement et de routage** qui diffère en fonction de la cible technologique est nécessaire. A l'issue de l'étape de placement routage, des fichiers **bistream** ou GPII sont générés, respectivement pour les circuits FPGA ou ASIC.

Cette méthodologie permet de concevoir des circuits performants au niveau de la latence et de la fréquence de fonctionnement. Toutefois, elle nécessite des cycles de développement conséquents. En effet, produire une description architecturale au niveau RTL est chronophage. De plus, les phases itératives de vérification et mise au point sont très longues. Cela est préjudiciable pour les applications qui nécessitent une mise sur le marché rapide.

Parmi les cibles technologiques adressables, les ASICs fournissent la meilleure solution d'intégration au niveau de l'efficacité énergétique, de la fréquence de fonctionnement et

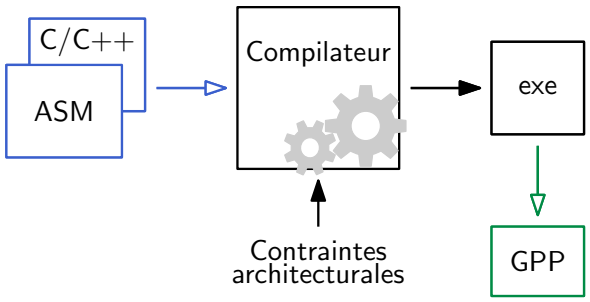

Figure 1.2 – Méthodologie de conception pour processeur à usage général

de la surface résultante. Cependant, ils nécessitent un temps de conception élevé et un investissement initial conséquent. Ils sont principalement utilisés pour la production en grande série de circuits intégrés.

Les circuits FPGA, conçus autour d'une matrice logique reprogrammable sont une solution alternative. Contrairement aux circuits ASIC qui sont figés dans le silicium, les circuits FPGA peuvent être configurés ou reconfigurés. Des techniques telles que la reconfiguration dynamique partielle [12] permettent de rajouter un niveau de flexibilté supplémentaire lors de l'exécution. Cependant, les circuits FPGA sont plus onéreux et moins efficaces au niveau consommation énergétique.

Les processeurs à usage général

Une approche orthogonale à la précédente consiste à utiliser des architectures déjà conçues, telles que des architectures de processeurs généralistes (**GPP**). Ces processeurs généralistes à jeu d'instructions sont souvent utilisés en premier lieu en raison de leurs simplicités de mise en oeuvre. Afin de déployer une fonction sur un processeur, il convient de le programmer à l'aide d'un langage de programmation. Lors du développement de systèmes embarqués, les langages les plus souvent employés pour la description d'une fonction sont le langage assembleur (**ASM**) ou le langage **C/C++**. La description algorithmique de la fonction à intégrer est transformée par le **compilateur** en un programme binaire tel que cela est présenté dans la Figure 1.2.

Le processus de compilation est contraint par les instructions disponibles dans le processeur. Le programme binaire résultant, une fois chargé dans la mémoire de l'architecture cible, permet d'exécuter la fonction décrite. Les architectures programmables de type GPP sont caractérisées par une grande flexibilité au prix de performances applicatives en retrait face à des implémentations sur des circuits dédiés. En effet, le jeu d'instructions

généraliste est rarement optimisé par rapport au domaine applicatif. Ainsi l'exécution de calculs arithmétiques simples peuvent nécessiter de nombreux cycles d'horloge, c'est le cas typiquement des calculs en virgule fixe saturés.

Selon le niveau d'abstraction et le langage de programmation retenu, le concepteur va minimiser le temps de développement et/ou améliorer les performances applicatives. Le faible temps nécessaire à la compilation des programmes couplé à des outils de mise au point performants permet de réduire le temps de conception. L'utilisation de langages de haut niveau, comme par exemple le langage C/C++, offre en parallèle une flexibilité et une évolutivité plus importante.

Malgré les récentes évolutions des GPP et en particulier de leurs ALU (qui sont devenues des architectures vectorielles), les GPP sont peu efficaces au niveau de la consommation d'énergie et de la puissance de calcul pour un large spectre d'applications. Le niveau de parallélisme offert n'est pas en adéquation avec les besoins de bon nombre d'applicatifs scientifiques et industriels.

Les architectures multi/many coeurs

L'augmentation du nombre de coeurs de calcul au sein des GPP est apparue peu après le début des années 2000 comme une solution permettant d'accroître la puissance de calcul des architectures programmables. En parallèle, des architectures massivement parallèles dédiés aux traitements graphiques (GPU) sont apparues.

La seconde approche repose sur un autre type d'architecture programmable que sont les processeurs graphiques (**GPU**). Les GPUs se différencient des GPPs par (1) une dissociation des coeurs de calcul et de la partie contrôle, (2) une simplification des coeurs de calcul et (3) un grand nombre de coeurs de calcul pouvant opérer en parallèle. Un GPU intègre de quelques centaines à quelques milliers de coeurs de calcul [13, 14]. Le grand nombre de coeurs de calcul présents fournit un avantage conséquent pour les applications nécessitant un haut niveaux de parallélisme.

Cependant afin d'exploiter efficacement ces systèmes composés de plusieurs coeurs de processeur, il est nécessaire de faire évoluer les descriptions algorithmiques des fonctions à intégrer. Pour ce faire des langages de programmation dédiés ou des extensions (OpenCL, Click, OpenMP, CUDA) ont été développés. Ces derniers facilitent le travail des concepteurs et permettent de tirer pleinement partie des architectures. Toutefois, l'adaptation d'une description logicielle séquentielle pour une implantation sur une architecture multi/many-coeurs peut être chronophage. Elle impacte nécessairement la flexibilité et

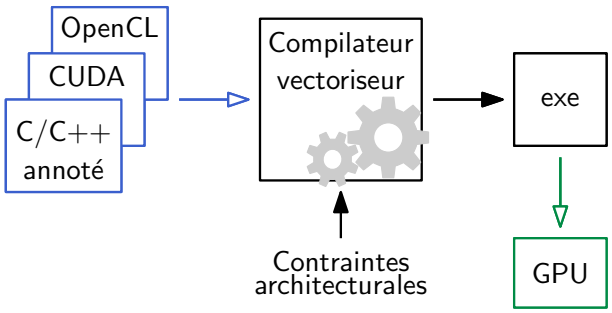

Figure 1.3 – Méthodologie de conception pour une architecture multi/many coeurs

l'évolutivité de l'implémentation logicielle.

Comme illustré dans la Figure 1.3, le développeur modélise le comportement de la fonction à implanter à l'aide d'un langage dédié de type **OpenCL** ou **CUDA**. La description est ensuite compilée pour générer un exécutable qui sera intégré sur l'architecture cible. La spécificité de ces langages de programmation est de permettre une description aisée des algorithmes composés de boucles imbriquées réalisant des calculs sur des données indépendantes. Ces langages de programmation permettent une expression fine des calculs qui peuvent être parallélisés à l'exécution et donc par conséquent tirer parti du nombre de coeurs de calcul disponibles pour accélérer le traitement. Le développement sur de telles architectures est plus complexe et plus chronophage que le développement sur une architecture GPP. Cependant l'implémentation sur architecture multi/many-coeurs permet de profiter d'une plus grande puissance de calcul et d'un meilleur compromis consommation énergétique / performance calculatoire.

En revanche cette approche est, comme pour les GPP : (1) contrainte par le jeu d'instructions des architectures cibles qui n'est pas toujours en phase avec les besoins applicatifs et (2) moins efficace au niveau énergétique par rapport aux circuits dédiés.

Les processeurs à jeu d'instructions spécifiques (ASIP)

La conception d'architectures dédiées est complexe et chronophage tandis que l'utilisation d'architectures programmables est peu efficace. Une solution intermédiaire consiste à développer un processeur à jeu d'instructions spécifiques (**ASIP**) et à l'implanter sur cible ASIC/FPGA. Les processeurs à jeu d'instructions spécialisés sont des processeurs qui intègrent des unités de calcul dédiées à un domaine applicatif. Ces unités de calcul sont utilisables par le concepteur lors du développement logiciel par l'intermédiaire d'instructions spécialisées.

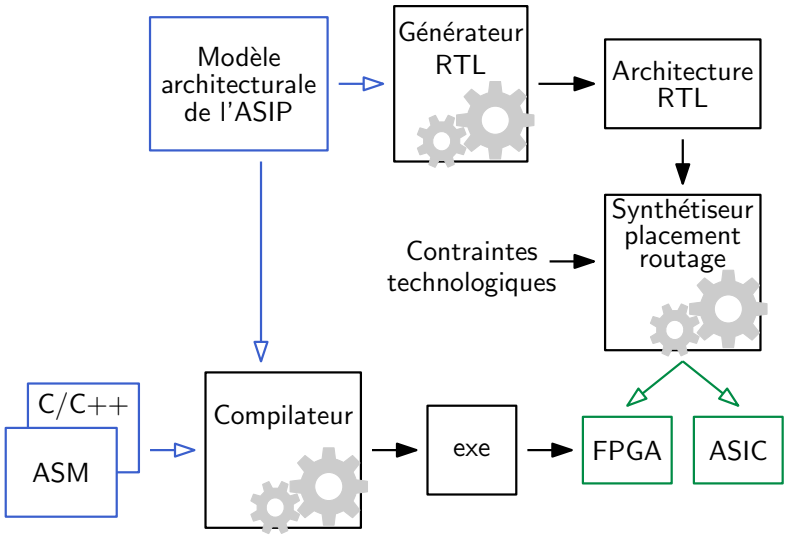

Figure 1.4 – Méthodologie de conception pour ASIP

Développer une solution à base d'ASIP nécessite 2 phases de développement interdépendantes comme cela est explicité dans la Figure 1.4. La première phase consiste à définir et à décrire l'architecture du processeur. Cette phase de conception peut être réalisée manuellement ou à l'aide d'outils spécialisés [15, 16]. A partir de ce **modèle architectural** des outils industriels ou académiques génèrent l'architecture de l'ASIP au niveau RTL. La seconde phase consiste à écrire un programme logiciel pour ce coeur de processeur. Des annotations particulières dans le programme ou bien l'utilisation d'instructions spécifiques permettent d'exploiter des ALU dédiées qui ont été instanciées dans le coeur de l'ASIP. Une fois ces étapes effectuées, il est possible à l'aide d'un synthétiseur logique et d'un compilateur logiciel d'implanter l'architecture matérielle et son programme sur une cible ASIC et/ou FPGA.

Cette méthodologie offre un niveau de flexibilité comparable aux architectures programmables. Cependant, elle souffre d'un temps de développement plus long en raison de la nécessité de décrire tout ou partie du coeur de processeur. La description logicielle doit être adaptée pour bénéficier au mieux des capacités de l'architecture. Par rapport aux architectures dédiées, les ASIP correspondent cependant à une solution plus générique.

La synthèse de haut niveau

L'utilisation d'ASIP offre de la flexibilité cependant cette solution architectural ne permet pas d'atteindre les performances des architectures dédiés. Les outils de synthèse d'architecture (**HLS**) représentent une alternative intéressante offrant flexibilité, temps

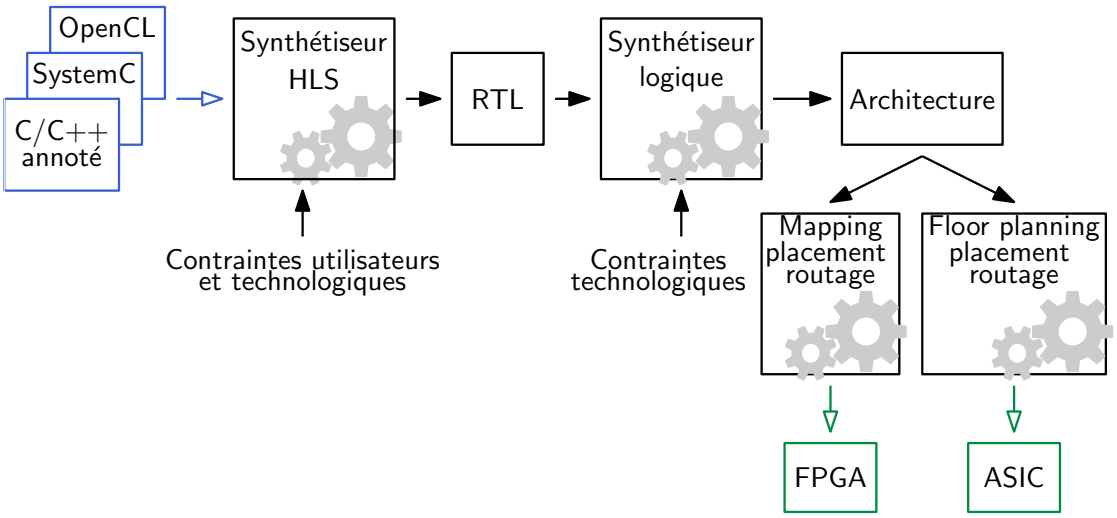

Figure 1.5 – Méthodologie de conception intégrant la synthèse de haut niveau

de conception réduit et performances.

Les méthodologies de conception basées sur la HLS automatisent la génération d'architectures de niveau RTL à partir d'une description comportementale de l'application. Une description comportementale de la fonctionnalité à intégrer est décrite à l'aide de langages de haut niveaux tels que **C/C++**, **OpenCL** ou **SystemC**. Les outils de synthèse HLS sont le pendant matériel des compilateurs logiciels. La différence provient du fait que l'architecture cible n'est pas figée mais construite. D'un point de vue méthodologique, la description comportementale est analysée par l'outil de synthèse HLS. Ce dernier va instancier des ressources matérielles à partir d'une modélisation interne du comportement et des calculs. Ensuite il va ordonnancer les calculs dans le temps et les assigner aux ressources matérielles. Enfin l'outil va générer une description de niveau RTL de l'architecture générée. La description de niveau RTL peut ensuite être traitée dans un processus de synthèse logique classique ciblant soit un ASIC soit un FPGA, comme explicité dans la Figure 1.5

Les descriptions algorithmiques fournies en entrée de l'outil HLS sont généralement annotées à l'aide de **directives**. Ces directives permettent de guider le processus de synthèse HLS lors de la construction de l'architecture. Elles permettent par exemple de spécifier un déroulage partiel ou total des boucle itératives *for*, de spécifier finement les protocoles des signaux d'entrées/sorties ou de choisir une implantation *pipeline* du chemin de données.

L'utilisation d'un langage de haut niveau associée à des annotations permet de décrire au

niveau algorithmique le comportement de l'architecture RTL à générer ainsi que certaines propriétés architecturales. Cette méthodologie favorise la conception d'architectures dédiées de manière similaire à celles employées pour développer des implémentation logicielles. En comparaison avec la conception d'architectures dédiées au niveau RTL, cette approche est moins chronophage car une partie du développement RTL est assumé par l'outil. Cependant, la pertinence des architectures RTL générées par un outil de synthèse HLS est fortement corrélée avec la qualité de la description comportementale produite par le concepteur.

Depuis quelques années, des travaux proposent la réutilisation de descriptions comportementales écrites en première intention pour des implémentations sur des GPPs ou des GPUs à travers un flot de conception HLS. Malheureusement, cela ne permet pas d'aboutir à la synthèse d'architectures efficaces [1]. En effet pour obtenir des architectures efficaces, il est nécessaire de prendre en considération à la fois le fonctionnement de l'outil de synthèse HLS et les caractéristiques des technologies ciblées (ASIC, FPGA).

Synthèse

Comme nous venons de le voir, il existe différentes solutions architecturales pour implanter une seule et même fonction numérique prenant en compte des contraintes applicatives. Ces solutions se répartissent en deux catégories :

— l'utilisation d'architectures programmables,

— la conception d'architectures dédiées.

L'ensemble des méthodologies permet de concevoir des systèmes électroniques embarqués à partir de différentes cibles technologiques que sont les GPPs, les GPUs, les ASIP, les FPGAs et les ASICs. Ces solutions architecturales offrent différent compromis aux

Tableau 1.1 – Caractérisation des différentes méthodologies de conception de systèmes embarqués

Cible	Performances	Latence	Consommation énergétique	Flexibilité	Coût	Effort de mise en oeuvre
GPP	+	+	++	++	−	−
GPU	++	++	+	+	+	+
ASIP	++	+	+	+	+	+
FPGA	+ + +	−	+	−	+	++
ASIC	+ + ++	−−	−	ø	++	+ + +

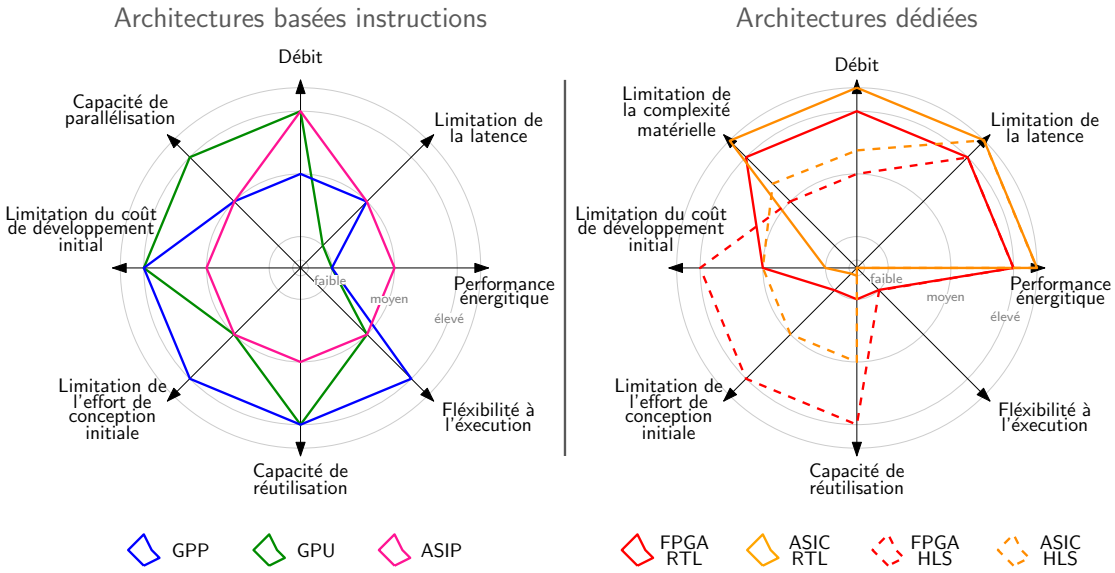

Figure 1.6 – Comparaison des différentes méthodologies de conception

concepteurs. Leurs avantages et inconvénients sont récapitulés dans le Tableau 1.1.

Les approches sont comparées au niveau de leurs performances calculatoires, leurs latences, leurs consommations énergétiques, leurs flexibilités à l'exécution, leurs coût initial et l'effort de conception. La Figure 1.6 propose une représentation de type diagramme en étoile des caractéristiques des différentes méthodologies de conception.

Dans le cadre de ces travaux de thèse, nous nous intéressons à la conception de systèmes de communications numériques. Ce cadre applicatif exigeant implique des niveaux de performances élevés : débit élevé, latence faible, faible consommation d'énergie, etc. L'émergence de la 5G fait monter en puissance les applications IoT, ce qui implique des temps de développement courts. La multiplicité des cadres applicatifs nécessitent de plus une flexibilité accrue des systèmes et des méthodologies de conception. Pendant très longtemps la conception d'architectures dédiées [17] a été perçue comme une approche adaptée pour la conception des systèmes de communications numériques. Par la suite des approches basées sur la conception d'ASIP [18] ont été proposées afin d'apporter plus de flexibilité. L'avènement de la SDR (radio logicielle) a motivé le développement de solutions sur cibles programmables (GPP, GPU) [19]. Cette flexibilité accrue des solutions logicielles s'est faite au dépend des performances.

Afin de répondre à la fois aux exigences de performance et de flexibilité, nous évaluons dans cette thèse l'intérêt de l'utilisation d'outils HLS pour l'implémentation de systèmes de communications numériques. Dans la suite de ce chapitre, les communications numériques et plus particulièrement les codes correcteurs d'erreurs sont introduits.

1.2 Les systèmes de communications numériques

Depuis deux décennies, nous observons une augmentation constante du nombre de systèmes communicants dans notre environnement. L'utilisation de ces derniers a fortement augmentée depuis la mise en place de la 4G au début des années 2010. La standardisation de la 5G couplée avec le déploiement de l'internet des objets (IoT) laisse présager d'une forte croissance de l'utilisation de systèmes communicants et une diversification des domaines applicatifs. Un des enjeux pour ces systèmes communicants est la conception d'éléments de communication efficaces. En effet, le nombre très important de contextes applicatifs est une problématique majeure pour le dimensionnement et la conception des unités de traitement dédiées aux futurs systèmes de communications numériques. C'est pourquoi, les travaux de recherches se sont focalisés sur l'implémentation d'éléments de traitement pour les communications numériques. Dans la section suivante nous présentons une chaîne typique de communications numériques et nous approfondirons la partie liée au codage / décodage canal qui possède une forte complexité calculatoire.

1.2.1 La chaîne de communications numériques

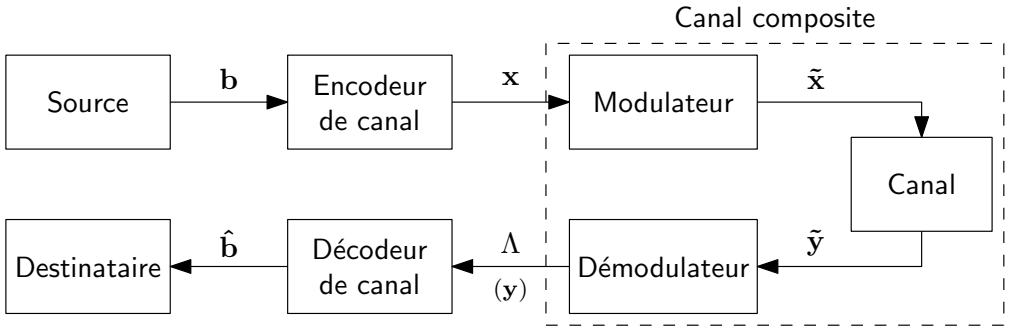

Figure 1.7 – Modèle simplifié d'une chaîne de communication numérique

Une chaîne de communications numériques typique est illustrée dans la Figure 1.7. Cette dernière a été conceptualisée par Claude Shannon [20] en 1948. Elle décrit les principales étapes de transmission de données numériques depuis une **source** vers un **destinataire**. L'échange d'informations se fait à travers un **canal**. Celui-ci représente par exemple une antenne d'émission, l'espace libre dans lequel transite l'information et une antenne de réception, dans le cas d'une communication sans fil. Les informations transmises à travers le canal sont dans le domaine continu tandis que les informations provenant de la source sont dans le domaine discret. C'est le rôle du **modulateur** de transformer le flux binaire en forme d'onde continue pour une transmission à travers le canal. Au niveau du canal les informations transmises subissent de nombreuses perturbations comme le bruit thermique des composants de la chaîne de transmission ou bien les interférences générées par les

autres utilisateurs du canal de transmission. Le **démodulateur** a pour but de transformer les informations provenant du canal dans le domaine discret. Si les perturbations issues du canal sont trop importantes alors le démodulateur peut fournir une estimation erronée des données transmises. Or, l'objectif d'une chaîne de communications numériques est de fournir l'exact réplique de la séquence binaire émise par la source au destinataire. Afin de mesurer la qualité d'une chaîne de transmission on utilise souvent la métrique du taux d'erreur binaire (BER).

Pour minimiser le BER et donc augmenter la qualité d'une transmission, un moyen efficace est d'ajouter un encodeur et un décodeur de canal respectivement dans les parties émission et réception de la chaîne de transmission. Si des codes en blocs sont considérés alors l'encodeur transforme une **séquence binaire b** de taille **K** en un **mot de code x** de **N** bits. Le rendement du code, défini par $\mathbf{R} = \mathbf{K}/\mathbf{N}$, est inférieur à 1 car de la redondance est introduite dans la séquence binaire. Cette redondance est utilisée par le décodeur de canal afin d'améliorer l'estimation des bits transmis et potentiellement corriger des erreurs de transmission. L'ajout de codes correcteurs d'erreurs améliore fortement la qualité d'une transmission. Dans la suite de l'étude nous considérerons l'ensemble modulateur - canal - démodulateur comme un seul module nommé le **canal composite**. Le canal composite a pour entrée un mot de code $\mathbf{x}$ et sa sortie est une estimation de $\mathbf{x}$ noté Λ.

Il existe plusieurs modèles de canal composite, dédiées à différents systèmes de communications numériques. Afin de simplifier l'étude, le modèle de canal composite qui sera considéré tout le long de ce manuscrit est composé d'une modulation et d'une démodulation à changement de phase binaire (BPSK). Cette modulation numérique associe à chaque valeur d'entrée binaire $\mathbf{x} \in \{0,1\}$ une valeur réelle $\tilde{\mathbf{x}} \in \{-1,1\}$. Le canal est modélisé à l'aide d'un bruit blanc additif gaussien à entrée binaire (BI-AWGN). Il additionne à chaque valeur de sortie du modulateur $\tilde{\mathbf{x}}$ une variable aléatoire ν à valeurs réelles ayant une distribution gaussienne centré en 0 et une densité spectrale N_0. Ainsi, la sortie du canal vaut $\tilde{y}_i = \tilde{x}_i + \nu$. Les performances de décodage des codes correcteur d'erreurs sont le plus souvent rapportées au **rapport signal à bruit** (SNR). Il se définit par E_b/N_0, avec $E_b = \dfrac{\mathbb{E}(\tilde{x}^2)}{R}$, l'énergie moyenne par bit d'information.

Les estimations en sortie du démodulateur sont exprimées sous la forme de rapport de vraisemblance logarithmique (LLR). Le signe du LLR détermine pour chaque valeur $y_i \in \mathbf{y}$ la valeur binaire d'entrée de $x_i \in \mathbf{x}$ la plus probable. La valeur absolue correspond au degré de fiabilité de l'information. Les LLRs $\lambda_i \in \Lambda$ en sortie du démodulateur sont définis par : $\lambda_i = \log\left(\dfrac{P_r(y_i|x_i=0)}{P_r(y_i|x_i=1)}\right)$

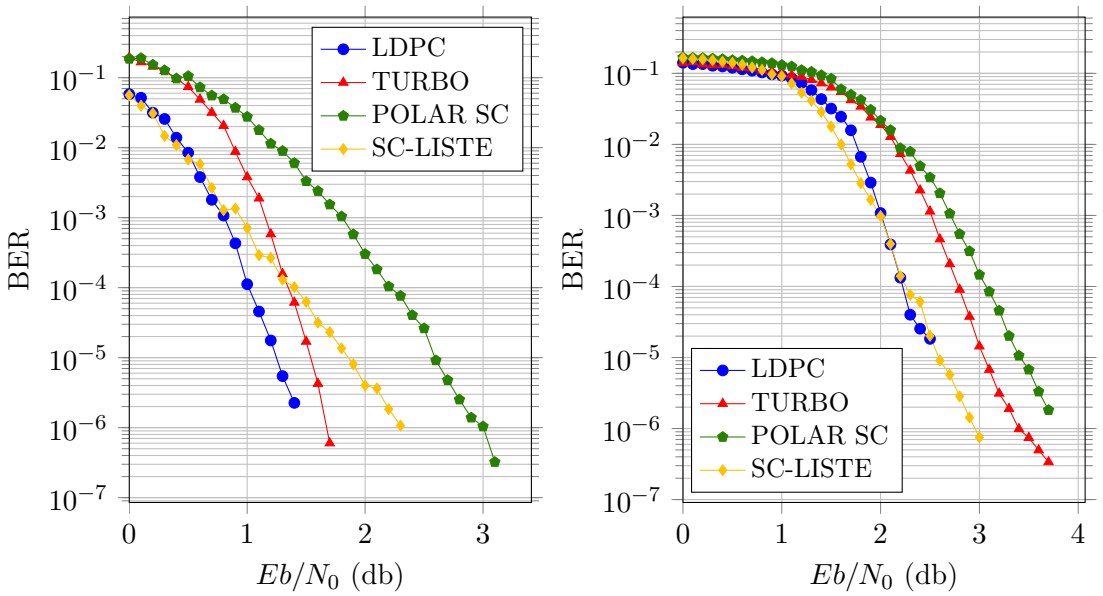

FIGURE 1.8 – *A gauche :* Courbes de BER tracées avec un rendement $R = 1/3$ pour un code LDPC de taille $N = 2040$, un turbo code de taille $N = 2028$, un code Polaire SC et un code Polaire SC-Liste de taille $N = 2048$.
A droite : Courbes de BER tracées avec un rendement $R = 2/3$ pour un code LDPC de taille $N = 2112$, un turbo code de taille $N = 2028$, un code Polaire SC et un code Polaire SC-Liste de taille $N = 2048$.

1.2.2 Les codes correcteurs d'erreurs

Dans une chaîne de communications numériques, le décodage de l'information représente un défi majeur au niveau de l'implémentation temps réel. En effet le décodage des codes correcteurs d'erreurs nécessite des algorithmes possédant de fortes complexités mémoires et calculatoires. Cela est problématique pour des implémentations à haut débit et nécessitant une complexité matérielle raisonnable. De plus, la majorité des algorithmes de décodage sont irréguliers au niveau de la complexité calculatoire et des accès aux données. Cela complexifie les implantations matérielles et/ou logicielles.

Il existe plusieurs familles de codes correcteurs d'erreurs. Parmi les plus utilisées actuellement nous pouvons citer les turbo codes [21], les codes LDPC [22] ou encore les codes polaires [23]. Chaque famille a ses propres caractéristiques au niveau de la complexité calculatoire et des performances de décodage. L'association de ces deux facteurs rend les comparaisons entre ces différentes familles complexes en l'absence de tout contexte applicatif.

Les turbo codes découverts au début des années 90 sont des codes convolutifs. Les codes LDPC sont quant à eux des codes en blocs adoptés depuis le début des années 2000. Plus

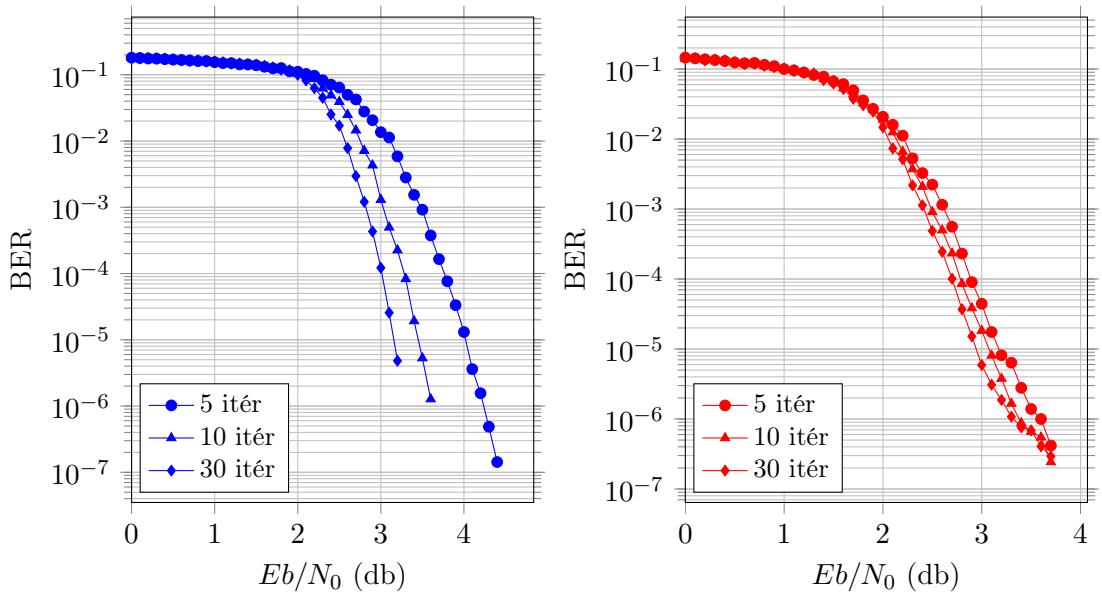

Figure 1.9 – *A gauche :* Courbes de BER de codes LDPC de taille 2112 et de rendement $R = 2/3$ pour différentes valeurs du nombre d'itérations de décodage.
A droite : Courbes de BER de turbo codes de taille 2028 et de rendement $R = 2/3$ pour différentes valeurs du nombre d'itérations de décodage.

récemment les codes polaires, eux aussi des codes en blocs, ont été découverts. Les turbo codes et les codes LDPC se décodent à l'aide d'un processus itératif, contrairement aux codes polaires qui se décodent par un processus séquentiel.

La Figure 1.8 fournit les performances BER en fonction du SNR pour ces 3 familles de code. Le décodage des turbo codes a été réalisé à l'aide d'un algorithme de type BCJR [24] avec 20 itérations de décodage. Le décodage des codes LDPC implémente l'algorithme SPA [22] avec 100 itérations de décodage. Le décodage des codes polaires est effectué par deux algorithmes de décodage qui sont l'algorithme SC et l'algorithme SC-Liste avec une taille de liste $L = 32$. Le graphe de gauche a été réalisé avec un rendement $R = 1/3$ et celui de droite avec un rendement $R = 2/3$. Toutes ces simulations ont été réalisées à l'aide de l'outil AFF3CT [25], qui est un outil de simulation logicielle de chaîne de communications numériques.

Dans le cadre d'une mise en oeuvre de ces codes, des simplifications et des modifications des algorithmes de décodage peuvent être appliquées afin de rendre l'implémentation plus efficace. Par exemple dans le cas du décodeur LDPC présenté, une réduction du nombre d'itérations et une simplification de l'algorithme sont des optimisations souvent appliquées. Ces simplifications peuvent avoir un impact sur les performances du code.

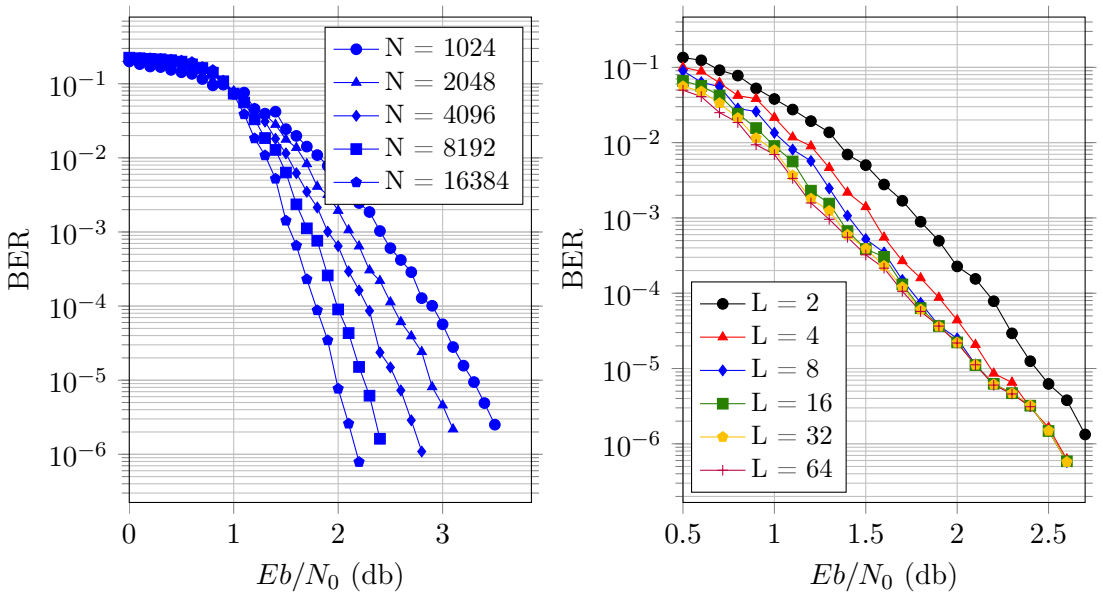

Figure 1.10 – *A gauche :* Courbes de BER de codes Polaires pour un algorithme SC selon différentes valeurs de N de rendement $R = 1/2$.
A droite : Courbes de BER de codes Polaires pour un algorithme SC-Liste selon différentes tailles de liste L avec $N = 2048$ et $R = 1/2$

Les performances des codes correcteurs d'erreurs et leurs complexité varient en fonction de la taille de trame utilisée et de l'algorithme de décodage implémentée. De plus, pour un code donné, les performances de décodage des codes LDPC et des turbo codes varient fortement en fonction du nombre d'itérations exécutées. Les codes polaires basés sur un algorithme SC-Liste voient leurs performances varier en fonction de la taille de la liste.

Le jeu de courbes en Figure 1.9 illustre l'impact du nombre d'itérations de décodage sur les performances BER pour les codes correcteurs d'erreurs itératifs comme les code LDPC (graphe de gauche) et les turbo codes (graphe de droite). Les performances ont été tracés pour 5, 10 et 30 itérations de décodage. L'augmentation du nombre d'itérations de décodage améliore de manière significative le pouvoir de correction, cependant d'un point de vue mise en oeuvre du décodeur cela a un impact négatif sur le débit et la latence de décodage.

Le jeu de courbes en Figure 1.10 illustre l'influence de la taille de trame N sur les performances de décodage des codes polaires SC (graphe de gauche) et l'influence de la taille de liste L sur les performances des codes polaires SC-Liste (graphe de droite). Le pouvoir de correction l'algorithme de décodage SC peut en théorie atteindre la capacité du canal lorsque N tend vers l'infini. L'augmentation du nombre de liste améliore de manière significative les performances de l'algorithme SC-Liste, cependant cela à un

impact conséquent sur la complexité mémoire et la complexité calculatoire.

Comme montré ci-dessus, plusieurs paramètres algorithmiques influent sur le pouvoir de correction des codes correcteurs d'erreurs, néanmoins ces mêmes paramètres impactent aussi fortement l'efficacité de leurs implémentations. Les familles de décodeurs se différencient aussi sur les optimisations qui peuvent être appliqués lors de leurs implantations. Les codes polaires décodé à l'aide de l'algorithme SC présentent la plus faible complexité calculatoire tandis que les codes LDPC ou les turbo codes sont plus propices à la parallélisation de leurs processus de décodage.

1.2.3 L'implémentation de codes correcteur d'erreurs

Il existe un nombre important de cas d'usage pour les codes correcteurs d'erreurs : communications spatiales, communications mobiles, communications filaires, stockage d'informations, etc. En conséquence de nombreux standards ont vu le jour durant les deux dernières décennies. L'hétérogénéité des standards à abouti au développement d'un nombre important de décodeurs matériels adaptés à différents cadres applicatifs. Ainsi un nombre important de décodeurs peuvent implanter un même code mais en répondant à des contraintes différentes. On peut distinguer deux types de paramètres lors de l'implantation :

— Les paramètres algorithmiques - ce sont les caractéristiques intrinsèques du code tel que de la taille N des trames à décoder ou le rendement R du code. Il existe aussi des caractéristiques propres aux différentes familles, comme par exemple la structure de la matrice de parité pour un code LDPC ou encore les positions des bits gelés pour un code polaire. Ces paramètres peuvent varier d'un standard à l'autre. De plus dans un même standard il peut être nécessaire de supporter plusieurs de ces configurations.

— Les paramètres architecturaux - ce sont les caractéristiques d'implémentation. Il peut s'agir du format de quantification des données internes du décodeurs, du facteur de parallélisme des calculs, du nombre d'itérations de décodage pour un algorithme itératif ou encore des simplifications algorithmiques. Ces paramètres sont dérivés des contraintes systèmes. Ils impactent sur l'architecture et donc sur la complexité matérielle et énergétique du décodeur. De plus, ils peuvent aussi impacter les performances de décodage.

La pluralité des cadres applicatifs dans les communications numériques nécessite un spectre continu dans l'espace des solutions pour l'implémentation d'un décodeur. Dans l'exemple de la 3GPP 5G [26], le cadre applicatif du standard va de décodeurs soumis

à de faibles débits pour des réseaux de capteurs [27] à des décodeurs très hauts débits (quelques Gbit/s) pour de la vidéo [28] en passant par des décodeurs à très faibles latences pour les voitures autonomes [29] ou des décodeurs plus hétérogènes dans le cadre de l'internet des objets (IoT) [30].

La diversité du standard 5G représente un cadre très intéressant pour l'étude de l'implémentation de décodeurs. C'est pourquoi dans le reste du manuscrit nous nous focaliserons sur l'étude de l'implémentation de 2 familles de codes correcteurs d'erreurs qui ont été sélectionnés dans le standard 3GPP 5G :

— les codes LDPC, décodés à l'aide d'un algorithme itératif de type min-sum,

— les codes polaires, décodés à l'aide d'un algorithme SC et d'un algorithme SC-Liste.

1.2.4 La méthodologie proposée

L'approche usuelle employée durant près de deux décennies afin d'implanter des décodeurs EEC au sein de systèmes communicants consistait à imaginer et à développer des architectures au niveau RTL [17]. Ces dernières avaient pour objectif de maximiser le débit. Ces architectures finement optimisées sur cible ASIC/FPGA pouvaient délivrer des débits élevés face aux besoins des cadres applicatifs.

Cependant, des besoins en terme de flexibilité lié au déploiement des systèmes de type radio logicielle (SdR) ou Cloud-RAN ont motivé le développement de décodeurs ECC plus flexibles. Ces derniers furent déployés sur des cibles programmables de type GPP ou GPU. L'utilisation de descriptions logicielles des décodeurs fournie une flexibilité et une évolutivité importante vis-a-vis des solutions antérieures. Cependant, même si l'efficacité de certaines implantations en termes de débit est comparable avec des implantation de type FPGA ou ASIC, leurs consommations énergétiques reste trop importantes.

La maturité des outils de synthèse HLS et des méthodologies associées, couplé avec l'augmentation de la capacité des FPGA a développé l'intérêt des chercheurs pour ces approches qui tentent de coupler flexibilité, débit et performance énergétique. La troisième génération d'outils HLS industriels [31, 32, 33] ou académiques [34, 35] permettent d'accélérer le développement de systèmes complexes [36]. Certaines propositions ont été faites concernant les codes LDPC. Toutefois ces précédents travaux portant sur l'utilisation de la méthodologie HLS pour l'implémentation de décodeurs LDPC fournissent des performances médiocres par rapport au reste de l'état de l'art. Ces travaux de la littérature se sont focalisés sur la réutilisation de descriptions ciblant une architecture GPU à travers un flot de conception HLS pour une implémentation sur cible FPGA. Or

l'utilisation de la synthèse HLS n'est pas aussi directe et triviale. Elle nécessite en plus des compétences en conception d'architectures numériques pour délivrer de bon résultats. En effet même si la génération des architectures RTL dépend fortement de l'outil HLS utilisé, une description non adaptée ne permet pas d'atteindre de bonnes performances.

Dans le cadre de ces travaux de thèse, nous allons évaluer la capacité des outils de HLS à :

1. supporter les algorithmes de décodage des codes correcteurs d'erreurs,
2. concevoir des architectures numériques efficaces en comparaison de celles décrites au niveau RTL.

2 Modèle d'architectures de décodeurs LDPC

Dans ce chapitre 2 la méthodologie HLS basée sur l'utilisation de modèles comportementaux est appliquée à la conception de décodeurs de codes LDPC. Dans la section 2.1 les codes LDPC sont d'abord présentés. Les codes LDPC sont introduits ainsi que leurs différents algorithmes de décodages, les stratégies d'implémentation matérielles de ces algorithmes sont ensuite présentées. La section 2.2 présente les contributions originales de ces travaux. Deux modèles architecturaux de décodeur LDPC sont proposés et évalués. Enfin les résultats expérimentaux des approches sont détaillés dans la section 2.3.

Introduction

Les codes LDPC (Low-density-parity-check) [22] sont des codes correcteurs d'erreurs (ECC) populaires car ils atteignent des performances proches de la capacité du canal gaussien. Ils furent découverts en 1962 par Gallager. Cependant la forte complexité calculatoire du processus de décodage a empêché leur utilisation à l'époque. Ils ont été exhumés au début des années 2000 grâce aux progrès en termes d'intégration de l'électronique moderne. Leur bon compromis entre complexité et performance de décodage a entraîné la sélection des codes LDPC dans les normes de communications numériques telles que WiMAX (worldwide interoperability for microwave access), WiFi (wireless fidelity), WiGig (wireless gigabit alliance) et 3GPP 5G [37, 38, 39, 40, 26].

Historiquement, pour des questions de performances, les décodeurs LDPC étaient implantés à l'aide de circuits de type ASICs ou FPGA [41, 17]. Les architectures numériques étaient conçues et optimisées spécifiquement par rapport au besoin applicatif. Ces développements chronophages permettaient cependant d'atteindre de très hauts niveaux de performances en termes de débit, de latence et de consommation d'énergie.

Cette approche indispensable pour les systèmes très contraints présente cependant de sérieuses limitations. En effet, les systèmes de communications numériques actuels fournissent de nombreux cadres applicatifs nécessitant un prototypage rapide afin d'étudier ou de valider des solutions techniques. Ainsi les délais de développement et de mise sur le marché (TTM) doivent être minimisés tout en conservant un niveau de performance élevé.

Depuis le début des années 2000 beaucoup de travaux se sont focalisés sur (1) la réduction de la complexité calculatoire et mémoire du processus de décodage et (2) l'amélioration des performances de décodage (BER/FER). La plus part de ces travaux ont été validés par prototypage sur cible ASIC/FPGA.

Cependant, le développement de systèmes de type radio logicielle (SDR) ou réseaux d'accès radio dans le cloud (C-RAN ou cloud-ran) a nécessité la mise en place d'approches alternatives offrant beaucoup plus de flexibilité. Afin d'atteindre cet objectif, l'utilisation d'architectures programmables logicielles a été investiguée.

Falcao *et al.* [19] et Le Gal *et al.* [42, 43] évaluent la pertinence des architectures multi-/many-coeurs, respectivement, les CPU et les GPU. Pour garantir (maintenir) un débit élevé et une faible latence dans les applications SdR ou C-RAN, des descriptions algorithmiques spécifiques permettant d'exhiber un parallélisme massif ont été proposées.

TABLEAU 2.1 – Implémentations logicielles de l'algorithme de décodage Min-Sum (ordonnancement par inondation)

Ref.	Année	Outil HLS	Langage	Cible FPGA	Débit	Complexité matérielle
[45]	2014	Altera OpenCL	OpenCL	Stratix-5	++++	++++
[46]	2016	Vivado HLS	SystemC	Spartan-6	+	++
[48]	2014	MaxCompiler	Java	Stratix-5	++	++++
[48]	2014	Altera OpenCL	OpenCL	Stratix-5	++	++++
[49]	2016	SDSoC	C/C++	Artix-7	+	++

La flexibilité de ce type de solutions provient de l'usage de langages de haut niveau [1] : C, C+++, CUDA (compute unified device architecture) et OpenCL (open computing language). Les travaux traitant du développement de décodeurs logiciels optimisés ont démontré qu'il était possible d'obtenir des débits proches de ceux atteints par des circuits ASIC tout en maintenant des latences acceptables vis à vis des standards de communication. Cependant, l'aspect consommation énergétique de ces solutions logicielles est en retrait vis à vis des solutions matérielles dédiées.

Actuellement, les systèmes reconfigurables représentent des plates-formes matérielles efficaces au niveau de la puissance de calcul et de la consommation énergétique pouvant être des solutions alternatives aux systèmes basés sur GPP ou GPUs.

Afin de pouvoir exploiter des cibles reconfigurables (FPGA), des approches basées sur l'utilisation d'outils de synthèse de haut niveau (HLS) sont possibles. Des efforts récents ont portés sur l'exploration de la génération automatique de décodeurs LDPC à partir de modèles comportementaux [44, 45, 46]. Une synthèse de ces travaux est fournie dans le Tableau 2.1. Les études sont basées sur des codes sources issus d'implémentations GPU basées sur CUDA ou OpenCL pour décrire les décodeurs LDPC [47, 1]. Les débits atteints sont comparables à ceux d'architectures dédiées décrites au niveau RTL. De plus, ces débits sont compétitifs par rapport à ceux d'implémentations sur cible GPP et/ou GPU. Toutefois, vis a vis des architectures développées au niveau RTL, la complexité matérielle et la consommation énergétique augmentent considérablement. Cela était prévisible vu que ni transformations ni optimisations particulières n'ont été appliquées aux modèles logiciels développés pour GPP/GPU [47].

Afin de résoudre ce problème et de concevoir des décodeurs LDPC flexibles et efficaces, deux modèles comportementaux ont été développés dans le cadre de cette thèse. Ces derniers, décrits en langage SystemC sont génériques et flexibles. Ce travail constitue une

1. En comparaison avec des langages RTL tels que VHDL ou Verilog

contribution indéniable par rapport aux solutions architecturales précédentes :

— Contrairement à l'état de l'art, la description du modèle comportemental bénéficie de l'ordonnancement de décodage LDPC en couches horizontales [50]. Cette organisation des calculs utilisée dans la plupart des architectures dédiées de décodeur LDPC réduit la complexité calculatoire au prix d'un léger impact sur le parallélisme des calculs.

— Pour augmenter le parallélisme de calcul, le modèle est basé sur le paradigme de vectorisation de données multiples à instruction unique (SIMD). La largeur de vectorisation est configurée en fonction de la matrice du code LDPC.

— Les avantages et les limitations des modèles comportementaux proposés sont estimés théoriquement puis validés expérimentalement.

— Les performances des décodeurs LDPC matériels générés sont similaires à celles de décodeurs RTL décrits manuellement. Par rapport aux autres travaux basés sur la HLS, l'approche apporte des améliorations significatives au niveau de la complexité matérielle et du débit de décodage.

La suite du chapitre est organisé comme suit. La section 2.1 présente les codes LDPC et les algorithmes de décodage. La section 2.2 porte sur les descriptions comportementales des algorithmes de décodage et des architectures qui en résultent. Les approches expérimentales et les résultats sont détaillés dans la section 2.3. Enfin, la section 2.3.3 conclut ce chapitre.

2.1 Codes LDPC et algorithmes de décodage

2.1.1 Introduction

Les codes LDPC sont des codes linéaire de correction d'erreurs [22] définis par une matrice de faible densité $\mathbf{H}$ comme illustré par l'Équation 2.1. La matrice $\mathbf{H}$ est composé de N colonnes and $M = N - K$ lignes. Un vecteur $\mathbf{x}$ de N bits est généré pour transmettre K bits d'information. Un vecteur Λ de N informations bruitées est reçu après diffusion à travers le canal de transmission. Puis, les M équations de parité sont utilisées pour corriger le vecteur Λ [19]. Le rendement du code LDPC est défini par $R = \frac{K}{N}$.

Les N colonnes et les M lignes de $\mathbf{H}$ représentent respectivement les nœuds variables (VN) et les nœuds de vérification de parité (CN). Pour chaque élément de $\mathbf{H}$, il existe une dépendance entre les éléments VN et CN. Le nombre d'éléments de la $j^{\text{ème}}$ ligne et de la $i^{\text{ème}}$ colonne de $\mathbf{H}$ sont respectivement les degrés de CN et VN. Ils sont sont notés $d_c(j)$ et $d_v(i)$. Un graphe de Tanner est un graphe bipartite qui peut être utilisé pour

représenter un code LDPC. Par exemple, le graphe de Tanner représenté en Figure 2.1 correspond au code LDPC (8,5) décrit dans l'équation 2.1.

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

Dans leur forme générale, les matrices **H** sont irrégulières et non structurées. Les valeurs non nulles sont distribuées aléatoirement de tel sorte que d_c et d_v ne soient pas constant. La conception d'une architecture de décodeur efficace pour un code LDPC non structuré est admise de tous comme étant une tâche ardue [51]. Ainsi pour faciliter la conception des décodeurs LDPC matériels, des codes LDPC quasi cycliques (QC-LDPC) ont été adoptés dans la plupart des normes de communications numériques [37, 38, 39, 40], y compris le standard 5G 3GPP [26].

Les codes QC-LDPC sont des codes LDPC structurés. Leur structure interne est composée de de $(N/Z) \times (M/Z)$ sous-matrices de taille $Z \times Z$. Ces sous-matrices sont soit des matrices identités circulantes soit des matrices nulles. Cette structure particulière définie des sous-ensembles de Z éléments CN et VN indépendants. Par conséquent, le facteur Z représente pour les codes QC-LDPC le niveau minimum de parallélisme possible pour des implémentations matérielles ou logicielles. Dans ce cadre, aucun conflit d'accès mémoire n'existe lors du décodage si Z bancs mémoires distincts sont disponibles pour le stockage

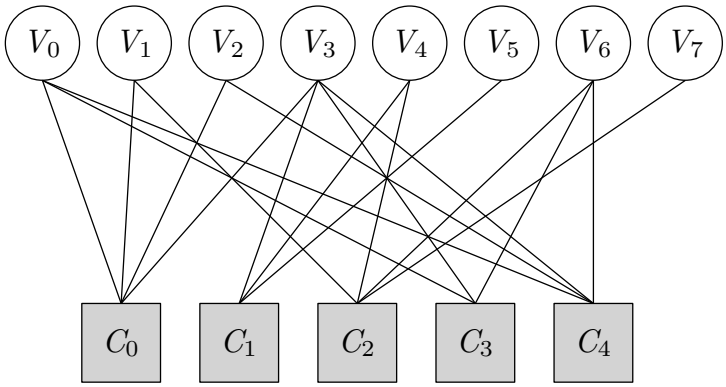

Figure 2.1 – Représentation d'un code LDPC sous forme d'un graphe de Tanner

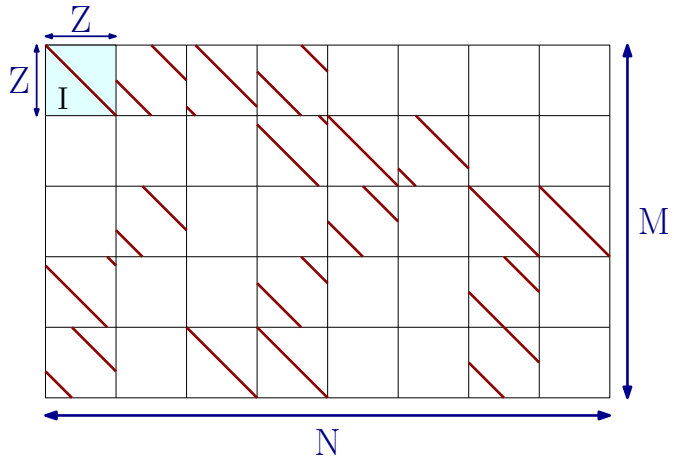

Figure 2.2 – Exemple de matrice de code LDPC Quasi-Cyclique

des éléments VN. La Figure 2.2 représente la matrice $\mathbf{H}_c$ d'un code QC-LDPC défini par l'Equation 2.2 sous sa forme compressée [2]. La forme compressée QC de $\mathbf{H}_c$ fournit des décalages appliqués à la matrice identité $Z \times Z$. Lorsque la valeur de décalage dans la matrice $\mathbf{H}_c$ est égale à -1, une sous-matrice nulle $Z \times Z$ est appliquée.

$$H_c = \begin{pmatrix} 0 & 4 & 1 & 5 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 7 & 0 & 2 & -1 & -1 \\ -1 & 3 & -1 & -1 & 4 & -1 & 0 & 0 \\ 7 & -1 & -1 & 5 & -1 & -1 & 4 & -1 \\ 3 & -1 & 0 & 0 & -1 & -1 & 4 & -1 \end{pmatrix} \tag{2.2}$$

2.1.2 Algorithmes de décodage

Le décodage de codes LDPC est réalisé à l'aide d'algorithmes de type passage de messages. L'algorithme somme-produit (SPA) [22] est l'algorithme de référence pour le décodage de codes LDPC. Le SPA nécessite que les entrées du décodeur soient sous la forme de rapports logarithmes de vraisemblance (LLR). Les LLRs sont définis par $L(c) = \log(\frac{P(c=0)}{P(c=1)})$ avec c un bit de la trame à décoder. Les calculs de vérification de parité associés aux éléments CN mettent à jour les messages $\mathcal{L}^i_{mn}$ transmis vers les VNs à partir des messages entrants $\mathcal{L}^i_{nm}$ reçus depuis les éléments VNs. Les équations 2.3 et 2.4 détaillent les calculs de signe

2. Cette forme compressée est couramment utilisée dans les normes de communications numériques pour les matrices $\mathbf{H}$ QC-LDPC

et de magnitude des messages $\mathcal{L}_{mn}^{i}$.

$$sign(\mathcal{L}_{mn}^{i}) = \left[\prod_{n' \in \Phi(m)/n} sign(\mathcal{L}_{n'm}^{(i-1)}) \right] \tag{2.3}$$

$$|\mathcal{L}_{mn}^{i}| = 2 \tanh^{-1} \left[\prod_{n' \in \Phi(m)/n} \tanh \frac{|\mathcal{L}_{n'm}^{(i-1)}|}{2} \right] \tag{2.4}$$

Dans les équations 2.3 et 2.4, $\mathcal{L}_{mn}^{i}$ est le message entrant dans CN_n et provenant de VN_m à la $i^{\text{ème}}$ itération. $\mathcal{L}_{nm}^{i}$ est le message entrant dans VN_m et provenant de CN_n à la $i^{\text{ème}}$ itération. $\Phi(m)$ est l'ensemble des noeuds variables (VN) connectés au CN_m. $\Phi(m)/n$ est l'ensemble des noeuds variables (VN) connectés au noeud de parité CN_m à l'exception du noeud VN_n à proprement parlé.

La nature des opérations réalisées dans l'algorithme SPA n'est pas adaptée à une implémentation matérielle efficace. En effet, il nécessite des fonctions transcendantales. Une simplification de l'algorithme visant à supprimer les opérations transcendantales et à faciliter le passage en virgule fixe de l'algorithme de décodage a été proposé dans [52] : l'algorithme Min-Sum. Cet algorithme approxime le calcul de la fonction arctan à l'aide d'un calcul de minimum comme cela est rapporté dans l'équation 2.5. Cette approximation et ses variantes [19, 17, 41] favorisent un compromis très intéressant entre d'une part la complexité calculatoire et d'autre part les performances de décodage.

$$|\mathcal{L}_{mn}^{i}| = \left[\min_{n' \in \Phi(m)/n} |\mathcal{L}_{n'm}^{(i-1)}| \right] \tag{2.5}$$

L'ordonnancement des calculs réalisé lors du décodage peut être réalisé à l'aide d'une approche par inondation. Cette approche se décompose en deux phases (TPMP). Tout d'abord, tous les messages $\mathcal{L}_{nm}$ des N VNs vers les M CNs sont calculés et transmis. Puis, tous les messages $\mathcal{L}_{mn}$ des M CNs vers les N VNs sont traités. Ces deux phases complémentaires sont exécutées successivement I_{max} fois pour corriger et ainsi décoder la

trame reçue par le décodeur. Cette ordonnancement des calculs favorise un haut niveau de parallélisme. En effet, les N VNs peuvent être évalués en parallèle afin de générer les messages $\mathcal{L}_{nm}$ durant la première phase. Durant la seconde phase, les M calculs de parité (CN) générant les messages $\mathcal{L}_{mn}$ peuvent aussi être exécutés simultanément.

L'empreinte mémoire Mem_f du décodeur résultant est égale à $N+2\times m$, avec m le nombre de messages échangés entre les éléments CNs et VNs. Cet ordonnancement favorisant un parallélisme de calcul important est adapté à des implémentations logicielles et matérielles [53, 54]. Les travaux antérieurs traitant de la génération automatique de décodeurs LDPC exploite aussi ce type d'ordonnancement comme cela est indiqué dans le tableau 2.1. Toutefois cet ordonnancement présente un sérieux inconvénient. En effet, l'échange de messages entre les éléments VNs et CNs nécessite un entrelacement des informations. Or, cet entrelacement est contraint par la structure de la matrice $\mathbf{H}$. Cette opération est complexe à gérer et pénalisante pour les architectures de décodage de codes LDPC qu'elles soient logicielles ou matérielles.

D'autres ordonnancements existent dans la littérature [50, 55, 56, 57, 58]. Ils possèdent des avantages et des limitations en ce qui concerne le niveau de parallélisme, l'empreinte mémoire, la régularité, etc. L'ordonnancement par couches horizontales décrit dans [50] et formalisé dans Algorithme 1, est fréquemment utilisé lors de la conception de décodeurs matériels. En effet, à pouvoir de correction équivalent ce dernier nécessite à peu près 2 fois moins d'itérations que l'approche par inondation. De plus, l'empreinte mémoire est réduite de 30%, car $Mem_L = N + m$ données doivent être stockées. Malheureusement, le séquencement des calculs introduit des dépendances de données qui réduisent le parallélisme de calcul. Cependant, en raison de sa convergence rapide et de sa faible empreinte mémoire, l'ordonnancement par couches horizontales est populaire pour les implémentations de décodeurs logiciels et matériels [54, 43, 59, 60, 42]. D'autres ordonnancements ont été proposés [55, 56, 57, 58], mais leurs propriétés les rendent moins pertinents.

Contrairement à l'état de l'art concernant la génération automatique d'architecture de décodeurs LDPC [1, 47, 46, 44, 45, 48, 48, 49], l'algorithme Min-Sum associé à un ordonnancement par couches horizontales a été sélectionné dans le cadre de ces travaux de thèse. La conception du modèle comportemental, la gestion des dépendances de données, les optimisations et les transformations appliquées sont détaillées dans les sections suivantes.

Algorithme 1: Algorithme Min-Sum basé sur un ordonnancement pour couches horizontales

Kernel 1 : Initialisation
Charge la trame dans la mémoire accumulatrice de VN
pour tout $m \in \Psi$,$n \in \Phi(m)$ **faire** $\{\Psi$ *est l'ensemble des CNs*$\}$
  Initialisation de la mémoire message : $\mathcal{L}_{mn}^{(0)} = 0$
fin pour
Kernel 2 : Traitement des Imax itérations de décodage
pour tout $i = 1 \rightarrow (Imax)$ **faire**
  pour tout $m \in \Psi$ **faire** $\{$*Pour tout CN*$\}$
Kernel 2.1 : Calcul les messages entrants $\mathcal{L}_{nm}$
    pour tout $n \in \Phi(m)$ **faire** $\{$*VNs contribuant à CN*$\}$
      $\mathcal{L}_{nm}^{i} = VN_n - \mathcal{L}_{mn}^{(i-1)}$
    fin pour
Kernel 2.2 : Génère les messages sortants $\mathcal{L}_{mn}$
    pour tout $n \in \Phi(m)$ **faire**
      $$sign(\mathcal{L}_{mn}^{i}) = \left[\prod_{n' \in \Phi(m)/n} sign(\mathcal{L}_{n'm}^{(i-1)}) \right]$$
      $$|\mathcal{L}_{mn}^{i}| = \left[\min_{n' \in \Phi(m)/n} |\mathcal{L}_{n'm}^{(i-1)}| \right]$$
    fin pour
Kernel 2.3 : Mise à jour des VNs
    pour tout $n \in \Phi(m)$ **faire**
      $VN_n = \mathcal{L}_{nm}^{(i)} + \mathcal{L}_{mn}^{(i)}$
    fin pour
  fin pour
fin pour
Kernel 3 : Décision dure sur chaque LLR de la trame
pour tout $n \in$ *trame* **faire**
  $$c_n = \begin{cases} 0, & \text{if } VN_n \leq 0 \\ 1, & \text{if } VN_n > 0 \end{cases}$$
fin pour
Génération de la trame décodée

2.1.3 Stratégies de Parallélisation

Afin d'atteindre des débits élevés, il est nécessaire de paralléliser le processus de décodage. Différentes approches de parallélisation sont possibles. Pour utiliser au mieux les unités de traitement, différents niveaux de parallélisme sont exploitables :

1. Le parallélisme intra-trame : il peut être aussi bien employé lors de la conception de décodeurs matériels que de décodeurs logiciels. Il exploite le parallélisme de calcul à l'intérieur de l'algorithme de décodage lors du traitement d'une trame. Il permet de réduire fortement la latence de décodage. Cependant, le parallélisme de calcul disponible est limité.
2. Le parallélisme inter-trame : il n'est réellement utile que pour les implémentations de

décodeurs logiciels. Il consiste à décoder plusieurs trames en parallèle [19, 59, 60]. Le parallélisme de calcul provient des Q trames indépendantes décodées en même temps. Il maximise le débit de décodage pour les architectures multi/many coeurs mais implique de larges empruntes mémoires et des latences de décodage relativement élevées.

La stratégie de parallélisation inter-trame [59] adapte le débit de décodage linéairement au nombre de trames décodées. D'un point de vue matériel, il rend possible la fusion de certains éléments de l'architecture tel que le contrôleur. Cependant, il n'améliore pas la latence comparé à un décodeur séquentiel mono-trame. De plus, il augmente linéairement la complexité mémoire avec Q. Ainsi, la conception de décodeurs LDPC à partir d'un modèle basé sur une stratégie de parallélisation inter-trame est inefficace au niveau du débit et de la complexité matérielle [61]. C'est la raison pour laquelle les travaux de thèse portent dans un premier temps exclusivement sur l'exploitation du parallélisme intra-trame.

2.2 Description des modèles architecturaux des décodeurs LDPC

Des travaux antérieurs génèrent des décodeurs LDPC à partir de modèles comportementaux conçus à l'origine pour programmer des cibles many-coeurs. Ces approches utilisent des modèles et des langages de programmation adaptés pour des architectures ayant un grand niveau de parallélisation [1, 47, 46, 44, 45, 48, 49]. Dans un souci de simplicité, tous ces travaux sont basés sur un ordonnancement par inondation afin d'éviter la gestion des dépendances de données présent par exemple dans un ordonnancement par couches horizontales. Pour décrire finement l'algorithme de décodage par couches horizontales et guider avec pertinence les outils HLS, un modèle de description architecturale basé sur des instructions uniques sur données multiples (SIMD) a été appliqué dans notre étude [59, 60, 43].

Cette étude se concentre sur l'utilisation de l'outil Xilinx Vivado HLS [32] qui supporte les modèles décrits en SystemC. L'expressivité du langage SystemC fournit une sémantique précise pour modéliser les entrées/sorties, le type des données et le niveau de parallélisme. Les modèles de décodeurs sont compatibles avec Mentor Catapult [31] et Intel HLS compiler [33] au prix de la conversion des codes sources SystemC en C/C++. Les modèles de décodeur LDPC proposés mettent en œuvre l'ordonnancement par couches horizontales [50] avec une stratégie de parallélisation intra-trame. Deux modèles de décodeurs LDPC comportementaux ont été successivement conçus puis analysés :

— Le premier modèle ($\mathcal{M}_1$) s'inspire d'une description d'un décodeur LDPC sur architecture x86 [43]. Une réécriture du modèle en incluant des optimisations spécifiques à la génération d'architectures a été cependant nécessaire. Cette première approche évalue la qualité de génération d'architectures matérielles via des outils HLS à partir d'une description orientée cible programmable.

— Le second modèle ($\mathcal{M}_2$) est une description comportementale spécialement écrite pour la méthodologie HLS. Cette description vise à lever les verrous identifiés dans le premier modèle $\mathcal{M}_1$. La parallélisation entre les tâches est clairement exposée afin d'améliorer les performances des implémentations matérielles. Les architectures générées avec ce modèle atteignent les débits théoriques avec une plus faible complexité matérielle que celles générées à partir du premier modèle M1.

2.2.1 Description du modèle $\mathcal{M}_1$

2.2.1.1 Caractéristiques spécifiques

Le modèle logiciel détaillé dans [43] est conçu et optimisé pour exploiter les architectures multicoeurs x86 avec le jeu d'instructions AVX2. Afin de générer des architectures matérielles, un framework a été développé. Ce framework fourni un jeu d'instructions de type SIMD dont la largeur est générique. A partir de ces instructions SIMD, une réécriture du modèle logiciel a été réalisée. En effet, les instructions SIMD développées sont plus adaptées à l'algorithme de décodage que celles présentes dans le jeu d'instructions des processeur x86. De plus, elles possèdent des largeurs paramétrables en fonction du besoin contrairement à celles des processeurs qui sont figées sur 128, 256 ou 512 bits. L'approche de programmation SIMD a été préservée pour conserver la même philosophie que celle appliquée dans les travaux décrits dans [43]. Ce travail permet d'aboutir à un modèle architectural générique de décodeur LDPC. Au final, la transformation du modèle a nécessité les étapes suivantes :

1. Le code source du décodeur LDPC décrit dans [43] est encapsulé dans un seul processus SystemC *sc_cthread*. Les entrées/sorties sont déclarées sous forme de ports *sc_fifo_in* et *sc_fifo_out* pour faciliter l'intégration dans des systèmes complets.
2. Selon [62], les sémantiques non synthétisables de la description C/C++ originale sont supprimées et/ou converties en structures synthétisables. Par exemple, les allocations de mémoire dynamiques sont remplacées par des allocations statiques. Les tailles des tableaux statiques sont optimisées en fonction de la matrice **H** à traiter.

```
1
2  template<int BITS>  /* 1ere couche : fonctions scalaires */
3  sc_int<BITS> qabs(const sc_int<BITS> value)
4  {
5  #pragma HLS INLINE
6          const sc_int<BITS> inverted = -value;
7          const sc_int<BITS> result   = (value[BITS-1] == 1) ? inverted : value;
8          return result;
9  }
10
11 template<int QC, int BITS> /* 2eme couche : fonctions vectorielles */
12 sc_int<QC * BITS> vect_abs(sc_int<QC * BITS> a)
13 {
14 #pragma HLS INLINE
15         sc_int<QC * BITS> c;
16         for(int i = 0; i < QC; i += 1){
17 #pragma HLS UNROLL
18                 const sc_int<BITS> oa = a.range(BITS * (i+1) - 1, BITS * i);
19                 c.range(BITS * (i+1) - 1, BITS * i) = qabs( oa );
20         }
21         return c;
22 }
```

Listing 2.1 – Exemple de description de fonctions scalaires et vectorielles pour le calcul de la valeur absolue d'une donnée

3. L'implémentation logicielle originale du décodeur LDPC x86 utilise les intrinsèques d'Intel pour exécuter les instructions AVX2. Ces instructions ne sont pas directement disponibles lors de la conception ciblant des dispositifs ASIC ou FPGA. Les appels de fonctions SIMD AVX2 employés dans le modèle x86 sont donc remplacés pour assurer la conformité matérielle. Une API SIMD personnalisée et générique pour le décodage LDPC a été développée.
4. Comme une largeur de vectorisation personnalisée est possible dans les architectures matérielles, la description comportementale a été réécrite pour bénéficier de la bibliothèque SIMD générique. Là où l'ancien modèle comportemental x86 était limité par des instructions AVX2 de largeur de données $Z <= 32$, le modèle comportemental pour de la HLS supporte toutes valeurs de Z. De plus, certaines opérations qui nécessitaient plusieurs instructions AVX2 pour être exécutées sur cibles x86 peuvent être remplacées par une seule instruction personnalisée.

Comme mentionné précédemment, une API SIMD a été conçue pour tirer pleinement parti des fonctionnalités des circuits FPGA. Le type de données $sc\_bigint<T>$ SystemC est utilisé pour décrire le format des données au bit près. L'API SIMD se scinde en deux couches distinctes comme indiqué dans l'exemple 2.1.

1. La couche de bas niveau définit les fonctions scalaires pour le décodage LDPC : addition saturée, minimum, XOR, extraction de signe, etc. Cette couche est configurable en largeur de bits avec le paramètre de template BITS. Il prend en charge les formats

de quantification variables. Cette flexibilité est rendue possible par l'utilisation des types de données *sc_ int<BITS >* et *sc_ uint<BITS >*. Le paramètre BITS fixe la largeur des données manipulées pour les entrées et les sorties des fonctions.

2. Au-dessus de la couche scalaire se trouve la couche vectorielle. Un ensemble de Z fonctions scalaires est implémenté dans les fonctions SIMD pour traiter les vecteurs de données de Z éléments. Le nombre d'éléments scalaires dans un vecteur est flexible. Contrairement au jeu d'instructions AVX2 qui ne peut traiter que des éléments de 32 bits. Les fonctions vectorielles prennent en charge deux paramètres en template pour faciliter leur utilisation et répondre aux besoins de modélisation. Le premier (QC) spécifie le nombre de calculs à effectuer en parallèle. Le second (BITS) décrit la largeur des données.

Le modèle comportemental complet du décodeur LDPC est construit à partir des fonctions SIMD implantées dans la bibliothèque vectorielle. Une formulation simplifiée du modèle est décrite dans l'Algorithme 2. La flexibilité des couches scalaires et vectorielles fournit un haut niveau de configurabilité à la description comportementale avec des paramètres architecturaux (largeur des accumulateurs VN, largeur des messages, valeurs de saturation) et applicatifs (N, K, M, Z).

Afin de maximiser les performances, des directives supplémentaires telles que *#pragma pipeline* sont incluses. Comme cela est détaillé dans l'Algorithme 2, ces directives permettent de pipeliner les coeurs de boucles et ainsi d'améliorer le débit de décodage. Ces directives de synthèse guident les choix effectués par l'outil HLS lors de la génération de l'architecture de niveau RTL. Leur usage est basé sur les connaissances du concepteur quant aux types et aux caractéristiques de l'architecture à générer. Des directives supplémentaires sont également utilisées pour les aspects liés à la mémorisation. Les tableaux permettant de stocker les messages ($\mathcal{L}$) et les accumulateurs (VN) définis dans l'Algorithme 2 sont déclarés comme des tableaux de données dans la partie dédiée aux attributs privés du module SystemC. Le tableau VN est composé de N/Z éléments de Z×LLLR_BITS bits. LLR_BITS représente le nombre de bits pour stocker un LLR. Les valeurs N, Z, LLR_BITS sont paramétrables dans le modèle comportemental.

Au final, le décodeur LDPC est décrit dans un seul module SystemC contenant un seul processus comme détaillé dans la Figure 2.3. Un fichier d'en-tête permet de spécifier l'ensemble des paramètres de synthèse (N, M, le facteur Z, la matrice $\mathbf{H}$ et les formats de quantification des LLR/MSG). Un outil personnalisé externe analyse la spécification de la matrice $\mathbf{H}$ pour tout code QC-LDPC. Il génère le fichier de paramètres approprié. La description comportementale du décodeur LDPC et des bibliothèques associées est plus synthétique qu'une description RTL faites par un concepteur. Le modéle algorithmique et

Algorithme 2: Algorithme min-sum basé sur ordonnancement par couches horizontales

Kernel 1 : Initialisation
pour tout VN bloc $n = 1 \rightarrow N$ **faire**
  # Pipeline directive
  Charge Z éléments de la trame dans la mémoire accumulatrice des VNs
fin pour
Kernel 2 : Décodage itératif
pour tout Itération $i = 1 \rightarrow (I_{max})$ **faire**
  pour tout ensemble de Z CNs **faire**
Kernel 2.1 : ($\mathcal{P}_1$) Calcul des messages entrants
    pour tout VN contribuant aux CNs sélectionnés **faire**
      # Pipeline directive
      Charge un ensemble de Z LLRs depuis la mémoire
      Entrelace l'ensemble de VN selon la matrice **H**
      Mise à jours des CNs
    fin pour
Kernel 2.2 : ($\mathcal{P}_2$) Calcul des messages sortants
    pour tout VN contribuant aux CNs sélectionnés **faire**
      # Pipeline directive
      Mise à jours des CNs
      Stockage des VNs mis à jour en mémoire Store
    fin pour
  fin pour
fin pour
Kernel 3 : Décision dure
pour tout VN bloc $n = 1 \rightarrow N$ **faire**
  # Pipeline directive
  Entrelace l'ensemble de VN
  Prend une décision dure sur chaque LLR de la trame
  Envoie les bits décodés
fin pour

la bibliothèque associée représentent respectivement environ 200 et 300 lignes de codes.

Le traitement du modèle $\mathcal{M}_1$ par un outil HLS produit l'architecture cible $\mathcal{A}_1$ dont l'organisation est détaillée dans la Figure 2.4. $\mathcal{A}_1$ possède Z bancs mémoires de largeur LLR_BITS stockant chacun N/Z éléments VN et Z bancs mémoires de largeur LLR_MSG stockant chacun m/Z messages. Le processus de décodage (Algorithme 2, Kernel 2.1) est implémenté dans Z unités de traitement. Chaque unité de traitement est divisée en deux parties : $\mathcal{P}_1$ calcule les messages entrants (Algorithme 2, Kernel 2.1) et $\mathcal{P}_2$ calcule les messages sortants (Algorithme 2, Kernel 2.2). Ces deux types d'unités de traitement communiquent entre-elles à l'aide de mémoires partagées. La profondeur des mémoires partagées dépend des caractéristiques de **H** et plus précisément de la valeur maximale de d_c.

L'opération d'entrelacement (Algorithme 2, Kernel 2.1) est décrite à l'aide d'une structure

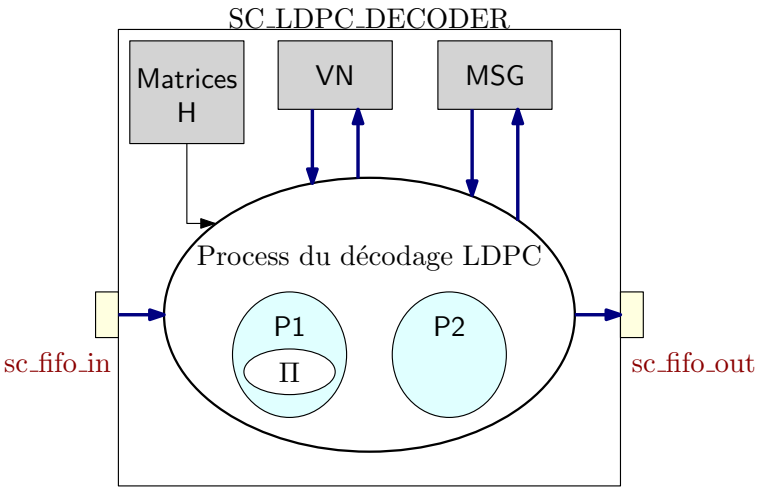

Figure 2.3 – Modèle de décodeur LDPC $\mathcal{M}_1$

conditionnelle de type *switch-case*. Les signaux de commande de l'entrelaceur sont stockés dans une mémoire dont le contenu dépend de $\mathbf{H}$. L'entrelaceur est implémenté matériellement sous la forme d'un barrel shifter (Π). Un second entrelaceur est normalement nécessaire pour l'opération de désentrelacement à la fin du Kernel 2.2 (Algorithme 2) [61]. Cependant, afin de réduire la complexité matérielle, l'optimisation proposée dans [63] a été appliquée. Cette optimisation consiste à ne pas désentrelacer les VNs et les stocker avec un facteur de rotation r. Étant donné que le prochain traitement de noeuds CNs nécessite son propre niveau d'entrelacement des noeuds VNs, une simple rotation approprié des données suffis. Il est ainsi possible de générer une autre matrice $\mathbf{H}_r$ qui contient les facteurs d'entrelacement relatif d'un traitement de CN à un autre. Par conséquent un seul entrelaceur est nécessaire au prix d'un léger surcoût mémoire.

Les performances de débit et de latence des architectures générées dépendent principalement des paramètres de la matrice $\mathbf{H}$ (Z, m). Lors du décodage, les kernels 3.1 et 3.2 sont exécutés séquentiellement car des dépendances de données peuvent exister entre deux itérations successives de la boucle les incluant (Algorithme 2, ligne 8). Un diagramme de Gantt détaillant le séquencement des calculs et des accès mémoires (Algorithme 2, ligne 8) est donné dans la Figure 2.5.

Pendant les $\Delta_{\mathcal{P}_1}$ premiers cycles, les Z premiers CNs sont traités par $\mathcal{P}_1$ (Algorithme 2, Kernel 2.1). La durée $\Delta_{\mathcal{P}_1}$ correspond aux cycles de calcul des CNs plus les cycles de pénalité introduit par le traitement du pipeline (épilogue et prologue). Cette pénalité ($\delta = \delta^i + \delta^f$) intervient lors de l'initialisation du pipeline (δ^i) et de son vidage (δ^f). Dans la figure 2.5, $\delta^i_{\mathcal{P}_1}$ est égal à deux cycles d'horloge. Un cycle provient de la lecture de la mémoire VN. Un second cycle est liée à l'opération d'entrelacement. Cette étape doit

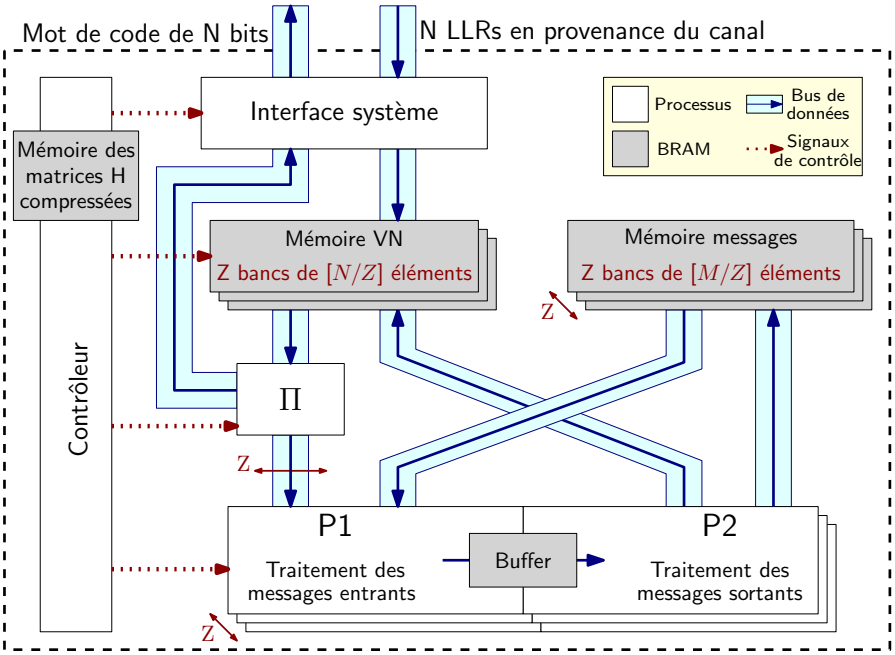

FIGURE 2.4 – Organisation de l'architecture matérielle cible pour le modèle $\mathcal{M}_1$

être exécutée avant le début des calculs d_c. Le stockage des valeurs dans les mémoires partagées implique que $\delta^f_{\mathcal{P}_1} = 1$. A l'opposé, la pénalité de $\mathcal{P}_2$ est plus faible. Un cycle d'horloge est nécessaire pour lire la valeur de la mémoire partagée ($\delta^i_{\mathcal{P}_2}$). Un autre cycle $\delta^f_{\mathcal{P}_2} = 1$ est nécessaire pour stocker les valeurs dans les mémoires VN et MSG. Le nombre total de cycles nécessaire pour les processus $\{\mathcal{P}_1\ \mathcal{P}_2\}$ dépend des pénalités $\{\delta^i\ \delta^f\}$ et de la fréquence d'horloge spécifiée comme contrainte à l'outil de synthèse HLS. La latence de calcul $\Delta(m)$ en nombre de cycles d'horloge pour la $m^{\text{ème}}$ itération de décodage associé à un ensemble de CN (Algorithme 2, Ligne 8) peut s'exprimer par l'équation 2.6 avec $d_c(m)$ le degré du $m^{\text{ème}}$ ensemble de noeuds CN.

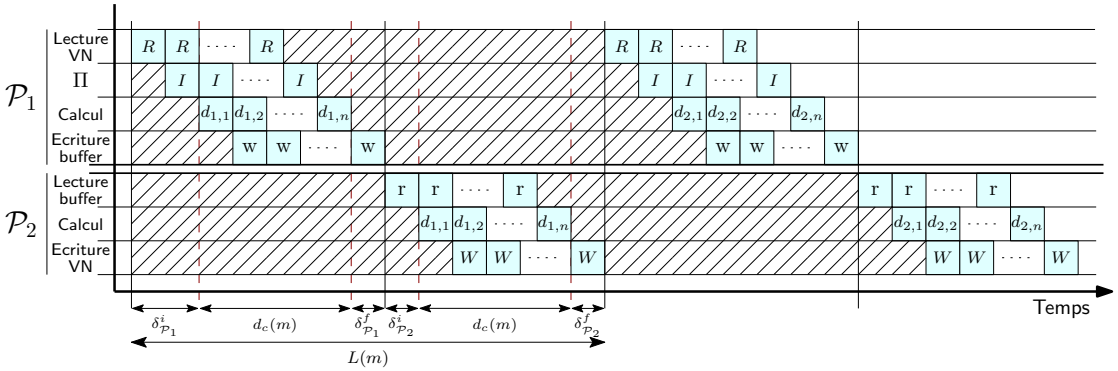

FIGURE 2.5 – $\mathcal{M}_1$: Diagramme de Gant portant sur les opérations des unités de calcul $\mathcal{P}_1$ et $\mathcal{P}_2$

$$\Delta(m) = (\delta^{i}_{\mathcal{P}_1} + d_c(m) + \delta^{f}_{\mathcal{P}_1}) + (\delta^{f}_{\mathcal{P}_2} + d_c(m) + \delta^{f}_{\mathcal{P}_2})$$
$$\Delta(m) = 2 \times d_c(m) + (\delta_{\mathcal{P}_1} + \delta_{\mathcal{P}_2}) \tag{2.6}$$

Les pénalités $\delta_{\mathcal{P}_1}$ et $\delta_{\mathcal{P}_2}$ ont un impact négatif plus important lors de l'implémentation de décodeurs de code LDPC à faible rendement. En effet, leurs valeurs $d_c(m)$ sont faibles (≤ 3). Pour des décodeurs LDPC à rendement élevé dont les valeurs $d_c(m)$ peuvent atteindre des valeurs supérieures à 32 l'impact est moins critique. La latence théorique $\Delta_{\mathcal{M}_1}$ en nombre de cycles d'horloge de $\mathcal{M}_1$ peut être formulé comme suit :

$$\Delta_{\mathcal{M}_1} = I_{max} \times \left[\sum_{m=1}^{M/Z} \Delta(m) \right] + 2 \times \frac{N}{Z} \tag{2.7}$$

Dans l'équation 2.7, $\frac{2 \times N}{Z}$ est le nombre de cycles d'horloge nécessaires pour charger/décharger les données du/vers le système. Les deux tâches nécessitent $\frac{N}{Z}$ cycles d'horloge car Z données sont lues ou transmises durant chaque cycle d'horloge [3].

2.2.1.2 Limites du modèle

Les processus de traitement $\mathcal{P}_1$ et $\mathcal{P}_2$ sont finement dimensionnés et aident le modèle $\mathcal{M}_1$ à aboutir à des architectures efficaces au niveau de la complexité matérielle comme le montreront les résultats de la section expérimentation. Certaines limitations existent cependant. En effet, les unités de traitement constituant $\mathcal{P}_1$ et $\mathcal{P}_2$ sont sous-utilisées en raison de la séquentialité des Kernel 2.1 et 2.2 dans l'Algorithme 2. Cette limitation est liée :

1. à la description comportementale car il est impossible de spécifier l'exécution parallèle de $\mathcal{P}_1$ et $\mathcal{P}_2$ au sein d'un seul et unique processus.
2. Il y a un problème de dépendance de données entre les ensemble de CN et de VN. Cela signifie qu'un ré-agencement de la matrice **H** capable de fournir un VN indépendant d'une ligne à l'autre doit être appliqué. Malheureusement, ceci n'est pas toujours possible pour chaque code QC-LDPC.

Dans la section suivante, nous détaillons un second modèle permettant de paralléliser l'exécution de $\mathcal{P}_1$ et $\mathcal{P}_2$, ce qui permet de gagner un facteur 2 à 3 en terme d'efficacité.

3. Les deux tâches peuvent être effectuées en parallèle pour économiser $\frac{N}{Z}$ cycles d'horloge.

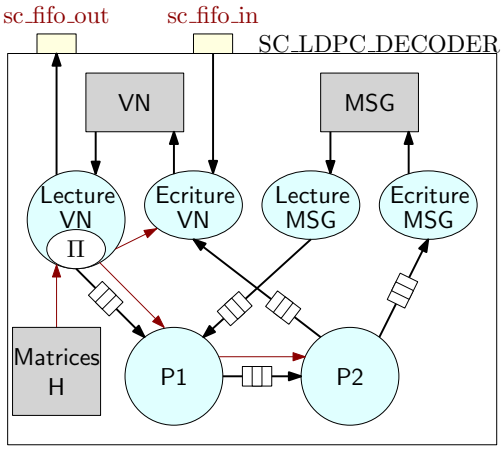

Figure 2.6 – Représentation du modèle $\mathcal{M}_2$ de décodeur LDPC

2.2.2 Description du modèle $\mathcal{M}_2$

Un second modèle de décodeur LDPC $\mathcal{M}_2$ a été mis au point pour tenter de résoudre les limitations du modèle $\mathcal{M}_1$. Le taux d'utilisation des unités de traitement sont améliorés par : 1) l'exécution de $\mathcal{P}_1$ et $\mathcal{P}_2$ en parallèle et 2) la suppression autant que possible des pénalités d'initialisation et de sortie du pipeline. Le modèle $\mathcal{M}_2$ est un modèle comportemental spécialement écrit pour prendre en compte les spécificités de la HLS. Contrairement au premier modèle $\mathcal{M}_1$, il décrit explicitement les tâches parallèles. Six processus SystemC distincts fonctionnent de manière concurrente. Ils sont synchronisés à l'aide de FIFOs. La description de l'algorithme min-sum suivant ce modèle est fourni dans l'Algorithme 3. L'organisation des processus (Figure 2.6) est détaillée ci-après. Ces processus sont exécutés simultanément sous réserve que les données soient disponibles.

— Le premier processus "Lecture VN" lit et entrelace les données VN. Les données VN une fois entrelacées sont fournies à $\mathcal{P}_1$. De plus, ce processus est aussi en charge d'envoyer la trame décodée au système à la fin du processus itératif. Enfin, ce processus maître est le seul à avoir accès aux données relatives à la matrice **H**. Il propage donc les signaux de commande aux autres processus.

— Le second processus "Ecriture VN" stocke les données dans les mémoires associées aux VN. Il récupère les blocs VN mis à jour à la sortie de $\mathcal{P}_2$ et les stocke dans la mémoire VN. Ce processus stocke également les trames arrivant à l'entrée du décodeur avant de commencer les itérations de décodage.

— Le troisième processus "Lecture MSG" lit linéairement les blocs de messages de la mémoire des MSG et les fournit à $\mathcal{P}_1$.

— Le quatrième processus "Ecriture MSG" reçoit les messages en sortie de $\mathcal{P}_2$ et les

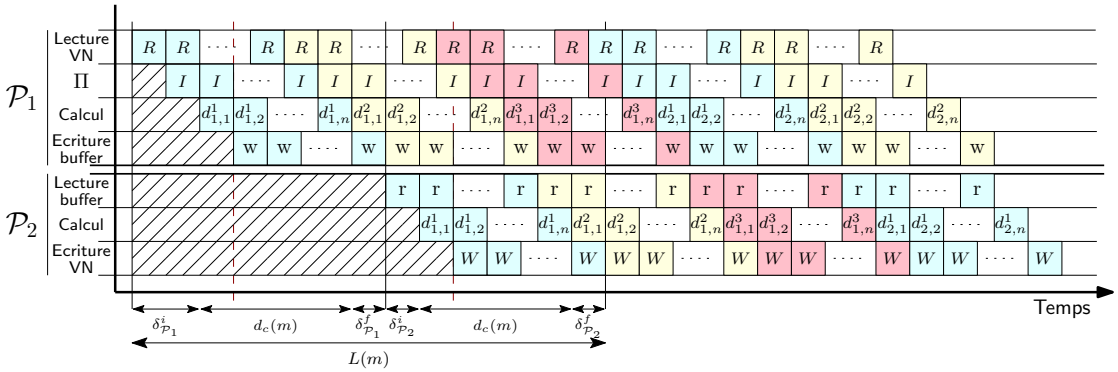

FIGURE 2.7 – $\mathcal{M}_2$: diagramme de Gantt pour les opérations des unités de calcul $\mathcal{P}_1$ et $\mathcal{P}_2$

stocke dans la mémoire des MSG.

— Le cinquième processus $\mathcal{P}_1$ calcule les valeurs internes du CN en fonction des messages entrants et des valeurs VN.

— Le sixième processus $\mathcal{P}_2$ calcule les messages sortants et met à jour les blocs VN en fonction des valeurs CN internes au processus $\mathcal{P}_1$.

Les processus $\mathcal{P}_1$ et $\mathcal{P}_2$ peuvent, en théorie, fonctionner simultanément. Toutefois, il convient de résoudre les contraintes qui résultent des dépendances de données. Une solution générique et flexible consiste à ajouter un parallélisme inter-trame en plus du parallélisme intra-trame [51, 64]. Idéalement, deux trames distinctes devraient être suffisantes pour éviter les conflits liés aux dépendance de données entre $\mathcal{P}_1$ et $\mathcal{P}_2$. Le processus $\mathcal{P}_1$ réaliserait des calculs associés à une première trame Tr1 tandis que $\mathcal{P}_2$ traiterait les données d'une seconde trame Tr2. Malheureusement, le processus "Lecture VN" pourrait avoir accès à des valeurs de VN qui ne sont pas encore mises à jour par "Ecriture VN". En effet, dans le modèle architectural, les unités de calcul $\mathcal{P}_1$ et $\mathcal{P}_2$ ont des chemins de données pipelinés. Pour résoudre ce problème, trois trames doivent être décodées en parallèle.

Le traitement de ces trois trames se fait de manière entrelacée durant le décodage. Lorsque $\mathcal{P}_2$ calcule $CN_{1;i}$ de la trame Tr1, $\mathcal{P}_1$ calcule $CN_{2;i}$ de la trame Tr2 et les VNs de la trame Tr3 sont stockés en mémoire. Un diagramme de Gantt explicitant ce comportement est détaillé en Figure 2.7. Cette solution nécessite de charger trois trames avant de démarrer le processus de décodage. Par conséquent, la latence de décodage d'une trame est augmentée et la taille des mémoires RAM pour les VN et les MSG est multipliée par trois. Ce coût mémoire est nécessaire afin de maximiser le taux d'occupation des unités de calcul.

Algorithme 3: Algorithme min-sum avec un ordonnancement par couches horizontales

PROCESSUS 1 : Lecture VN

 # Pipeline directive
pour tout itérations de décodage **faire**
  Lit VN depuis la mémoire
  Entrelace les blocs de VN selon la matrice **H**
  Fourni les VN au processus $\mathcal{P}_1$
fin pour
Envoie la trame décodée

PROCESSUS 2 : Ecriture VN

Charge une trame dans les mémoires accumulatrices de VN
pour tout itérations de décodage **faire**
  # Pipeline directive
  Récupère les VN depuis le processus $\mathcal{P}_2$
  Écrit les VN en mémoire
fin pour

PROCESSUS 3 : Lecture MSG

pour tout itérations de décodage **faire**
  # Pipeline directive
  Lit les MSG depuis la mémoire
  Fourni les MSG au processus $\mathcal{P}_1$
fin pour

PROCESSUS 4 : Ecriture MSG

pour tout itérations de décodage **faire**
  # Pipeline directive
  Récupère les MSG au processus $\mathcal{P}_2$
  Ecrit les MSG en mémoire
fin pour

PROCESSUS 5 : $\mathcal{P}_1$

pour tout Itération $i = 1 \rightarrow (I_{max})$ **faire**
  pour tout Ensemble de Z CN **faire**
    pour tout VN contribuant au CNs sélectionnés **faire**
      # Pipeline directive
      Calcule les messages entrants
    fin pour
  fin pour
fin pour

PROCESSUS 6 : $\mathcal{P}_2$

pour tout Itération $i = 1 \rightarrow (I_{max})$ **faire**
  pour tout Ensemble de Z CN **faire**
    pour tout VN contribuant au CNs sélectionnés **faire**
      # Pipeline directive
      Calcule les messages sortants
    fin pour
  fin pour
fin pour

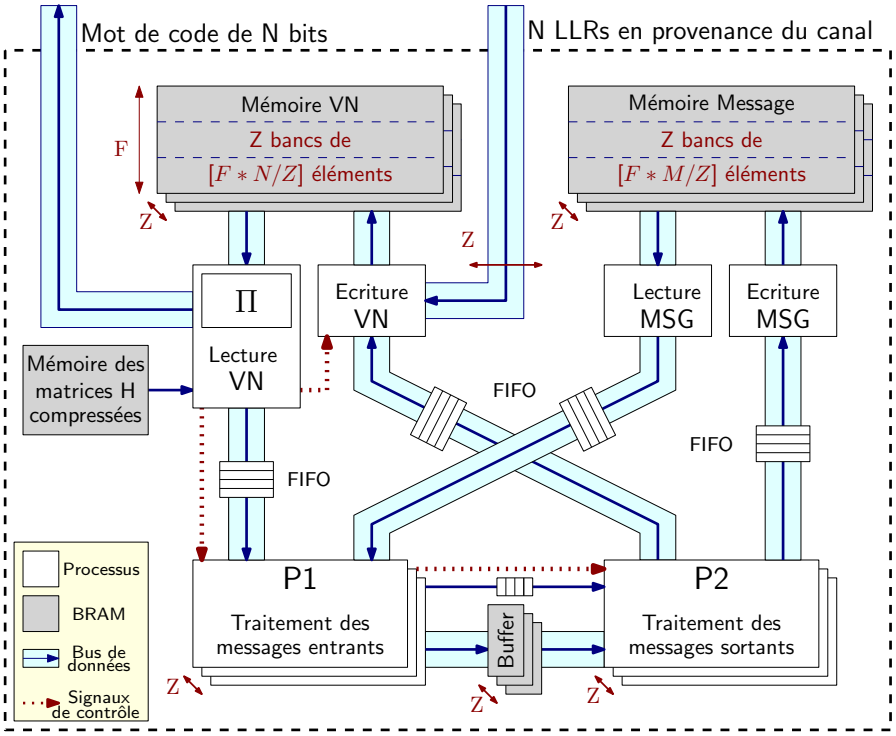

Figure 2.8 – Organisation de l'architecture matérielle pour l'implémentation de décodeur avec le modèle $\mathcal{M}_2$

Le modèle $\mathcal{M}_2$ produit l'architecture cible $\mathcal{A}_2$ présentée dans la Figure 2.8. Il y a plusieurs différences par rapport à l'architecture cible $\mathcal{A}_1$. Au lieu d'un contrôleur unique, plusieurs contrôleurs distribués de faible complexité sont associés à chaque sous-module matériel indépendant (un pour chaque processus comportemental SystemC). Dans l'architecture $\mathcal{A}_2$, les unités de calcul $\mathcal{P}_1$ et $\mathcal{P}_2$ sont également indépendantes. Elles peuvent fonctionner simultanément. Tous les sous-modules communiquent au travers de FIFOs implémentées sous la forme de mémoires distribuées ou de blocs RAM en fonction de leur largeur et de leur profondeur. Cela engendre un coût matériel supplémentaire mais permet une exécution parallèle du processus de décodage. Ainsi, le débit de décodage est multiplié par un facteur supérieur à 2 par rapport à celui de l'architecture cible $\mathcal{A}_1$. Il est à noter que pour le modèle comportemental $\mathcal{M}_2$, le surcoût mémoire de $\mathcal{A}_2$ est souvent inférieur à trois fois le coût mémoire de $\mathcal{A}_1$ pour une implémentation sur cible FPGA. Il peut même dans certaines circonstances être nul comme illustré dans la section 2.3. En effet, du fait de la taille fixe des BRAM sur cible FPGA, il est parfois possible de stocker 3 trames dans une BRAM. La latence théorique $\Delta_{\mathcal{M}_2}$ en nombre de cycles d'horloge obtenue pour

l'architecture cible $\mathcal{A}_2$ est estimée comme suit :

$$\Delta_{\mathcal{M}_2} = I_{max} \times \left[\sum_{m=1}^{F \times M} d_c(m) \right] + \delta_{\mathcal{P}_1} + \delta_{\mathcal{P}_2} + 2F\frac{N}{Z} \tag{2.8}$$

Où F est le nombre de trames fixé pour le niveau de parallélisme inter-trames. Par rapport à $\Delta_{\mathcal{M}_1}$, les pénalités d'initialisation et de sortie du pipeline n'apparaissent qu'une seule fois pendant tout le processus de décodage des F trames. Le modèle $\mathcal{M}_2$ a été conçu de manière à utiliser un pipeline architectural le plus efficacement possible.

2.2.3 Conclusion

Deux modèles comportementaux permettant la génération de décodeurs LDPC à l'aide d'un outil HLS ont été présentés. L'accent a été mis sur la génération d'architectures efficaces avec de faibles latences de décodage et une maîtrise de la complexité matérielle. Cette approche basée sur un modèle comportemental apporte de la flexibilité à la conception et à l'exécution. Les deux modèles sont configurables et supportent des codes LDPC avec une structure QC. Les largeurs de données pour les variables et les messages sont configurables. De nombreuses expérimentations ont été menées pour évaluer la pertinence de ces travaux et évaluer le coût réel de la flexibilité. Dans la section suivante, nous présentons les résultats obtenus et les comparons à ceux de la littérature.

2.3 Résultats expérimentaux

2.3.1 Performances absolues

Les performances absolues des modèles comportementaux $\mathcal{M}_1$ et $\mathcal{M}_2$ sont d'abord évaluées dans cette première partie. Pour comprendre les avantages de l'approche par rapport aux architectures existantes, des codes LDPC de standard 802.16e, 802.11n et 3GPP 5G ont été retenus.

L'outil Vivado HLS 2017.1 de chez Xilinx a été sélectionné pour générer des architectures RTL à partir des modèles comportementaux. Le circuit ciblé est un FPGA Xilinx Virtex-7 xc7vx485tffg1761-2. Deux contraintes de fréquence d'horloge (100 MHz et 300 MHz) ont été appliquées pendant la synthèse HLS afin d'aboutir à différents compromis architecturaux. Le temps de synthèse HLS varie de quelques minutes à une heure sur un ordinateur

possédant 8 coeurs logiques et 16 Go de mémoire vive.

Les résultats expérimentaux au niveau de la complexité matérielle, du débit (Γ), de la latence en μs et de la latence (Δ) en nombre de cycles d'horloge sont présentés dans le tableau 2.2 pour le modèle $\mathcal{M}_1$ et dans le tableau 2.3 pour le modèle $\mathcal{M}_2$. Toutes les valeurs données sont des résultats obtenus après placement routage (PAR).

Performances du modèle $\mathcal{M}_1$

Les débits de décodage Γ indiqués dans la partie gauche du tableau 2.2 pour le modèle $\mathcal{M}_1$ varient de 29 à 227 Mbps lorsque la contrainte de synthèse HLS est de 100 MHz. Lorsque la fréquence cible est constante (ex : 100 MHz) et le rendement fixe (ex : R = 1/2), le débit de décodage (Γ) augmente en fonction du facteur d'expansion Z choisi pour construire la matrice $\mathbf{H}$. Nous pouvons remarquer que la fréquence de fonctionnement F_{max} du décodeur diminue lorsque le facteur Z augmente. Cela s'explique par l'implémentation de l'entrelaceur (barrel shifter) qui voit sa complexité augmenter de manière exponentielle avec Z. Par conséquent la durée du chemin critique augmente pour l'entrelaceur pendant la phase de placement routage. La complexité matérielle de l'architecture évolue également de façon linéaire avec Z. Si le paramètre Z est considéré comme constant, le rendement et les valeurs d_c ont une influence significative sur le débit de décodage. Le modèle $\mathcal{M}_1$ est fortement impacté par les pénalités d'initialisation et de sortie du pipeline. Nous pouvons remarquer l'impact des valeurs d_c pour le standard 802.11 avec $N = 648$. Le débit codé du décodeur avec un rendement $R = \frac{5}{6}$ est supérieur de 30% par rapport au débit du décodeur avec un rendement $R = \frac{1}{2}$, et ce alors que le nombre de message des 2 codes LDPC est similaire. La différence de débit s'explique par le plus grand nombre de pénalités introduites par le pipeline dans le cas d'un code de rendement $R = \frac{1}{2}$. En effet, les valeurs de d_c sont plus faibles tandis que dans le même temps, le nombre de CN, est lui plus élevé.

Les résultats obtenus lors de la génération d'une architecture sous contrainte d'une fréquence de fonctionnement de 300 MHz sont présentés dans la partie droite du tableau 2.2. Nous constatons que le débit et la latence de décodage sont plus favorable dans certains cas. Des débits de décodage variant de 41 à 257 Mbps sont atteints. Cependant, nous observons également que dans la majorité des cas, l'augmentation de la contrainte de fréquence améliore peu ou pas le débit final de l'architecture. En effet, pouvoir travailler à 300 MHz, les unités de traitement du CN ($\mathcal{P}_1$ et $\mathcal{P}_2$) ont des profondeurs de pipeline plus grande, ce qui engendre des pénalités plus importantes. De plus, l'architecture de l'entrelaceur (barrel shiter) limite la fréquence maximale de l'horloge en deçà de 300 MHz lorsque $Z > 32$. Pour l'ensemble des expérimentation donné, les débits de décodage

Tableau 2.2 – Performances des décodeurs LDPC-QC avec parallélisme intra-trame généré à partir du modèle $\mathcal{M}_1$ sur FPGA Virtex-7 (10 itérations de décodages)

	Code LDPC			Fréquence cible = 100 MHz								Fréquence cible = 300 MHz							
	N	R	Z	F_{max}	Δ	Γ	μs	LUTs	FFs	Slices	RAM	F_{max}	Δ	Γ	μs	LUTs	FFs	Slices	RAM
802.11n	648	1/2	27	137	3097	29	23	3358	917	1018	22	351	5500	41	16	4142	5805	1647	22
	648	5/6	27	130	2257	37	17	3395	914	1032	22	343	3060	73	9	4263	5813	1620	22
	1296	1/2	54	116	3055	49	26	7156	1634	2114	43	245	5458	58	22	8527	11303	3579	43
	1296	5/6	54	114	2194	67	19	7051	1596	2104	45	268	2997	116	11	8397	11275	3325	45
	1944	1/2	81	108	3055	69	28	11663	2366	3451	65	237	5458	84	23	13759	17081	5432	65
	1944	5/6	81	106	2068	100	19	11583	2303	3419	68	218	2871	147	13	13333	16749	5158	68
802.16e	1152	1/2	48	120	2845	48	24	5815	1469	1744	39	304	5248	67	17	6805	10069	2725	39
	1920	1/2	80	101	2845	68	28	11497	2335	3401	65	208	5248	76	25	13365	16590	5401	65
	2304	1/2	96	106	2845	86	27	13892	2774	4114	77	223	5248	98	24	15897	19850	6048	77
5G 3GPP	1024	3/4	32	130	3627	37	28	3771	1059	1159	29	356	5630	65	16	4450	6846	1819	29
	2048	3/4	64	113	3627	64	32	7568	1864	2275	55	251	5630	91	22	8913	13314	3523	55
	4096	3/4	128	104	3627	118	35	15922	3543	4866	107	230	5630	167	25	18178	26309	7145	107
	8192	3/4	256	101	3627	227	36	34823	6909	10803	209	177	5630	257	32	39313	52182	16703	209
	2376	8/9	88	107	2214	115	21	13181	2474	3942	74	219	3217	162	15	15263	18802	5477	74
	4048	1/2	88	108	6903	63	64	13439	2498	4002	75	174	11706	60	67	15905	18170	6570	75
	4752	8/9	176	102	2214	219	22	29992	4804	8750	146	159	3217	235	20	33213	35929	13424	146
	8096	1/2	176	101	6903	118	69	30478	4812	8920	147	168	11706	116	70	32589	35951	13025	147

évoluent de -2% à $+90\%$ lorsque la fréquence de fonctionnement passe de 100 MHz à 300 MHz. Une analyse plus approfondie montre qu'en parallèle du débit, une augmentation de 10 à 20% de la complexité matérielle est observée. Le coût au niveau des bascules Flip Flop (FF) est multiplié par un facteur variant de 6 à 8.

Le nombre théorique de cycles d'horloge nécessaire pour une exécution complète du processus de décodage pour le modèle $\mathcal{M}_1$ est comparé dans la Figure 2.9 à des estimations obtenues après la synthèse d'architecture. Les valeurs théoriques ont été calculées en considérant une durée d'un cycle d'horloge par itération des boucles des Kernels 2.1 et 2.2 dans l'Algorithme 2. Cette hypothèse est optimiste car elle omet les pénalités liées au pipeline ($\delta_{\mathcal{P}_1} = 0$, $\delta_{\mathcal{P}_2} = 0$). Un facteur de valeur 1,5 entre le modèle théorique et les estimations pratiques est observé dans la Figure 2.9 lorsque nous considérons les architectures générées avec une contrainte d'horloge de 100 MHz. Pour les architectures produites lorsque la contrainte de fonctionnement est fixée à 300 MHz, le différentiel entre performances théoriques et estimations est plus élevé.

Cet écart significatif illustre l'impact de l'augmentation de la fréquence de fonctionnement sur le nombre de tranches de pipeline dans les unités de traitement associées à $\mathcal{P}_1$ et $\mathcal{P}_2$ et par conséquent sur la latence de l'architecture de décodage. Cette analyse explique les résultats au niveau du débit et de la latence présentés dans le Tableau 2.2.

En résumé, il est possible de concevoir des décodeurs LDPC plus ou moins efficaces à l'aide du modèle comportemental $\mathcal{M}_1$. Différents compromis sont possibles en fonction

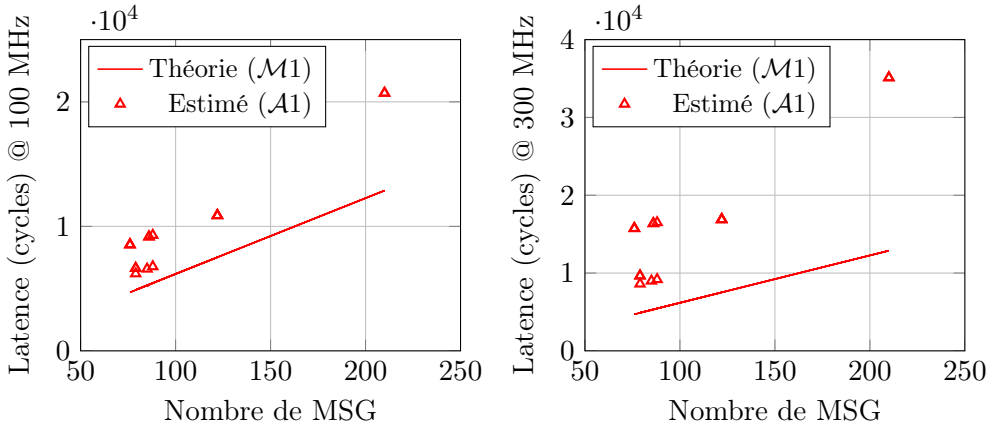

Figure 2.9 – Latence exprimée en nombre de cycles d'horloge pour le modèle $\mathcal{M}1$ selon deux contraintes de fréquence

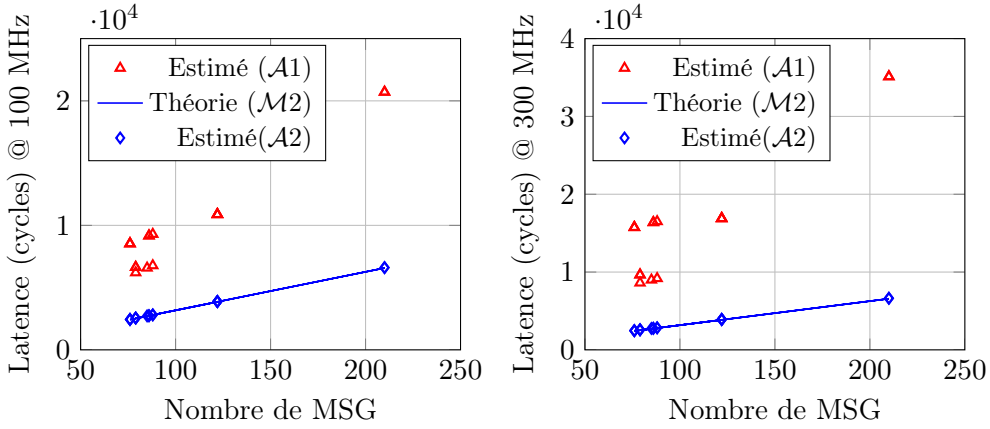

Figure 2.10 – Latence exprimée en nombre de cycles d'horloge pour le modèle $\mathcal{M}2$ selon deux contraintes de fréquence

des contraintes applicatives et des contraintes de synthèse HLS. Toutefois, d'une part l'exécution séquentielle de $\mathcal{P}_1$ et de $\mathcal{P}_2$ et d'autre part les pénalités liées à un traitement pipeliné impactent de manière significative les performances.

Performances du modèle $\mathcal{M}_2$

Le modèle de décodeur LDPC $\mathcal{M}_2$ favorise une forte augmentation de l'efficacité des décodeurs LDPC produits au niveau de la latence et du débit au prix d'une complexité matérielle accrue. Les résultats relatifs au modèle $\mathcal{M}_2$ sont récapitulés dans dans le Tableau 2.3. Au cours des expérimentations, le niveau de parallélisme inter-trames est fixé à 3 trames. En effet, cela permet de saturer les unités de calcul durant l'exécution du processus de décodage. Les décodeurs LDPC basés sur le modèle $\mathcal{M}_2$ atteignent des débits

variant de 88 Mbps à 567 Mbps lorsqu'une contrainte de fréquence de fonctionnement de 100 MHz est spécifié lors de la synthèse HLS. Une amélioration substantielle du débit de décodage est observée lorsque la synthèse HLS est contrainte à 300 MHz. En effet, étant donné que les pénalités liées au pipeline ont été supprimées, le débit évolue linéairement avec la fréquence de fonctionnement. Même si toutes les architectures de décodage n'atteignent pas la fréquence de fonctionnement de 300 MHz après placement routage, les fréquences atteintes sont en moyenne 1,9× plus élevées. Le modèle $\mathcal{M}_2$ résout donc les problèmes identifiés dans le modèle $\mathcal{M}_1$ (les pénalités de pipeline + la séquentialité de $\mathcal{P}_1/\mathcal{P}_2$). En moyenne le débit des architectures $\mathcal{A}_2$ est 1,8 fois supérieur à celui des architectures $\mathcal{A}_1$ lorsque la fréquence de fonctionnement est contrainte à 300 MHz. Ces débits varient de 161 Mbps à 1010 Mbps.

La complexité matérielle des décodeurs LDPC dépend quasi-éclusivement du facteur Z. Elle varie de 2036 à 18635 slices par décodeur pour des architectures limitées à 100 MHz. Ces architectures occupent de 2% à 17% du circuit FPGA Virtex-7 retenu. Le nombre de slices consommées par les architectures produites peut être approximé par $78 \times Z$ lorsque $F = 100$ MHz. L'augmentation de la fréquence de fonctionnement ($F = 300$ MHz) implique en moyenne une augmentation de 35% des besoins au niveau des slices. Cette augmentation s'explique par l'utilisation massive de bascules FF afin de garantir des chemins critiques conforment à la contrainte de fréquence. En effet, leur nombre est multiplié en moyenne par 2,5 tandis que l'augmentation des LUTs est "marginal" (+10%).

Tableau 2.3 – Performances des décodeurs LDPC-QC en parallélisation intra-trame générés à partir du modèle $\mathcal{M}_2$ sur FPGA Virtex-7 (10 itérations de décodage)

	Code LDPC			Fréquence cible = 100 MHz								Fréquence cible = 300 MHz							
	N	R	Z	F_{max}	Δ	Γ	μs	LUTs	FFs	Slices	RAM	F_{max}	Δ	Γ	μs	LUTs	FFs	Slices	RAM
802.11n	648	1/2	27	134	2806	92	21	5868	4934	2036	19	237	2831	163	12	6777	12449	2811	19
	648	5/6	27	127	2820	88	22	5911	4938	2077	19	236	2845	161	12	6836	12452	2831	19
	1296	1/2	54	115	2746	163	24	12483	9769	4319	35	230	2771	323	12	13083	24293	5681	35
	1296	5/6	54	105	2730	149	26	12514	9773	4297	35	224	2755	316	12	13161	24318	5786	35
	1944	1/2	81	113	2746	241	24	19724	14453	6535	52	216	2771	455	13	20902	36140	8694	52
	1944	5/6	81	101	2548	231	25	19749	14459	6571	52	219	2573	496	12	20932	36152	8853	52
802.16e	1152	1/2	48	103	2445	146	24	10857	8728	3640	32	226	2470	317	11	10897	21658	4833	32
	1920	1/2	80	105	2445	248	23	19146	14270	6340	52	209	2470	488	12	20664	35721	8533	52
	2304	1/2	96	113	2445	321	22	22816	17168	7573	62	211	2470	591	12	24798	42735	10757	62
5G 3GPP	1024	3/4	32	129	3885	102	30	6737	5790	2366	24	230	3910	181	17	7719	14673	3273	24
	2048	3/4	64	116	3885	184	33	13682	11523	4728	43	208	3910	328	19	14402	28722	6637	43
	4096	3/4	128	112	3885	353	35	27383	22782	9387	82	195	3910	614	20	30292	56664	12547	82
	8192	3/4	256	103	3885	650	38	55092	45175	18635	159	129	3910	812	30	64535	112557	26655	159
	2376	8/9	88	107	2565	297	24	21280	15671	7062	57	219	2590	602	12	23063	39260	9444	57
	4048	1/2	88	101	6609	186	65	21315	15707	7175	72	223	6634	409	30	23611	39276	9389	72
	4752	8/9	176	102	2565	567	25	43514	31288	14536	112	183	2590	1010	14	48775	77639	18984	112
	8096	1/2	176	102	6609	375	65	43514	31288	14536	112	183	6634	672	36	48775	77639	18984	112

La Figure 2.10 compare la latence théorique du modèle comportemental $\mathcal{M}_2$ avec des

estimations réalisées après la synthèse d'architecture. Ces données considèrent une exécution complète du processus de décodage. Les performances théoriques du modèle $\mathcal{M}_2$ ont été calculées en considérant (1) une exécution parallèle de $\mathcal{P}_1$ et $\mathcal{P}_2$ et (2) aucune pénalité liée à un traitement de type pipeline. Lorsque les architectures issues du modèle $\mathcal{M}_2$ ont été produites sous une contrainte de fonctionnement à 100 MHz, nous observons que les latences de décodage théoriques et estimées sont similaires. Contrairement à ce qui a été observé précédemment pour le modèle $\mathcal{M}_1$. Cette similitude entre les latences théoriques et estimées est aussi observée pour les architectures générées en considérant $F = 300$ MHz.

L'ensemble de ces résultats démontre la pertinence du modèle $\mathcal{M}_2$. Il contraint efficacement le processus de génération HLS à partir d'une description explicite des niveaux de parallélisme de l'algorithme de décodage.

Comparaison des performances des deux modèles comportementaux

Comparées aux architectures $\mathcal{A}_1$ générées à partir du modèle $\mathcal{M}_1$, les architectures $\mathcal{A}_2$ générées à partir du modèle $\mathcal{M}_2$ à 100 MHz atteignent un débit 3 fois supérieur. Le modèle $\mathcal{M}_2$ exécute en parallèle les calculs de $\mathcal{P}_1$ et de $\mathcal{P}_2$. Ainsi, il favorise le doublement du débit. Le différentiel de performance entre $\mathcal{M}_1$ et $\mathcal{M}_2$ provient aussi de la capacité du modèle $\mathcal{M}_2$ à supprimer les pénalités liées aux traitements pipelinés ($\mathcal{P}_1$ et $\mathcal{P}_2$). En effet, les cycles perdus durant l'initialisation et le flush du pipeline dans les architectures $\mathcal{A}_1$ sont exploités dans les architectures $\mathcal{A}_2$ pour décoder une $3^{\text{ème}}$ trame. Les gains observés entre les architectures $\mathcal{A}_1$ (300 MHz) et $\mathcal{A}_2$ (300 MHz) sont encore plus significatifs. En effet, les architectures $\mathcal{A}_2$ (300MHz) peuvent atteindre des débits pouvant être jusqu'à 5 fois supérieurs à ceux des architectures $\mathcal{A}_1$ (300 MHz).

Les architectures $\mathcal{A}_1$ (100 MHz) et $\mathcal{A}_2$ (100 MHz) ont des valeurs de latence équivalentes. Ces résultats sont à pondérer par le fait que l'architecture $\mathcal{A}_2$ (100 MHz) décode 3 trames en parallèle tandis que $\mathcal{A}_1$ (100 MHz) n'en décode qu'une seule. A l'opposé, les architectures $\mathcal{A}_2$ (300 MHz) qui bénéficient de l'augmentation de fréquence peuvent réduire les latences de traitement de tous les décodeurs avec des gains pouvant atteindre 56% (25% en moyenne). De manière similaire, si nous considérons les complexités matérielles (SLICE, BRAM), les architectures $\mathcal{A}_1$ et $\mathcal{A}_2$ ont des complexité équivalentes. Le stockage de 3 trames dans l'architecture $\mathcal{A}_2$ est possible sur circuit FPGA sans augmenter le nombre de blocs RAM. Dans les architectures $\mathcal{A}_1$, le nombre de blocs RAM qui sert à stocker les LLRS et les MSG est défini par la largeur des données à stocker ($Z \times LLR\_BITS$) et non par le nombre de données à stocker. Ces mémoires sous-utilisées n'ont pas besoin d'être étendues pour stocker 3 trames. Certaines des données supplémentaires à mémoriser entre

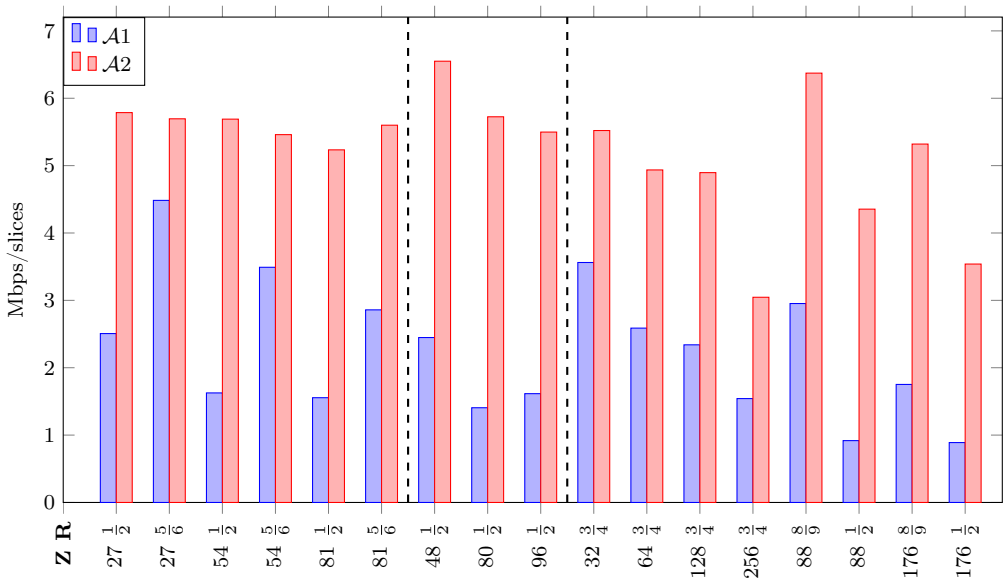

Figure 2.11 – Efficacité matérielle des architectures $\mathcal{A}_1$ et $\mathcal{A}_2$ en Mbps/Slices en fonction du code LDPC étudié (**Z**, **R**) @ 300MHz

les unités de traitement $\mathcal{P}_1$ et $\mathcal{P}_2$ peuvent quant à elles être implémentées en mémoire distribuée (slices) au lieu d'assigner des blocs RAM. Cela résulte d'une modification dans le modèle comportemental où les données sont définies dans un tableau de données dans le modèle $\mathcal{M}_1$ et par une FIFO explicite (*sc_fifo*) dans le modèle $\mathcal{M}_2$. Les architectures $\mathcal{A}_2$ (100 MHz) nécessitent en moyenne 20% de blocs RAM en moins que les architectures $\mathcal{A}_1$ (100MHz). En revanche elles consomment jusqu'à 90% de slices supplémentaires.

En conclusion, le modèle $\mathcal{M}_1$ est plus facile à décrire et à faire évoluer. Il minimise les coûts de mise en œuvre. Toutefois, au niveau de l'efficacité (débit, latence, etc.) le modèle $\mathcal{M}_1$ est moins efficace que le modèle $\mathcal{M}_2$. Ceci peut être affirmé à partir de l'analyse des efficacités des architectures $\mathcal{A}_1$ (300 MHz) et $\mathcal{A}_2$ (300 MHz) comme le montre la Figure 2.11.

2.3.2 Comparaison avec les travaux de la littérature

Afin d'évaluer la pertinence de notre approche, les implémentations de décodeurs LDPC sont comparées avec des résultats de l'état de l'art. Les solutions proposées sont comparées successivement à architectures générées par des outils HLS, à des architectures dédiée décrite au niveau RTL et à des implémentations logicielles sur architectures programmables. Des résultats de comparaisons sont récapitulés dans les tableaux 2.5 et Tableau 2.6. Toutefois, il est important de souligner que les comparaisons sont difficiles à effectuer. Par exemple, les travaux antérieurs utilisent des formats de quantification qui ne sont

Tableau 2.4 – Comparaison de nos travaux avec des implémentations sur FPGA utilisant des méthodologies de conception HLS

Code LDPC		Travaux de l'état de l'art						Approche proposée				
N	R	FPGA	Iters	Γ	μs	Logic[1]	RAM[2]	Iters	Γ	μs	Logic[1]	RAM[2]
768	1/2	Stratix-5 [45]	10○	99	–	147000	1641	5●	454	5	7057	21
1152	1/2	Stratix-5 [45]	10○	104	–	161000	1703	5●	682	5	14851	30
1920	1/2	Stratix-5 [45]	10○	81	–	144000	1516	5●	935	6	24385	45
648	1/2	Spartan-6 [46]	3○	27	24	9072	28	3●	142	14	9148	18
1944	1/2	Stratix-5 [48]	10○	21	–	187000	1349	5●	844	7	24710	48

[1] : LUTs pour FPGA Xilinx, ALMs pour FPGA Altera
[2] : BRAM 18K pour Xilinx, BRAM 20K pour Altera
○ Ordonnancement par inondation ● Ordonnancement par couches

Tableau 2.5 – Comparaison de nos travaux avec des implémentations sur FPGA de décodeurs LDPC décrits au niveau RTL

Code LDPC		Travaux de l'état de l'art						Approche proposée				
N	R	FPGA	Iters	Γ	μs	Logic[1]	RAM[2]	Iters	Γ	μs	Logic[1]	RAM[2]
672	3/4	Virtex-6 [65]	10○	3360	–	56832	⊘	5●	127	16	4517	13
1944	1/2	Kintex-7 [66]	8●	608	6	33351	102	8●	557	10	18389	70
648	1/2	Virtex-6 [67]	6●	281	–	35668	81	6●	221	9	6115	16
16384	3/4	Zinq [64]	10●	2130	–	49121	50	10●	723	68	30916	183
576	1/2	Virtex-6 [68]	10●	33	–	8685	106	10●	106	16	4896	11
2304	1/2	Virtex-6 [68]	10●	307	–	34104	281	10●	354	20	21308	41
2304	1/2	Virtex-6 [51]	10●	2130	75	33028	258	10●	354	20	21308	41
1024	1/2	Virtex-6 [69]	30●	–	–	25035	72	30●	136	56	22232	57
1944	1/2	Kintex-7 [70]	4●	608	–	33351	102	10●	557	10	18389	70
2304	1/2	Virtex-7 [71]	20○	290	–	< 15520	76	10●	308	22	13481	60

[1] : LUTs pour FPGA Xilinx, ALMs pour FPGA Altera
[2] : BRAM 18K pour Xilinx, BRAM 20K pour Altera
○ Ordonnancement par inondation ● Ordonnancement par couches

pas toujours indiqués dans les publications. De plus, certaines optimisations spécifiques au code LDPC considéré ou à l'objectif des travaux de recherche présentés sont parfois appliquées. Nous avons donc tenté de faire des comparaisons aussi équitables que possible en détaillant les forces et les faiblesses des solutions proposées.

La plupart des travaux précédents ciblent des circuits FPGA de la famille Stratix de chez Intel ou d'anciennes familles FPGA de chez Xilinx. L'outil Vivado HLS de Xilinx génère quant à lui, des descriptions d'architectures au niveau RTL en VHDL pour des cibles FPGA Xilinx. C'est pourquoi, pour réaliser nos comparaisons, des architectures de décodeur LDPC ont été générées pour la cible Virtex-7 puis implémentées avec Quartus II[4] ou Xilinx ISE 14.7. Il est à noter que l'outil Vivado HLS ne génère pas des

4. Le code source VHDL a été modifié pour les déclarations mémoire

descriptions d'architectures au niveau RTL optimisées pour les FPGA de chez Intel ou les anciens FPGA Xilinx. En effet, l'outil Vivado HLS ne possède pas d'analyse de temps ou d'estimations de complexité matérielle pour les ressources matérielles d'anciens circuits FPGA Xilinx ou des circuits FPGA d'Intel. Pour cette raison, les résultats donnés dans le tableau 2.5 correspondent aux cas les moins favorables de notre approche. Tous les résultats fournis sont issus des rapports après placement et routage.

2.3.2.1 Comparaison avec des architectures produites via la HLS

L'analyse du Tableau 2.4, montre l'attrait de l'approche proposée, et plus particulièrement du modèle $\mathcal{A}_2$ pour la génération de décodeurs LDPC. Ce dernier améliore de manière notable les performances obtenues par rapport aux travaux antérieurs basés l'utilisation d'outils HLS [46, 48, 49, 45]. En effet, les complexités matérielles des architectures de décodeurs générées sont considérablement réduites (max=95%, moy=90%) pour toutes les implémentations évaluées, sauf pour la quatrième comparaison. Pour cette dernière, la complexité matérielle est équivalente mais la consommation de BRAM est divisée par 1,5. Pour des travaux connexes, les implémentations de décodeurs proposées présentent des améliorations significatives (2× à 20×) au niveau de consommation de logique et de l'utilisation de blocs RAM par rapport aux résultats présentés dans [45] et [49]. Par ailleurs, les débits obtenus par l'approche proposée sont de 4,5 à 40 fois supérieurs. En conclusion, les architectures proposées offrent des débits plus élevés et une complexité matérielle moindre que les travaux antérieurs dans le domaine. Pour une complexité matérielle équivalente, de nombreux cœurs de décodeur LDPC peuvent donc être instanciés selon notre approche. Dans ce cas, l'écart au niveau du débit à équi-complexité s'en trouve renforcé.

2.3.2.2 Comparaison avec des architecture RTL décrites manuellement

Une comparaison avec les précédents travaux proposant des implémentations sur FPGA à partir de descriptions au niveau RTL est faite dans le Tableau 2.5. L'analyse comparative est plus sensible pour ces architectures RTL obtenues à partir de l'expertise de conception [65, 66, 67, 72]. La plupart des travaux précédents incluent des optimisations algorithmiques ou matérielles spécifiques. Les résultats que nous obtenons sont plus nuancés et démontrent que la méthodologie proposée permet d'atteindre des performances de débit et complexité de même ordre de magnitude que les travaux de la littérature. Par rapport à [66, 67, 68, 69, 70, 71], l'approche proposée atteint un meilleur compromis débit/complexité matérielle. Comparé à [65], cette approche peut être instancié environ 12× pour un coût matériel équivalent. Cela signifie qu'un débit d'environ 1600 Mbps peut être

Tableau 2.6 – Comparaison des implémentations de l'approche proposée avec des travaux de la littérature sur CPU/GPU

Code LDPC			Travaux relatifs				Approche proposée					
Type	N	R	Archi	Iters	Γ	μs	FPGA	Iters	Γ	μs	Logic	RAM
CPU	2304	1/2	i7-4960HQ [59]	10•	998	130	Virtex-7	10•	616	11	24798	62
	2304	1/2	i7-5650U [43]	10•	385	12	Virtex-7	10•	616	11	24798	62
	1944	1/2	i7-2600K [73]	5•	30	–	Virtex-7	5•	914	6	20925	52
GPU	2304	1/2	GTX Titan [54]	10○	316	–	Virtex-7	5•	1232	6	24798	62
	2304	1/2	GTX 470 [53]	10○	52	–	Virtex-7	5•	1232	6	24798	62

○ Ordonnancement par inondation • ordonnancement par couches

atteint. Par conséquent, l'approche proposée délivre un débit diminué de moitié dans ce cas de figure. Cette première analyse montre que notre approche est pertinente par rapport aux architectures de décodeur décrit au niveau RTL. En effet, comme cela était attendu, une réduction des performances est observée mais les architectures produites à partir de notre modèle sont dans les mêmes ordres de magnitude au niveau des performances que les architectures dédiées.

2.3.2.3 Comparaison avec des implémentations logicielles

Une analyse de l'approche proposée par rapport à des implémentations sur CPU et sur GPU a également été menée. Le Tableau 2.6 présente les débits et les latences pouvant être atteints actuellement sur des cibles de type GPP/GPU. Les résultats rapportés pour les architectures de décodeurs LDPC générés et implantés sur circuit FPGA, le sont lorsque seulement une instance de décodeur est considérée.

A partir des données présentées dans ce tableau, nous observons que les architecture $\mathcal{A}_2$ possèdent des débits supérieurs à la moitié des travaux [73, 53] et inférieurs à l'autre moitié [43, 54]. Cependant les travaux sur les architectures programmables utilisent de multiples coeurs de calculs tandis qu'un seul coeurs est utilisé sur FPGA. Or, jusqu'à 20 décodeurs LDPC peuvent être instanciés sur un FPGA Virtex-7. Une telle architecture est capable de supporter un débit d'environ 2,5 Gbps comparé à [43] et un débit de plus de 5 Gbps comparé à [54]. Ces résultats démontrent l'intérêt des implémentations de décodeurs sur circuit FPGA basés sur la HLS par comparaison avec des décodeurs logiciels.

2.3.3 Prototypage de décodeur LDPC matériel

Les résultats de débit et de latence pour les implémentations de décodeurs LDPC générées par HLS ont été présentés dans la section précédente. Des prototypages sur carte FPGA

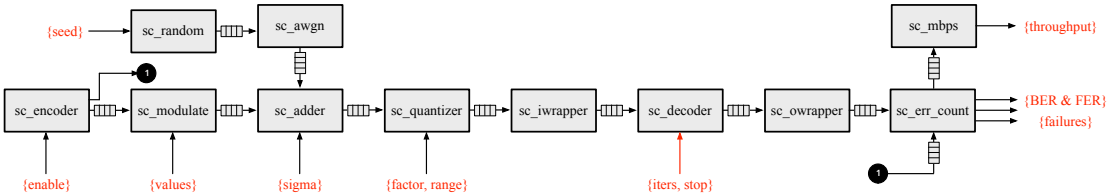

Figure 2.12 – Système d'émulation matérielle développé reposant sur la méthodologie HLS

ont également été effectués pour vérifier le bon fonctionnement des implémentations matérielles de décodeurs LDPC. Les performances de correction d'erreurs des décodeurs LDPC générés ont été estimées dans un premier temps à l'aide de simulations de Monte-Carlo. Une configuration d'émulation entièrement matérielle a été privilégiée pour évaluer les décodeurs LDPC dans des conditions réelles d'utilisation. Cet environnement matériel fiable et complet est illustré par la Figure 2.12. Une chaîne complète de communications numériques a été décrite à partir de modèles SystemC. Ces modèles ont été implémenté avec l'outil Vivado HLS. Des modules tels que les générateurs de bruit AWGN ont été décrits en utilisant des opérations en virgule flottante pour garantir la qualité du bruit généré (cela garanti une équivalence entre les simulations logicielles basées sur langage C et les simulations matérielles [74]). Jusqu'à 8 chaînes de génération d'échantillons bruités mise en parallèles ont été instanciées afin de saturer le décodeur LDPC matériel.

Dans une chaîne complète de communication, la précision du canal AWGN est très importante afin de garantir la fiabilité des courbes de performances BER/ FER obtenues. Dans le cadre de ces travaux de thèse cinq modèles de canaux AWGN utilisant cinq méthodes différentes de génération de bruit [75, 76, 77, 78, 79] ont été décrits. Ces modèles permettent la génération de différents compromis d'architectures entre précision du bruit gaussien, débit et complexité matérielle. Ces modèles permettent une rapidité de prototypage des chaînes de communications tout en conservant une fiabilité du bruit généré. Ces travaux ont fait l'objet d'une publication en conférence [80].

Les courbes de performance d'erreurs FER sont tracées jusqu'à 10^{-8} dans la Figure 2.13 pour un code LDPC 5G 3GPP avec 10 itérations de décodage. Le modèle de décodeur LDPC associé aux modèles de la plate-forme d'émulation matérielle démontre la cohérence de l'approche pour générer des cœurs de décodeur LDPC performants et efficaces. Il montre également la pertinence d'une méthodologie de conception basée sur l'utilisation de modèles comportementaux pour concevoir rapidement des systèmes complexes.

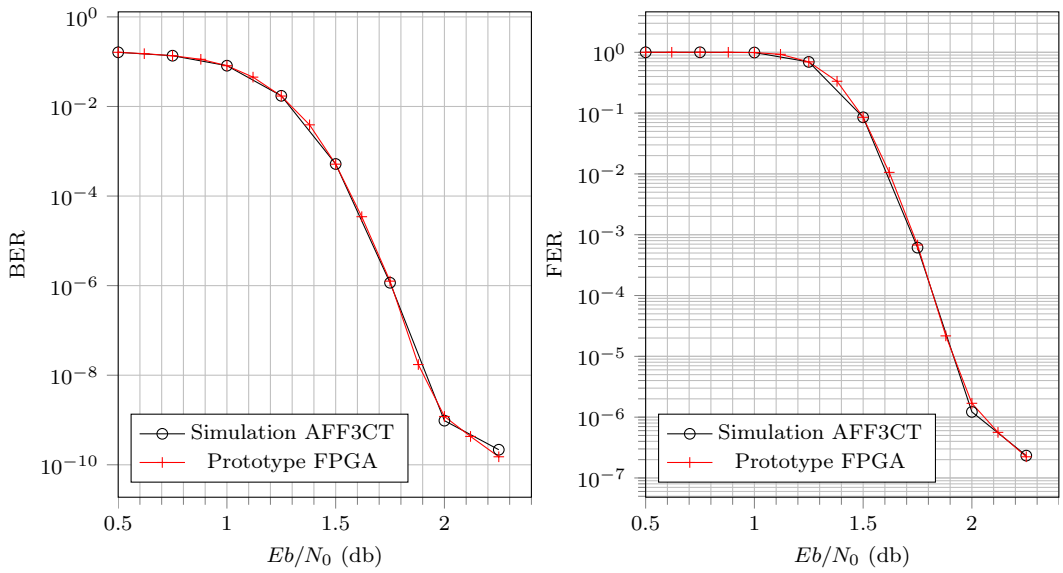

Figure 2.13 – Courbe de BER et FER sur prototype FPGA pour un code LDPC 3GPP 5G (4048,2112,1/2)

Conclusion

Une nouvelle approche pour concevoir des décodeurs LDPC efficaces basés sur des modèles comportementaux a été détaillée tout au long de ce chapitre. Contrairement aux travaux précédents, la formulation en ordonnancement par couches horizontales du processus de décodage a été privilégiée. Deux modèles comportementaux ont été proposés et développés avec succès. Les avantages et les inconvénients de ces deux modèles ont été étudiés d'un point de vue théorique et évalués expérimentalement. Les décodeurs LDPC conçus permettent d'atteindre des débits de décodage de 161 Mbps à 1010 Mbps sur un FPGA de type Xilinx Virtex-7. Ces performances dépendent des paramètres du code LDPC. Par rapport à des travaux similaires, l'approche fournit des résultats plus intéressant au niveau du débit et de la complexité matérielle. Enfin les architectures générées sont assez proche des architectures dédiées décrites manuellement au niveau RTL.

Les travaux présentés dans ce chapitre ont fait l'objet de publications scientifiques dans la conférence nationale COMPAS [81], dans les conférences internationales ICECS [80, 82] et ISTC [83] ainsi qu'une publication dans une revue internationale JSPS [84].

3 Modèle d'architectures de décodeurs Polaire SC

Dans le chapitre 2 il a été démontré la pertinence de la méthodologie pour la conception d'architectures de décodeurs LDPC. Les algorithmes de décodages des codes LDPC sont itératifs et présentent un fort potentiel de parallélisation. Cependant, les codes polaires ont des caractéristiques différentes. En effet les algorithmes de décodage des codes polaires sont fortement séquentiels et présentent beaucoup d'irrégularités lorsqu'ils sont optimisés. Dans ce chapitre 3, la méthodologie de conception basée sur l'utilisation de modèles comportementaux est éprouvée pour la génération d'architectures de décodeurs polaires. La section 3.1 introduit les codes polaires. La contribution originale est présentée dans la section 3.2, puis les résultats expérimentaux sont détaillés dans la section 3.3.

Introduction

Les codes polaires [23] sont des codes correcteurs d'erreurs (ECC) qui permettent d'atteindre théoriquement des performances proches de la capacité du canal gaussien. Récemment, ils ont été adoptés dans la cinquième génération du standard de communications numériques mobiles (3GPP-5G) pour la protection des canaux de contrôle. Il existe différents algorithmes de décodage pour décoder les codes polaires. Chacun d'entre eux offre un compromis différent entre d'une part le pouvoir de correction et d'autre part la complexité calculatoire. L'un des algorithmes de décodage les plus étudiés est l'algorithme par annulation successive (SC). L'algorithme SC fournit un assez bon pouvoir de correction et peut être utilisé seul ou associé à d'autres techniques comme par exemple le SC-Flip [85, 86]. Cette association permet d'augmenter son pouvoir de correction afin de se rapprocher des performances de l'algorithme SC-List [87]. Différentes architectures matérielles dédiées de décodeur polaire par annulation successives (SC), décrites au niveau RTL sont proposées dans la littérature. Les travaux visent à réduire la complexité matérielle du décodeur [88, 89, 90] et/où à améliorer la latence et le débit du décodeur [91, 92, 93].

Les travaux portant sur l'implantation de décodeurs matériels pour les codes polaires reposent majoritairement sur des architectures ayant un parallélisme de calcul figé. Ces architectures nécessitent de longs cycles de développement. Cela rend l'exploration de l'espace des solutions complexe à grande échelle. De plus les spécifications au niveau du parallélisme, du débit, de la quantification et de la taille du code doivent être optimisées en fonction du domaine applicatif. La recherche de flexibilité est particulièrement criante dans les applications de communications numériques du standard 5G 3GPP [94] et sont à prendre en compte dans les travaux exploratoires pour la 6G [95]. Or, l'hétérogénéité des systèmes de communications numériques conduit à concevoir des architectures matérielles selon des compromis orthogonaux (débit, complexité matérielle, consommation énergétique). Par exemple, les exigences de la 5G selon l'application considérée sont différentes. Une diffusion vidéo haute définition 8k nécessite plusieurs Gbit/s de débit alors que des réseaux de capteurs n'ont besoin que de quelques Kbit/s de débit.

Pour répondre à cette difficulté de l'exploration de l'espace des solutions architecturales, nous avons évalué dans le cadre des travaux de thèse la pertinence des méthodologies de conception basées sur des modèles comportementaux afin de générer des décodeurs auto-adaptatifs efficaces de codes polaires SC sur circuit FPGA. Contrairement aux codes LDPC qui peuvent être modélisés à l'aide de boucles, l'algorithme de décodage SC est décrit de manière récursive. Il ne nécessite pas d'accès irréguliers aux données. Ces propriétés différentes par rapports à celles des algorithmes de décodage de codes LDPC

offrent un cadre d'étude complémentaire concernant la maturité des technologies de type HLS.

Le reste du chapitre est organisé comme suit. La section 3.1 présente le principe de décodage des codes polaires à l'aide de l'algorithme de type SC. La section 3.2 se concentre sur le modèle comportemental favorisant la génération de décodeurs SC. L'approche expérimentale et les résultats sont détaillés dans la section 3.3. Enfin, une conclusion est présentée en fin de chapitre.

3.1 Les codes polaires

3.1.1 Algorithmes de décodage de codes polaires

Les codes polaires sont des codes linéaires en blocs de taille $N = 2^n$ avec n un nombre entier. Dans [23], les auteurs définissent une construction basée sur la $n^{\text{ème}}$ puissance de Kronecker d'une matrice noyau $\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Le processus de codage consiste à multiplier $\kappa^{\otimes n}$ par un vecteur u de N bits contenant K bits d'information et $N - K$ bits gelés fixés à une valeur connue. L'emplacement des bits gelés de u est connu à la fois par l'émetteur et le récepteur. La polarisation dépend du type de canal de transmission et du niveau de bruit [96]. Le rendement du code est défini par $R = \frac{K}{N}$. Après avoir été diffusé à travers le canal de transmission, une version bruitée du mot de code est reçue sous la forme d'un logarithme du rapport de vraisemblance (LLR) appelé vecteur Λ. Plusieurs algorithmes de décodage existent pour décoder le vecteur Λ. L'algorithme de décodage originel est l'algorithme par annulation successif (SC) [97]. L'algorithme SC offre des caractéristiques de mise en œuvre matérielle intéressantes. Par ailleurs, il permet d'atteindre des performances de débit intéressante [92, 98, 99] pour une complexité calculatoire modérée.

D'autres algorithmes ayant des capacités de correction plus élevées existent pour le décodage de codes polaires. Néanmoins l'algorithme de référence est l'algorithme de décodage SC. C'est la raison pour laquelle ce chapitre adresse exclusivement cet algorithme.

3.1.2 Algorithme de décodage par annulation successive

Comme mentionné précédemment, un vecteur Λ est reçu via la canal de transmission. Le processus de décodage vise à faire successivement une estimation pour chaque valeur des

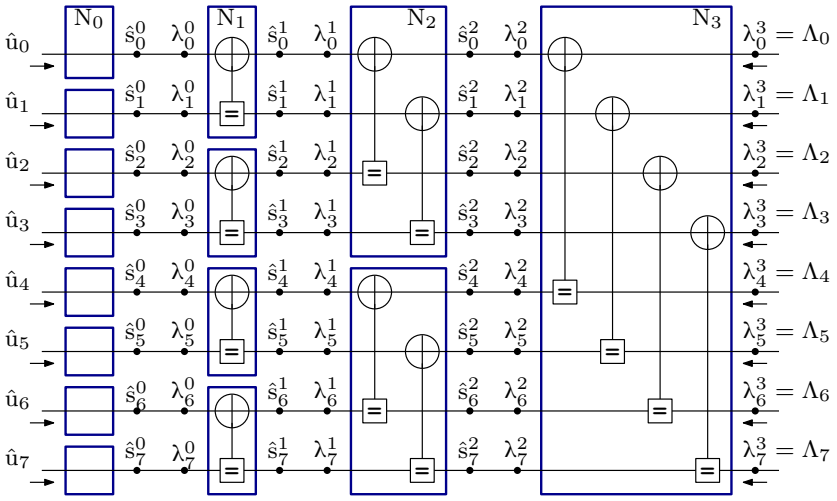

Figure 3.1 – Représentation en *factor graph* du décodage SC d'un code polaire de taille $N = 8$

bits u_i sur la base des bits estimés précédemment $(\hat{u}_{0:i-1})$ [1]. Comme proposé dans [23], une représentation en *factor graph* des codes polaires peut être utilisée pour représenter le calcul de λ_i^0. La Figure 3.1 contient un exemple de *factor graph* pour une largeur de code $N = 8$. Le décodeur estime les bits $\hat{u}$ à partir du calcul des LLRs injectés à la droite du graphe. Afin d'estimer chaque bit u_i, le décodeur calcule une valeur de LLR comme suit :

$$\lambda_i^0 = \log \frac{\Pr(\Lambda, \hat{u}_{0:i-1} | u_i = 0)}{\Pr(\Lambda, \hat{u}_{0:i-1} | u_i = 1)} \tag{3.1}$$

Le bit estimé $\hat{u}_i$ est ensuite calculé de la manière suivante :

$$\hat{u}_i = \begin{cases} 0 & \text{si} \lambda_i^0 > 0, \\ 1 & \text{sinon} \end{cases} \tag{3.2}$$

Puisque le décodeur a connaissance de l'emplacement des bits gelés, si u_i est un bit gelé, le bit estimé $\hat{u}_i = 0$ est utilisé quelle que soit la valeur calculée de λ_i^0. Le LLR d'entrée λ_i^j

1. $\hat{u}_{0:i-1} = [\hat{u}_0 ... \hat{u}_{i-1}]$

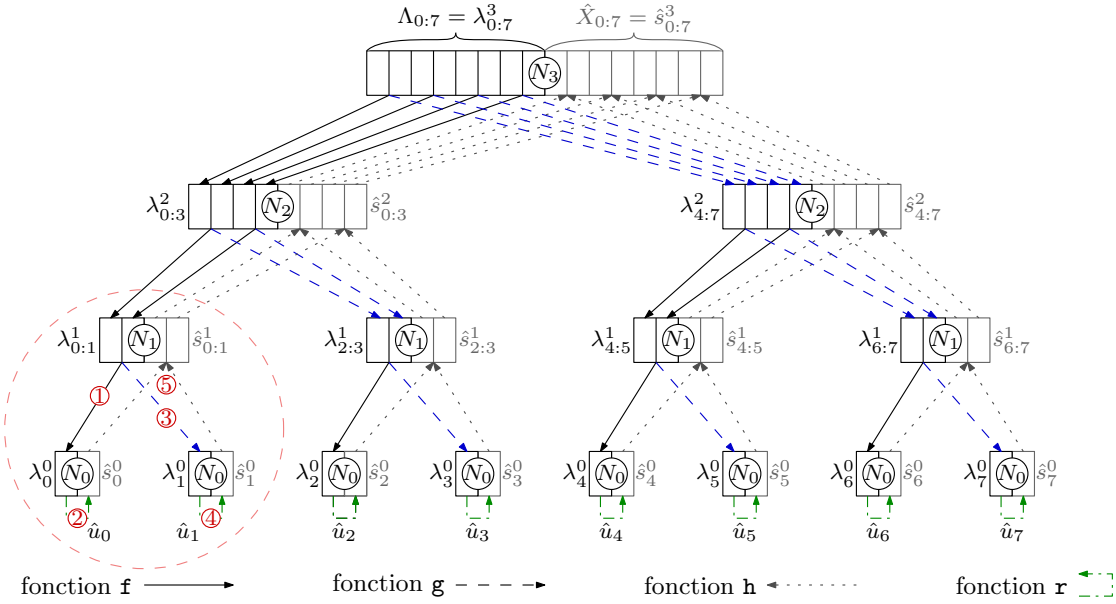

Figure 3.2 – Représentation sous forme d'arbre du processus de décodage d'un décodeur polaire SC $N = 8$

est calculé de telle sorte que :

$$\lambda_i^j = \begin{cases} \mathbf{f}\left(\lambda_i^{j+1}, \lambda_{i+2^j}^{j+1}\right) & \text{quand } B(i,j) = 0 \\ \mathbf{g}\left(\lambda_{i-2^j}^{j+1}, \lambda_i^{j+1}, \hat{s}_{i-2^j}^j\right) & \text{sinon} \end{cases} \tag{3.3}$$

où $B(i,j) \equiv \lfloor \frac{i}{2^j} \rfloor \mod 2$, $0 \leq i < N$ et $0 \leq j < n$. $\hat{s}_i^j$ représente la somme partielle. Elle correspond à la propagation des décisions dures dans le *factor graph*. Dans la Figure 3.1, nous avons $\hat{s}_1^2 = \hat{u}_1 \oplus \hat{u}_3$ (somme modulo 2). Par ailleurs, les fonctions $\mathbf{f}$ et $\mathbf{g}$ sont définies comme :

$$\mathbf{f}(\lambda_a, \lambda_b) = sign(\lambda_a . \lambda_b) . \min(|\lambda_a|, |\lambda_b|) \tag{3.4}$$

$$\mathbf{g}(\lambda_a, \lambda_b, \hat{u}) = (-1)^{1-2\hat{u}} \lambda_a + \lambda_b, \tag{3.5}$$

où λ_a et λ_b sont des LLRs entrants et $\hat{u}$ est une somme partielle. Pour les codes polaires systématiques, le décodeur estime le vecteur à droite du graphe au lieu de $\hat{u}$ à gauche. Le vecteur estimé $\hat{u}$ est multiplié par $\kappa^{\otimes n}$ pour obtenir l'estimation $\hat{X}$ de x. La nature séquentielle de l'algorithme de décodage SC implique de fortes dépendances de données

Algorithme 4: Fonction de mise à jour du noeud $\mathcal{N}_j$

Nécessite : $\lambda^j_{a:a+J-1}$

Calcul $J/2$ fonctions f concurrentes :

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

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

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

Calcule $J/2$ fonctions g concurrentes :

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

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

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

entre les nœuds. Cette caractéristique limite le parallélisme algorithmique exploitable. Les dépendances de données de l'algorithme de décodage SC peuvent être modélisées à l'aide d'un arbre binaire comme suggéré dans [91]. Lors du processus de décodage, cet arbre binaire représenté dans la Figure 3.2 est parcouru de façon récursive dans l'ordre suivant : noeud racine vers sous-noeud gauche puis vers sous-noeud droit. Le noeud racine est situé au niveau $d = \log_2(N) - 1$ et les noeuds feuilles sont les noeuds du niveau $d = 0$. Le noeud racine $\mathcal{N}_3$ reçoit l'information de canal L et échange successivement des données avec les sous-noeuds gauche et droit $\mathcal{N}_2$. Les nœuds feuilles $\mathcal{N}_0$ sont appelés par les nœuds $\mathcal{N}_1$ et ils génèrent l'estimation $\hat{u}_i$. En supposant qu'un noeud non feuille/non racine $\mathcal{N}_j$ reçoit $\lambda_{a:a+J-1}$, il exécute l'Algorithme 4 avec :

$$\begin{cases} J = 2^j, \\ a = m.2^j, \\ 0 \leq m < 2^{n-j}. \end{cases} \tag{3.6}$$

L'encodage des codes polaires est majoritairement systématique. C'est pourquoi, le vecteur $\hat{X}$ doit être calculé. Dans ce cas un ré-encodage du vecteur $\hat{u}$ est nécessaire. Il équivaut à la propagation de décisions dures $\hat{u}_i$ vers le haut de l'arbre. Le vecteur $\hat{X}$ est obtenu en calculant le vecteur somme partielle $\hat{s}^n_{0:N-1}$. Chaque nœud du graphe contient $\frac{N}{2^{n-d}}$ LLRs et leurs sommes partielles. Les LLRs et les valeurs des sommes partielles sont utilisés pendant le processus de décodage pour calculer les valeurs requises par leurs nœuds fils/racines respectifs. Il est à noter que la complexité calculatoire par nœud est divisée

par deux à chaque niveau. Le nombre d'occurrences des fonctions **f** et **g** pour un noeud situé au niveau d, est noté C_d. Il peut être estimé par $C_d = \frac{N}{2^{n-d+1}}$. Par exemple, lorsque $N = 8$ au niveau $d = 3$, $\frac{N}{2}$ calculs de type **f** et $\frac{N}{2}$ calculs de type **g** sont nécessaires pour produire les valeurs d'entrée du noeud $\mathcal{N}_2$. Chaque noeud $\mathcal{N}_2$ exécute $\frac{N}{4}$ fonctions **f** et $\frac{N}{4}$ fonctions **g** pour les $\frac{N}{2}$ valeurs reçues du noeud $\mathcal{N}_3$ afin d'alimenter les noeuds fils gauche et droit. Le processus de décodage est illustré dans la Figure 3.2 et peut se décrire comme suit :

1. La fonction **f** est exécutée C_d fois pour calculer les $\frac{N}{2^{n-d+1}}$ LLRs du noeud fils gauche.
2. La fonction **r** calcule les sommes partielles du noeud fils gauche. Lorsque le noeud est un noeud feuille i.e. quand $d = 0$, **r** est défini par $\mathbf{r}(\lambda_a) = sign(\lambda_a)$.
3. La fonction **g** est exécutée C_d fois pour calculer les $\frac{N}{2^{n-d+1}}$ LLRs du noeud fils droit.
4. De manière similaire à l'étape 2, la fonction **r** calcule les sommes partielles du noeud fils droit.
5. La fonction **h** transfère les sommes partielles au niveau supérieur. **h** est défini par $\mathbf{h}(\hat{u}_a, u\hat{u}_b) = \hat{u}_a \otimes u_b$.

L'algorithme de décodage SC est bien adapté pour une implémentation matérielle car il possède une faible complexité calculatoire qui a pour ordre de grandeur $\mathcal{O}(N \log N)$). Cependant, le processus de décodage étant principalement séquentiel, il nécessite une traversée pré-ordonnée de l'arbre binaire [100]. Cette traversée fait varier le niveau de parallélisme exploitable en fonction du niveau de l'arbre considéré.

3.1.3 Optimisations algorithmiques et architecturales

Depuis la découverte des codes polaires par Arikan en 2009 [23], de nombreux travaux concernent les optimisations algorithmiques mais aussi les aspects architecturaux afin d'explorer les différents le parallélismes. L'objectif est à la fois de réduire la latence de décodage et d'augmenter le débit des architectures matérielles.

3.1.3.1 Optimisations algorithmiques

Une optimisation notable de niveau algorithmique a été proposée dans [91]. L'algorithme SC simplifié (SSC) réduit fortement la complexité calculatoire du processus de décodage en élaguant les branches de l'arbre dont les résultats peuvent être pré-déterminés. Cet élagage réduit drastiquement le nombre de noeuds de bas niveau à évaluer. Cela est illustré dans la Figure 3.3.

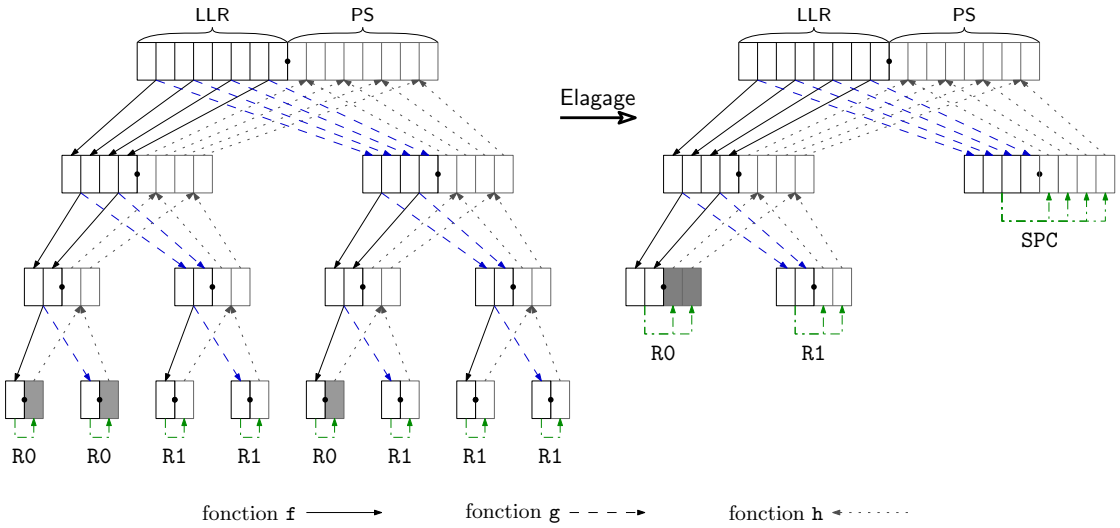

FIGURE 3.3 – Arbre de décodage polaire simplifié

Dans l'algorithme SSC, les sous-arbres qui sont uniquement composés de nœuds dont les bits sont gelés (`R0`) ou de nœuds de bits d'information (`R1`) ne sont pas traités par le processus de décodage. En effet, comme les noeuds feuilles d'un noeud `R0` sont tous des bits gelés, le résultat du traitement est un vecteur composé uniquement de valeurs 0. Les noeuds feuilles d'un noeud `R1` sont tous des bits d'information. Le résultat du décodage des sous-arbres de rendement `R0` et `R1` peut être pré-calculé de la manière suivante :

$$\texttt{R0}(\lambda_a) = 0 \tag{3.7}$$

$$\texttt{R1}(\lambda_a) = \begin{cases} 0 & \text{si } \lambda_a \geq 0, \\ 1 & \text{sinon} \end{cases} \tag{3.8}$$

Afin de réduire encore la complexité calculatoire et d'améliorer la latence de décodage, d'autres types de sous-arbres ont été identifiés. Les différents types et les algorithmes de décodage sont détaillés dans [92] et étendus dans [101]. Par exemple, le type REPetition (`REP`) est composé de bits gelés et d'un seul bit d'information sur le dernier bit ($f_b = [0000 \ldots 001]$). Pour pré-calculer un noeud de type `REP`, la somme de ses valeurs λ est réalisée. Une décision dure est appliqué sur le résultat de la somme des LLRs. Puis, la valeur binaire issue de ce calcul est affectée à l'ensemble des sommes partielles.

Par ailleurs, les noeuds de type contrôle de parité simple (`SPC`) sont constitués d'un premier bit gelé et de bits d'information ($f_b = [0111 \ldots 111]$). Pour pré-calculer ce motif, il est nécessaire :

1. de détecter le LLR avec la plus faible probabilité,
2. de calculer l'équation de parité à partir des signes des LLRs,
3. si l'équation de parité n'est pas respectée le signe du LLR le moins fiable est inversé lors de l'affectation des sommes partielles.

D'autres types sont répertoriés dans la littérature [92, 102, 101, 103]. Certains d'entre eux sont des combinaisons (`G-R0`, `R0-R1`, et `R0-SPC`) de types présentés précédemment [92, 101, 103] tandis que d'autres adressent des motifs plus complexes [102]. Comme l'illustre la Figure 3.3, les élagages réduisent la latence globale de décodage de l'arbre.

Les architectures matérielles basées sur l'algorithme SC avec de l'élagage [92, 101, 103] voient leurs latences et leurs débits grandement varier en fonction du code polaire. L'élagage de l'arbre qui est lié à l'emplacement des bits gelés dépend du rendement du code ainsi que du niveau de bruit du canal de transmission (σ) considéré pour concevoir le code polaire. Les décodeurs matériels intégrant l'élagage sont plus performants au niveau du débit et de la latence par comparaison aux décodeurs implémentant l'algorithme SC original. Malheureusement, l'ajout de l'élagage rompt la régularité et la flexibilité de l'architecture de décodage [104, 105, 88, 89, 106, 90, 107]. Par conséquent, un contrôleur micro-codé spécifique à chaque code polaire est nécessaire au niveau architectural [92, 101]. Ce type de contrôleur nécessite un programme pré-généré et impacte la flexibilité durant l'exécution. De plus, le coût de la mémoire du programme devient un goulot d'étranglement pour les systèmes devant supporter plusieurs codes polaires.

3.1.3.2 Optimisations architecturales

Afin d'aboutir à des architectures matérielles fournissant des débits proche du Gbps, différentes techniques de parallélisation ont été proposées. Si plusieurs unités de calcul sont disponibles, la parallélisation des calculs peut être effectuée au niveau des noeuds de l'arbre pour réduire le temps d'exécution du processus de décodage. Comme l'illustre la Figure 3.2, le traitement du noeud $\mathcal{N}_d^l$, fils gauche du noeud $\mathcal{N}_{d+1}$, nécessite $\frac{N}{2^{n-d+1}}$ calculs de type $\mathbf{f}$. De même, le traitement du noeud $\mathcal{N}_d^r$, fils droit du noeud $\mathcal{N}_{d+1}$, nécessite $\frac{N}{2^{n-d+1}}$ calculs de type $\mathbf{g}$. $\frac{N}{2^{n-d+1}}$ calculs de type $\mathbf{h}$ sont également nécessaires pour propager les résultats de somme partielle des noeuds $\mathcal{N}_d^l$ et $\mathcal{N}_d^r$ au noeud parent $\mathcal{N}_{d+1}$.

Sachant que les calculs $\mathbf{f}$ et $\mathbf{g}$ sur les LLR sont indépendants, il est possible de paralléliser les traitements. Ainsi, le temps de calcul nécessaire à l'exécution du noeud $\mathcal{N}_d$ peut être divisé par $\mathbf{P}$ lorsque $\mathbf{P}$ éléments de traitement (PE) sont alloués à chaque type de fonction ($\mathbf{f}$, $\mathbf{g}$ et $\mathbf{h}$). De nombreuses architectures de décodage semi-parallèles sont décrites dans la littérature avec $\mathbf{P} = 64$ PEs [100, 88, 89]. Ce nombre important de PE est pertinent pour

le haut de l'arbre lorsque la valeur de $\frac{N}{2^{n-d+1}}$ est élevé. Cependant, la plupart des $\mathbf{P}$ PE se sont pas assignés lors du traitement des noeuds en bas de l'arbre, c'est à dire lorsque $\mathbf{P} > \frac{N}{2^{n-d+1}}$.

Cette sous-utilisation des unités fonctionnelles ne permet pas de diviser le temps de décodage total par $\mathbf{P}$. De plus, le temps de calcul lié au traitement des noeuds en bas de l'arbre devient prédominant. Une solution pour supprimer d'une partie des pénalités consiste à fusionner le traitement des noeuds feuilles avec leurs noeuds parents comme expliquer dans [106]. Il devient ainsi possible de calculer deux nœuds feuilles durant un cycle d'horloge. Cette technique de fusion pour les nœuds inférieurs est étendue dans [90, 108] où un élément de traitement terminal $\mathbf{T_p}$ a été proposé. L'élément $\mathbf{T_p}$ est un sous-décodeur polaire de taille $\mathbf{P}$ complètement déroulé. Cette approche minimise la latence de traitement des feuilles terminales au prix d'une augmentation de la complexité matérielle.

3.1.4 Motivations des travaux

La conception de décodeurs SC de niveau RTL avec ou sans l'exploitation des techniques d'élagage est chronophage, en raison de l'espace des solutions architecturales à explorer. Développer et optimiser une architecture au niveau RTL afin de répondre à plusieurs contraintes applicatives et technologiques devient très délicat. La définition d'un cadre favorisant l'exploration des solutions architecturales de décodeurs polaires est détaillé dans la suite de ce chapitre. Un modèle générique et flexible est proposé. Il permet d'adapter les niveaux de parallélisme, les choix d'élagage et les formats de quantification pour aboutir à des architectures offrant différents compromis.

3.2 Modèle comportemental générique de décodeur de codes polaires

Dans la section suivante, un modèle comportemental générique de décodeur SC de codes polaires est présenté. La stratégie de conception et les optimisations du modèle pour des architectures matérielles flexibles et efficaces sont détaillées. Notre méthodologie de conception est basée sur l'outil Xilinx Vivado HLS 2018.2 pour générer des décodeurs matériels SC à partir du modèle comportemental. Cette stratégie de conception est résumée par la Figure 3.4. La description comportementale se base sur une API en 3 couches. Cette API a été spécialement conçue pour la synthèse HLS, elle contient les fonctions nécessaires à l'algorithme de décodage SC. La description comportementale et l'API sont paramétrables avec des contraintes algorithmiques et architecturales. Le

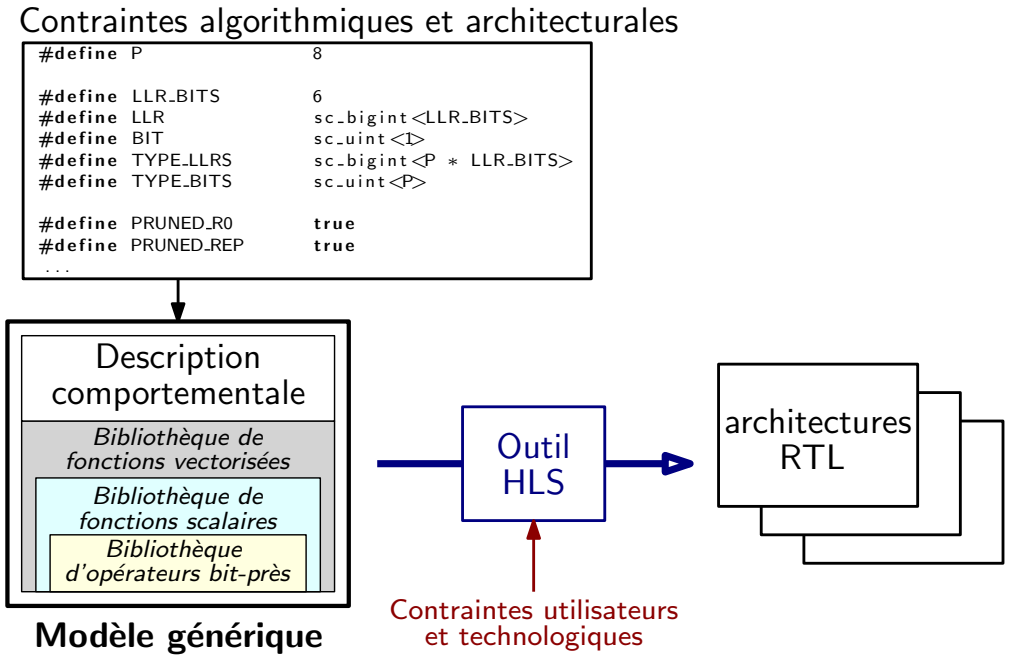

Figure 3.4 – Méthodologie basée sur un modèle comportemental pour la génération de décodeurs polaires SC

modèle comportemental peut alors être synthétisé dans l'outil HLS.

3.2.1 Modèle architectural

Depuis l'émergence des codes polaires [109], de nombreuses études ont été menées pour concevoir des décodeurs de codes polaires sur des cibles ASIC et FPGA. Ces architectures efficaces au niveau de la complexité matérielle et/ou du débit de décodage sont basés sur deux modèles architecturaux distincts :

— *Les architectures entièrement déroulées* comme dans [99, 98, 93, 110, 111, 103]. Dans ces architectures, une ressource de calcul est instanciée pour chaque opération présente dans l'arbre de décodage. Cette approche implique une complexité matérielle élevée et une faible flexibilité. Cependant, pour des codes de petites tailles, les débits de ces architectures peuvent atteindre plusieurs centaines de Gbps sur technologie ASIC.

— *Les architectures semi-parallèles* comme dans [104, 105, 88, 89, 106, 90, 107] sont des architectures qui séquencent l'exécution des traitements de l'arbre de décodage. Toutes les opérations à exécuter sont ordonnancées sur un ensemble de **P** unités de traitement. Ce type d'architecture offre des débits pouvant varier de quelques Mbps à quelques centaines de Mbps sur une cible FPGA. Les performances dépendent de

la taille du code (N), du degré de parallélisme ($\mathbf{P}$) en interne du décodeur et des techniques d'élagage mises en oeuvre.

Dans le cadre de ce travail, nous nous intéressons à l'exploration de l'espace des solutions architecturales pour la conception de décodeur SC. Or, les architectures entièrement déroulées, de part leur nature, présentent une diversité de compromis performances/-complexité matérielle assez limité. Par conséquent la suite de l'étude est focalisée sur la conception d'architectures semi-parallèles. Le large spectre de compromis architecturaux qu'elles offrent fournit un cadre d'étude intéressant.

Un modèle d'architectures commun pour les architectures semi-parallèles découle des travaux précédents [104, 105, 88, 89, 106, 90, 107]. Ce dernier est présenté dans la Figure 3.5. Cette architecture est constituée des éléments suivants :

— Trois bancs mémoires distincts stockent : $2 \times N$ sommes partielles ($\hat{u}$,$\hat{s}$), $2 \times N$ valeurs de LLR et N bits gelés. La mémoire LLR est généralement organisée en 2 parties : la première partie stocke les N informations du canal (Λ) et la seconde sauvegarde les N LLRs (λ) provenant des calculs internes. Ce partitionnement réduit l'empreinte mémoire lorsque le format de quantification des données de traitement est supérieur à celui des données reçus par le décodeur ($Q(\Lambda) < Q(\lambda)$).
— Une unité de traitement (PU) dédiée au décodage des codes polaires. Lorsqu'il n'y a pas d'élagage, le PU est composé de quatre types d'éléments de traitement (PE), à savoir un élément de calcul par fonction ($\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$ et $\mathbf{T_p}$). Afin d'augmenter le débit de l'architecture matérielle, les PE de chaque type peuvent être instanciés $\mathbf{P}$ fois. Les décodeurs intégrant des techniques d'élagage [92] utilisent des PEs supplémentaires pour décoder les sous-codes considérés (`R0`, `R1`, `REP`, `SPC`, ...).
— Un contrôleur qui planifie les transferts de données entre le système et qui pilote les éléments de calculs lors du décodage.

Afin de pouvoir explorer efficacement l'espace de conception et trouver la solution la plus adéquate vis-à-vis des contraintes applicatives, ce modèle d'architecture doit être finement paramétré. En effet, au moins quatre paramètres doivent être considérés :

— Paramètres algorithmiques :
  1 La longueur du code N - La longueur a un impact linéaire sur l'empreinte mémoire du décodeur matériel. Comme la complexité de calcul de l'algorithme de décodage SC est de l'ordre de $\mathcal{O}(N \log N)$, cela limite le débit. A l'inverse, une valeur N plus élevée améliore le pouvoir de correction d'erreurs. De plus, cela favorise un parallélisme de calcul plus élevé ($\mathbf{P}$).

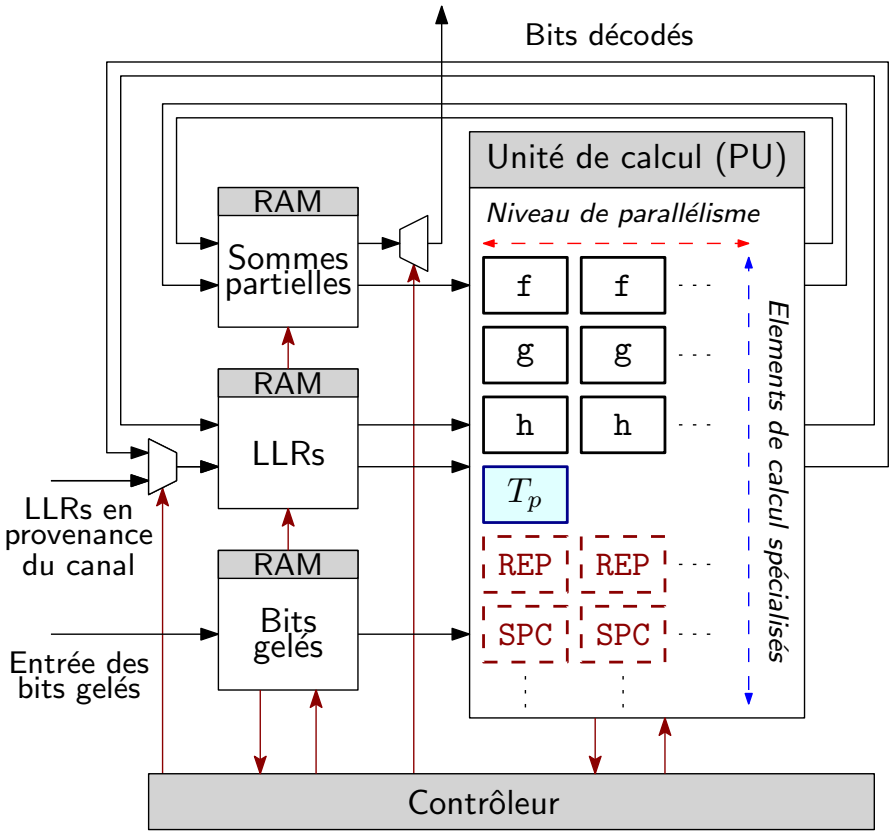

Figure 3.5 – Modèle d'architecture générique pour le décodage semi-parallèle des codes polaires basé sur l'algorithme SC

2 Le format des données (**Q**) - Les valeurs de LLR qui proviennent du canal sont des informations quantifiées. Suivant le contexte, la quantification des valeurs d'entrée et la longueur du code, il est important de spécifier un format de quantification interne approprié. Cependant, il faut savoir que la complexité matérielle de l'architecture évolue linéairement avec le format de quantification retenu. Enfin, comme les paramètres **Q** et N peuvent impliquer des saturations durant le décodage, cela peut impacter le pouvoir de correction du décodeur SC.

— Paramètres architecturaux :

3 Le niveau de parallélisme des calculs **P** - Cette valeur impacte le nombre de PE qui doit être instancié dans le PU. Cela signifie que ce paramètre permet d'augmenter ou de diminuer le débit du décodeur. Comme attendu, ce paramètre a une influence directe sur la complexité de l'architecture matérielle.

4 L'ensemble des optimisations d'élagage **E** - Augmenter le nombre de types de noeuds pris en charge par l'élagage (**E**) est intéressant pour améliorer le débit du décodeur. Cependant, cela accroît la complexité matérielle du décodeur.

L'exploitation ou non des différents types d'élagage dépend directement des codes polaires à prendre en charge par le décodeur. En effet, les positions des bits gelées changent l'organisation de l'arbre [2]. Par conséquent, toutes les techniques d'élagage ne sont pas forcément pertinentes pour un code polaire spécifique. Ainsi sélectionner les unités de traitement associées aux différents types en adéquation avec les besoins réels est nécessaire. Une option de sélection de l'ensemble pertinent des ressources d'élagage **E** est nécessaire durant de la synthèse pour améliorer l'efficacité de l'architecture générée.

Au niveau algorithmique, le but de l'approche est de spécifier un modèle d'architecture de décodage finement paramétrable à partir de N, **P**, **Q** et **E**. Ce modèle comportemental flexible est décrit dans la section suivante.

3.2.2 Description du modèle comportemental

Différentes approches ont été étudiées pour décrire des décodeurs polaires SC ciblant des architectures programmables (GPP/GPU). Les implémentations logicielles reposent traditionnellement sur la récursivité [112, 113, 114] telle que présentée dans l'Algorithme 4. La récursivité est adaptée au parcours séquentiel de l'arbre associé au processus de décodage. Elle facilite donc la description comportementale et le développement d'implémentations logicielles génériques [113, 114]. Par ailleurs, plusieurs travaux ont proposé de dérouler complètement la traversée de l'arbre de décodage afin de maximiser les performances d'implémentation pour les codes courts [115, 116, 114].

Ces approches qui favorisent le développement d'implémentations logicielles efficaces ne sont pas transposables pour la génération d'architectures matérielles [117]. Les deux raisons principales sont :

— La récursivité est mal interprétée par les outils de synthèse de haut niveau. En effet, ces outils doivent déterminer la profondeur de la récursivité durant la synthèse pour pouvoir générer l'architecture. Pour chaque niveau de récursivité, ils génèrent un chemin de données spécifique, introduisant un coût matériel considérable.

— L'approche déroulée [115, 116, 114] est également inefficace pour une implémentation matérielle. En effet, elle implique la génération d'un contrôleur (machine d'états) comprenant des centaines, voire des milliers d'états. Cela implique une complexité matérielle élevée ainsi que de longs chemins critiques.

Ce sont les raison pour lesquelles il n'est pas pertinent de se référer aux travaux antérieurs

2. Les positions des bits gelées sont liées à la longueur du code (N), au rendement du code (R) et à la valeur σ utilisée pour optimiser le code polaire.

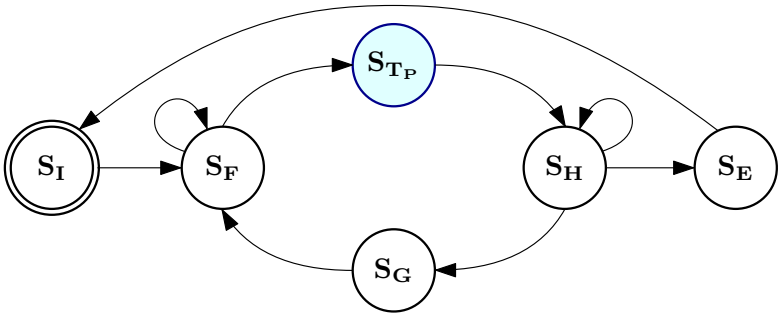

Figure 3.6 – Diagramme d'états utilisé pour le modèle comportemental sans élagage

portant sur des implémentations logicielles. Une formulation de l'algorithme SC spécifique à la synthèse de haut niveau doit être proposée pour aboutir à la génération d'architectures matérielles efficaces.

Le modèle comportemental de décodeur SC de codes polaires que nous avons développé est basé sur une machine à états finis (FSM). Ce modèle permet d'extraire les propriétés déterministes de la récursivité avant la synthèse HLS. Une représentation graphique simplifiée du diagramme d'états associé à la machine à états finis est donnée dans la Figure 3.6. Ce diagramme d'états n'intègre pas les optimisations. Le diagramme d'états est composé de six macro-États :

— L'état initial $\mathbf{S_I}$ est responsable du chargement des N LLRs provenant du système dans les mémoire internes. Les LLRs sont stockés par paquet de $\mathbf{P}$ données.

— L'état final $\mathbf{S_E}$ est en charge de l'envoie des bits décodés au système par paquet de $\mathbf{P}$ bits [3].

— Les 4 macro-états nommés $\mathbf{S_F}$, $\mathbf{S_G}$, $\mathbf{S_H}$ et $\mathbf{S_{T_P}}$ implémentent les fonctions $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$ et $\mathbf{T_p}$.

Chaque macro-état, correspondant à une partie de l'algorithme de décodage SC, inclut :

1. La gestion des pointeurs permettant de lire et d'écrire dans les mémoires λ et $\hat{U}$,
2. La gestion de la pile et des variables globales (par exemple le niveau actuel d) et le nombre de valeurs λ à traiter,
3. L'exécution des calculs $\mathbf{f}$, $\mathbf{g}$, $\mathbf{h}$ ou $\mathbf{T_p}$,
4. Le calcul des conditions de transition et la transition vers l'état suivant.

3. Il est à noter que les états $\mathbf{S_I}$ et $\mathbf{S_E}$ peuvent être fusionnés à la fin de la première exécution du décodage afin d'améliorer la latence du décodeur. Ils récupèrent et stockent les valeurs dans différents tableaux de mémorisation.

Algorithme 5: Description générique de l'algorithme SC pour le décodage de codes polaires basé sur une formulation de type FSM.

$\mathcal{S} \leftarrow \mathbf{S_I}$
$d \leftarrow n$
répéter
  si $\mathcal{S} = \mathbf{S_I}$ **alors**
    Load N LLRs from the decoder input

  sinon si $\mathcal{S} = \mathbf{S_F}$ **alors**
    Execute 2^{d-1} **f** computations on λ^d values
    si $d = 1$ **alors**
      $\mathcal{S} \leftarrow \mathbf{S_T}$
    sinon
      $\mathcal{S} \leftarrow \mathbf{S_F}$ and $d \leftarrow d - 1$
    fin si

  sinon si $\mathcal{S} = \mathbf{S_G}$ **alors**
    Execute 2^{d-1} **g** computations on λ^d values
    si $d = 1$ **alors**
      $\mathcal{S} \leftarrow \mathbf{S_T}$
    sinon
      $\mathcal{S} \leftarrow \mathbf{S_F}$ and $d \leftarrow d - 1$
    fin si

  sinon si $\mathcal{S} = \mathbf{S_T}$ **alors**
    Process 1 partial sum
    si even **alors**
      $\mathcal{S} \leftarrow \mathbf{S_G}$
    sinon
      $\mathcal{S} \leftarrow \mathbf{S_H}$
    fin si

  sinon si $\mathcal{S} = \mathbf{S_H}$ **alors**
    Execute 2^{d-1} **h** computations on $\hat{u}^d$ values
    si stack_condition **alors**
      $\mathcal{S} \leftarrow \mathbf{S_G}$ and $d \leftarrow d + 1$
    sinon
      $\mathcal{S} \leftarrow \mathbf{S_H}$ and $d \leftarrow d + 1$
    fin si

  sinon
    Send $\hat{u}^n$ values back to the system
    $\mathcal{S} \leftarrow \mathbf{S_I}$
  fin si
fin répéter

D'un point de vue implémentation, tous les macro-états nécessitent au minimum un cycle d'horloge lors de leur exécution. Par exemple, le macro-état $\mathbf{S_F}$ comprend $\frac{N}{2^{n-d+1}}$ calculs de type $\mathbf{f}$ lorsqu'il est appliqué au niveau d. Idéalement $\mathbf{\Psi}(\mathbf{S_F}) = \frac{N}{\mathbf{P}\times 2^{n-d+1}}$ cycles d'horloge sont requis si nous considérons $\mathbf{\Psi}(\mathbf{f}) = 1$. La durée d'exécution du macro-état $\mathbf{\Psi}(\mathbf{S_F})$ dépend du niveau de l'arbre (d) dans lequel le processus du décodage se situe et du nombre de PE disponibles ($\mathbf{P}$). Le principe est le même pour les états $\mathbf{S_G}$ et $\mathbf{S_H}$. La gestion des pointeurs, le compteur de niveau et la transition vers le prochain état de la FSM nécessitent quelques cycles d'horloge supplémentaires lors de l'exécution.

Le modèle comportemental du décodeur explicité dans l'Algorithme 5 a été décrit en langage SystemC. Un module qui encapsule la description dans un processus de type *sc_ cthread* a été défini. La description interne du comportement repose sur l'utilisation d'une structure de contrôle de type *switch case*. Au sein des différentes branches conditionnelles (états de la FSM), une ou plusieurs boucles itératives de type *for* sont utilisées pour décrire les calculs $\mathbf{f}$, $\mathbf{g}$ et $\mathbf{h}$.

Afin de guider l'outil Xilinx Vivado HLS lors de la génération des décodeurs au niveau RTL, des directives (*pragma*) ont été ajoutées dans le modèle comportemental. Ces directives de synthèse sont employées pour :

1. Organiser les mémoires sous forme de bancs indépendant autorisant des accès en lecture et écriture en parallèle.
2. Spécifier les boucles itératives des traitements {$\mathbf{F}$,$\mathbf{G}$,$\mathbf{H}$} dans les états {$\mathbf{S_F}$,$\mathbf{S_G}$,$\mathbf{S_H}$} qui doivent appliquer un traitement pipeline.
3. Identifier les fausses dépendances RAW (lecture-après-écriture). En effet, l'utilisation de la même zone mémoire pour lire et écrire les données λ génère de fausses dépendances qui empêchent l'outil d'appliquer un traitement pipeline.

Ce modèle comportemental permet la génération d'architectures conformes à l'organisation décrite dans la Figure 3.5. Toutefois afin d'améliorer les performances, il est nécessaire de compléter ce modèle pour qu'il intègre l'élagage.

3.2.3 Élagage dynamique à l'exécution

Les architectures proposées pour décoder des codes polaires intègrent de manière partielle [90] ou de manière complète [92] le principe de l'élagage de l'arbre de décodage. L'application de l'élagage permet de multiplier par 10 le débit de décodage des architectures [92, 118, 119]. Il paraît donc pertinent que notre modèle comportemental intègre de l'élagage.

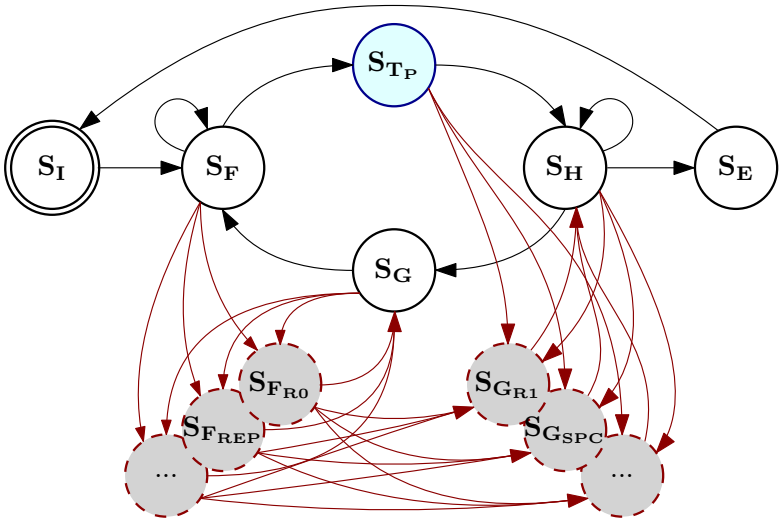

Figure 3.7 – Diagramme d'états utilisé pour le modèle comportemental intégrant l'élagage

Les architectures proposées dans la littérature [92, 118] ont besoin d'un micro-code afin de gérer l'élagage lors du processus de décodage. Malheureusement, le micro-code est généré off-line. Il est par ailleurs spécifique à un code polaire. Cela réduit la flexibilité de la solution architecturale ou bien augmente drastiquement son coût mémoire lorsque plusieurs codes polaires doivent être supportés.

Afin d'améliorer sa pertinence, le modèle comportemental présenté dans la section précédante a été amendée afin d'intégrer un mécanisme d'élagage dynamique. Ce mécanisme analyse lors de l'exécution les schémas de bits gelés présents dans le code polaire. Puis, il adapte le parcours de l'arbre pour garantir un haut niveau de performances. Ce mécanisme fonctionne sans manipulation externe et il est de plus indépendant du code polaire considéré.

L'analyse de la structure des bits gelés afin d'identifier les types d'élagage est réalisée en temps réel lors du processus de décodage. Cela permet de (1) minimiser l'emprunte mémoire du décodeur et (2) d'assurer une flexibilité vis à vis du code polaire. Lorsqu'un noeud **F** ou **G** est exécuté, la structure de bits gelés associée aux noeuds fils gauche (**F**) ou droit (**G**) est analysée en parallèle des calculs **f** ou **g**. A la fin de l'exécution de **F**/**G**, en fonction du résultat de l'analyse, un état normal {$\mathbf{S_F}$,$\mathbf{S_G}$,} ou un état d'élagage {$\mathbf{S_{F_{R0}}}$,$\mathbf{S_{G_{R1}}}$,...} est alors retenu comme prochain état de la FSM. Le nombre de bits gelés à analyser est égal au nombre de LLR à traiter au niveau de chaque noeud. C'est pourquoi, le mécanisme d'élagage n'a pas d'impact sur la latence de décodage. Un autre avantage de l'approche proposée vient de sa capacité à s'adapter aux ressources de l'architecture. Par exemple, si une architecture n'implémente qu'une seule technique d'élagage à l'aide

```
1
2  template<int Q> /* fonction de 1er niveau */
3  sc_int<Q> f_abs(const sc_int<Q> a) {
4          if( f_sign<Q>(a) == 1  ) return (sc_int<Q>)(-a);
5          else                     return (sc_int<Q>)( a);
6  }
7
8
9  template<int Q> /* fonction de 1er niveau */
10 sc_uint<1> f_sign(const sc_int<Q> llr){
11         const sc_uint<1> res = (sc_uint<1>)llr[Q-1];
12 }
13
14
15 template<int Q> /* fonction de 2nd niveau */
16 sc_int<Q> F(const sc_int<Q> a, const sc_int<Q> b) {
17     const sc_uint<1> sign1  = f_sign<Q>(a);
18     const sc_uint<1> sign2  = f_sign<Q>(b);
19     const sc_int<Q>  mag1   = f_abs <Q>(a);
20     const sc_int<Q>  mag2   = f_abs <Q>(b);
21     const sc_int<Q>  resMag = f_min <Q>(mag1, mag2);
22     const sc_int<Q>  minus  = -resMag;
23     return (sign1 != sign2) ? minus : resMag;
24 }
25
26
27 template<int P, int Q>  /* fonction de 3eme niveau */
28 sc_bigint<P * Q> VECT_F(const sc_bigint<P * Q> a, const sc_bigint<P * Q> b){
29     sc_bigint<P * Q> c;
30     for(int i = 0; i < P; i += 1)
31     {
32         const sc_int<Q> oa = (sc_int<Q>)a.range(Q * (i+1) - 1, Q * i);
33         const sc_int<Q> ob = (sc_int<Q>)b.range(Q * (i+1) - 1, Q * i);
34         c.range(Q * (i + 1) - 1, Q * i) = F<Q>(oa, ob);
35     }
36     return c;
37 }
```

Listing 3.1 – Exemple de spécification de fonctions au niveau des couches scalaires et vectorielles

d'une unité `REP`, alors seule cette optimisation est recherchée. Au final, en plus des six macro-états constituant un décodeur élémentaire (Figure 3.6), un ensemble d'états complémentaires $\{\mathbf{S_{F_{R0}}}, \mathbf{S_{G_{R1}}}, \ldots\}$ est nécessaire pour supporter les différents types d'élagage. Dans la Figure 3.7, l'ensemble des macro-états $\{\mathbf{S_{F_{R0}}}, \mathbf{S_{F_{REP}}}, \mathbf{S_{G_{R1}}}, \mathbf{S_{G_{SPC}}}\}$ représente les macro-états nécessaires aux différentes possibilités d'élagage.

3.2.4 Détails d'implémentation bas niveaux

Comme cela a été évoqué précédemment (Figure 3.4), trois niveaux d'abstraction sont nécessaires pour décrire de manière flexible le modèle comportemental permettant de générer un décodeur polaire. Quelques exemples de description issus de chacune des couches sont présentés dans le Listing 3.1. Il est à noter que les annotations (*pragma*) utilisées pour piloter l'outil HLS ont été supprimées du listing pour faciliter la lecture. Le

rôle propre des 3 niveaux est explicité ci-après :

— Le 1^{èr} niveau décrit les opérations élémentaires au niveau bit tels que l'addition saturée, la valeur absolue (Listing 3.1, lignes 2-6) et l'extraction du signe (Listing 3.1, lignes 9-12). Ce premier niveau peut être paramétré par le format des données Q (nombre de bits). De plus, ce premier niveau peut être configuré afin de manipuler des données codées en complément à deux ou bien en signe plus magnitude.

— Le 2ème niveau se base sur le 1er niveau. Il décrit les fonctions scalaires nécessaires pour implémenter l'algorithme de décodage SC (**f**, **g**, **h**, `REP`, `SPC`,...). Ces fonctions (par exemple la fonction **f** fournie dans Listing 3.1, (lignes 15-24)) sont paramétrables au niveau de la largeur des données (**Q**). Le format de représentation est quand à lui masqué

— Le 3ème niveau décrit les fonctions vectorisées mise en oeuvre dans la description du modèle de décodeur SC. Ces fonctions vectorisées sont construites à partir du 2ème niveau. Cette API décrit le parallélisme de calcul à l'aide du nombre de PE (**P**). Le 3ème niveau est paramétrable à l'aide des paramètres **P** et **Q**. Comme indiqué dans la liste 3.1 (lignes 27-37), le paramètre **P** est utilisé pour allouer et exécuter **P** fonctions de type **f** qui manipulent des valeurs de **Q** bits.

En excluant les lignes blanches et les commentaires, le modèle comportemental du décodeur SC basé sur les 3 couches d'abstraction est décrit en environ 600 lignes de code (LOC). La version du modèle intégrant l'élagage dynamique est d'environ 1.5× plus complexe. En parallèle du modèle comportemental, un fichier de paramètres contenant la définition de quelques constantes est aussi nécessaire. Les constantes suivantes, renseignées dans ce fichier, servent à configurer finement l'architecture avant la synthèse HLS :

— La taille maximale du code (**N**),

— Le format de quantification des LLRs (**Q**),

— Le niveau de parallélisme (**P**),

— La stratégie d'élagage (**T**),

— Le format de représentation des données.

A l'aide de ces paramètres, il est possible de générer des architectures variées offrant des compromis différents entre le débit, la latence, la complexité matérielle et les capacités de correction. Si certaines fonctions d'élagage sont désactivées, les parties correspondantes dans la description de la FSM sont également désactivées. L'architecture matérielle générée avec des unités PE dédiées et l'élagage dynamique sont adaptés en conséquence.

3.3 Expérimentations

Pour valider la pertinence de notre modèle comportemental et de l'approche proposée plusieurs expérimentations ont été menées. Les performances des décodeurs de codes polaires générés sont analysées. Dans cette section, nous présentons les résultats concernant le débit et la complexité matérielle. La première partie concerne l'analyse des performances absolues des décodeurs matériels et l'influence des configurations possibles dans le cadre d'une approche analytique. La seconde partie contient une comparaison des décodeurs générés avec des architectures issues de la littérature.

3.3.1 Performances absolues

Dans cette première partie, nous nous intéressons à l'impact des paramètres sur les performances des architectures générées. L'objectif est de comparer les résultats mesurés avec les performances et complexité théoriques afin de caractériser la qualité du modèle comportemental. Dans cette partie, l'outil Vivado HLS 2018.2 est employé. La cible technologique est un circuit FPGA Virtex-7 (xc7vx485tffg1761-2)

3.3.1.1 Relation entre la complexité matérielle et la longueur du code N.

Le premier paramètre générique étudié est la taille du code (N). Pour ce faire, des décodeurs ont été générés pour des valeurs de N variant de $N = 2^{10}$ à $N = 2^{15}$. Les autres paramètres du modèle ont été définis tels que $\mathbf{Q} = 8$ bits et $\mathbf{P} = \{8, 16\}$ sans élagage dans un premier temps pour simplifier l'analyse. Les résultats au niveau de la complexité matérielle sont récapitulés dans la Figure 3.8.

La complexité matérielle en slices varie légèrement (maximum 20%) en fonction de N et cela lorsque $\mathbf{P} = 4$ ou $\mathbf{P} = 16$. Cela s'explique par le fait que N a un impact sur la taille des compteurs d'adresses. La complexité de la mémoire (Mem) évolue théoriquement de manière linéaire en fonction de N, $Mem = 2 \times \mathbf{Q} \times N + N$ bits. Nous observons une augmentation du nombre de BRAMs allouées lorsque N augmente. Cependant, cette variation n'est pas linéaire lorsque $N < 2^{14}$. L'organisation interne des blocs mémoires dans les FPGA Xilinx [120] explique ce comportement. Un bloc mémoire peut stocker jusqu'à 36 kbits d'informations. Sa largeur est de 72 bits au maximum. Ainsi, lorsque $N = 2^{10}$ l'utilisation du nombre BRAM est limité. Le taux d'utilisation croit tant que $N < 2^{14}$. Si $N \geq 2^{14}$ alors il est nécessaire d'instancier de nouveaux blocs mémoires. La différence dans l'utilisation des blocs BRAM entre $\mathbf{P} = 4$ et $\mathbf{P} = 16$ est liée à la taille des

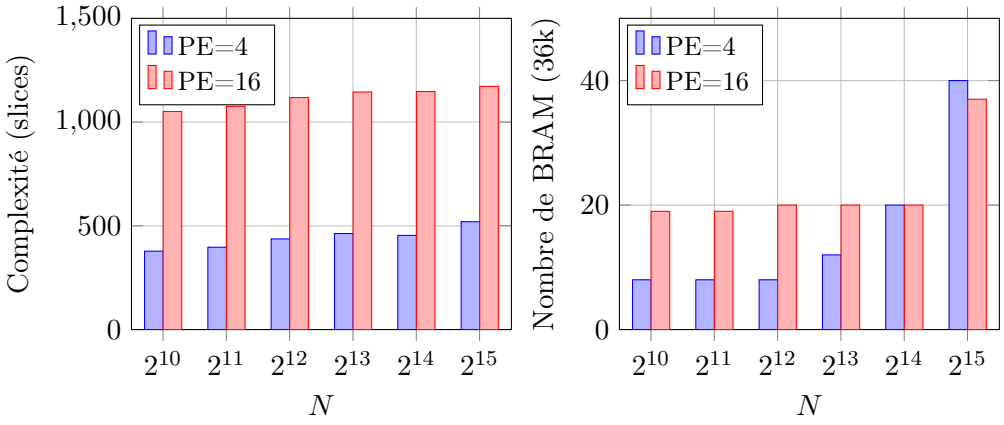

Figure 3.8 – Complexités matérielles des architecture générées en fonction de la longueur de mot de code N. Le FPGA ciblé est un Xilinx Virtex-7 avec une contrainte de fréquence à 100 MHz

bus mémoire. En effet, $\mathbf{P}$ n'influence pas l'empreinte mémoire du décodeur mais a un impact sur la largeur du bus mémoire ($\mathbf{Q} * \mathbf{P}$ bits) et sur le nombre de mots ($\frac{2\times\mathbf{N}}{\mathbf{P}}$ bits) à stocker.

3.3.1.2 Influence de P sur la complexité matérielle et les performances

Nous avons ensuite étudié l'influence de $\mathbf{P}$ sur la complexité matérielle et les performances des architectures produites. Différentes architectures ont été générées pour $\mathbf{P} \in [2,64]$ avec $N \in \{1024,32768\}$. Les résultats au niveau de la complexité matérielle sont donnés dans la Figure 3.9. Nous observons que lorsque $\mathbf{P}$ double la complexité matérielle est multiplié par un facteur supérieur à 2. Cela provient de l'implémentation de l'opérateur $\mathbf{T_p}$ nécessite l'allocation d'un décodeur SC déroulé de taille $\mathbf{P}$. La complexité $\mathcal{C}(\mathbf{T_p})$ de l'opérateur $\mathbf{T_p}$ évolue selon l'équation :

$$\mathcal{C}(\mathbf{T_p}) = 2 \times \mathcal{C}(\mathbf{T_{\frac{p}{2}}}) + \frac{\mathbf{P}}{2} \times \mathcal{C}(\mathbf{f}) + \frac{\mathbf{P}}{2} \times \mathcal{C}(\mathbf{g}) + \frac{\mathbf{P}}{2} \times \mathcal{C}(\mathbf{h}) \tag{3.9}$$

où $\mathcal{C}(\mathbf{T_{\frac{p}{2}}})$ est la complexité de la tuile $\mathbf{T_{\frac{p}{2}}}$ du niveau inférieur et $\mathcal{C}(\mathbf{f})$, $\mathcal{C}(\mathbf{g})$ et $\mathcal{C}(\mathbf{h})$ sont respectivement les complexités des fonctions $\mathbf{f}$, $\mathbf{g}$ et $\mathbf{h}$. L'augmentation significative de la complexité matérielle est également accompagné d'une augmentation du débit de décodage. Cette amélioration est néanmoins inférieure à un facteur 2. Les explications à ce résultat sont les suivantes :

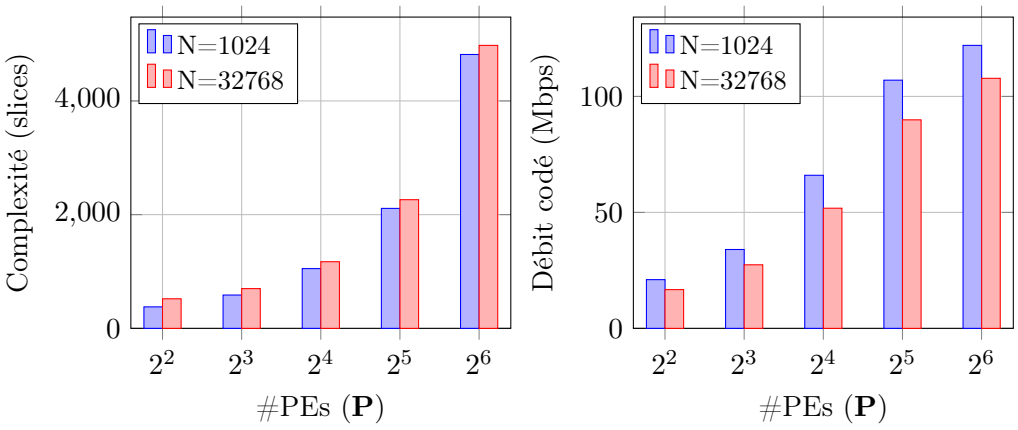

Figure 3.9 – Complexités matérielles des architectures générées en fonction du nombre de PE. Le FPGA ciblé est un Xilinx Virtex-7 avec une contrainte de fréquence à 100 MHz

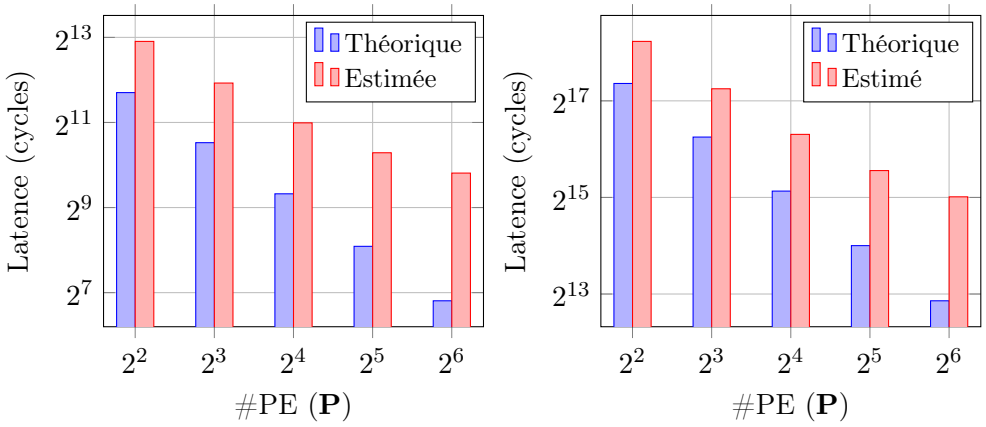

Figure 3.10 – Nombre de cycles d'horloge nécessaires pour décoder une trame en fonction du nombre de PE, (a) N = 1024, (b) N = 32768

— Plus la valeur de **P** est élevée, moins les éléments de traitement sont utilisés dans les nœuds inférieurs de l'arbre. Le parallélisme de calcul disponible est inférieur au nombre de PE.

— Lorsque **P** est multiplié par 2, le temps de calcul de l'opérateur $\mathbf{T_p}$ est multiplié par un facteur supérieur à 2. En effet la tuile $\mathbf{T_p}$ implémente de manière séquentielle : $\frac{\mathbf{P}}{2}$ fonctions **f**, une tuile $\mathbf{T}_{\frac{\mathbf{P}}{2}}$, $\frac{\mathbf{P}}{2}$ fonctions **g**, une autre tuile $\mathbf{T}_{\frac{\mathbf{P}}{2}}$, puis $\frac{\mathbf{P}}{2}$ fonctions **h**.

Pour mettre en évidence ces disparités, la latence de décodage estimée par le modèle a été comparée à la latence théorique de l'architecture. Ces informations sont présentées dans la Figure 3.10. Les résultats sont présentés pour deux valeur de N. Nous constatons que plus la valeur de **P** est faible plus les valeurs estimées sont proches de la théorie. A l'inverse plus **P** est grand plus l'écart avec la théorie est important. La latence théorique a été

calculée en considérant un cycle d'horloge pour le traitement des fonctions **f**, **g** et **h** ainsi que pour le traitement de la tuile $\mathbf{T_p}$. Or, comme cela a été mentionné précédemment il faut plus d'un cycle d'horloge pour traiter la tuile $\mathbf{T_p}$. Ce nombre de cycles nécessaires augmente lorsque **P** augmente. Le temps d'exécution de la tuile $\mathbf{T_p}$ permet d'expliquer en partie le décalage avec la latence théorique. En effet, même pour $\mathbf{P} \leq 2$ la latence estimée est supérieure à la latence théorique. Une explication plus précise de ces écarts de latence peut être obtenue en analysant le temps passé sur chaque niveau de l'arbre.

3.3.1.3 Analyse du temps passé par niveaux dans l'arbre de décodage

En théorie, le temps de traitement sur chaque niveau est constant (si aucun type d'élagage est appliqué). En effet, en descendant dans l'arbre, chaque nœud contient deux fois moins de calculs, mais il y a aussi deux fois plus de nœuds. La Figure 3.11 représente le nombre de cycles d'horloge nécessaires pour traiter l'ensemble des fonctions **f** d'un niveau. Dans cette figure, nous pouvons observer que pour les noeuds situés en haut de l'arbre ($d > 4$) le temps d'exécution est quasi constant et conforme à la théorie. Les disparités apparaissent surtout dans les couches basses ($d < 4$). Ainsi, le temps passé dans les noeuds proches des feuilles est plus élevé que ce que prévoit la théorie. Ceci s'explique par la démultiplication du nombre de noeuds en bas de l'arbre et donc une démultiplication des pénalités de traitement.

Les courbes théoriques de latence présentées ont été obtenues en considérant une architecture matérielle où le temps nécessaire à l'exécution d'un noeud $\mathcal{N}_d$ au niveau d s'exprime de la manière suivante :

$$\Psi(\mathcal{N}_d) = 3 \times \frac{N}{\mathbf{P} \times 2^{n-d+1}} \text{ cycles d'horloge} \tag{3.10}$$

où N est la longueur du mot de code, **P** le niveau de parallélisation et $n = \log_2(N)$. Ce temps d'exécution correspond au le nombre de calculs des trois fonctions **f**, **g** et **h** associé à ce noeud. Il suppose qu'un calcul est effectué en un cycle d'horloge

Or, dans la pratique les traitements des trois fonctions **f**, **g** et **h** impliquent respectivement la traversée des états $\mathbf{S_F}$, $\mathbf{S_G}$ et $\mathbf{S_H}$. La traversée d'un état s'accompagne de cycles pénalités comme expliqué plus haut. C'est pourquoi, le temps d'exécution d'un noeud est $\Psi(\mathcal{N}_d) + \delta$ cycles d'horloge où δ incluent les conditions de saut, les mises à jour des pointeurs et les pénalités associées au traitement pipeline. La pénalité de latence varie en fonction de la

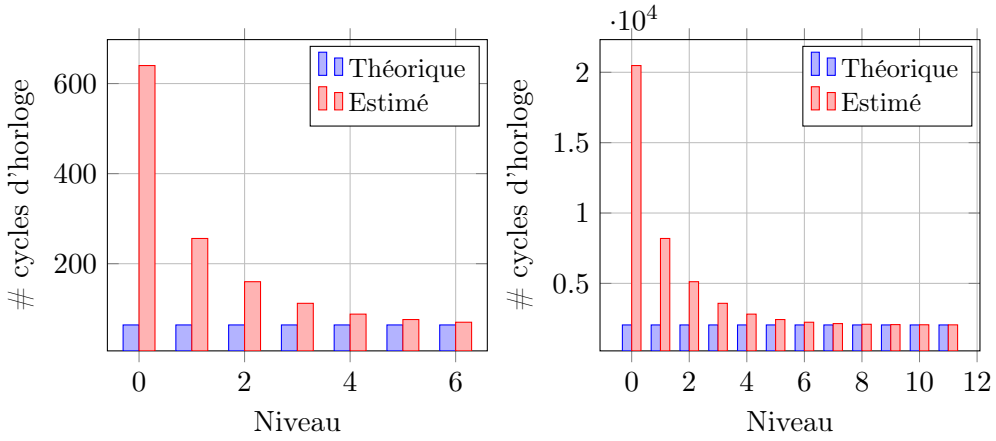

Figure 3.11 – Nombre de cycles d'horloge nécessaires pour traiter l'ensemble des fonctions **f** d'un niveau de l'arbre, $\mathbf{P} = 8$, (a) $N = 1024$, (b) $N = 32768$

fonction à exécuter. La variation s'explique par les différentes profondeurs de pipeline. L'expression de la pénalité peut se décomposer comme suit : $\delta = \delta_f + \delta_g + \delta_h$. Dans notre étude de cas, δ_f, δ_g et δ_h sont respectivement égaux à 6, 5 et 4 cycles d'horloges pour les états $\mathbf{S_F}$, $\mathbf{S_G}$ et $\mathbf{S_H}$. Il est à noter que les valeurs δ_i dépendent principalement de la fréquence d'horloge imposée lors de la synthèse HLS et du circuit FPGA retenu.

L'utilisation de la directive *pipeline* dans le modèle comportemental augmente considérablement le débit des unités de traitement (PE). Cependant, lorsque le nombre d'itérations dans les noeuds est faible ($d \to \log_2(\mathbf{P})$), les pénalités de prologue (accès aux données en mémoire et calculs initiaux) et d'épilogue (écriture des résultats en mémoire) deviennent prédominants.

3.3.1.4 Impact de l'élagage dans l'algorithme de décodage sur le débit

L'élagage permet d'augmenter le débit d'un décodeur SC [91, 92, 101]. Ce mécanisme supprime la traversée des chemins de l'arbre qui peuvent être regroupés et évalués. L'élagage implique un coût matériel supplémentaire liéé à l'instanciation d'éléments dédiés à la détection des motifs et aux calculs spécifiques associés. Afin de démontrer la flexibilité du modèle comportemental et d'évaluer les performances des décodeurs, différentes architectures ont été générées. Nous avons simplifié la lecture des résultats en ajoutant progressivement les optimisations dans le modèle comportemental en commençant par `R0`. Le modèle final intègre toutes les techniques d'élagage de base (`R0`,`R1`,`REP`,`SPC`). Les Figures 3.12 et 3.13 présentent les performances de débit et la complexité du matériel pour une taille de trame N=32768 et des rendements de codage 1/2 et 8/9.

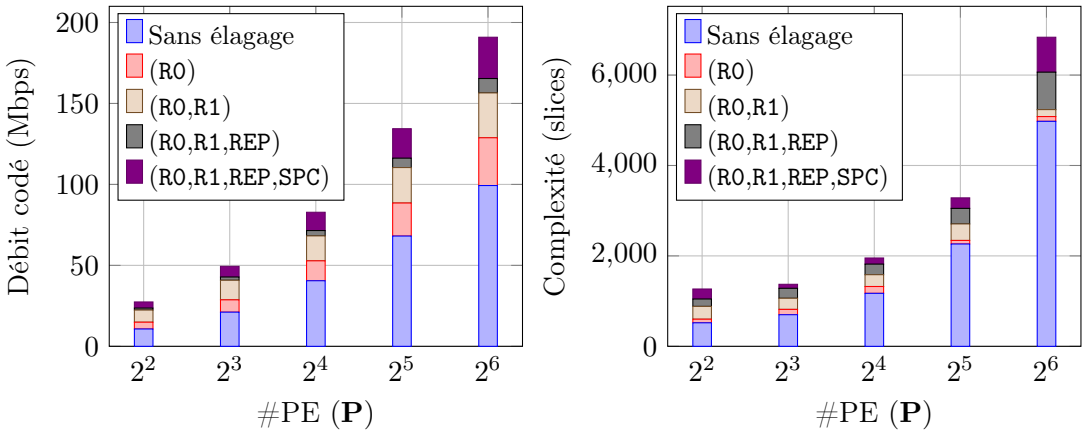

Figure 3.12 – Débit des décodeurs de code polaire en fonction du nombre de PEs et des options d'élagage ($N = 32768$, $R = 1/2$). Le FPGA ciblé est un circuit Xilinx Virtex-7 avec une fréquence de 100 MHz

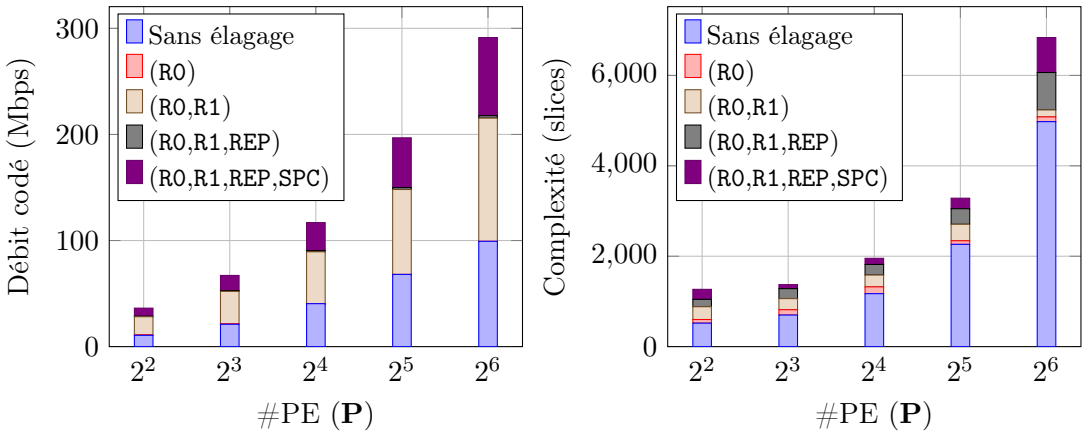

Figure 3.13 – Débit des décodeurs de codes polaires en fonction du nombre de PEs et des options d'élagage ($N = 32768$, $R = 8/9$). Le FPGA ciblé est un circuit Xilinx Virtex-7 avec une fréquence de 100 MHz

En analysant l'effet du rendement de codage nous observons que l'élagage de l'arbre permet des gains significatifs allant de 2× lorsque $R = 1/2$ à 3× lorsque $R = 8/9$. Il est intéressant de noter que pour ces deux rendements, les gains ne proviennent pas des mêmes techniques d'élagage. Par exemple, lorsque $R = 8/9$ les élagages `REP` et `R0` ne fournissent aucun gain alors que lorsque $R = 1/2$ ils induisent un gain d'environ 40%.

Le surcoût matériel introduit par les opérateurs d'élagage et les opérateurs d'analyse des motifs de bits gelés est assez faible par rapport aux gains de débit observés. Ce surcoût dépend de la complexité calculatoire de l'opérateur spécialisé d'élagage. Le coût de l'opérateur `R0` est presque nul alors que l'opérateur `REP` est plus coûteux. En effet, ce dernier comprend plusieurs calcul. Lorsque les élagages sont activés, le surcoût matériel

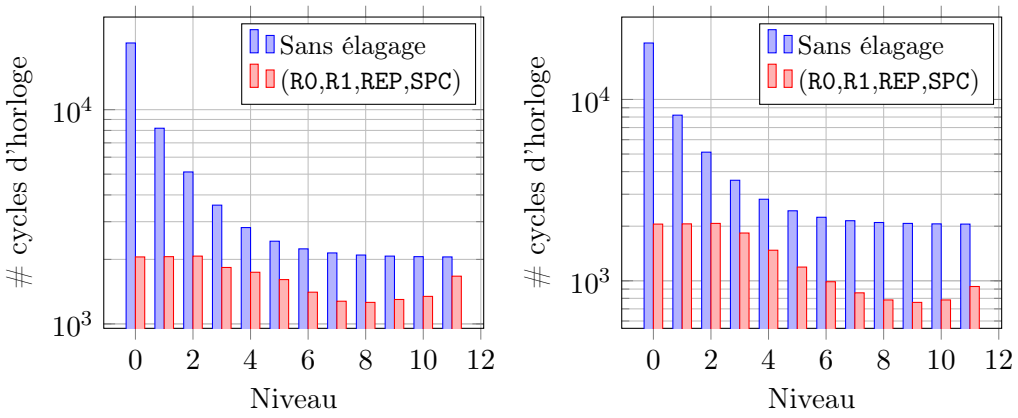

Figure 3.14 – Nombre de cycles d'horloge requis pour traiter l'ensemble des fonctions **f** selon les différents niveau de l'arbre, N=32768, (a) R =1/2, (b) R = 8/9

est d'environ 30% pour le décodeur résultant. La sélection des types d'élagage est un point clé pour s'adapter aux spécificités du code polaire à considérer. Par exemple, les élagages `R0` et `REP` sont inutiles pour des codes polaires à haut rendement.

Les techniques d'élagage réduisent le temps de traitement dans les nœuds en bas de l'arbre comme le montre la Figure 3.14.

3.3.1.5 Impact du format de quantifications des LLR

Deux autres paramètres ont des implications directes sur le compromis débit/complexité. Le premier est le format de représentation des données (complément à deux ou signe et magnitude). Le second concerne le nombre de bits à utiliser pour la quantification. Ces deux paramètres peuvent être configurés avant la synthèse dans notre modèle comportemental. L'impact de ces paramètres sur les décodeurs générés est illustré dans la Figure 3.15. Sur la cible FPGA spécifiée, le format signe + amplitude fournit un débit légèrement supérieur à la représentation en complément à deux lorsque $\mathbf{Q} = 6$ bits et $\mathbf{Q} = 8$ bits. La complexité matérielle des architectures générées est fortement impactée par le format de représentation des données. Pour une valeur identique de **Q**, le format signe + amplitude est plus efficace que le format complément à deux. Il est possible d'atteindre des gains allant jusqu'à 20% de la complexité matérielle totale. En effet, certaines opérations de l'algorithme SC sont plus adaptées à un format de représentation en signe + magnitude. C'est le cas, par exemple de la fonction **f** qui se définit par : $\mathbf{f}(\lambda_a,\lambda_b) = sign(\lambda_a.\lambda_b).\min(|\lambda_a|,|\lambda_b|)$. L'opération valeur absolue est une simple lecture d'amplitude en représentation signe + magnitude, alors qu'elle nécessite un inverseur, un additionneur et un multiplexeur en complément à deux.

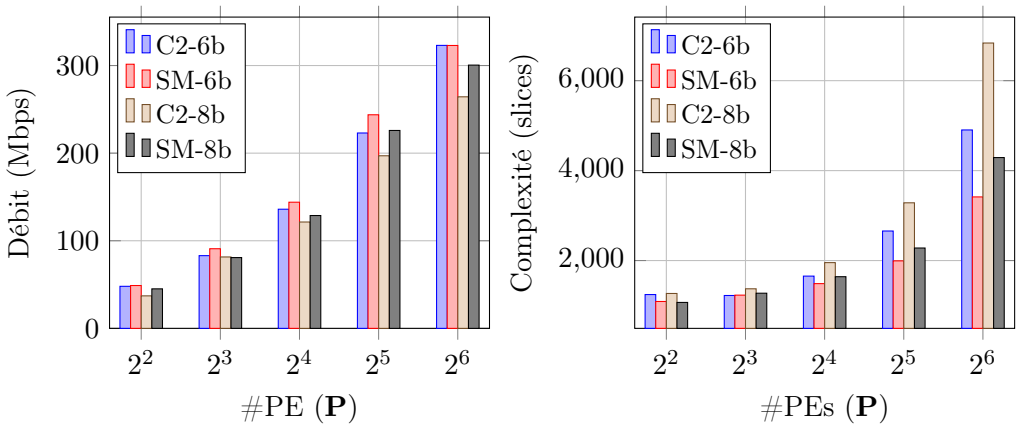

Figure 3.15 – Performances des décodeurs pour différent formats de représentation des données, élagage complet activé, $N = 32768$

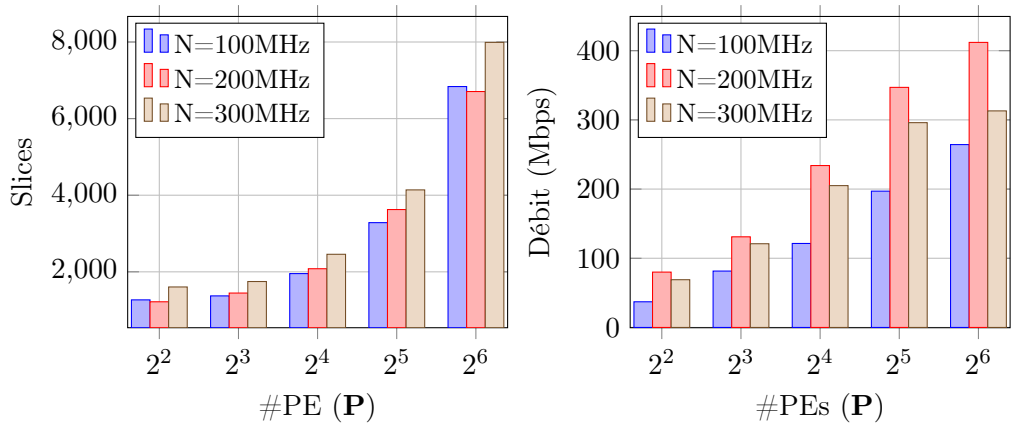

Figure 3.16 – Complexité matérielle selon les fréquences de fonctionnement spécifiées à la synthèse HLS. Le FPGA ciblé est un Xilinx Virtex-7

3.3.1.6 Influence de la fréquence de fonctionnement

La fréquence de fonctionnement ciblée est spécifiée lors de la synthèse HLS. Elle a une répercussion importante sur les performances des architectures générées. Selon la fréquence, fixée par le concepteur, l'outil HLS fera différents choix architecturaux (enchaînement et fusion d'opérations, génération de tranches de pipeline,...). Ainsi ce paramètre influence à la fois la complexité matérielle et le débit. Le changement du nombre de tranche du pipeline impacte les pénalités δ, mentionnées plus tôt. Trois fréquences de fonctionnement F={100, 200, 300} MHz ont été sélectionnées pour évaluer quantitativement l'évolution des performances. La Figure 3.16 rapporte l'évolution de la complexité matérielle et du débit. Comme prévu, la complexité matérielle des architectures générées augmente légèrement avec la fréquence. En effet, il est nécessaire d'introduire davantage de tranches

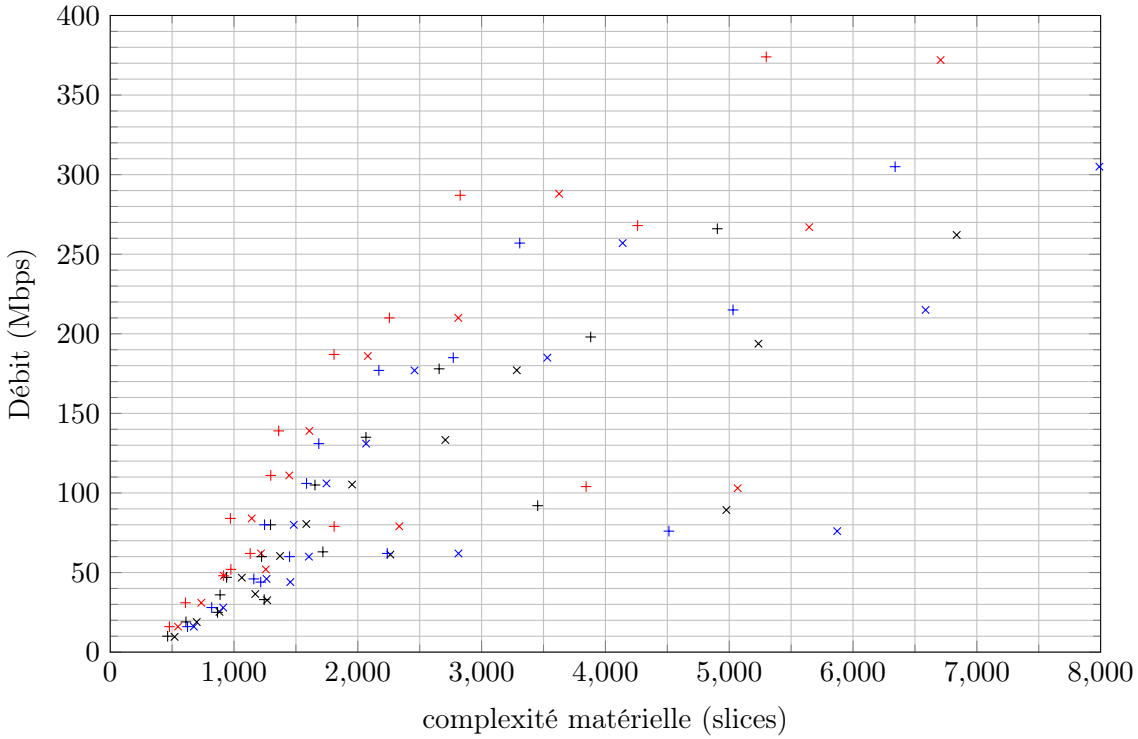

Figure 3.17 – Compromis pour les architectures de décodeurs SC avec $N = 2^{15}$. Les marques indiquent le format de quantification choisi ($\times$ = 8 bits, $+$ = 6 bits), les couleurs indiquent la contrainte de fréquence de fonctionnement retenue avant HLS (noir = 100MHz, rouge = 200 MHz et bleu = 300 Mhz)

de pipeline pour atteindre la fréquence de fonctionnement. Cependant, l'analyse du débit montre que les "meilleures" architectures sont obtenues lorsque la contrainte de fréquence est fixée à 200 MHz. Pour les architectures générées avec une contrainte de 300 MHz, les gains issus de l'augmentation de la fréquence de fonctionnement sont annihilés par l'augmentation des pénalités δ.

3.3.1.7 Compromis débit / complexité matérielle

Afin de démontrer la capacité de notre modèle comportemental à favoriser l'exploration efficace de l'espace des solutions architecturales, la Figure 3.17 présente un large spectre d'architecture générées à l'aide de contraintes spécifiques. Cette Figure répertorie des décodeurs générés avec $N = 32768$ et $R = 8/9$ avec des paramètres variables (fréquences, nombre de PEs, activation ou non de l'élagage et formats de quantification). L'analyse des résultats donne un aperçu des différents compromis débit/complexité possibles lors de la mise en œuvre d'un décodeur de codes polaires.

Tableau 3.1 – Comparaison avec des implémentations de décodeurs SC de l'état de l'art, pour un circuit FPGA Altera Stratix IV EP4SGX530KH40C2

Code Polaire			Etat de l'art							Approche proposée				
Elagage	N	Q	Ref	P	ALUT	FF	RAM bits	f MHz	Débit Mbps	ALUT	FF	RAM bits	f_{max} MHz	Débit Mbps
Aucun	2^{16}	6	[88]	64	3414	1316	821248	157	57.R	13630	5240	984064	125	124.R
Aucun	2^{15}	7	[88]	64	3927	1427	444672	153	57.R	17980	6241	557568	130	133.R
Aucun	2^{15}	8	[89]	64	4161	1629	510464	156	60.R	20053	6690	623104	125	124.R
Aucun	2^{15}	9	[88]	64	4673	1689	510720	159	59.R	22286	8195	688640	119	116.R
Semi	2^{16}	6	[90]	8	1717	868	622592	165	59.R	1551	1106	1015808	156	32.R
Semi	2^{15}	7	[90]	8	1784	899	327680	190	71.R	1558	1029	573440	148	33.R
Semi	2^{15}	8	[90]	8	2254	883	376832	173	64.R	1667	1049	638976	143	32.R
Semi	2^{15}	9	[90]	8	2029	997	360448	183	69.R	1730	1069	704512	130	29.R
Semi	2^{16}	6	[90]	16	2578	1002	622592	196	111.R	2292	1427	999424	165	72.R
Semi	2^{15}	7	[90]	16	2782	1086	327680	196	116.R	2374	1346	565248	147	69.R
Semi	2^{15}	8	[90]	16	3581	1120	376832	187	111.R	2551	1410	630784	139	65.R
Semi	2^{15}	9	[90]	16	3211	1146	360448	189	112.R	2733	1468	696320	142	66.R
Complet+	2^{15}	6	[92]	64	6830	1388	571800	108	547	17575	7900	493568	130	346

+ Le débit a été estimé pour un rendement R=9/10% comme dans [92].

3.3.2 Performances relatives par rapport à l'état de l'art

La section précédente a permis d'évaluer la flexibilité du modèle comportemental et d'analyser l'exploration architecturale selon différents jeux de paramètres. Dans cette section, nous allons comparer les architectures générées à l'aide de notre approche avec les travaux issus de la littérature.

Les architectures produites à l'aide de notre modèle sont comparées à des décodeurs décrits au niveau RTL pour des cas d'usage identiques [88, 89, 90, 92]. Les résultats obtenus sont récapitulés dans le Tableau 3.1. La première partie compare l'approche proposée avec des architectures n'intégrant pas d'élagage [88, 89, 90]. La seconde partie se concentre sur la comparaison avec des architectures intégrant partiellement ou complètement des techniques d'élagage [92].

Les travaux précédents ciblaient des FPGA Stratix IV d'INTEL. L'outil Vivado HLS génère une architecture RTL en VHDL pour des cibles FPGA de chez Xilinx. Une synthèse logique peut être réalisée avec Quartus II au prix de quelques modifications du code VHDL [4]. Il est important de noter que l'outil Vivado HLS ne génère pas d'architectures RTL optimales pour les circuits FPGA d'INTEL. En effet, la synthèse d'architecture ne s'accompagne pas d'une analyse temporelle correcte ou d'estimations de la complexité matérielle pour ces ressources. Les résultats présentés dans le tableau 3.1 sont donc les

4. Les déclarations de mémoire doivent être modifiées pour instancier les primitives BRAM de Stratix

cas les moins favorables. Pour être aussi juste que possible, les résultats de comparaison proviennent des rapports après placement-routage.

Concernant les décodeurs sans optimisation d'élagage [88, 89], les architectures générées possèdent un débit environ 2 fois supérieur et sont dans le même temps environ 4 fois plus complexes. L'approche proposée atteint un meilleur débit grâce à l'instanciation de l'unité de traitement T_p qui est non présente dans les travaux relatifs. Cependant, comme cette unité est coûteuse quand $\mathbf{P} = 64$, cela implique un surcoût matériel. En termes d'efficacité matérielle, il est à noter que les décodeurs SC générés via la synthèse d'architecture ne sont que deux fois moins efficaces.

La seconde comparaison a été faite par rapport à des architectures semi-élaguées [90] [5]. La comparaison est plus facile car le modèle comportemental conçu implémente la même stratégie d'élagage T_p. Dans ce cas, nous observons que la complexité matérielle (ALUT, FF) est similaire à celle des architectures décrites au niveau RTL, sauf au niveau du coût mémoire. Contrairement à [90], le modèle comportemental ne différencie pas actuellement le format de quantification des mémoires pour les données provenant du canal et les données internes. En ce qui concerne le débit, les performances des architectures basées sur le modèle comportemental sont environ 2 fois plus lentes que celles rapportées dans [90]. Cela s'explique par le fait que les optimisations spécifiques décrites dans [90] ne sont pas incluses dans les modèle comportemental pour des raisons de généricité et de flexibilité. Il est intéressant de noter que les architectures semi-élagués obtenues lorsque $\mathbf{P} = 16$ sont plus efficaces au niveau du débit et de la complexité matérielle par comparaison avec les résultats rapportés dans [88, 89].

Enfin, l'approche proposée est comparée au décodeur polaire SC avec élagage de [92]. Des paramètres équivalents pour N, R et σ ont été appliqués. Les décodeurs générés par synthèse d'architecture utilisent $\approx 2.5\times$ plus d'ALUTs/FFs que le décodeur polaire matériel rapporté dans [92]. Ils sont $\approx 1.5\times$ inférieur au niveau du débit. Notre modèle comportemental ne supporte pas les élagages de type `REP-SPC` et `ML` intégrées dans [92]. Cela a un impact direct sur le débit du décodeur. Par contre, le coût de mémorisation est réduit de 13%. L'approche basé sur le modèle utilise des techniques d'élagage auto-adaptatives contrairement à [92]. Cela explique en partie le surcoût matériel. Cependant, contrairement à [92] aucune mémoire supplémentaire n'est nécessaire pour addresser plusieurs code polaires.

Ces comparaisons avec les décodeurs polaires RTL conçus par le concepteur montrent la pertinence de l'approche d'exploration de l'espace des solutions pour produire des déco-

5. Ces architectures appliquent des optimisations d'élagage uniquement en bas de l'arbre au niveau T_p

deurs matériels efficaces. Des performances proches des architectures décrites au niveau RTL sont possibles. Il est à noter que des améliorations du modèle de décodeur SC avec d'autres optimisations algorithmiques rapportées dans [90, 92, 108] sont envisageables. En effet, il est possible de rajouter toujours plus de types pour l'élagage. Ces types d'élagages sont très spécialisés pour certains types de code polaire. De plus, une différenciation des formats de quantification des données provenant du canal et des données internes du décodeur peut être ajouté.

Conclusion

Une approche basée sur l'usage de modèles comportementaux pour explorer l'espace de conception des décodeurs SC a été décrite en détail. Ce modèle comportemental a été conçu spécifiquement à l'aide d'une description basée sur une machine à états finis. La grande diversité des paramètres algorithmiques et architecturaux du modèle comportemental permet la génération d'un large spectre de compromis architecturaux. Toutefois, des limitations qui ont été identifiées réduisent l'efficacité matérielle des décodeurs générés par rapport à ceux de l'état de l'art. Cela correspond au prix à payer pour de la flexibilité et de la généricité. Néanmoins, les performances atteintes par les architectures générées sont de même ordre de grandeur. De plus, l'approche fournit plus de flexibilité à l'exécution que les architectures de l'état de l'art grâce à un mécanisme d'élagage dynamique.

Ces travaux démontre la pertinence d'une approche basée sur de la synthèse de haut niveau pour la génération de décodeurs SC. Ce cas d'utilisation souligne aussi le fait que des compétences en matière de conception d'architectures matérielles sont nécessaires pour concevoir des architectures efficaces avec cette méthodologie.

Les travaux présentés dans ce chapitre ont fait l'objet d'une publication scientifique dans la conférence internationale SIPS, ainsi qu'une soumission dans la revue internationale TRETS (révision majeure).

4 Modèle d'architectures de décodeurs polaire SC-Liste

La génération d'architectures de décodeurs de codes polaires implémentant un algorithme SC est pertinent avec une méthodologie HLS. Il existe cependant une variante de l'algorithme SC qui apporte des contraintes supplémentaires pour la génération de décodeurs matériels. L'algorithme SC-Liste nécessite des éléments de traitement efficaces et optimisés pour atteindre des performances acceptables.

Le chapitre 4 évalue la capacité de la méthodologie basée modèle pour générer des structures de calcul complexes de manière efficace dans le cadre des décodeurs SC-Liste. La section 4.1 présente l'algorithme SC-Liste. La contribution originale est détaillée dans la section 4.2, et les résultats expérimentaux sont présentés dans la section 4.3.

Introduction

Dans le chapitre précédent, nous avons étudié la capacité des outils de synthèse HLS à implanter efficacement des décodeurs de codes polaires. Pour cela nous avons mis en oeuvre un modèle comportemental décrivant l'algorithme SC de manière appropriée. Cependant, pour des longueurs de trame courtes et modérées, les performances en termes de correction d'erreurs de l'algorithme SC en deçà de celles obtenues avec d'autres familles de codes correcteurs d'erreurs [121].

Afin d'améliorer les performances de décodage, d'autres algorithmes de décodage ont été proposés pour les codes polaires. Dans [87], l'algorithme de décodage par annulations successives en liste (SCL) est présenté. L'algorithme SC, lors du processus de décodage prend des décisions dures et ne remet jamais en question ses décisions antérieures. Contrairement à l'algorithme SC, l'algorithme SCL ne se focalise pas sur une solution unique. Lors du décodage, il conserve les **L** chemins ayant les décisions les plus fiables à chaque prise de décision. A la fin du décodage, la décision finale entre les **L** chemins retenus est basée sur la fiabilité des listes. Malheureusement, le chemin le plus probable à la fin du décodage, n'est pas toujours opportun. Afin d'améliorer la sélection du chemin lors du processus de décodage, l'algorithme de décodage SCL peut être associé à un contrôle par redondance cyclique (CRC). Cet algorithme nommé CA-SCL a été présenté dans [122]. Cette approche basée sur 2 codes concaténés améliore significativement les performances de décodage. Par conséquent, l'algorithme CA-SCL est proche de l'optimum en termes de performance de décodage pour les codes polaires de courte taille [121]. C'est la raison pour laquelle il a est appliqué aux codes polaires du standard 5G [26]. Cependant cette amélioration des performances de décodage est obtenue au prix d'une augmentation significative de la complexité calculatoire et par voie de conséquence de la complexité matérielle des décodeurs [123]. En effet, cet algorithme proche de l'algorithme SC nécessite à minima **L** fois plus de calculs et **L** fois plus de données à mémoriser qu'un décodeur SC. De plus, il nécessite des opérations de tri et de sélection supplémentaires.

Dans le chapitre précédent nous avons modélisé au niveau comportemental de manière efficace l'algorithme de décodage SC. Ce travail était une étape nécessaire en vue du développement d'un modèle pour l'algorithme SCL. En effet, comme nous allons le voir dans la suite du chapitre, l'algorithme SCL dans sa modélisation nécessite par exemple l'utilisation de fonctions de tri génériques et d'accès mémoires indirects. Ces opérations non triviales à implanter matériellement sont fonction de **L**. Ce nouveau niveau de parallélisation offre un angle d'étude supplémentaire concernant l'évaluation de la maturité des outils de synthèse de haut niveau.

Le reste du chapitre est organisé comme suit. La section 4.1 présente l'algorithme de décodage SCL, la section 4.2 se concentre sur le modèle architectural du décodeur SCL. L'approche expérimentale et les résultats sont détaillés dans la section 4.3. Enfin, une conclusion est donnée en fin de chapitre.

4.1 L'algorithme de décodage SC-Liste

Lors du décodage de codes polaires par l'algorithme à annulation successive (SC), les prises de décision dures dans les noeuds feuilles induisent une perte d'information sur les valeurs des LLR λ_i^0. Ces valeurs λ_0^i sont alors "oubliées" lors de la suite du processus de décodage. Ainsi une prise de décision erronée sur un LLR peu fiable peut entraîner des erreurs de décodage pour tout le reste de la trame. Afin de pallier à cette limitation, une variante de l'algorithme SC : l'algorithme SCL a été proposé dans [87]. L'algorithme consiste à :

— mettre à jour une métrique à partir de la valeur de λ_0^i lors de chaque décision dure,
— conserver **L** chemins de décodage avec leurs métriques propres,
— identifier les **L** chemins fiables à l'aide de métriques.

4.1.1 Présentation de l'algorithme SC-Liste

L'algorithme de décodage SC-Liste (SCL) est une extension de l'algorithme de décodage SC. Contrairement à l'algorithme SC, lors du traitement d'un noeud feuille, pour un bit d'information, les deux résultats de la prise de décision $\hat{u}_i = 0$ et $\hat{u}_i = 1$ sont considérés. Lors de la première estimation du bit $\hat{u}_0$, deux chemins sont créés :

1. le chemin qui correspond à l'estimation $\hat{u}_0 = 0$,
2. le chemin qui correspond à l'estimation $\hat{u}_0 = 1$.

Ensuite, le processus de décodage est appliqué en parallèle sur les 2 chemins résultants. Lors de l'estimation du bit $\hat{u}_1$, 2 nouveaux chemins sont à considérer. Ces derniers correspondent aux prises de décisions suivantes :

1. $\hat{u}_0 = 0$,$\hat{u}_1 = 0$,
2. $\hat{u}_0 = 0$,$\hat{u}_1 = 1$,
3. $\hat{u}_0 = 1$,$\hat{u}_1 = 0$,
4. $\hat{u}_0 = 1$,$\hat{u}_1 = 1$.

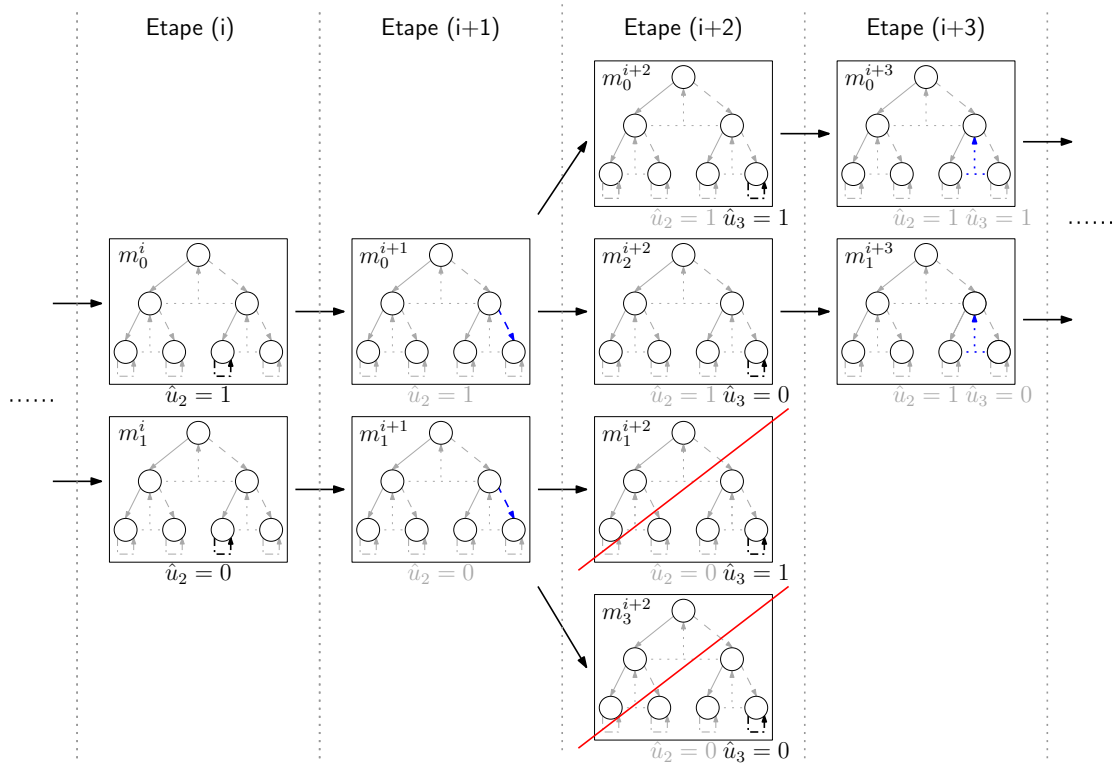

Figure 4.1 – Principe de décodage de l'algorithme SC-Liste

Le décodage est appliqué suivant le même principe en parallèle sur tous les chemins. A chaque feuille de l'arbre, le nombre de chemins est multiplié par 2 à l'exception des feuilles où le bit correspond à un bit gelé. En théorie, à la fin du processus de décodage nous avons 2^K chemins. En pratique le nombre de chemins conservé lors du décodage est limité à $\mathbf{L}$ pour éviter une explosion à la fois du coût de mémorisation et de la complexité calculatoire. Les $\mathbf{L}$ chemins conservés sont ceux possédant les plus fortes fiabilités. Pour maintenir $\mathbf{L}$ chemins tandis qu'à chaque feuille leur nombre est multiplié par 2, il est nécessaire de procéder à une étape de tri des chemins en fonction de leurs fiabilités puis de sélection des chemins à conserver. Pour ce faire, une métrique m_l est affectée à chaque chemin. Cette métrique représente son niveau de fiabilité. Le principe de l'algorithme SCL est illustré Figure 4.1, dans cet exemple $\mathbf{L} = 2$. A l'étape i le décodage possède 2 chemins :

1. pour l'estimation $\hat{u}_i = 1$,
2. pour l'estimation $\hat{u}_i = 0$.

Les métriques m_0^i et m_1^i sont affectées respectivement aux chemins 0 et 1 à l'étape i. A l'étape $(i+1)$ le décodage est exécuté en parallèle sur les 2 chemins, les métriques $m_0^{(i+1)}$

et $m_1^{(i+1)}$ demeurent inchangées. A l'étape $(i+2)$ une décision dure est appliquée, 2 nouveaux chemins sont créés à partir des 2 chemins existants. Les 4 chemins représentent les combinaisons suivantes :

(1) $\hat{u}_2 = 0$,$\hat{u}_3 = 0$, (2) $\hat{u}_2 = 0$,$\hat{u}_3 = 1$, (3) $\hat{u}_2 = 1$,$\hat{u}_3 = 0$, (4) $\hat{u}_2 = 1$,$\hat{u}_3 = 1$.

Les métriques $m_0^{(i+2)}$, $m_1^{(i+2)}$, $m_2^{(i+2)}$ et $m_3^{(i+2)}$ sont mis à jour respectivement pour les chemins numéro [0,1,2,3]. A l'issue de la mise à jour des métriques, les chemins sont triés entre eux et les $\mathbf{L} = 2$ chemins possédant la plus faible métrique sont conservés. Puis le décodage se poursuit de manière similaire pour les 2 chemins sélectionnés.

A la fin du processus de décodage, le chemin dont la métrique est la plus faible est sélectionnée comme décision à associer à la trame envoyée par l'émetteur.

4.1.2 Mise en oeuvre de l'algorithme

L'algorithme SCL possède de fortes similitudes avec l'algorithme SC. Toutefois, la gestion des listes induit des différences au niveau calculatoire et en termes de mémorisation. Tout d'abord, la mise en oeuvre de l'algorithme SCL nécessite 3 nouveaux traitements par rapport à l'algorithme SC. En effet, même si le parcours des branches de l'arbre est similaire, il est nécessaire de : mettre à jour des métriques, de trier les chemins en fonction de la valeur des métriques et de sélectionner les **L** chemins survivants. Ensuite, l'algorithme SCL implique un coût mémoire au moins **L** fois supérieur à celui de l'algorithme SC classique. En effet, **L** bancs mémoires sont nécessaires pour stocker les LLRs et les sommes partielles associés aux **L** chemins. D'autre mémoires de tailles inférieures sont aussi nécessaires afin de mémoriser par exemple les valeurs de métrique et les structures des chemins.

Du point de vue de la complexité mémoire, le coût d'implantation de l'algorithme SCL est nécessairement supérieur d'un facteur **L**. En revanche, le coût matériel lié aux **P** unités de traitement peut soit :

- augmenter légèrement si seules de nouvelles unités de traitement sont ajoutées. Dans ce cas, le débit du décodeur sera divisé au minimum par un facteur **L** par rapport aux performances d'un décodeur SC équivalent.
- augmenter fortement si en plus de nouvelles unités de calculs, le nombre d'opérateurs usuels (**f**, **g**, **h**) est multiplié par **L** afin de prendre en compte l'augmentation de la complexité calculatoire. Les performances en termes de débit devraient alors être assez proche de celles d'un décodeur SC.

Ces estimations théoriques des performances atteignables ne sont cependant valables que si le modèle comportemental développé est efficace et bien interprété par l'outil de synthèse HLS. Nous allons voir dans les prochaines sections comment les traitements supplémentaires liés à l'implantation de l'algorithme SCL peuvent être mise en oeuvre.

4.1.2.1 Mise à jour des métriques

La première différence majeure entre les algorithmes SC et SCL est liée à la gestion des listes de décodage dans l'algorithme SCL. Une métrique de fiabilité est associée à chacune des listes. Lors du décodage d'un bit dans les feuilles de l'arbre, ces métriques doivent être mises à jour.

La fonction de mise à jour des métriques dépend de la nature du bit décodé. Si le bit à la position i est un bit gelé alors les $\mathbf{L}$ métriques sont mises à jour selon la formule suivante :

$$m'_l = \begin{cases} m_l & \text{si } \lambda_i^0 \geq 0 \\ m_l + |\lambda_i^0| & \text{sinon.} \end{cases} \tag{4.1}$$

avec m_l, la métrique du chemin numéro l, m'_l la métrique mise à jour et λ_i^0 la valeur du LLR à la position i.

Dans le cas où le bit traité est un bit d'information, $2\mathbf{L}$ nouvelles métriques sont calculées à partir des $\mathbf{L}$ métriques existantes. Les nouvelles métriques obtenues à partir des expressions suivantes :

$$m'_{l0} = \begin{cases} m_l & \text{si } \lambda_i^0 \geq 0 \text{ et } \hat{u}_i = 0 \\ m_l + |\lambda_i^0| & \text{si } \lambda_i^0 < 0 \text{ et } \hat{u}_i = 0 \end{cases} \tag{4.2}$$

$$m'_{l1} = \begin{cases} m_l + |\lambda_i^0| & \text{si } \lambda_i^0 \geq 0 \text{ et } \hat{u}_i = 1 \\ m_l & \text{si } \lambda_i^0 < 0 \text{ et } \hat{u}_i = 1 \end{cases} \tag{4.3}$$

Dans ces équations, m'_{l0} et m'_{l1} sont les métriques des 2 chemins créés à partir du chemin l. La métrique m'_{l0} représente la fiabilité du chemin m_l si l'on considère que le bit i vaut 0 tandis que m'_{l1} représente cette même information pour une décision opposée.

Dans l'algorithme de décodage SCL, seules **L** listes sont propagées. En conséquence, suite à la génération de 2**L** listes lorsque le bit est un bit d'information, il est nécessaire de supprimer les **L** listes les moins fiables. Afin de simplifier la suite du document m'_{l0} et m'_{l1} serons notés : m_l et $m_l + \gamma_l$. m_l représentera la métrique de la liste qui a pris a priori la décision la plus probable. $m_l + \gamma_l$ représentera le métrique de la liste qui a été pénalisée.

4.1.2.2 Tri des métriques

Afin de supprimer les **L** listes les moins fiables [124] lors du décodage des bits d'information, il est nécessaire de trier les métriques de manière croissante. En conséquence un module de tri doit être présent. Cet opérateur de tri opère sur les données issues de la fonction de mise à jour.

Selon la nature du bit qui vient d'être décodé deux cas sont possibles : **L** ou 2**L** métriques doivent être triées. En sortie de la fonction de tri, seuls les **L** plus faibles métriques sont retenues car elles correspondent aux chemins les plus probables. Cette opération de tri doit être réalisée pour chaque bit décodé, il est donc nécessaire que ce traitement s'effectue avec une latence faible.

Cependant comme cela a été mentionné dans la partie 4.2, les opérateurs de tri rapide possèdent une forte complexité calculatoire et une forte latence du point de vue architectural surtout pour une taille de liste **L** élevée ($\mathbf{L} \geq 4$).

Dans la littérature, les modules de tri sont considérés comme le goulot d'étranglement lors de l'implantation de décodeurs SCL efficaces [124, 125, 126]. Ainsi une attention particulière doit être apportée à l'optimisation de cette fonction en vue d'une implémentation matérielle performante.

4.1.2.3 Sélection des chemins

En sortie du module de tri les **L** métriques les plus faibles sont retournées. Dans le cas où un bit d'information est traité, il est nécessaire de sélectionner les **L** chemins les plus probables parmi les 2**L** chemins créés.

Les **L** chemins les plus fiables peuvent être d'anciens chemins ou bien des nouveaux. Cela est illustré dans la Figure 4.1. Dans cette exemple à l'étape $(i + 2)$ les $\mathbf{L} = 2$ chemins les plus fiables sont tous deux issus de l'ancien chemin numéro 0. Dans ce cas il faut supprimer l'ancien chemin numéro 1 pour pouvoir mémoriser les nouveaux chemins numéro 0 et 2. Deux approches permettent de mémoriser les chemins sélectionnés et de supprimer les

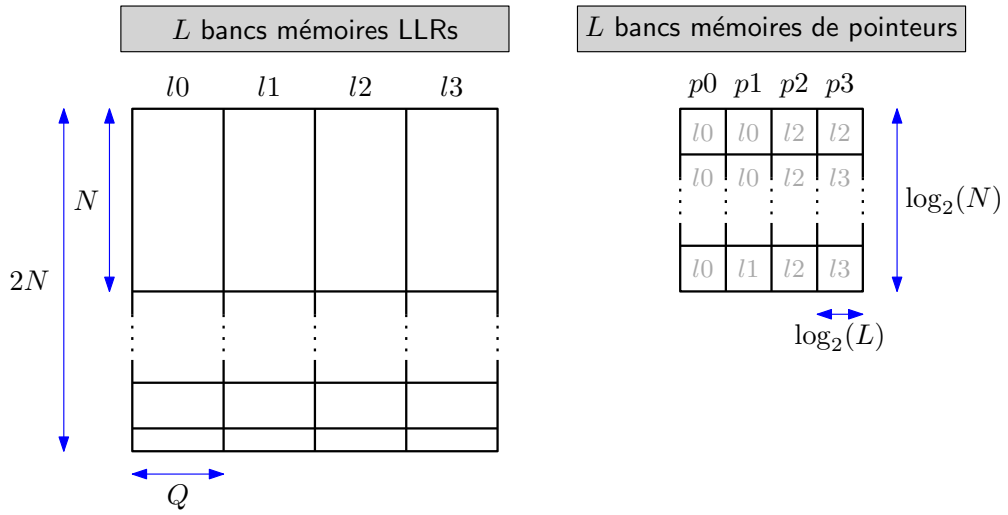

Figure 4.2 – Implémentation des mémoires LLRs des **L** chemins et de leurs tableaux de pointeurs associés

autres :

1. La première approche consiste à effectuer des recopies mémoires. Les mémoires contenant les valeurs des LLRs et des sommes partielles des chemins jugés inutiles sont remplacées par les valeurs issues des nouveaux chemins. Pour chaque liste, la taille de la mémoire des LLRs est de N éléments. Ces derniers doivent être mis à jour après chaque prise de décision. En conséquence, cette approche naïve s'avère inefficace.
2. La seconde approche vise à éviter des recopies mémoire incessantes. Elle est basée sur l'utilisation d'un niveau d'indirection entre les listes et les données. Cette méthode employée pour l'implantation des décodeurs matériels [127, 128, 129, 123] et logiciels [130] est basée sur l'utilisation de pointeurs comme cela est illustré dans la Figure 4.2. Un tableau de pointeurs sur les **L** bancs mémoires contenant les LLRs des **L** chemins est instancié. Un tableau de pointeurs de profondeur $\log_2(N)$ est associé à chaque chemin. Ce tableau contient un pointeur pour chaque niveau de l'arbre. Ce pointeur identifie le banc mémoire parmi les **L** disponibles qui contient les LLRs nécessaires à la réalisation des calculs élémentaires (**f**, **g** ou **h**). En conséquence, la fonction de sélection des chemins nécessite uniquement la recopie des tableaux de profondeur $\log_2(N)$ au lieu de N. Cela réduit drastiquement le temps nécessaire à l'exécution de cette opération.

Afin d'obtenir le meilleur niveau de performance, la seconde approche a été retenue pour le développement de notre modèle.

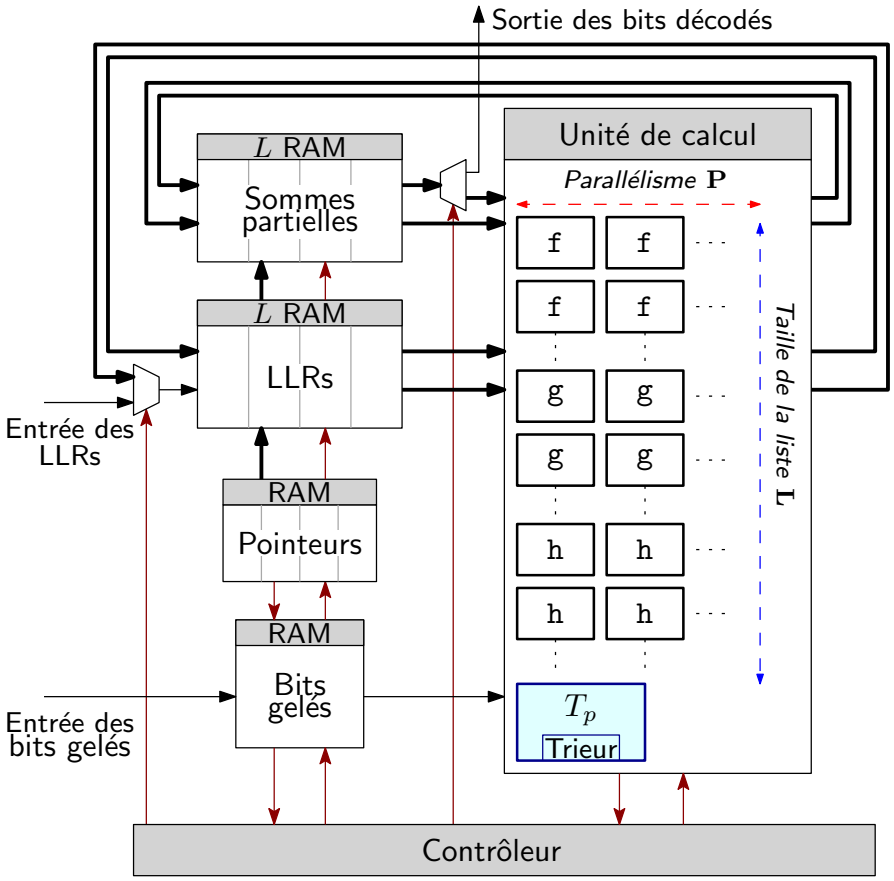

Figure 4.3 – Détails de l'architecture cible

4.2 Modèle architectural de décodeur SC-Liste

Dans cette section le modèle architectural ciblé pour l'implantation des décodeurs SCL est détaillé.

4.2.1 Modèle d'architecture cible

Le modèle d'architecture cible du décodeur SCL est basée sur celle de l'algorithme de décodage SC. L'architecture est présentée dans la Figure 4.3. Cependant, cette dernière a été amendée afin d'implanter les traitements et les mémorisations supplémentaires qui résultent de la gestion des listes. Cette architecture se compose des éléments suivants :

1. **Une unité de calcul** - Cette unité comprend les fonctions de décodage **f**, **g** et **h** ainsi qu'une tuile de traitement des noeuds feuilles $\mathbf{T_p}$. Contrairement au modèle employé pour le décodeur SC, $\mathbf{L} \times \mathbf{P}$ éléments de calcul sont instanciés pour les fonctions **f**, **g** et **h**. Cette augmentation du nombre d'unités de calcul permet de

traiter les **L** chemins en parallèle avec un facteur de parallélisme égal à **P**. De plus, le modèle de la tuile terminale $\mathbf{T_p}$ a été mis à jour. Il contient maintenant les fonctionnalités nécessaire à la gestion des métriques et des chemins.

2. **Des bancs mémoires** - Pour implanter le décodeur SCL, 4 groupes de mémoires sont nécessaires. Le premier est utilisé pour stocker les LLRs des **L** listes. Un groupe de **L** bancs mémoires de taille $2N \times \mathbf{Q}$ bits est instancié. Le second groupe est employé pour le stockage des **L** listes de sommes partielles. Chaque mémoire a une taille de N bits. De plus, une mémoire est en charge de la mémorisation des N bits gelés. Enfin, le dernier groupe de bancs mémoires stocke les $\log_2(N) \times \mathbf{L}$ pointeurs qui sont associés aux **L** différents chemins.
3. **Une unité de contrôle** - Un contrôleur pilote l'architecture et planifie les transferts de données entre les différents éléments.

L'architecture ciblée présente des similitudes avec celle utilisée pour l'implantation de décodeurs SC. L'architecture du décodeur SCL est globalement une généralisation de l'architecture SC avec un degré de parallélisation en plus (**L**). En effet, le nombre d'unités de calcul et de bancs mémoires sont multiplié par **L**. La différence majeure entre ces deux architectures provient du traitement $\mathbf{T_p}$. En effet dans le décodeur SC présenté dans le Chapitre 3, cette tuile est un sous-décodeur polaire de taille **P**. Pour des raisons de performance, ce sous-décodeur est entièrement déroulé. Il est composé de fonctions **f**, **g**, **h** et d'une unité de prise de décision dure sur LLR. Dans l'architecture du décodeur SCL, cette tuile $\mathbf{T_p}$ est plus complexe. En effet, elle contient en plus des fonctions de mise à jour, de tri et de sélection des chemins sachant que l'opération de tri présente une forte complexité calculatoire. Cette évolution implique une adaptation de la gestion de l'architecture de $\mathbf{T_p}$.

4.2.2 Tuile de traitement des feuilles $\mathbf{T_p}$

Afin d'implémenter la tuile $\mathbf{T_p}$, différents types d'architectures sont possibles en fonction du compromis entre la latence et la complexité matérielle souhaitée. Deux types de compromis sont récapitulés dans la Figure 4.4.

— La structure nommée FULLY est une architecture entièrement-déroulée. Elle contient un sous-décodeur polaire entièrement déroulé. Cette architecture permet de minimiser la latence de traitement. Cependant la tuile de niveau $\mathbf{T_{2^{n+1}}}$ instancie 2 tuiles de niveau inférieur $\mathbf{T_{2^n}}$. Il en résulte une complexité matérielle importante.

— La structure nommée SEMI est une architecture semi-déroulée. Elle contient un sous-décodeur polaire qui fonctionne séquentiellement. Cette architecture minimise

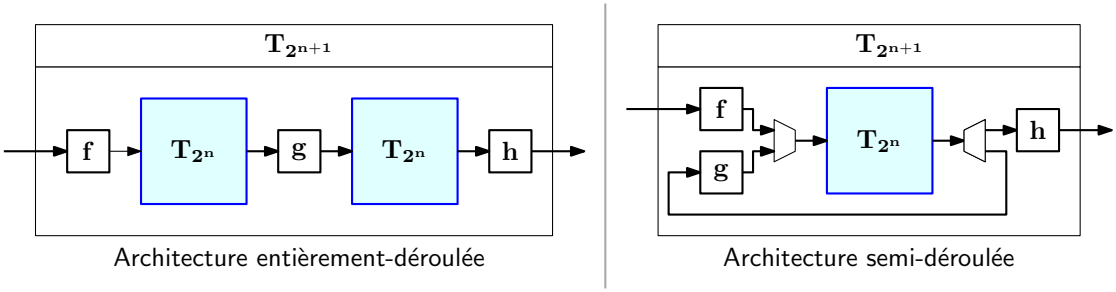

Figure 4.4 – Types d'architectures pour l'implantation de la tuile $\mathbf{T_p}$

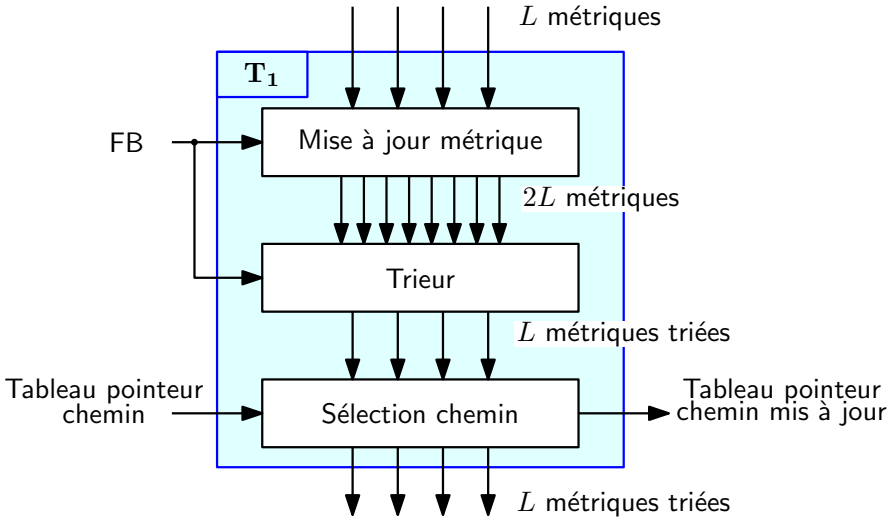

Figure 4.5 – Organisation de la tuile $\mathbf{T_1}$

la complexité en n'instanciant qu'une seule tuile de niveau inférieur. La nature séquentielle du traitement induit une latence plus importante.

Dans le cas de décodeurs SC, $\mathbf{T_1}$ ne réalise qu'une décision dure sur le LLR. Sa complexité matérielle est donc faible. C'est pourquoi, la structure FULLY pour la tuile $\mathbf{T_p}$ a été retenue. Elle permet de minimiser la latence. En revanche dans le cas des décodeurs SCL, le coût de la tuile $\mathbf{T_1}$ est plus conséquent, comme illustré dans la Figure 4.5. La tuile $\mathbf{T_1}$ contient des fonctions de mise à jour, de tri des et de sélection des chemins. L'implantation d'un opérateur de tri implique une augmentation conséquente de la complexité matérielle ($\mathbf{L} \geq 4$). Par conséquent, le choix de la structure de $\mathbf{T_p}$ n'est pas aussi trivial : l'architecture SEMI minimise la complexité, tandis que l'architecture FULLY permet de minimiser la latence.

Étant donné que la complexité matérielle de l'opérateur de tri est non linéaire tout comme la latence de traitement, il est difficile de statuer a priori entre les deux approches. Pour

cette raison, les 2 types de structures ont été implantées et comparées dans nos travaux.

4.2.3 Trieur de métriques

Dans l'architecture du décodeur SCL les caractéristiques et les propriétés de l'opérateur de tri utilisé au sein de la tuile $\mathbf{T_1}$ sont cruciales. En effet l'opérateur de tri a a priori un impact conséquent sur la latence globale du décodeur SCL ainsi que sur sa complexité matérielle. Il est donc important d'identifier une manière générique de le décrire tout en minimisant sa latence et sa complexité matérielle.

De nombreux algorithmes ont été proposés dans la littérature afin de trier des données. Dans le cadre de cette étude, 3 algorithmes de tri ont été étudiés. Le premier est un algorithme de tri classique. Les deux autres sont des algorithmes de tri spécialement mise au point pour l'algorithme SCL. Ces derniers ont été proposés pour la mise au point de décodeurs SCL matériels au niveau RTL [131]. Cependant, la possibilité de les décrire efficacement au niveau comportemental et d'assurer leur implantation efficace à l'aide de Vivado HLS doit être évaluée. Les algorithmes retenus sont les suivants :

1. Le tri à bulles [132] - Cet algorithme est une méthode de tri classique qui a pour avantage d'être générique avec $\mathbf{L}$. Il est basé sur la comparaison et la permutation séquentielle des données. Cet algorithme est rapide à mettre en oeuvre et nécessite une description comportementale concise.

2. Le tri "rank order" - Cet algorithme simplifié est une méthode de tri basée sur le calcul des positions de chacune des entrées. Une méthode simplifiée du "rank order" exploite les hypothèses sur le calcul des métriques. Cette approche est massivement parallèle mais nécessite beaucoup d'opérations et donc une complexité matérielle élevée.

3. Le tri personnalisé - Cet algorithme est une version simplifiée du tri à bulles adapté à l'algorithme SCL. Cette méthode a été proposée afin de minimiser la complexité matérielle et la latence tout en préservant la généricité de la description comportementale.

Ces 3 méthodes de tri ont été modélisées, implantées et évaluées dans la partie 4.3 en termes de performances et complexité matérielle. En effet, il est possible de comparer ces approches d'un point de vue théorique, cependant l'impact des décisions d'implantation de l'outil Vivado HLS rend ces prévisions caduques.

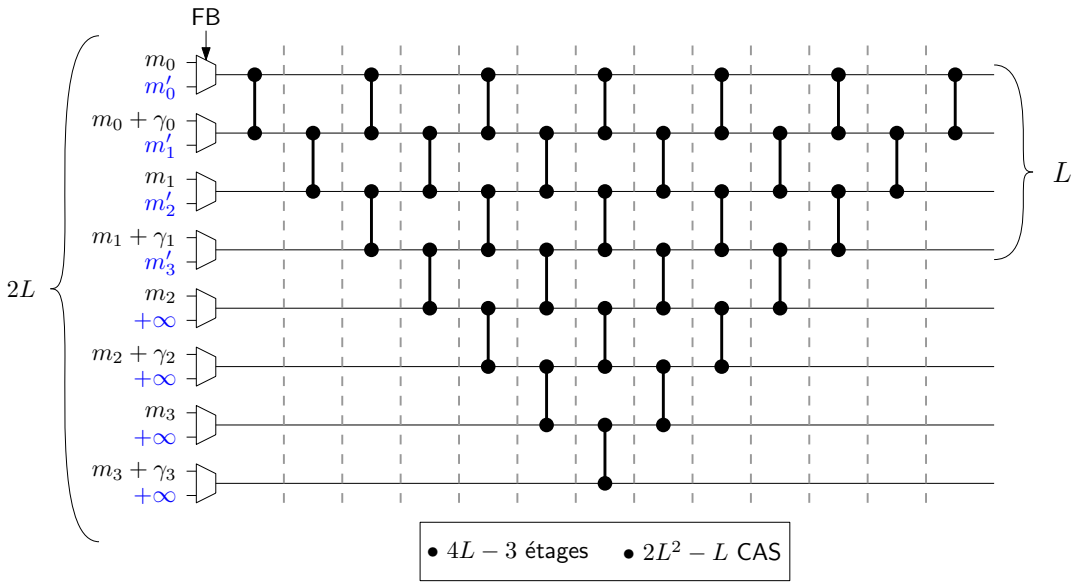

Figure 4.6 – Exemple d'architecture de tri à bulle

4.2.3.1 Tri à bulle

Le tri à bulles permet de trier $2\mathbf{L}$ données. La Figure 4.6 illustre une structure du tri à bulles permettant d'ordonner 8 données ($\mathbf{L} = 4$). Les données en entrée de la structure de tri dépendent de l'état du bit en cours de décodage (FB) :

— Lorsque le bit en cours de décodage est gelé, seules les $\mathbf{L}$ métriques m'_l mises à jour doivent être triées. Les $\mathbf{L}$ autres entrées sont fixées à $+\infty$.

— Lorsque le bit en cours de décodage correspond à un bit d'information, $2\mathbf{L}$ métriques sont à trier. Elles sont sous la forme m_l et $m_l + \gamma_l$

En sortie de la structure de tri, les métriques sont obtenues dans l'ordre croissant.

La structure de tri à bulles présentée dans la Figure 4.6 utilise des unités de comparaison et de permutation (Compare And Select, CAS). Ces dernières sont détaillées dans la Figure 4.7. L'algorithme de tri à bulles nécessite $2L^2 - L$ unités CAS qui se répartissent sur $4L - 3$ étages.

Décrire cet algorithme de tri de manière générique en fonction de $\mathbf{L}$ est assez aisé. En effet, le comportement peut se décrire de manière générique à l'aide de 2 boucles itératives *pour* imbriquées. Cela représente environ 10 lignes de code comportemental en langage *SystemC*.

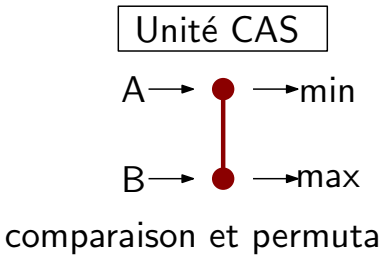

Figure 4.7 – Unité de comparaison et de permutation

4.2.3.2 Tri personnalisé

Un algorithme de tri efficace basé sur une structure de CAS doit posséder un faible nombre d'étages pour minimiser la latence et un faible nombre d'unité pour minimiser la complexité matérielle. L'algorithme de tri à bulle, ne possède aucun de ces attributs. Toutefois, les propriétés des $2\mathbf{L}$ métriques à trier dans le cas du décodage SCL rend possible une simplification de cette structure de tri. En effet, dans le cas de l'algorithme SCL deux hypothèses sont toujours vérifiées :

1. Seules les $\mathbf{L}$ plus faibles métriques sont nécessaires en sortie. Il est inutile de trier les $\mathbf{L}$ plus grandes valeurs.
2. Il est possible de conserver les chemins dans l'ordre des métriques croissant. m_0 sera la plus faible métriques et m_L sera la plus grande. Par conséquent on sait que $m_0 \leq m_1 \leq ... \leq m_L$. Cela permet de supprimer des comparaisons entre des donnée déjà connues.

Sachant ces 2 hypothèses il est possible de simplifier la structure du tri à bulles en supprimant les CAS inutiles. Comme proposé dans [124], la structure du tri à bulles tronqué pour $2\mathbf{L} = 8$ est détaillée dans la Figure 4.8.

Pour pouvoir utiliser cette structure au sein d'un décodeur SCL, il est nécessaire de pouvoir garantir l'inégalité $m_0 \leq m_1 \leq ... \leq m_L$. Or lorsque le bit correspond à un bit gelé, les métriques mis à jour m'_0, m'_1, ..., m'_L peuvent remettre en cause cette inégalité. Il est donc nécessaire de pouvoir trier les $\mathbf{L}$ métriques mis à jour m'_L. Pour pouvoir réaliser cette opération, quelques unités CAS supplémentaires s'avèrent nécessaires. Ces unités CAS implantent un tri de type "odd-even" [133] de taille $N = \mathbf{L}$ afin de trier les $\mathbf{L}$ métriques m'_L . Sa structure est donnée dans la Figure 4.8.

La structure de tri personnalisée est donc une concaténation d'une structure de tri à bulles tronquée de taille $2\mathbf{L}$ et d'une structure de tri "odd-even" de taille $\mathbf{L}$. Sachant que ces 2 structures ont de nombreuses unités CAS communes, la complexité finale de la

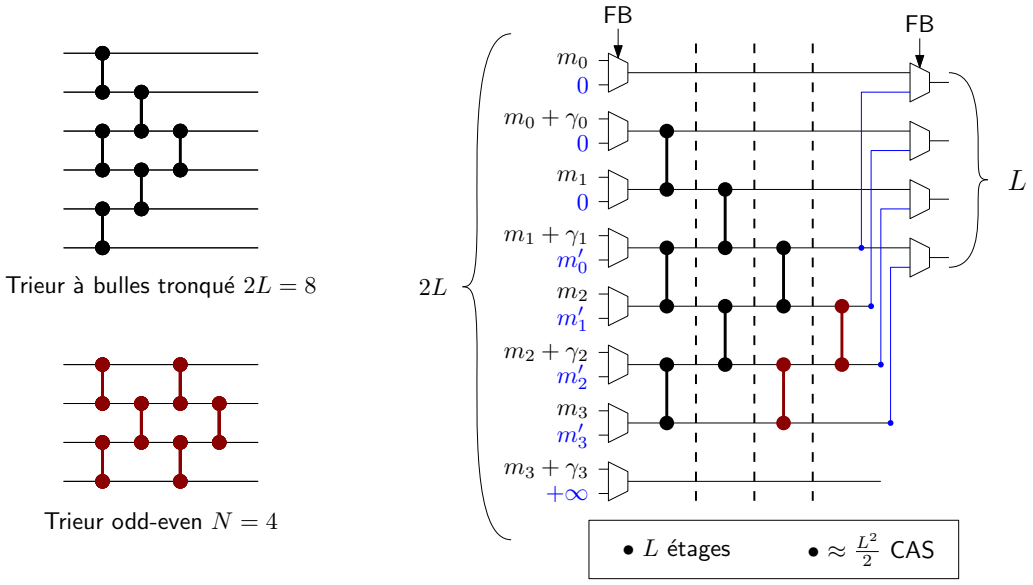

Figure 4.8 – Trieur personnalisé

structure de tri personnalisée est proche de celle de la structure de tri à bulles tronqué ($\approx \frac{L^2}{2}$ CAS). De plus, sa latence résultante est limitée à **L** étages.

La structure proposée améliore fortement les propriétés d'implantation de la fonction de tri. Cependant un multiplexage différent des entrées/sorties est nécessaire afin de l'intégrer dans la tuile $\mathbf{T_p}$. En effet, les entrées/sorties sont configurées en fonction de l'état du bit en cours de décodage (FB) :

— Si FB est un bit d'information alors les entrées sont $\{m_0, m_0 + \gamma_0, m_1, m_1 + \gamma_1, ..., m_L, m_L + \gamma_L\}$. Les **L** sorties sont les **L** sorties supérieures.

— Si FB est un bit gelé alors les entrées sont $\{0, 0, ..., 0, m'_0, m'_1, ..., m'_L, +\infty\}$. Les **L** sorties sont les **L** avant dernières sorties.

Cette méthode de tri est optimisée pour l'algorithme SCL. Le nombre d'unités CAS est réduit et le nombre d'étages nécessaires au tri est approximativement divisé par 3 par rapport à un tri à bulles classique. En revanche, la description comportementale de cet algorithme de tri est moins générique et plus complexe. En effet, une description comportementale est nécessaire pour chaque valeur de **L**. De plus, le nombre de lignes de code est élevé : environ $12 \times \mathbf{L}$ lignes de code comportemental.

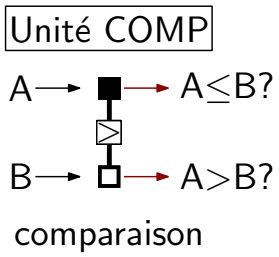

Figure 4.9 – Fonction de comparaison

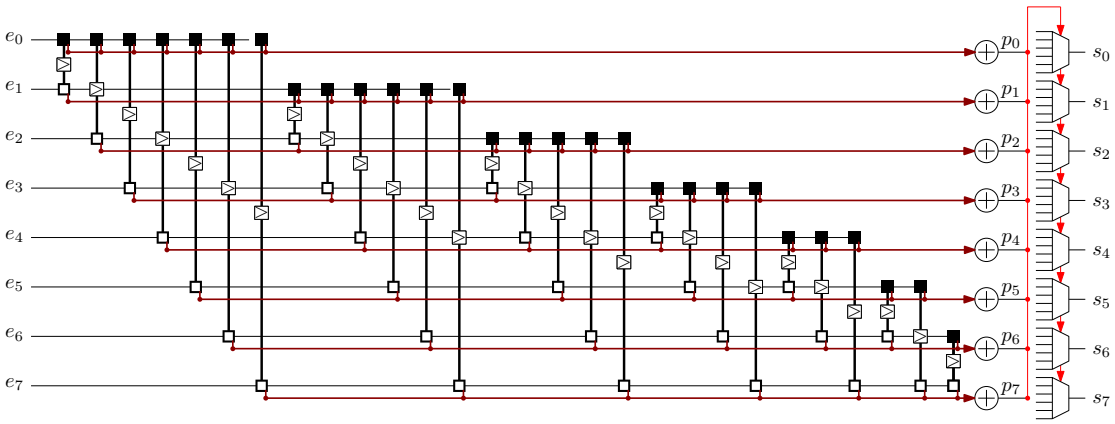

Figure 4.10 – Architecture de tri de type "Rank Order"

4.2.3.3 Tri "rank order" simplifié

Les deux algorithmes de tri présentés jusqu'à présent sont tous les deux basés sur l'utilisation d'opérations CAS. Cette structure a pour inconvénient majeur d'être séquentielle. En effet, les opérations CAS de l'étage $n+1$ doivent attendre la fin des traitements des opérations CAS de l'étage n. Par conséquent, une architecture résultante n'est pas optimale pour minimiser la latence. Il existe d'autres approches permettant de réduire la latence de traitement au prix d'une complexité calculatoire supérieure [131]. Par exemple il est possible de calculer la position de chaque entrée dans la liste triée en sortie. L'algorithme nommé "rank order" est présenté dans la Figure 4.10. Cette approche emploie des opérations de comparaison pour comparer les données 2 à 2. Ces comparaisons renvoient en sortie les résultat des comparaison $A \leq B$? et $A > B$? comme illustré par la Figure 4.9. La sortie $A \leq B$? vaut 0 lorsque A est inférieur ou égal à B et 1 lorsque A est supérieur à B. la sortie $A > B$? produit le résultat complémentaire.

Enfin, les résultats des comparaisons sont additionnés de manière à calculer les positions

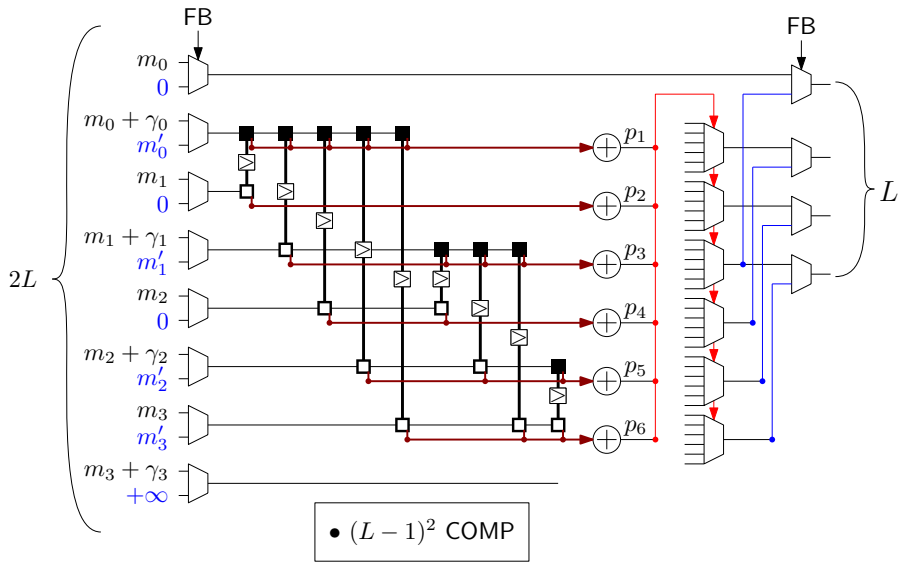

Figure 4.11 – Architecture simplifié de tri de type "Rank Order"

p_l des entrées dans la liste de sortie. La position p_l de l'entrée e_l est calculé comme suit :

$$p_l = \sum_{i=0,\ i\neq l}^{L} e_l \leq e_i? \tag{4.4}$$

Enfin une étape de multiplexage permet d'orienter les entrées e_l vers les sorties s_l adéquates en fonction des positions p_l. L'avantage d'une telle structure est de pouvoir paralléliser les étages de comparaison et d'addition. C'est pourquoi, le nombre d'étages nécessaire pour réaliser le tri des données évolue peu en fonction de **L**.

Dans le cas de l'algorithme de décodage SCL, l'architecture de tri "rank order" peut elle aussi être simplifiée en utilisant l'hypothèse $m_0 \leq m_1 \leq ... \leq m_L$. Cette version de l'architecture de tri "rank order" simplifiée est présentée dans la Figure 4.11.

Les comparaisons déjà connues ont été supprimées. Cela permet de réduire le nombre d'unités COMP à $(\mathbf{L}-1)^2$. Les entrées/sorties sont, là aussi, multiplexées différemment en fonction de l'état du bit en cours de décodage (FB) :

— Lorsque FB est un bit d'information alors les entrées sont $\{m_0,\ m_0+\gamma_0,\ m_1,\ m_1+\gamma_1,\ ...,\ m_L,\ m_L+\gamma_L\}$. Les sorties sont récupérées sur les **L** données de plus faible position.

— Lorsque FB est un bit gelé il faut multiplexer les entrées sur les comparateurs. Les entrées sont $\{0, m'_0, 0, m'_1, ..., 0, m'_{L-1}, m'_L, +\infty\}$. Les sorties sont obtenues sur les $\mathbf{L}$ avant dernières données.

Comme pour l'architecture de tri personnalisé, la description comportementale associée à un tri simplifiée de type "rank order" est complexe. Il est nécessaire de décrire pour chaque valeur de $\mathbf{L}$ un code source dédié. Chaque code nécessite environ $20 \times \mathbf{L}$ lignes.

4.2.4 Description comportementale du décodeur SCL

De manière similaire à l'approche employée pour le décodeur SC, le modèle architectural SCL a été décrit au niveau comportemental de manière générique en langage SystemC. Ce modèle comportemental implémente l'architecture de décodeur SCL présenté en Figure 4.3. Cette description générique a différents niveaux de configuration :

— Le nombre de listes $\mathbf{L}$,
— Le facteur de parallélisme $\mathbf{P}$,
— Le format de quantification des LLRs $\mathbf{Q}$,
— L'algorithme de tri utilisé dans la tuile $\mathbf{T_p}$,
— La structure de la tuile $\mathbf{T_p}$ (déroulée ou semi-parallèle).

La description du modèle générique de décodeur SCL est également basé sur une machine à états finis (FSM) avec pile, comme le modèle présenté pour les décodeurs SC au Chapitre 3. La FSM est illustré dans la Figure 4.12. La FSM se compose de six macro-états :

— L'état initial $\mathbf{S_I}$ charge les N LLRs provenant du système dans les mémoires internes. Les valeurs de LLR sont chargées par paquet de $\mathbf{P}$ données.
— L'état final $\mathbf{S_E}$ envoie les bits décodés par la liste de plus forte probabilité au système par paquet de $\mathbf{P}$ bits.
— 4 macro-états $\mathbf{S_F}$, $\mathbf{S_G}$, $\mathbf{S_H}$ et $\mathbf{S_{T_p}}$ qui implémentent respectivement les fonctions $\mathbf{f}$, $\mathbf{g}$, $\mathbf{h}$ et $\mathbf{T_p}$.

Au sein des états $\mathbf{S_F}$, $\mathbf{S_G}$ et $\mathbf{S_H}$, les $\mathbf{P} \times \mathbf{L}$ unités de calcul réalisent les traitements sur les LLRs et les sommes partielles. En conséquence, le temps de traitement dans ces états est identique à celui mesuré pour un décodeur SC. En effet, l'augmentation de la complexité calculatoire d'un facteur $\mathbf{L}$ s'accompagne d'un accroissement des ressources de calcul d'un facteur $\mathbf{L}$. La latence est donc constante. Cependant, comme cela sera quantifié dans la

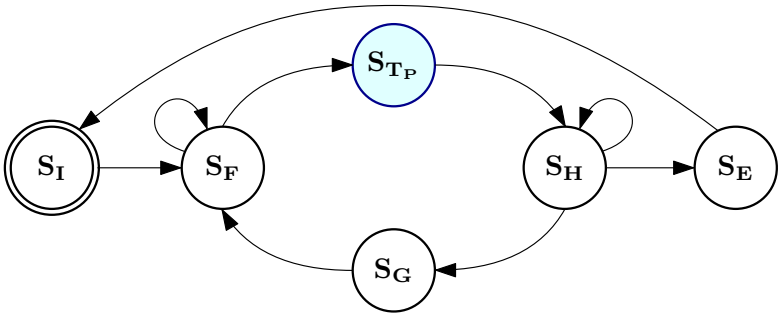

Figure 4.12 – FSM implémentée dans le modèle de décodeur SCL

section Expérimentations, il n'en va pas de même pour la latence de traitement de l'état $\mathbf{S_{T_p}}$. En effet, le temps d'exécution de la tuile $\mathbf{T_p}$ augmente fortement avec $\mathbf{L}$.

Afin de décrire la machine d'états qui implémente l'algorithme SCL, il est nécessaire d'introduire une boucle d'itérations supplémentaire sur le facteur $\mathbf{L}$ dans chaque macro-état. Comme pour le facteur de parallélisme $\mathbf{P}$, une directive HLS de type *pragma* permet de spécifier le déroulement complet de cette boucle itérative afin d'obtenir le traitement des $\mathbf{L}$ listes en parallèle.

De manière similaire au modèle de description de décodeur SC, le modèle de description du décodeur SCL repose sur trois niveaux d'abstraction complémentaires :

— Le 1er niveau décrit les opérations élémentaires au bit près. Ce premier niveau peut être paramétré par le format des données $\mathbf{Q}$ (nombre de bits).

— Le 2nd niveau se base sur le 1er. Il décrit les fonctions scalaires qui sont nécessaires pour implémenter l'algorithme de décodage SCL ($\mathbf{f}$, $\mathbf{g}$, $\mathbf{h}$). Ces fonctions sont configurables en fonction de la largeur des données ($\mathbf{Q}$).

— Le 3ème niveau décrit les fonctions vectorisées mises en oeuvre dans la description du modèle de décodeur SCL. Ces fonctions vectorisées sont construites à partir du 2nd niveau. Cette API décrit le parallélisme de calcul à l'aide du nombre de PE ($\mathbf{P}$). La 3ème couche est configurable à l'aide des paramètres $\mathbf{P}$ et $\mathbf{Q}$. Dans cette couche sont aussi décrites les fonctions spécifiques à l'algorithme de décodage SCL, telle que la fonction de mise à jour des métriques, les différentes fonctions de tri, ...

La description du modèle comportemental basé sur les 3 niveaux d'abstraction se fait en environ 600 lignes de code. Toutefois, la description des différentes fonctions spécialisées pour le décodage de l'algorithme SCL nécessite environ 10000 lignes de code. Cette augmentation du nombre de lignes de code intègre la fonctions de mise à jour des métriques, les différentes fonctions de tri (à bulle, personnalisé, rank-order) et les tuiles

$\mathbf{T_p}$. Il est nécessaire d'avoir des descriptions de la méthode de tri personnalisé et de la méthode de trie rank-order différentes pour chaque valeur du facteur **L**. Ceci explique le grand nombre de lignes de code qui sont nécessaire afin de décrire cette bibliothèque de fonctions dédiées au décodage SCL.

4.3 Expérimentations

Afin d'étudier les performances des architectures de décodeurs SCL générées à partir du modèle comportemental développé, plusieurs expérimentations ont été menées. De multiples architectures de décodeurs SCL ont été générées en fonction de différents jeux de paramètres. Les résultats sont détaillés et analysés dans la suite de cette section. La première partie de la section évalue les performances absolues des décodeurs matériels et l'influence des différents paramètres sur les architectures générées. La seconde partie positionne les architectures de décodeur générées parmi les décodeurs SCL issus de la littérature.

4.3.1 Performances absolues

Dans cette partie l'impact des différents paramètres comportementaux sur les architectures générées est étudié. Les caractéristiques des architectures générées sont comparées avec les valeurs théoriques afin de caractériser la qualité du modèle développé. Pour ce faire, l'outil de synthèse d'architecture Vivado HLS 2018.2 a été utilisé. La cible technologique choisie est un circuit FPGA de type Virtex-7 (xc7vx485tffg1761-2).

4.3.1.1 Influence du paramètre N

L'évolution de la complexité matérielle en fonction de la taille du code N a été étudiée dans un premier temps en considérant 2 tailles de liste $\mathbf{L} = 4$ et $\mathbf{L} = 16$. La tuile $\mathbf{T_p}$ est en mode SEMI, la méthode de tri est de type tri à bulles et la valeur de **P** est fixée à 1. Les résultats obtenus sont présentés dans la Figure 4.13. La complexité matérielle est exprimée en nombre de LUTs tandis que la complexité de mémorisation est rapportée en nombre de BRAMs (36k).

Les données présentées dans la Figure 4.13, montrent que le nombre de LUT utilisé est quasi constant lorsque N varie. Par contre le nombre de BRAM évolue de manière linéaire avec N. En effet ,le taille de N influe principalement sur la taille des mémoires qui doivent stocker les LLRs et les sommes partielles. Cependant, nous remarquons que pour $N = 2^{10}$ et $N = 2^{11}$ le nombre de BRAM utilisé est similaire. Ceci s'explique par la profondeur

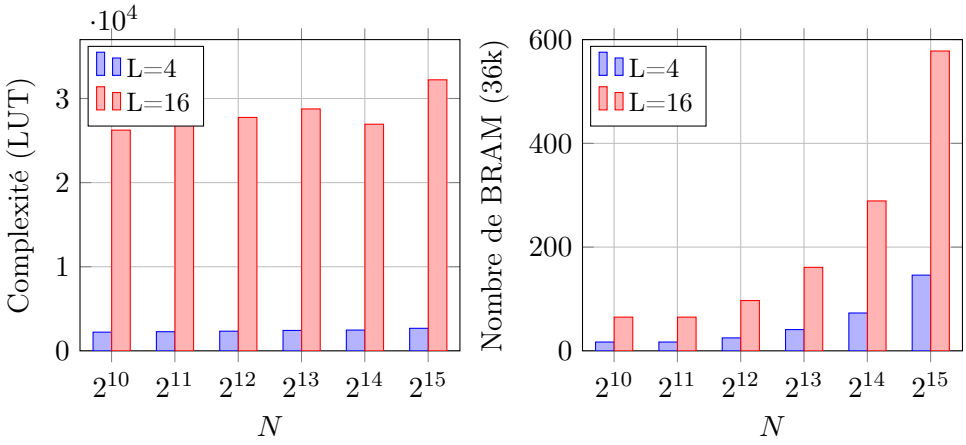

Figure 4.13 – Complexités matérielles des architecture générées pour plusieurs valeurs de N avec $\mathbf{L} = 4$ et $\mathbf{L} = 16$. Le circuit FPGA ciblé est un Xilinx Virtex-7 pour une contrainte de fréquence à 100 MHz

des BRAM qui est de 36 kbits sur un Virtex-7. Lorsque $N = 2048$ la taille des mémoires nécessaires pour stocker les LLRs d'une liste est de $2 \times N \times \mathbf{Q} = 32$ kbits avec $\mathbf{Q} = 8$, 1 BRAM est donc suffisante dans ce cas. Ce n'est plus le cas lorsque $N = 4096$ car 2 BRAMs sont s'avèrent nécessaires. Ces observations sont cohérentes avec celles issues du modèle comportemental de décodeur SC.

Le nombre de listes $\mathbf{L}$ influe à la fois sur le nombre de LUTs utilisées et sur le nombre de BRAMs utilisées. En effet, le nombre de BRAM utilisées dépend directement de $\mathbf{L}$ car les informations des $\mathbf{L}$ listes sont stockées dans $\mathbf{L}$ bancs mémoires distincts pour permettre des accès parallèles en lecture et en écriture. En parallèle, la complexité logique croit de manière significative avec $\mathbf{L}$. En effet, le nombre de listes est multiplié par 4 tandis que le nombre de LUTs utilisées est ≈ 12 fois supérieur. Cette non linéarité du surcoût s'explique par l'évolution de la complexité matérielle de la tuile $\mathbf{T_P}$ et plus particulièrement de la fonction de tri des métriques.

4.3.1.2 Influence du paramètre L et de la méthode de tri

Une seconde de série d'expérimentations vise à mettre en évidence l'impact de l'évolution du nombre de listes $\mathbf{L}$ et de la méthode de tri. En effet, le nombre de listes $\mathbf{L}$ va impacter l'architecture et les performances de la fonction de tri. Afin de réaliser cette étude la taille de la trame N a été fixée à 1024 et le format de quantification $\mathbf{Q}$ à 8 bits. Les résultats obtenus en termes de latence et de débit sont rapportés respectivement dans les Figures 4.14 et 4.15. La Figure 4.14 détaille les résultats lorsque le niveau de parallélisme $\mathbf{P} = 1$ tandis que la Figure 4.15 fournit les résultats lorsque $\mathbf{P} = 16$.

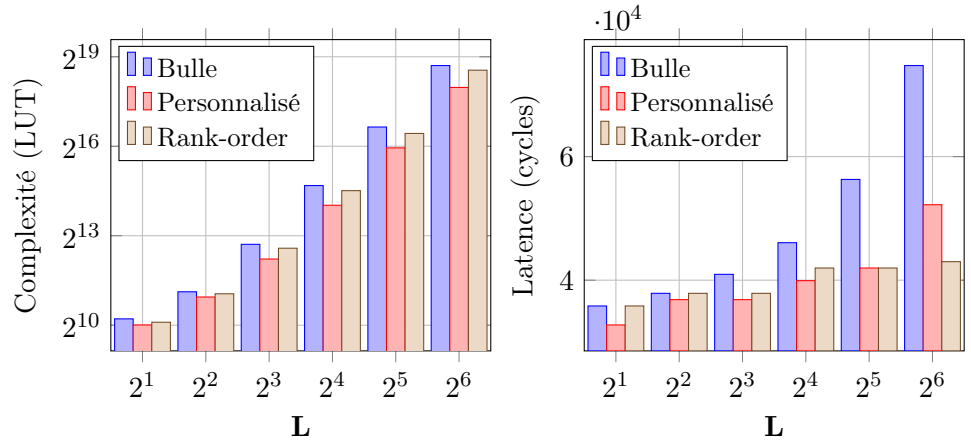

Figure 4.14 – Complexité matérielle et latence des décodeurs SCL en fonction du nombre de listes et de la méthode de tri - $N = 1024$, $\mathbf{P} = 1$ $\mathbf{Q} = 8$, $f = 100$ MHz, $\mathbf{T_p}$ =SEMI

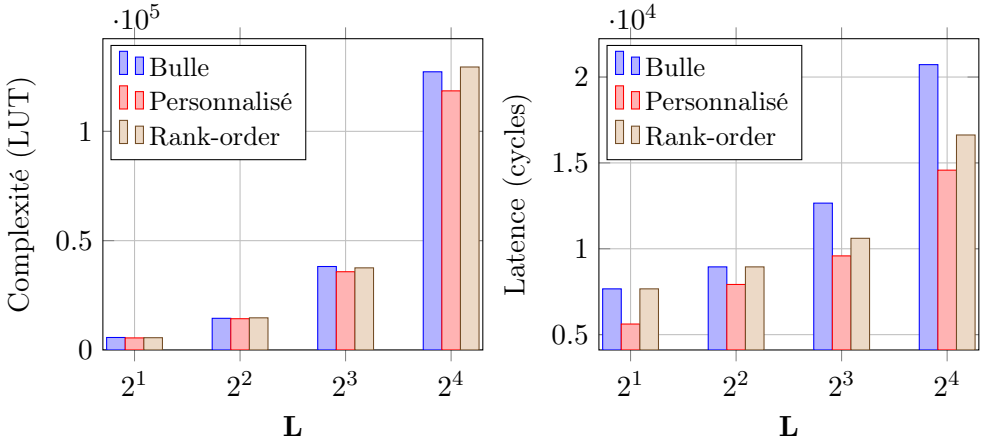

Figure 4.15 – Complexité matérielle et latence des décodeurs SCL en fonction du nombre de listes et de la méthode de tri - $N = 1024$, $\mathbf{P} = 16$ $\mathbf{Q} = 8$, $f = 100$ MHz, $\mathbf{T_p}$ =SEMI

Les résultats en termes de complexité matérielle présentés dans la Figure 4.14 lorsque $\mathbf{P} = 1$ montrent tout d'abord que le nombre de LUTs fait plus que doubler lorsque $\mathbf{L}$ double. Comme nous l'avons évoqué précédemment cela est dû à l'opérateur de tri dont la complexité évolue fortement avec $\mathbf{L}$. Cependant, nous pouvons remarquer que l'algorithme retenu pour implanter l'opération de tri impacte significativement la complexité globale du décodeur généré. L'algorithme de tri personnalisé s'avère être la solution la moins onéreuse.

Du point de vue de la latence de traitement, des différences significatives entre les architectures sont aussi observées. Tout d'abord, la latence de décodage augmente avec $\mathbf{L}$. Ensuite, les architectures utilisant l'algorithme de tri à bulles sont les moins performantes. Lorsque $L \leq 16$, les architectures utilisant l'algorithme de tri personnalisé sont les plus

efficaces. Cependant, pour $\mathbf{L} = 32$ et $\mathbf{L} = 64$, nous constatons que les architectures à base de "rank-order" ont les plus faibles latences. Ces résultats contre intuitifs vis-à-vis des performances rapportées dans [131] s'expliquent par l'incapacité de l'outil Vivado HLS à implanter efficacement les descriptions comportementales de l'algorithme rank-order. En effet, ce dernier devrait en théorie être plus onéreux mais aussi toujours plus rapide que les deux autres approches.

Les conclusions issues de l'analyse des résultats présentés dans la Figure 4.14 lorsque $\mathbf{P} = 1$ sont confirmées par celles que nous pouvons faire en analysant la Figure 4.15 ou $\mathbf{P} = 16$. Toutefois, lorsque $\mathbf{P} = 16$, le nombre de ressources matérielles nécessaires pour les fonction $\mathbf{f}$ et $\mathbf{g}$ est beaucoup plus important. Cela réduit les écarts au niveau de la complexité matérielle entre les architectures. En effet, le coût matériel qui résulte de l'opération de tri demeure lui inchangé.

4.3.1.3 Influence de la structure de $\mathbf{T_P}$

Afin d'améliorer les performances au niveau de la complexité matérielle ou de la latence, le modèle développé supporte 2 configurations pour la tuile terminale $\mathbf{T_P}$. Dans cette partie, l'influence de la structure de la tuile $\mathbf{T_P}$ sur la complexité matérielle et la latence de traitement est évaluée. La structure $\mathbf{T_P}$ entièrement-déroulée (FULLY) et la structure $\mathbf{T_P}$ semi-déroulée (SEMI) sont comparées. Les expérimentations ont été menées en configurant le modèle de la manière suivante : $N = 1024$, $\mathbf{P} = 4$, $\mathbf{Q} = 8$. Les résultats obtenus en prenant en considération deux algorithme de tri (tri à bulles et tri personnalisé) sont présentés dans les Figures 4.16 et 4.17.

Les résultats de la Figure 4.16 ont été obtenus pour des architectures implantant un tri à bulles et un facteur de parallélisme $\mathbf{P} = 4$. La seconde Figure a été obtenue pour des architectures utilisant la fonction de tri personnalisé et $\mathbf{P} = 4$. Dans un premier temps, nous observons que conformément aux prévisions, la structure $\mathbf{T_P}$ entièrement-déroulée utilise plus de LUTs que la structure $\mathbf{T_P}$ semi-déroulée. Cela se vérifie pour les deux types d'architectures de tri. En effet dans la structure entièrement-déroulée, l'opérateur de tri est instancié $\mathbf{P}$ fois au lieu d'une seule pour la structure semi-déroulée. Les architectures de décodeur basées sur la structure semi-déroulée sont 10% à 45% moins complexes que celles issues de la structure entièrement-déroulée lorsque l'algorithme tri à bulles est employé. La différence entre les architectures est limitée à 5% et à 20% dans le cas des architectures implantant le tri personnalisé.

La structure $\mathbf{T_P}$ entièrement-déroulée est plus complexe matériellement. En revanche, elle permet d'obtenir des latences plus faibles. En effet, lorsque $\mathbf{L}$ vaut 2, 4 ou 8 la latence

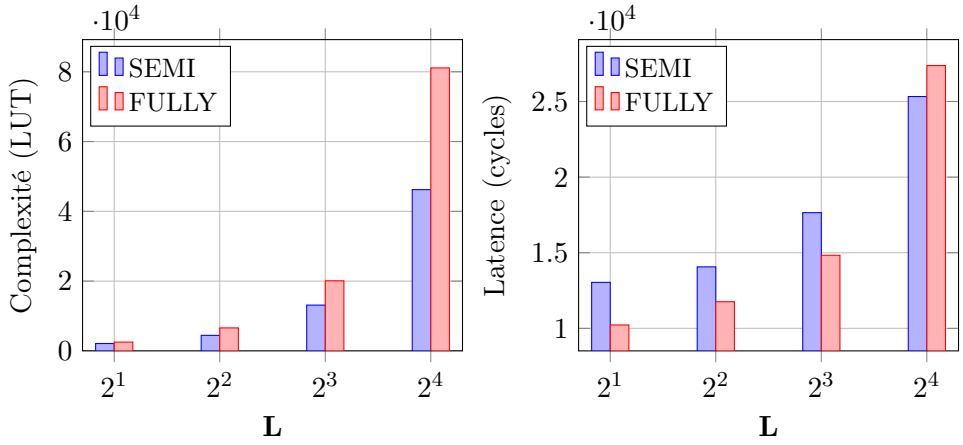

Figure 4.16 – Comparaison des architectures utilisant un $\mathbf{T_P}$ SEMI ou un $\mathbf{T_P}$ FULLY. Résultats pour les architectures implémentant la méthode de tri à bulle, $N = 1024$, $P = 4$, $Q = 8$

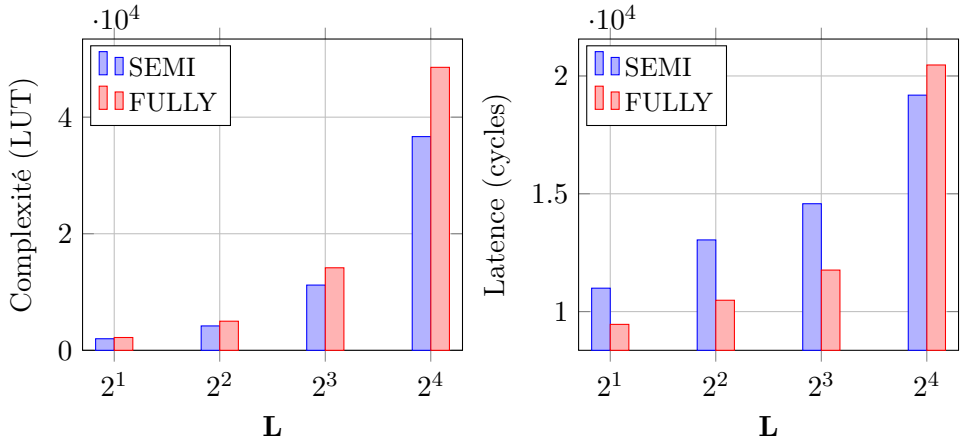

Figure 4.17 – Comparaisons des architectures utilisant un $\mathbf{T_P}$ SEMI ou un $\mathbf{T_P}$ FULLY. Résultats pour les architecture implémentant la méthode de tri personnalisée, $N = 1024$, $P = 4$, $Q = 8$

des architectures utilisant une structure $\mathbf{T_P}$ entièrement-déroulée diminue de 25% à 50% par rapport aux architectures utilisant une structure $\mathbf{T_P}$ semi-déroulée. Cependant, dans les 2 jeux d'expérimentations une anomalie lorsque $\mathbf{L} = 16$ est à noter. En effet la latence de traitement des architectures utilisant la structure FULLY devient anormalement supérieure à celle des architectures utilisant la structure SEMI. Cela s'explique par la trop grande complexité de la tuile $\mathbf{T_P}$ lorsque $\mathbf{L} = 16$ et $P = 4$. En effet, il semblerait que l'outil Vivado HLS ne parvienne pas à ordonnancer correctement les opérations en un nombre de cycles raisonnables.

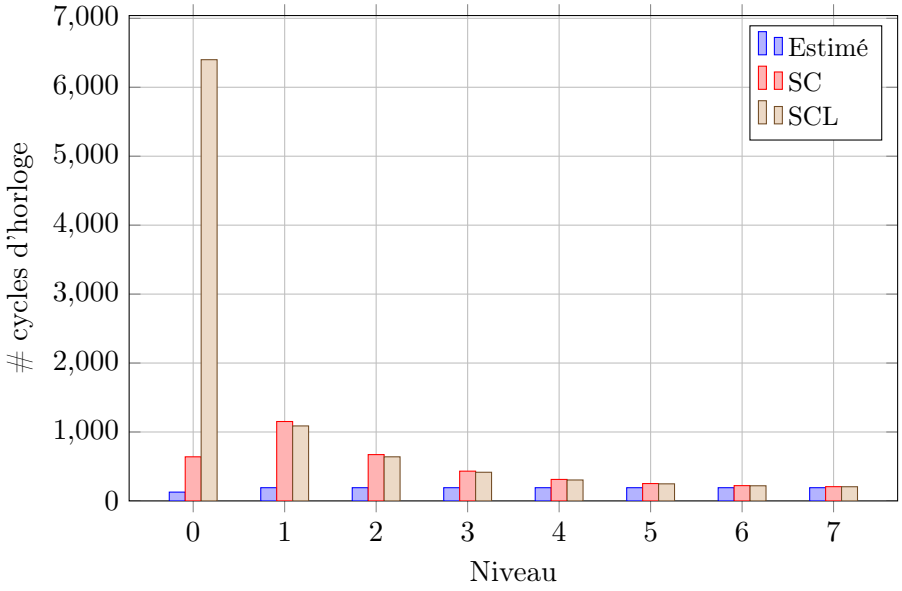

Figure 4.18 – Nombre de cycles d'horloge nécessaire pour traiter l'ensemble des noeuds d'un niveau de l'arbre, $\mathbf{L} = 4$, $\mathbf{P} = 8$

4.3.1.4 Latence par niveau

Afin de mieux comprendre, l'impact de la tuile $\mathbf{T_P}$ et du choix de l'algorithme de tri sur les performances des décodeurs générés, une étude de la répartition du temps d'exécution lors du décodage a été réalisée. Les courbes présentées dans les Figures 4.18 et 4.19 décrivent le nombre de cycles d'horloge nécessaires pour traiter chaque niveau d de l'arbre de décodage. Ces résultats ont été obtenus lorsque $N = 1024$ et $\mathbf{Q} = 8$.

La Figure 4.18 détaille le temps passé dans chaque niveau de l'arbre lorsque $\mathbf{L} = 4$ et $\mathbf{P} = 8$ pour une architecture SCL utilisant l'algorithme de tri personnalisé. Les résultats de l'architecture SCL sont comparés à ceux d'une architecture SC et à des estimations théoriques. Ces estimations théoriques ont été obtenues en considérant un cycle d'horloge pour le traitement des fonctions $\mathbf{f}$, $\mathbf{g}$ et $\mathbf{h}$ ainsi que pour le traitement $\mathbf{T_p}$. Elles correspondent donc au cas le plus optimiste. Dans les niveaux supérieurs de l'arbre de décodage le nombre de cycles d'horloge nécessaires au traitement des noeuds est équivalent à celui prédit par la théorie. Cependant, plus nous descendons dans l'arbre, plus le temps de traitement augmente pour les mêmes raisons que pour les décodeur SC. Nous remarquons que pour les niveaux de l'arbre $d \geq 1$, les nombres de cycles d'horloge nécessaires aux traitements des noeuds sont équivalents pour l'algorithme SC et l'algorithme SCL.

La différence majeure provient du temps de traitement du niveau $d = 0$. En effet c'est

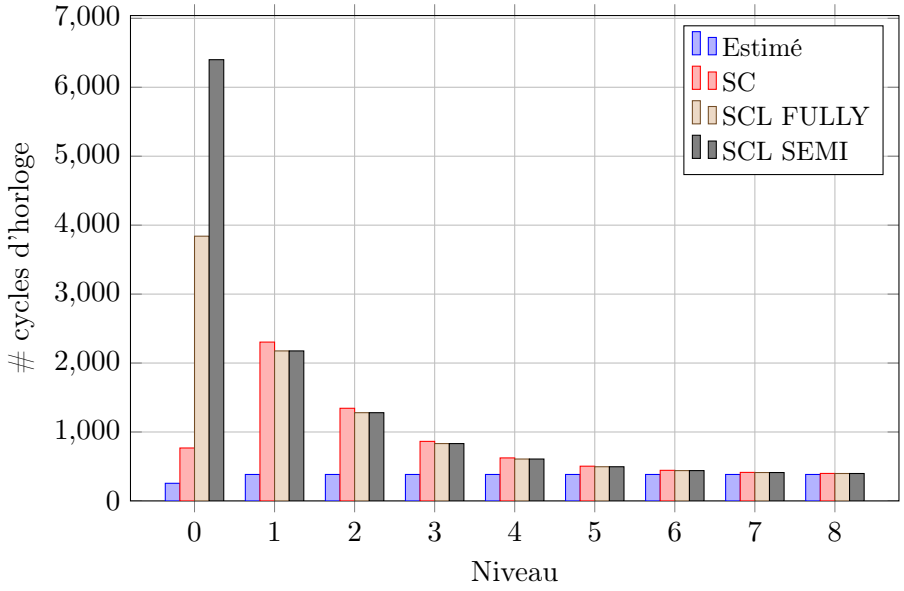

Figure 4.19 – Nombre de cycles d'horloge nécessaire pour traiter l'ensemble des noeuds d'un niveau de l'arbre, $\mathbf{L} = 4$, $\mathbf{P} = 4$

à ce niveau que les opérations de mise à jour des métriques, de tri des métriques et de sélection des chemins sont effectuées. Pour rappel, ces traitements ne sont pas nécessaires dans l'algorithme SC.

Les résultats de la Figure 4.19 obtenues pour $\mathbf{L} = 4$ et $\mathbf{P} = 4$ confirment ces observations. Les résultats sont présentés pour des architectures SCL utilisant des structures de la tuile $\mathbf{T_p}$ de type FULLY et SEMI. La structure FULLY offre une nette amélioration de la latence nécessaire pour le traitement du niveau $d = 0$. Cependant, le temps de traitement demeure grandement supérieur à celui des décodeurs SC ($\approx 5\times$).

Les résultats présentés illustrent le fait que 30% à 80% du temps de décodage est consacré uniquement au traitement du niveau $d = 0$ pour les architectures SCL. Ces courbes expliquent la dégradation importante des performances en termes de débit qu'implique le passage d'un algorithme de décodage SC à un algorithme de décodage SCL.

4.3.2 Performances relatives

Afin d'évaluer la qualité des architectures de décodeur SCL générées, nous avons tenté de positionner leurs performances par rapport aux approches proposées dans la littérature. Contrairement aux chapitres précédents traitant des décodeurs de codes LDPC ou des décodeurs de codes polaires SC où de nombreuses solutions architecturales ont été publiées, le nombre de solutions architecturales proposées pour l'implantation de décodeurs de

codes polaires SCL est relativement limité. En effet, peu de travaux sont publiées sur l'implantation de décodeurs SCL. De plus, ce nombre se réduit encore lorsque des circuits FPGA sont ciblés. Les travaux précédents se concentrent essentiellement sur la conception d'architectures pour des technologies ASIC ou sur l'exploitation d'architectures programmables. De plus, la comparaison s'avère encore plus complexe car les différentes travaux n'implémentent pas les mêmes optimisations : méthode de trie, élagage, etc.

Les seuls travaux comparables ont été publiés dans [134]. Il s'agit de l'implantation d'un décodeur SCL sur une cible FPGA INTEL Stratix 5. Ce décodeur manipule 32 listes et des trames de taille $N = 4096$ sont considérées. Le coût de l'implantation matérielle du décodeur issu de [134] est comparable à celui généré par notre modèle comportemental. Cependant les performances en termes de débit sont en retrait avec notre approche. En effet, le décodeur présenté dans [134] possède un débit de 27 Mbps tandis que celui généré par notre approche plafonne à 3 Mbps. Cet écart de performance est lié à la sous optimalité de l'opération de tri qui impacte de manière importante la latence du décodeur lorsque $\mathbf{L} \geq 4$. Des travaux complémentaires concernant la modélisation de ce traitement permettrait assurément de réduire cette différence.

Conclusion

Une extension du modèle comportemental développé pour le décodage SC des codes polaires afin de supporter l'algorithme de décodage SCL est détaillée tout au long de ce chapitre. Ce nouveau modèle permet de favoriser l'exploration de l'espace de conception de décodeurs SCL. Le modèle développé possède plusieurs niveaux de configuration afin de s'adapter aux contraintes du système. Ce modèle, en plus des niveaux de configuration déjà présent dans le modèle SC supporte aussi différentes tailles de liste, différents algorithmes de tri ainsi que deux structures pour la tuile T_p. Tous ces niveaux de flexibilité permettent de concevoir un large spectre d'architectures matérielles. Cela offre différents compromis entre débit, latence et complexité matérielle. Cependant, il est à noter que les performances obtenues par les décodeurs SCL sont en retrait par rapport aux décodeurs de type SC. Un constat similaire est fait dans la littérature [123, 130]. Toutefois, l'écart entre ces deux types de décodeurs est accentué dans notre cas par la difficulté pour l'outil de synthèse HLS à implanter efficacement l'opération de tri. Les performances actuelles des décodeurs générés à partir du modèle comportemental devront à l'avenir être améliorées en intégrant l'élagage et en réécrivant finement le code comportemental des fonction de tri.

Les travaux présentés dans ce chapitre ont fait l'objet d'une soumission dans une revue scientifique en communs avec les travaux présentés dans le chapitre 3. Ces travaux ont été

soumis dans la revue internationale TRETS (révision majeure). De plus une contribution a été apporté pour une soumission dans la revue international TCAS-II (soumis) portant sur les méthodes de trie spécifique à l'algorithme SCL.

Conclusions et perspectives

Conclusions

Ces travaux de thèse présentent une approche originale pour la conception d'architectures de décodeurs pour différentes familles de codes correcteurs d'erreurs. Une méthodologie basée sur l'utilisation de modèles comportementaux associée à l'utilisation d'outil de synthèse de haut niveau a été détaillée dans ce document. Les chapitres 2, 3 et 4 présentent respectivement des modèles architecturaux pour des décodeurs LDPC et des décodeurs de codes polaires (SC et SC-Liste).

Dans le chapitre 2, deux modèles architecturaux pour le décodage de codes LDPC sont proposés. Cette approche s'avère intéressante à plusieurs titres. Premièrement, la généricité des modèles. En effet, ces dernier peuvent supporter tout type de codes LDPC quasi cycliques, en matière de taille de code, de rendement et de matrice $\mathbf{H}$. Deuxièmement, la flexibilité des modèles : les différents paramètres architecturaux permettent une vaste exploration de l'espace de conception architectural. Troisièmement, la généricité des architectures générées : il est possible de générer des architectures matérielles pouvant supporter plusieurs variétés de codes LDPC. Quatrièmement, le flot de conception basé sur la synthèse de haut niveau permet un prototypage rapide sur circuit FPGA des architectures de décodeurs : la conception est beaucoup moins chronophage. Cette approche facilite le respect des délais de mise sur le marché. Enfin, les performances des architectures générées sont compétitives à celles des architectures conçues manuellement. Le modèle proposé aboutit en une nette amélioration de l'état de l'art pour des architectures de décodeurs LDPC générées à partir d'outils de synthèse de haut niveau.

La première génération d'architecture de décodeurs de codes polaires de type SC à l'aide d'outils de synthèse de haut niveau est décrite dans le chapitre 3. Ce travail constitue un défi scientifique car l'algorithme SC est moins adapté au regard des caractéristiques des outils de synthèse de haut niveau. Dans ce chapitre un modèle comportemental générique est proposé pour concevoir des architectures de décodeurs SC. Ce modèle est paramétrable

en fonction de la taille du code, de la matrice de bits gelés, du facteur de parallélisation, du format de quantification des données et des optimisations d'élagages retenues. Grâce au modèle comportemental une grande diversité d'architectures de décodeurs SC a pu être générée, permettant ainsi une vaste exploration de l'espace de conception. Cela montre qu'il est possible de générer différents compromis dans un délai de conception raisonnable. La performance des architectures générées est similaire à celles des précédents travaux de la littérature. Les limitations de la méthodologie employée ont pu également être identifiées et expliquées. Ce travail montre qu'en utilisant une approche de conception adaptée du modèle comportemental, il est possible d'obtenir des performances compétitives au niveau des architectures générées.

En matière de conception d'architectures matérielles de décodeurs SC-Liste sur circuit FPGA peu de travaux sont proposés dans la littérature. En effet la conception de décodeurs SC-Liste matériels s'avère ardue. Dans le chapitre 4, un modèle de décodeur SC-Liste est défini. L'intérêt de ce modèle est qu'il permet une vaste exploration de l'espace de conception pour un algorithme complexe à mettre en oeuvre. Il est montré qu'avec la méthodologie employée il est possible de générer des structures complexes et efficaces. La fonction de tri du décodeur SC-Liste peut être implantée par différentes approches et différentes valeurs de profondeur de liste grâce au modèle. Cependant les limitations identifiées pour le modèle comportemental associé à l'algorithme de décodage SC s'avèrent encore plus critique pour le modèle du décodeur SC-Liste.

Perspectives

À la suite de ces travaux de thèse, plusieurs perspectives de recherche se dégagent :

1. La première perspective est en lien direct avec les résultats du chapitre 4. Elle concerne le modèle comportemental du décodeur SC-Liste. Le modèle proposé ne permet pas actuellement d'intégrer des optimisations d'élagage comme cela a été implémenté dans la littérature comme [123]. En effet, les optimisations d'élagage sont relativement complexes à implémenter pour l'algorithme SC-Liste. Cependant, nous avons décrit dans le chapitre 3 un modèle du décodeur SC qui supporte ce type d'optimisation. Il est donc a priori possible d'implémenter des optimisations d'élagage similaires à court terme.
2. La seconde perspective concerne l'amélioration des modèles proposés, en particulier pour les décodeurs de codes polaires. Comme cela a été précisé dans les chapitres 3 et 4, des limitations issues des modèles induisent des pertes de performance pour les architectures générées. Ces limitations s'expliquent par l'incapacité de l'outil Vivado

HLS à interpréter et synthétiser efficacement certaines sémantiques des descriptions comportementales. Il en résulte des ruptures dans le flot de données, induisant des cycles d'horloge de pénalités. Comme cela a été montré dans le manuscrit, ces cycles d'horloge de pénalité peuvent dégrader fortement les performances au niveau de la latence des architectures de décodage. Néanmoins la constante évolution des outils de synthèse de haut niveau devrait permettre, à l'avenir, à la méthodologie basée sur l'utilisation de modèles comportementaux d'égaler les performances des architectures dédiées.

3. Une troisième perspective concerne plus précisément l'outil de synthèse de haut niveau employé dans le cadre de cette étude. La totalité des architectures présentées dans ce manuscrit a été générée à l'aide de l'outil Vivado HLS. Le choix de l'outil Vivado HLS a été justifié par sa capacité à supporter le langage SystemC et sa capacité à supporter des directives de synthèse. Néanmoins, les résultats présentés dans ce manuscrit sont fortement dépendants de cet outil. Durant ces travaux de thèse, de nombreuses limitations de l'outil ont été identifiées et des mesures ont dû être prises pour contourner ces limitations. Comme les résultats présentés dans ces travaux sont dépendants de l'outil, il serait intéressant d'évaluer l'approche avec d'autres outils tels que Intel Compiler [33], Catapult [31] ou encore GAUT [34].

Enfin de nombreuses pistes de recherche concernent la conception de modèles comportementaux pour d'autres familles de codes correcteurs d'erreurs. En effet, d'autres familles de codes sont utilisés dans les systèmes de communications numériques modernes par exemple les turbo codes [21].

Bibliographie

[1] G. Falcao, J. Andrade, V. Silva, and L. Sousa, "GPU-based DVB-S2 LDPC decoder with high throughput and fast error floor detection," *IET Electronics Letters*, 2011.

[2] Motorola Inc, "Motorola demonstrates portable telephone to be available for public use by 1976," 1973.

[3] European Telecommunications Standards Institute, "GSM," 2011.

[4] BBC News, "First 3G mobiles launched in Japan," 2001.

[5] TechSpot, "Everything you need to know about 4g wireless technology," 2010.

[6] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "ASIC clouds : Specializing the datacenter," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 178–190.

[7] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W. Hwu, "GPU clusters for high-performance computing," in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug 2009, pp. 1–8.

[8] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[9] W. Kleitz, *Digital Electronics with VHDL, Quartus II Version.* Prentice Hall, 2006.

[10] Xilinx, *Vivado Design Suite User Guide*, Xilinx, Jun. 2012.

[11] SYNOPSYS, *Design Compiler® User*, SYNOPSYS, Dec. 2011.

[12] W. Lie and W. Feng-yan, "Dynamic partial reconfiguration in FPGAs," in *2009 Third International Symposium on Intelligent Information Technology Application*, vol. 2, Nov 2009, pp. 445–448.

[13] NVIDIA, *NVIDIA GeForce GTX 550 Ti*.

[14] ——, *NVIDIA GeForce GTX 1080*.

[15] S. Leibson, *Designing SOCs with Configured Cores : Unleashing the Tensilica Xtensa and Diamond Cores*, Elsevier, Ed. Morgan Kaufmann Publishers, 2006.

[16] H. Corporaal and R. Lamberts, "TTA processor synthesis," in *First Annual Conf. of ASCI*, 1995.

[17] P. Hailes and et al, "A survey of FPGA-based LDPC decoders," *IEEE Communications Surveys & Tutorials*, 2016.

[18] X. Zhang, Y. Tian, J. Cui, Y. Xu, and Z. Lai, "An multi-rate LDPC decoder based on ASIP for DMB-TH," in *2009 IEEE 8th International Conference on ASIC*, Oct 2009, pp. 995–998.

[19] J. Andrade, G. Falcao, V. Silva, and L. Sousa, "A survey on programmable LDPC decoders," *IEEE Access*, 2016.

[20] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, 1948.

[21] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding : Turbo-codes," in *International Conference on Communications (ICC)*, vol. 2. IEEE, May 1993, pp. 1064–1070 vol.2.

[22] R. G. Gallager, "Low density parity check codes," *IRE Transactions on Information Theory*, 1962.

[23] E. Arikan, "Channel polarization : A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory (TIT)*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[24] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory (TIT)*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[25] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo, "AFF3CT : A fast forward error correction toolbox !" *Elsevier SoftwareX*, vol. 10, p. 100345, Oct. 2019.

[26] *Technical specification ; 5G ; NR ; Multiplexing and channel coding (3GPP TS 38.212 version 15.2.0 Release 15)*, 3GPP, July 2018.

[27] P. Lynggaard and K. Skouby, "Deploying 5G-technologies in smart city and smart home wireless sensor networks with interferences," *Wireless Pers Commun*, 2015.

[28] N. Vo, T. Q. Duong, H. D. Tuan, and A. Kortun, "Optimal video streaming in dense 5G networks with D2D communications," *IEEE Access*, vol. 6, pp. 209–223, 2018.

[29] C. Campolo, A. Molinaro, A. Iera, and F. Menichella, "5G network slicing for vehicle-to-everything services," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 38–45, Dec 2017.

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

[31] MENTOR, *Catapult High-Level Synthesis - Datasheet*, 2017.

[32] Xilinx, *Vivado Design Suite User Guide - High-Level Synthesis*, UG902 (v2017.1) ed., 2017.

[33] INTEL, *Intel High Level Synthesis Compiler - Reference Manual*, 19th ed., 2019.

[34] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin, *GAUT : A High-Level Synthesis Tool for DSP Applications*. Coussy P., Morawiec A. (eds) High-Level Synthesis. Springer, 2008, pp. 147–169.

[35] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp : An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Transactions on Embedded Computing Systems (TECS) - Special issue on application-specific processors*, vol. 13, no. 2, pp. 24 :1–24 :27, September 2013.

[36] M. Owaida and et al., "Enhancing design space exploration by extending CPU/GPU specifications onto FPGAs," *ACM Transactions on Embedded Computing Systems (TECS)*, 2015.

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

[38] *Local and metropolitan area networks - Specific requirements Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE 802.11n, 2009.

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

[40] 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.

[41] R. Maunder, "Survey of ASIC implementations of LDPC decoders," 2016.

[42] B. Le Gal, J. Crenne, and C. Jego, "A high throughput efficient approach for decoding LDPC codes onto GPU devices," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 29–32, June 2014.

[43] B. Le Gal and C. Jego, "Low-latency software LDPC decoders," in *Proc. of SIPS*, 2017.

[44] F. Pratas, J. Andrade, G. Falcao, V. Silva, and L. Sousa, "Open the gates : High-level synthesis high-level synthesis towards programmable LDPC decoders on FPGAs," in *Prococeedings of GlobalSIP*, 2013.

[45] J. Andrade, G. Falcao, , and V. Silva, "Flexible design of wide-pipeline based WiMAX QC-LDPC decoder architectures on FPGAs using high-level synthesis," *IET Electronics Letters*, 2014.

[46] E. Scheiber and et al., "Implementation of an LDPC decoder for IEEE 802.11n using Vivado high-level synthesis," in *Proc. of ESPCO*, 2013.

[47] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC decoding on multicores using OpenCL," *IEEE Signal Processing Magazine*, 2012.

[48] J. Andrade and et al, "Combining flexibility with low power : Dataflow and wide-pipeline LDPC decoding engines in the Gbit/s era," in *2014 IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, June 2014, pp. 264–269.

[49] S.-D. Roh, K. Cho, and K.-S. Chung, "Implementation of an LDPC decoder on a heterogeneous FPGA-CPU platform using SDSoC," in *Proc. of TENCON*, 2016.

[50] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. of SIPS*, 2004.

[51] B. Le Gal, C. Jego, and C. Leroux, "A flexible NISC-based LDPC decoder," *IEEE Transactions on Signal Processing (TSP)*, vol. 62, no. 10, pp. 2469—2479, May 2014.

[52] X. Wu, Y. Song, M. Jiang, and C. Zhao, "Adaptive-normalized/offset min-sum algorithm," *IEEE Communications Letters*, 2010.

[53] G. Wang and et al, "A massively parallel implementation of QC-LDPC decoder on GPU," in *2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, June 2011, pp. 82–85.

[54] ——, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Proc. of GlobalSIP*, 2013.

[55] A. I. V. Casado, M. Griot, and R. D. Wesel, "LDPC decoders with informed dynamic scheduling," *IEEE Transactions on Communications*, vol. 58, no. 12, pp. 3470–3479, December 2010.

[56] X. Liu, Y. Zhang, and R. Cui, "Variable-node-based dynamic scheduling strategy for belief-propagation decoding of ldpc codes," *IEEE Communications Letters*, vol. 19, no. 2, pp. 147–150, February 2015.

[57] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for LDPC decoding," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4076–4091, November 2007.

[58] C. A. Aslam, Y. L. Guan, K. Cai, and G. Han, "Low-complexity belief-propagation decoding via dynamic silent-variable-node-free scheduling," *IEEE Communications Letters*, vol. 21, no. 1, pp. 28–31, January 2017.

[59] B. Le Gal and C. Jego, "High-throughput multi-core LDPC decoders based on x86 processor," *IEEE Transactions on Parallel and Distributed Systems*, 2016.

[60] ——, "High-throughput LDPC decoder on low-power embedded processors," *IEEE Communication Letters*, 2015.

[61] Y. Delomier, B. Le Gal, J. Crenne, and C. Jego, "Model-based design of efficient LDPC decoder architectures," in *Proceedings of the* 10^{th} *International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, Hong Kong, China, December 2018, pp. 1–5.

[62] Accelera systems initiative, *SystemC Synthesizable Subset Version 1.4.7*, March 2016.

[63] C. Marchand and E. Boutillon, "LDPC decoder architecture for DVB-S2 and DVB-S2X standards," in *Proceedings fo the IEEE Workshop on Signal Processing Systems (SiPS)*, October 2015, pp. 1–5.

[64] V. Pignoly, B. Le Gal, C. Jego, and B. Gadat, "High data rate and flexible hardware QC-LDPC decoder for satellite optical communications," in *Proceedings of the International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, December 2018.

[65] H. Gharaee, M. Kiaee, and N. Mohammadzadeh, "A high-throughput FPGA implementation of quasi-cyclic LDPC decoder," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, pp. 140–149, March 2017.

[66] S. Mhaske, H. Kee, T. Ly, A. Aziz, and P. Spasojevic, "High-throughput FPGA-based QC-LDPC decoder architecture," in *Proc. of VTC Fall*, 2015.

[67] S. A. Zied, A. T. Sayed, and R. Guindi, "Configurable low complexity decoder architecture for quasi-cyclic LDPC codes," in *2013 21st SoftCOM*, Sept 2013, pp. 1–5.

[68] B. Le Gal and C. Jego, "Design of an ASIP LDPC decoder compliant with digital communication standards," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS)*, October 2012, pp. 19–24.

[69] W. H. Zhao and J. P. Long, "Implementing the NASA deep space LDPC codes for defense applications," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, 2013, pp. 803–808.

[70] S. Mhaske, H. Kee, T. Ly, A. Aziz, and P. Spasojevic, "FPGA-based channel coding architectures for 5G wireless using high-level synthesis," *International Journal of Reconfigurable Computing*, 2017.

[71] A. Amaricai, O. Boncalo, and I. Mot, "Memory efficient FPGA implementation for flooded LDPC decoder," in *Proceedings of the 23rd Telecommunications Forum Telfor (TELFOR)*, 2015, pp. 500–503.

[72] I. Tanyanon and S. Choomchuay, "A hardware design of MS/MMS-based LDPC decoder," in *2012 IEEE International Conference on Electron Devices and Solid State Circuit (EDSSC)*, Dec 2012.

[73] X. Han, K. Niu, and Z. He, "Implementation of IEEE 802.11n LDPC codes based on general purpose processors," in *2013 15th IEEE International Conference on Communication Technology (ICCT)*, Nov 2013, pp. 218–222.

[74] Y. Delomier, B. Le Gal, J. Crenne, and C. Jego, "Fast design of reliable, flexible and high-speed AWGN architectures with high level synthesis," in *Proceedings of the 25th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, Bordeaux, France, December 9-12 2018, pp. 661–664.

[75] D. U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, "A hardware gaussian noise generator using the Box-Muller method and its error analysis," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 659–671, June 2006.

[76] D. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. W. Leong, "A hardware Gaussian noise generator using the Wallace method," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 8, pp. 911–920, Aug 2005.

[77] J. Choi, J. Jung, and I. C. Park, "Area-efficient approach for generating quantized gaussian noise," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 63, no. 7, pp. 1005–1013, July 2016.

[78] J. S. Malik, J. N. Malik, A. Hemani, and N. D. Gohar, "Generating high tail accuracy gaussian random numbers in hardware using central limit theorem," in *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, Oct 2011, pp. 60–65.

[79] G. Zhang, P. H. W. Leong, D.-U. Lee, J. D. Villasenor, R. C. C. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *International Conference on Field Programmable Logic and Applications, 2005.*, Aug 2005, pp. 275–280.

[80] Y. Delomier, B. Le Gal, J. Crenne, and C. Jego, "Fast design of reliable, flexible and high-speed AWGN architectures with high level synthesis," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2018, pp. 661–664.

[81] ——, "Génération d'architectures de décodeur LDPC à l'aide d'un outil de synthèse de haut niveau," in *Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS)*, 2018.

[82] ——, "From multicore LDPC decoder implementations to FPGA decoder architectures : a case study," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2018, pp. 89–92.

[83] ——, "Model-based design of efficient LDPC decoder architectures," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Dec 2018, pp. 1–5.

[84] ——, "Model-based design of flexible and efficient ldpc decoders on fpga devices," *Springer, Journal of Signal Processing Systems (JSPS)*, 2020.

[85] O. Afisiadis, A. Balatsoukas, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Proceedings of the Asilomar Conference*, 2015.

[86] Y. Zhou, J. Lin, and Z. Wang, "Improved Fast-SSC-Flip decoding of polar codes," *IEEE Communications Letters*, 2019.

[87] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[88] A. J. Raymond and W. J. Gross, "Scalable successive-cancellation hardware decoder for polar codes," in *2013 IEEE Global Conference on Signal and Information Processing*, 2013.

[89] A. J. Raymond and W. J. Gross, "A scalable successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing (TSP)*, 2014.

[90] B. Le Gal, C. Leroux, and C. Jego, "A scalable 3-phase polar decoder," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016.

[91] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, December 2011.

[92] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders : Algorithm and implementation," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2014.

[93] O. Dizdar and 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, March 2016.

[94] *3GPP, Multiplexing and channel coding, ETSI TS 138 212 V15.2.0 (2018-07).*

[95] K. David and H. Berndt, "6G vision and requirements : Is there any need for beyond 5G ?" *IEEE Vehicular Technology Magazine*, vol. 13, no. 3, pp. 72–80, September 2018.

[96] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, October 2013.

[97] K. Niu, K. Chen, J. Lin, and Q. T. Zhang, "Polar codes : Primary concepts and practical decoding algorithms," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 192–203, July 2014.

[98] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Multi-mode unrolled architectures for polar decoders," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 63, no. 9, pp. 1443–1453, September 2016.

[99] P. Giard, G. Sarkis, C. Thibeault, and W. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, May 2015.

[100] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Springer Journal of Signal Processing Systems (JSPS)*.

[101] P. Giard, A. Balatsoukas-Stimming, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast low-complexity decoders for low-rate polar codes," *Journal of Signal Processing Systems*, vol. 90, no. 5, pp. 675–685, May 2018.

[102] M. Hanif and M. Ardakani, "Fast successive-cancellation decoding of polar codes : Identification and decoding of new nodes," *IEEE Communitation Letters*, vol. 21, no. 11, pp. 2360–2363, November 2017.

[103] X. Zhang, X. Yan, Q. Zeng, J. Cui, N. Cao, and R. Higgs, "High-throughput Fast-SSC polar decoder for wireless communications," *Wireless Communications and Mobile Computing, Hindawi*, vol. 2018, pp. 1–10, July 2018.

[104] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing (TSP)*, 2013.

[105] C. Zhang and K. K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2429–2441, May 2013.

[106] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 61, no. 4, pp. 1241–1254, April 2014.

[107] G. Berhault, C. Leroux, C. Jego, and D. Dallet, "Memory requirement reduction method for successive cancellation decoding of polar codes," *Journal of Signal Processing Systems, Springer*, vol. 88, no. 3, pp. 425–438, September 2017.

[108] B. Le Gal, C. Leroux, and C. Jego, "Successive cancellation decoder for very long polar codes," in *Proceedings of the IEEE Internationnal Workshop on Signal Processing Systems (SIPS)*, Lorient, France, October 3–5 2017.

[109] E. Arikan, "A performance comparison of polar codes and reed-muller codes," *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, June 2008.

[110] H.-Y. Yoon, S.-J. Hwang, and T.-H. Kim, "A 655 Mbps successive-cancellation decoder for a 1024-bit polar code in 180nm CMOS," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, November 2018, pp. 281 –284.

[111] C. Kestel, S. Weithoffer, and N. Wehn, "Polar code decoder exploration framework," *Advances in Radio Science (ARS)*, vol. 16, pp. 1–8, 2018.

[112] B. Le Gal, C. Leroux, and C. Jego, "Software polar decoder on an embedded processor," in *Proceedings of the IEEE International Workshop on Signal Processing Systems (SIPS)*, Belfast, UK, October 20–22 2014, pp. 1–6.

[113] ——, "Multi-Gb/s software decoding of polar codes," *IEEE Transactions on Signal Processing (TSP)*, vol. 63, no. 2, pp. 349–359, January 2015.

[114] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-latency software polar decoders," *Journal of Signal Processing Systems (JSPS), Springer*, vol. 90, no. 5, pp. 761–775, May 2018.

[115] A. Cassagne, B. Le Gal, C. Leroux, O. Aumage, and D. Barthou, "An efficient, portable and generic library for successive cancellation decoding of polar codes," in *Proceedings of the 28th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, vol. 303–317, Raleigh, NC, USA, September 2015.

[116] B. L. et al., "ClickNP : Highly flexible and high performance network processing with reconfigurable hardware," in *Proc. of SIGCOMM*, 2016.

[117] Xilinx, *Vivado Design Suite User Guide - High-Level Synthesis*, ug902 (v2017.1) ed., 2017.

[118] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Flexible and low-complexity encoding and decoding of systematic polar codes," *IEEE Transactions on Communications (TCOM)*, 2016.

[119] M. Hanif and M. Ardakani, "Fast successive-cancellation decoding of polar codes : Identification and decoding of new nodes," *IEEE Communications Letters*, 2017.

[120] Xilinx, *7 Series FPGAs Memory Resources - User Guide, UG473 (v1.14)*, July 2019.

[121] G. Liva, L. Gaudio, T. Ninacs, and T. Jerkovits, "Code design for short blocks : A survey," in *Proceedings of the Ultra-Reliable and Mission Critical Communication Workshop*, Athens, Greece, June 2016.

[122] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, October 2012.

[123] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive-cancellation list decoders for polar codes," in *IEEE Transactions on Signal Processing*, vol. 65, no. 21, November 2017, pp. 5756–5769.

[124] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "On metric sorting for successive cancellation list decoding of polar codes," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2015, pp. 1993–1996.

[125] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *International Workshop on Signal Processing Systems (SiPS)*. IEEE, Oct. 2014, pp. 1–6.

[126] K. Wang, L. Li, F. Han, F. Feng, J. Lin, Y. Fu, and J. Sha, "Optimized sorting network for successive cancellation list decoding of polar codes," *IEICE Electronics Express*, vol. advpub, 2017.

[127] Y. Fan, J. Chen, C. Xia, C. ying Tsui, J. Jin, H. Shen, and B. Li, "Low-latency list decoding of polar codes with double thresholding," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, QLD, Australia, April 2015, pp. 1042–1046.

[128] X. Liang, C. Yang, J. Zhang, W. Song, and X. You, "Hardware efficient and low-latency CA-SCL decoder based on distributed sorting." in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, December 2016.

[129] A. Sural, "An FPGA implementation of successive cancellation list decoding for polar codes," Master's thesis, Bilkent University, January 2016.

[130] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 34, no. 2, pp. 318–328, Feb. 2016.

[131] B. Le Gal, Y. Delomier, C. Leroux, and C. Jego, "Low-latency sorter architecture for polar codes successive-cancellation-list decoding," *IEEE Transactions on Circuits and Systems II : Express Briefs (TCAS-II),*, 2020.

[132] O. Astrachan, "Bubble sort : An archaeological algorithmic analysis," *SIGCSE Bull.*, vol. 35, no. 1, p. 1–5, Jan. 2003.

[133] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA : Association for Computing Machinery, 1968, p. 307–314.

[134] C. Xia, Y. Fan, J. Chen, C. ying Tsui, C. Zeng, J. Jin, and B. Li, "An implementation of list successive cancellation decoder with large list size for polar codes," in *Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, Belgium, September 2017.