Efficient Hardware Implementations of LDPC Decoders, through Exploiting Impreciseness in Message-Passing Decoding Algorithms

Thien Truong Nguyen Ly

THÈSE

présentée

à l'Université de Cergy Pontoise
École Nationale Supérieure de l'Électronique de ses Applications

pour obtenir le grade de :

Docteur en Science de l'Université de Cergy Pontoise
Spécialité : Sciences et Technologies de l'Information et de la Communication

Par

Thien Truong NGUYEN LY

Équipes d'accueil :
Équipe Traitement des Images et du Signal (ETIS) – CNRS UMR 8051
Laboratoire des Systèmes sans fil Haut Débit (LSHD) – CEA/LETI

Titre de la thèse

Efficient Hardware Implementations of LDPC Decoders, through Exploiting Impreciseness in Message-Passing Decoding Algorithms

Soutenue le 03/05/2017 devant la commission d'examen composée de :

Prof.	Emmanuel Boutillon	Lab-STICC, Université Bretagne Sud	Président
Prof.	Christophe Jégo	IMS, Institut Polytechnique de Bordeaux	Rapporteur
Prof.	Charly Poulliat	INP-ENSEEIHT, Université de Toulouse	Rapporteur
Dr.	Oana Boncalo	University Politehnica Timisoara, Romania	Examinateur
Dr.	Fakhreddine Ghaffari	ENSEA, Université de Cergy-Pontoise	Examinateur
Dr.	Valentin Savin	CEA-LETI, MINATEC, Grenoble	Encadrant
Prof.	David Declercq	ENSEA, Université de Cergy-Pontoise	Directeur de thèse

To my parents, my two sisters and my sweetheart

Acknowledgment

This thesis could not have been completed without the kind help of the following people to whom I would like to express my special thanks.

First, I would like to express my deepest gratitude to my two advisors, Prof. David DECLERCQ and Dr. Valentin SAVIN for their whole-hearted guidance, invaluable support, helpful advice, useful suggestions, and encouragement throughout my PhD work. Especially, I have no words to describe how thankful I am to Valentin SAVIN for everything he has done for me. He not only gave me professional knowledge but also helped me improve the soft skills. I could never forget the period when he helped me correct the articles, as well as my PhD manuscript. I will always remember the long discussions, and the rehearsals for my presentations. He has listened carefully and given me invaluable comments. To be honest, I would not be able to complete my PhD defense without his help. Besides, he helped me improve my speaking and writing skills. I also learned a lot from him, especially his enthusiasm, dedication, thoughtfulness, and meticulosity. I am sure that once I come back to Vietnam, I will also treat my students as he has done for me. In all sincerity, I would like to say thank Valentin again.

Second, my special thanks go to Prof. Christophe JEGO, and Prof. Charly POULLIAT for their acceptance as my PhD reviewers. Thank you so much for your comments on my PhD manuscript. I also would like to thank Prof. Emmanuel, Oana, and Fakhreddine for their acceptance as my PhD committee members.

Third, I would like to express my thanks to all the colleagues at Laboratoire des Systèmes sans fil Haut Débit (LSHD) – CEA/LETI for their friendship, support, encouragement, and fun, especially Minh, Quynh, Mickael, Yoann, Ludovic, Réda, Gourab, Moisés, Remun, Ioan-Sorin, Florian, Jimmy, Valerian, Robin, David, Luiz, Elodie, Rida, Antonio, Luc, Nicolas, Benoît, François, Sylvie, Jean-Baptiste, Manuel, and Xavier. I also would like to thank Lam, and Khoa, my colleagues, as well as my best friends at Équipe Traitement des Images et du Signal (ETIS). Thank you so much for all your help and support during my PhD work. I am also very grateful to Annick BERTINOTTI, Sandrine BERTOLA for helping me with administrative procedures, and Dimitri KTENAS, Fabien CLERMIDY for supporting me attend to the conferences.

Fourth, I would like to express my sincere gratitude to Bach Khoa University (BKU), Vietnam, especially Prof. Dinh-Thanh VU, Prof. Hong-Tuan DO, and Prof. Trang HOANG for giving me the opportunity to carry out my PhD study in France.

Last but not least, I am warmly grateful to my parents, Van-Chanh NGUYEN, and Thi-Kim LY, my two sisters, Nhu-An NGUYEN, and Ngoc-Khang NGUYEN, and my sweetheart, Kim-Anh NGUYEN for their love, moral support, and encouragement throughout my life. They have inspired me strength, shared with me moments of stress, disappointment, and encouraged me overcome all difficulties and challenges. Thank you very much.

Author's publications

Published papers

[A1] T. Nguyen-Ly, K. Le, F. Ghaffari, A. Amaricai, O. Boncalo, V. Savin, and D. Declercq, "FPGA design of high throughput LDPC decoder based on imprecise offset min-sum decoding", *IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1-4, Grenoble, France, June 2015.

[A2] T. T. Nguyen-Ly, K. Le, V. Savin, D. Declercq, F. Ghaffari, and O. Boncalo, "Non-surjective finite alphabet iterative decoders", *IEEE International Conference on Communications (ICC)*, pages 1-6, Kuala Lumpur, Malaysia, May 2016.

[A3] Z. Mheich, T. Nguyen-Ly, V. Savin, and D. Declercq, "Code-aware quantizer design for finite-precision min-sum decoders", *IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, Varna, Bulgaria, June 2016.

[A4] T. T. Nguyen-Ly, T. Gupta, M. Pezzin, V. Savin, D. Declercq, and S. Cotofana, "Flexible, cost-efficient, high-throughput architecture for layered LDPC decoders with fully-parallel processing units", *Euromicro Conference on Digital System Design (DSD)*, pages 230-237, Limassol, Cyprus, September 2016.

[A5] T. T. Nguyen-Ly, V. Savin, X. Popon, and D. Declercq, "High throughput FPGA implementation for regular non-surjective finite alphabet iterative decoders", *IEEE International Conference on Communications Workshops (ICC)*, Paris, France, May 2017.

Submitted papers

[A6] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, "Analysis and design of cost-effective, high-throughput LDPC decoders", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017, submitted.

Participation to research projects

[DIAMOND] The author participated to the research project *"Message Passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-Area-Delay Optimization"* (DIAMOND), supported by the Franco-Romanian (ANR-UEFISCDI) Joint Research Programme "Blanc-2013".

Résumé

Les codes correcteurs d'erreurs sont une composante essentielle de tout systéme de communication, capables d'assurer le transport fiable de l'information sur un canal de communication bruitè. Les systémes de communication de nouvelle génération devront faire face à une demande sans cesse croissante en termes de débit binaire, pouvant aller de 1 à plusieurs centaines de gigabits par seconde. Dans ce contexte, les codes LDPC (pour Low-Density Parity-Check, en anglais), sont reconnus comme une des solutions les mieux adaptées, en raison de la possibilité de paralléliser massivement leurs algorithmes de décodage et les architectures matérielles associées. Cependant, si l'utilisation d'architectures massivement parallèles permet en effet d'atteindre des débits très élevés, cette solution entraine également une augmentation significative du coût matériel.

L'objectif de cette thèse est de proposer des implémentations matérielles de décodeurs LDPC très haut débit, en exploitant la robustesse des algorithmes de décodage par passage de messages aux imprécisions de calcul. L'intégration dans le décodage itératif de mécanismes de calcul imprécis, s'accompagne du développement de nouvelles approches d'optimisation du design en termes de coût, débit et capacité de correction.

Pour ce faire, nous avons considéré l'optimisation conjointe de (i) le bloc de quantification qui fournit l'information à précision finie au décodeur, et (ii) les unités de traitement imprécis des données, pour la mise à jour des messages échangés pendant de processus de décodage. Ainsi, nous avons tout d'abord proposé un quantificateur à faible complexité, qui peut être optimisé par évolution de densité en fonction du code LDPC utilisé et capable d'approcher de très près les performances d'un quantificateur optimal. Le quantificateur proposé a été en outre optimisé et utilisé pour chacun des décodeurs imprécis proposés ensuite dans cette thèse.

Nous avons ensuite proposé, analysé et implémenté plusieurs décodeurs LDPC imprécis. Les deux premiers décodeurs sont des versions imprécises du décodeur « Offset Min-Sum » (OMS) : la surestimation des messages des noeuds de contrôle est d'abord compensée par un simple effacement du bit de poids faible (« Partially OMS »), ensuite le coût matériel est d'avantage réduit en supprimant un signal spécifique (« Imprecise Partially OMS »). Les résultats d'implémentation sur cible FPGA montrent une réduction importante du coût matériel, tout en assurant une performance de décodage très proche du OMS, malgré l'imprécision introduite dans les unités de traitement.

Nous avons ensuite introduit les décodeurs à alphabet fini non-surjectifs (NS-FAIDs, pour « Non-Surjective Finite Alphabet Iterative Decoders », en anglais), qui étendent le concept d'« imprécision » au bloc mémoire du décodeur LDPC. Les décodeurs NS-FAIDs ont été optimisés par évolution de densité pour des codes LDPC réguliers et irréguliers. Les résultats d'optimisation révèlent différents compromis possibles entre la performance de décodage et l'efficacité de la mise en oeuvre matérielle. Nous avons également proposé trois architectures matérielles haut débit, intégrant les noyaux de décodage NS-FAID. Les résultats d'implémentation sur cible FPGA et ASIC montrent que les NS-FAIDs permettent d'obtenir des améliorations significatives en termes de coût matériel et de débit, par rapport au décodeur Min-Sum, avec des performances de décodage meilleures ou très légèrement dégradées.

Abstract

The increasing demand of massive data rates in wireless communication systems will require significantly higher processing speed of the baseband signal, as compared to conventional solutions. This is especially challenging for Forward Error Correction (FEC) mechanisms, since FEC decoding is one of the most computationally intensive baseband processing tasks, consuming a large amount of hardware resources and energy. The conventional approach to increase throughput is to use massively parallel architectures. In this context, Low-Density Parity-Check (LDPC) codes are recognized as the foremost solution, due to the intrinsic capacity of their decoders to accommodate various degrees of parallelism. They have found extensive applications in modern communication systems, due to their excellent decoding performance, high throughput capabilities, and power efficiency, and have been adopted in several recent communication standards.

This thesis focuses on cost-effective, high-throughput hardware implementations of LDPC decoders, through exploiting the robustness of message-passing decoding algorithms to computing inaccuracies. It aims at providing new approaches to cost/throughput optimizations, through the use of imprecise computing and storage mechanisms, without jeopardizing the error correction performance of the LDPC code. To do so, imprecise processing within the iterative message-passing decoder is considered in conjunction with the quantization process that provides the finite-precision information to the decoder. Thus, we first investigate a low complexity code and decoder aware quantizer, which is shown to closely approach the performance of the quantizer with decision levels optimized through exhaustive search, and then propose several imprecise designs of Min-Sum (MS)-based decoders. Proposed imprecise designs are aimed at reducing the size of the memory and interconnect blocks, which are known to dominate the overall area/delay performance of the hardware design. Several approaches are proposed, which allow storing the exchanged messages using a lower precision than that used by the processing units, thus facilitating significant reductions of the memory and interconnect blocks, with even better or only slight degradation of the error correction performance.

We propose two new decoding algorithms and hardware implementations, obtained by introducing two levels of impreciseness in the Offset MS (OMS) decoding: the Partially OMS (POMS), which performs only partially the offset correction, and the Imprecise Partially OMS (I-POMS), which introduces a further level of impreciseness in the check-node processing unit. FPGA implementation results show that they can achieve significant throughput increase with respect to the OMS, while providing very close decoding performance, despite the impreciseness introduced in the processing units.

We further introduce a new approach for hardware efficient LDPC decoder design, referred to as Non-Surjective Finite-Alphabet Iterative Decoders (FAIDs). NS-FAIDs are optimized by Density Evolution for regular and irregular LDPC codes. Optimization results reveal different possible trade-offs between decoding performance and hardware implementation efficiency. To validate the promises of optimized NS-FAIDs in terms of hardware implementation benefits, we propose three high-throughput hardware architectures, integrating NS-FAIDs decoding kernels. Implementation results on both FPGA and ASIC technology show that NS-FAIDs allow significant improvements in terms of both throughput and hardware resources consumption, as compared to the Min-Sum decoder, with even better or only slightly degraded decoding performance.

Contents

List of Figures

List of Tables

Glossary

AP	A Posteriori
AP-LLR	A Posteriori Log-Likelihood Ratio
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BF	Bit-Flipping
BI-AWGN	Binary-Input Additive White Gaussian Noise
BP	Belief-Propagation
BPSK	Binary Phase-Shift Keying
BRAM	Block RAM
BSC	Binary Symmetric Channel
CLA	Carry Lookahead Adder
CMOS	Complimentary Metal Oxide Semiconductor
CN	Check Node
CNU	Check Node Unit
DE	Density Evolution
DL	Decision Level
DVB-S2	Digital Video Broadcasting - Satellite - Second Generation
EE	Energy Efficiency
FAID	Finite Alphabet Iterative Decoder
FEC	Forward Error Correction

FER	Frame Error Rate
FMIG	First Minimum and Index Generator
FPGA	Field-Programmable Gate Array
GF	Gain Factor
HUE	Hardware Usage Efficiency
HW	Hardware
IG	Index Generator
I-POMS	Imprecise Partially Offset Min-Sum
LDPC	Low-Density Parity-Check
LLR	Logarithmic Likelihood Ratio
LSB	Least Significant Bit
LUT	Look-Up Table
MAP	Maximum A Posteriori
MI	Mutual Information
ML	Maximum Likelihood
MP	Message-Passing
MS	Min-Sum
MSB	Most Significant Bit
NEE	Normalized Energy Efficiency
NMS	Normalized Min-Sum
NS-FAID	Non-Surjective Finite Alphabet Iterative Decoder
NTAR	Normalized Throughput to Area Ratio
OMS	Offset Min-Sum
PAR	Place-and-Route
pdf	Probability density function
PEG	Progressive Edge Growth
pmf	Probability mass function

POMS	Partially Offset Min-Sum
QC	Quasi-Cyclic
RAM	Random Access Memory
RPL	Rows Per Layer
SCMS	Self-Corrected Min-Sum
SIPO	Serial Input Parallel Output
SNR	Signal-to-Noise Ratio
SP	Sum-Product
TAR	Throughput-to-Area Ratio
TS	Tree Structure
VN	Variable Node
VNU	Variable Node Unit
WiFi	Wireless Fidelity
WiGig	WIreless GIGabit
WiMAX	WorldWide Interoperability for Microwave Access
WPAN	Wireless Personal Area Network

Notation

[Related to parity-check matrix and Tanner graphs]

$\mathbf{H}$: parity-check matrix

M: number of rows of $\mathbf{H}$

N: number of columns of $\mathbf{H}$

$K = N - \text{rank}(\mathbf{H})$: dimension of $\mathbf{H}$

$r = \dfrac{K}{N}$: coding rate

$\mathbf{B}$: base matrix (used for QC-LDPC codes)

R: number of rows of $\mathbf{B}$

C: number of columns of $\mathbf{B}$

RPL: number of rows per layer

$L = R/\text{RPL}$: number of decoding layers

z: expansion factor of $\mathbf{B}$

$Z = z \times \text{RPL}$: number of parity checks within one decoding layer, and reffered to as the parallelism degree (of the hardware architecture)

$\mathcal{H}$: Tanner graph associated with $\mathbf{H}$

$n \in \{1, 2, \ldots, N\}$: a variable-node of $\mathcal{H}$

$m \in \{1, 2, \ldots, M\}$: a check-node of $\mathcal{H}$

$\mathcal{H}(n)$:a set of check-nodes connected to the variable-node n, also referred to as the set of neighbor check-nodes of n

$\mathcal{H}(m)$: a set of variable-nodes connected to the check-node m, also referred to as the set of neighbor variable-nodes of m

$d_n = |\mathcal{H}(n)|$: number of check-nodes connected to n also referred to as the degree of the variable-node n

$d_m = |\mathcal{H}(m)|$: number of variable-nodes connected to m also referred to as the degree of the check-node m

[Related to channel and transmitted/received words]

σ^2: noise variance (BI-AWGN model)

$\mathcal{X}$: channel input alphabet

$\mathcal{Y}$: channel output alphabet

$\underline{\boldsymbol{x}} = (x_1, \ldots, x_N) \in \mathcal{X}^N$: transmitted

$\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N)$: estimated codeword

$\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$: received word

$\boldsymbol{P}_n = \Pr(x_n = 1 \mid y_n)$: probability of the transmitted bit x_n being equal to 1, conditional on the received value y_n

$\boldsymbol{L}_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)} = \log \dfrac{1 - \boldsymbol{P}_n}{\boldsymbol{P}_n}$: LLR of the transmitted bit x_n, conditional on the received value y_n

[Related to message-passing decoding]

γ_n: *a priori* information of the decoder concerning variable-node n

$\tilde{\gamma}_n$: *a posteriori* information provided by the decoder, concerning variable node n

$\alpha_{m,n}$: message sent from variable-node n to check-node m

$\beta_{m,n}$: message sent from check-node m to variable-node n

$\nu \in]0, 1[$: normalization (scaling) factor used for Normalized Min-Sum decoding

$\delta > 0$: offset factor used for Offset Min-Sum decoding

[Related to quantization]

φ: quantization map

$\mu > 0$: gain factor used to scale the received signal

q: number of bits for representation of a priori information and exchanged messages

$\tilde{q}$: number of bits for representation of a posteriori information

$\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$, where $Q = 2^{q-1} - 1$, is the alphabet of the a priori information and exchanged messages.

$\tilde{\mathcal{M}} = \{-\tilde{Q}, \ldots, -1, 0, +1, \ldots, +\tilde{Q}\}$, where $\tilde{Q} = 2^{\tilde{q}-1} - 1$, is the alphabet of the a posteriori information.

F: framing function used for NS-FAIDs decoding

W: weight of F, *i.e.*, the number of distinct entries in the vector $[|F(0)|, F(1), \ldots, F(Q)]$

$w = \lceil \log_2(W) \rceil + 1$: framing bit-length

Chapter 1

Introduction

1.1 Context and Motivations

There has been a significant growth in the volume of mobile data traffic in recent years. This is due to proliferation of smart phones and other mobile devices that support a wide range of broadband applications and services. It is projected in [23] that the volume of the so called smart traffic will be 96% of the global data traffic by 2018, and among this more than 50% data will be routed using the WiFi access points or femtocells, while the remaining will be covered by the cellular networks. This tendency is confirmed by the multi-Gigabit/s data rate target of the current evolution of WLAN standards (IEEE802.11ac/ad), or of the 5th generation of cellular networks (5G). Moreover, short range communications in the extremely high frequency spectrum (from 30 to 300 Gigahertz) are expected to reach throughput from 100 Gigabit/s to 1 Terabit/s (Figure 1.1).

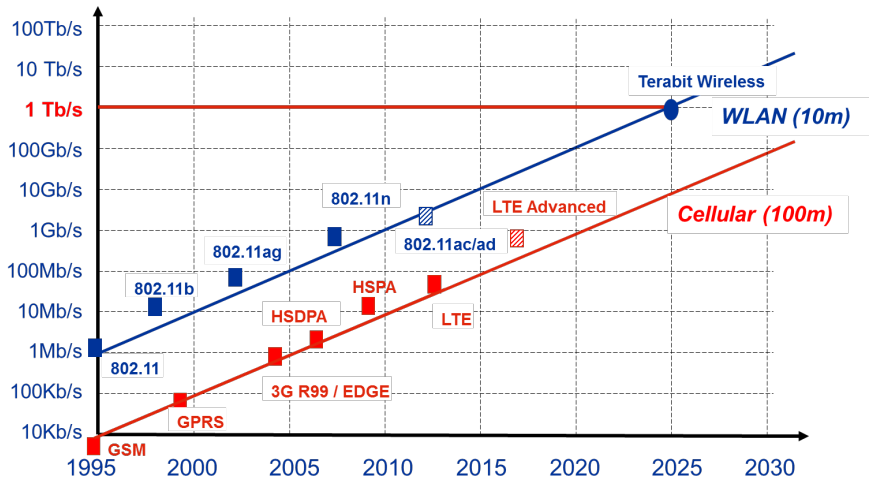


Figure 1.1: Wireless roadmap [32]

The increasing demand of massive data rates in wireless communications, supported by the use of very large bandwidths, will result in stringent, application-specific, requirements in terms of both throughput and latency. They will necessitate significantly higher processing speed of the baseband signal, as compared to conventional solutions, which is especially challenging for the Forward Error Correction (FEC) mechanism. Indeed, due to the erroneous nature of communication channels, FEC mechanisms are essential to any communication system, no matter the way the information is transported, for instance using a point-to-point, multicast, or broadcast technology, a wired or wireless link, and a terrestrial, satellite-based or hybrid infrastructure. However, these error-correction mechanisms require intensive computing tasks, consuming a large amount of hardware resources and energy.

The conventional approach to increase throughput is to use massively parallel architectures. In this context, Low-Density Parity-Check (LDPC) codes are recognized as the foremost solution, due to the intrinsic capacity of their decoders to accommodate various degrees of parallelism. They have found extensive applications in modern communication systems, due to their excellent decoding performance, high throughput capabilities, and power efficiency, and have been adopted in several communication standards, such as IEEE 802.11n (WiFi), IEEE 802.16e (WiMAX), IEEE 802.15.3c (WPAN), ETSI DVB-S2/T2/C2. More recently, to support increasing throughput requirements in wireless standards, LDPC codes have been adopted by IEEE 802.11ac (WiFi evolution) and IEEE 802.11ad (WiGig), as well as by the 3rd Generation Partnership Project (3GPP) for enhanced Mobile BroadBand (eMBB) 5G data channels [2].

In this context, the main objective of this thesis is to contribute to the development of very high-throughput, yet cost-effective, LDPC decoder designs, so that to address in the most suitable way the new generation of communication systems, requiring increased data rates and reduced energy footprint. The approach proposed in this thesis, is to exploit the robustness of message-passing LDPC decoders to computing inaccuracies. The idea is actually inspired by the approximate computing paradigm: the computation returns a possibly inaccurate result, for a situation where an approximate result is sufficient for a specific purpose [70]. Another closely related parading is the one of approximate storage: instead of storing data values exactly, they are stored approximately, *e.g.*, by truncating the lowest significant bits. In this thesis, the use of approximate computing and storage in message passing LDPC decoders is seen as an enabler for throughput and cost optimization. By modeling such approximations in a specific way, we are actually able to address the multi-objective optimization of the design, with respect to cost, throughput, and error correction performance. The main contributions of the thesis will be outlined in the next section.

1.2 Main Contributions and Thesis Outline

This thesis targets the design of cost-effective, high-throughput LDPC decoders, so as to enable a smooth transition towards the new generation of multi-Gigabit/s communication systems. The main novelty of the research comes from the expansion of the design space, so as to harness the potential of approximate computing and storage. In this line of research, the thesis is aimed at (i) designing novel decoder architectures based on the integration of approximate computing and storage units, while still providing reliable error protection, and (ii) providing new approaches for the multi-objective optimization of the design with respect to cost, throughput, and error correction performance.

Typical LDPC decoder architectures consist of three main building blocks: memory, interconnects, and processing units. One important characteristic of LDPC decoders is that the memory and interconnect blocks usually dominate the overall area/power/delay performance of the hardware design. This emphasizes the potential benefits of approximate storage techniques, as they may significantly reduce the memory size requirements. Besides, as a side benefit, such mechanisms may also reduce the footprint of the routing network, which carries messages from the memory to the processing units.

To avoid degrading the error correction performance of the decoder, approximate storage techniques should be investigated along with the use of novel processing units, capable to compensate the information loss due to approximate storage. Such processing units are necessarily "approximate versions" of the conventional ones, but they can be subject to further circuit-level approximations, *e.g.*, by removing some specific signals from the implementing circuit.

Finally, it appears clear that approximate computing and storage techniques must be investigated in conjunction with the quantization rules for computing the finite-alphabet information supplied to the decoder, which actually determines the starting point of the iterative decoding process.

The main contributions of this thesis are summarized below, along with the thesis outline.

Chapter 2: Low-Density Parity-Check Codes and Message-Passing Decoders

Chapter 2 provides a brief introduction to LDPC codes and Message-Passing (MP) decoders. We first introduce the bipartite graph representation of LDPC codes and the principle of iterative MP decoding algorithms. Then we focus on some specific decoding algorithms that will be used in this thesis, namely the Min-Sum (MS) and several MS-based algorithms. The asymptotic analysis of finite alphabet MS decoders, based on the density evolution method, is also discussed.

We further investigate the main algorithmic and architectural choices that impact the decoding performance and the hardware complexity of LDPC decoders. In particular, we discuss different scheduling strategies for MP decoders and their

impact on the hardware architecture. Finally, several state-of-the-art approaches on low-cost, high-throughput hardware implementations are also reviewed in this chapter.

Chapter 3: Code-Aware Quantizer Design for Finite-Alphabet Min-Sum Decoders

Classically, the quantization of the soft information supplied to a finite-alphabet decoder is chosen to optimize a certain criterion which does not depend on the characteristics of the existing code. Chapter 3 investigates quantizers that are both code and decoder aware: such quantizers optimize the density evolution noise threshold of a given decoder and a given ensemble of LDPC codes. Throughout this chapter the LDPC decoder under consideration is the finite-alphabet MS decoder, and thus we shall simply refer to such quantizers as code-aware quantizers.

We propose a code-aware quantizer with lower complexity than that obtained by optimizing all decision levels and approaching its performance, for few quantization bits. We show that code-aware quantizers outperform code-independent quantizers in terms of noise threshold for both regular and irregular LDPC codes. To overcome the error floor behavior of LDPC codes, we propose the design of the quantizer for a target error probability at the decoder output. The results show that the quantizer optimized to get a zero error probability could lead to a very bad performance for practical range of signal to noise ratios.

We further propose to design jointly irregular LDPC codes and code-aware quantizers for the finite-alphabet MS decoder. We show that they achieve significant decoding gains with respect to LDPC codes designed for infinite-alphabet belief propagation decoding, but decoded by finite-alphabet MS.

Finally, we note that the proposed code-aware quantizer can easily be adapted to the MS-based decoders investigated in the next chapters of this thesis. This will allow the optimization of the code-aware quantizer jointly with the integration of approximate computing and storage techniques into the MS decoder.

Chapter 4: Design of High Throughput LDPC Decoder based on Imprecise Offset MS Decoding

Chapter 4 proposes two new decoding algorithms and hardware implementations, obtained by introducing two levels of impreciseness in the Offset MS (OMS) decoding: the Partially OMS (POMS), which performs only partially the offset correction, and the Imprecise Partially OMS (I-POMS), which introduces a further level of impreciseness in the check-node processing unit. We show that they allow significant reduction in the memory (25% with respect to the conventional OMS) and interconnect, and we further propose a cost-efficient check-node unit (CNU) architecture, yielding a cost reduction of 56% with respect to the conventional CNU architecture.

We further implement FPGA-based layered decoder architectures using the proposed algorithms as decoding kernels, for a (3,6)-regular Quasi-Cyclic LDPC code of

length 1296 bits, and evaluate them in terms of cost, throughput and decoding performance. Implementation results on Xilinx Virtex 6 FPGA device show that they can achieve a throughput between 1.95 and 2.41 Gbps for 20 decoding iterations (48% to 83% increase with respect to OMS), while providing decoding performance close to the OMS decoder, despite the impreciseness introduced in the processing units.

Chapter 5: Non-Surjective Finite Alphabet Iterative Decoders

Chapter 5 introduces a new family of Non-Surjective Finite Alphabet Iterative Decoders (NS-FAIDs), characterized by specific processing rules, which can be theoretically analyzed and optimized for different trade-offs between hardware cost and error correction performance. The proposed approach builds upon the approximate storage technique, and allows storing the exchanged messages using a smaller precision than the processing units. It is also shown to provide a unified approach for several designs previously proposed in the literature.

NS-FAIDs are optimized by density evolution for regular and irregular LDPC codes, and are shown to provide different trade-offs between hardware complexity and decoding performance. In particular, we show that optimized NS-FAIDs may yield significant reductions of the memory size requirements, while providing even better or only slightly degraded error correction performance with respect to other MS-based decoders.

Chapter 6: Low-Cost, High-Throughput Hardware Architectures

Chapter 6 proposes three Quasi-Cyclic (QC)-LDPC decoder architectures targeting low-cost, high-throughput and efficient use of the hardware resources. All of the proposed architectures implement layered scheduled decoding with fully parallel processing units.

The first architecture is an enhanced version of the architecture introduced in Chapter 4. It is based on a specific design of the datapath processing units (VNUs, CNUs, and AP-LLR units) that allow an efficient reuse of the hardware resources, thus yielding significant cost reduction.

In the second architecture, efficient use of the hardware resources is ensured by pipelining the datapath, which also allows increasing the operating frequency, thus increasing the achieved throughput. However, the use of pipelining imposes specific constraints on the decoding layers, in order to ensure proper execution of the layered decoding process. Finally, the third architecture does not make use of pipelining, but allows maximum parallelism to be exploited through the use of full decoding layers, thus resulting in significant increase in throughput.

The second and third architectures may accommodate both MS and NS-FAID decoding kernels. Thus, to validate the promises of the NS-FAID approach, both MS and NS-FAID decoding kernels are integrated into each of these architectures, and compared in terms of throughput and resources consumption. Implementation

results on both FPGA and ASIC platforms show that regular and irregular NS-FAIDs allow significant improvements in terms of both throughput and hardware resources consumption, as compared to the conventional MS solution, with even better or only slightly degraded decoding performance.

We use the Throughput to the Area Ratio (TAR) metric, expressed in Gbps/mm^2, to capture the performance of the proposed designs in terms of both throughput and cost. ASIC synthesis results targeting a 65nm CMOS technology, for our designs achieving the highest TAR performance, are summarized below:

- For irregular QC-LDPC codes, with rate 1/2 and code-length 2304 bits, specified by the IEEE 802.16e (WiMAX) standard, proposed designs achieve a TAR metric up to 2.7 Gbps/mm^2 at 20 decoding iterations, corresponding to a Normalized TAR of 54 Gbps/mm^2/iteration (Normalized TAR corresponds to the TAR value for one decoding iteration).
- For $(3, 6)$-regular QC-LDPC codes, with rate 1/2 and code-length 1296 bits, proposed designs achieve a TAR metric up to 6.1 Gbps/mm^2 at 20 decoding iterations, corresponding to a Normalized TAR of 122 Gbps/mm^2/iteration.

Chapter 7: Conclusion and Perspectives

Chapter 7 summarizes the results obtained in this thesis, and provides perspectives for future works.

Chapter 2

Low-Density Parity-Check Codes and Message-Passing Decoders

This chapter provides a brief introduction to Low Density Parity Check (LDPC) codes and Message-Passing (MP) decoders. We first introduce the bipartite graph representation of LDPC codes and the principle of iterative MP decoding algorithms. Then we focus on some specific decoding algorithms that will be used in this thesis, namely the Min-Sum (MS) and several MS-based algorithms. The asymptotic analysis of finite alphabet MS decoders, based on the density evolution method, is also discussed.

We further investigate the main algorithmic and architectural choices that impact the decoding performance and the hardware complexity of LDPC decoders. In particular, we discuss different scheduling strategies for MP decoders and their impact on the hardware architecture. Finally, several state-of-the-art approaches on low-cost, high-throughput hardware implementations are also reviewed in this chapter.

2.1 Introduction

This chapter aims to answer three main questions such as *What are LDPC codes?*, *Why are we interested in them?* and *How do they work?*. LDPC codes have been introduced by Gallager in his doctoral thesis in 1962 [35, 36], as a class of linear Forward Error Correction (FEC) codes defined by sparse parity-check matrices, which can be advantageously decoded by iterative message-passing (MP) decoders. Unlike other linear codes introduced at that time (*e.g.*, Hamming, BCH, Reed-Solomon, etc.), which are short codes, based on algebraic constructions and with known error correction capacity, LDPC codes are usually long and often constructed pseudo-randomly, with only a probabilistic notion of their expected error correction performance [50]. Because of the computational effort in implementing encoder and decoder at that time, LDPC codes were mostly ignored for a long time. In 1996, they were rediscovered by Mackay [65], after the power of iterative decoding techniques had been also confirmed by the discovery of Turbo codes [6]. Since then, they have been developed rapidly and attracted a lot of research interest, especially focused on the analysis of iterative decoding algorithms [20, 22, 28, 58, 82, 83], as well as techniques developed for code construction and optimization [34, 42, 53].

Nowadays, LDPC codes have found extensive applications in modern communication systems due to their excellent decoding performance [21, 82, 83], as well as their advantages related to hardware implementations, such as low-cost [8, 10, 29], high-throughput capabilities [49, 52, 110, 111], and power efficiency [66, 75, 103]. Thus, LDPC codes are gradually replacing other well-established FEC schemes [84]. For instance, they have been adopted in several recent communication standards, such as 802.11n (Wi-Fi) [44], 802.11ad (WiGig) [43], 802.16e (WiMAX) [46], 802.15.3c (WPAN) [45], and DVB-S2 [31], and are being considered for a range of application areas, from optical networks to digital storage [50]. Moreover, LDPC codes have been proposed as a potential candidate for 5G cellular system [68].

2.2 LDPC Codes

2.2.1 Definition, Tanner graphs

An LDPC code is a linear block code defined by a (M, N) *sparse* parity-check matrix denoted by $\mathbf{H}$. As their name suggests, *sparse* or *low-density* means that the parity-check matrix contains only a few 1's in comparison to the amount of 0's. This sparseness of $\mathbf{H}$ is essential for an iterative decoding complexity that increases only linearly with the code length. Figure 2.1(a) illustrates an example of parity-check matrix. Of course, the matrix in this example cannot be really called low-density. In order to satisfy low-density property, the size of parity-check matrix should usually be very large, of several hundreds of thousands of columns.

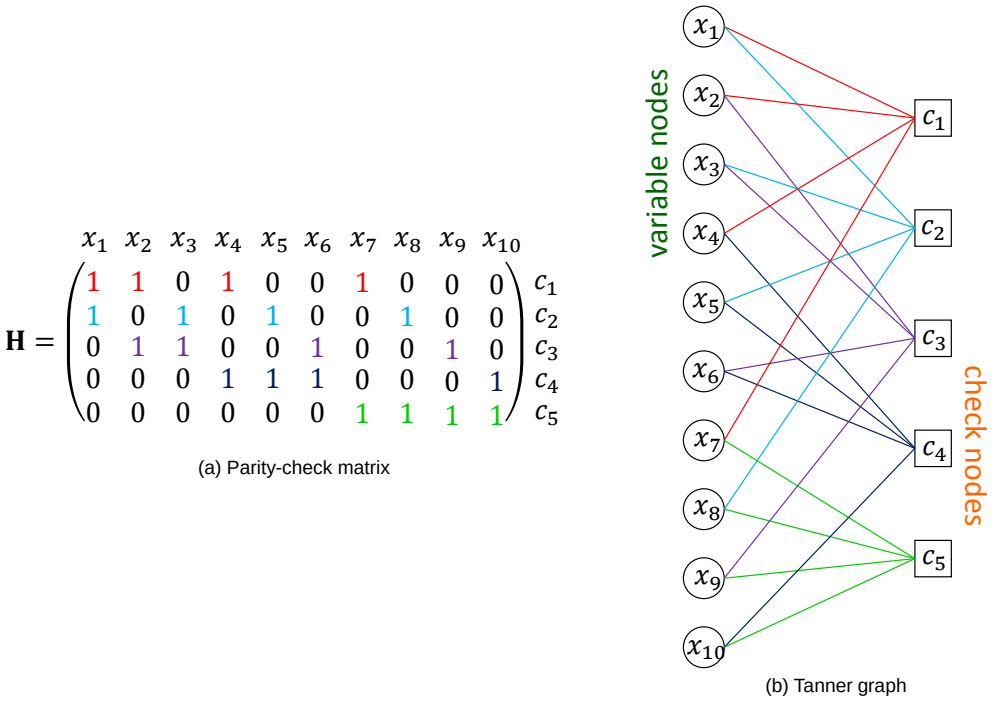


Figure 2.1: Example of parity-check matrix and corresponding Tanner graph

The N columns of the parity-check matrix $\mathbf{H}$ correspond to coded bits, and the M rows correspond to parity-check equations. A vector $\underline{\boldsymbol{x}} = (x_1, \ldots, x_N) \in \{0, 1\}^N$ is a codeword if and only if:

$$\mathbf{H}\underline{\boldsymbol{x}}^{\mathbf{T}} = \mathbf{0} \,(\text{mod } 2) \tag{2.1}$$

where $\mathbf{0}$ is the length-M all zeros vector and $\underline{\boldsymbol{x}}^T$ denotes the transpose of $\underline{\boldsymbol{x}}$.

Assuming that the parity-check matrix $\mathbf{H}$ is full rank (*i.e.*, $\text{rank}(\mathbf{H}) = M$) and the code is systematic, then any length-N codeword is comprised of $K = N - M$ information bits and M parity bits. The fraction $r = K/N$ is called the *coding rate* of the code. In other words, it characterizes the amount of redundancy added by the error correction code.

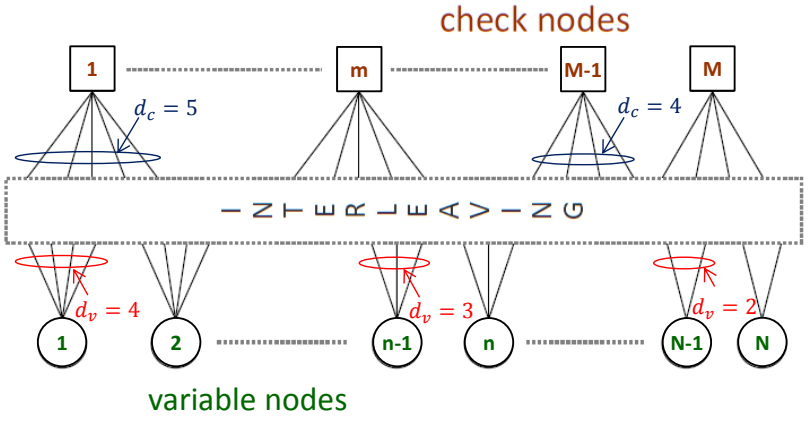


Figure 2.2: Example of irregular Tanner graph

The parity-check matrix can also be represented by a bipartite graph, denoted by $\mathcal{H}$, introduced by Tanner [94], and therefore usually referred to as *Tanner graph* of the code. The bipartite graph consists of two sets of nodes, namely the set of variable-nodes, corresponding to coded bits (or columns of $\mathbf{H}$), and the set of check-nodes correspond to parity-check equations (or rows of $\mathbf{H}$). The edges connecting variable and check nodes correspond to the 1's entries of $\mathbf{H}$. This bipartite graph representation is very useful for describing the message-passing decoding algorithms (see Section 2.3). Figure 2.1(b) shows the bipartite Tanner graph representation of the parity-check matrix from Figure 2.1(a).

Now, we will answer the question why we are interested in LDPC codes. The main reason comes from the fact that they are able to achieve excellent decoding performance. In 2001, Richardson and co-authors showed that irregular LDPC codes can be optimized such that to approach the Shannon limit [82, 83]. They showed that it is possible to construct irregular LDPC codes, with decoding performance at only 0.0045 dB from the Shannon limit [21].

It is worth noting that LDPC codes can be *regular* and *irregular*, depending on their variable and check node degrees. The node degree is defined as the number of adjacent edges connected to that node. Variable-node degrees will be denoted by $d_v(n)$, and check-node degrees by $d_c(m)$. An example is shown in Figure 2.2. It can be easily seen that variable node-1 has 4 edges connected to, so it is of degree 4. Similarly, variable node-$(n-1)$ and variable node-$(N-1)$ have degrees equal to 3 and 2, respectively. In the same manner, for check-nodes, it can be observed that check node-1 is of degree 5, and check node-$(M-1)$ is of degree 4.

An LDPC code is said to be regular if all variable-nodes have the same degree d_v, and all check-nodes have the same degree d_c. Otherwise, the code is said to be irregular. It is worth noticing that most of the practical constructions of irregular LDPC codes are actually irregular on variable-node degrees, but regular or semi-regular in check-node degrees (semi-regular means that all check-node degrees take on two consecutive values). The irregularity of an LDPC code can be defined by the edge perspective degree-distributions polynomials, given in Eq. (2.2).

$$\lambda(x) = \sum_{d>1} \lambda_d x^{d-1}, \quad \rho(x) = \sum_{d>1} \rho_d x^{d-1} \tag{2.2}$$

where λ_d denotes the fraction of edges connected to variable-nodes of degree d, and ρ_d denotes the fraction of edges connected to check-node of degree d.

Finally, it is important to mention that irregularity allows more degree of freedom for code design, and carefully designed irregular LDPC codes have better decoding performance than their regular counterparts, as long as the decoding performance in the waterfall region is concerned (assuming the same codeword length). However, they also exhibit higher error floors compared to regular LDPC codes, usually attributed to fact that irregular graphs are more prone to the occurrence of small topological structures, discussed below, and known to be responsible for the failures of iterative message passing decoders.

The *waterfall region* corresponds to the Signal to Noise Ratio (SNR) region where the Bit Error Rate (BER) curve steadily decreases with increasing SNR. However, graph-based iteratively decodable codes, such as LDPC codes, there is a point after which the BER curve does not fall as quickly as before, in other words, there is a region in which the BER performance flattens. This region is referred to as the error floor region. The error floor phenomenon is directly attributed to small topological structures in the bipartite graph, such as stopping sets [28] (BEC channel) or trapping sets [80], near codewords [64], pseudo codewords [97] (BSC and AWGN channel). Irregular LDPC codes are more prone to such structures, mainly due variable-nodes of low degree. Besides, quantization effects also cause the error floor [113].

2.2.2 Quasi-Cyclic LDPC codes

Quasi-cyclic (QC) LDPC codes [34] are a class of structured LDPC codes, widely used in practical applications. They are defined by a base matrix $\mathbf{B}$ of size $R \times C$, with integer entries $b_{i,j} \geq -1$, where $i \in \{1, \ldots, R\}$ and $j \in \{1, \ldots, C\}$, as shown in Figure 2.3. The parity-check matrix $\mathbf{H}$ is obtained by expanding the base matrix $\mathbf{B}$ by an expansion factor z. Thus, each entry of $\mathbf{B}$ is replaced by a square matrix of size $z \times z$, defined as follows:

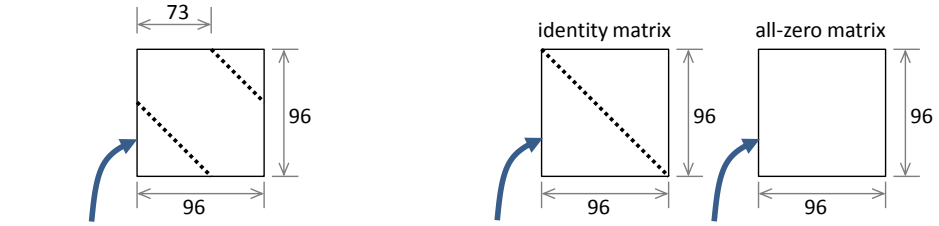
- entries $b_{i,j} = -1$ are replaced by the all-zero matrix,
- entries $b_{i,j} \geq 0$ are replaced by a circulant matrix, obtained by right-shifting the identity matrix by $b_{i,j}$ positions.

It follows that the parity-check matrix $\mathbf{H}$ is of size $M \times N$, with $M = z \times R$ rows and $N = z \times C$ columns.

Figure 2.4 gives an example of base matrix of size $R \times C = 12 \times 24$, and expansion factor $z = 96$ for WiMAX QC-LDPC code with rate of 1/2.

$$\mathbf{B} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,C} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ b_{R,1} & b_{R,2} & \cdots & b_{R,C} \end{pmatrix}$$

Figure 2.3: Base matrix of QC-LDPC code



$\mathbf{B} =$

-1	94	73	-1	-1	-1	-1	-1	55	83	-1	-1	7	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	27	-1	-1	-1	22	79	9	-1	-1	-1	12	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	24	22	81	-1	33	-1	-1	-1	0	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
61	-1	47	-1	-1	-1	-1	-1	65	25	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1
-1	-1	39	-1	-1	-1	84	-1	-1	41	72	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	46	40	-1	82	-1	-1	-1	79	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1
-1	-1	95	53	-1	-1	-1	-1	-1	14	18	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1
-1	11	73	-1	-1	-1	2	-1	-1	47	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1
12	-1	-1	-1	83	24	-1	43	-1	-1	-1	51	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1
-1	-1	-1	-1	-1	94	-1	59	-1	-1	70	72	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1
-1	-1	7	65	-1	-1	-1	-1	39	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
43	-1	-1	-1	-1	66	-1	41	-1	-1	-1	26	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0

Figure 2.4: Base matrix of WiMAX QC-LDPC code with rate of 1/2

The main advantages of the quasi-cyclic structure are as follows:

- QC-LDPC codes can provide comparable error correction performance compared with random-LDPC codes, while allowing simple and structured construction [34],
- QC-LDPC codes are well-suited to hardware implementations, with parallelizable architecture and simple interconnection network, due to the specific structure of the parity-check matrix [98, 110],
- QC-LDPC codes can support for flexible block lengths, by considering different expansion factors z [10, 111]

Due aforementioned advantages, QC-LDPC codes have been selected by several advanced communication standards, such as IEEE 802.11n (Wi-Fi) and IEEE 802.16e (WiMAX).

2.3 Decoding algorithms

In the previous sections of this chapter, we discussed about *What are LDPC codes?* and *Why are we interested in them?*. In this section we discuss several decoding algorithms, in order to answer the last question *How do LDPC codes work?*

2.3.1 Message-Passing algorithms

The class of algorithms used to decode LDPC codes are collectively termed as Message-Passing (MP) iterative algorithm. In MP iterative decoding, messages are passed between variable and check-nodes along the edges of the bipartite Tanner graph. This process takes place in several rounds, or *iterations*. At each new iteration, new messages are computed in an *extrinsic manner*, meaning that a message that is sent on an edge does not depend on the message just received on the same edge. The computation of the exchanged messages and of the a posteriori information is illustrated in Figure 2.5.

In the following, we denote by $\mathcal{H}$ the Tanner graph of the code, by $\mathcal{H}(m)$ the set of variable-nodes connected to the check-node m, and by $\mathcal{H}(n)$ the set of check-nodes connected to the variable-node n. The main steps of an MP decoder can be summarized as follows:

Initialization: The a priori information (γ_n) is computed for each variable-node n, and variable-to-check messages ($\alpha_{m,n}$) are initialized accordingly.

Iteration Loop: Each decoding iteration consists of the following steps:

- **CN-processing** (check-node processing): check-to-variable messages ($\beta_{m,n}$) are updated according to the current value of $\alpha_{m,n'}$ messages, with $n' \in \mathcal{H}(m) \setminus \{n\}$.
- **VN-processing** (variable-node processing): variable-to-check messages ($\alpha_{m,n}$) are updated according to the input γ_n value and the current value of $\beta_{m',n}$ messages, with $m' \in \mathcal{H}(n) \setminus \{m\}$.
- **AP-update** (a posteriori information update): the a posteriori information ($\tilde{\gamma}_n$) is updated according to the input γ_n value and the current value of $\beta_{m,n}$ messages, with $m \in \mathcal{H}(n)$.

In addition, at each iteration the decoder also computes a hard-decision vector, denoted by $\underline{\hat{\boldsymbol{x}}}$ and corresponding to the estimation of the transmitted codeword, and the corresponding syndrome $\underline{\boldsymbol{s}} = \mathbf{H}\underline{\hat{\boldsymbol{x}}}^T$. The decoder stops if a codeword has been found (*i.e.*, $\underline{\boldsymbol{s}} = \mathbf{0}$), or if the maximum number of iterations has been reached.

Depending on the update rules that are used to compute check and variable-node messages, there are several message-passing decoding algorithms, such as Bit-Flipping (BF), Belief-Propagation (BP), Min-Sum (MS) and MS-based algorithms [86]. In this thesis, we mainly focus on MS and MS-based decoding. In order to understand the rationale behind the MS decoding, it is necessary to take a look first to the BP decoding.

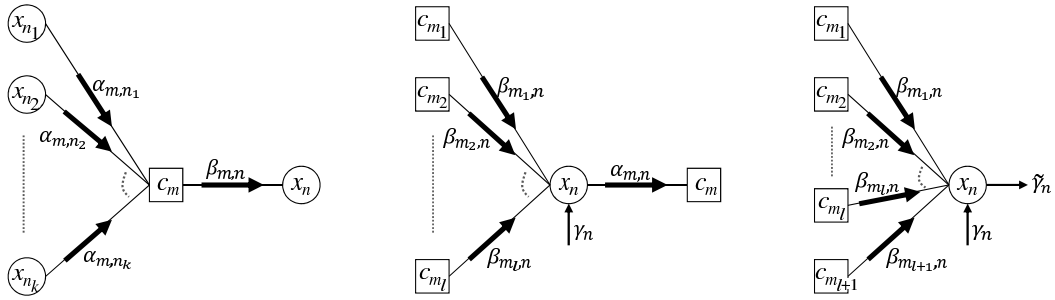


Figure 2.5: Computation of extrinsic messages and of the a posteriori information

2.3.2 Belief-Propagation decoding

The Belief-Propagation (BP), also referred to as Sum-Product (SP), is a soft decision message-passing (MP) algorithm. In the probability domain, the a priori information (γ_n) of the BP decoding consists of the probability of each coded bit (x_n) being equal to 0 or 1, conditional on the received value (y_n). These probabilities are updated by the iterative message-passing process. Thus, the extrinsic information passed between variable and check-nodes is also given in terms of probabilities. The BP decoding is known to be optimal for codes defined by cycle-free bipartite graphs, in the sense it outputs the *Maximum A Posteriori* (MAP) estimates of the coded bits which is given by:

$$\hat{x}_n^{(\mathrm{MAP})} = \underset{b\in\{0,1\}}{\operatorname{argmax}} \Pr(x_n = b \mid \underline{\boldsymbol{y}}) \tag{2.3}$$

Alternatively, BP decoding can be implemented in logarithmic or log-likelihood ratio (LLR) domain. It is usually more convenient to consider the BP decoding in LLR-domain instead of probability-domain, without decoding performance loss. The most benefit of the representation of probabilities in LLR-domain comes from the fact that multiplication is replaced by addition operation. In terms of hardware implementation, this can reduce significantly the implementation complexity. The algorithm of BP decoding in LLR-domain is described in Algorithm 1.

2.3.3 Min-Sum decoding

The most advantage of BP decoding is represented by its excellent decoding performance. Although practical codes are defined by bipartite graphs with cycles, and thus BP fails to produce MAP estimates of the coded bits, it is often referred to in the literature as being the "optimal" messqage-passing decoder. However, in practical implementations, BP decoding has a number of drawbacks related to its complexity, numerical instability, and its sensitivity to channel signal-to-noise ratio (SNR) estimation mismatch. One way to deal with complexity and numerical instability issues is to simplify the computation of messages exchanged within the BP decoding. From Algorithm 1, it can easily be seen that the most complex step of the BP decoding is the computation of check-node messages (CN-processing step), which makes

Algorithm 1 LLR-domain Belief Propagation (BP) decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word

Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization

for all $n = 1, \ldots, N$ **do** $\gamma_n = \boldsymbol{L}_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}$;

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\alpha_{m,n} = \gamma_n$;

Iteration Loop

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \left(\prod_{n' \in \mathcal{H}(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \right) . \Phi \left(\sum_{n' \in \mathcal{H}(m) \setminus n} \Phi(|\alpha_{m,n'}|) \right);$$

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**

$$\alpha_{m,n} = \gamma_n + \sum_{m' \in \mathcal{H}(n) \setminus m} \beta_{m',n};$$

for all $n = 1, \ldots, N$ **do** ▷ **AP-update**

$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in \mathcal{H}(n)} \beta_{m,n};$$

for all $n = 1, \ldots, N$ **do** $\hat{x}_n = \frac{1 - \operatorname{sgn}(\tilde{\gamma}_n)}{2}$; ▷ hard decision

if $\underline{\hat{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check

End Iteration Loop

The function Φ, used to compute the check-to-variable messages, is given by:

$$\Phi(u) = -\log\left(\tanh\frac{u}{2}\right) = \log\left(\frac{1+e^{-u}}{1-e^{-u}}\right), \forall u > 0$$

use of computationally intensive hyperbolic tangent functions. The MS algorithm [20, 30, 33] is aimed at reducing the computational complexity of the BP, by using "max-log" approximations for these messages. The only computations required by the MS decoding are additions and comparisons, which solves the complexity and numerical instability problems. The performance of the MS decoding is also known to be insensitive to channel SNR estimation mismatch.

MS decoding is based on an approximation of the hyperbolic tangent function Φ. Figure 2.6 shows the shape of the $y = \Phi(x)$ curve. It can be easily observed that the curve of Φ decreases rapidly along the y-axis for small x values, then it continues decreasing smoothly for intermediate x values, getting increasingly closer to the x-axis for increasing values of x. Hence, for any set of positive values $\{x_i\}_{i \in I}$, one can use the approximation:

$$\Phi\left(\sum_{i \in I} x_i\right) \approx \Phi\left(\max_{i \in I} x_i\right) \tag{2.4}$$

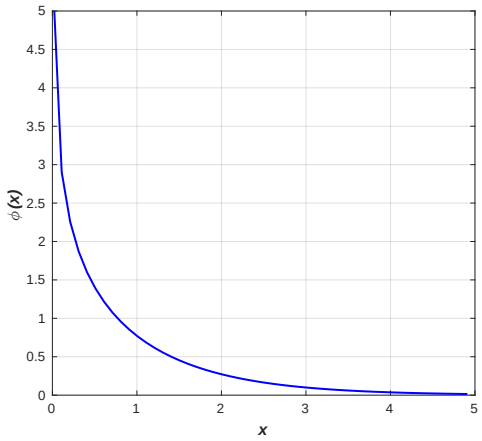


Figure 2.6: Φ function, where $\Phi(x) = -\log(\tanh \frac{x}{2}),\ \forall x > 0$

which is known as the max-log approximation. In follows that:

$$\Phi\left(\sum_{i\in I}\Phi(x_i)\right) \approx \Phi\left(\max_{i\in I}\Phi(x_i)\right) = \Phi\left(\Phi\left(\min_{i\in I}x_i\right)\right) = \min_{i\in I}x_i \tag{2.5}$$

The MS decoding, using the above approximation, is described in Algorithm 2.

2.3.4 Min-Sum-based decoding

As mentioned in the previous section, MS decoding has been seen as a promising solution to reduce the complexity of the BP decoding, by using max-log approximation of the check-node processing side. However, the max-log approximation used in the MS decoding causes an overestimation of check-node messages, which leads to a degradation in the error-rate performance of the decoder. Indeed, since Φ is a decreasing function satisfying $\Phi(\Phi(x)) = x$, it follows that $\Phi\left(\sum_{i\in I}\Phi(x_i)\right) \leq \Phi\left(\Phi(x_j)\right) = x_j\ (\forall j \in I)$, and consequently:

$$\Phi\left(\sum_{i\in I}\Phi(x_i)\right) \leq \min_{i\in I}(x_i) \tag{2.6}$$

Several approaches have been proposed in the literature to compensate the overestimation of check-node messages, as for instance using scaling or offset factors [16, 17, 108]. These algorithms are known in the literature as Normalized MS (NMS), Offset MS (OMS), and several enhancements have been further proposed by different authors, such as the Modified OMS [105] and the Adaptive NMS/OMS decoding algorithms [48, 102]. A different approach which deals with the overestimation issue at the variable-node processing side, referred to as Self-Corrected Min-Sum (SCMS), was introduced in [85]. We refer to such decoding algorithms as *MS-based decoding*. It is worth noting that all MS-based decoding algorithms aim at improving the decoding performance, with only small or negligible increase of the algorithmic (computational) complexity.

Algorithm 2 Min-Sum (MS) decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word

Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization

for all $n = 1, \ldots, N$ **do** $\gamma_n = \boldsymbol{L}_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}$;

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\alpha_{m,n} = \gamma_n$;

Iteration Loop

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \left(\prod_{n' \in \mathcal{H}(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \right) \left(\min_{n' \in \mathcal{H}(m) \setminus n} |\alpha_{m,n'}| \right);$$

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**

$$\alpha_{m,n} = \gamma_n + \sum_{m' \in \mathcal{H}(n) \setminus m} \beta_{m',n};$$

for all $n = 1, \ldots, N$ **do** ▷ **AP-update**

$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in \mathcal{H}(n)} \beta_{m,n};$$

for all $n = 1, \ldots, N$ **do** $\hat{x}_n = \dfrac{1 - \operatorname{sgn}(\tilde{\gamma}_n)}{2}$; ▷ hard decision

if $\underline{\hat{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check

End Iteration Loop

2.3.4.1 Normalized and Offset Min-Sum decoding

The Normalized Min-Sum (NMS) and Offset Min-Sum (OMS) decoding algorithms are slightly modified versions of the MS decoder that rely on the use of either a normalization or an offset factor to compensate the overestimation of check-to-variable messages:

- The NMS decoder (Algorithm 3) compensates this overestimation by introducing a normalization (scaling) factor $\nu \in]0, 1[$ within the CN-processing step.
- The OMS decoder (Algorithm 4) compensates this overestimation by introducing an offset factor $\delta > 0$ within the CN-processing step.

The NMS and OMS decoding algorithms are probably the most popular ones, mainly because of their simplicity. Apart from the use of a normalization or offset factor for the computation of check-to-variable messages, the other steps of NMS and OMS decoding algorithms are identical to those of the MS decoding. The normalization (resp. offset) factor can either be constant (*e.g.*, for regular LDPC codes, the same normalization/offset factor value is used for all the check-nodes), or vary, usually as a function of the check-node degree. The optimal value of the normalization (resp. offset) factor can be determined by either Monte-Carlo simulation or density evolution (DE) analysis (DE will be discussed in Section 2.4.2).

Algorithm 3 Normalized Min-Sum (NMS) decoding

... ▷ same as MS decoding

Iteration Loop

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \nu. \left(\prod_{n' \in \mathcal{H}(m) \backslash n} \operatorname{sgn}(\alpha_{m,n'}) \right) . \left(\min_{n' \in \mathcal{H}(m) \backslash n} |\alpha_{m,n'}| \right);$$

... ▷ same as MS decoding

End Iteration Loop

Normalization (scaling) factor $\nu \in]0, 1[$; may either be a constant value, or vary according to the check-node degree.

Algorithm 4 Offset Min-Sum (OMS) decoding

... ▷ same as MS decoding

Iteration Loop

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \left(\prod_{n' \in \mathcal{H}(m) \backslash n} \operatorname{sgn}(\alpha_{m,n'}) \right) . \max \left\{ \left(\min_{n' \in \mathcal{H}(m) \backslash n} |\alpha_{m,n'}| \right) - \delta, 0 \right\};$$

... ▷ same as MS decoding

End Iteration Loop

Offset factor $\delta > 0$; may either be a constant value, or vary according to the check-node degree.

It is important to note that these normalization/offset factors must be "finely tuned" (optimized) in order to avoid creating "artificial" *error floors* (*i.e.*, error floors that are not directly attributable to topological structures of the bipartite-graph). To overcome this problem, two-dimensional (2-D) NMS and OMS decoding algorithms have been proposed in [108]. They rely on normalization (resp. offset) factors used to normalize (resp. offset) both variable-to-check and check-to-variable messages, and whose values depend on variable and check-node degrees. DE analysis can be used to derive optimal values for 2-D normalization (resp. offset) factors [15].

2.3.4.2 Self-Corrected Min-Sum decoding

The Self-Corrected Min-Sum (SCMS) [85] addresses the overestimation issue at the variable-node processing side of the algorithm. The rationale behind the SCMS is that the overestimation of check-node messages is not critical, unless any given variable-node message is updated to map a different bit state. Hence, the characteristic of the SCMS decoding is to "erase" (*e.g.*, set to zero) any variable-node message that experiences a sign change between two consecutive iterations. The SCMS decoding is described in Algorithm 5. It can also be shown that such a sign change occurs if and only if the corresponding computation tree contains unreliable infor-

Algorithm 5 Self-Corrected Min-Sum (SCMS) decoding

... ▷ same as MS decoding

Iteration Loop

... ▷ same as MS decoding

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**

$$\alpha_{m,n}^{\text{tmp}} = \gamma_n + \sum_{m' \in \mathcal{H}(n) \setminus m} \beta_{m',n}$$

$$\alpha_{m,n} = \begin{cases} 0, & \text{if } \operatorname{sgn}(\alpha_{m,n}^{\text{tmp}}) \neq \operatorname{sgn}(\alpha_{m,n}) \text{ and } \alpha_{m,n} \neq 0 \\ \alpha_{m,n}^{\text{tmp}}, & \text{otherwise} \end{cases}$$

... ▷ same as MS decoding

End Iteration Loop

mation and, as a consequence, the SCMS decoding behaves as the MS decoding on a computation tree that has been pruned of its unreliable branches.

2.4 Quantized Min-Sum decoding

2.4.1 Finite alphabet Min-Sum decoding

Since the focus of this thesis is on efficient hardware implementation of LDPC decoders, in this section we deal with quantized versions of the MS decoding, also referred to as *finite-alphabet MS* decoding.

The finite-alphabet MS decoding is described in Algorithm 6. Throughout this thesis, we will assume that the a priori information (γ_n) and the exchanged messages $(\alpha_{m,n}, \beta_{m,n})$ are quantized on q bits, while the a posteriori information $(\tilde{\gamma}_n)$ is quantized on $\tilde{q} > q$ bits. The alphabet of the a priori information and of the exchanged messages is denoted by $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$, where $Q \triangleq 2^{q-1} - 1$. Similarly, the alphabet of the a posteriori information is denoted by $\tilde{\mathcal{M}} = \{-\tilde{Q}, \ldots, -1, 0, +1, \ldots, +\tilde{Q}\}$, where $\tilde{Q} \triangleq 2^{\tilde{q}-1} - 1$.

We shall further assume that the a priori information $\gamma_n = \varphi(\boldsymbol{L}_n)$ is a quantized version of the LLR value $\boldsymbol{L}_n$, with φ denoting the *quantization map*. Hence, in the initialization step of Algorithm 6, the LLR values $\boldsymbol{L}_n$ are first computed, and then quantized to q bits, by using $\gamma_n = \varphi(\boldsymbol{L}_n)$. Variable-to-check node messages $\alpha_{m,n}$ are then initialized according to the corresponding γ_n value. During the decoding iterations, the only arithmetic operations required to update variable and check-node messages are additions and comparisons. To adhere to usual hardware implementations, we shall assume that all additions are performed by using $\tilde{q}$-bit saturated adders. Then, the VN-processing step in Algorithm 6, first computes the value of $\alpha_{m,n}$ on $\tilde{q}$-bits, then saturates it to q-bits. This convention will be also reflected in the density evolution analysis from Section 2.4.2.

Of course the decoding performance of the finite-alphabet MS decoding directly depends on the way the input information is quantized, *i.e.*, on the quantization map φ. Several quantization methods will be investigated in Chapter 3. We describe below the *gain factor* quantizer that will be used in Chapters 3 to 6 of this thesis. We assume a Binary-Input Additive White Gaussian Noise (BI-AWGN) channel model, with noise variance σ^2. Hence, the channel output y_n is given by $y_n = x_n + z_n$, where z_n follows a Gaussian distribution with mean 0 and variance σ^2, and the LLR value $\boldsymbol{L}_n$ can be computed by:

$$\boldsymbol{L}_n = \frac{2}{\sigma^2} y_n \tag{2.7}$$

We consider a constant $\mu > 0$, referred to as *gain factor*, and define

$$\varphi(\boldsymbol{L}_n) = \left[\mu \frac{\sigma^2}{2} \boldsymbol{L}_n \right]_{\mathcal{M}} = [\mu y_n]_{\mathcal{M}} \tag{2.8}$$

where $[x]_{\mathcal{M}}$ denotes the closest integer to x that belongs to $\mathcal{M} = \{-Q, \ldots, +Q\}$. The map φ will be referred to as *gain factor quantizer*. Since, the value of $\gamma_n = \varphi(\boldsymbol{L}_n)$ can actually be directly computed from the received value y_n, when no confusion is possible we shall also write $\varphi(y_n)$ instead of $\varphi(\boldsymbol{L}_n)$, and thus:

$$\gamma_n = \varphi(y_n) = [\mu y_n]_{\mathcal{M}} \tag{2.9}$$

Algorithm 6 Finite Alphabet Min-Sum decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word

Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization

for all $n = 1, \ldots, N$ **do** $\boldsymbol{L}_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}$; $\gamma_n = \varphi(\boldsymbol{L}_n)$;

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\alpha_{m,n} = \gamma_n$;

Iteration Loop

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \left(\prod_{n' \in \mathcal{H}(m) \setminus n} \operatorname{sgn}(\alpha^{\text{SAT}}_{m,n'}) \right) \left(\min_{n' \in \mathcal{H}(m) \setminus n} |\alpha^{\text{SAT}}_{m,n'}| \right);$$

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**

$$\alpha_{m,n} = s_{\tilde{\mathcal{M}}} \left(s_{\tilde{\mathcal{M}}}(\gamma_n + \beta_{m_1,n}), \ldots + \beta_{m_{d_v-1},n} \right);$$

$$\alpha^{\text{SAT}}_{m,n} = s_{\mathcal{M}} \left(\alpha_{m,n} \right);$$

for all $n = 1, \ldots, N$ **do** ▷ **AP-update**

$$\tilde{\gamma}_n = s_{\tilde{\mathcal{M}}} \left(s_{\tilde{\mathcal{M}}}(\gamma_n + \beta_{m_1,n}), \ldots + \beta_{m_{d_v},n} \right);$$

for all $n = 1, \ldots, N$ **do** $\hat{x}_n = \operatorname{sgn}(\tilde{\gamma}_n)$; ▷ hard decision

if $\underline{\hat{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check

End Iteration Loop

- $\varphi(x)$ is the quantization map of x
- $\alpha^{\text{SAT}}_{m,n}$ is the value of $\alpha_{m,n}$ (output of the VNU) saturated to q-bit
- $s_{\mathcal{M}}$ is the q-bit saturation map defined by:

$$s_{\mathcal{M}}(z) = \begin{cases} -Q, & \text{if } z < -Q \\ z, & \text{if } z \in \mathcal{M} \\ +Q, & \text{if } z > +Q \end{cases}$$

where $Q \triangleq 2^{q-1} - 1$

The $\tilde{q}$-bit saturation map $s_{\tilde{\mathcal{M}}}$ is defined in a similar manner.

One of the contributions of this thesis, is to show that is spite of its simplicity, the gain factor quantizer provides performance close to the optimal quantizer (see Chapter 3).

2.4.2 Density evolution analysis

Density Evolution (DE) is a method where the evolution of probability density functions (pdf) of the exchanged messages is tracked through the message-passing algorithm. This is done by deriving recursive equations, referred to as DE equations, that express the pdf of the exchanged messages at iteration $\ell + 1$ of a function of their pdf at iteration ℓ. The method has been first introduced in [83], and it applies to MP decoders over both continuous and finite alphabets. The finite alphabet case

greatly simplifies the computation, since instead of tracking the pdf's of continuous random variables, one has to determine the recursion between the probability mass functions (pmf) of random variables with finite support.

The classical assumptions when deriving DE equations, are that (i) both the channel and the decoder are symmetric, and (ii) incoming messages to any check and variable node are independent. The first assumption allows reducing the analysis to the all-zero codeword, since the symmetry of both the channel and the decoder implies that the decoding performance is independent on the transmitted codeword. The second assumption holds as long as the number of iterations is less than half the girth of the graph. In practice, one assumes that both the girth of the graph and the length of the code go to infinity, thus (ii) always holds. It also follows that the DE equations predict the *asymptotic behavior* of an ensemble of LDPC codes, with both girth and code-length going to infinity [83]. The predicted behavior depends on the irregularity profile of the code, defined by the degree distribution polynomials (Eq. (2.2)), and the decoding algorithm. Even if the predicted behavior does not apply to finite-length codes, DE is a useful method for both code and decoding design, and we will make intensive use of this method in Chapter 3 for the analysis and design of code aware quantizers, and in Chapter 5 for the optimization of Non-Surjective Finite Alphabet Iterative Decoders (NS-FAIDs).

In this section, we derive the DE equations for the finite alphabet MS decoding. The equations are given for regular LDPC codes. The same method can be used for irregular LDPC codes, simply by averaging according to the degree distribution polynomials.

Let $\ell > 0$ denote the decoding iteration. Superscript (ℓ) will be used to indicate the messages and the a posteriori information computed at iteration ℓ. To indicate the value of a message on a randomly selected edge, we drop the variable and check node indexes from the notation (and we proceed in a similar manner for the a priori and a posteriori information). The corresponding probability mass functions are denoted as follows:

- $G(z) = \Pr(\gamma = z)$, $\forall z \in \mathcal{M}$,
- $\tilde{G}^{(\ell)}(\tilde{z}) = \Pr(\tilde{\gamma}^{(\ell)} = \tilde{z})$, $\forall \tilde{z} \in \tilde{\mathcal{M}}$,
- $A^{(\ell)}(z) = \Pr(\alpha^{(\ell)} = z)$, $\forall z \in \mathcal{M}$
- $B^{(\ell)}(z) = \Pr(\beta^{(\ell)} = z)$, $\forall z \in \mathcal{M}$

Since G is the pmf of the quantized input LLR values (γ_n), and assuming that the all zero codeword has been transmitted, it follows that the input error probability is given by:

$$P_e^{(0)} = \frac{1}{2}G(0) + \sum_{z=-Q}^{-1} G^{(\ell)}(z), \tag{2.10}$$

where the $\frac{1}{2}G(0)$ term accounts for the fact that $\text{sign}(0) = \pm 1$ with equal probability.

Similarly, at iteration $\ell > 0$, the probability that the decoder output is erroneous is given by:

$$P_e^{(\ell)} = \frac{1}{2}\tilde{G}(0) + \sum_{\tilde{z}=-\tilde{Q}}^{-1} \tilde{G}^{(\ell)}(\tilde{z}) \tag{2.11}$$

While G depends only on the channel model and the quantization of the input LLRs, pmf's $A^{(\ell)}$, $B^{(\ell)}$, and $\tilde{G}^{(\ell)}$ can be computed recursively as explained in the next sections. The following notation will be used throughout in the following sections, for any pmf C:

$$C_{[x,y]} \triangleq \sum_{z=x}^{y} C(z) \tag{2.12}$$

2.4.2.1 Expression of the input pmf G

The probability mass function G depends only on the channel and the quantization map $\varphi : \mathcal{Y} \to \mathcal{M}$, where $\mathcal{Y}$ is the channel output alphabet, and $\mathcal{M} = \{-Q, \ldots, +Q\}$ the input alphabet of the decoder. Assuming the BI-AWGN channel with noise variance σ^2, and the gain factor quantization map from Eq. (2.9), one has:

$$G(z) = \begin{cases} 1 - q\left(\frac{-Q+0.5-\mu}{\mu\sigma}\right), & \text{if } z = -Q \\ q\left(\frac{z-0.5-\mu}{\mu\sigma}\right) - q\left(\frac{z+0.5-\mu}{\mu\sigma}\right), & \text{if } -Q < z < +Q \\ q\left(\frac{Q-0.5-\mu}{\mu\sigma}\right), & \text{if } z = +Q \end{cases} \tag{2.13}$$

where $q(x) = \dfrac{1}{\sqrt{2\pi}} \displaystyle\int_x^{+\infty} \exp\left(-\frac{u^2}{2}\right) du$ is the tail probability of the standard normal distribution (also known as the *Q-function*).

Finally, we define $A^{(0)} \triangleq G$.

2.4.2.2 Expression of $B^{(\ell)}$ as a function of $A^{(\ell-1)}$

For the sake of clarity, the iteration index will be removed from the notation of exchanged messages and their probability mass functions. Hence, let $A \triangleq A^{(\ell-1)}$ and $B \triangleq B^{(\ell)}$. We proceed by recursion on $i = 2, \ldots, d_c - 1$, where d_c denotes the check-node degree. Let $\beta_1 = \alpha_1$, and define β_i for $i = 2, \ldots, d_c - 1$ by

$$\beta_i = \mathrm{sgn}(\beta_{i-1}) \cdot \mathrm{sgn}(\alpha_i) \cdot \min(|\beta_{i-1}|, |\alpha_i|) \tag{2.14}$$

and B_i by its probability mass function. At $z = 0$, $B_i(0)$ can be expressed as follows:

$$\begin{aligned} B_i(0) = \Pr(\beta_i = 0) =& A(0)B_{i-1}(0) + B_{i-1}(0)\Big(1 - A(0)\Big) \\ &+ A(0)\Big(1 - B_{i-1}(0)\Big) \end{aligned} \tag{2.15}$$

For $z > 0$, $B_i(z)$ is determined using the following steps. First, define $F_i^+(z)$ by

$$\begin{aligned} F_i^+(z) &= \Pr(\beta_i \geq z) \\ &= B_{i-1_{[z,Q]}} A_{[z,Q]} + B_{i-1_{[-Q,-z]}} A_{[-Q,-z]} \end{aligned} \tag{2.16}$$

Then, $B_i(z)$ for $z > 0$ is given by

$$B_i(z) = \Pr(\beta_i = z) = \begin{cases} F_i^+(z) - F_i^+(z+1), & \text{if } z < +Q \\ F_i^+(z), & \text{if } z = +Q \end{cases} \tag{2.17}$$

Similarly, we proceed for $z < 0$. Hence, define $F_i^-(z)$ by

$$\begin{aligned} F_i^-(z) &= \Pr(\beta_i \leq z) \\ &= B_{i-1_{[-z,Q]}} A_{[-Q,z]} + B_{i-1_{[-Q,z]}} A_{[-z,Q]} \end{aligned} \tag{2.18}$$

Then, $B_i(z)$ for $z < 0$ is given by

$$B_i(z) = \Pr(\beta_i = z) = \begin{cases} F_i^-(z) - F_i^-(z-1), & \text{if } z > -Q \\ F_i^-(z), & \text{if } z = -Q \end{cases} \tag{2.19}$$

Finally, we have $B = B_{d_c - 1}$.

2.4.2.3 Expressions of $A^{(\ell)}$ and $\tilde{G}^{(\ell)}$ as functions of $B^{(\ell)}$ and G

As in the previous section, we drop the iteration index for clarity. Hence, let $A \triangleq A^{(\ell)}$, $B \triangleq B^{(\ell)}$ and $\tilde{G} \triangleq \tilde{G}^{(\ell)}$. We proceed by recursion on $i = 0, \ldots, d_v$, where d_v denotes the variable-node degree. For $i = 0$, we make the following definitions:

$$\begin{aligned} \tilde{\omega}_0 &\triangleq \gamma \in \mathcal{M} \subseteq \tilde{\mathcal{M}} \\ \tilde{\Omega}_0(\tilde{z}) &\triangleq \Pr(\tilde{\omega}_0 = \tilde{z}) = \begin{cases} G(z), & \text{if } \tilde{z} \in \mathcal{M} \\ 0, & \text{if } \tilde{z} \in \tilde{\mathcal{M}} \backslash \mathcal{M} \end{cases} \end{aligned} \tag{2.20}$$

For $i = 1, \ldots, d_v$, we make the following definitions:

$$\omega_i \triangleq \tilde{\omega}_{i-1} + \beta_{m_i,n} \in \mathbb{Z}, \tag{2.21}$$

$$\Omega_i(\omega) \triangleq \Pr(\omega_i = \omega) = \sum_u \tilde{\Omega}_{i-1}(u) B(\omega - u), \forall \omega \in \mathbb{Z}, \tag{2.22}$$

$$\tilde{\omega}_i \triangleq s_{\tilde{\mathcal{M}}}(\omega_i) \in \tilde{\mathcal{M}} \tag{2.23}$$

$$\tilde{\Omega}_i(\tilde{\omega}) \triangleq \Pr(\tilde{\omega}_i = \tilde{\omega}) = \begin{cases} \Omega_i(\tilde{\omega}), & \text{if } \tilde{\omega} \in \tilde{\mathcal{M}} \backslash \{+\tilde{Q}, -\tilde{Q}\} \\ \sum_{\omega \leq -\tilde{Q}} \Omega_i(\omega), & \text{if } \tilde{\omega} = -\tilde{Q} \\ \sum_{\omega \geq +\tilde{Q}} \Omega_i(\omega), & \text{if } \tilde{\omega} = +\tilde{Q} \end{cases} \tag{2.24}$$

Finally, we have

$$A = s_{\mathcal{M}}(\tilde{\Omega}_{d_v - 1}) \tag{2.25}$$

$$\tilde{G} = \tilde{\Omega}_{d_v} \tag{2.26}$$

In Eq. (2.25) above, applying the saturation operator $s_{\mathcal{M}}$ on the probability mass function $\tilde{\Omega}_{d_v-1}$ means that all the probability weights corresponding to values $\tilde{\omega}$ outside $\mathcal{M}$ must be accumulated to the probability of the corresponding boundary value of $\mathcal{M}$ (that is, either $-Q$ or $+Q$, according to whether $\tilde{\omega} < -Q$ or $\tilde{\omega} > +Q$).

2.4.2.4 Asymptotic error probability and noise threshold

Using the DE equations one may recursively determine the pmf's of the exchanged messages and of the AP-LLLs, and thus the decoding error probability $P_e^{(\ell)}$, at any iteration $\ell > 0$:

$$P_e^{(\ell)} = \frac{1}{2}\tilde{G}(0) + \sum_{\tilde{z}=-\tilde{Q}}^{-1} \tilde{G}^{(\ell)}(\tilde{z}) \tag{2.27}$$

The asymptotic error probability is defined by:

$$P_e^{(+\infty)} = \lim_{\ell \to +\infty} P_e^{(\ell)} \tag{2.28}$$

Even if the limit is taken only over ℓ, it is worth noticing that $P_e^{(+\infty)}$ actually corresponds to the error probability when both the code-length and the number of iterations go to infinity. This follows from the fact that the DE recursion is only valid as long as incoming messages to any variable or check node are independent, which holds when the girth of the graph, and thus the code length, go to infinity. We also note that $P_e^{(+\infty)}$ actually depends on (i) the channel, (ii) the quantization, (iii) the irregularity profile of the LDPC code.

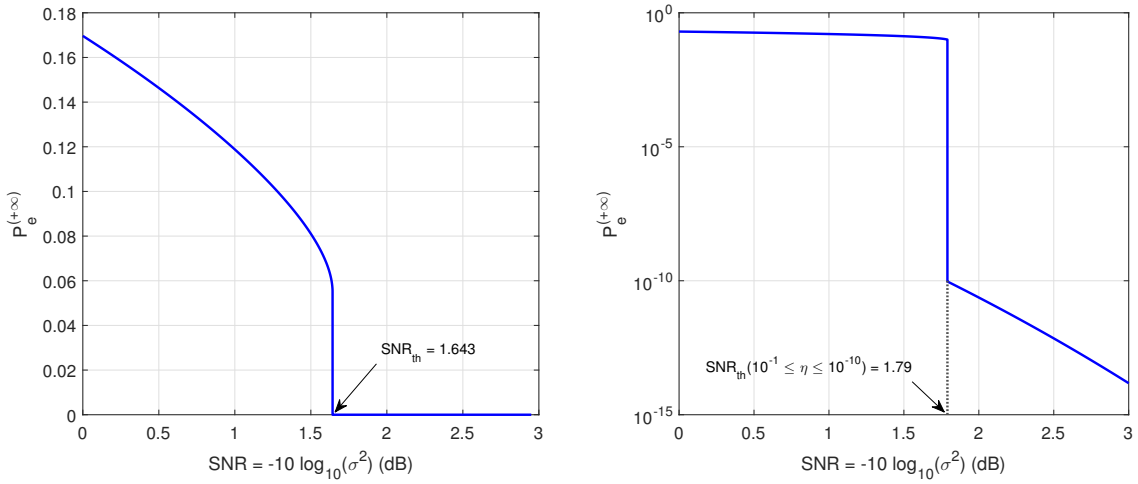
Finally, the noise threshold, denoted by σ^2_{th}, is defined as:

$$\sigma^2_{\text{th}} = \sup\{\sigma^2 \mid P_e^{(+\infty)} = 0\} \tag{2.29}$$

It corresponds to the maximum channel noise for which the bit error probability can be made arbitrarily small, assuming that both the code length and the number of iterations go to infinity.

To illustrate the threshold phenomenon, Figure 2.7(a) shows the asymptotic error probability $P_e^{(+\infty)}$ as function of the Signal to Noise Ratio ($\text{SNR} = -10\log_{10}(\sigma^2)$), for the ensemble of regular LDPC codes with variable-node degree $d_v = 3$ and check-node degree $d_c = 6$. We have considered the finite-alphabet MS decoder, with input LLR values and exchanged messages quantized on $q = 4$ bits, and AP-LLR values quantized on $\tilde{q} = 6$ bits. The gain factor used for the quantization of the input LLRs is $\mu = 5.6$. The SNR threshold is equal to 1.643 and it separates the SNR region where $P_e^{(+\infty)} > 0$ (on its left) from that where $P_e^{(+\infty)} = 0$ (on its right).

It is worth noticing that for finite alphabet decoders, the asymptotic error probability $P_e^{(+\infty)}$ may exhibit an error floor phenomenon, which is attributable to specific dynamics of the DE recursion. This may happen especially for irregular codes. To illustrate this phenomenon, Figure 2.7(b) shows the asymptotic error probability $P_e^{(+\infty)}$ as function of the SNR, for the ensemble of irregular LDPC codes with edge-perspective degree distribution polynomials $\lambda(x) = 0.2895x + 0.3158x^2 + 0.3947x^5$ and $\rho(x) = 0.6316x^5 + 0.3684x^6$ (these polynomials correspond to the irregularity profile of the irregular LDPC code with rate 1/2 specified by the IEEE 802.16e (WiMAX) standard [46]). We have considered the finite-alphabet MS decoder with $(q, \tilde{q}) = (4, 6)$ quantization, while the gain factor used for the quantization of the input LLRs is $\mu = 1.4$. To take into account the asymptotic error floor phenomenon,



(a) $(3, 6)$-regular LDPC code (b) Irregular LDPC code (WiMAX, rate 1/2)

Figure 2.7: Asymptotic error probability $P_e^{(+\infty)}$ as function of the SNR

the noise threshold can be alternatively defined by requiring $P_e^{(+\infty)}$ to be below a target error probability. This leads to the following definition, referred to as η-threshold, where $\eta \geq 0$ is the target error probability:

$$\sigma^2_{\text{th}}(\eta) = \sup\{\sigma^2 \mid P_e^{(+\infty)} \leq \eta\} \tag{2.30}$$

The value of the η-threshold is shown in Figure 2.7(b), for a target error probability $\eta = 10^{-10}$. Note that at this point, $P_e^{(+\infty)}$ value actually drops from 10^{-1} to 10^{-10}. Hence, the η-threshold value is actually the same for any $10^{-10} \leq \eta \leq 10^{-1}$.

2.5 Scheduling strategies

This section discuses scheduling strategies for message-passing (MP) decoders. MP decoders may deal with different scheduling strategies, according to the order in which variable and check-node messages are updated during the message passing iterative process. The classical convention is that, at each iteration, all check-nodes and subsequently all variable-nodes pass new messages to their neighbors. This message-passing schedule is usually referred to as *flooded scheduling* [54]. A different approach is to split the parity-check matrix in several horizontal layers, then process horizontal layer sequentially, while check-nodes (rows) within the same layer are processed by using a flooding schedule strategy. Each time a layer is processed the decoder updates the neighbor variable-nodes, so as to profit from the propagated messages, and then proceeds to the next layer. This message scheduling, known as *layered scheduling* [40], propagates information faster and converges in about half the number of iterations compared to the flooded scheduling, thus yielding a lower decoding latency [109].

2.5.1 Flooded scheduling

We first discuss the flooded scheduling. In this case, exchanged messages are passed along the edges of the Tanner graph, first from variable to check nodes, and then from check to variable nodes. Considering for instance the MS decoding, this corresponds exactly to the description from Algorithm 2. A generic hardware architecture of a flooded MP decoder, with all variable and check node processing units instantiated in hardware, is shown in Figure 2.8. The process is split into two phases. In the first half of each iteration (phase one), all check-node units (CNUs) are processed and all check-node messages ($\beta_{m,n}$) are sent from check to variable-nodes. At the second half of each iteration (phase two) all variable-node units (VNUs) are processed and all variable-node messages ($\alpha_{m,n}$) are sent from variable to check-nodes. Therefore, this scheduling strategy is also referred to as *two phases message-passing decoding*. Messages can be exchanged through an interconnection network (connecting variable and check nodes), or through a shared memory, as shown in Figure 2.8.

In case that processing units are instantiated in hardware for all variable and check-nodes, in which case the decoder is said to be *fully parallel*, VNUs and CNUs can be directly connected through an interconnection network. An alternative solution is to instantiate in hardware only a reduced number of VNUs and CNUs, in which case the decoder is said to be *partially parallel*, and to reuse them to process all the variable and check nodes. In this case messages can be exchanged through the use of a shared memory. We discuss below the main advantages together with the main drawbacks of the flooded scheduling strategy. Note that more details related to hardware implementation aspects will be discussed in Section 2.6.2.

- Fully parallel flooded decoders exploit the maximum hardware parallelism, and thus are able to achieve very high throughput. Each decoding iteration can be executed in 2 clock cycles, corresponding tho the two aforementioned

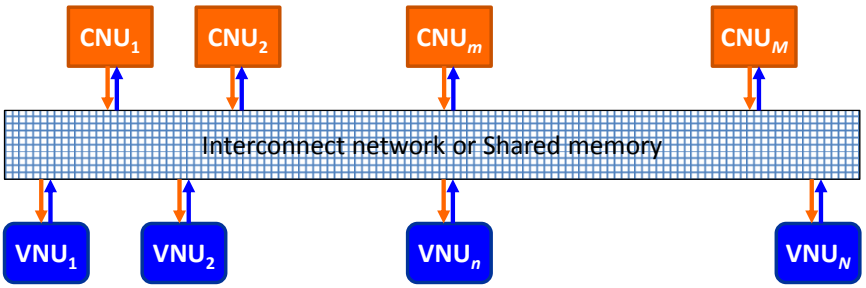


Figure 2.8: Message-passing decoding with flooded scheduling

decoding phases. However, the main bottleneck is the interconnection network of a fully parallel decoder, and in practice fully parallel implementations are limited to codes of a few hundreds of bits length.

- Partially parallel flooded decoders may represent a solution to the interconnection network problem, but they require the use of large memories for the storage of the input information and of the exchanged messages in both directions (*i.e.*, both $\alpha_{m,n}$ and $\beta_{m,n}$).

- Flooded decoders (both fully and partially parallel) make an inefficient use of the hardware resources, since that VNUs and CNUs can not work in parallel (VNUs have to wait for the execution of CNUs, and vice versa).

- The lack of flexibility represents another drawback, especially for fully parallel decoders, since each implementation is specific to only one given code.

2.5.2 Layered scheduling

Unlike the flooded scheduling, in case of layered scheduling, the parity-check matrix $\mathbf{H}$ of size $M \times N$ is partitioned in L horizontal layers, also refereed to as *decoding layers* (see Figure 2.9). Each decoding layer contains M/L consecutive rows of $\mathbf{H}$, such that any variable-node is connected at most once to any layer. We denote by $\mathcal{M}_\ell$ is the set of consecutive rows of $\mathbf{H}$ corresponding to layer $\ell \in \{1, \ldots, L\}$. To help understanding the layered decoding principle, the MS decoding with layered scheduling is described in Algorithm 7. For the sake of simplicity we consider the unquantized version of the decoder, but the quantized (finite alphabet) version follows the same principle.

Initialization

- The decoding starts by initializing the a posteriori information $\tilde{\gamma}_n = \gamma_n$ and the check-node messages $\beta_{m,n} = 0$.

Iterations

- During the iterative decoding process, the variable-node messages $\alpha_{m,n}$ are computed by subtracting the check-node messages incoming on the same edge

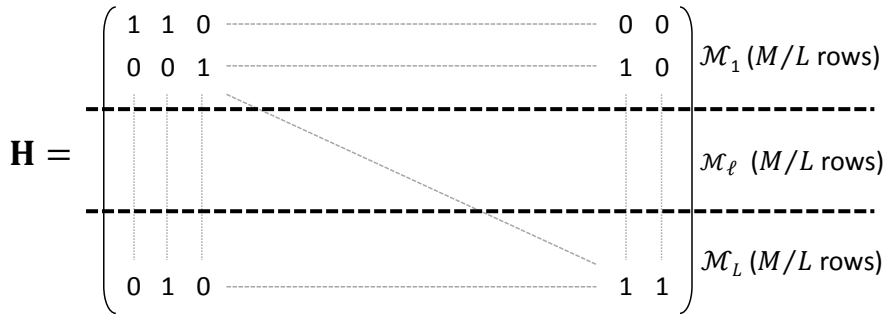


Figure 2.9: Parity-check matrix with layered scheduling

from the a posteriori information of variable node n, that is $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$ (VN-processing step). These messages are only computed for the variable nodes n connected to check-nodes m in the current decoding layer $\mathcal{M}_\ell$.

- Messages outgoing from check-nodes $m \in \mathcal{M}_\ell$ are then computed (CN-processing step), and used to update the $\tilde{\gamma}_n$ values, $n \in \mathcal{H}(m)$ (AP-update step).

  Hence, subsequent layers will use updated valuers of $\tilde{\gamma}_n$, thus ensuring a faster convergence of the iterative decoding process.

The fact that layered scheduling converges faster than flooded scheduling can be explained as follows: AP-LLR values of variable-nodes n connected to a given layer are updated after the check-nodes in that layer have been processed. When the next layer is processed, the incoming check-node messages ($\alpha_{m,n}$) are derived from the AP-LLR values of the corresponding variable-nodes. Hence, incoming messages incorporate the contribution of the check-nodes from the previous layers (even those processed in the same iteration). This yield a faster propagation of messages through the decoding graph, thus speeding up the decoder convergence. In case the maximum number of decoding iterations is relatively low (as it may be the case in practical applications), the faster convergence speed may also translate into an improved decoding performance. In any case, the faster convergence speed yields a reduction of the average number of decoding iterations performed by the decoder, which in turn translates into energy savings (assuming that an early stopping criterion, *e.g.*, syndrome check, is implemented by the decoder).

A generic hardware architecture of a layered MP decoder, is shown in Figure 2.10. The number of VNUs and CNUs instantiated in hardware is equal to the number of rows per decoding layer (denoted in the figure by $\omega \triangleq M/L$). Concerning memory requirements, such an architecture only need to store the AP-LLR values $\tilde{\gamma}_n$ and the check-node messages $\beta_{m,n}$, thus resulting in significant memory reduction as compared to partially parallel flooded scheduling based solutions.

Algorithm 7 Layered MS decoding algorithm

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word

Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization

for all $n = 1, \ldots, N$ **do** $\tilde{\gamma}_n = \gamma_n = \boldsymbol{L}_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}$;

for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** $\beta_{m,n} = 0$;

Iteration Loop

for all $\ell = 1, \ldots, L$ **do** ▷ Loop over horizontal layers

for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **VN-processing**

$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$;

for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**

$$\beta_{m,n} = \prod_{n' \in \mathcal{H}(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \cdot \min_{n' \in \mathcal{H}(m) \setminus n} |\alpha_{m,n'}|;$$

for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **AP-update**

$\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}$;

end (horizontal layers loop)

for all $n = 1, \ldots, N$ **do** $\hat{x}_n = \dfrac{1 - \operatorname{sgn}(\tilde{\gamma}_n)}{2}$; ▷ **hard decision**

if $H \cdot \underline{\hat{\boldsymbol{x}}} = 0$ **then** exit iteration loop ▷ syndrome check

End Iteration Loop

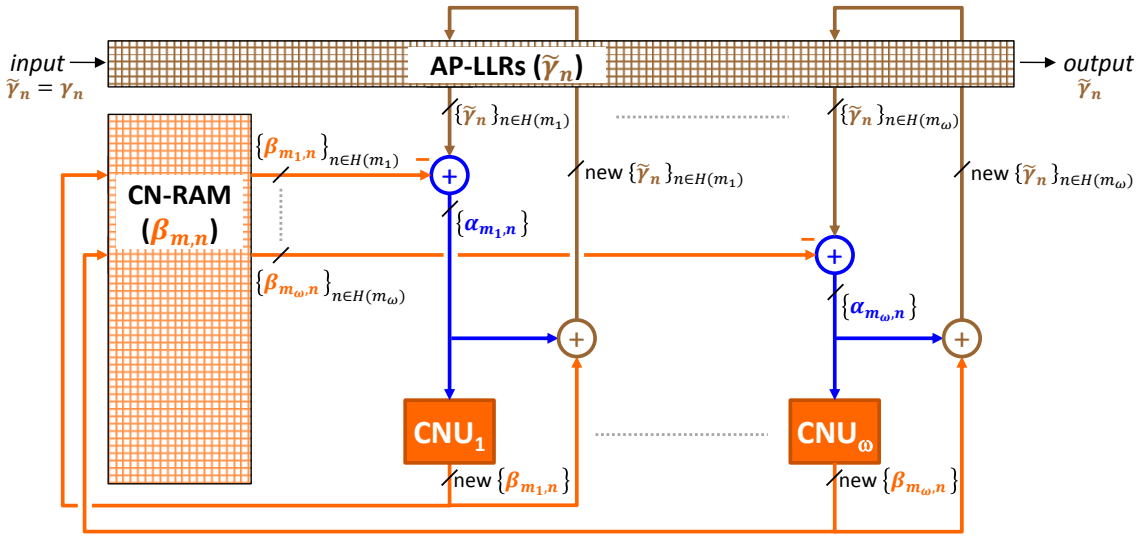


Figure 2.10: Message-passing decoding with layered scheduling

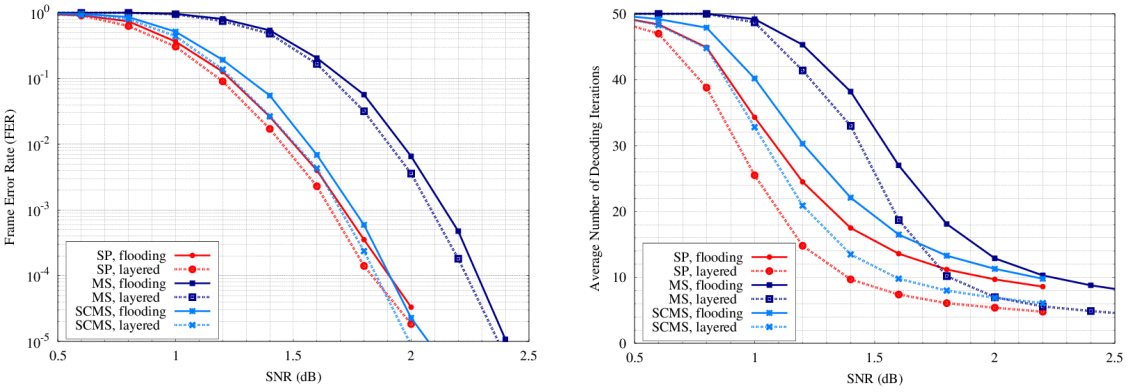
Layered scheduling for QC-LDPC codes

Layered scheduling applies when the parity-check matrix can be split in horizontal layers. This is the case of QC-LDPC codes, due their construction based on circulant matrices. A decoding layer is usually comprised of the z consecutive rows of the parity check-matrix $\mathbf{H}$, corresponding to one row of the base matrix $\mathbf{B}$, where z denotes the expansion factor (Section 2.2.2). More generally, it is possible to define a decoding layer as corresponding not only to one row of $\mathbf{B}$, but to a number of consecutive rows of $\mathbf{B}$, as long as these rows do not overlap. Clearly, due to the fact that each non-negative entry of $\mathbf{B}$ is replaced by a circulant (permutation) matrix, any column of $\mathbf{H}$ has at most one non-zero entry in each decoding layer.

Another advantage of QC-LDPC codes is that the routing network, used to route the AP-LLRs and check-node messages to the appropriate processing units, can be simply implemented by barrel shifters. This significantly simplifies the implementation of the layered architecture from Figure 2.10. Due to the numerous advantages of layered architectures, QC-LDPC codes with layered message passing decoders became the predominant solution for most practical applications [9].

Figure 2.11 shows Monte-Carlo simulation results over the BI-AWGN channel, for WiMAX-irregular LDPC code with rate of 1/2, by using different decoders discussed in the previous sections, namely BP (SP), MS, and SCMS, with either flooded and layered scheduling. Results are provided in terms of Frame Error Rate (FER) performance, as well as average number of decoding iterations. The maximum number of decoding iterations is set to 50. We can observe the following:

- In terms of error correction performance, BP and SCMS decoders provide similar FER performance, outperforming the MS decoding by $\approx 0.4\,\mathrm{dB}$.
- Layered scheduling provide slightly better error correction performance than flooded scheduling, due the its faster convergence. It is worth noticing that the maximum number of decoding iterations (50) is sufficiently high to narrow the



(a) Frame Error Rate (FER)

(b) Convergence speed

Figure 2.11: Error correction performance and convergence speed of various MP decoder, with flooded and layered scheduling

gap between the two scheduling strategies. However, this gap should increase with decreasing maximum number of decoding iterations.

- For SNR values in the second part of the waterfall region, layered scheduling can converge approximately twice faster than flooded scheduling.

2.6 From decoding algorithms to their hardware implementation

2.6.1 Algorithmic choices

In pratical LDPC decoder implementations (*e.g.*, FPGA or ASIC designs) [37, 67], one the major concerns to be addressed is the design of a decoder architecture that yields small area, low power consumption, fast processing speed and good error correction capacity. Unfortunately, it is very hard to meet all the above criteria simultaneously. Therefore, depending on a specific application, one may consider different trade-offs between the aforementioned figure of merits of the design.

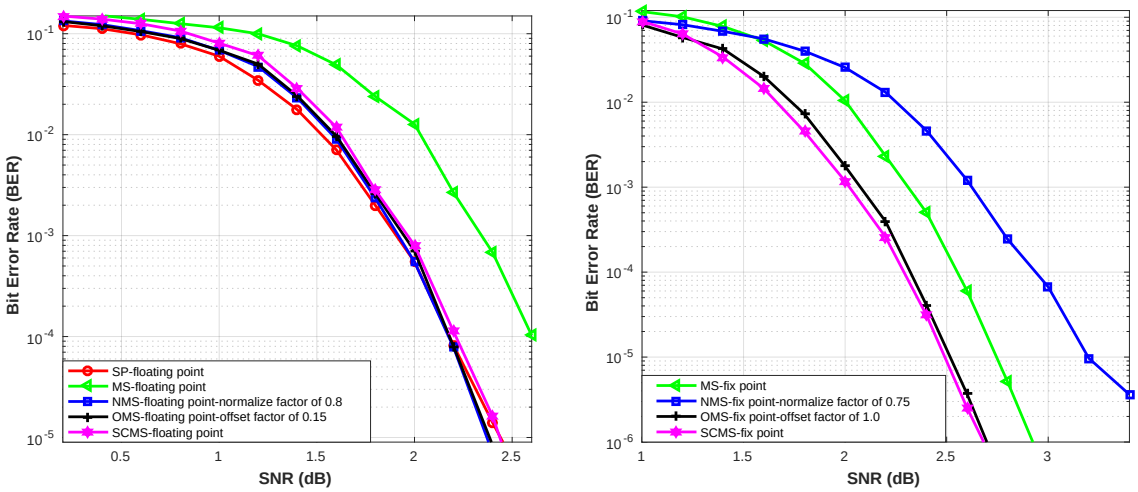
Such as trade-off usually starts with algorithmic choices: choice of an LDPC code, of a decoding algorithm, of a quantization scheme, or even of a scheduling strategy. We note however that the scheduling strategy has such a significant impact on the hardware architecture, that it can be actually considered as both an algorithmic and architectural choice. Then a number of architectural choices have to be made, *e.g.*, related to degree of parallelism of the architecture (*i.e.*, number of processing units operating in parallel) and that of the processing units (*i.e.*, number of inputs processed by clock cycle), messages storing and routing (*e.g.*, choice of memory type, number of read/write ports, etc.), pipeline stages, etc. Of course, algorithmic and architectural choices are not completely decorrelated, since for instance the choice of the LDPC code and of the scheduling strategy will also impose a number of architectural choices.

In this section, we discuss various algorithmic choices, related to the decoding algorithm, the quantization scheme, the scheduling strategy, and the number of iterations. Architectural choices, with a focus on high-throughput applications, will be discussed in Section 2.6.2.

2.6.1.1 Decoding algorithm

The main criteria that dictate the choice of a MP decoding algorithm consist of its computational complexity, and its error correction performance, which can be expressed in terms of either Bit Error Rate (BER) or Frame Error Rate (FER). By way of example, we consider the BER performance of several MP decoding algorithms, for a $(3, 6)$-regular QC-LDPC code, with base matrix of size $R \times C = 12 \times 24$, and expansion factor $z = 54$. Hence, the parity check matrix $\mathbf{H}$ is of size $M \times N$, with $M = zR = 648$ and $N = zC = 1296$.

Figure 2.12(a) shows the BER performance of BP (SP), MS, NMS (with optimal normalized factor of 0.8 [15]), OMS (with optimal offset factor of 0.15 [15]), and SCMS decoding algorithms (layered scheduling is considered, and the number of decoding iterations is set to 20). At BER $= 10^{-4}$, MS decoding suffers a decoding loss of approximately 0.4 dB compared to SP, NMS, OMS and SCMS decoding. As mentioned in Section 2.3.3, although SP achieves excellent decoding performance, it is not suitable for practical implementations due to the complexity of check-node processing. Thus, practical LDPC decoder implementations mostly rely on the MS



(a) Floating point Message-Passing decoders

(b) MS-based finite precision decoders

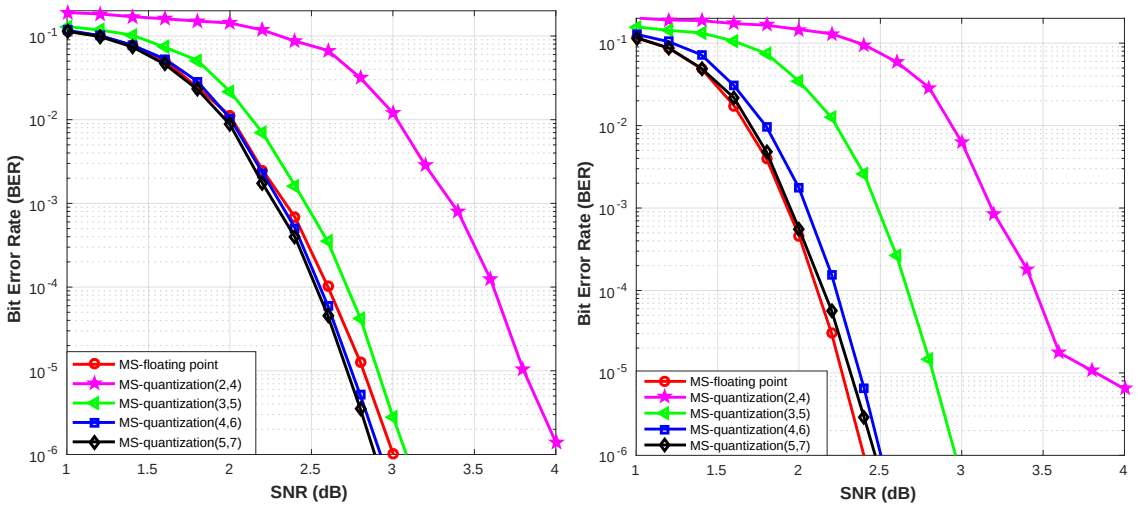
Figure 2.12: BER performance of various decoding algorithms for $(3, 6)$-regular QC-LDPC code, with code-length $N = 1296$

or MS-based decoding algorithms, which might come at the price of an acceptable decoding performance loss. Figure 2.12(b) shows BER curves for finite precision (finite alphabet) MS and MS-based (NMS, OMS, SCMS) decoders, which are more suitable for hardware implementations. These decoders use 4-bit quantization for the exchanged messages, and 6-bit quantization for the AP-LLR values. It is worth noting that the finite precision NMS exhibits worst error correction performance than the finite precision MS. Thus, for practical hardware implementations, MS OMS, and SCMS decoding are good choices. The quantization of finite precision decoders is further discussed in the next section.

2.6.1.2 Quantization

The size of the quantization for finite-precision decoders influences significantly the memory requirements (used to store the quantized messages), as well as the size of the interconnection network (used to route messages from memory to processing units), and the complexity of the processing units. In terms of error correction performance, it is worth mentioning that the quantization is one of the factors that may aggravate the error floor phenomenon. Thus, in practical implementations of finite precision decoders, in order to trade-off between hardware complexity and decoding performance, it is very important to decide how many quantization bits are needed to represent the finite precision values.

Let MS-quantization$(q, \tilde{q})$ be the Finite-Alphabet Min-Sum decoding (Section 2.4.1), with input LLRs and exchanged messages are quantized on q-bits, and AP-LLRs quantized on $\tilde{q}$-bits ($\tilde{q} > q$). Figure 2.13 shows BER curves of Min-Sum decoding according to several different quantization for (3,6)-regular and irregular WiMAX codes.



(a) (3,6)-regular LDPC code, $N = 1296$ (b) WiMAX-irregular LDPC code, $N = 2304$

Figure 2.13: Impact of the quantization to BER of LDPC decoders

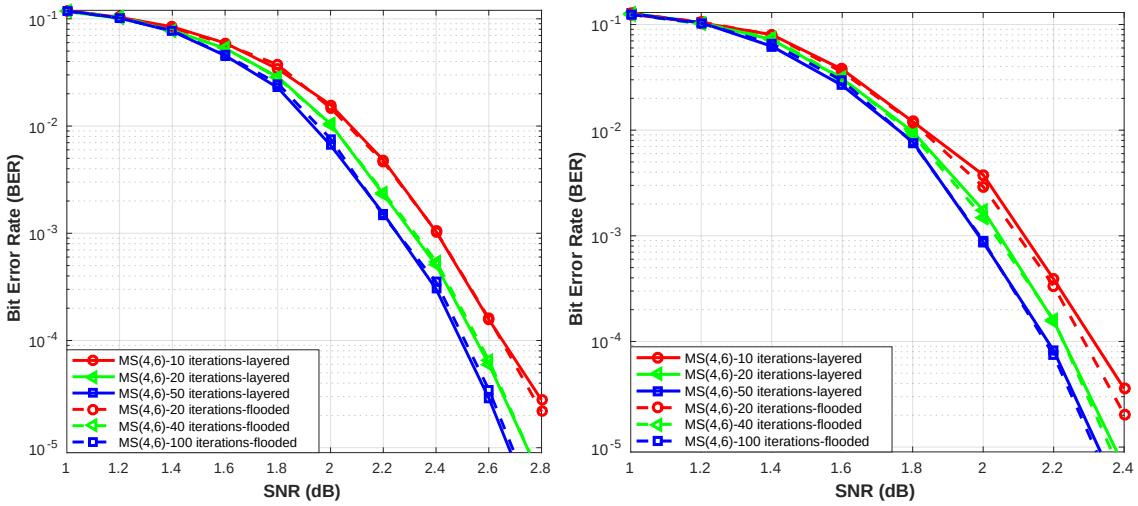
It can be observed that:

- The gap between quantization $(q, \tilde{q}) = (2, 4)$ and the other quantization schemes is quite significant, of approximately 1 dB from the $(q, \tilde{q}) = (3, 5)$ quantization (at BER $= 10^{-5}$).
- The gap between the other quantization schemes narrows with increasing quantization size. In particular, quantization schemes $(q, \tilde{q}) = (4, 6)$ and $(q, \tilde{q}) = (5, 7)$ exhibit decoding performance very close to the infinite precision decoder, and even outperform it for the (3,6)-regular code.
- The gap between $(q, \tilde{q}) = (4, 6)$ and $(q, \tilde{q}) = (5, 7)$ quantization schemes is very small, and may be considered negligible, for both (3,6)-regular and WiMAX code. Therefore, $(q, \tilde{q}) = (4, 6)$ quantization is a good choice for practical implementations, since it results in lower hardware complexity, as discussed above.

According to the above analysis, $(q, \tilde{q}) = (4, 6)$ is the main quantization scheme that will be used throughout this thesis.

2.6.1.3 Scheduling strategy

In this section, we investigate the effect of the scheduling strategy on the decoding performance, through Monte-Carlo simulations. Figure 2.14 shows BER curves for finite-alphabet MS decoding, with $(q, \tilde{q}) = (4, 6)$ quantization, using both flooded and layered scheduling. We compare layered scheduling with maximum number of iterations I_{layered}, with flooded scheduling with maximum number of iterations $I_{\text{flooded}} = 2I_{\text{layered}}$. From simulation results, it can be easily seen that at the same target BER, layered scheduling improves convergence speed by nearly 2, as compared



(a) (3,6)-regular LDPC code, $N = 1296$ (b) WiMAX-irregular LDPC code, $N = 2304$

Figure 2.14: Impact of the scheduling strategy to BER of LDPC decoders

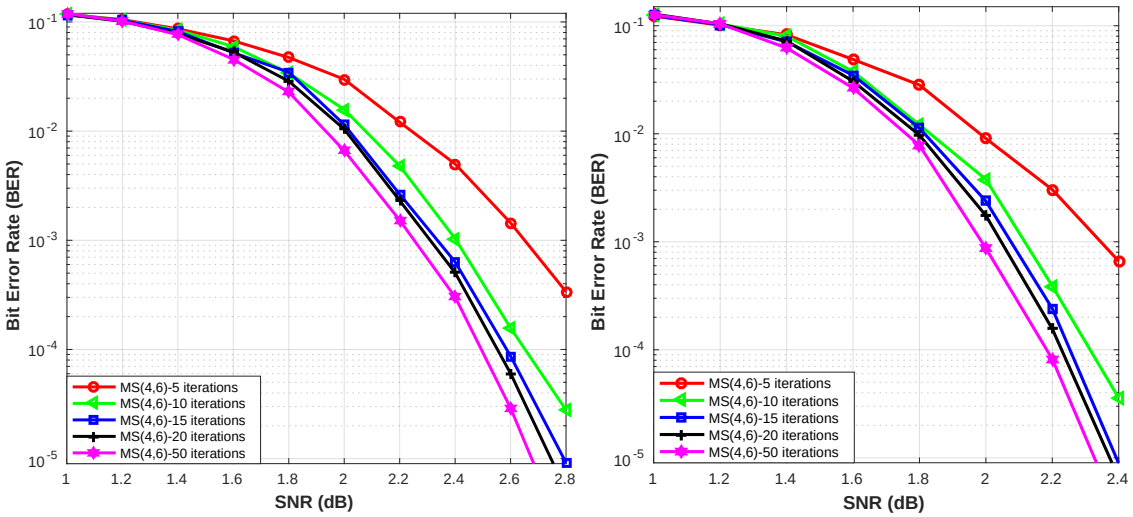
to the flooded one. Besides, layered scheduling advantageously applies to QC-LDPC codes [34], which are naturally equipped with a layered structure, and also known to significantly reduce the complexity of the interconnection network. Due to their benefits in terms of area/throughput/flexibility, layered QC-LDPC decoders have been widely adopted, and can be considered as a de facto standard solution in most applications [9].

2.6.1.4 Number of iterations

Figure 2.15 shows the BER performance of the finite-alphabet MS-quantization$(4, 6)$ decoder, as a function of the maximum number of decoding iterations. It can be observed that increasing the number of iterations improves decoding performance. However, this improvement reaches a limit, after which extra iterations provide only little benefit. Most of the practical implementations that will be reported in this thesis use a maximum number of decoding iterations equal to 20.

It is also worth noticing that the maximum number of iterations has little impact on the average number of decoding iterations performed by the decoder. Indeed, most practical implementations have an *early termination criterion* [27, 59], which detects convergence (*e.g.*, by checking the syndrome) and may stop the decoder before the maximum number of iterations is reached. However, the maximum of number iterations determines the worst case latency of the decoder.

If the decoder is required to have a fixed latency (which is actually the case is most of practical applications), this latency is necessarily equal to the worst case latency, which therefore determines the throughput of the decoder. A different possibility would be to allow some variation of the decoding latency, but to use additional buffers for the input and output data, such that make the overall latency fixed (from the moment when data is written to the input buffer, to the moment when the cor-



(a) (3,6)-regular LDPC code, $N = 1296$ (b) WiMAX-irregular LDPC code, $N = 2304$

Figure 2.15: Impact of the number of iterations to BER of LDPC decoders

responding result is read from the output buffer). However, such a solution requires an additional and non-negligible cost, determined by the size of the input and output buffers. The throughput achieved by the hardware implementations proposed in this thesis, will always be reported for the maximum number of decoding iterations, which is sometimes referred to in the literature as *worst case throughput*.

Table 2.1 summarizes the different algorithmic aspects discussed in this section, the impact they may have on the hardware complexity and decoding performance, as well as the main issues to be considered.

Table 2.1: Factors impact to hardware complexity and decoding performance

Items	**Hardware complexity**	**Decoding performance**	**Main issues to be considered**
Algorithm	✓	✓	• Processing units • Memory requirements • Error correction capacity
Quantization	✓	✓	• Processing units • Memory requirements • Interconnections • Error floor phenomenon
Scheduling	✓		• Interconnections
No. iterations		✓	• Error correction capacity • Energy consumption • Worst case throughput

2.6.2 State-of-the-art on hardware implementations

In this section, we discuss several LDPC decoder hardware architectures, with a focus on low-cost, high-throughput implementations. We also provide a state of the art on ASIC designs, and discuss the trade-off between cost, throughput and power.

2.6.2.1 LDPC decoder architectures

A typical hardware architecture of an LDPC decoder consists of several variable-node units (VNUs) and check-node units (CNUs), optional memories for a priori/posteriori LLRs and exchanged messages, and an interconnection (routing) network [7, 26], as shown in Figure 2.16. In case of MS-based decoders, the VNUs' main operation is represented by addition, while the CNUs compute the first and second minimum values among the input values (*e.g.*, by employing a tree of comparators). Note that VNUs and CNUs can be interconnected in many different ways (depending on the hardware architecture of the decoder) and sometimes they can be merged into a single processing unit.

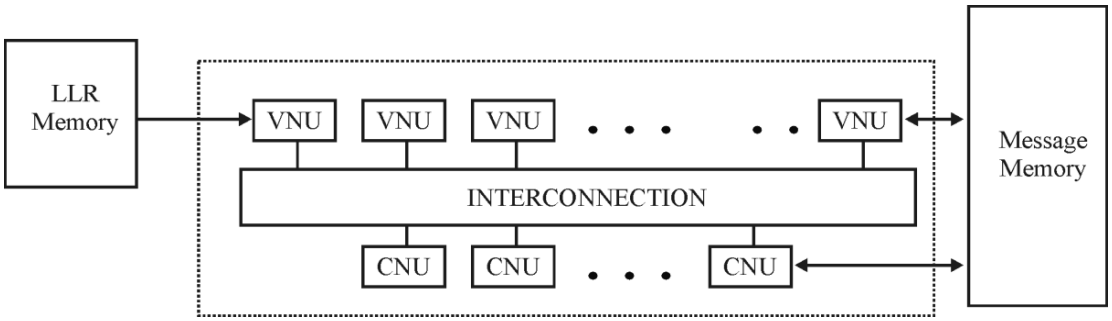


Figure 2.16: General hardware architecture of an LDPC decoder

Generally, LDPC decoder architectures can be classified into three main categories, namely *fully parallel*, *serial*, and *partly parallel* architectures. Fully parallel architectures are naturally implementing a flooded scheduling, as discussed in Section 2.5.1. Such implementations are characterized by a very large number of CNUs and VNUs and a dense interconnection network. Due to the large, and usually highly irregular, interconnection network, fully parallel decoders pose significant problems in the place-and-route or wiring the decoder. Therefore, fully parallel implementations have been seldom adopted in practical implementations.

At the opposite design spectrum, serial architectures uses only one VNU and one CNU, which are then reused to process all the variable- and check-nodes of the bipartite graph. Message passing is implemented by storing the computed messages in a dedicated memory, and reading them from the memory whenever they are needed by another processing unit. Serial architectures have a number of advantages: (i) very low cost, (ii) no routing congestion problems, (iii) may implement any of the flooded or layered scheduling, which may be simply controlled by the control unit, (iv) high flexibility, as they support a large variety of codes. But they also have a

significant drawback: they have a very low throughput, which may be way too low for practical applications in modern communication systems, characterized by an increased need for higher and higher data rates.

The partially parallel architecture inherits the main features and advantages of the two above architectures. In the partially parallel architecture, the number of VNUs and CNUs instantiated in hardware is lower than the number of variable and check nodes, and multiple nodes share a same processing unit. Different trade-offs between area, throughput and flexibility can be obtained by adjusting the number of processing units. Although the concept of partly parallel architecture is not necessarily related to a specific decoding scheduling, it is naturally suited to the layered scheduling strategy. Consequently, most of the partly parallel architectures are actually partly parallel layered architectures, using structured QC-LDPC codes, in order to further reduce the complexity of the interconnection network [24, 88–90]. The characteristics of the fully parallel, serial and partly parallel architectures are summarized in Table 2.2.

Table 2.2: Main characteristics of decoder architectures

Decoder architecture	**Area**	**Throughput**	**Flexibility**
Fully parallel	Large	Very high	✗
Serial	Small	Very low	✓
Partly parallel	Medium	High	✓

2.6.2.2 High-throughput optimizations

Cost, throughput and power optimizations mainly rely on architectural choices. The choice of the hardware architecture (fully parallel, partially parallel, or serial) is one of them. The parallelization degree of the partially parallel architecture (*i.e.*, the number of processing units instantiated in hardware) is another. One possible way to achieve even higher throughput is to unroll hardware resources, *i.e.*, to instantiate dedicated hardware for each iteration and then pipeline such hardware [87]. Unrolling hardware (HW) resources further increase the throughput, at the price of a significant increase in the area. Low-power optimizations have also been proposed, based either on an interconnect-driven code design approach to eliminate the need for a complex interconnection network [66, 93], or on the early detection of the iterative decoding's convergence, to terminate the computations, thereby reducing the dynamic power [93]. Further optimizations can be obtained by considering different processing unit architectures, *e.g.*, implementing different decoding algorithms or processing the input data in either a serial or a parallel manner [9].

Regarding the parallelisation of the processing units (*i.e.*, number of inputs processed by clock cycle), highest flexibility can be achieved by using serial processing units: VNUs and CNUs process incoming messages in a serial manner, which makes their implementation independent of the variable or check-node degree [9]. However, this comes at the cost of a reduced throughput. A practical solution to increase throughput, consists in the use of partly parallel layered LDPC decoder

architectures, with fully parallel processing units. Further throughput increase can be obtained by pipelining the datapath, which also ensures an efficient use of the hardware resources. However, the number of stages in the datapath may impose some specific constrains on the base matrix of the QC-LDPC code, in order to ensure that no memory conflicts occur during the read/write operations from/to the memory storing the exchanged messages or the a posteriori logarithmic likelihood ratios (AP-LLR) values.

Since the focus of this thesis is on high throughput applications, most of the architectures investigated in this thesis are partly parallel layered architectures, will fully parallel processing units. Several enhancements will be proposed in Chapter 4 and Chapter 6, so that to ensure an efficient use of the hardware resources.

2.6.2.3 Cost optimizations

As mentioned before, the main components of an LDPC decoder are processing units (VNUs and CNUs), memories, and interconnects. To obtain better cost, main solutions usually focus on reducing the hardware complexity of processing units and/or the memory size requirements.

Regarding the complexity of the processing units, one of the proposed solutions consists in the use of *imprecise arithmetic operations* (*e.g.*, imprecise additions and comparisons), or more generally imprecise computational circuits [71]. An imprecise circuit is obtained by pruning the exact circuit. This amounts to removing a certain number of logic gates from the circuit (depending on the decoder's tolerance to errors), which may result in significant savings in energy, delay and area [60]. The authors in [71] evaluated the robustness of several LDPC decoding algorithms, whose VNUs and CNUs have been implemented by using imprecise adders and comparators. The arithmetic circuits used are a weighted pruned parallel prefix adder and a truncated comparator. They showed that for (3,6)-regular LDPC codes, by using imprecise arithmetic, the total logic resources reduction for processing units can achieve up to 23.7% compared to the use of exact arithmetic.

While imprecise arithmetic (*e.g.*, adders and comparators), or more generally imprecise processing of the datapath, may reduce the cost of the processing units, such an approach has little of no impact on the memory and interconnection blocks of the decoder. However, one important characteristic of LDPC decoders is that the memory and interconnection blocks dominate the overall area/delay/power performance of the hardware design. Other approaches deal with reducing the memory size requirements, by reducing the size of the exchanged messages, instead of interfering in the internal structure of the processing circuits. This also leads to a reduction of the interconnect network used to route messages from memory to the processing units.

Reducing the memory size requirements has been been addressed by several authors. For instance, the authors in [3, 13] proposed modified versions of the MS decoding, with check-node messages quantized on a very small number of bits (*e.g.*, 2-bit [13] or 3-bit [3]). To limit the degradation of the decoding performance, the

variable-nodes processing operates in a higher precision quantization domain (*e.g.*, 4-bit [13] or 7-bit [3]).

Reducing the memory size requirements is also an approach that will be further explored in this thesis. However, the approach proposed in this thesis builds upon the Finite Alphabet Iterative Decoders (FAIDs) framework introduced in [77, 78]. FAIDs are MS-based decoders, characterized by the fact that variable-node update functions are defined by specific Look-up Tables (LUTs), which can be designed with the goal of increasing the guaranteed error-correction capability by using the knowledge of potentially harmful subgraphs (*e.g.*, trapping set). We will make a different use of this framework, and actually divert FAIDs from their initial purpose: precisely, in our approach, variable-node update functions are defined by specific Look-up Tables (LUTs), which are designed with the goal of reducing the memory requirements of the decoder, without degrading the error correction performance.

2.6.2.4 Cost/power/throughput trade-offs in state of the art designs

In this section we discuss the trade-off between different figures of merit of the hardware design, as reported in state of the part implementations. We start by discussing first the relationship between the computational effort and the information throughput required for prominent applications relying on LDPC codes, which is illustrated in Figure 2.17 [84]. The diagonal lines in the double-logarithmic scale are representative for the overall complexity required by a corresponding decoder. It is interesting to see that the computational effort required per information bit remains approximately the same for all considered standards. However, the throughput requirements across standards vary by more than three orders of magnitude, whereas the throughput within a standard may also vary by one order of magnitude.

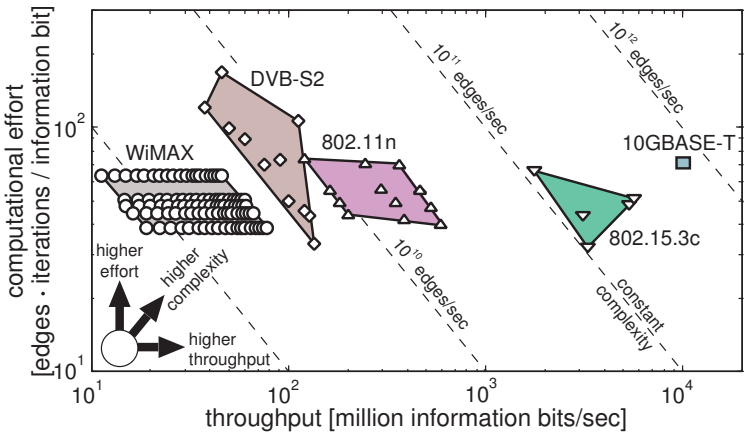


Figure 2.17: Computational effort (assuming 10 iterations) and throughput overview of several standards employing LDPC codes [84]

Table 2.3: Comparison of state-of-the-art ASIC designs for LDPC decoders

Paper	**[93]**	**[110]**	**[96]**	**[4]**	**[87]**	**[25]**	**[74]**	**[100]**	**[55]**
Parameters related to Software Implementations									
Flexibility	802.16e/.11n	No	No	820.11ad	No	No	No	802.11ad	802.11n
Code-length	2304	2304	2048	672	672	4095	672	672	1944
Coding-rate	0.5	0.5	0.84	0.5	0.81	0.82	0.50	0.81	0.5
Algorithm	BP	MS	OMS	MS	MS	MDD-BMP	OMS	–	OMS
Decoder architecture	Partly parallel	Parallel layer	Partly parallel	Partly parallel	Fully parallel	Fully parallel	Row parallel	Fully parallel	Block parallel
Decoding performance	Good	Good	–	Not good	Not good	–	Not good	Not good	Good
Parameters related to Hardware Implementations									
Quantization	–	3	4	8	4	6	5	5	5
Iterations	10	10	8	6	9	31	10	3.75	10
Technology (nm)	90	90	90	65	65	65	65	28	90
Frequency (MHz)	450	950	303	215	257	180	540	260	336
Area (mm^2)	3.5	2.9	6.14	1.1	12.09	15.37	1.6	0.63	5.2
Area scale to 28 nm	0.34	0.28	0.59	0.20	2.24	2.85	0.30	0.63	0.50
Power (mW)	410	870	1392	210	5360	5354	782.9	180	451.3
Power scale to 28 nm	39.68	84.21	134.73	38.97	994.61	993.50	145.28	180.00	43.68
Throughput (Gbps)	1.0	2.2	9.7	6.0	160.8	23.0	9.0	12.0	1.71
Throughput scale to 28 nm	3.21	7.07	31.18	13.93	373.29	53.39	20.89	12.00	5.50
EE (pJ/bit)	410.00	395.45	143.51	35.00	33.33	232.78	86.99	15.00	263.92
EE scale to 28 nm	**12.35**	**11.91**	**4.32**	**2.80**	**2.66**	**18.61**	**6.95**	**15.00**	**7.95**
NEE (pJ/bit/iter)	41.00	39.55	17.94	5.83	3.70	7.51	8.70	4.00	26.39
NEE scale to 28 nm	1.23	1.19	0.54	0.47	0.30	0.60	0.70	4.00	0.79
TAR (Gbps/mm^2)	0.29	0.76	1.58	5.45	13.30	1.50	5.63	19.05	0.33
TAR scale to 28 nm	**9.49**	**25.19**	**52.46**	**68.24**	**166.39**	**18.72**	**70.37**	**19.05**	**10.92**
NTAR (Gbps/mm^2/iter)	2.86	7.59	12.64	32.73	119.70	46.39	56.25	71.43	3.29
NTAR scale to 28 nm	94.88	251.93	419.71	409.43	1497.50	580.34	703.70	71.43	109.21

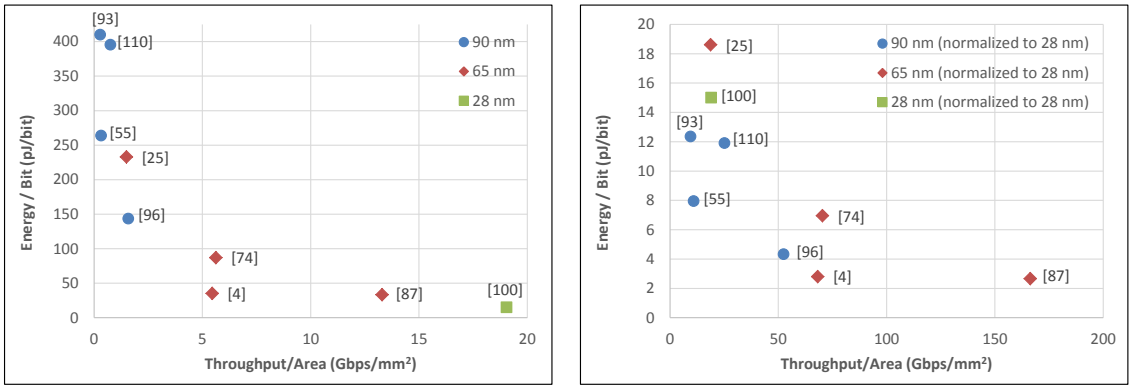
- EE (Energy Efficiency) = Energy/bit = Power/Throughput
- NEE (Normalized EE) = EE/Iterations = Energy/Bit/Iteration
- TAR (Throughput to Area Ratio) = Throughput/Area
- NTAR (Normalized TAR) = Throughput/Area/Iteration = TAR × Iterations

(N.B. Throughput is inversely proportional to the number of iterations)

- TNR (Target to Nominal Ratio) = target_technology_size / nominal_technology_size
- scaled_area = nominal_area × $(\text{TNR})^2$
- scaled_power = nominal_power × $(\text{TNR})^2$
- scaled_throughput = nominal_throughput / TNR

Concerning state-of-the-art implementations of LDPC decoders, we provide in Table 2.3 a comparison between several high-throughput ASIC designs. It is worth noting that making a fair comparison between different designs is a very difficult task. This is due to the large number of algorithmic and architectural choices that can be made, resulting in different trade-offs between error correction capacity, cost, power, throughput. These criteria cannot be optimized simultaneously. Whether a trade-off is "better" than another depends on which of the above criteria is given highest priority in the optimization process, and on how much we are willing to pay for the optimal solution, in terms of performance degradation of the lowest priority criteria. In particular, regarding the results reported in Table 2.3, it should be noticed that:

- they may be obtained using different technologies (*e.g.*, 28 nm, 45 nm, 65 nm, 90 nm), which of course has significant impact on the reported area, power, and frequency,
- they may implement different LDPC codes, with different code-lengths and coding-rates,
- they may implement different decoding algorithms, and may use different quantizations,



(a) Without technology scaling

(b) Results scaled to 28nm technology

Figure 2.18: Throughput vs. energy consumption trade-offs for state-of-the-art ASIC designs

- they may have different error correction performance, *e.g.*, due to the LDPC code itself, or to the chosen decoding algorithm or quantization,
- they may use a different number of decoding iterations; besides some authors report the worst case throughput (corresponding to the maximum number of decoding iterations), some others report average throughput (corresponding to the average number of decoding iterations for some SNR value),
- they may have different design flexibility, *e.g.*, some designs implement one single LDPC decoder, some others are compliant with all the decoders specified in one or several standards.

To keep the comparison as fair as possible, we further define a number of relative and normalized (with respect to the number of iterations) metrics, as follows:

- Energy Efficiency (EE) is the energy per decoded bit,
- Normalized Energy Efficiency (NEE) is the energy per decoded bit and per iteration
- Throughput to Area Ratio (TAR) is the throughput achieved per area unit (mm^2)
- Normalized Throughput to Area Ratio (NTAR) is the throughput achieved per area unit (mm^2), assuming that only one single decoding iteration is performed.

It is worth noticing that for all the reported implementations, the throughput is inverse proportional to the number of iterations, which justifies the normalized metrics defined above. We have computed the above metrics according to the implementation results reported for the nominal technology used for the designed, as well as after scaling all of the reported implementation results to 28 nm technology. Area, power, and throughput scaling is detailed in the footnote to Table 2.3. Parameters scaling is done as suggested in [38].

The trade-off between EE and TAR is illustrated in Figure 2.18 (both with and without technology scaling). It is worth noticing that the designs achieving the best EE / TAR trade-offs ([4], [74], [87], [100]) implement decoding of the QC-LDPC codes specified in the 802.11ad (WiGig) [43] standard. These codes rely indeed on a specific structure known to allow very high throughput implementation, which in turn is also directly responsible for their very poor error correction performance. [93], [110], and [55] designs implement decoding of the QC-LDPC codes specified in either the 802.16e (WiMAX) [46], 802.11n (WiFi) [44] standard: they have better error correction performance, but inferior performance in terms of TAR and EE.

2.7 Conclusion

This chapter provided a brief introduction to LDPC codes and iterative message-passing decoders. We focused on some specific decoders that will be used in this thesis, namely the MS, as well as several MS-based decoders. We also discussed the finite alphabet MS decoding and its density evolution analysis, which will be extensively used throughout the remaining of this thesis. Then, we discussed a number of algorithmic and architectural choices that impact both the decoding performance and the complexity of the hardware implementation. Finally, we have presented a state of the art on low-cost, high-throughput implementations, and discussed the trade-off between different figures of merit of the design, such as cost, throughput, energy efficiency and error correction performance.

Chapter 3

Code-Aware Quantizer Design for Finite-Alphabet Min-Sum Decoders

Classically, the quantization of the soft information supplied to a finite-alphabet decoder is chosen to optimize a certain criterion which does not depend on the characteristics of the existing code. This chapter investigates quantizers that are both code and decoder aware: such quantizers optimize the density evolution noise threshold of a given decoder and a given ensemble of LDPC codes. Throughout this chapter the LDPC decoder under consideration is the finite-alphabet MS decoder, and thus we shall simply refer to such quantizers as code-aware quantizers.

We propose a code-aware quantizer with lower complexity than that obtained by optimizing all decision levels and approaching its performance, for few quantization bits. We show that code-aware quantizers outperform code-independent quantizers in terms of noise threshold for both regular and irregular LDPC codes. To overcome the error floor behavior of LDPC codes, we propose the design of the quantizer for a target error probability at the decoder output. The results show that the quantizer optimized to get a zero error probability could lead to a very bad performance for practical range of signal to noise ratios.

We further propose to design jointly irregular LDPC codes and code-aware quantizers for the finite-alphabet MS decoder. We show that they achieve significant decoding gains with respect to LDPC codes designed for infinite-alphabet belief propagation decoding, but decoded by finite-alphabet MS.

Finally, we note that the proposed code-aware quantizer can easily be adapted to the MS-based decoders investigated in the next chapters of this thesis. This will allow the optimization of the code-aware quantizer jointly with the integration of approximate computing and storage techniques into the MS decoder.

3.1 Introduction

This chapter focuses on LDPC codes with Min-Sum decoder, which is more practical to implement than the sum-product decoder at the cost of a slight degradation in performance. The Min-Sum algorithm uses simple arithmetic (additions) and logical (comparisons) operations; hence it is suitable for hardware implementation. Nowadays, using few bits of precision to represent channel output and the soft information propagating during the iterative decoding process is crucial for high speed applications and for reducing hardware complexity. However, the performance of the decoder will decrease when the number of quantization bits decreases. Therefore, a good design of quantizer at the input of the decoder is necessary to achieve the best performance under a constrained low-precision hardware. Recently, many works are focusing on the field of FPGA implementation of LDPC decoders, see, *e.g.*, [5], [12] and references therein. On the other hand, most of works in literature, consider the design of the quantizer at the channel output independently of the channel code used by the communication system. A review of literature about the code-independent quantizers is presented further in this chapter.

To evaluate the asymptotic performance of a family of LDPC codes, an iterative process called density evolution (DE) is used, assuming that the Tanner graph of this family is cycle-free. Density evolution can be used to find the maximum level of channel noise, called noise threshold, which can be corrected by a family of LDPC codes using the message passing algorithm. In this work, the density evolution is used as a tool to search for the quantizer that can achieve the best noise threshold for a specific family of LDPC codes. There are few works in literature that analyze the dependency of optimal quantizers on the channel code. In [19], the authors consider the code-dependent quantizers for BI-AWGN channel when regular LDPC codes are used and evaluate the quantizer performance by density evolution. They demonstrate that quantizers that maximize the noise threshold are superior to Lloyd quantizers. Although the authors consider quantized decoder inputs, a belief-propagation decoding was considered with *infinite-alphabet* messages exchanged between variable and check nodes in density evolution. Other works investigate also quantization methods to lower the error floor for LDPC codes [1] and reduce the effects of weak absorbing sets [115].

The contributions of this work are the following. First, we present a review of the literature about the quantizers which are designed independently of the existing channel code. We propose a low complexity code-aware quantizer, whose performance approaches that of the code-aware quantizer with decision levels optimized through exhaustive search. We demonstrate that code-aware quantizers are more efficient than code-independent quantizers in terms of noise threshold for *finite-alphabet* Min-Sum decoders. We further show that finite-alphabet Min-Sum decoders exhibit an asymptotic error-floor phenomenon, which limits the performance of the LDPC code family. Therefore, the code-aware quantizer that maximizes the DE threshold (to get zero error probability at the decoder output), may not be suitable for lower bit error rate. Thus we propose the design of code-aware quantizer for a target bit error rate and we show that the quantizer optimality is highly dependent on the

target bit error rate. Finally, we propose to design jointly irregular LDPC codes and code-aware quantizers for finite-alphabet Min-Sum decoders.

3.2 System model

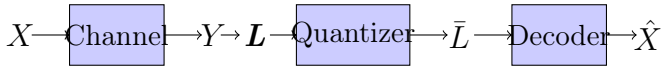


Figure 3.1: Point to point communication system with quantized-input decoder

This work deals with the point to point communication system as shown in Figure 3.1. The study could be extended to other channel models. We assume that the coded bits are modulated using a BPSK constellation, $X \in \mathcal{X} = \{+1, -1\}$, and are transmitted over a Gaussian channel with noise variance σ^2. The choice of low-order BPSK modulation is to simplify the analysis of density evolution. At the receiver side, a metric denoted by $\boldsymbol{L}$ is calculated from the channel output Y and then quantized to $\bar{L} \triangleq \varphi(\boldsymbol{L})$ using a quantizer φ[1] to be used as input for the finite-alphabet Min-Sum decoder as mentioned in Section 2.4.1. Usually, $\boldsymbol{L}$ represents the log-likelihood ratio (LLR). However, in this work, it could be not the LLR. A q-bit quantizer φ quantifies its input $\boldsymbol{L}$ on q bits. Since the performance of the quantizer will be evaluated according to its resulting noise threshold using density evolution, a symmetric-output quantizer will be considered for convenience. Thus the q-bit quantizer has $2^q - 1$ output values belonging to $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$, where $Q \triangleq 2^{q-1} - 1$. The cardinality of the set $\mathcal{M}$ is denoted by $|\mathcal{M}| = 2 \cdot Q + 1$. For convenience, the non-negative elements of the set $\mathcal{M}$ are denoted by $\bar{L}_i$ such that $\bar{L}_i = i$, $\forall i \geq 0$. In the following, the *reconstruction levels* of the q-bit quantizer are constrained to the values of $\mathcal{M}$ and only the *decision levels* can be optimized for practical purpose. Since a symmetric quantizer is used, the quantizer can be fully characterized by the non-negative decision levels denoted by T_i, $i \in \{1, \ldots, Q\}$, such that $T_i \leq T_{i+1}$. By convention $T_{Q+1} = +\infty$. The quantizer quantifies its input value $\boldsymbol{L}$ into $\varphi(\boldsymbol{L})$ such that

$$\varphi(\boldsymbol{L}) = \begin{cases} +\bar{L}_i & \text{if} \quad L \in \mathcal{I}_i \triangleq [T_i, T_{i+1}[, \\ -\bar{L}_i & \text{if} \quad L \in \mathcal{I}_{-i} \triangleq]-T_{i+1}, -T_i], \\ +\bar{L}_0 & \text{if} \quad L \in \mathcal{I}_0 \triangleq]-T_1, +T_1[\end{cases} \tag{3.1}$$

where $\bar{L}_i = i \in \{1, \ldots, Q\}$. Note that if $T_i = T_{i+1}$ the values $\bar{L}_i$ and $-\bar{L}_i$ are never taken by the quantizer $\forall i \geq 1$. Besides, if $T_1 = 0$, then the value $\bar{L}_0$ is never taken by the quantizer.

Equation (3.1) gives the general definition of a quantizer and in the next section we will present several code-independent or code-aware optimization criteria. By a slight abuse of language, we shall use the same terminology to refer to both the optimization criterion and the resulting quantizer.

[1]By abuse of notation, the quantizer is denoted by its associated *quantization map* φ.

3.3 Code-Independent Quantizers

This section reviews some of the well known quantizers which do not depend on the existing channel code.

3.3.1 $\mathrm{MI}_{X\bar{L}}$: the quantizer which maximizes $I(X;\bar{L})$

The design of quantizers based on information-theoretic measures has been widely considered in literature. Quantizers which maximize the mutual information between the channel input and the quantizer output have been of special interest. This is because the channel capacity is obtained by maximizing the mutual information between the channel input and its output. Nowadays, there exits powerful codes which approach the channel capacity of point to point Gaussian channel as LDPC codes. The expression of the mutual information $I(X;\bar{L})$, where $\bar{L} \in \mathcal{M}$, is given by

$$I(X;\bar{L}) = \sum_{x\in\mathcal{X}} p_X(x) \sum_{\bar{\ell}\in\mathcal{M}} p_{\bar{L}|X}(\bar{\ell}|x) \cdot \log \frac{p_{\bar{L}|X}(\bar{\ell}|x)}{\sum_x p_X(x) \cdot p_{\bar{L}|X}(\bar{\ell}|x)} \tag{3.2}$$

The input $\boldsymbol{L}$ of the $\mathrm{MI}_{X\bar{L}}$ quantizer is the LLR value. For the Gaussian channel with noise variance σ^2, the LLR value $\boldsymbol{L}$ is related to the channel output Y as $\boldsymbol{L} = \mu \cdot Y$ where $\mu \triangleq \frac{2}{\sigma^2}$. It is easy to demonstrate that in this case the conditional pdf of $\boldsymbol{L}$, given that $X = x$ was sent, follows a normal distribution of mean $\mu_L = \mu \cdot x$ and of variance $\sigma_L^2 = 2\mu$ [79]. Thus, the expression of $p_{\bar{L}|X}(\bar{\ell}|x)$, for $\bar{\ell} \in \mathcal{M}$, is given by:

$$p_{\bar{L}|X}(\bar{\ell} = i|X = x) = \int_{\ell\in\mathcal{I}_i} p_{L|X}(\ell|x)d\ell \tag{3.3}$$

where $i \in \mathcal{M}$. Using the definition of Q-function defined by $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{+\infty} \exp^{-\frac{u^2}{2}} du$, (3.3) can be written as

$$\begin{cases} p_{\bar{L}|X}(\bar{\ell} = i|X = x) = Q\Big(\frac{T_i-\mu_L}{\sigma_L}\Big) - Q\Big(\frac{T_{i+1}-\mu_L}{\sigma_L}\Big) \\ p_{\bar{L}|X}(\bar{\ell} = -i|X = x) = Q\Big(\frac{-T_{i+1}-\mu_L}{\sigma_L}\Big) - Q\Big(\frac{-T_i-\mu_L}{\sigma_L}\Big) \\ p_{\bar{L}|X}(\bar{\ell} = 0|X = x) = Q\Big(\frac{-T_1-\mu_L}{\sigma_L}\Big) - Q\Big(\frac{T_1-\mu_L}{\sigma_L}\Big) \end{cases} \tag{3.4}$$

where $i \in \{1,\ldots,Q\}$. For BPSK modulation, the maximal mutual information is achieved using equally probable input *i.e.*, $p_X(x = +1) = p_X(x = -1) = \frac{1}{2}$ [91]. In order to design the quantizer which maximize the mutual information between X and $\bar{L}$, the following optimization problem should be solved:

$$\max_{T_i, i=1,\ldots,Q} I(X;\bar{L}) \tag{3.5}$$

where $I(X;\bar{L})$ depends on T_i through (3.4). The problem (3.5) appears difficult to solve even with a low order modulation (BPSK). In [63], the authors consider the case of the binary input and ternary output channel which is simple to solve since the maximization in (3.5) is done over one parameter. When the number of output values

increase, the optimization becomes difficult to perform. In many works such as in [91] and [61], local optimization algorithm were used to solve the multi-variable case of the problem. Reference [57] solves this problem using a greedy search algorithm for any finite-input discrete memoryless channels. In [56], the authors propose an algorithm which finds the global optimal quantizer which maximizes $I(X;\bar{L})$ for arbitrary binary input channels and for an arbitrary number of decision levels. The algorithm has a cubic complexity in the cardinality of channel output.

3.3.2 $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$: the quantizer which maximizes $I(\boldsymbol{L};\bar{L})$

Since the design of quantizer which maximizes the mutual information between channel input and quantizer output has high complexity, especially when the input alphabet size increases, an alternative method used in literature is to maximize the mutual information between the quantizer input and its output [72]. The optimal quantizer in this case is obtained by solving the following optimization problem

$$\max_{T_i, i=1,\ldots,Q} I(\boldsymbol{L};\bar{L}) \tag{3.6}$$

where $\boldsymbol{L}$ refers also the LLR in the case of $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$ quantizer. The mutual information $I(\boldsymbol{L};\bar{L})$ can be written as $I(\boldsymbol{L};\bar{L}) = H(\bar{L}) - H(\bar{L}|\boldsymbol{L})$ where H denotes the entropy. Since $\bar{L}$ is a deterministic function of $\boldsymbol{L}$, $H(\bar{L}|\boldsymbol{L}) = 0$. Consequently, the optimal quantizer should maximizes the entropy of $\bar{L}$. It is well known that the entropy of a discrete variable is maximized with its uniform distribution. Using (3.4), the solution of problem (3.6) for the binary input Gaussian channel is such that

$$\begin{aligned} p_{\bar{L}}(\bar{L}_i) = \frac{1}{2}\Big[Q\Big(\frac{T_i-\mu}{\sigma_L}\Big) - Q\Big(\frac{T_{i+1}-\mu}{\sigma_L}\Big)\Big] \\ + \frac{1}{2}\Big[Q\Big(\frac{T_i+\mu}{\sigma_L}\Big) - Q\Big(\frac{T_{i+1}+\mu}{\sigma_L}\Big)\Big] = \frac{1}{|\mathcal{M}|} \end{aligned} \tag{3.7}$$

for $i = 1, \ldots, Q$. These equations are solved starting from step $i = Q$, where there is only one variable to solve in equation (3.7) since $T_{Q+1} = +\infty$. Then the problem is solved going from $i = Q$ to $i = 1$. For any step $i < Q$, T_{i+1} is calculated from the previous step $i+1$ thus we have only to determine T_i which solves (3.7) given T_{i+1}. In other terms at step i, we have to solve the following equation of one variable T_i

$$f(T_i) \triangleq Q\Big(\frac{T_i-\mu}{\sigma_L}\Big) + Q\Big(\frac{T_i+\mu}{\sigma_L}\Big) - \Big[A_1 + A_2\Big] - \frac{2}{|\mathcal{M}|} = 0 \tag{3.8}$$

where $A_1 = Q\Big(\frac{T_{i+1}-\mu}{\sigma_L}\Big)$ and $A_2 = Q\Big(\frac{T_{i+1}+\mu}{\sigma_L}\Big)$ are constants since T_{i+1} is known at step i.

Proposition 3.1 *The equation (3.8) has a unique solution which can be obtained using the bisection method.*

Proof. First, define the function $u(T_i) = Q\Big(\frac{T_i-\mu}{\sigma_L}\Big) + Q\Big(\frac{T_i+\mu}{\sigma_L}\Big)$. We can easily demonstrate using the derivative of $u(T_i)$ with respect to T_i that u is a strictly monotone

function. For $i = Q$, solving (3.8) is equivalent to solve $u(T_Q) - u(T_{Q+1}) - \frac{2}{|\mathcal{M}|} = 0$ where $u(T_{Q+1}) = 0$ using the definition of the Q-function and the fact that $T_{Q+1} = +\infty$. We can demonstrate recursively that at each step i, the equation to solve is $u(T_i) = \frac{2(Q-i+1)}{|\mathcal{M}|}$ where $0 < \frac{2(Q-i+1)}{|\mathcal{M}|} < 2$. Since the Q-function values are bounded between 0 and 1, $u(T_i)$ is bounded between 0 (achieved for $T_i = +\infty$) and 2 (achieved for $T_i = -\infty$). Thus $u(T_i) = \frac{2(Q-i+1)}{|\mathcal{M}|}$ has always a solution which can be obtained efficiently by the bisection method. This proof applies for arbitrary input alphabet size. □

3.3.3 Others

There exists other criteria in literature to design the quantizer at the decoder input. The well known Lloyd algorithm allows to obtain the optimal reconstruction and decision levels for the quantizer which minimizes the mean square reconstruction error called also distortion [62]. Other quantizers considered in literature maximizes the cutoff-rate since it has simpler expression than the mutual information as a function of the transition probabilities [61]. Some works considered also the quantizers that maximize random coding exponents since this metric is more suitable for finite length codes than mutual information [106].

All the quantizers presented above maximize certain criteria that do not depend on the characteristics of the existing LDPC code as its degree distribution, etc. The next section is dedicated to code-aware quantizer design.

3.4 Code-aware quantizers

3.4.1 Decision levels quantizer (DL)

This section investigates code-aware quantizer design for BI-AWGN channel when LDPC codes are used along with Min-Sum decoding. Our study differs from [19] by the use of finite-alphabet LDPC decoders where the messages exchanged using density evolution are also quantized on a fixed number of bits as well as the decoder input. Moreover, our study includes both regular and irregular LDPC codes. We refer the reader to Chapter 2 for the density evolution equations for finite-alphabet Min-Sum decoding. The density evolution equations allow to obtain the error probability $P_e^{(\ell)}$ at each iteration ℓ of the iterative message-passing algorithm. The criterion used to optimize the quantizer is the noise threshold σ_{th}^2 defined by

$$\sigma_{\text{th}}^2 = \sup\{\sigma^2 \ : \ \lim_{\ell\to\infty} P_e^{(\ell)} = 0\} \tag{3.9}$$

The finite-alphabet Min-Sum decoder is known to have high error floors. Thus a quantizer φ_1 which is better than another quantizer φ_2 for a target bit error rate may be worse than φ_2 for a different target bit error rate. Moreover, in practice the target bit error rate is usually fixed to a practical value (*e.g.*, 10^{-5}) because a very low target error probability requires higher SNR in general. Hence, we investigate the optimal quantization which maximizes the noise threshold for zero error probability using density evolution as well as the optimal quantization which maximizes the noise threshold for a target bit error rate. The η-threshold for a target bit error rate η is defined by

$$\sigma_{\text{th}}^2(\eta) = \sup\{\sigma^2 \ : \ \lim_{\ell\to\infty} P_e^{(\ell)} \leq \eta\} \tag{3.10}$$

When $\eta = 0$, the $\eta-$threshold will be called "DE threshold" and is equivalent to the threshold defined in (3.9). The code-aware quantizer design for a target bit error rate η requires to solve the following optimization problem

$$\max_{T_i, i=1,\ldots,Q} \sigma_{\text{th}}^2(T_1, \ldots, T_Q; \eta) \tag{3.11}$$

This problem is solved using exhaustive search method to avoid local solutions.

3.4.2 Gain factor quantizer (GF)

Since the complexity of exhaustive search methods increases with the number of quantization bits, we provide a code-aware quantizer design with lower complexity based on a uni-parametric optimization. The idea is to relax the constraint on the value of $\mu \triangleq \frac{2}{\sigma^2}$ in the expression of the LLR given by $\boldsymbol{L} = \mu \cdot Y$ and optimize μ. This case uses the symmetric quantizer such that the non-negative decision levels are given by $T_i = \frac{\bar{L}_{i-1}+\bar{L}_i}{2}$ for $i = 1, \ldots, Q$, however the channel output Y is scaled by the *gain factor* μ. In other terms, $\boldsymbol{L}$ is quantized to the nearest integer in $\mathcal{M}$. To obtain the optimal gain factor for a target bit error rate η we should solve the following optimization problem

$$\max_{\mu} \quad \sigma_{\text{th}}^2(\mu; \eta) \tag{3.12}$$

Problem (3.12) is a uni-dimensional optimization problem and is solved using exhaustive search method.

3.4.3 Summary and remarks

The quantizers under study and their parameters are summarized in Table 3.1. The quantizers $\mathrm{MI}_{X\bar{L}}$, $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$ are code-independent quantizers where the LLR ($\boldsymbol{L} = \frac{2}{\sigma^2} \cdot Y$) is used as the input of the decoder and the quantizer decision levels are those maximizing a mutual information. DL and GF are code-aware quantizers. For DL quantizer, the LLR is used as decoder input while for GF, $\boldsymbol{L} = \mu \cdot Y$ is used as an input of the decoder where g is optimized.

Table 3.1: Parameters of the quantizers under study

Quantizer	μ	Decision levels
$\mathrm{MI}_{X\bar{L}}$	$\frac{2}{\sigma^2}$	optimized
$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	$\frac{2}{\sigma^2}$	optimized
DL	$\frac{2}{\sigma^2}$	optimized
GF	optimized	$T_i = \frac{\bar{L}_{i-1}+\bar{L}_i}{2}$, $i = 1, \ldots, Q$

Finally some remarks on the parameters in Table 3.1 are presented in the following.

- For the quantizer $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$, the gain factor μ will not affect the performance since the value of the mutual information $I(\boldsymbol{L}; \bar{L})$ is independent of μ, where $\boldsymbol{L} = \mu \cdot Y$. However, it can be proven easily using (3.4) and the expression of the mutual information, that the values of the optimal decision levels when $\boldsymbol{L} = \mu \cdot Y$ are equal to the optimal decision levels when $\boldsymbol{L} = Y$ multiplied by a factor μ. Hence, we choose the conventional LLR as input for the decoder in this case where $\mu = \frac{2}{\sigma^2}$. Similarly, for the cases of $\mathrm{MI}_{X\bar{L}}$ and DL quantizers, where the LLR is used as decoder's input.

- For GF quantizer, we optimize μ such that $\boldsymbol{L} = \mu \cdot Y$ but the decision levels are fixed to be $T_i = \frac{\bar{L}_{i-1}+\bar{L}_i}{2}$ for $i = 1, \ldots, Q$. The GF quantizer gives the same performance of the quantizer with $\boldsymbol{L} = Y$ and $T_i = \frac{1}{\mu} \cdot \frac{\bar{L}_{i-1}+\bar{L}_i}{2}$ for $i = 1, \ldots, Q$ or the quantizer with $\boldsymbol{L} = \frac{2}{\sigma^2} \cdot Y$ and $T_i = \frac{2}{\mu\sigma^2} \cdot \frac{\bar{L}_{i-1}+\bar{L}_i}{2}$ for $i = 1, \ldots, Q$.

3.5 Performance evaluation

Throughout this section, we assume that the a posteriori information of the finite-alphabet Min-Sum decoder is quantized on $\tilde{q} = q + 1$ bits, where q is the number of quantization bits for the a priori information and exchanged messages of the finite-alphabet Min-Sum decoder.

3.5.1 (Semi-) Regular LDPC codes

Table 3.2: DE threshold of some independent-code quantizers and code-aware quantizers for the family of (semi-)regular LDPC codes of rate r and variable node degree $d_v = 3$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on $q = 2$ bits and the a posteriori information is quantized on $\tilde{q} = 3$ bits.

r	GF	DL	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
1/3	0.8453	0.8453	0.8403	0.7713	1.1969
1/2	0.5422	0.5422	0.5403	0.4248	1.5352
2/3	0.3527	0.3527	0.3526	-	1.8151
3/4	0.2810	0.2810	0.2809	-	1.9426
5/6	0.2176	0.2176	0.2167	-	2.0730

Table 3.3: DE threshold of some independent-code quantizers and code-aware quantizers for the family of (semi-)regular LDPC codes of rate r and variable node degree $d_v = 3$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on $q = 3$ bits and the a posteriori information is quantized on $\tilde{q} = 4$ bits.

r	GF	DL	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
1/3	1.0234	1.0234	1.0116	0.9898	2.2592
1/2	0.6625	0.6625	0.6591	0.6266	2.7726
2/3	0.4399	0.4399	0.4389	0.3934	3.1871
3/4	0.3554	0.3554	0.3544	0.2885	3.4510
5/6	0.2799	0.2799	0.2785	-	3.8480

Table 3.2 and Table 3.3 show the noise thresholds (σ^2_{th}) for some LDPC codes of rate r and variable node degree d_v when the a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized both on $q = 2$ and 3 bits respectively. Since we have fixed r and d_v, the check nodes can be all of the

same degree (regular code) or not (semi-regular code). The thresholds are given in Table 3.2 and Table 3.3 for $\eta = 0$ (zero error rate on the decoder output). The precision on the threshold value is fixed to 10^{-4}. The results show that the GF quantizer gives the same performance of the DL quantizer of high computational complexity. The quantizer maximizing the mutual information between the channel input and the quantizer output ($\mathrm{MI}_{X\bar{L}}$) gives good performance which coincides with the performance of DL for some codes. The quantizer $\mathrm{MI}_{X\bar{L}}$ gives better performance than the quantizer maximizing the mutual information between the input of the quantizer and its output $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$ for regular codes. It can be observed that the quantizer $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$ does not achieve a target zero error probability when the code rate is higher than a certain value (*e.g.*, when $r \in \{\frac{2}{3}, \frac{3}{4}, \frac{5}{6}\}$ in Table 3.2).

3.5.2 Irregular LDPC codes

In this section, we study the performance of irregular LDPC codes in terms of noise threshold using code-aware and code-independent quantizers. Consider the family of irregular LDPC codes of rate one-half, variable node degree distribution $\lambda(x) = 0.23882x + 0.29515x^2 + 0.03261x^3 + 0.43342x^{10}$ and check node degree distribution $\rho(x) = 0.43011x^6 + 0.56989x^7$. This code was shown to be powerful for belief-propagation decoding and has a noise DE threshold of $\sigma^2_{\mathrm{th}} = 0.9162$ [47]. However, for the fixed point Min-Sum decoder, the performance of this code seems to be worse than the regular code of rate one half in the previous section, especially when the number of quantization bits is small as shown in Table 3.4, due to the error floor.

The DL quantizer is not considered when $q > 3$ due to the complexity of the exhaustive search in this case. Tables 3.5, 3.6 and 3.7 show the $\eta-$threshold for the same irregular code when the number of quantization bits is $q = 2$, 3 and 4 respectively and $\eta \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-10}\}$. We observe also that the quantizer $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$ is better than $\mathrm{MI}_{X\bar{L}}$ in most cases contrarily to the case of regular codes in the previous section. In general, we can see that any of the two quantizers ($\mathrm{MI}_{X\bar{L}}$ and $\mathrm{MI}_{\boldsymbol{L}\bar{L}}$) can be better than the other depending on the values of η and q because these quantizers are not code-aware.

The code aware quantizer GF can achieve noticeable gains comparing to the best quantizer among the independent-code quantizers depending on η and q. For $q = 2$, GF quantizer has the same performance as the DL quantizer. This may be obvious because there is one variable to optimize in both cases. For $q = 3$, the performance of GF quantizer is very close to that of DL quantizer. The decision levels obtained for both code-aware quantizers are very close as shown in Figure 3.2 when $\eta = 10^{-5}$. It is worth to note that in Figure 3.2, the decisions levels are given when $L = \frac{2}{\sigma^2_{\mathrm{th}}(DL)} \cdot Y$, where $\sigma^2_{\mathrm{th}}(DL)$ is the $\eta-$threshold for DL quantizer with $\eta = 10^{-5}$, for both quantizers in order to compare them for the same $\boldsymbol{L}$. Then, the decision levels for GF quantizer are given by $T_i = \frac{2}{\mu \cdot \sigma^2_{\mathrm{th}}(DL)} \cdot \frac{\bar{L}_{i-1} + \bar{L}_i}{2}$ for $i = 1, \ldots, Q$ (cf. Section 3.4.3) where μ is the optimal gain factor for $\eta = 10^{-5}$ which is given in Table 3.6.

Table 3.4: DE threshold of some independent-code quantizers and code-aware quantizers for the family of irregular LDPC codes of rate $\frac{1}{2}$, variable node degree distribution $\lambda(x) = 0.23882x + 0.29515x^2 + 0.03261x^3 + 0.43342x^{10}$ and check node degree distribution $\rho(x) = 0.43011x^6 + 0.56989x^7$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on q bits and the a posteriori information is quantized on $\tilde{q} = q + 1$ bits.

q	GF	DL	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
2	0.0560	0.0570	0.0156	0.0525	0.5092
3	0.1179	0.1179	0.0359	0.0557	0.5135
4	0.2932	Not given	0.0759	0.0570	0.5322

Table 3.5: η-threshold of some independent-code quantizers and code-aware quantizers for the family of irregular LDPC codes of rate $\frac{1}{2}$, variable node degree distribution $\lambda(x) = 0.23882x + 0.29515x^2 + 0.03261x^3 + 0.43342x^{10}$ and check node degree distribution $\rho(x) = 0.43011x^6 + 0.56989x^7$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on $q = 2$ bits and the a posteriori information is quantized on $\tilde{q} = q + 1$ bits.

η	GF	DL	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
10^{-3}	0.5362	0.5362	0.4330	0.5107	0.7952
10^{-4}	0.4536	0.4536	0.2825	0.4351	0.6146
10^{-5}	0.3760	0.3760	0.1947	0.3490	0.5592
10^{-10}	0.1868	0.1868	0.0667	0.1704	0.5179

Table 3.6: η-threshold of some independent-code quantizers and code-aware quantizers for the family of irregular LDPC codes of rate $\frac{1}{2}$, variable node degree distribution $\lambda(x) = 0.23882x + 0.29515x^2 + 0.03261x^3 + 0.43342x^{10}$ and check node degree distribution $\rho(x) = 0.43011x^6 + 0.56989x^7$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on $q = 3$ bits and the a posteriori information is quantized on $\tilde{q} = q + 1$ bits.

η	GF	DL	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
10^{-3}	0.6733	0.6798	0.6668	0.6620	2.3247
10^{-4}	0.6600	0.6650	0.4792	0.6590	1.5193
10^{-5}	0.6350	0.6384	0.3373	0.5432	1.3183
10^{-10}	0.4869	0.4870	0.1182	0.2716	0.6536

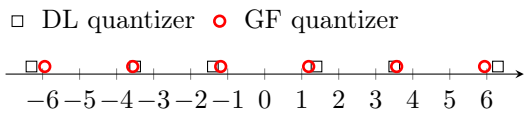


Figure 3.2: Decision levels of DL and GF quantizers obtained when $\eta = 10^{-5}$, $q = 3$ and using the irregular LDPC code of rate one-half in Section 3.5.2

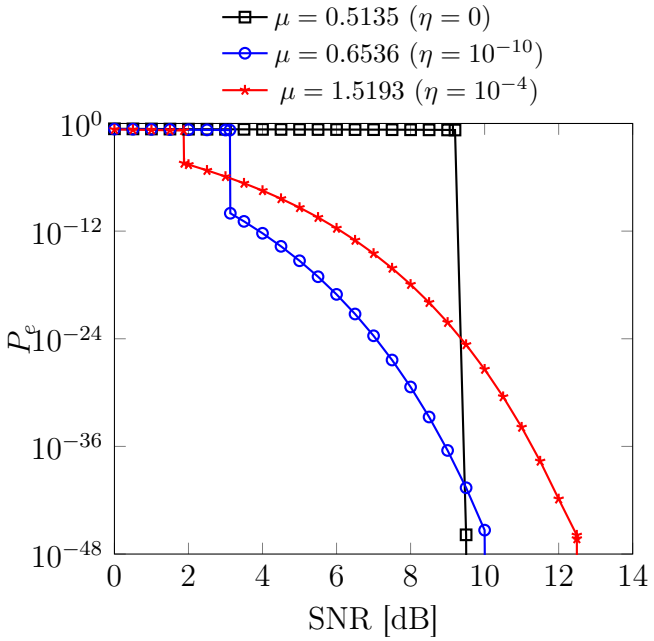


Figure 3.3: Error probability P_e obtained via DE using the GF quantizer with parameter μ as a function of the channel SNR when $q = 3$ bits. The value of η for which the quantizer is optimal is given between parentheses.

Figure 3.3 shows the asymptotic error probability (*i.e.*, $P_e = \lim_{\ell \to \infty} P_e^{(\ell)}$) at the decoder output obtained via density evolution as a function of the channel SNR for the GF quantizer and for different values of the gain factor μ. It can be observed that the optimal value of μ which maximizes the noise threshold (equivalently minimizes the SNR threshold) for a zero target bit error rate has a bad performance for the SNR values smaller than 9 dB. This is due to the error floor in the curve of error probability when the slope changes from a certain value of SNR. Thus, it is necessary to optimize the quantizer for a target error probability to avoid having an error floor behavior in the curve or bad performance at this target value.

3.5.3 Finite length performance of GF quantizer

In this section, we study the performance of GF quantizer for finite length irregular LDPC codes, with node-degree distribution polynomials λ and ρ from Section 3.5.2. Figures 3.4 and 3.5 show the bit error rate (BER) curves, with $q = 4$, for infinite length code based on density evolution and for finite length codes with a codeword length N, when $\eta = 10^{-4}$, and $\eta = 10^{-5}$ respectively. Two methods for constructing finite length codes are under consideration. The first method is when the code is constructed randomly (RAND). The second is when the code is constructed using the "progressive edge growth" (PEG) algorithm [41] in order to avoid undesirables graph topologies (such as short cycles, small trapping sets, etc.). The corresponding optimal μ value for each η can be obtained from Table 3.7.

Table 3.7: η-threshold of some independent-code quantizers and GF quantizer for the family of irregular LDPC codes of rate $\frac{1}{2}$, variable node degree distribution $\lambda(x) = 0.23882x + 0.29515x^2 + 0.03261x^3 + 0.43342x^{10}$ and check node degree distribution $\rho(x) = 0.43011x^6 + 0.56989x^7$. The a priori information and the exchanged messages of the finite-alphabet Min-Sum decoder are quantized on $q = 4$ bits and the a posteriori information is quantized on $\tilde{q} = q + 1$ bits.

η	GF	$\mathrm{MI}_{X\bar{L}}$	$\mathrm{MI}_{\boldsymbol{L}\bar{L}}$	Optimal μ for GF
10^{-3}	0.7074	0.6962	0.7063	3.9905
10^{-4}	0.7068	0.5600	0.7063	3.8010
10^{-5}	0.6999	0.4010	0.6095	2.9582
10^{-10}	0.6457	0.1225	0.3170	1.4497

For the infinite length code, one can see that the BER curve is subject to a fall at the $\eta-$threshold SNR for $\eta = 10^{-4}$ and $\eta = 10^{-5}$ in Figures 3.4 and 3.5 respectively. Then the curves experience an error floor starting from a value of BER equal (in general less or equal) to the target value η for which the GF quantizer is optimized. Consequently, one can conclude that optimizing the quantizer is crucial to avoid error floor at the target BER. To the best of our knowledge, this is the first time where it is shown that the error floor is not due solely to specific topologies in the "finite-length graph", but also to an intrinsic asymptotic behavior that occurs even for infinite length codes. For the finite length code, the curves exhibit a similar behavior; however, the curve for the finite length code presents a smooth fall rather than an infinite-slope fall as in the asymptotic case.

We observe that the error probability of the finite length code, which is constructed randomly, is lower bounded by the error probability obtained via density evolution during the error floor part and the curves are not exactly matching. This is due to the topology of the finite length code, which makes its performance affected by other factors (*e.g.*, trapping sets [80], absorbing sets [114], etc.). Moreover, we have observed that the minimum distance of the randomly constructed code is

very small, such that in the error floor region the decoder converges quite often to a wrong codeword (different from the one actually sent over the channel). Both codes constructed by the PEG algorithm ($N = 4000$ and $N = 20000$) have girth equal to 8 and good minimum distance properties. For these codes, in all the simulations that we ran, the decoder never converged to a wrong codeword. Surprisingly, the error probability of the PEG constructed codes in the error floor region is smaller than the one predicted by density evolution.

To explain this, we have investigated the edge-perspective joint degree distribution of the constructed codes. If (λ, ρ) is the degree distribution pair of the irregular LDPC code ensemble, an implicit assumption made in the classical density evolution is that the fraction of edges connected to a variable-node of degree i and check-nodes of degree j is given by $f_{ij} = \lambda_i \cdot \rho_j$. However, for the PEG constructed codes, we observed a mismatch between the constructed f_{ij} distribution and the theoretical one (even if the constructed λ and ρ distributions are very close to the theoretical ones). To overcome this issue, future works should consider the density evolution for multi-edge type LDPC [81].

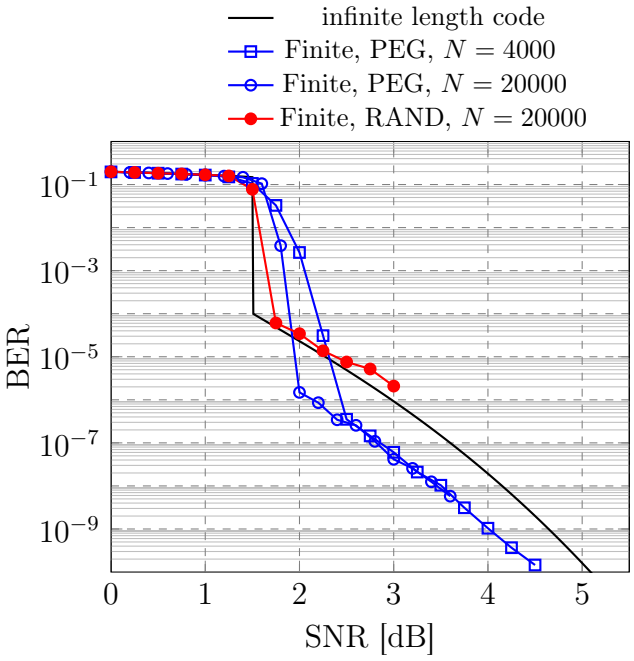


Figure 3.4: BER curves for the GF quantizer with finite and infinite length codes when $q = 4$, $\eta = 10^{-4}$ ($\mu = 3.8010$) and using the irregular LDPC code of rate one-half in Section 3.5.2

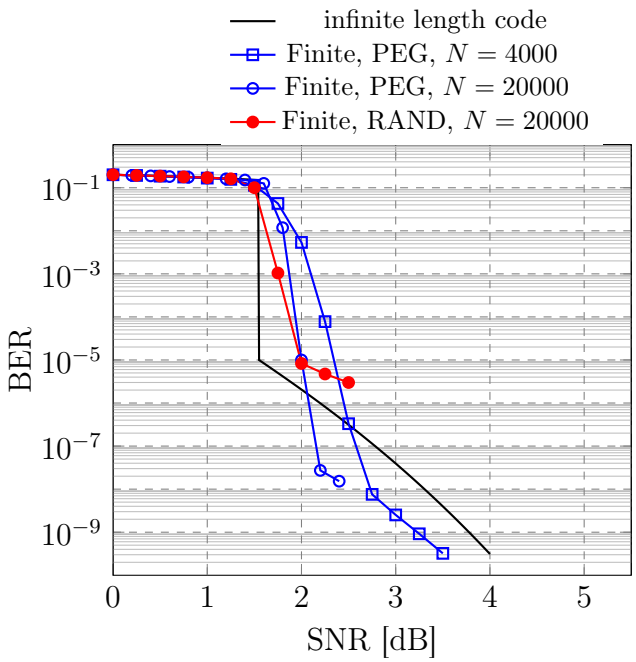


Figure 3.5: BER curves for the GF quantizer with finite and infinite length codes when $q = 4$, $\eta = 10^{-5}$ ($\mu = 2.9582$) and using the irregular LDPC code of rate one-half in Section 3.5.2

3.5.4 Irregular LDPC code design for finite-alphabet Min-Sum decoder

It was shown in Section 3.5.2 that irregular codes designed for infinite-alphabet decoders based on belief propagation decoding may lead to very bad thresholds when used for finite-alphabet Min-Sum decoders (see Table 3.4). In this section, we propose to design irregular LDPC codes for finite-alphabet Min-Sum decoders operating with a certain (small) number of quantization bits q (for the a priori information and exchanged messages) and $\tilde{q}$ (for the a posteriori information). The design of a code for a fixed rate, consists in finding the optimal degree distribution pairs, usually for some fixed maximum variable node and check node degrees (d_v and d_c) to simplify the optimization.

In our simulations, we have chosen to fix the maximum variable node degree $d_v = 11$. Ideally, we should consider a variable-node degree distribution of the form $\lambda(x) = \sum_{i=2}^{11} \lambda_i x^{i-1}$ and optimize all λ_i. However, this will make the optimization very complex due to the large number of variables. Thus, we have considered a variable-node degree distribution of the form $\lambda(x) = \lambda_2 x + \lambda_3 x^2 + \lambda_4 x^3 + \lambda_{11} x^{10}$ which has the same form of the irregular code in Section 3.5.2. The code rate r is fixed to one half. The check-node degree distribution ρ is chosen to be semi-regular, according to the value of λ and the coding rate r. We consider a GF quantizer, with a gain factor g optimized jointly with the degree distribution pair to maximize the η-threshold for a target error probability equal to η.

Table 3.8: Good degree distribution pairs of rate one-half with variable node degrees fixed to 2, 3, 4 and 11 for $q = 2$, 3 and 4, when $\eta = 10^{-10}$. For each degree distribution pair, the η-threshold value in terms of noise variance σ^2_{th} and the corresponding SNR in decibels are given, as well as the optimized gain factor μ for the GF quantizer.

q	2	3	4
λ_2	0	0.000276893	0.24805492
λ_3	0.95587734	0.876706068	0.49037245
λ_4	0.00015006	0.000000024	0.00000559
λ_{11}	0.04397260	0.123017015	0.26156704
ρ_6	0.775877	0.37470064	0.53656845
ρ_7	0.224123	0.62529936	0.46343155
μ	1.51	2.776	1.4
σ^2_{th}	0.5423889	0.6660461	0.717987
SNR_{th} [dB]	2.656892	1.764957	1.438834

The optimization problem under consideration, for fixed q and r, is the following:

$$\max_{\lambda(x),\mu} \quad \sigma^2_{\text{th}}(\lambda(x), \rho(x), \mu; \eta) \tag{3.13}$$

$$\text{subject to:} \sum_i \lambda_i = 1 \tag{3.14}$$

$$0 \leq \lambda_i \leq 1 \quad \forall i \tag{3.15}$$

$$\mu > 0 \tag{3.16}$$

where ρ is the unique (semi-)regular polynomial such that

$$\int_0^1 \rho(x)dx = (1-r) \cdot \int_0^1 \lambda(x)dx \tag{3.17}$$

Optimization techniques to solve such problems were described in [47]. In our simulations, we have used "differential evolution" method [92] to solve this problem. Table 3.8 gives some good codes found by solving problem (3.16) for $q = 2, 3, 4$ and $\eta = 10^{-10}$. Figure 3.6 shows the error probability as a function of the channel SNR using the optimized code for belief propagation decoding in Section 3.5.2 and the optimized code for the finite-alphabet Min-Sum decoding in Table 3.8 for each $q \in \{2, 3, 4\}$ and for target error probability $\eta = 10^{-10}$. We observe that noticeable gains of 4.63 dB, 1.36 dB, and 0.46 dB in SNR thresholds are obtained with respect to optimized code for belief propagation decoding, when $q = 2$, 3 and 4 respectively. Thus, it is crucial to consider the "finite-alphabet" property of the decoder to design irregular LDPC codes for finite-alphabet Min-Sum decoders.

We observe in Table 3.8 that when $q = 2$, the optimized code is very similar to a semi-regular code with $r = \frac{1}{2}$ and $d_v = 3$, since $\lambda_3 \approx 1$. The η-threshold obtained for $q = 2$ ($\sigma^2_{\text{th}} = 0.5423889$) is also close to that of the (semi-) regular code of rate one-half with $d_v = 3$ given in Table 3.2. In Table 3.8, we observe that the optimized codes exhibit higher irregularity with increasing q. Thus, we conclude that (semi-) regular LDPC codes achieve good thresholds for finite-alphabet decoders with a very few number of quantization bits.

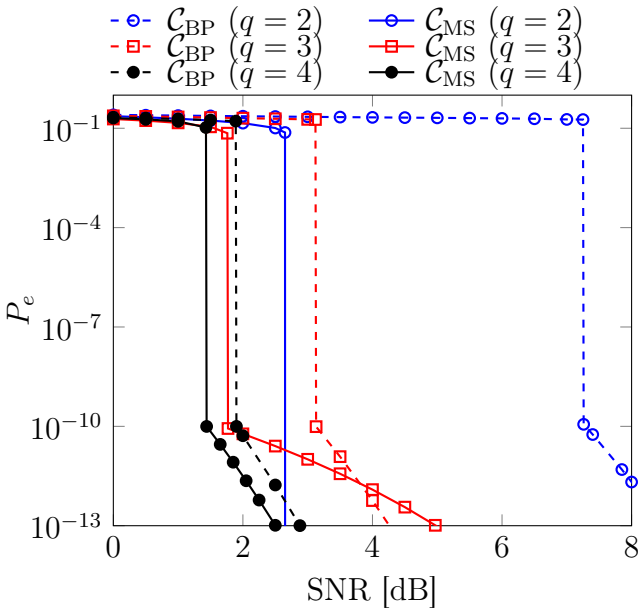


Figure 3.6: Error probability P_e obtained via density evolution using the GF quantizer as a function of the channel SNR for $\eta = 10^{-10}$ and $q \in \{2, 3, 4\}$. For each q, the error probability is plotted for $\mathcal{C}_{\text{MS}}$, the optimized code for the finite-alphabet min sum decoder given in Table 3.8 and $\mathcal{C}_{\text{BP}}$, the optimized code for belief propagation decoding given in Section 3.5.2

3.6 Conclusion

In this chapter, we investigated the design of code-aware quantizers for finite-alphabet Min-Sum decoders which maximize the noise threshold of the existing family of LDPC code. We have presented some non-aware-code quantizers existing in literature and shown that code-aware quantizers outperform them in terms of noise threshold. We also proposed a low complexity code-aware quantizer. Besides the quantizer itself, one of the main contributions of the work was to show that its performance is close to that of the quantizer whose all decision levels are optimized exhaustively. Moreover, we proposed to design the code-aware quantizer for a target BER and it was shown that this can prevent error floor phenomenon at the target BER. Finally, we proposed the joint design of good irregular LDPC code and code-aware quantizer for a Min-Sum decoder with finite-alphabet resulting into important gain in threshold with respect to irregular LDPC codes designed for belief propagation decoders with infinite-alphabet. This work can be considered as a benchmark for practical implementation of LDPC codes on a finite-alphabet hardware.

Chapter 4

Design of High Throughput LDPC Decoder based on Imprecise Offset Min-Sum Decoding

This chapter proposes two new decoding algorithms and hardware implementations, obtained by introducing two levels of impreciseness in the Offset MS (OMS) decoding: the Partially OMS (POMS), which performs only partially the offset correction, and the Imprecise Partially OMS (I-POMS), which introduces a further level of impreciseness in the check-node processing unit. We show that they allow significant reduction in the memory (25% with respect to the conventional OMS) and interconnect, and we further propose a cost-efficient check-node unit (CNU) architecture, yielding a cost reduction of 56% with respect to the conventional CNU architecture.

We further implement FPGA-based layered decoder architectures using the proposed algorithms as decoding kernels, for a (3,6)-regular Quasi-Cyclic LDPC code of length 1296 bits, and evaluate them in terms of cost, throughput and decoding performance. Implementation results on Xilinx Virtex 6 FPGA device show that they can achieve a throughput between 1.95 and 2.41 Gbps for 20 decoding iterations (48% to 83% increase with respect to OMS), while providing decoding performance close to the OMS decoder, despite the impreciseness introduced in the processing units.

4.1 Introduction

Most hardware implementations of LDPC decoders are based on the Min-Sum (MS) algorithm [33] or enhanced versions of it, such as Normalized MS (NMS) and Offset MS (OMS) [16]. Further enhancements have been proposed in the literature, as follows. In [105], the authors proposed a modification of the OMS, in which the offset factor is adjusted iteratively, by using information from previous decoding steps. Similarly, adaptive NMS or OMS decoding algorithms have been proposed in [48, 102], where the normalization/offset factor is adjusted at each iteration according to information gained from the check-node processing step. In [48] the offset factor is determined by the magnitude of the minimum output of the check-node, while in [102] the normalization/offset factor is adjusted according to the check-node state (verified or not). It should be noticed that all the above-mentioned papers focus on improving the error correction performance of the NMS/OMS decoders, especially in case of irregular codes. This is achieved at the price of an increased computational complexity, required by the adaptation mechanism, which translates into cost and throughput penalties in case of hardware implementation.

In order to address cost-efficiency and high-throughput issues, we propose two new decoding algorithms and hardware implementations, obtained by introducing two levels of impreciseness in the OMS decoding. The first level of impreciseness concerns the offset factor: rather than subtracting a constant offset factor from check-node messages, we simply "erase" the value of the least significant bit (LSB). We refer to this decoding algorithm as Partially OMS (POMS) and show that it has the following advantages in terms of hardware implementation: (i) significant reduction of interconnect and memory requirements, and (ii) simple architecture for the check-node processing unit (CNU), which avoids the use of comparator trees. Besides, the error-correction performance of the proposed POMS is very close to that of the OMS, both outperforming the conventional MS decoding. Moreover, we introduce a second level of impreciseness in the CNU, by suppressing some of the signals computed by the POMS decoder. The corresponding decoder is referred to as Imprecise-POMS (I-POMS) and we show that it allows further improvements in hardware cost and throughput.

The rest of the chapter is organized as follows. Section 4.2 reminds the MS, OMS in shortly as well as the proposed POMS decoding algorithms. The hardware architecture of the MS, OMS, POMS decoders for (3,6)-regular QC-LDPC codes is discussed in Section 4.3. Section 4.4 presents the I-POMS decoder. Implementation results are presented in Section 4.5, and Section 4.6 concludes the chapter.

Algorithm 8 Layered MS / OMS / POMS decoding algorithms

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word
Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$ ▷ estimated codeword

[Initialization]

for all $n = 1, \ldots, N$ **do** $\boldsymbol{L}_n = \log \frac{\Pr(x_n=0|y_n)}{\Pr(x_n=1|y_n)}$;

for all $n = 1, \ldots, N$ **do** $\tilde{\gamma}_n = \gamma_n = \varphi(\boldsymbol{L}_n)$; ▷ input quantization

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\beta_{m,n} = 0$;

[Decoding Iterations]

for all $\ell = 1, \ldots, L$ **do** ▷ Loop over horizontal layers

  for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **VNU**

    $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$;

  for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **CNU**

    $\beta_{m,n} = \Big(\prod_{n' \in \mathcal{H}(m)\setminus n} \operatorname{sgn}(\alpha_{m,n'})\Big) \cdot |\beta|_{m,n}$;

    // where $|\beta|_{m,n}$ is defined in the algorithm footnote below

  for all $m \in \mathcal{M}_\ell$ and $n \in \mathcal{H}(m)$ **do** ▷ **AP-LLR**

    $\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}$;

end (horizontal layers loop)

Let $\alpha^{\text{SAT}}_{m,n}$ be the value of $\alpha_{m,n}$ (output of the VNU) saturated to a lower quantization level. In the CNU, the message amplitude $|\beta|_{m,n}$ is computed as follows, according to the decoding algorithm:

MS : $|\beta|_{m,n} = \min_{n' \in \mathcal{H}(m)\setminus n} \big(|\alpha^{\text{SAT}}_{m,n'}|\big)$

OMS: $|\beta|_{m,n} = \max\big(\min_{n' \in \mathcal{H}(m)\setminus n}\big(|\alpha^{\text{SAT}}_{m,n'}|\big) - \delta,\ 0\big)$
// where $\delta > 0$ is the offset factor

POMS: $|\beta|_{m,n} = \min_{n' \in \mathcal{H}(m)\setminus n} \big(|\alpha^{\text{SAT}}_{m,n'}|_{[\text{LSB}\leftarrow 0]}\big)$
// $x_{[\text{LSB}\leftarrow 0]}$ means that the LSB of x is set to 0

4.2 Proposed Partially Offset Min-Sum Decoding

MS, OMS, and POMS decoding algorithms are described in Algorithm 8. While our description assumes layered decoding [40] and finite quantization (for the needs of the hardware implementation), it can be easily extended to more general settings. We shall further assume that input LLRs and exchanged messages are quantized on q bits, while AP-LLR values are quantized on $\tilde{q}$ bits, with $q < \tilde{q}$. Subtractions and additions used in the Variable Node Unit (VNU) and AP-LLR update steps are implemented through the use of $\tilde{q}$-bit saturated adders. Variable-node messages ($\alpha_{m,n}$) are saturated to q bits just before entering the Check Node Unit (CNU). Hence, the amplitude values $|\alpha^{\text{SAT}}_{m,n}|$ used in the CNU are $(q-1)$-bit values. It can be noticed that the three decoders only differ in the computation of the amplitude of check-node messages, denoted by $|\beta|_{m,n}$. For the POMS decoding, the LSB of $|\alpha^{(\text{SAT})}_{m,n}|$ is set to zero prior to the minimum computation. It can be easily seen that

this is equivalent to first computing the minimum and then setting the LSB of the result to zero. Moreover, using superscripts to denote the corresponding algorithm, and assuming that the offset factor for the OMS decoding is $\delta = 1$, we have:

$$|\beta|_{m,n}^{(\mathrm{POMS})} = \begin{cases} |\beta|_{m,n}^{(\mathrm{MS})} & \text{if } \mathrm{LSB}\left[|\beta|_{m,n}^{(\mathrm{MS})}\right] = 0 \\ |\beta|_{m,n}^{(\mathrm{OMS})} & \text{otherwise } (\text{for } \delta = 1) \end{cases} \tag{4.1}$$

The proposed POMS operates the same as the OMS decoding only if $\mathrm{LSB}\left[|\beta|_{m,n}^{(\mathrm{MS})}\right] = 1$, and thus the offset correction (with $\delta = 1$) is only partially achieved. Consequently, the decoding performance of POMS is expected to be in between MS and OMS. Note that $\delta = 1$ is the optimal offset factor in case that exchanged messages are quantized on a relatively small number of bits (*e.g.*, $q = 4$). For such a quantization scheme, since the LSBs of $\beta_{m,n}^{(\mathrm{POMS})}$ messages are always zero and need not be stored, POMS may lead to significant savings in memory and interconnects.

4.3 Hardware Architecture for QC-LDPC Decoders with Layered Scheduling

4.3.1 Hardware Architecture for Min-Sum Based Decoders

As mentioned in Chapter 2, QC-LDPC codes are known to achieve error correction performance comparable to that of random codes, while facilitating the hardware implementation of the decoder. In this section, we consider a (3,6)-regular QC-LDPC code defined by a base matrix $\mathbf{B}$ of size $R \times C = 12 \times 24$, and expansion factor $z = 54$, corresponding to a parity check matrix $\mathbf{H}$ of size $M \times N$, with $M = z \cdot R = 54 \cdot 12 = 648$ and $N = z \cdot C = 54 \cdot 24 = 1296$. The code is $(3, 6)$-regular, meaning that $\mathbf{H}$ has 3 non-zero entries per column, and 6 non-zero entries per row. The base matrix of the (3,6)-regular LDPC code is given in Figure 4.1. It is worth noting that $\mathbf{B}$ is designed such that $\mathbf{H}$ can be divided in $L = 3$ horizontal *decoding layers*[1], with each layer corresponding to $\text{RPL} = 4$ consecutive rows of $\mathbf{B}$. This special case allows maximum parallelism to be exploited through the use of *full decoding layers*[2]. We further define $Z = z \times \text{RPL}$, corresponding to the number of parity checks (rows of $\mathbf{H}$) within one decoding layer, and referred to as the parallelism degree (of the hardware architecture). Each column of $\mathbf{H}$ has exactly one non-zero entry in each layer.

49	-1	-1	-1	-1	43	-1	-1	-1	-1	50	-1	-1	-1	-1	2	-1	27	-1	-1	-1	-1	-1	49
-1	-1	-1	10	41	-1	-1	-1	-1	52	-1	-1	32	-1	-1	-1	-1	-1	50	-1	50	-1	-1	-1
-1	-1	20	-1	-1	-1	-1	20	-1	-1	-1	51	-1	10	-1	-1	47	-1	-1	-1	-1	-1	33	-1
-1	24	-1	-1	-1	-1	22	-1	53	-1	-1	-1	-1	-1	31	-1	-1	-1	-1	18	-1	47	-1	-1
10	-1	-1	-1	15	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	50	-1	13	-1	-1	-1	-1	-1	53
-1	-1	44	-1	-1	6	-1	-1	-1	-1	-1	29	-1	40	-1	-1	16	-1	-1	-1	13	-1	-1	-1
-1	2	-1	-1	-1	-1	-1	13	41	-1	-1	-1	-1	-1	42	-1	-1	-1	-1	48	-1	49	-1	-1
-1	-1	-1	36	-1	-1	24	-1	-1	50	-1	-1	12	-1	-1	-1	-1	-1	10	-1	-1	-1	48	-1
-1	-1	47	-1	50	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	9	-1	7	-1	-1	-1	-1	-1	28
6	-1	-1	-1	-1	-1	5	-1	-1	-1	-1	13	-1	3	-1	-1	29	-1	-1	-1	16	-1	-1	-1
-1	-1	-1	35	-1	16	-1	-1	37	-1	-1	-1	4	-1	-1	-1	-1	-1	24	-1	-1	-1	29	-1
-1	24	-1	-1	-1	-1	-1	51	-1	38	-1	-1	-1	-1	6	-1	-1	-1	-1	23	-1	16	-1	-1

Figure 4.1: Base matrix of the $(3, 6)$-regular QC-LDPC code

The block diagram of MS, OMS and POMS decoders is illustrated in Figure 4.2. It can be summarized by the following main blocks:

Input/Output buffers. The input buffer, implemented as a number of Serial Input Parallel Output (SIPO) shift registers, is used to store the input LLR values (γ_n) received by the decoder. The output buffer, is used to store the hard bit estimates of the decoded word. Input/output buffers allow data load/offload operations to take place concomitantly with the decoding of the current codeword.

[1] A decoding layer consists of one or several consecutive rows of $\mathbf{B}$, assuming that they do not overlap (*i.e.*, each column has at most one non-negative entry within each layer).

[2] A decoding layer is said to be full if each column of the base matrix has one non-negative entry in one of the rows composing the layer.

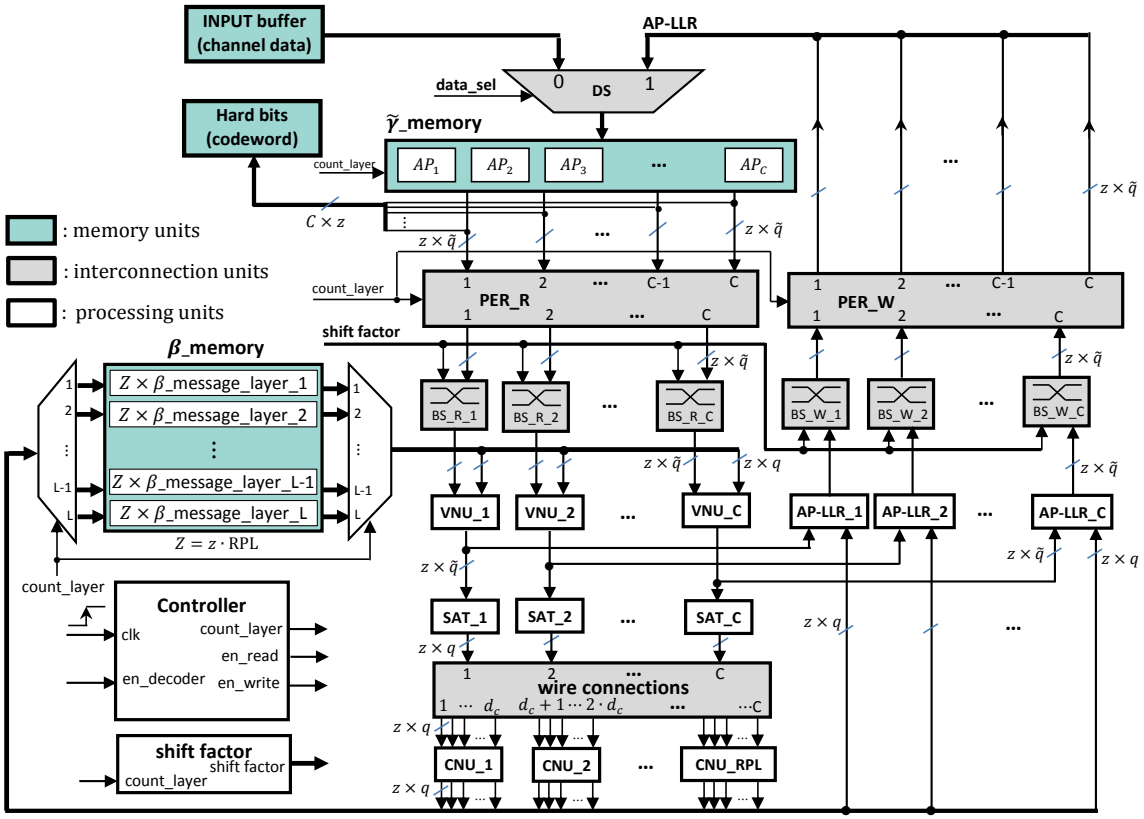


Figure 4.2: Block diagram for $(3, 6)$-regular QC-LDPC decoder

Memory blocks. Two memory blocks are used, one for the $\tilde{\gamma}_n$ values ($\tilde{\gamma}$_memory) and one for the $\beta_{m,n}$ messages (β_memory). $\tilde{\gamma}_n$ values are quantized on $\tilde{q} = 6$ bits, and $\beta_{m,n}$ values on $q = 4$ bits. Depending on the Check Node Unit (CNU) implementation, check-to-node messages (β_message) can be either "uncompressed" (*i.e.*, for a check-node m, the corresponding β_message is given by the d_c values $[\beta_{m,n_1}, \ldots, \beta_{m,n_{d_c}}]$, where $n_1, \ldots, n_{d_c}$ denote the variable nodes connected to m) or "compressed" (*i.e.*, for a check-node m, the corresponding β_message is given by the signs of the above β_{m,n_i} messages, their first and second minimum, denoted by min1 and min2, and the index of the first minimum, denoted by indx_min1) [99]. Figure 4.3 and Figure 4.4 show β_message in uncompressed and compressed format, respectively. Memory requirements for the $\beta_{m,n}$ values are further reduced for the POMS decoder, as will be discussed in the next section. All data is read and processed at during one clock cycle, then written during the consecutive clock cycle, and so on. Therefore, the decoder takes 2 clock cycles for each layer processing.

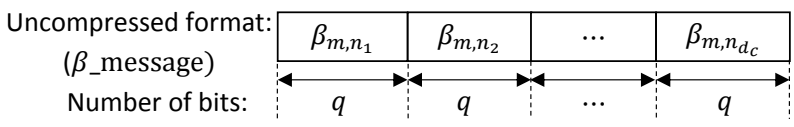


Figure 4.3: Uncompressed β-message

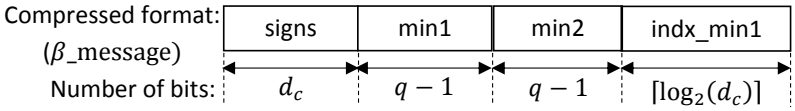


Figure 4.4: Compressed β-message

Read and Write Permutations (PER_R, PER_W). PER_R permutation is used to rearrange the data read from $\tilde{\gamma}$_memory, according to the processed layer, so as to ensure processing by the proper VNU/CNU. PER_W block operates oppositely to PER_R.

Read and Write Barrel Shifters (BS_R, BS_W). Barrel shifters are used to implement the cyclic (shift) permutations corresponding to the non-negative entries of the base matrix $\mathbf{B}$. For the architecture in Figure 4.2, we use 24 BS_R and 24 BS_W blocks, corresponding to the number of columns of $\mathbf{B}$. Each of them has 54 inputs and 54 outputs (for expansion factor $z = 54$).

Variable Node Units (VNUs) and AP-LLR Units. These units compute VN-messages ($\alpha_{m,n}$) and AP-LLR values ($\tilde{\gamma}_n$). Each VN-message is computed by subtracting the corresponding CN-message from the AP-LLR value, that is $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$. This operation is implemented by a $\tilde{q}$-bit subtractor, hence the $\alpha_{m,n}$ value outputted by the VNU is quantized on $\tilde{q}$ bits. The AP-LLR value is updated by the AP-LLR unit, by $\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}^{\text{new}}$, where $\beta_{m,n}^{\text{new}}$ is the corresponding CN-message computed at the current iteration (see below).

Saturators (SATs). Prior to CNU processing, $\alpha_{m,n}$ values are saturated to q bits. In other words, after saturation, $\alpha_{m,n}^{(\text{SAT})}$ values are within the range $\{-Q, \ldots, +Q\}$, where $Q = 2^{q-1} - 1$.

Check Node Units (CNUs). These processing units compute the CN-messages ($\beta_{m,n}$). Each CNU_i ($i = 1, \ldots, \text{RPL}$) block in Figure 4.2 consists of d_c inputs, each one of size $z \times q$ bits. Thus, each CNU block actually includes z computing units, used to process in parallel the z check-nodes within one row of the base matrix $\mathbf{B}$. The CNU is implemented by using either: (i) the high-speed low-cost tree-structure (TS) approach proposed in [101] for "compressed" CN-messages, or (ii) comparator trees for "uncompressed" CN-messages. The CNU architecture will be further detailed in Section 4.3.2, where the proposed CNU architecture for the POMS decoder will also be presented.

AP-LLR Units. These units compute the $\tilde{\gamma}_n$ values. Each AP-LLR_i ($i = 1, \ldots, C$) in Figure 4.2 consists of 54 6-bit saturated adders, for the parallel execution of 54 variable-nodes (corresponding to one column of $\mathbf{B}$).

Controller. This block generates control signals such as *count_layer* for indicating which layer is being processed, *en_read* and *en_write* for reading and writing data, etc. It also controls the synchronous execution of the other blocks.

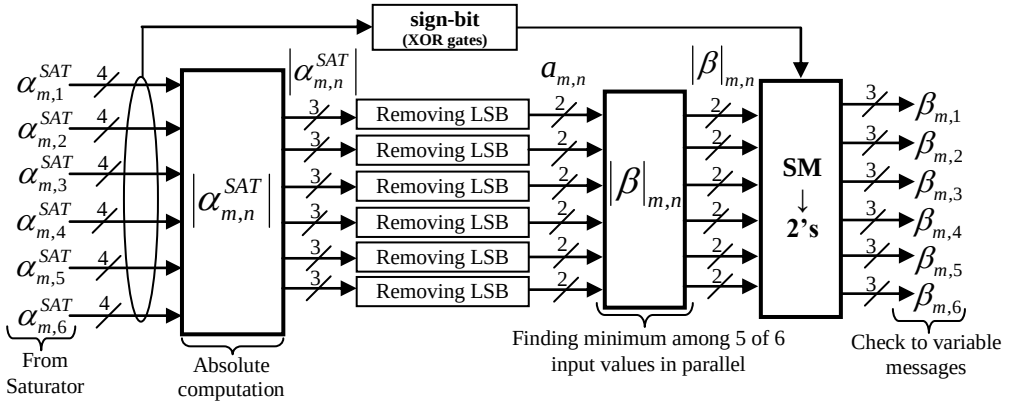


Figure 4.5: Proposed CNU architecture for POMS ($d_c = 6$)

4.3.2 Hardware Architecture for Proposed POMS Decoder

To ensure efficient implementation of the proposed POMS decoder, the CNU, VNU and AP-LLR blocks are modified as discussed below.

CNU Architecture. Many CNU architectures compute only the first minimum, the second minimum and index of the first minimum of the incoming $|\alpha_{m,n}^{(\mathrm{SAT})}|$ messages (based on the observation that the amplitudes of the outgoing check-node message are equal to the first minimum, except for the corresponding index for which the amplitude of the check-node message is given by the second minimum). Yet, as indicated in [107], this method is suitable for ASIC implementation, but for FPGA implementation it is three times more complex than parallel computation of $|\beta|_{m,n}$ by comparator trees. In this work, the CNU architecture proposed in [107] has been implemented, both for the MS and the OMS decoders.

We discuss now the CNU architecture for the POMS decoder. As mentioned before, the $\alpha_{m,n}$ messages are saturated to $q = 4$ bits by the SAT blocks. After saturation, the absolute value $|\alpha_{m,n}^{(\mathrm{SAT})}|$ is stored on 3 bits, and the LSB is set to zero by the POMS decoder. Let $a_{m,n}$ denote the 2-bit value obtained from $|\alpha_{m,n}^{(\mathrm{SAT})}|$ by removing LSB as in Figure 4.5. Let d_c be the number of incoming messages into the CNU (in our case $d_c = 6$). The computation of $|\beta|_{m,n}^{(\mathrm{POMS})}$ requires finding the minimum among $(d_c - 1)$ 2-bit values (precisely the $a_{m,n'}$ values, for $n' \in H(m) \setminus n$). Let us define:

$$AndMsb = \mathbf{AND}\left(a_{m,n'}[1]\right); \text{// AND of MSB of } (d_c-1) \text{ inputs} \tag{4.2}$$

$$AndLsb = \mathbf{AND}\left(a_{m,n'}[0]\right); \text{// AND of LSB of } (d_c-1) \text{ inputs} \tag{4.3}$$

$$Detect\_0 = \begin{cases} 0, & \text{if at least one input equals to 0} \\ 1, & \text{otherwise} \end{cases} \tag{4.4}$$

// signal that detects if at least one input equals to 0

With the above notation, the $|\beta|_{m,n}^{(\text{POMS})}$ 2-bit value (omitting LSB that is equal to 0) can be computed by:

$$|\beta|_{\substack{m,n \\ \text{(in binary format)}}}^{(\text{POMS})} (2 \text{ bits}) = \begin{cases} 01, & \text{if } [Detect\_0, AndMsb, AndLsb] = 100 \\ [AndMsb,\ AndLsb], & \text{otherwise} \end{cases} \tag{4.5}$$

where $[x, y]$ denotes concatenation operator.

VNU and AP-LLR Architecture. According to the above discussion, only 3 bits are required for each $\beta_{m,n}^{(\text{POMS})}$ message (2 bits for $|\beta|_{m,n}^{(\text{POMS})}$ and 1 bit for the sign), representing a reduction by 25% of the $\beta_$memory size, with respect to conventional MS and OMS decoders. To accommodate this change, in the VNU and AP-LLR blocks the $\beta_{m,n}^{(\text{POMS})}$ message is extended to 4 bits, by appending a zero LSB, as illustrated in Figure 4.6.

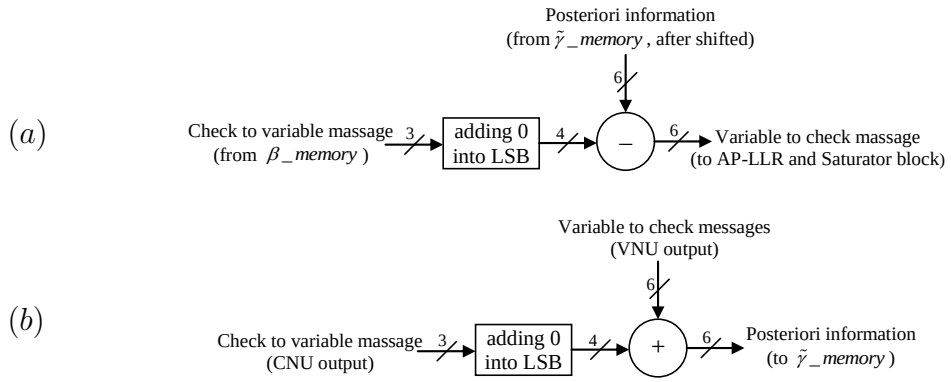


Figure 4.6: VNU (a) and AP-LLR (b) architectures for POMS decoder

4.4 Imprecise Partially Offset Min-Sum Decoder

This section describes the I-POMS decoder, obtained by introducing a second level of impreciseness in the CNU processing. According to the above section, the CNU of the POMS decoder comprises a *Detect_0* signal, which is used to obtain the correct minimum value in case that both *AndMsb* and *AndLsb* signals are 0. In case that $AndMsb = AndLsb = 0$, the *Detect_0* signal allows one distinguish between the following two possible cases: (i) at least one input is equal to 0 (hence the minimum is equal to 0), or (ii) no input is equal to zero, in which case there must be at least one input equal to 1 and another one equal to 2 (hence the minimum is equal to 1). However, compared to the *AndMsb* and *AndLsb* signals, the *Detect_0* signal requires much more logic gates to be implemented. To further simplify the CNU architecture, we propose an imprecise CNU by suppressing the *Detect_0* signal. Using the notation from the previous section, let us further define:

$$a^*_{m,n} = \begin{cases} a_{m,n}, & \text{if } a_{m,n} = 0, 1, 3 \\ 1, & \text{if } a_{m,n} = 2 \end{cases} \tag{4.6}$$

The I-POMS decoder computes $|\beta|^{(\text{I-POMS})}_{m,n} = [AndMsb^*, AndLsb^*]$, with $AndMsb^*$ and $AndLsb^*$ signals computed from $a^*_{m,n}$ values. Note that $|\beta|^{(\text{I-POMS})}_{m,n} \neq |\beta|^{(\text{POMS})}_{m,n}$ only if $a_{m,n'} \geq 2, \forall n'$ and there exist at least one $a_{m,n'}$ value equal to 2. Figure 4.7 illustrates the implementation of the CNU for the I-POMS decoder. It only uses 24 AND gates to compute six $|\beta|^{(\text{I-POMS})}_{m,n}$ messages in parallel.

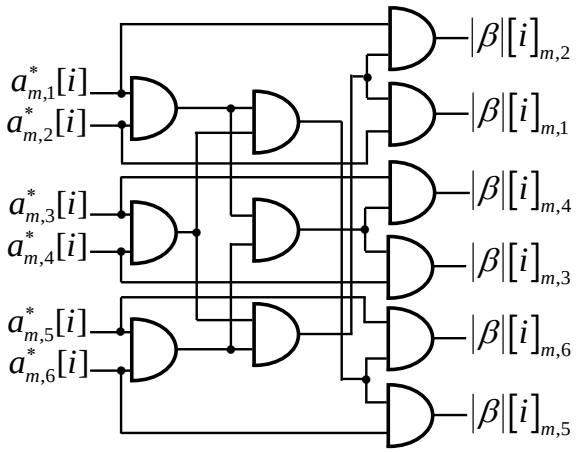


Figure 4.7: Diagram circuit for computing $|\beta|_{m,n}$ messages in parallel ($d_c = 6$) for I-POMS decoder (one such circuit is used for each of LSB and MSB)

4.5 Implementation Results

FPGA synthesis results reported in this section are after place and route and have been obtained by using the Xilinx tool ISE 14.6. Table 4.1 shows the hardware (HW) resources required to implement one CNU block for the four decoders under investigation. It can be seen that the CNU architectures proposed for the POMS and I-POMS decoders achieve significant HW resources reduction compared to the conventional CNU architecture used for MS and OMS (*e.g.*, the number of LUTs is reduced by 56%). Full implementation results for the four decoders are shown in Table 4.2. It can be seen that the proposed POMS not only saves 25% memory resources on FPGA, but also increases the throughput by 10% and 48%, compared with conventional MS and OMS, respectively. Since all data is processed in full parallel at each layer, the number of BRAMs may appear to be large, but only 3 among 512 words are used for each BRAM.

Table 4.1: CNU hardware resources for MS, OMS ($\delta = 1$), POMS, and I-POMS decoders ($d_c = 6$)

Decoder	**MS**	**OMS**	**POMS**	**I-POMS**
Device	Xilinx Virtex 6 (XC6VLX240T-1FF1156)–ISE 14.6			
No. slice LUTs/CNU unit (post PAR)	54	69	42	30
Cost reduction/CNU unit (compared to OMS)	21%	0%	39%	56%

Table 4.2: Implementation results for MS, OMS ($\delta = 1$), POMS, and I-POMS decoders

Decoder	**MS**	**OMS**	**POMS**	**I-POMS**
Device	**Xilinx Virtex 6 (XC6VLX240T-1FF1156)–ISE 14.6**			
LDPC code	**QC-Regular (3,6), code rate: 0.5, codeword length: 1296**			
No. slice registers	23352	23352	23352	23352
No. slice LUTs	95188	97604	93202	90872
No. RAMB36E1	144	144	96	96
No. RAMB18E1	0	0	24	24
Max freq (MHz)	164	122	181	223
Throughput (Gbps)	1.77	1.32	1.95	2.41

We use two different signals to control memory reading and writing in consecutive clock cycles. This significantly improves the maximum operating frequency, compared to using 1 signal for both reading and writing, but also leads to an increased number of registers. Depending on the application, one may trade-off between throughput and area. Decoding throughput is computed by:

$$\text{Throughput} = (\text{CodeLength} \times f_{\max}) \,/\, (n_{\text{iter}} \times n_{\text{cyc}}) \tag{4.7}$$

where $f_{\max}$ is the maximum operating frequency, n_{iter} is the maximum number of decoding iterations, and n_{cyc} is the number of clock cycles needed to complete one iteration. The throughput reported in Table 4.2 corresponds to $n_{\text{iter}} = 20$, with $n_{\text{cyc}} = 6$ (2 clock cycles per layer).

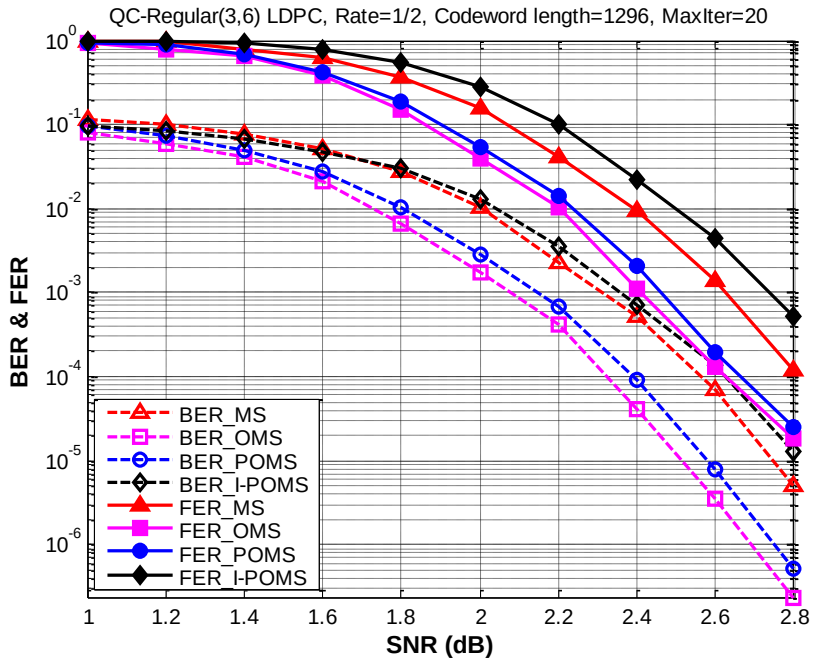


Figure 4.8: Decoding performance for proposed algorithms (AWGN channel)

Finally, to verify the error correction performance of POMS and I-POMS decoding algorithms, we have conducted Monte Carlo simulations for the (3,6)-regular code with rate 0.5, and codeword length 1296 bits. The maximum number of decoding iterations is set to 20. The Bit Error Rate (BER) and Frame Error Rate (FER) performance are shown in Figure 4.8. For comparison purposes, we have also included the BER and FER performance of the conventional MS and OMS ($\delta = 1$) decoders. It can be seen that the proposed POMS decoder operates very close to the OMS decoder. For a BER of 10^{-5}, the POMS is at only 0.07 dB from the OMS, while showing a gain of 0.16 dB with respect to the MS decoder. I-POMS performance is also comparable to that of the MS decoder (gap of 0.08 dB), in spite of the impreciseness introduced in the CNU, while the throughput is considerably increased (36%).

4.6 Conclusion

This chapter proposed the use of imprecise processing units for the OMS decoder, in order to improve the cost and the throughput of the hardware implementation. Impreciseness has been introduced at two levels: (i) within the offset correction, which is only partially achieved, and (ii) within the minimum computation required by the CNU. We further demonstrated the merits of the proposed POMS and I-POMS decoders in terms of hardware implementation. Hardware architectures have been proposed for $(3, 6)$-regular QC-LDPC codes, which highlighted the following advantages of our proposal: (i) significant memory and interconnect savings, (ii) simple CNU architecture, requiring only a very small number of AND gates, (iii) increased throughput, up to 2.41 Gbps and (iv) provides good to very good decoding performance, despite the imprecise processing units.

Chapter 5

Non-Surjective Finite Alphabet Iterative Decoders

This chapter introduces a new family of Non-Surjective Finite Alphabet Iterative Decoders (NS-FAIDs), characterized by specific processing rules, which can be theoretically analyzed and optimized for different trade-offs between hardware cost and error correction performance. The proposed approach builds upon the approximate storage technique, and allows storing the exchanged messages using a smaller precision than the processing units. It is also shown to provide a unified approach for several designs previously proposed in the literature.

NS-FAIDs are optimized by density evolution for regular and irregular LDPC codes, and are shown to provide different trade-offs between hardware complexity and decoding performance. In particular, we show that optimized NS-FAIDs may yield significant reductions of the memory size requirements, while providing even better or only slightly degraded error correction performance with respect to other MS-based decoders.

5.1 Introduction

This chapter targets the design of cost-effective Low-Density Parity-Check (LDPC) decoders, suitable for the new generation of communication systems, requiring increased data rates and reduced energy footprint. One important characteristic of LDPC decoders is that the memory and interconnect blocks dominate the overall area/delay/power performance of the hardware design. To address this issue, we build upon the concept of Finite Alphabet Iterative Decoders (FAIDs), introduced in [11, 76–78]. FAIDs have been seen as a novel approach to the design of finite precision iterative decoders. In this approach the messages are not log likelihood ratios, but "confidence" levels that belong to a finite alphabet. The finite alphabet contains a relatively small number of symbols (the one proposed by authors is 7), which requires a small number of bits for its representation. The main difference with the MS decoding consists in the update rule used in the variable-node processing units. A variable-node message is computed as an "inexact" sum of the incoming check-node messages. The inexact sum is defined through an inexact sum table, which is derived using knowledge of trapping sets [80] and are chosen to increase the guaranteed error correction capability of the code in the error floor region. Moreover, empirical results show that FAIDs with 3 bits messages may outperform floating-point SP decoding [76].

While FAIDs have been previously investigated for variable-node regular LDPC codes over the binary symmetric channel (BSC), this work extends their use to any channel model, and to both regular and irregular LDPC codes. The approach considered in this chapter is to allow storing the exchanged messages using a lower precision (smaller number of bits) than that used by the processing units. The basic idea is to reduce the size of the exchanged messages, once they have been updated by the processing units. Hence, to some extent, the proposed approach is akin to the use of *imprecise storage*, which is seen as an enabler for cost-effective, high-throughput, and/or low-power decoder designs.

The proposed approach, referred to as *Non-Surjective FAIDs* (NS-FAIDs), is shown to provide a unified framework for several designs previously proposed in the literature, including the Normalized and Offset **M**in-**S**um (MS) decoders [17], the Partially Offset MS decoder in Chapter 4, the MS-based decoders proposed in [13, 73], or the recently introduced Dual-quantization domain MS decoder [3]. It was shown that NS-FAIDs can be optimized by using the Density Evolution (DE) technique, such as to obtain the best possible decoding performance for given hardware constraints, expressed in terms of memory/interconnect reduction. The DE optimization is illustrated for (3,6)-regular and the WiMAX irregular LDPC codes [46]. As a results, we proposed a set of 12 *regular* NS-FAIDs and 27 *irregular* NS-FAIDs, with different trade-offs between hardware complexity and decoding performance. For instance, irregular NS-FAID-433 may provide decoding gains up to 0.36 dB, while yielding a VN-message size reduction by 17.76% compared to the MS decoder. Finally, to corroborate the analytic results obtained by DE, we conducted Monte-Carlo simulations for the (3,6)-regular with codeword length of 1296 bits and the WiMAX codes with rate 0.5, codeword length of 2304 bits.

The chapter is organized as follows. NS-FAIDs are introduced in Section 5.2, in which their implementation benefits and the DE analysis are also discussed. The optimization of regular and irregular NS-FAIDs is presented in Section 5.3. Section 5.4 concludes the chapter.

5.2 Non-Surjective Finite Alphabet Iterative Decoders

LDPC codes are defined by sparse bipartite graphs, comprising a set of variable-nodes (VNs), corresponding to coded bits, and a set of check-nodes (CNs), corresponding to parity-check equations. Finite Alphabet Iterative Decoders (FAIDs) are message-passing LDPC decoders that have been introduced in [76–78]. We state below the definition of FAIDs, in a slightly less general form than the one in [78]. Let Q be a positive integer. A $(2Q+1)$-level FAID is a 4-tuple $(\mathcal{M}, \Gamma, \Phi_v, \Phi_c)$, where:

- $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$ is the alphabet of the exchanged messages, and is also referred to as the *decoder alphabet*,
- $\Gamma \subseteq \mathcal{M}$ is the input alphabet of the decoder, *i.e.*, the set of all possible values of the quantized soft information supplied to the decoder,
- Φ_v and Φ_c denote the update rules for VNs and CNs, respectively.

The CN-update function Φ_c is the same for any FAID decoder, and is equal to the update function used by the MS decoder. Precisely, for a CN of degree d_c, the update function $\Phi_c : \mathcal{M}^{d_c-1} \to \mathcal{M}$ is given by:

$$\Phi_c\left(m_1, \ldots, m_{d_c-1}\right) = \left(\prod_{i=1}^{d_c-1} \operatorname{sgn}(m_i)\right) \min_{i=1,\ldots,d_c-1} |m_i| \tag{5.1}$$

The VN-update function $\Phi_v : \Gamma \times \mathcal{M}^{d_v-1} \to \mathcal{M}$, for a VN of degree d_v, is defined as:

$$\Phi_v\left(\gamma, m_1, \ldots, m_{d_v-1}\right) = F\left(\gamma + \sum_{j=1}^{d_v-1} m_j\right) \tag{5.2}$$

where the function $F : \mathbb{Z} \to \mathcal{M}$ is defined based on a set of threshold values $\mathcal{T} = \{T_0, T_1, \ldots, T_{Q+1}\} \subset \bar{\mathbb{R}}_+$, with $T_0 = 0$, $T_{Q+1} = +\infty$, and $T_i < T_j$ for any $i < j$:

$$F(x) = \operatorname{sgn}(x)m, \quad \text{where } m \text{ is s.t. } T_m \leq |x| < T_{m+1} \tag{5.3}$$

It can be easily seen that any non-decreasing odd function satisfies Eq. (5.3). Precisely, the following proposition holds.

Proposition 5.1 *For any function $F : \mathbb{Z} \to \mathcal{M}$, there exists a threshold set $\mathcal{T}$ such that F is given by Eq. (5.3), if and only if F satisfies the following two properties:*

(i) F is an odd function, i.e., $F(-x) = -F(x)$, $\forall x \in \mathbb{Z}$,

(ii) F is non-decreasing, i.e., $F(x) \leq F(y)$ for any $x < y$.

We note that the above proposition also implies that $F(0) = 0$ and $F(x) \geq 0, \forall x > 0$. In this chapter, we further extend the definition of FAIDs by allowing $F(0)$ to take on non-zero values. To ensure symmetry of the decoder, we shall write $F(0) = \pm\lambda$,

with $\lambda \geq 0$, meaning that $F(0)$ takes on either $-\lambda$ or $+\lambda$ with equal probability. In the following, F will be referred to as *framing function*.

As the focus of this work is on practical implementations, we will further assume that the sum $\gamma + \sum_{j=1}^{d_v-1} m_j$ in Eq. (5.2) is saturated to $\mathcal{M}$, prior to applying F on it. Consequently, in the sequel we shall only consider framing functions $F : \mathcal{M} \to \mathcal{M}$, and the VN-update function Φ_v from Eq. (5.2) is redefined as:

$$\Phi_v\left(\gamma, m_1, \ldots, m_{d_v-1}\right) = F\left(s_{\mathcal{M}}\left(\gamma + \sum_{j=1}^{d_v-1} m_j\right)\right) \tag{5.4}$$

where $s_{\mathcal{M}} : \mathbb{Z} \to \mathcal{M}$, $s_{\mathcal{M}}(x) = \operatorname{sgn}(x)\min(|x|, Q)$, is the saturation function.

Since $F(-x) = -F(x), \forall x \in \mathcal{M}$, F is completely determined by the vector $[|F(0)|, F(1), ..., F(Q)]$, further referred to as the *Look-Up Table (LUT)* of F, which satisfies the following inequalities (Proposition 5.1):

$$0 \leq |F(0)| \leq F(1) \leq \cdots \leq F(Q) \leq Q \tag{5.5}$$

Summarizing, the subclass of FAIDs considered in this chapter is defined by Eq. (5.4), where F is a framing function satisfying Eq. (5.5). Furthermore, for any integer $q > 0$, the expression *q-bit FAID* is used to refer to a $(2Q + 1)$-level FAID, with $Q = 2^{q-1} - 1$. It follows that messages exchanged within the FAID decoder are q-bit messages (including 1 bit for the sign).

5.2.1 Non-Surjective FAIDs

The finite alphabet MS decoder is a particular example of FAID, with framing function $F : \mathcal{M} \to \mathcal{M}$ being the identity function. Using Eq. (5.5) it can be easily verified that this is the only FAID for which the framing function F is *surjective* (or equivalently *bijective*, since $\mathcal{M}$ is finite). For any other framing function F, there exists at least one element of $\mathcal{M}$, which is not in the image of F. The class of FAIDs defined by non-surjective framing functions is investigated in this section.

Although we restrict the study in this work to non-surjective framing functions $F : \mathcal{M} \to \mathcal{M}$, it can be readily extended to non-surjective framing functions $F : \mathbb{Z} \to \mathcal{M}$. While such an extension would widen the class of NS-FAIDs, our preliminary investigations have shown that the best NS-FAIDs are actually those within the subclass of NS-FAIDs defined by $F : \mathcal{M} \to \mathcal{M}$, investigated in this chapter.

Definition 5.2 *The* weight *of a framing function $F : \mathcal{M} \to \mathcal{M}$, denoted by W, is the number of distinct entries in the vector $[|F(0)|, F(1), ..., F(Q)]$. It follows that $1 \leq W \leq Q + 1$. By a slight abuse of terminology, we shall also refer to W as the weight of the NS-FAID. We further define the "framing bit-length" as $w = \lceil \log_2(W) \rceil + 1$.*

Definition 5.3 *A non-surjective FAID (NS-FAID) is a FAID of weight $W < Q+1$. Hence the framing function F is non-surjective, meaning that the image set of F is a strict subset of $\mathcal{M}$.*

Table 5.1: Examples of 4-bit framing functions of weight $W = 4$

m	0	1	2	3	4	5	6	7
$F_1(m)$	0	1	1	3	3	3	7	7
$F_2(m)$	± 1	1	1	3	3	4	4	7

Table 5.1 provides two examples of $q = 4$-bit NS-FAIDs (hence $Q = 7$), both of which are of weight $W = 4$, hence framing bit-length $w = \lceil \log_2 4 \rceil + 1 = 3$. Note that F_1 maps 0 to 0, while F_2 maps 0 to ± 1. The image sets of F_1 and F_2 are $\mathrm{Im}(F_1) = \{0, \pm 1, \pm 3, \pm 7\}$ and $\mathrm{Im}(F_2) = \{\pm 1, \pm 3, \pm 4, \pm 7\}$.

The main motivation for the introduction of NS-FAIDs is that they allow reducing the size of the memory required to store the exchanged messages. Clearly, for a NS-FAID with framing bit-length w ($w \leq q$), the exchanged messages can be represented using only w instead of q bits (including 1 bit for the sign). Moreover, as a consequence of the message size reduction, the size of the interconnect network that carries the messages from the memory to the processing units is also reduced.

Proposition 5.4 *The number of $(2Q+1)$-level NS-FAIDs of weight W is given by:*

$$N_{\text{NS-FAID}}(Q, W) = \binom{Q}{W-1}\binom{Q+1}{W} \tag{5.6}$$

where $\binom{y}{x}$ denotes the binomial coefficient.

5.2.2 Examples of NS-FAIDs

As mentioned above, if the framing function F is the identity function, the corresponding FAID is equivalent to the MS decoder with finite alphabet $\mathcal{M}$. Some examples of NS-FAIDs are provided below.

Example 1. Let $F : \mathcal{M} \to \mathcal{M}$ be defined by:

$$F(x) = [\nu \cdot x]_{\mathcal{M}} \tag{5.7}$$

where $\nu \in]0, 1[$, and $[\xi]_{\mathcal{M}}$ denotes the closest integer to ξ that belongs to $\mathcal{M}$. Then, the corresponding NS-FAID is the Normalized Min-Sum (NMS) decoder with normalization (scaling) factor ν.

Example 2. Let $F : \mathcal{M} \to \mathcal{M}$ be defined by:

$$F(x) = \mathrm{sgn}(x) \max(|x| - \delta, 0) \tag{5.8}$$

where δ is an integer value, such that $0 < \delta < Q$. Then, the corresponding NS-FAID is the Offset Min-Sum (OMS) decoder with offset factor δ.

Example 3. Let $F : \mathcal{M} \to \mathcal{M}$ be defined by:

$$F(x) = \begin{cases} x, & \text{if } |x| \text{ is even} \\ \mathrm{sgn}(x)(|x| - 1), & \text{if } |x| \text{ is odd} \end{cases} \tag{5.9}$$

Then, the corresponding NS-FAID is the Partially OMS (POMS) decoder in Chapter 4.

Moreover, it can be seen that the MS-based decoders proposed in [13, 73] and the dual-quantization domain decoder proposed in [3] are particular realizations of NS-FAIDs.

While the main reason behind the NS-FAIDs definition consists in their ability to reduce memory and interconnect requirements, we can also argue that they may allow improving the error correction performance (with respect to MS). This is the case of both OMS and POMS decoders mentioned above. Given a target message bit-length w (*e.g.*, corresponding to some specific memory constraint), one may try to find the framing function F of corresponding weight W, which yields the best error correction performance. The optimization of the framing function can be done by using the DE technique, which will be discussed in Section 5.2.4.

Since F is a non-decreasing function, the framing function F can alternatively be applied at the CN-processing step (instead of VN-processing), while resulting in an equivalent decoding algorithm. Whether F is applied at the VN-processing or the CN-processing step is rather a matter of implementation. When F is applied at the VN-processing step, both VN- and CN-messages belong to a strict subset of the alphabet $\mathcal{M}$, namely $\mathcal{M}' = \mathrm{Im}(F) \subset \mathcal{M}$. When F is applied at the CN-processing step, only CN-messages belong to $\mathcal{M}'$. However, it is worth noting that many hardware implementations of Quasi-Cyclic (QC) LDPC decoders rely on a layered architecture, which only requires storing the check-node messages [9].

5.2.3 Irregular NS-FAIDs

In case of irregular LDPC codes, *irregular NS-FAIDs* are NS-FAIDs using different framing functions F_{d_v} for VNs of different degrees d_v. Framing functions F_{d_v} may have different weights W_{d_v}. In this case, messages outgoing from degree-d_v VNs can be represented by using only $w_{d_v} = \lceil \log_2(W_{d_v}) \rceil + 1$ bits. However, the message size reduction does not necessarily apply to CN-messages, due to the fact that a CN may be connected to VNs of different degrees. Let $\mathcal{M}'_{d_v} = \mathrm{Im}(F_{d_v})$. Then, messages outgoing from a CN m can be represented by using:

$$\left\lceil \log_2 \left(\left| \cup_{d_v \in \mathcal{D}_m} \mathcal{M}'_{d_v} \right| \right) \right\rceil \text{ bits}, \tag{5.10}$$

where $\mathcal{D}_m$ is the set of degrees of VNs connected to m, and $|\cdot|$ is used to denote the number of elements of a set.

Alternatively, it is also possible to define CN-irregular NS-FAIDs in a similar manner. However, in this work we only deal with VN-irregular NS-FAIDs, since most of the practical irregular LDPC codes are irregular on VNs, while almost regular (or semi-regular) on CNs. In order to reduce the size of the CN-messages, in Section 5.3.2 we will further impose certain conditions on the framing functions F_{d_v}, by requiring their images being included in one another.

5.2.4 Density Evolution Analysis

For the sake of simplicity, we only consider transmission over *binary-input* memoryless noisy channels. We assume that the channel input alphabet is $\mathcal{X} = \{-1, +1\}$, with the usual convention that $+1$ corresponds to the 0-bit and -1 corresponds to the 1-bit, and denote by $\mathcal{Y}$ the output alphabet of the channel. We denote by $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ the transmitted and received symbols, respectively.

We further consider a function $\varphi : \mathcal{Y} \to \Gamma$ that maps the output alphabet of the channel to the input alphabet of the decoder, and set $\gamma = \varphi(y)$. Hence, φ encompasses both the computation of the soft (unquantized) log-likelihood ratio (LLR) value and its quantization. We shall refer to φ as *quantization map* and to γ as the *input LLR* of the decoder.

Assumption: For transmission over the binary-input Additive White Gaussian Noise (AWGN) channel, we shall consider that the decoder's input information as well as the exchanged messages are quantized on the same number of bits; therefore $\Gamma = \mathcal{M}$ unless otherwise stated. In this case, $y = x + z$, where z is the white Gaussian noise with variance σ^2, and the quantization map $\varphi : \mathcal{Y} \to \mathcal{M}$ is defined by:

$$\varphi(y) = [\mu \cdot y]_{\mathcal{M}} \tag{5.11}$$

where $\mu > 0$ is a constant referred to as *gain factor*, and $[x]_{\mathcal{M}}$ denotes the closest integer to x that belongs to $\mathcal{M}$ (see also in Chapters 2 and 3).

The objective of the DE technique is to recursively compute the probability mass function (pmf) of the exchanged messages, through the iterative decoding process. This is done under the assumption that exchanged messages are independent, which holds in the asymptotic limit of the code length. In this case, the decoding performance converges to the cycle free case. DE equations for the NS-FAID can be derived in a similar way as for the finite-alphabet MS decoder in Chapter 2. The only modification required is to take into account the framing function F applied at the VN-processing step, which can be easily done using the following:

Proposition 5.5 *Let $A_{MS}^{(\ell)}$ denote the pmf of VN-messages $\alpha_{MS} \overset{\text{def}}{=} s_{\mathcal{M}}\left(\gamma + \sum_{j=1}^{d_v-1} m_j\right)$ computed by the MS decoder at iteration ℓ, and $F : \mathcal{M} \to \mathcal{M}$ be a framing function. Then the pmf of $\alpha_{NS\text{-}FAID} = F(\alpha_{MS})$, denoted by $A_{NS\text{-}FAID}^{(\ell)}$, is given by:*

$$A_{NS\text{-}FAID}^{(\ell)}(m) = \sum_{x \in \mathcal{M}: F(x) = m} A_{MS}^{(\ell)}(x)$$

Similar to Chapter 2, the DE is used to compute the asymptotic error probability, defined as:

$$P_e^{(+\infty)} = \lim_{\ell \to +\infty} P_e^{(\ell)} \tag{5.12}$$

where $P_e^{(\ell)}$ is the bit error probability at iteration ℓ.

For a target bit error probability $\eta > 0$, the *η-threshold* is defined as the worst channel condition for which decoding error probability is less than η. Assuming the binary-input AWGN channel model, the η-threshold corresponds to the maximum noise variance σ^2 (or equivalently minimum SNR), such that the asymptotic error probability is less than η:

$$\sigma^2_{\text{thres}}(\eta) = \sup\left\{\sigma^2 \mid P_e^{(+\infty)} \leq \eta\right\} \tag{5.13}$$

In case that $\eta = 0$, the η-threshold is simply referred to as DE threshold [83]. However, the asymptotic decoding performance of finite-precision MS-based decoders is known to exhibit an error floor phenomenon at high SNR [69]. This makes the η-threshold definition more appropriate in practical cases, when the target bit error rate can be fixed to a practical non-zero value.

Finally, it is worth noting that the $\sigma^2_{\text{thres}}(\eta)$ value depends on:

(i) the irregularity of the LDPC code, parametrized as usual by the degree distribution polynomials $\lambda(x)$ and $\rho(x)$ [83],

(ii) the NS-FAID, *i.e.*, the size of the decoder alphabet and the framing function F,

(iii) the channel quantizer φ, or equivalently the gain factor μ used in Eq. (5.11).

Therefore, assuming that the degree distribution polynomials $\lambda(x)$ and $\rho(x)$ and the size of the decoder alphabet are fixed, we use the DE technique to jointly optimize the framing function F and channel quantizer.

5.3 Density Evolution Optimization of NS-FAIDs

Throughout this section, we consider q = 4-bit NS-FAIDs (hence, $Q = 7$). To illustrate the trade-off between hardware complexity and decoding performance, we consider the optimization of both regular and irregular NS-FAIDs.

5.3.1 Optimization of Regular NS-FAIDs

In this section, we consider the optimization of regular NS-FAIDs for $(d_v = 3, d_c = 6)$-regular LDPC codes. To illustrate the trade-off between hardware complexity and decoding performance, we consider $q = 4$-bit NS-FAIDs (hence, $Q = 7$), with framing bit-length parameter $w \in \{2, 3\}$. According to Proposition 5.4, the number of NS-FAIDs is given by:

$$N_{\text{NS-FAID}}(q = 4, w = 3) = N_{\text{NS-FAID}}(Q = 7, W = 4) = \binom{7}{3}\binom{8}{4} = 2450 \tag{5.14}$$

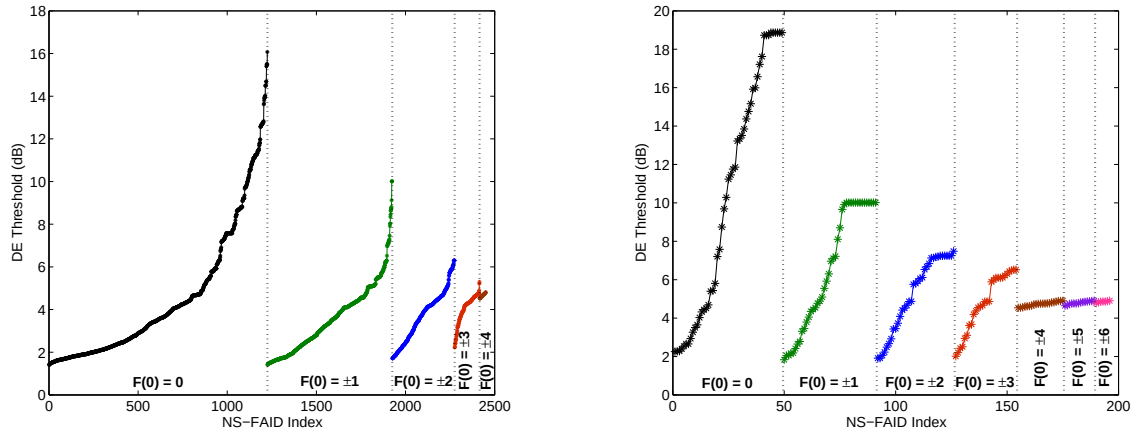
$$N_{\text{NS-FAID}}(q = 4, w = 2) = N_{\text{NS-FAID}}(Q = 7, W = 2) = \binom{7}{1}\binom{8}{2} = 196 \tag{5.15}$$

All regular NS-FAIDs have been evaluated by using the DE technique from Section 5.2.4, and their DE thresholds ($\eta = 0$) are shown in Figure 5.1 (note that the DE threshold computation also encompasses the optimization of the gain factor μ). NS-FAIDs are ordered according to the $F(0)$ value, and then according to their DE threshold. Best NS-FAIDs are those minimizing the DE threshold.

Figure 5.2 shows only the NS-FAIDs with DE thresholds less than 3 dB. For comparison purposes, the DE threshold of the baseline $q = 4$-bit MS decoder is also shown (MS threshold is equal to 1.643 dB, for a gain factor $\mu = 5.6$).

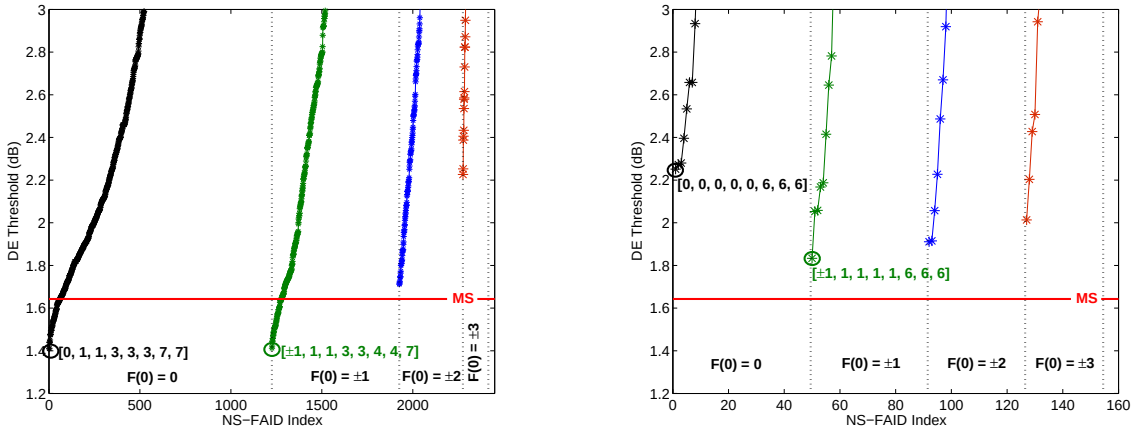
For $w = 3$, it can be observed that several NS-FAIDs with $F(0) = 0$ or $F(0) = \pm 1$ have better DE thresholds than the MS decoder. The best NS-FAID is given by the framing function $F = [0, 1, 1, 3, 3, 3, 7, 7]$ and its DE threshold is equal to 1.409 dB (gain factor $\mu = 3.8$), representing a gain of 0.23 dB compared to MS. Also shown in the figure is the best NS-FAID with $F(0) = \pm 1$: it is given by the framing function $F = [\pm 1, 1, 1, 3, 3, 4, 4, 7]$ and its DE threshold is equal to 1.412 dB (gain factor $\mu = 5.1$).

For $w = 2$, the best NS-FAID is given by the framing function $F = [\pm 1, 1, 1, 1, 1, 6, 6, 6]$ and its DE threshold is equal to 1.834 dB (gain factor $\mu = 6.4$), which represents a performance loss of only 0.19 dB compared to MS. It is worth noticing that for $w = 2$, the best NS-FAID is defined by a framing function with $F(0) = \pm 1$. Moreover, it can be seen that there are NS-FAIDs with $F(0) = \pm 2$ or $F(0) = \pm 3$ with better DE thresholds than the best NS-FAIDs with $F(0) = 0$. The latter is given by the framing function $F = [0, 0, 0, 0, 0, 6, 6, 6]$ and its DE threshold is equal to 2.251 dB (gain factor $\mu = 8.6$), thus resulting in a performance loss of 0.61 dB compared to MS.



(a) $w = 3$ (b) $w = 2$

Figure 5.1: Density evolution thresholds of regular $q = 4$-bit NS-FAIDs with $w = 3$ and $w = 2$



(a) $w = 3$ (b) $w = 2$

Figure 5.2: Density evolution thresholds of best regular $q = 4$-bit NS-FAIDs with $w = 3$ and $w = 2$

Table 5.2: Best NS-FAIDs for $(3, 6)$-regular LDPC codes

	$w = 2$		$w = 3$	
	F (LUT)	SNR-thres (dB) & gain factor μ	F (LUT)	SNR-thres (dB) & gain factor μ
$F(0) = 0$	$[0, 0, 0, 0, 0, 6, 6, 6]$	2.251 ($\mu = 8.6$)	$[\mathbf{0, 1, 1, 3, 3, 3, 7, 7}]$	**1.409** ($\mu = \mathbf{3.8}$)
$F(0) = \pm 1$	$[\pm\mathbf{1, 1, 1, 1, 1, 6, 6, 6}]$	**1.834** ($\mu = \mathbf{6.4}$)	$[\pm 1, 1, 1, 3, 3, 4, 4, 7]$	1.412 ($\mu = 5.1$)
$F(0) = \pm 2$	$[\pm 2, 2, 2, 2, 2, 2, 2, 7]$	1.911 ($\mu = 8.3$)	$[\pm 2, 2, 2, 3, 3, 3, 4, 7]$	1.712 ($\mu = 7.1$)
$F(0) = \pm 3$	$[\pm 3, 3, 3, 3, 3, 3, 3, 7]$	2.014 ($\mu = 9.4$)	$[\pm 3, 3, 3, 3, 3, 4, 5, 7]$	2.227 ($\mu = 10.0$)
$F(0) = \pm 4$	$[\pm 4, 4, 4, 4, 4, 7, 7, 7]$	4.535 ($\mu = 10.5$)	$[\pm 4, 4, 4, 4, 5, 6, 6, 7]$	4.526 ($\mu = 9.8$)
$F(0) = \pm 5$	$[\pm 5, 5, 5, 5, 5, 5, 7, 7]$	4.648 ($\mu = 10.9$)	N/A	
$F(0) = \pm 6$	$[\pm 6, 6, 6, 6, 6, 6, 6, 7]$	4.783 ($\mu = 11.0$)	N/A	

Table 5.2 summarizes the best NS-FAIDs according to the $F(0)$ value; DE thresholds and corresponding gain factors (μ) are also reported. Best NS-FAIDs for $w = 2$ and $w = 3$ are emphasized in bold.

5.3.2 Optimization of Irregular NS-FAIDs

As a case study, we consider the optimization of irregular NS-FAIDs for the WiMAX irregular LDPC codes with rate 1/2 [46] (of course, the proposed method can be applied to any other irregular codes in the same manner). The *edge-perspective* degree distribution polynomials are given by:

$$\lambda(x) = 0.2895x + 0.3158x^2 + 0.3947x^5 \tag{5.16}$$
$$\rho(x) = 0.6316x^5 + 0.3684x^6 \tag{5.17}$$

Hence, VNs are of degree $d_v \in \{2, 3, 6\}$. For each VN-degree d_v, we consider that the corresponding framing function F_{d_v} may be of any weight $W_{d_v} \in \{2, 4, 8\}$, corresponding to a framing bit-length $w_{d_v} \in \{2, 3, 4\}$. Therefore, the total number of framing functions is given by (see Proposition 5.4):

$$N_{\text{NS-FAID}}(7, 2) + N_{\text{NS-FAID}}(7, 4) + N_{\text{NS-FAID}}(7, 8) = 2645 \tag{5.18}$$

Since a different framing function may be applied for each VN-degree, it follows that the total number of irregular NS-FAIDs is equal to $2645^3 = 18\,504\,486\,125$. Clearly, even though we rely on DE, it is practically impossible to evaluate the decoding performance of all the irregular NS-FAIDs. To overcome this problem, we proceed as described below.

5.3.2.1 Optimization procedure

First, we evaluate the DE thresholds of NS-FAIDs applying one and the same framing function to all the variable-nodes, irrespective of their degree, which for simplicity will be referred to as *uniform* NS-FAIDs throughout this section. Uniform NS-FAIDs with framing bit-length $w = \{2, 3, 4\}$ are then sorted with increasing DE threshold value, from the best to the worst decoder. Note that the case $w = 4$ represents a slight abuse of terminology, since there is only one such decoder, corresponding to the original MS decoder. We further denote by $\mathcal{U}^{(\text{best})}$-NS-FAID-$w$ the set of best uniform NS-FAIDs with framing bit-length w, determined as follows:

- $\mathcal{U}^{(\text{best})}$-NS-FAID-2 is comprised of the uniform NS-FAIDs with $w = 2$, whose DE threshold is less than or equal to 5 dB; this represents 121 decoders out of the total of $N_{\text{NS-FAID}}(Q = 7, w = 2) = 196$ decoders.

- $\mathcal{U}^{(\text{best})}$-NS-FAID-3 is comprised of the uniform NS-FAIDs with $w = 3$, whose DE threshold is less than or equal to 3 dB; this represents 946 decoders out of the total of $N_{\text{NS-FAID}}(Q = 7, w = 3) = 2450$ decoders.

- $\mathcal{U}^{(\text{best})}$-NS-FAID-4 is comprised of the MS decoder only; its DE threshold is equal to 1.374 dB.

For irregular NS-FAIDs, we denote by NS-FAID-$w_2w_3w_6$ the ensemble of NS-FAIDs defined by a triplet of framing functions (F_2, F_3, F_6), corresponding to variable node degrees d_v=2, 3, 6, with framing bit-lengths w_2, w_3, w_6. Since $w_2, w_3, w_6 \in \{2, 3, 4\}$, there are 27 such ensembles. Since the number of NS-FAIDs in these ensembles can be very large, we only evaluate part of them, by further imposing the following two constraints:

(i) *Decoding performance constraint*: We only consider irregular NS-FAIDs defined by triplets of framing functions (F_2, F_3, F_6), such that $F_{d_v} \in \mathcal{U}^{(\text{best})}$-NS-FAID-$w_{d_v}$, for any $d_v \in \{2, 3, 6\}$.

(ii) *Memory size reduction constraint*: We further impose the following inclusion constraint between the image sets of framing functions used for different VN-degrees. Let $w^{(\max)} = \max_{d_v} w_{d_v}$ and $d_v^{(\max)} = \text{argmax}_{d_v} w_{d_v}$. We require that $\text{Im}\,(F_{d_v}) \subseteq \text{Im}\left(F_{d_v^{(\max)}}\right)$, $\forall d_v \in \{2, 3, 6\}$. According to Eq. (5.10), this constraint ensures that CN-messages can be represented by using only $w^{(\max)}$ bits, which is particularly suitable for layered architectures.

The number of irregular NS-FAIDs that satisfy the above two constraints (all NS-FAID-$w_2w_3w_6$ ensembles included) is equal to $N_{\text{NS-FAIDs}}^{\text{irregular}} = 7\,017\,762$.

5.3.2.2 Density Evolution evaluation

For each of the above $N_{\text{NS-FAIDs}}^{\text{irregular}}$ irregular NS-FAIDs, we compute its decoding threshold for a target bit error rate $\eta = 10^{-6}$, using the DE technique from Section 5.2.4. The threshold computation also encompasses the optimization of the gain factor μ. Hence, for each NS-FAID, we first determine the gain factor μ that maximizes the η-threshold defined in Eq. (5.13). The corresponding η-threshold value is then reported as the η-threshold of the NS-FAID.

Density evolution results for the MS decoder (indicated as NS-FAID-444), as well as NS-FAID decoders with the best η-thresholds from five NS-FAID-$w_2w_3w_6$ ensembles, are shown in Table 5.3: the framing functions used for VN-degrees $d_v = 2, 3, 6$ are shown in columns 2, 3, and 4, while the η-threshold value (in dB) and the corresponding gain factor μ are shown in column 5.

The SNR gain (+) or loss (−) reported in column 6 corresponds to the differences between the the SNR threshold of the MS decoder (NS-FAID-444) and the SNR threshold of the best NS-FAID-$w_2w_3w_6$. The memory size reduction of the NS-FAID-$w_2w_3w_6$ decoders compared to the MS decoder is reported in columns 7-9, for both VN and CN-messages. For CN-messages, two possibilities are considered, according to whether they are stored in an *uncompressed* or *compressed* format, where compressed format means that only the signs, first minimum, second minimum, and index of the first minimum are stored [99]. Finally, the framing functions' LUTs of the best NS-FAID-$w_2w_3w_6$ decoders are reported in Table 5.4 (L`x` in Table 5.4 corresponds to LUT`x` in Table 5.3).

Table 5.3: Hardware Complexity vs. Decoding Performance Trade-Off for Optimized Irregular NS-FAIDs

NS-FAIDs Ensemble	Framing functions applied to			SNR-thres (dB) (& gain factor μ) @BER $= 10^{-6}$	SNR gain/loss (+/− dB)	Memory size reduction (%)		
	$d_v = 2$	$d_v = 3$	$d_v = 6$			VN-mess.	CN-messages	
							uncomp.	comp.
NS-FAID-444	LUT0	LUT0	LUT0	1.374 (μ=3.2)	0.000	0.00	0.00	0.00
NS-FAID-443	LUT0	LUT0	LUT1	1.073 (μ=2.9)	+0.301	−9.87	0.00	0.00
NS-FAID-434	LUT0	LUT8	LUT0	1.073 (μ=2.8)	+0.301	−7.90	0.00	0.00
NS-FAID-344	LUT10	LUT0	LUT0	1.203 (μ=2.6)	+0.171	−7.24	0.00	0.00
NS-FAID-442	LUT0	LUT0	LUT15	1.224 (μ=3.1)	+0.150	−19.74	0.00	0.00
NS-FAID-424	LUT0	LUT15	LUT0	1.352 (μ=3.1)	+0.021	−15.79	0.00	0.00
NS-FAID-244	LUT18	LUT0	LUT0	2.106 (μ=2.3)	−0.732	−14.47	0.00	0.00
NS-FAID-432	LUT0	LUT3	LUT17	1.188 (μ=3.0)	+0.186	−27.63	0.00	0.00
NS-FAID-423	LUT0	LUT17	LUT2	1.262 (μ=3.1)	+0.112	−25.66	0.00	0.00
NS-FAID-324	LUT4	LUT17	LUT0	1.464 (μ=2.6)	−0.091	−23.03	0.00	0.00
NS-FAID-342	LUT10	LUT0	LUT17	1.278 (μ=2.6)	+0.096	−26.97	0.00	0.00
NS-FAID-234	LUT21	LUT6	LUT0	1.998 (μ=2.7)	−0.624	−22.37	0.00	0.00
NS-FAID-243	LUT21	LUT0	LUT5	1.997 (μ=2.8)	−0.624	−24.34	0.00	0.00
NS-FAID-422	LUT0	LUT17	LUT19	1.509 (μ=3.4)	−0.135	−35.52	0.00	0.00
NS-FAID-242	LUT21	LUT0	LUT15	2.049 (μ=2.9)	−0.676	−34.21	0.00	0.00
NS-FAID-224	LUT21	LUT15	LUT0	2.155 (μ=2.9)	−0.781	−30.27	0.00	0.00
NS-FAID-433	LUT0	LUT9	LUT8	1.015 (μ=2.8)	+0.359	−17.76	0.00	0.00
NS-FAID-343	LUT10	LUT0	LUT8	1.085 (μ=2.4)	+0.289	−17.11	0.00	0.00
NS-FAID-334	LUT10	LUT8	LUT0	1.102 (μ=2.3)	+0.272	−15.13	0.00	0.00
NS-FAID-233	LUT21	LUT7	LUT7	2.046 (μ=2.6)	−0.672	−32.24	−25.00	−13.04
NS-FAID-323	LUT11	LUT17	LUT5	1.457 (μ=2.9)	−0.083	−32.90	−25.00	−13.04
NS-FAID-332	LUT10	LUT9	LUT17	1.273 (μ=2.6)	+0.101	−34.87	−25.00	−13.04
NS-FAID-333	LUT10	LUT10	LUT9	1.110 (μ=2.4)	+0.264	−25.00	−25.00	−13.04
NS-FAID-223	LUT20	LUT17	LUT12	2.154 (μ=2.9)	−0.780	−40.13	−25.00	−13.04
NS-FAID-232	LUT21	LUT7	LUT14	2.080 (μ=2.7)	−0.706	−42.11	−25.00	−13.04
NS-FAID-322	LUT13	LUT17	LUT19	1.667 (μ=3.4)	−0.293	−42.76	−25.00	−13.04
NS-FAID-222	LUT18	LUT16	LUT16	2.299 (μ=2.3)	−0.925	−50.00	−50.00	−26.09

Table 5.4: LUTs used by NS-FAIDs in Table 5.3

m	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21
0	0	0	0	0	0	0	0	0	0	0	0	0	0	±1	±1	±1	±1	±1	±1	±2	±2	±2
1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
2	2	1	2	2	2	1	1	1	1	1	1	1	2	2	1	1	1	1	1	2	2	2
3	3	1	2	2	2	1	1	2	2	3	3	4	2	2	1	1	1	1	5	2	2	2
4	4	3	2	3	4	4	4	2	2	3	3	4	2	5	1	1	5	7	5	2	2	6
5	5	3	4	3	4	4	4	6	7	3	7	4	2	5	6	7	5	7	5	2	7	6
6	6	7	4	7	7	4	7	6	7	7	7	7	2	7	6	7	5	7	5	2	7	6
7	7	7	7	7	7	7	7	6	7	7	7	7	7	7	6	7	5	7	5	7	7	6
w	4	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2

Figure 5.3 captures the trade-off between decoding performance and memory size reduction, for all the 27 NS-FAID-$w_2w_3w_6$ ensembles. The height of vertical bars indicates the VN memory size reduction (values on the left vertical axis), while their color indicates the CN memory size reduction (uncompressed CN-message storage is assumed in the legend). The red stems indicate the SNR threshold gain or loss compared to MS decoder (values on the right vertical axis). It can be seen that the NS-FAID-332 allows a significant memory size reduction for both variable- and check-node messages, while still performing 0.1 dB ahead the MS decoder. The NS-FAID-433 is also a very good candidate for applications requiring increased decoding performance: it achieves the best SNR gain (0.36 dB), while providing a VN memory size reduction by 17.76% with respect to the MS decoder.

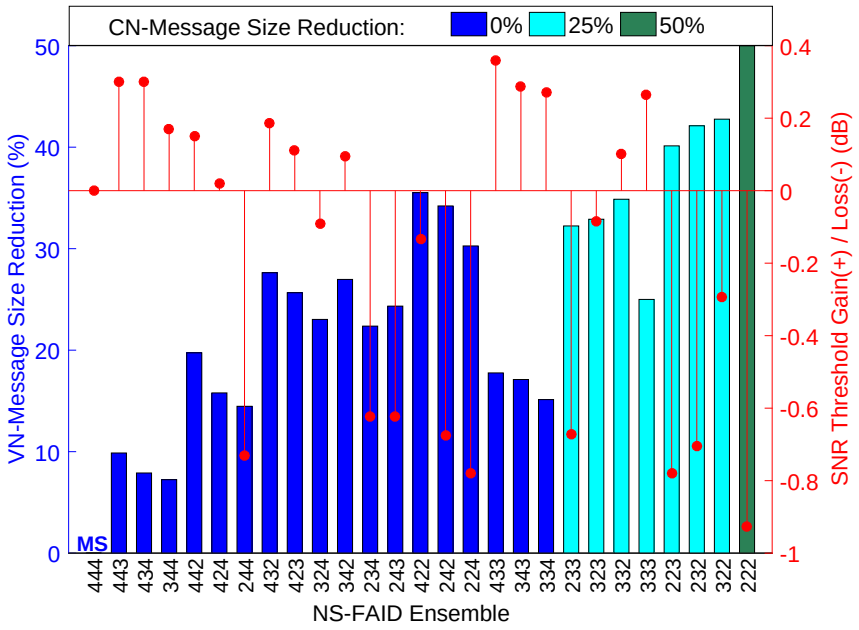


Figure 5.3: Memory size reduction vs. decoding performance

Finally, it is worth noticing that the reported memory size reductions do not necessarily translate as such in hardware implementations, for several reasons. First, depending on the hardware architecture, VN-messages may or may not be stored in a dedicated memory. For instance, layered architectures only require the storage of CN-messages, which can be further stored in either a compressed or uncompressed format. Moreover, VN processing units (VNUs) need to be equipped with a framing and a deframing module, which may offset part of the promised gains. This is even more true in case of irregular codes, for which some VNUs may need to implement more than one single framing function, since the same VNU may be reused to process VN of different degrees (except for fully parallel architectures). To asses the gains of NS-FAIDs in practical implementations, the integration of the NS-FAID mechanism (framing/deframing modules) into two different MS decoder architectures is discussed in the next chapter.

5.4 Conclusion

In this chapter, we introduced the new framework of Non-Surjective FAIDs, which allows trading off decoding performance for hardware complexity reductions. NS-FAIDs have been optimized by density evolution and we showed that they exhibit better or similar decoding performance compared to the MS decoder, while providing significant savings in memory and/or interconnects. While this chapter focused on the theoretical aspects of NS-FAIDs, hardware implementation results will be reported in the next chapter.

Chapter 6

Low-Cost, High-Throughput Hardware Architectures

In this chapter, we propose three QC-LDPC decoder architectures targeting low-cost, high-throughput and efficient use of the hardware resources. All of the proposed architectures implement layered scheduled decoding with fully parallel processing units.

The first architecture is an enhanced version of the architecture introduced in Chapter 4. It is based on a specific design of the datapath processing units (VNUs, CNUs, and AP-LLR units) that allow an efficient reuse of the hardware resources, thus yielding significant cost reduction. In the second architecture, efficient use of the hardware resources is ensured by pipelining the datapath, which also allows increasing the operating frequency, thus increasing the achieved throughput. However, the use of pipelining imposes specific constraints on the decoding layers, in order to ensure proper execution of the layered decoding process. Finally, the third architecture does not make use of pipelining, but allows maximum parallelism to be exploited through the use of full decoding layers, thus resulting in significant increase in throughput.

The second and third architectures may accommodate both MS and NS-FAID decoding kernels. Thus, to validate the promises of the NS-FAID approach (see Chapter 5), both MS and NS-FAID decoding kernels are integrated into each of these architectures, and compared in terms of throughput and resources consumption. Implementation results on both FPGA and ASIC platforms show that regular and irregular NS-FAIDs allow significant improvements in terms of both throughput and hardware resources consumption, as compared to the conventional MS solution, with even better or only slightly degraded decoding performance.

6.1 Introduction

In this chapter, we propose three QC-LDPC decoder architectures targeting (i) flexibility, (ii) high-throughput, and (iii) low cost and efficient use of the hardware resources.

Highest flexibility can be achieved by using serial processing units: VNUs and CNUs process incoming messages in a serial manner, which makes their implementation independent of the variable or check-node degree. However, this comes at the cost of a reduced throughput. Therefore, in order to improve the high-throughput capabilities of the proposed designs, we focus on layered LDPC decoder architectures with fully parallel processing units. Such architectures have some inherent limitations in terms of flexibility, mainly concerning the maximum number of incoming messages into VNUs and CNUs, corresponding to the maximum degrees of variable and check nodes in the Tanner graph.

To increase throughput as well as to ensure an efficient use of hardware resources, the classical solution is to pipeline the datapath. However, for layered architectures, the number of stages in the datapath may impose specific constraints on the base matrix of the QC-LDPC code, in order to ensure that no memory conflicts occur during the read/write operations from/to the memory storing the exchanged messages or the a posteriori logarithmic likelihood ratios (AP-LLR) values. Moreover, pipelined architectures may violate the layered scheduling principle: due to the pipeline depth, it may happen that some of the AP-LLR values used when processing a given layer, might not have been yet updated by the previous layers. This may lead to a significant degradation of the decoding performance (in case the information from some layers is lost) or to a reduction of the convergence speed (because the information propagates slowly). A pipelined architecture which avoids these limitations is proposed and discussed in Section 6.3.1.

In case the datapath is not pipelined, we propose two solutions to make an efficient use of the hardware resources in layered architecture. The first solution, presented in Section 6.2, consists of a specific design of the processing units, so that each layer is processed in two clock cycles, with the same processing units being reused at each clock cycle. Finally, the architecture presented in Section 6.3.2, allows maximum parallelism to be exploited through the use of full decoding layers (*i.e.*, a layer comprised of several rows of the base matrix, such that each column of the base matrix has exactly one non-negative entry within the rows composing the layer) as well as a complete reorganization of the data path, such that each layer can be efficiently processed in only one clock cycle.

Throughout this chapter, we shall use the notations and conventions from Chapter 4. In particular, we recall that $\mathbf{B}$ denotes the base matrix of the QC-LDPC code, of size $R \times C$, and z denotes the expansion factor. A decoding layer may consist of one or several rows of $\mathbf{B}$, assuming that they do not overlap (*i.e.*, they do not have common non-negative entries). The number of rows of $\mathbf{B}$ per decoding layer is denoted by RPL. Hence, the number of decoding layers is given by $L = R/\text{RPL}$.

6.2 Hardware reusing architecture

In this section, we propose a specific design of the datapath processing units (VNUs, CNUs, and AP-LLR units) that allow an efficient reuse of the hardware resources, thus yielding significant cost reduction. Accordingly, the main contribution consists of:

(1) **A low-cost VNU/AP-LLR processing unit** that merges in an efficient way the logical functionalities of the VNU and AP-LLR units, and can be executed by selecting either the VNU or the AP-LLR mode.

(2) **A high-speed, low-cost CNU architecture**, which only computes the first minimum (`min1`) and index of the first minimum (`indx_min1`), instead of first two minima and `indx_min1` as required by the MS decoding algorithm. To compute the second minimum (`min2`), the CNU is executed a second time with `indx_min1` input set to the maximum value (according to the bit-length of the exchanged messages). Due to a specific organization of the datapath, the second execution of the CNU does not induce any penalty in terms of throughput, as explained below.

(3) **A splitting of the iteration processing in two perfectly symmetric stages**, executed in two consecutive clock cycles, each one using exactly the same processing resources; the processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. In the first clock cycle decoder performs read operations, then execute the VNU/AP-LLR unit in VNU mode, and the CNU to compute `min1` and `indx_min1`. In the second clock cycle decoder executes the CNU to compute `min2`, the VNU/AP-LLR unit in AP-LLR mode, and performs write back operations. The processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. In particular, the second execution of the CNU during the second clock cycle does not impose any penalty on the operating clock frequency.

For the sake of simplicity, we shall first assume that all the check-nodes have the same degree, which will be denoted in the sequel by d_c. No further assumptions are made regarding the base matrix $\mathbf{B}$. The case of check-node irregular codes will be discussed in Section 6.2.2.

Figure 6.1 illustrates the baseline architecture of the layered MS decoder. This is essentially the same architecture as in Chapter 4, except that in this chapter we consider that check-to-variable node messages are stored in a *compressed* format (see Figure 4.4, page 71) to reduce memory requirements. We therefore refer to Chapter 4, for the description o the main block of this architecture. Each decoding iteration takes two clock cycles. All data are read and processed at the first rising edge clock, then written at the second rising edge clock.

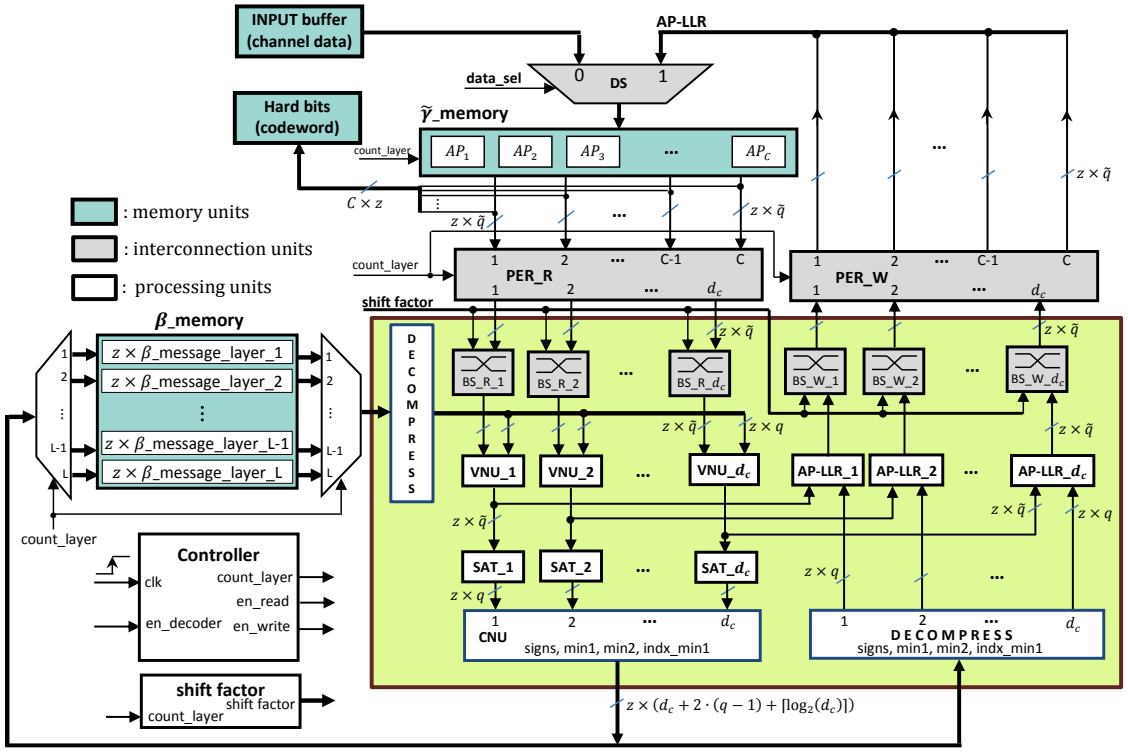


Figure 6.1: Block diagram of the baseline layered MS decoder architecture. Assuming that a decoding layer corresponds to a row of the base matrix (hence RPL $= 1$ and $Z = z$).

6.2.1 Description of the proposed enhancements

We discuss now the main enhancements that we are incorporating into the baseline architecture, which consist of (1) a low-cost VNU/AP-LLR processing unit that merges the logical functionalities of the VNU and AP-LLR units, (2) a low-cost CNU architecture, which is executed twice in order to complete computation of the check-node messages, (3) a splitting of the iteration processing in two perfectly symmetric stages. VNU/AP-LLR unit and the new CNU substitute to the VNU, AP-LLR, and the old CNU units in the baseline architecture, as shown in Figure 6.2 (where VNU/AP-LLR is shortened to VN/AP). All the other blocks of the architecture remain the same.

6.2.1.1 VNU/AP-LLR Unit

The main difference between VNU and AP-LLR processing units is that subtractors are used within the first, while adders are used within the second. We propose a new VNU/AP-LLR processing unit that merges their logical functionalities, controlled by a specific signal (sel) to allow selecting between the VNU or AP-LLR mode. The control signal is generated by the controller, such that VNU mode is selected during the first clock, and AP-LLR mode during the second. The block diagram of the VNU/AP-LLR unit is detailed in Figure 6.3. At the input, two multiplexers

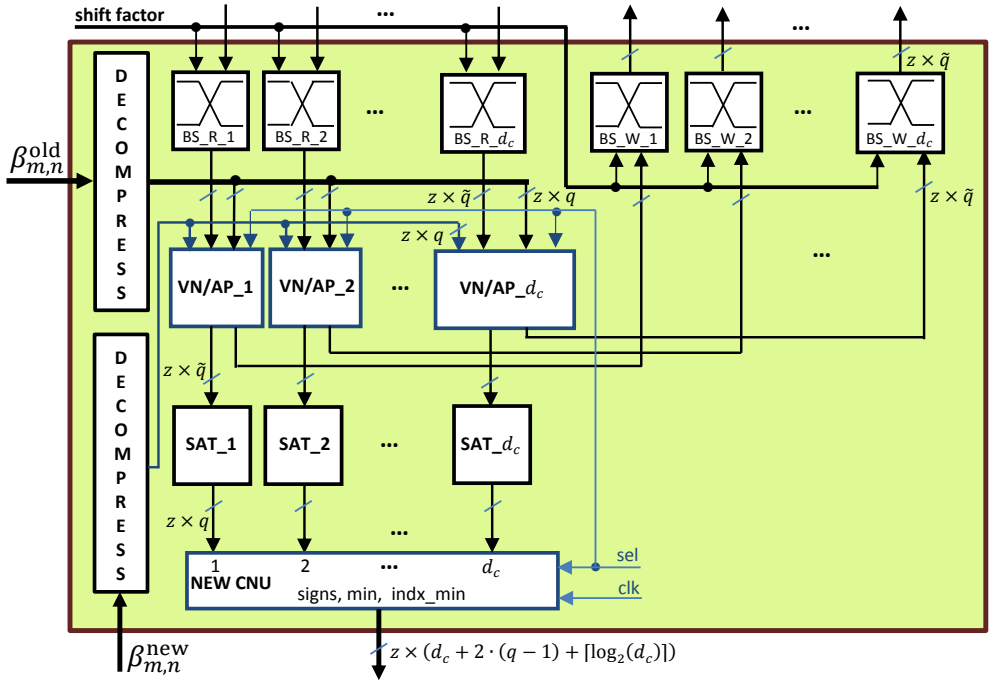


Figure 6.2: New processing units for the layered MS decoder architecture

are used to select the input data according to either the VNU or AP-LLR mode. Similarly, at the output, a de-multiplexer is used to choose the value of either $\alpha_{m,n}$ or $\tilde{\gamma}_n$, depending on the `sel` signal. The block in the middle, which may acts as either a subtractor or an adder is detailed in Figure 6.4 (for the sake of simplicity, we illustrate this block for $\tilde{q} = 4$ bits). It consist of a modified Ripple Carry Adder (RCA) with carry in given by the complement of the sel signal ($C_0 = \overline{\texttt{sel}}$), and which is further XORed to all the bits of the second input. It can be easily seen that the VNU/AP-LLR unit operate in VNU mode if $\texttt{sel} = 0$ ($C_0 = 1$), or in AP-LLR mode if $\texttt{sel} = 1$ ($C_0 = 0$).

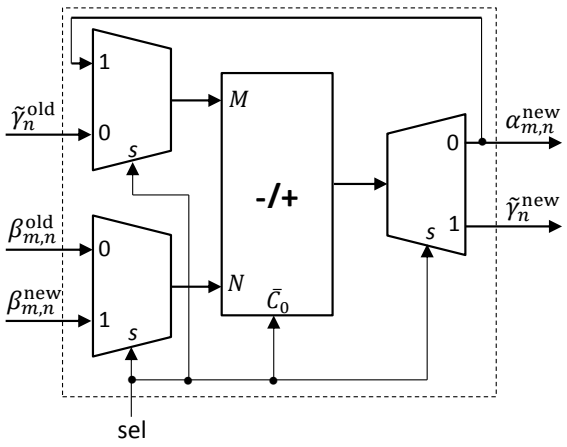


Figure 6.3: Proposed VNU/AP-LLR processing unit

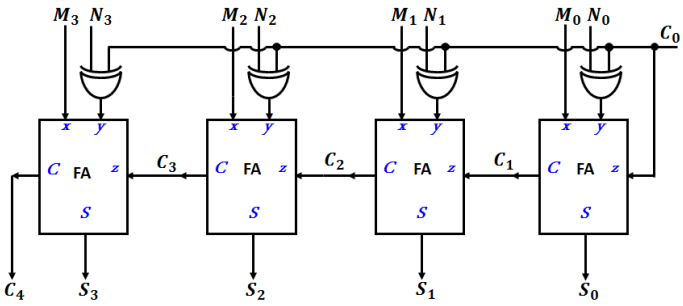


Figure 6.4: Adder/subtractor block used within the VNU/AP-LLR unit

6.2.1.2 CNU Unit

We focus only on the computation of `min1`, `min2`, and `indx_min1`, as the signs of the output messages can be simply computed by XORing the adequate signs of input messages. We propose a high-speed low-cost CNU architecture inspired by the tree structure (TS) architecture proposed in [101], which is further simplified so as to compute only the value and the index of the first minimum. As shown in Figure 6.5, our CNU is executed during the first clock cycle to compute `min1` and `indx_min1`, then it is re-executed during the second clock cycle with `indx_min1` input set to the maximum value, so that to compute `min2`. The `sel` control signal is used to indicate whether the CNU is in first or second minimum mode (first or second clock cycle). The compare and select block is used to set the `indx_min1` input to the maximum value, in case that the `sel` signal indicates that the second minimum is being computed (second clock cycle).

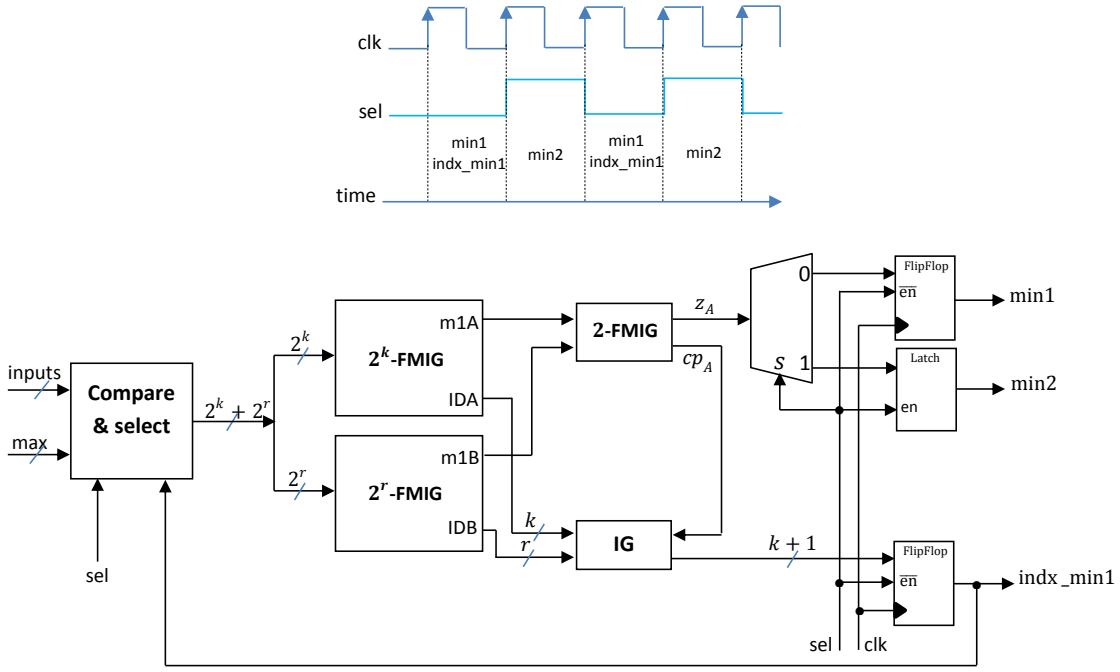


Figure 6.5: Block diagram of the proposed CNU architecture

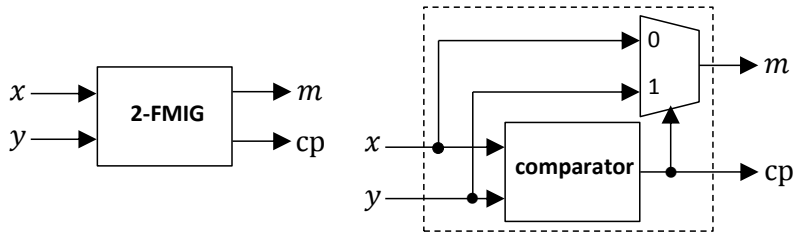


Figure 6.6: 2-FMIG architecture

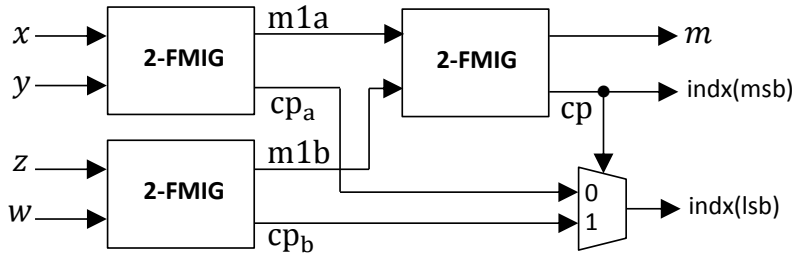


Figure 6.7: 4-FMIG architecture

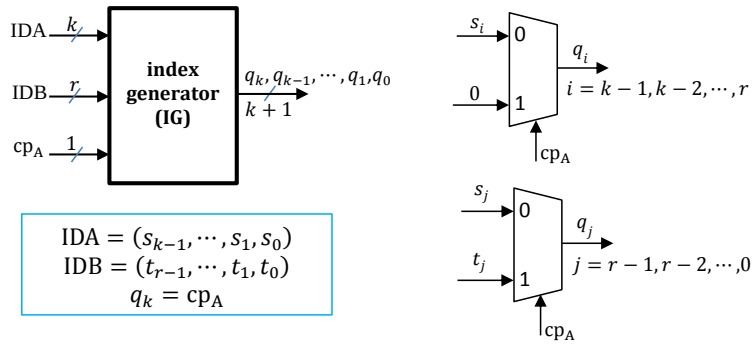


Figure 6.8: IG (Index Generator) architecture

The proposed CNU architecture is detailed in Figure 6.5 for a number of inputs $(2^k + 2^r)$ equal to the sum of two powers of 2. The general case can be worked out by decomposing the number of inputs as a sum of powers of 2, then combining corresponding blocks similarly to the technique used in [101]. The 2^k-FMIG (First Minimum and Index Generator) block computes the value and the index of the first minimum among the 2^k input values. The 2-FMIG block includes one comparator and one multiplexer, as shown in Figure 6.6. The 4-FMIG consists of three 2-FMIG blocks for finding the minimum value and one multiplexer for indicating its index, as shown in Figure 6.7. Similarly, the 2^{k+1}-FMIG block can be constructed from three 2^k-FMIG blocks and one multiplexer. The IG (Index Generator) block in Figure 6.5 is used to determine the index of the minimum value, and is further detailed in Figure 6.8.

6.2.1.3 Layer Processing Split

As shown in Figure 6.2, in the new architecture the clock signal is fed to the CNU. This allows splitting the layer processing in two perfectly symmetric stages, executed in two consecutive clock cycles, each one using the same processing units, but in different mode.

In the first clock cycle decoder performs read operations, then execute the VNU/AP-LLR unit in VNU mode, and the CNU to compute `min1` and `indx_min1`. In the second clock cycle decoder executes the CNU to compute `min2`, the VNU/AP-LLR unit in AP-LLR mode, and performs write back operations. The processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. In particular, the second execution of the CNU during the second clock cycle does not impose any penalty on the operating clock frequency. The baseline CNU (*i.e.*, computing `min1`, `min2`, and `indx_min1`) executed in one of the two clock cycles would lead to an increased critical path, and therefore a reduced clock frequency, while splitting its execution between the two clock cycles would have resulted in an inefficient use of the hardware resources.

6.2.2 Case of Check-Node Irregular Codes

To accommodate QC-LDPC codes with variable check-node degree $d_c \in [d_{\text{cmin}}, d_{\text{cmax}}]$, some extra control logic is required in order to "inactivate" the last $d_{\text{cmax}} - d_c$ VNU/APLLR units as well as the last $d_{\text{cmax}} - d_c$ inputs of the CNU, for check-nodes of degree d_c. If the check-node degree d_c varies between d_{cmin} and d_{cmax}. A VNU/AP-LLR unit is inactivated by setting the corresponding β-inputs to 0, while an input of the CNU is inactivated by setting it to the maximum value ($2^{q-1} - 1$, where q is the number of quantization bits of input $\alpha_{m,n}^{\text{SAT}}$ values, including the sign bit). The modified VNU/AP-LLR and CNU architectures are shown in Figure 6.9 and Figure 6.10, respectively, for $d_{\text{cmin}} = d_{\text{cmax}} - 1$.

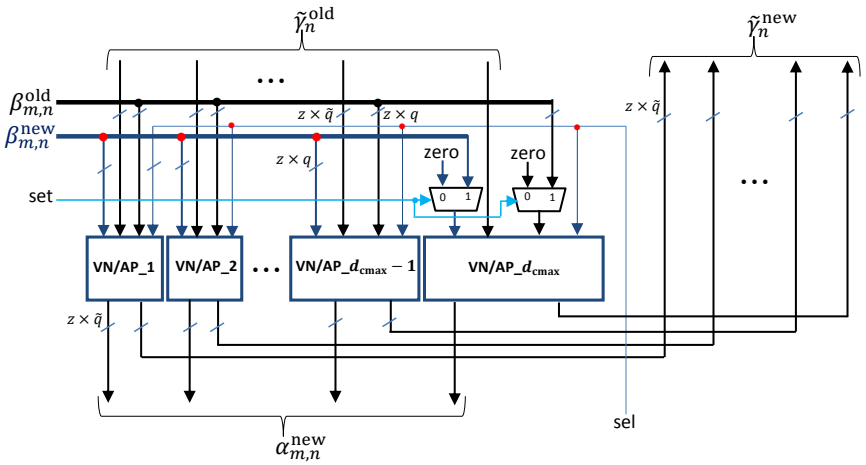


Figure 6.9: Modified VNU/AP-LLR to accommodate variable check-node degree (example for $d_{\text{cmin}} = d_{\text{cmax}} - 1$)

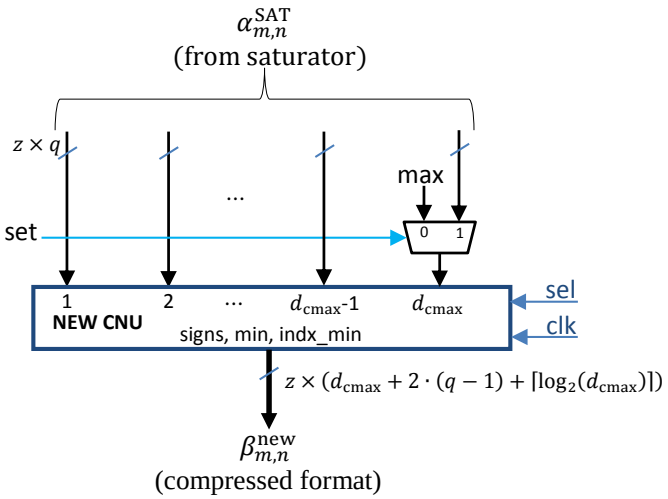


Figure 6.10: Modified CNU to accommodate variable check-node degree (example for $d_{\text{cmin}} = d_{\text{cmax}} - 1$)

6.2.3 Implementation results

We have implemented the baseline and enhanced layered MS decoder architectures for a regular QC-LDPC code with variable-nodes of degree $d_v = 3$, and for the irregular WiMAX QC-LDPC code with rate 1/2 [46]. For both codes, the size of the base is equal to $R \times C = 12 \times 24$. For the regular code, the base matrix $\mathbf{B}$ is shown in Figure 4.1. It can be divided in $L = 3$ horizontal layers, with each layer corresponding to RPL $= 4$ consecutive rows of $\mathbf{B}$. For the WiMAX code, the base matrix $\mathbf{B}$ is shown in Figure 2.4 and the RPL value is set to 1, thus the number of decoding layers is equal to $L = 12$. Configuration parameters of the two decoders are further detailed in Table 6.1.

Table 6.1: Parameters of the QC-LDPC codes

	R	C	z	RPL	d_{cmin}	d_{cmax}	$\tilde{q}$	q	n_{iter}
$(3,6)$-regular	12	24	54	4	6	6	6	4	20
WiMAX	12	24	96	1	6	7	6	4	20

Table 6.2: Comparison between enhanced and baseline architectures for $(3,6)$-regular and WiMAX QC-LDPC codes

	$(3,6)$-regular QC-LDPC		WiMAX QC-LDPC	
	Baseline	Enhanced	Baseline	Enhanced
Max. Freq. (MHz)	111	250	83	250
Throughput (Mbps)	1198	2700	398	1200
Area (mm^2)	0.95	0.72	0.88	0.86
Frequency (MHz)	111	111	83	83
Area (mm^2)	0.95	0.71	0.88	0.76

ASIC synthesis results targeting a 65nm CMOS technology are shown in Table 6.2. The top part of the table reports the maximum operating frequency, the corresponding throughput, and the area. The reported throughput is given by Eq. (4.7). First, we note that the enhanced architecture provides a significant increase in the maximum operating frequency compared to the baseline architecture, by a factor of $\times 2.25$ and $\times 3$, for the $(3, 6)$-regular and the WiMAX code, respectively. This is due to the proposed increased-speed CNU together with the proposed split of the iteration processing. Regarding the area, it can be seen that the enhanced architecture provides a significant area reduction for the $(3, 6)$-regular code, by 24.2% compared to the baseline architecture. However, the area reduction is of only 2.27% for the WiMAX code.

In oder to keep the area comparison on an equal basis with respect to synthesis timing constraints, in the bottom part of Table 6.2 we report area figures when the same timing constraints are applied to both the baseline and the enhanced architecture. We consider timing constrains corresponding to the maximum operating frequency for the baseline architecture. In this case, it can be seen that the proposed cost-efficient VNU/AP-LLR and CNU processing units yield an area reduction by 25.26% for the $(3, 6)$-regular code, and by 13.64% for the WiMAX code.

To keep the throughput comparison on an equal basis with respect to technology, area, and number of iterations. We also report the *Throughput to Area Ratio* (TAR) and the *Normalized TAR* (NTAR) metrics, which are defined by Eq. (6.1) and Eq. (6.2), respectively. The TAR metric is very useful to evaluate how much throughput a decoder can achieve per area unit on ASIC.

$$\text{TAR} = \frac{\text{Throughput}}{\text{Area}} \tag{6.1}$$

$$\text{NTAR} = \text{TAR} \times \text{Iterations} \tag{6.2}$$

Table 6.3: Comparison between the proposed enhanced architecture and state-of-the-art implementations for the WiMAX QC-LDPC code

Author/Year	**Y. Ueng'08**	**K. Zhang'09**	**T. Heidari'13**	**K. Kanchetla'16**	**Proposed**
Paper	[95]	[110]	[39]	[51]	**decoder**
Code length	2304	2304	2304	576-2304	2304
Technology (nm)	180	90	130	90	65
Frequency (MHz)	200	950	100	149	250
Iterations	4.6 (avarage)	10	10	5-16	20
Throughput (Mbps)	106	2200	183	955	1200
Tput. scaled to 65nm (Mbps)	294	3036	366	1318	1200
Area (mm^2)	-	$2.90^{(*)}$	$6.90^{(**)}$	$11.42^{(*)}$	$0.86^{(*)}$
Area scaled to 65nm (mm^2)	-	$1.51^{(*)}$	$1.73^{(**)}$	$5.94^{(*)}$	$0.86^{(*)}$
TAR (Mbps/mm^2)	-	2010.60	211.56	221.89	1395.35
NTAR (Mbps/mm^2/iter)	-	20106	2115.6	1109.45	27907

$^{(*)}$ only core area is reported
$^{(**)}$ total chip area is reported
TAR = (Throughput scaled to 65nm) / (Area scaled to 65nm)
NTAR = TAR × Iterations

Finally, the proposed enhanced architecture is further compared with other state-of-the-art implementations in Table 6.3 for WiMAX QC-LDPC code. We mention that the decoder proposed in [51] is a reconfigurable decoder that supports the IEEE 802.16e (WiMAX) and and the IEEE 802.11n (WiFi) wireless standards. The reported throughput is the maximum achievable coded throughput for the $(1152, 2304)$ WiMAX code with 5 decoding iterations. To scale throughput and area to 65nm, we use scale factors (technology size/65) and $(65/\text{technology size})^2$, as suggested in [38]. Note that for all the reported implementations, the achieved throughput is inversely proportional to the number of iterations, hence the NTAR metric corresponds to the TAR value assuming that only one decoding iteration is performed. From Table 6.3 it can be seen that the proposed enhanced architecture compares favorably with state-of-the-art implementations, yielding a NTAR value of 27.9 Gbps/mm^2/iteration.

6.3 Hardware architectures with MS and NS-FAID kernels

NS-FAD decoders, introduced in Chapter 5, allow storing the exchanged messages using a lower precision than the one used by the processing units. They are expected to facilitate significant reductions of the memory and interconnect blocks, with no or only slight degradation of the error correction performance, as compared to the conventional MS decoder. Since these blocks usually dominate the overall performance of the hardware implementation, NS-FAIDs are seen as a promising approach to further optimizations of cost-effective, high-throughput designs.

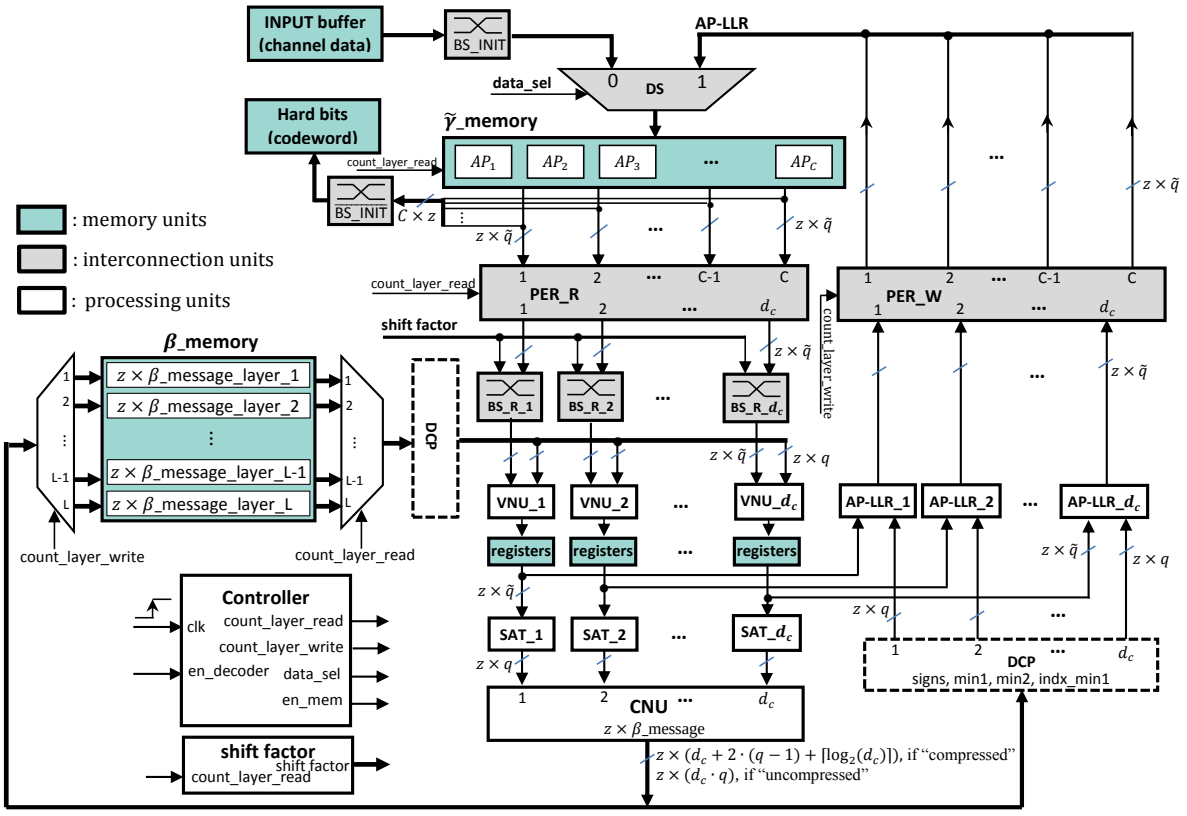
To validate the promises of the NS-FAID approach, in this section we propose two different hardware architectures, integrating both MS and NS-FAID decoding kernels. The first architecture is pipelined, so as to increase throughput and ensure an efficient use of the hardware resources, which in turn imposes specific constraints on the decoding layers, in order to ensure proper execution of the layered decoding process. The second architecture does not make use of pipelining, but allows maximum parallelism to be exploited through the use of full decoding layers, thus resulting in significant increase in throughput. Both MS and NS-FAID decoding kernels are integrated to each of the two proposed architectures, and compared in terms of throughput and resource consumption.

To ease the description of the hardware architectures proposed in this section, we shall assume that all CNs have the same degree, denoted by d_c (however, we note that the case if check-node irregular codes can be addressed similarly as in Section 6.2.2). No assumptions are made concerning VN degrees. We present each architecture assuming the MS decoding kernel is being implemented, then we discuss the required changes in order to integrate regular and irregular NS-FAID decoding kernels.

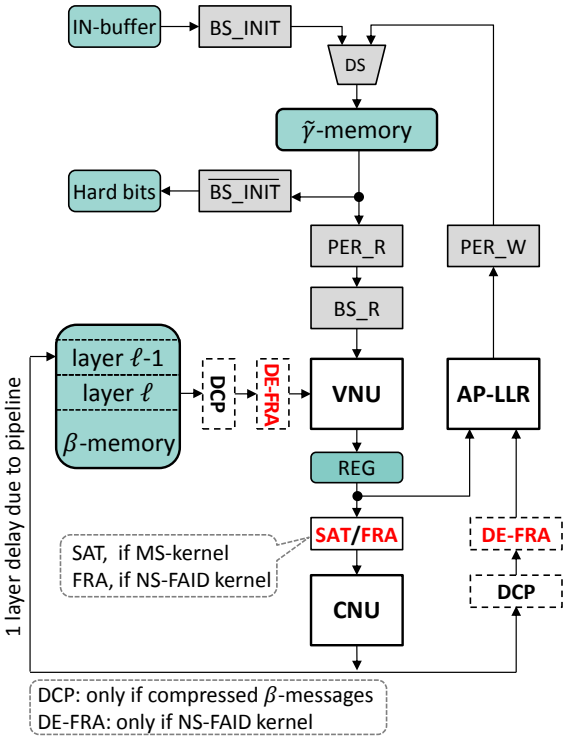
6.3.1 Pipelined architecture

Proposed architecture with MS decoding kernel is detailed in Figure 6.11(a). A high-level representation is also shown in Figure 6.11(b), for both MS and NS-FAID decoding kernels. The main blocks of the architecture are discussed below. For the remaining blocks such as VNU, CNU, AP-LLR, see the description from Chapter 4.

Memory blocks. Two memory blocks are used, one for AP-LLR values ($\tilde{\gamma}$_memory) and one for CN-messages (β_memory). $\tilde{\gamma}$_memory is implemented by registers, in order to allow massively parallel read or write operations. It is organized in C blocks, denoted by AP_j ($j = 1, \cdots, C$), corresponding to the columns of base matrix, each one consisting of $z \times \tilde{q}$ bits. Data are read from/write to blocks corresponding to non-negative entries in the decoding layer being processed. β_memory is implemented as a dual port Random Access Memory (RAM), in order to support pipelining. Each memory word consists of $Z \times \beta$_message, corresponding to one decoding layer. As discussed in Chapter 4, depending on how CNUs are implemented, β_message can be stored into β_memory either in "uncompressed" or "compressed" format.



(a) Detailed pipelined hardware architecture with MS-kernel



(b) High-level description of the hardware architecture with MS and NS-FAID kernels

Figure 6.11: Block diagram of the proposed pipelined architecture. Assuming that a decoding layer corresponds to a row of the base matrix (hence RPL $= 1$ and $Z = z$).

Barrel Shifters (BS_INIT, BS_R). Barrel shifters are used to implement the cyclic (shift) permutations, according to the non-negative entries of the base matrix. The $\tilde{\gamma}$_memory is initialized from the input LLR values stored in the input buffer. However, input LLR values are shifted by BS_INIT block before being written to the $\tilde{\gamma}$_memory, according to the *last* non-negative shift factor on the corresponding base matrix column. BS_R blocks are then used to shift the LLR values read from the $\tilde{\gamma}$_memory, such that to properly align them with the appropriate VNU. Note that there are d_c BS_R_$\{1, \ldots, d_c\}$ blocks. In case RPL $= 1$, a decoding layer corresponds to a row of $\mathbf{B}$, and each BS_R block is used to shift the LLR values within one of the d_c columns with non-negative entries in the current row. Let i be the index of the current row. The cyclic shift implemented by a BS_R block, corresponding to a column j with $b_{i,j} \geq 0$, is given by $-b_{i',j} + b_{i,j}$, where $b_{i',j}$ is the previous non-negative entry in column j (*i.e.*, the previous row i' with $b_{i',j} \geq 0$). In case RPL > 1, each BS_R block actually consists of RPL sub-blocks as above, with one sub-block for each row in the layer. The values of the cyclic shifts are computed offline for each layer ℓ. This avoids the use of barrel shifters when the data is written back to the $\tilde{\gamma}$_memory, thus reducing the critical path of the design. Finally, the $\overline{\text{BS\_INIT}}$ block operates oppositely to BS_INIT, and is used to shift back the hard decision bits into appropriate positions.

Decompress (DCP). This block is only used in case that the CN-messages are in compressed format (signs, min1, min2, indx_min1). It converts the β_messages from compressed to the uncompressed format.

Controller. This block generates control signals such as *en_mem* to enable data reading and writing, *count_layer_read*, *count_layer_write* to indicate which layers are being processed, etc. It also controls the synchronous execution of the other blocks.

Pipelining. To increase the operating frequency, the data path is pipelined by adding a set of registers after the VNU-blocks. Hence, processing one layer takes 2 clock cycles, but at each clock cycle the two pipeline stages work on two consecutive layers of the base matrix. This imposes specific constraints on the base matrix, as consecutive layers must not overlap, in order to avoid $\tilde{\gamma}$_memory conflicts (note that memory stall cycles would cancel the pipelining effect). An example of $d_c = 6$ regular base matrix without overlap between consecutive layers is given in Figure 4.1, assuming that each layer corresponds to a row of the base matrix.

6.3.1.1 Regular NS-FAID kernel

The changes required to integrate the NS-FAID decoding kernel are shown in Figure 6.11(b). First, the Saturation (SAT) block used within the MS-decoding kernel is replaced by a Framing (FRA) block. Note that the output of the VNU consists of $\tilde{q}$-bit (unsaturated) VN-messages. Hence, the FRA block actually implements the concatenation of the following operations:

$$[-\tilde{Q},\ldots,+Q] \xrightarrow{s_{\mathcal{M}}} [-Q,\ldots,+Q] \xrightarrow{F} \mathrm{Im}(F) \xrightarrow{\sim} [-W,\ldots,+W], \tag{6.3}$$

where $[-\tilde{Q},\ldots,+\tilde{Q}]$ is the alphabet of unsaturated messages ($\tilde{Q} = 2^{\tilde{q}-1}-1$), F is the framing function being used, $\mathrm{Im}(F)$ is the image of F (which is a subset of $[-Q,\ldots,+Q]$ according to the framing function definition), and the last operation consists of a re-quantization of the $\mathrm{Im}(F)$ values on a number of w-bit where $w = \lceil \log_2(W) \rceil + 1$ is the framing bit-length.

The De-framing (DE-FRA) block simply converts back from w-bit to q-bit values, *i.e.*, it inverts the re-quantization operation above.

$$[-W,\ldots,+W] \xrightarrow{\sim} \mathrm{Im}(F) \subset [-Q,\ldots,+Q] \tag{6.4}$$

Although we have to add the de-framing blocks, the reduction of the CN-messages size may still save significant hardware resources, as compared to MS decoding. This will be discussed in more details in Section 6.3.3.

6.3.1.2 Irregular NS-FAID kernel

We consider the irregular WiMAX QC-LDPC code of rate 1/2, for which irregular NS-FAIDs have been optimized in Chapter 5. First, we note that the pipeline architecture proposed in this section can be applied to the WiMAX QC-LDPC code, by assuming that each decoding layer consists of one row of the base matrix. Indeed, it is known that for this code, the rows of the base matrix can be reordered, such that any two consecutive rows do not overlap [112]. The base matrix with reordered rows is shown in Figure 6.12. Parameters a (b) on the leftmost column show the row index after (before) reordering.

Regarding the integration of an irregular NS-FAID decoding kernel, the same framing (FRA) or de-framing (DE-FRA) block is reused for several VNs, which may be of different degrees. This may require several framing functions to be implemented within the FRA/DE-FRA blocks, thus increasing the hardware complexity. To overcome this problem, one may change the way the VNs are mapped to the processing units, by reordering the columns of the base matrix processed within each decoding layer. We determine offline such a reordering for each decoding layer, so as to minimize the number of FRA/DE-FRA blocks implementing more than one single framing function. Hence, the PER_R and PER_W blocks, which ensure the proper alignment between data and processing units, are redefined accordingly.

	$v1$	$v2$	$v3$	$v4$	$v5$	$v6$	$v7$	$v8$	$v9$	$v10$	$v11$	$v12$	$v13$	$v14$	$v15$	$v16$	$v17$	$v18$	$v19$	$v20$	$v21$	$v22$	$v23$	$v24$
1 (01)	-1	94	73	-1	-1	-1	-1	-1	55	83	-1	-1	7	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2 (03)	-1	-1	-1	24	22	81	-1	33	-1	-1	-1	0	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
3 (05)	-1	-1	39	-1	-1	-1	84	-1	-1	41	72	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
4 (12)	43	-1	-1	-1	-1	66	-1	41	-1	-1	-1	26	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0
5 (07)	-1	-1	95	53	-1	-1	-1	-1	-1	14	18	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1
6 (09)	12	-1	-1	-1	83	24	-1	43	-1	-1	-1	51	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1
7 (11)	-1	-1	7	65	-1	-1	-1	-1	39	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0
8 (02)	-1	27	-1	-1	-1	22	79	9	-1	-1	-1	12	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
9 (04)	61	-1	47	-1	-1	-1	-1	-1	65	25	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1
10 (06)	-1	-1	-1	-1	46	40	-1	82	-1	-1	-1	79	0	-1	-1	-1	-1	0	0	-1	-1	-1	-1	-1
11 (08)	-1	11	73	-1	-1	-1	2	-1	-1	47	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1
12 (10)	-1	-1	-1	-1	-1	94	-1	59	-1	-1	70	72	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1

Figure 6.12: Modified base matrix of the irregular WiMAX code, rate of 1/2, with rows reordered [112]. In black: variable nodes of degree 2, in red: variable-nodes of degree 3, in blue: variable-nodes of degree 6.

The optimal mapping between variable-nodes and VNUs is shown in Figure 6.13(b), for the base matrix with reordered rows from Figure 6.12. For each VNU, we indicate the index of the variable-node processed by the VNU within each decoding layer. For comparison purposes, we also provide in Figure 6.13(a) the conventional (unoptimized) mapping, which assumes that VNU_k processes the variable-node corresponding to the k-th non-negative entry in the layer (row of $\mathbf{B}$). The last row of each table indicates the number of framing functions that have to be implemented within the FRA/DE-FRA blocks corresponding to each VNU.

	VNU1	VNU2	VNU3	VNU4	VNU5	VNU6	VNU7
Layer 1	$v2$	$v3$	$v9$	$v10$	$v13$	$v14$	–
Layer 2	$v4$	$v5$	$v6$	$v8$	$v12$	$v15$	$v16$
Layer 3	$v3$	$v7$	$v10$	$v11$	$v17$	$v18$	–
Layer 4	$v1$	$v6$	$v8$	$v12$	$v13$	$v24$	–
Layer 5	$v3$	$v4$	$v10$	$v11$	$v19$	$v20$	–
Layer 6	$v1$	$v5$	$v6$	$v8$	$v12$	$v21$	$v22$
Layer 7	$v3$	$v4$	$v9$	$v10$	$v23$	$v24$	–
Layer 8	$v2$	$v6$	$v7$	$v8$	$v12$	$v14$	$v15$
Layer 9	$v1$	$v3$	$v9$	$v19$	$v16$	$v17$	–
Layer 10	$v5$	$v6$	$v8$	$v12$	$v13$	$v18$	$v19$
Layer 11	$v2$	$v3$	$v7$	$v10$	$v20$	$v21$	–
Layer 12	$v6$	$v8$	$v11$	$v12$	$v22$	$v23$	–
No. FRAs	2	2	2	2	3	1	1

(a) Conventional (unoptimized) mapping

	VNU1	VNU2	VNU3	VNU4	VNU5	VNU6	VNU7
Layer 1	$v2$	$v13$	$v9$	$v10$	$v3$	$v14$	–
Layer 2	$v4$	$v6$	$v5$	$v8$	$v12$	$v15$	$v16$
Layer 3	$v7$	$v3$	$v11$	$v10$	$v17$	$v18$	–
Layer 4	$v1$	$v6$	$v13$	$v12$	$v8$	$v24$	–
Layer 5	$v4$	$v3$	$v11$	$v10$	$v19$	$v20$	–
Layer 6	$v1$	$v6$	$v5$	$v8$	$v12$	$v21$	$v22$
Layer 7	$v4$	$v3$	$v9$	$v10$	$v23$	$v24$	–
Layer 8	$v2$	$v6$	$v7$	$v8$	$v12$	$v14$	$v15$
Layer 9	$v1$	$v3$	$v9$	$v19$	$v16$	$v17$	–
Layer 10	$v5$	$v6$	$v13$	$v12$	$v8$	$v18$	$v19$
Layer 11	$v2$	$v3$	$v7$	$v10$	$v20$	$v21$	–
Layer 12	$v6$	$v8$	$v11$	$v12$	$v22$	$v23$	–
No. FRAs	2	2	1	1	2	1	1

(b) Optimized mapping

Figure 6.13: Mapping between variable-nodes and VNUs. In black: variable nodes of degree 2, in red: variable-nodes of degree 3, in blue: variable-nodes of degree 6.

6.3.2 Full layers architecture

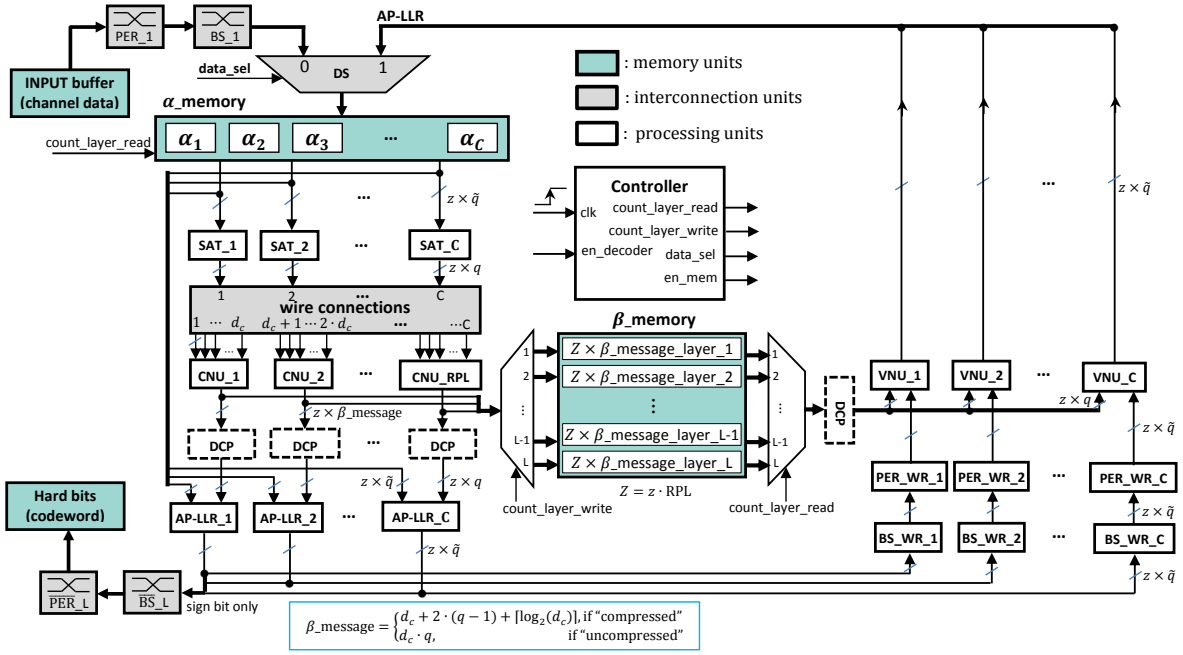
A different possibility to increase throughput is to increase the hardware parallelism, by including several non-overlapping rows of the base matrix in one decoding layer. For instance, for the base matrix in Figure 4.1, we may consider $\text{RPL} = 4$ consecutive rows per decoding layer, thus the number of decoding layers is $L = 3$. In this case, each column of the base matrix has one (and only one) non-zero entry in each decoding layer; such a decoding layer is referred to as being *full*. Full layers correspond to the maximum hardware parallelism that can be exploited by layered

architectures, but they also prevent the pipelining of the data path. Proposed full layers architecture with MS decoding kernel is detailed in Figure 6.14(a). A high-level representation is also shown in Figure 6.14(b), for both MS and regular NS-FAID decoding kernels. The architecture proposed in this section is aimed at providing an effective way to benefit from the increased hardware parallelism enabled by the use of full layers.

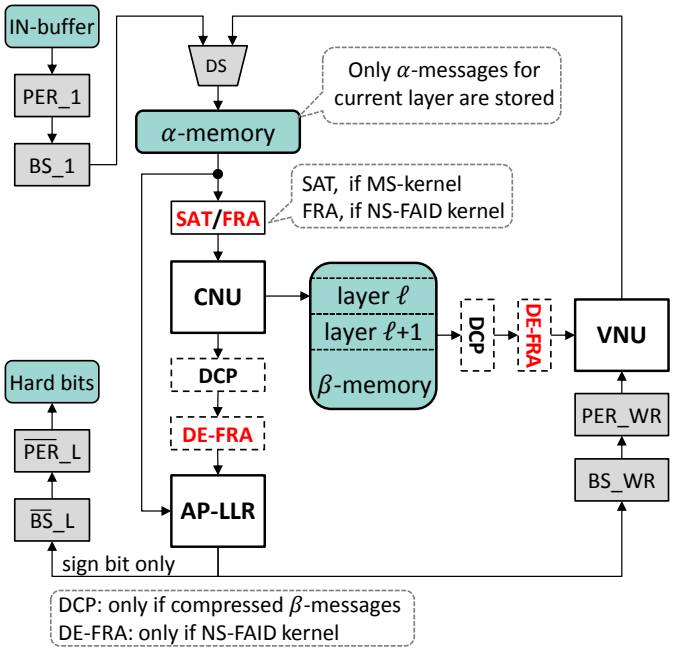
We discuss below the main changes with respect to the pipelined architecture from the previous section, consisting of the α_memory and the barrel shifters blocks (the other blocks are the same as for the pipelined architecture) as well as a complete reorganization of the data path. However, it can be easily verified that both architectures are logically equivalent, *i.e.*, they both implement the same decoding algorithm.

α_memory. This memory is used to store the VN-messages for the current decoding layer (unlike the previous architecture, the AP-LLR values are not stored in memory). Since only one $\tilde{q}$-bit (unsaturated) VN-message is stored for each variable-node, this memory has exactly the same size as the $\tilde{\gamma}$_memory used within the previous pipelined architecture. VN-messages for current layer ℓ are read from the α_memory, then saturated or framed depending on the decoding kernel, and supplied to the corresponding CNUs. CN-messages computed by the CNUs are stored in the β_memory (location corresponding to layer ℓ), and also forwarded to the AP-LLR unit, through the DCP (decompress) and DE-FRA (de-framing) blocks, according to the CNU implementation (compressed or uncompressed) and the decoding kernel (MS of NS-FAID). The AP-LLR unit computes the sum of the incoming VN- and CN-messages, which corresponds to the AP-LLR value to be used at layer $\ell + 1$ (since already updated by layer ℓ). The AP-LLR value is forwarded to the VNU, through corresponding BS and PER blocks. Eventually, the VN-message for layer $\ell + 1$ is computed as the difference between the incoming AP-LLR and the corresponding layer-$(\ell + 1)$ CN-message computed at the previous iteration, the latter being read from the β_memory.

PER / BS blocks. PER_1 / BS_1 blocks permute / shift the data read from the input buffer, according to the positions / values of the non-negative entries in the first decoding layer. Similarly to the BS_R blocks in the pipelined architecture, the PER_WR / BS_WR blocks permute / shift the AP-LLR values, according to the *difference* between the positions / values of the current layer's (ℓ) non-negative entries and those of the next layer ($\ell + 1$). This way, VN-messages stored in the α_memory are already permuted and shifted for the subsequent decoding layer. Finally, PER_L / BS_L blocks permute / shift the hard decision bits (sign of AP-LLR values), according to the positions / values of the non-negative entries in the last decoding layer.



(a) Detailed full layers hardware architecture with MS-kernel



(b) High-level description of the hardware architecture with MS and NS-FAID kernels

Figure 6.14: Proposed full layers architecture with MS and NS-FAID kernels

6.3.3 Implementation results

This section reports implementation results (cost, throughput, etc.) on both FPGA and ASIC platforms as well as the error correction performance of the implemented codes, in order to corroborate the analytic results obtained in Chapter 5. As mentioned in the previous section, both architectures are logically equivalent, thus they both yield the same decoding performance (assuming that they implement the same MS/NS-FAID decoding kernel), but they may have different performance in terms of area and throughput).

Throughput reported in this section is given by the formula:

$$\text{Throughput} = \begin{cases} \frac{N \times f_{\max}}{L \times n_{\text{iter}}}, & \text{full layers architecture} \\ \frac{N \times f_{\max}}{1 + L \times n_{\text{iter}}}, & \text{pipelined architecture} \end{cases} \tag{6.5}$$

where N is the codeword length, $f_{\max}$ is the maximum operating frequency, L is the number of decoding layers, and n_{iter} is the number of decoding iterations (set to 20).

To keep the throughput comparison on an equal basis, for ASIC designs, we further define the Throughput to Area ratio (TAR) metric as:

$$\text{TAR} = \frac{\text{Throughput}}{\text{Area}} \, (\text{Mbps/mm}^2) \tag{6.6}$$

Similarly, for FPGA designs, we define the *Hardware Usage Efficiency* (HUE) metric, as being the throughput corresponding to 100% utilization of the hardware resources:

$$\text{HUE} = \frac{\text{Throughput}}{\% \, \text{Occupied slices}} \, (\text{Mbps}) \tag{6.7}$$

6.3.3.1 Regular NS-FAIDs

We consider the $(3, 6)$-regular QC-LDPC code with base matrix $\mathbf{B}$ of size $R \times C = 12 \times 24$, shown in Figure 4.1. The expansion factor $z = 54$, thus the codeword length is $N = zC = 1296$ bits. The base matrix can be divided in either $L = 12$ decoding layers (RPL $= 1$), for the pipelined architecture, or $L = 3$ horizontal decoding layers (RPL $= 4$), for the full-layer architecture.

Figure 6.15 shows the Bit-Error Rate (BER) performance of the MS decoder with quantization parameters $(q, \tilde{q}) = (4, 6)$ as well as $q = 4$-bit NS-FAIDs with $w = 2$ and $w = 3$ (framing functions F corresponding to w and $F(0)$ values in the legend are those in Chapter 5). Binary input AWGN channel model is considered, with 20 decoding iterations. It can be seen that the simulation results corroborate the analytic results in Chapter 5, in terms of SNR gain / loss provided by NS-FAIDs, as compared to MS. For comparison purposes, we have further included simulations results for the Offset MS (OMS) decoder with $(4, 6)$-quantization and offset factor $= 1$ as well as the MS decoder with $(3, 5)$ and $(2, 4)$-quantization.

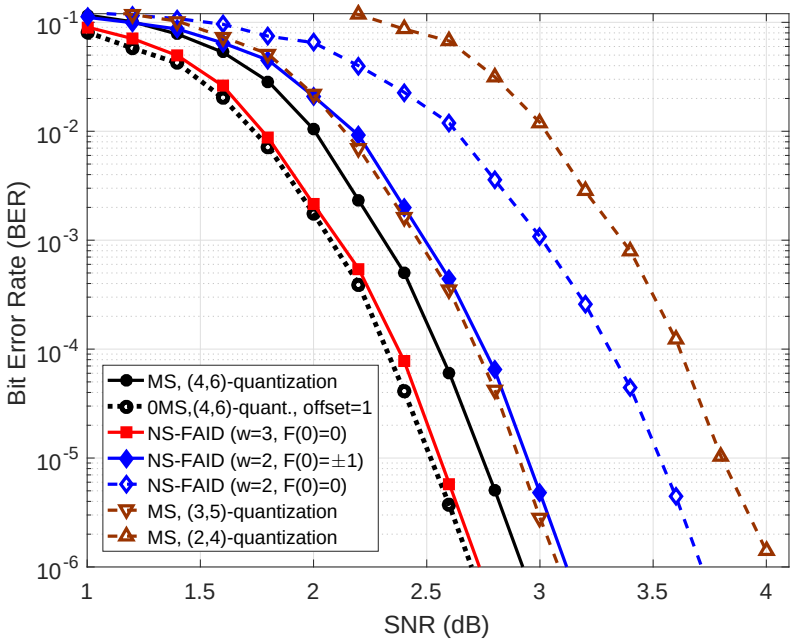


Figure 6.15: BER performance of optimized NS-FAIDs for (3,6)-regular LDPC code

FPGA Implementation results

FPGA implementation (post place-and-route) results on Xilinx Zynq-7000 FPGA device are shown in Table 6.4, for the MS$(4,6)$ decoder and the NS-FAIDs with $(w=3, F(0)=0)$ and $(w=2, F(0)=\pm 1)$, indicated in the table as NS-FAID-3 and NS-FAID-2, respectively. The rows in Table 6.4, named by "Variant", indicate the architecture (pipelined or full layers) and the CNU type (compressed or uncompressed). We also note that for the NS-FAID-2, the assumption that 0 is mapped to either -1 or $+1$, with equal probability, is only needed for theoretical analysis (the symmetry of the decoder allows reducing the analysis to the all-zero codeword). However, in practical situations one may always map 0 to $+1$, since random codewords are transmitted (for instance, in telecommunications systems, pseudo-randomness of the transmitted data is ensured by a scrambling mechanism).

While the NS-FAID-3 outperforms the baseline MS$(4,6)$ decoder by 0.19 dB at BER $= 10^{-5}$ (Figure 6.15), it can be seen from Table 6.4 that it also exhibits a HUE improvement between 8.28% and 42.53%, depending on the hardware architecture and CNU type. As predicted, the NS-FAID-2 exhibits a performance loss of 0.21 dB compared to MS$(4,6)$, but yields a significant HUE improvement, by 35.72% to 57.78%.

To further emphasize the high-throughput characteristic of the proposed architectures, Table 6.5 provides a comparison between state-of-the-art FPGA implementations of $(3,6)$-regular LDPC decoders and the uncompressed full layers architecture with NS-FAID-3 decoding kernel, presented in this work. Our implementation achieves a significantly increased throughput, which has also to be reported to the number of decoding iterations.

Table 6.4: FPGA Post-PAR Implementation Results on Zynq-7000 (XC7Z045FFG900-1)

Variant	**pipeline.uncompressed**			**pipeline.compressed**		
Decoder	MS(4,6)	NS-FAID-3	NS-FAID-2	MS(4,6)	NS-FAID-3	NS-FAID-2
No. occupied slices (% utilization)	9776 (17.89)	9204 (16.84)	8844 (16.18)	10163 (18.60)	10091 (18.46)	9617 (17.60)
Max. freq (MHz)	98	108	138	95	102	125
No. layers (L)	12	12	12	12	12	12
Throughput (Mbps)	527	580	742	510	548	672
HUE (Mbps)	2945	3444	4585	2741	2968	3818
±% w.r.t. MS(4,6)	0%	+16.94%	+55.69%	0%	+8.28%	+39.29%

Variant	**full_layers.uncompressed**			**full_layers.compressed**		
Decoder	MS(4,6)	NS-FAID-3	NS-FAID-2	MS(4,6)	NS-FAID-3	NS-FAID-2
No. occupied slices (% utilization)	17578 (32.16)	16812 (30.76)	15018 (27.48)	21317 (39.01)	18313 (33.51)	17712 (32.41)
Max. freq (MHz)	69	75	80	58	71	76
No. layers (L)	3	3	3	3	3	3
Throughput (Mbps)	1490	1620	1728	1252	1533	1641
HUE (Mbps)	4633	5266	6288	3209	4574	5063
±% w.r.t. MS(4,6)	0%	+13.66%	+35.72%	0%	+42.53%	+57.78%

Table 6.5: Comparison of FPGA implementations for $(3, 6)$-regular LDPC codes

Author/Year Paper	**Karkooti'04 [52]**	**Chen'11 [18]**	**Vikram'15 [14]**	**This work NS-FAID-3**
Device	Virtex 2	Virtex 2	Virtex 5	Zynq-7000
Codeword length	1536	1536	2304	1296
No. occupied slices	11352	6102	8430	16812
No. BRAMs	66	24	232	0
No. iterations	20	3 (avg)	8 (avg)	20
Max. freq (MHz)	121	149.8	114	75
Throughput (Mbps)	127	830.6 (avg)	1096 (avg)	1620

ASIC Implementation results

ASIC post-synthesis implementation results on 65nm-CMOS technology are shown in Table 6.6. TAR results and corresponding gain/loss (+/-) with respect to the MS(4,6) decoder are also reported. It can be seen from Table 6.6 that NS-FAID-3 exhibits a TAR improvement between 18.88% and 31.61%, depending on the hardware architecture and CNU type. The NS-FAID-2 exhibits a performance loss of 0.21 dB compared to MS$(4, 6)$, but yields a significant TAR improvement, by 34.37% to 58.75%.

Table 6.6: ASIC post-synthesis implementation results on 65nm-CMOS technology for optimized $(3, 6)$ regular NS-FAIDs

Variant	**pipeline.uncompressed**			**pipeline.compressed**		
Decoder	MS(4,6)	NS-FAID-3	NS-FAID-2	MS(4,6)	NS-FAID-3	NS-FAID-2
Frequency (MHz)	200	222	227	175	200	208
Throughput (Mbps)	1075	1193	1220	941	1075	1118
Area (mm^2)	0.45	0.42	0.38	0.41	0.38	0.36
TAR ($Mbps/mm^2$)	2389	2840	3210	2295	2828	3105
±% w.r.t. MS(4,6)	0%	+18.88%	+34.37%	0%	+23.22%	+35.29%

Variant	**full_layers.uncompressed**			**full_layers.compressed**		
Decoder	MS(4,6)	NS-FAID-3	NS-FAID-2	MS(4,6)	NS-FAID-3	NS-FAID-2
Frequency (MHz)	151	172	192	125	147	172
Throughput (Mbps)	3261	3715	4147	2700	3175	3715
Area (mm^2)	0.80	0.72	0.68	0.75	0.67	0.65
TAR ($Mbps/mm^2$)	4076	5159	6098	3600	4738	5715
±% w.r.t. MS(4,6)	0%	+26.57%	+49.61%	0%	+31.61%	+58.75%

6.3.3.2 Irregular NS-FAIDs

We consider the irregular WiMAX QC-LDPC code of rate 1/2, with base matrix of size $R \times C = 12 \times 24$ is given in Figure 6.12. The expansion factor $z = 96$, thus resulting in a codeword length $N = zC = 2304$ bits. The pipelined architecture from Section 6.3.1 is implemented, with RPL $= 1$ row per decoding layer, after reordering the rows of the base matrix, such that any two consecutive rows do not overlap [112]. Note that the full layers architecture does not apply to irregular WiMAX LDPC codes, since it is not possible to group the rows of the base matrix in full decoding layers.

BER results for the MS$(4, 6)$ decoder and the NS-FAID-$w_1w_2w_3$ are shown in Figure 6.16, while ASIC post-synthesis implementation results on 65nm-CMOS technology are shown in Table 6.7. It is worth noting that the NS-FAID-433 and NS-FAID-432 decoders outperform the MS decoder by 0.3 dB and 0.15 dB (at BER $= 10^{-5}$), respectively, at the price of a small degradation of the TAR. NS-FAIDs-333 improves the BER performance by 0.12 dB, with TAR improvement by 13.51% to 16.39%, depending on the CNU type (compressed or uncompressed). NS-FAIDs-332 exhibits similar BER performance, with TAR improvement by 13.51% to 19.30%. The NS-FAID-222 decoder yields the most significant TAR improvement (up to 42.09%), but this comes at the price of a significant BER degradation by ≈ 1 dB as estimated in Chapter 5 (BER curve not shown in Figure 6.16).

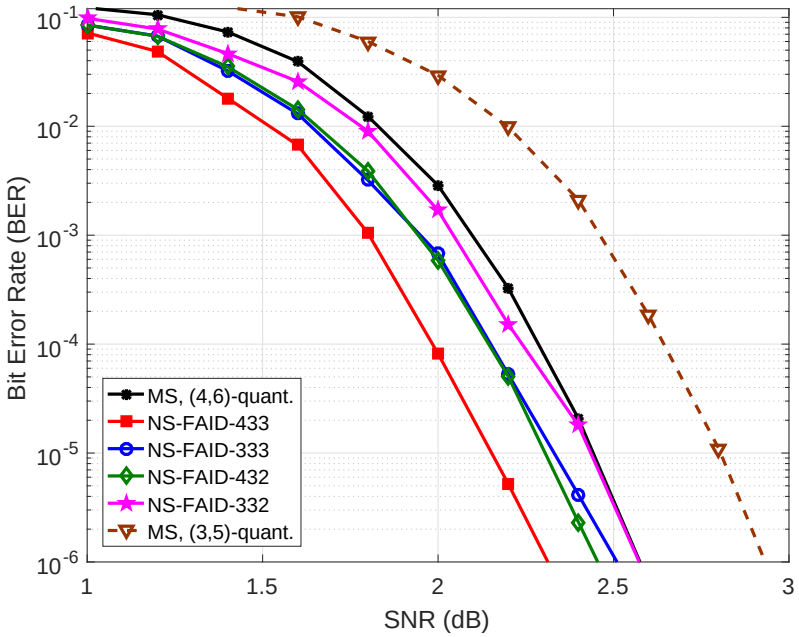


Figure 6.16: BER performance of optimized NS-FAIDs for WiMAX irregular LDPC code

Table 6.7: ASIC post-synthesis implementation results on 65nm-CMOS technology for optimized irregular NS-FAIDs

Variant	**pipeline.uncompressed**					
Decoder	MS(4,6) (WiMAX)	NS-FAID-433	NS-FAID-432	NS-FAID-333	NS-FAID-332	NS-FAID-222
Frequency (MHz)	175	172	178	192	192	200
Throughput (Mbps)	1673	1644	1701	1835	1835	1912
Area (mm^2)	0.87	0.88	0.90	0.82	0.80	0.70
TAR ($Mbps/mm^2$)	1922	1868	1890	2237	2293	2731
±% w.r.t. MS(4,6)	0%	-2.81%	-1.66%	+16.39%	+19.30%	+42.09%

Variant	**pipeline.compressed**					
Decoder	MS(4,6) (WiMAX)	NS-FAID-433	NS-FAID-432	NS-FAID-333	NS-FAID-332	NS-FAID-222
Frequency (MHz)	161	156	161	178	178	200
Throughput (Mbps)	1539	1491	1539	1701	1701	1912
Area (mm^2)	0.77	0.79	0.79	0.75	0.75	0.72
TAR ($Mbps/mm^2$)	1998	1887	1948	2268	2268	2655
±% w.r.t. MS(4,6)	0%	-5.56%	-2.50%	+13.51%	+13.51%	+32.88%

Table 6.8: Comparison between the proposed NS-FAID and state-of-the-art implementations for the WiMAX QC-LDPC code

Author/Year	**K. Zhang'09**	**B. Xiang'11**	**T. Heidari'13**	**W. Zhang'15**	**K. Kanchetla'16**	**This work**
Paper	**[110]**[b]	**[104]**[b]	**[39]**[b]	**[112]**[b]	**[51]**[a]	**NS-FAID-332**[b]
Code length	2304	576-2304	2304	576-2304(†)	576-2304(†)	2304
Technology (nm)	90	130	130	40	90	65
Frequency (MHz)	950	214	100	290	149	192
Iterations	10	10	10	10	5	20
Throughput (Mbps)	2200	955	183	2227	955	1835
Tput scale to 65nm (Mbps)	3036	1910	366	1370	1318	1835
Area (mm^2)	2.90(*)	3.03(*)	6.90(**)	2.26(*)	11.42(*)	0.80(*)
Area scale to 65nm (mm^2)	1.51(*)	0.76(*)	1.73(**)	5.97(*)	5.94(*)	0.80(*)
TAR ($Mbps/mm^2$)	2011	2513	212	229	222	2293
NTAR ($Mbps/mm^2/iter$)	20110	25130	2120	2290	1110	45860

[a] Post-layout results
[b] Post-synthesis results
(†) support both WiMAX and Wi-Fi standards
(*) only core area is reported
(**) total chip area is reported
TAR = (Throughput scaled to 65nm) / (Area scaled to 65nm)
NTAR = TAR × Iterations

Finally, to emphasize the high-throughput characteristic of the proposed architecture, the irregular NS-FAID-332 decoder is further compared with other state-of-the-art implementations of WiMAX decoders in Table 6.8. We also report the TAR and Normalized TAR (NTAR) metrics, so as to keep the throughput comparison on an equal basis with respect to technology, area, and number of iterations. From Table 6.8 it can be seen that the proposed irregular NS-FAID compares favorably with state-of-the-art implementations, providing a NTAR value up to 45.86 Gbps/mm^2/iteration.

6.4 Conclusion

This chapter first presented an enhanced version of the previous hardware architecture for layered QC-LDPC decoders with fully-parallel processing units. The proposed enhancements allow a more efficient hardware usage, thus yielding a significant cost reduction and increase in the maximum operating frequency. Then, two different hardware architectures were presented, making use of either pipelining or increased hardware parallelism in order to increase throughput. Both MS and NS-FAID decoding kernels were integrated into each of the two proposed architectures, and compared in terms of area and throughput. ASIC post-synthesis implementation results demonstrated the effectiveness of the NS-FAID approach in yielding significant improvements in terms of area and throughput, as compared to the MS decoder, with even better or only slightly degraded decoding performance.

Chapter 7

Conclusion and Perspectives

7.1 General Conclusion

The research conducted in thesis was oriented towards the design of efficient hardware implementations of LDPC decoders, through exploiting approximate/imprecise computing and storage techniques in message-passing decoding algorithms.

Approximate computing has been previously investigated for image and video processing applications, which can trade the accuracy of the circuit for area, delay, and/or power improvements. We have shown that a similar trade-off is possible for LDPC decoders. However, one issue specific to LDPC – or more generally FEC – decoders, is to guarantee their ability to provide reliable error protection, despite the integration of approximate computing and storage mechanisms. This is even more problematic in case of iterative message-passing decoders, since such approximations could propagate and accumulate in a catastrophic way during the iterative decoding process. To avoid this happening, specific models and analytic tools are required, so as to determine the impact of such approximations on the iterative decoding process. Proposing such models and analytic tools has been one of the main contributions of this thesis, materialized by the proposed framework of NS-FAID decoders: approximate storage has been modeled through the use of framing functions, while analytic tools based on density evolution and code-aware quantizers have been used to derive optimal solutions in terms of error correction performance. Our findings have further been validated through low-cost hardware implementations, targeting multi-Gigabit/s applications.

We conclude the manuscript by summarizing our contributions on several levels, from theoretical tools, to algorithmic and architectural designs.

A first contribution of the thesis was to investigate code-aware quantizers for finite-alphabet MS decoders. The study revealed that a very simple quantizer – defined as the uniform quantization with unitary step size, of a linearly scaled version of the received signal – provides nearly the optimal solution with regard to the

error correction performance of the finite-alphabet MS decoder. Therefore, finding the optimal quantizer reduces to a 1-dimensional optimization problem, over the possible values of the parameter used to scale the received signal (referred to as gain factor). The approach can be easily adapted to any MS-based decoder, and the simplicity of the solution makes possible the joint optimization of the quantizer and the approximate computing techniques, integrated afterward in various forms into the MS decoder.

At the algorithmic level, the main contributions of the thesis have been to propose several MS-based decoding algorithms, that integrate techniques of approximate computing and storage at different levels. Our contribution is twofold:

- First, we proposed two new decoding algorithms, obtained by introducing two levels of impreciseness in the OMS decoding: the Partially OMS (POMS), which performs only partially the offset correction, and the Imprecise Partially OMS (I-POMS), which introduces a further level of impreciseness in the check-node processing unit.
- The previous contribution has been further generalized to the framework of NS-FAID decoders, which allow storing the exchanged messages using a lower precision than the processing units. NS-FAIDs have been optimized by density evolution for regular and irregular LDPC codes, and shown to yield significant reductions of the memory size requirements, while providing even better or only slightly degraded error correction performance, as compared to conventional MS decoding.

Finally, several low-cost, high-throughput architectures have been proposed, and integrated with both conventional and proposed decoding kernels, so as to demonstrate the benefits of our approximate-computing based solutions. Three hardware architectures have been proposed, optimizing throughput in a cost-efficient way, by making use of (i) efficient hardware reuse, (ii) hazard-free pipelining, or (iii) maximum hardware parallelism. To ensure very high throughput capabilities, all architectures implement layered decoding with fully parallel processing units.

Implementation results for both ASIC and FPGA designs have confirmed the low-cost, high-throughput characteristics of the proposed architectures, as compared to state of the art implementations. Moreover, we have shown that the proposed decoding kernels allow up to 58% improvement – as compared to conventional MS decoding – in the Throughput to Area Ratio (TAR) metric for ASIC designs, or the equivalent Hardware Usage Efficiency (HUE) metric for FPGA designs.

7.2 Perspectives

As discussed in the first chapter of the thesis, cost, throughput and power consumption are key challenges for the next generation of communication systems. High-throughput is required to support the continuously increasing demand of traffic volume, while power consumption incorporates sustainability concerns and is a major constraint in mobile devices.

Therefore, one of the first perspectives of this thesis, is to investigate the impact of the proposed solutions in terms of power consumption. Including power consumption as a parameter of our optimization problem may also require the development of specific solutions, so as to take into account possible trade-offs between power, area, and frequency, the latter directly related to throughput.

A second perspective is to investigate the proposed solutions in conjunction with other high throughput architectures. Unrolling the hardware resources for several decoding iterations is known to allow achieving very-high throughput [87], yet at the price of a significant increase in area. Hence, the integration of approximate computing and storage techniques is expected to increase the cost-efficiency of such a solution.

Finally, the ultimate goal would be reduce the storage requirements to only 1 bit per exchanged message, without degrading – or only with acceptable degradation of – the error correction performance or the convergence speed. Such an approach is closely related to the so-called probabilistic, or "noisy" hard-decision decoders, that have already gained a lot of interest over the last years.

Bibliography

[1] Zhang, Xiaojie and Siegel, P.H. Quantized min-sum decoders with low error floor for LDPC codes. In *proc. of IEEE International Symposium on Information Theory Proceedings*, pages 2871–2875, July 2012.

[2] 3GPP. Study on New Radio (NR) Access Technology Physical Layer Aspects, November 2016. 3rd Generation Partnership Project (3GPP), Tech. Rep. TR-38.802.

[3] S. Abu-Surra, E. Pisek, T. Henige, and S. Rajagopal. Low-power dual quantization-domain decoding for LDPC codes. In *IEEE Global Communications Conference (GLOBECOM)*, pages 3151–3156, 2014.

[4] S. Ajaz and H. Lee. Reduced-complexity local switch based multi-mode QC-LDPC decoder architecture for Gbit wireless communication. *Electronics Letters*, 49(19):1246–1248, 2013.

[5] A. Balatsoukas-Stimming and A. Dollas. FPGA-based design and implementation of a multi-GBPS LDPC decoder. In *proc. of FPL*, pages 262–269, Aug 2012.

[6] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications*, 44(10):1261–1271, 1996.

[7] A. J. Blanksby and C. J. Howland. A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37 (3):404–412, 2002.

[8] O. Boncalo, A. Amaricai, A. Hera, and V. Savin. Cost-efficient FPGA layered LDPC decoder with serial AP-LLR processing. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, 2014.

[9] E. Boutillon and G. Masera. Hardware design and realization for iteratively decodable codes. In D. Declercq, M. Fossorier, and E. Biglieri, editors, *Channel Coding: Theory, Algorithms, and Applications*. Academic Press Library in Mobile and Wireless Communications, Elsevier, June 2014.

[10] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N. E. L'Insalata, F. Rossi, M. Rovini, and L. Fanucci. Low complexity LDPC code decoders for next generation standards. In *Proceedings of the conference on Design, automation and test in Europe*, pages 331–336, 2007.

[11] F. Cai, X. Zhang, D. Declercq, S. K. Planjery, and B. Vasić. Finite alphabet iterative decoders for LDPC Codes: Optimization, architecture and analysis. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(5):1366–1375, 2014.

[12] V. Chandrasetty and S. Aziz. FPGA Implementation of High Performance LDPC Decoder Using Modified 2-Bit Min-Sum Algorithm. In *Computer Research and Development, 2010 Second International Conference on*, pages 881–885, May 2010.

[13] V. A. Chandrasetty and S. M. Aziz. An area efficient LDPC decoder using a reduced complexity min-sum algorithm. *Integration, the VLSI Journal*, 45(2): 141–148, 2012.

[14] V. A. Chandrasetty and S. M. Aziz. Resource efficient LDPC decoders for multimedia communication. *INTEGRATION, the VLSI journal*, 48:213–220, 2015.

[15] J. Chen and M. Fossorier. Density evolution for two improved BP-based decoding algorithms of LDPC codes. *IEEE Communications Letters*, 6(5):208–210, 2002.

[16] J. Chen and M. P. Fossorier. Near optimum universal belief propagation based decoding of low density parity check codes. *IEEE Trans. on Communications*, 50(3):406–414, 2002.

[17] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Hu. Reduced-complexity decoding of ldpc codes. *IEEE Trans. on Communications*, 53(8): 1288–1299, 2005.

[18] X. Chen, J. Kang, S. Lin, and V. Akella. Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(1):98–111, 2011.

[19] J. Cho, J. Kim, and W. Sung. Optimal Output Quantization of Binary Input AWGN Channel for Belief-Propagation Decoding of LDPC Codes. In *proc. of IEEE Workshop on Signal Processing Systems*, pages 282–287, Oct 2012.

[20] S. Chung. *On the construction of some capacity-approaching coding schemes*. PhD thesis, Massachusetts Institute of Technology, 2000.

[21] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 db of the Shannon limit. *IEEE Communications letters*, 5(2):58–60, 2001.

[22] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation. *IEEE Transactions on Information Theory*, 47(2):657–670, 2001.

[23] Cisco. Cisco visual networking index: global mobile data traffic forecast update, 2014–2019, 2015. Tech. Repport.

[24] Z. Cui, Z. Wang, and Y. Liu. High-throughput layered LDPC decoding architecture. *IEEE transactions on very large scale integration (VLSI) systems*, 17 (4):582–587, 2009.

[25] K. Cushon, S. Hemati, C. Leroux, S. Mannor, and W. J. Gross. High-throughput energy-efficient LDPC decoders using differential binary message passing. *IEEE Transactions on Signal Processing*, 62(3):619–631, 2014.

[26] A. Darabiha, A. C. Carusone, and F. R. Kschischang. Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity. In *2005 IEEE International Symposium on Circuits and Systems*, pages 5194–5197, 2005.

[27] A. Darabiha, A. C. Carusone, and F. R. Kschischang. Power reduction techniques for LDPC decoders. *IEEE Journal of Solid-State Circuits*, 43(8):1835–1845, 2008.

[28] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on Information Theory*, 48(6):1570–1579, 2002.

[29] J. Dielissen, A. Hekstra, and V. Berg. Low cost LDPC decoder for DVB-S2. In *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, pages 130–135. European Design and Automation Association, 2006.

[30] E. Eleftheriou, T. Mittelholzer, and A. Dholakia. Reduced-complexity decoding algorithm for low-density parity-check codes. *IET Electronics Letters*, 37 (2):102–104, 2001.

[31] ETSI. ETSI EN 302 307 v1.3.1 digital video broadcasting (DVB); second generation, 2013. [online]. Available: https://www.dvb.org/standards/dvb-s2.

[32] G. P. Fettweis. The tactile internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9(1):64–70, 2014.

[33] M. Fossorier, M. Mihaljevic, and H. Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Trans. on Communications*, 47(5):673–680, 1999.

[34] M. P. Fossorier. Quasicyclic low-density parity-check codes from circulant permutation matrices. *IEEE Transactions on Information Theory*, 50(8):1788–1793, 2004.

[35] R. G. Gallager. Low-density parity-check codes. *IRE Trans. on Information Theory*, 8(1):21–28, 1962.

[36] R. G. Gallager. Low density parity check codes. MIT Press, Cambridge, 1963. Research Monograph series.

[37] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo. A survey of FPGA-based LDPC decoders. *IEEE Communications Surveys & Tutorials*, 18(2):1098–1122, 2016.

[38] J. Hauser. MOSFET device scaling. In *Handbook of Semiconductor Manufacturing Technology*, pages 8–21. Boca Raton, FL: CRC Press, 2008.

[39] T. Heidari and A. Jannesari. Design of high-throughput QC-LDPC decoder for WiMAX standard. In *21st Iranian Conference on Electrical Engineering (ICEE)*, pages 1–4, 2013.

[40] D. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *IEEE Workshop on Signal Processing Systems (SIPS)*, pages 107–112, 2004.

[41] X.-Y. Hu, E. Eleftheriou, and D. Arnold. Regular and irregular progressive edge-growth tanner graphs. *IEEE Transactions on Information Theory*, 51 (1):386–398, Jan 2005.

[42] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold. Regular and irregular progressive edge-growth tanner graphs. *IEEE Transactions on Information Theory*, 51(1): 386–398, 2005.

[43] IEEE-802.11ad. Standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications amendment 3: Enhancements for very high throughput in the 60 GHz band, 2012.

[44] IEEE-802.11n. Standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput, 2009.

[45] IEEE-802.15.3c. [Online]: http://www.ieee802.org/15/pub/TG3c.html.

[46] IEEE-802.16e. Physical and medium access control layers for combined fixed and mobile operation in licensed bands, 2005. Amendment to Air Interface for Fixed Broadband Wireless Access Systems.

[47] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. on Information Theory*, 47(2):619–637, Feb 2001.

[48] M. Jiang, C. Zhao, L. Zhang, and E. Xu. Adaptive offset min-sum algorithm for low-density parity check codes. *IEEE Communications Letters*, 10(6):483–485, 2006.

[49] N. Jiang, K. Peng, J. Song, C. Pan, and Z. Yang. High-throughput QC-LDPC decoders. *IEEE Transactions on Broadcasting*, 55(2):251–259, 2009.

[50] S. J. Johnson. *Iterative error correction: turbo, low-density parity-check and repeat-accumulate codes*. Cambridge University Press, 2009.

[51] V. K. Kanchetla, R. Shrestha, and R. Paily. Multi-standard high-throughput and low-power quasi-cyclic low density parity check decoder for worldwide interoperability for microwave access and wireless fidelity standards. *IET Circuits, Devices & Systems*, 10(2):111–120, 2016.

[52] M. Karkooti and J. R. Cavallaro. Semi-parallel reconfigurable architectures for real-time LDPC decoding. In *Proc. of Int. Conf. on Inf. Technology: Coding and Computing (ITCC)*, volume 1, pages 579–585, 2004.

[53] Y. Kou, S. Lin, and M. P. Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information Theory*, 47(7):2711–2736, 2001.

[54] F. R. Kschischang and B. J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16(2):219–230, 1998.

[55] S. Kumawat, R. Shrestha, N. Daga, and R. Paily. High-throughput LDPC-decoder architecture using efficient comparison techniques & dynamic multi-frame processing schedule. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(5):1421–1430, 2015.

[56] B. Kurkoski and H. Yagi. Quantization of binary-input discrete memoryless channels. *IEEE Transactions on Information Theory*, 60(8):4544–4552, Aug 2014.

[57] B. Kurkoski, K. Yamaguchi, and K. Kobayashi. Noise thresholds for discrete LDPC decoding mappings. In *proc. of IEEE Global Telecommunications Conference*, pages 1–5, Nov 2008.

[58] G. Li, I. J. Fair, and W. A. Krzymien. Density evolution for nonbinary ldpc codes under gaussian approximation. *IEEE Transactions on Information Theory*, 55(3):997–1015, 2009.

[59] J. Li, G. He, H. Hou, Z. Zhang, and J. Ma. Memory efficient layered decoder design with early termination for LDPC codes. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 2697–2700, 2011.

[60] A. Lingamneni, C. Enz, J. L. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2011.

[61] A. Liveris and C. Georghiades. On quantization of low-density parity-check coded channel measurements. In *proc. of IEEE Global Telecommunications Conference*, volume 3, pages 1649–1653, Dec 2003.

[62] S. Lloyd. Least Squares Quantization in PCM . *IEEE Transactions on Information Theory*, 28(2):129–137, Mar. 1982.

[63] X. Ma, X. Zhang, H. Yu, and A. Kavcic. Optimal quantization for soft-decision decoding revisited. In *proc. of ISITA*, Oct. 2002.

[64] D. J. MacKay and M. S. Postol. Weaknesses of margulis and ramanujan-margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science*, 74:97–104, 2003.

[65] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. on Information Theory*, 45(2):399–431, 1999.

[66] M. Mansour and N. Shanbhag. Low-power VLSI decoder architectures for LDPC codes. In *IEEE Int. Symp. on Low Power Electronics and Design (ISLPED)*, pages 284–289, 2002.

[67] R. G. Maunder. Survey of ASIC implementations of LDPC decoders, 2016.

[68] R. G. Maunder. A vision for 5G channel coding. *AccelerComm White Paper*, 2016.

[69] Z. Mheich, T. Nguyen-Ly, V. Savin, and D. Declercq. Code-aware quantizer design for finite-precision min-sum decoders. In *IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, Varna, Bulgaria, June 2016.

[70] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016.

[71] C. L. K. Ngassa, V. Savin, and D. Declercq. Design of min-sum-based LDPC decoders using imprecise arithmetic. In *IEEE Int. Conf. on Computer as a Tool (EUROCON)*, pages 375–382, 2013.

[72] C. Novak, P. Fertl, and G. Matz. Quantization for soft-output demodulators in bit-interleaved coded modulation systems. In *proc. of IEEE ISIT*, pages 1070–1074, June 2009.

[73] D. Oh and K. Parhi. Min-sum decoder architectures with reduced word length for LDPC codes. *IEEE Trans. on Circuits and Systems I: Regular Papers*, 57 (1):105–115, 2010.

[74] Y. S. Park, D. Blaauw, D. Sylvester, and Z. Zhang. Low-power high-throughput LDPC decoder using non-refresh embedded DRAM. *IEEE Journal of Solid-State Circuits*, 49(3):783–794, 2014.

[75] X. Peng, Z. Chen, X. Zhao, D. Zhou, and S. Goto. A 115mW 1Gbps QC-LDPC decoder ASIC for WiMAX in 65nm CMOS. In *IEEE Asian Solid State Circuits Conference (A-SSCC)*, pages 317–320, 2011.

[76] S. Planjery, D. Declercq, L. Danjean, and B. Vasic. Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders. *Electronics Letters*, 47(16):919–921, 2011.

[77] S. K. Planjery, S. K. Chilappagari, B. Vasić, D. Declercq, and L. Danjean. Iterative decoding beyond belief propagation. In *IEEE Information Theory and Applications Workshop (ITA)*, pages 1–10, 2010.

[78] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic. Finite alphabet iterative decoders – part I: Decoding beyond belief propagation on the binary symmetric channel. *IEEE Transactions on Communications*, 61(10):4033–4045, 2013.

[79] W. Rave. Quantization of log-likelihood ratios to maximize mutual information. *IEEE Signal Processing Letters*, 16(4):283–286, April 2009.

[80] T. Richardson. Error floors of LDPC codes. In *Proceedings of the annual Allerton conference on communication control and computing*, volume 41, pages 1426–1435, 2003.

[81] T. Richardson and R. Urbanke. Multi-edge type LDPC codes. In *Workshop honoring Prof. Bob McEliece on his 60th birthday, California Institute of Technology, Pasadena, California*, pages 24–25, 2002.

[82] T. Richardson, M. Shokrollahi, and R. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. on Information Theory*, 47(2):619–637, 2001.

[83] T. J. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2001.

[84] C. Roth, A. Cevrero, C. Studer, Y. Leblebici, and A. Burg. Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1772–1775, 2011.

[85] V. Savin. Self-corrected min-sum decoding of LDPC codes. In *Proc. of IEEE Int. Symp. on Information Theory (ISIT)*, pages 146–150, 2008.

[86] V. Savin. LDPC decoders. In D. Declercq, M. Fossorier, and E. Biglieri, editors, *Channel coding: Theory, algorithms, and applications*, pages 211–260. Elsevier, 2014.

[87] P. Schläfer, N. Wehn, M. Alles, and T. Lehnigk-Emden. A new dimension of parallelism in ultra high throughput LDPC decoding. In *2013 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 153–158, 2013.

[88] J. Sha, M. Gao, Z. Zhang, Z. Wang, et al. Efficient decoder implementation for QC-LDPC codes. In *2006 International Conference on Communications, Circuits and Systems*, volume 4, pages 2498–2502, 2006.

[89] J. Sha, Z. Wang, M. Gao, and L. Li. Multi-Gb/s LDPC code design and implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(2):262–268, 2009.

[90] E. Sharon, S. Litsyn, and J. Goldberger. An efficient message-passing schedule for LDPC decoding. In *Proc. of 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pages 223–226, 2004.

[91] J. Singh, O. Dabeer, and U. Madhow. On the limits of communication with low-precision analog-to-digital conversion at the receiver. *IEEE Trans. on Commun.*, 57(12):3629–3639, Dec. 2009.

[92] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. ISSN 0925-5001.

[93] Y. Sun and J. R. Cavallaro. A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards. In *2008 IEEE International on SOC Conference*, pages 367–370, 2008.

[94] R. Tanner. A recursive approach to low complexity codes. *IEEE Trans. on Inf. Theory*, 27(5):533–547, 1981.

[95] Y.-L. Ueng, C.-J. Yang, Z.-C. Wu, C.-E. Wu, and Y.-L. Wang. VLSI decoding architecture with improved convergence speed and reduced decoding latency for irregular LDPC codes in WiMAX. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 520–523, 2008.

[96] Y.-L. Ueng, C.-J. Yang, K.-C. Wang, and C.-J. Chen. A multimode shuffled iterative decoder architecture for high-rate RS-LDPC codes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(10):2790–2803, 2010.

[97] P. O. Vontobel and R. Koetter. Graph-cover decoding and finite-length analysis of message-passing iterative decoding of ldpc codes. *arXiv preprint cs/0512078*, 2005.

[98] Z. Wang and Z. Cui. Low-complexity high-speed decoder design for quasi-cyclic LDPC codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(1):104–114, 2007.

[99] Z. Wang and Z. Cui. A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 15(4):483–488, 2007.

[100] M. Weiner, M. Blagojevic, S. Skotnikov, A. Burg, P. Flatresse, and B. Nikolic. 27.7 A scalable 1.5-to-6Gb/s 6.2-to-38.1 mW LDPC decoder for 60GHz wireless networks in 28nm UTBB FDSOI. In *2014 IEEE International on Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 464–465, 2014.

[101] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin. Algorithms of finding the first two minimum values and their hardware implementation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(11):3430–3437, 2008.

[102] X. Wu, Y. Song, M. Jiang, and C. Zhao. Adaptive-normalized/offset min-sum algorithm. *IEEE Communications Letters*, 14(7):667–669, 2010.

[103] B. Xiang and X. Zeng. A 4.84 mm^2 847–955 Mb/s 397 mW dual-path fully-overlapped QC-LDPC decoder for the WiMAX system in 0.13 μm CMOS. In *IEEE Symp. on VLSI Circuits (VLSIC)*, pages 211–212, 2010.

[104] B. Xiang, D. Bao, S. Huang, and X. Zeng. An 847–955 Mb/s 342–397 mW dual-path fully-overlapped QC-LDPC decoder for WiMAX system in 0.13 μm CMOS. *IEEE Journal of Solid-State Circuits*, 46(6):1416–1432, 2011.

[105] M. Xu, J. Wu, and M. Zhang. A modified offset min-sum decoding algorithm for LDPC codes. In *IEEE Int. Conf. on Computer Science and Information Technology (ICCSIT)*, volume 3, pages 19–22, 2010.

[106] H. Yagi and B. Kurkoski. Channel quantizers that maximize random coding exponents for binary-input memoryless channels. In *proc. of IEEE ICC*, pages 2228–2232, June 2012.

[107] R. Zarubica, S. G. Wilson, and E. K. Hall. Multi-Gbps FPGA-based low density parity check (LDPC) decoder design. In *IEEE Global Communications Conference (GLOBECOM)*, pages 548–552, 2007.

[108] J. Zhang, M. Fossorier, and D. Gu. Two-dimensional correction for min-sum decoding of irregular ldpc codes. *IEEE Communications Letters*, 10(3):180–182, 2006.

[109] J. Zhang, Y. Wang, M. P. Fossorier, and J. S. Yedidia. Iterative decoding with replicas. *IEEE Transactions on Information Theory*, 53(5):1644–1663, 2007.

[110] K. Zhang, X. Huang, and Z. Wang. High-throughput layered decoder implementation for quasi-cyclic LDPC codes. *IEEE Journal on Selected Areas in Communications*, 27(6):985–994, 2009.

[111] K. Zhang, X. Huang, and Z. Wang. A high-throughput LDPC decoder architecture with rate compatibility. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(4):839–847, 2011.

[112] W. Zhang, S. Chen, X. Bai, and D. Zhou. A full layer parallel QC-LDPC decoder for WiMAX and Wi-Fi. In *2015 IEEE 11th International Conference on ASIC (ASICON)*, pages 1–4, 2015.

[113] X. Zhang and P. H. Siegel. Quantized min-sum decoders with low error floor for LDPC codes. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 2871–2875, 2012.

[114] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright. Investigation of error floors of structured low-density parity-check codes by hardware emulation. In *IEEE GLOBECOM*, Nov 2006.

[115] Zhang, Zhengya and Dolecek, L. and Nikolic, B. and Anantharam, V. and Wainwright, M.J. Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices. *IEEE Trans. on Communications*, 57(11):3258–3268, Nov 2009.