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

LE TRUNG Khoa

Équipes d'accueil :
Équipe Traitement des Images et du Signal (ETIS) – CNRS UMR 8051
École Nationale Supérieure de l'Électronique et de ses Applications

Titre de la thèse

New Direction on Low Complexity Implementation of Probabilistic Gradient Descent Bit-Flipping Decoder

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

Emmanuel Boutillon	Professeur, Lab-STICC, Université Bretagne Sud	Rapporteur
Chris Winstead	Professeur, Utah University, USA	Rapporteur
Christophe Jégo	Professeur, IMS, Institut Polytechnique de Bordeaux	Examinateur
Charly Poulliat	Professeur, INP-ENSEEIHT Toulouse	Examinateur
Valentin Savin	Dr., CEA-LETI, MINATEC, Grenoble	Examinateur
Fakhreddine Ghaffari	MCF, Université de Cergy Pontoise	Encadrant
David Declercq	Professeur, ENSEA, Université de Cergy Pontoise	Directeur de thèse

Dành cho Ba Má thân yêu của con,
Ba Lê Trung Nhân và Má Nguyễn Thị Kim Chấn

Dành cho Chị và các em,
Dành cho vợ và con gái yêu dấu,

Cho tình thương của bố, mẹ dành cho con
Cho tình yêu của Chị và các em,
Cho tình yêu của Vợ và con

To my parents,
To my brothers and sisters,
To my wife and daughter,
For your love,

À mes parents,
À mes frères et soeurs,
À ma femme et ma pettite fille,

Acknowledgment

I would like to express my deep gratitude to my advisors, Prof. David Declercq and Assoc. Prof. Fakhreddine Ghaffari, for their continuously guidance, support and corrections throughout the duration of my PhD work. In particular, I would like to thank them for believing in my potential and agreeing to become my doctoral advisors, for providing meaningful ideas, for initiating fruitful collaborations with partners which enabled me to finish my thesis successfully.

I would like to thank Prof. Emmanuel Boutillon and Prof. Chris Winstead for acting as my thesis reviewers, Prof. Christophe Jégo for serving as the president of the PhD committee and Prof. Charly Poulliat, Dr. Valentin Savin for being the examiners. The comments and corrections from the committee helped me significantly improve my thesis as well as my future career.

During my PhD study, I had the opportunities of doing some research visits to Error Correction Coding Laboratory in University of Arizona, USA, under supervision of Prof. Bane Vasíc. I would like to thank him for all of his supports, for providing me with very interesting ideas and discussions. I want to thank Xin Xiao, Nithin, Mohsen for discussing with me. For the research visit to University Politehnica Timisoara, Romania, I would like to thank Oana Boncalo and Alexandru Amaricai for their help and discussions .

I extend my thanks to all the colleagues in ETIS, ENSEA for their friendship, funs and encouragements especially Lam Nguyen, Hong Phan, Diouf Madiagne, Alexandre Marcastel.. and Truong Nguyen-Ly from CEA-LETI, Grenoble. The administrate assistant of our laboratory, Annick Bertinoti, and administrative assistant of the doctoral school, Emmanuelle Travet, Naima Chalabi, were always very helpful. Many thanks go to them for taking care of the administrative issues.

I would like to express my sincere gratitude to my colleagues in University of Technology (Bach Khoa University), Viet Nam National University Ho Chi Minh City, especially Ho Trung My, Huynh Thu, Hoang Trang, Do Hong Tuan, Duong Hoai Nghia for encouraging me to pursue the PhD study.

Last, but not least, my profound gratitude to my family, especially my beloved parents, Le Trung Nhan and Nguyen Thi Kim Chan, my brothers and sisters, Thanh Huong, Ngoc Lan, Minh Tuong, Trung Nghia, my wife, Van Nga and especially my lovely daughter, Sophie Vinh An, for their moral supports and encouragement throughout my life. They have inspired me and given me strength throughout my whole life.

Cergy - France, May 2017

LE TRUNG KHOA

Author's publications related to the PhD

Published papers

[J1] **K. Le**, F. Ghaffari, D. Declercq and B. Vasić, "Efficient Hardware Implementation of Probabilistic Gradient Descent Bit-Flipping", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1-12, 2016.

[C1] **K. Le**, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis and B. Vasíc, "Efficient realization of probabilistic gradient descent bit flipping decoders", *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1494-1497, May 2015.

[C2] **K. Le**, F. Ghaffari, D. Declercq and B. Vasić, "Hardware Optimization of the Perturbation for Probabilistic Gradient Descent Bit Flipping Decoders", *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017 **(accepted)**.

[C3] B. Vasić, P. Ivanis, D. Declercq and **K. Le**, "Approaching Maximum Likelihood Performance of LDPC Codes by Stochastic Resonance in Noisy Iterative Decoders", *Information Theory and Applications Workshop (ITA 2016)*, San Diego, CA, Feb. 2016.

Participation to research projects

The author participated to the research project *"Innovative Reliable Chip Designs from Low-Powered Unreliable Components"* (i-RISC), supported by the European Commission under the Seventh Framework Programme (Grant agreement number 309129) and the research project *"Message passing Iterative Decoders based on Imprecise Arithmetic for Multi-Objective Power-Area-Delay Optimization"* (DIAMOND) supported by the Agence National de la Recherche (ANR) under the Franco-Romanian (ANR-UEFISCDI) Join Research Program.

Résumé

L'algorithme de basculement de bits à descente de gradient probabiliste (Probabilistic Gradient Descent Bit Flipping - PGDBF) est récemment introduit comme un nouveau type de décodeur de décision forte pour le code de contrôle de parité à faible densité (Low Density Parity Check - LDPC) appliqué au canal symétrique binaire. En suivant précisément les étapes de décodage du décodeur déterministe Gradient Descent Bit-Flipping (GDBF), le PGDBF intègre en plus la perturbation aléatoire dans l'opération de basculement des Nœuds de Variables (VNs) et produit ainsi une performance de décodage exceptionnelle qui est meilleure que tous les décodeurs à basculement des bits (Bit Flipping - BF) connus dans la littérature, et qui approche les performances du décodeur de décision souple. Nous proposons dans cette thèse plusieurs implémentations matérielles du PGDBF, ainsi qu'une analyse théorique de sa capacité de correction d'erreurs. Avec une analyse de chaîne de Markov du décodeur, nous montrons qu'en raison de l'incorporation de la perturbation aléatoire dans le traitement des VNs, le PGDBF s'échappe des états de piégeage qui empêchent sa convergence. De plus, avec la nouvelle méthode d'analyse proposée, la performance du PGDBF peut être prédite et formulée par une équation de taux de trames erronées en fonction du nombre des itérations, pour un motif d'erreur donné. L'analyse fournit également des explications claires sur plusieurs phénomènes de PGDBF tels que le gain de re-décodage (ou de redémarrage) sur un motif d'erreur reçu. La problématique de l'implémentation matérielle du PGDBF est également abordée dans cette thèse. L'implémentation classique du décodeur PGDBF, dans laquelle un générateur de signal probabiliste est ajouté au-dessus du GDBF, est introduite avec une augmentation inévitable de la complexité du décodeur. Plusieurs procédés de génération de signaux probabilistes sont introduits pour minimiser le surcoût matériel du PGDBF. Ces méthodes sont motivées par l'analyse statistique qui révèle les caractéristiques critiques de la séquence aléatoire binaire requise pour obtenir une bonne performance de décodage et suggérer les directions possibles de simplification. Les résultats de synthèse montrent que le PGDBF déployé avec notre méthode de génération des signaux aléatoires n'a besoin qu'une très faible complexité supplémentaire par rapport au GDBF tout en gardant les mêmes performances qu'un décodeur PGDBF théorique. Une implémentation matérielle intéressante et particulière du PGDBF sur les codes LDPC quasi-cyclique (QC-LDPC) est proposée dans la dernière partie de la thèse. En exploitant la structure du QC-LDPC, une nouvelle architecture pour implémenter le PGDBF est proposée sous le nom d'architecture à décalage des Nœuds de Variables (Variable-Node Shift Architecture - VNSA). En implémentant le PGDBF par VNSA, nous montrons que la complexité matérielle du décodeur est même inférieure à celle du GDBF déterministe tout en préservant la performance de décodage aussi élevée que celle fournie par un PGDBF théorique. Enfin, nous montrons la capacité de cette architecture VNSA à se généraliser sur d'autres types d'algorithmes de décodage LDPC.

Abstract

Probabilistic Gradient Descent Bit Flipping (PGDBF) algorithm have been recently introduced as a new type of hard decision decoder for Low-Density Parity-Check Code (LDPC) applied on the Binary Symmetric Channel. By following precisely the decoding steps of the deterministic Gradient Descent Bit-Flipping (GDBF) decoder, PGDBF additionally incorporates a random perturbation in the flipping operation of Variable Nodes (VNs) and produces an outstanding decoding performance which is better to all known Bit Flipping decoders, approaching the performance of soft decision decoders. We propose in this thesis several hardware implementations of PGDBF, together with a theoretical analysis of its error correction capability. With a Markov Chain analysis of the decoder, we show that, due to the incorporation of random perturbation in VN processing, the PGDBF escapes from the trapping states which prevent the convergence of decoder. Also, with the new proposed analysis method, the PGDBF performance can be predicted and formulated by a Frame Error Rate equation as a function of the iteration, for a given error pattern. The analysis also gives a clear explanation on several phenomenons of PGDBF such as the gain of re-decoding (or restarting) on a received error pattern. The implementation issue of PGDBF is also addressed as a main part in this thesis. The conventional implementation of PGDBF, in which a probabilistic signal generator is added on top of the GDBF, is shown with an inevitable increase in hardware complexity. Several methods for generating the probabilistic signals are introduced which minimize the overhead complexity of PGDBF. These methods are motivated by the statistical analysis which reveals the critical features of the binary random sequence required to get good decoding performance and suggesting the simplification directions. The synthesis results show that the implemented PGDBF with the proposed probabilistic signal generator method requires a negligible extra complexity with the equivalent decoding performance to the theoretical PGDBF. An interesting and particular implementation of PGDBF for the Quasi-Cyclic LDPC (QC-LDPC) is shown in the last part of the thesis. Exploiting the structure of QC-LDPC, a novel architecture to implement PGDBF is proposed called Variable-Node Shift Architecture (VNSA). By implementing PGDBF with VNSA, it is shown that the decoder complexity is even smaller than the deterministic GDBF while preserving the decoding performance as good as the theoretical PGDBF. Furthermore, VNSA is also shown to be able to apply on other types of LDPC decoding algorithms.

Contents

List of Figures

List of Tables

Glossary

APP: A Posteriori Probability

AWGN: Additive White Gaussian Noise

BER: Bit Error Rate

BI-AWGN: Binary-Input Additive White Gaussian Noise

BP: Belief-Propagation

BSC: Binary Symmetric Channel

CN: Check Node

CNU: Check Node Processing Unit

DE: Density Evolution

EM: Edge-Memory

FER: Frame Error Rate

LDPC: Low Density Parity Check

LLR: Log-Likelihood Ratio

LSB: Least Significant Bit

ML: Maximum Likelihood

MP: Message Passing

MS: Min-Sum

NMS: Normalized Min-Sum

OMS: Offset Min-Sum

QC: Quasi-Cyclic

SNR: Signal-to-Noise Ratio

VN: Variable Node

VNU: Variable Node Processing Unit

Chapter 1

Introduction

1.1 Context and motivations

Figure 1.1 illustrates an application system where the LDPC coding is employed. A vector of K information bits is sent through a noisy channel in which errors may occur. The noisy channel could be any communication channel such as wire, wireless communication system... where errors may appear during the transmission or the noisy storage medium such as flash memory... where errors happen during the storing, writing to or reading out of the memory. With targeting to detect and correct the errors, LDPC is deployed by adding the LDPC encoder and decoder to the system as in Figure 1.1. LDPC encoder forms a N-bits codeword by adding M parity bits to the K information bits sequence before sending through the channel. At the channel output, N received bits are processed by an LDPC decoder targeting to accurately recover K information bits.

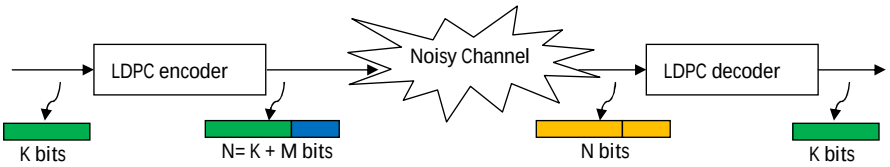


Figure 1.1: An illustrating system where LDPC code is applied.

LDPC codes have attracted high attention in the past several years due to their excellent performance under iterative decoding process. These studies focus either on improving the error correction performance of the LDPC codes or on reducing the implementation complexity of the decoders for practical applications. Soft decision iterative decoding, such as Belief Propagation (BP) or Min-Sum (MS) algorithms offers the best error correction performance, close to the theoretical coding bounds, but comes along with an intensive computation cost [27]. On the contrary, the class of hard-decision iterative decoders is often seen as a very low-complexity solution,

very fast in computation, with an associated performance loss. This work focuses on the class of A Posteriori Probability (APP) based hard decision decoders, belonging to the family of Bit-Flipping (BF) decoders where, contrary to message-passing decoders, both the extrinsic and the intrinsic information are exchanged between the nodes of the Tanner graph [3][28].

From the recent literature, it can be seen that the hard decision decoding algorithms are an attractive research topic due to their low complexity and fast computation. In the next generation of communication systems, throughput and power consumption are the key challenges. The high throughput system is required to support the continuously increasing demand of traffic volume, while power consumption, especially in the mobile devices, becomes a significant concern. LDPC error correction modules are a part of the system and so, need to be optimized to fulfill these high throughput and energy saving targets. From this point, hard decision decoders, *i.e.* Bit Flipping decoders, with their advantages of simple and fast computation, become promising candidates provided that their error correction ability is improved. Indeed, hard decision decoders require very simple computations which may lead to an extremely high throughput. Also, the simple computations with very low decoder complexity may result less energy consumption in hard decision decoders than the soft decision decoders. The problem is that the decoding performance of hard decision decoders is usually weak and need to be improved in order to cope with error correction demand.

In the direction of improving BF decoding performance by modifying the algorithm computations, the recent study of iterative LDPC decoders implemented on faulty-hardware has led to the counter-intuitive conclusion that the noisy decoders could perform better than their noiseless versions. In other words, the random perturbation on algorithm computations, in some case, helps improving the error correction capability. This random perturbation (probabilistic) effect on decoding performance appears to be very attractive and worth to study. The reason of this is twofold. First, since random perturbation improves the decoding performance, it seems simpler to inject the randomness to the computation than modifying the decoding algorithm with the same decoding gain. The probabilistic algorithms studied in this thesis are, actually, better than all similar deterministic decoders in term of performance. Second, with the understanding on the improving principle of perturbation on decoding performance, it facilitates the design of fault-tolerant decoders.

This thesis focuses on a newly proposed probabilistic BF-based decoder which is a random perturbation version of Gradient Descent Bit Flipping (GDBF) [8], called as Probabilistic Gradient Descent Bit Flipping (PGDBF) proposed by Rasheed *et al.* in [15], applied on BSC channel. PGDBF provides the best error correction compared to all known BF-based decoders and approaches the soft information decoding performance such as Min-Sum. Although the very promising performance capability, several issues of PGDBF were left unexplored so far. First, the outstanding performance of PGDBF is observed by simulation. **There is lack of an analytic method to analyse and explore its asymptotic performance gain.** Second, the advantage of PGDBF comes from the presence of a probabilistic signal generator on top of the deterministic decoder. One of our preliminary work

in [29] showed that the additional cost for this probabilistic signal generator is too large and becomes the bottleneck of PGDBF. **The efficient implementation of PGDBF is, therefore, needed to minimize the complexity overhead while maintaining its performance gain**.

1.2 Main contributions and thesis outline

The main contributions of this thesis are:

- Propose an analytic method to analyze the PGDBF decoding algorithm. This analyzing method can be extended to other hard decision decoders.
- Propose an efficient hardware realization of PGDBF decoder for generic LDPC codes that minimizes the complexity overhead.
- Propose a novel, hardware efficient architecture for LDPC decoding suitable for QC-LDPC algorithms demonstrated by the PGDBF proof-of-concept implementation.

We briefly summarize in the following the content of each chapter of the thesis correspondingly to the above contributions.

Chapter 2: Hard decision decoders

Chapter 2 starts by providing a brief introduction of LDPC codes and LDPC decoding concept. The channel models, *i.e.* Binary Symmetric Channel and Additive White Gaussian Noise channel, in which the distortion affects on the transmitted signals, are also presented. The Quasi-Cyclic construction of LDPC codes is introduced by highlighting its structure which is used in the proposed architecture called Variable-Node Shift Architecture in Chapter 5. The second part of this chapter provides a literature review on the development of the hard decision decoders. We first discuss the BF-based decoders. BF decoder family is a type of A Posteriori Probability propagation decoders and several solutions have been proposed in the litterature to modify and improve the original BF decoder [2][8][14][15]. From the recent literature, it is clear that BF decoders made a big evolution from a "toy" algorithm with a very weak in error correction capability to very powerful decoder that can be competitive with the soft decision decoders. *Extrinsic* message passing decoders such as Gallager-A, Gallager-B... are also presented in this chapter together with the new type of noise-added BF decoders - Noisy Gradient Descent Bit Flipping decoder applied on AWGN and Probabilistic Gradient Descent Bit Flipping decoder applied on BSC channel. The noise-added BF decoders share the same principle that adds a random perturbation to the selection of the flipped bits, helps the decoders to escape from the trapping points which prevent the convergence of the decoders. Beside the good decoding performance of these noise-added BF decoders, their hardware complexity becomes the emerging issue which comes from

the exhaustive implementation of the randomness generation. Smart implementations are required such that the performance gain is preserved while minimizing the additional hardware overhead.

Chapter 3: Theoretical analysis of Probabilistic Gradient Descent Bit Flipping

Chapter 3 introduces an analysis method for hard decision decoders denoted as Finite State Tracking (FST). Although FST is shown to be able to apply on different type of hard decision decoders, we limit the FST presentation in this chapter only on PGDBF decoder. FST represents the PGDBF decoding process from an iteration to another as a state transition in a Markov Chain (MC). A state of PGDBF decoding is represented as a state in this MC and by analyzing the MC, the closed-form expression of Frame Error Rate as a function of number of iteration can be derived. The uncorrectable and partial-uncorrectable error pattern definition is newly introduced in this chapter in order to indicate correspondingly the error pattern that can not be corrected and the one which can be corrected with some probability by PGDBF. The uncorrectable and partial-uncorrectable error pattern can be also determined by analyzing the MC. We illustrate the utilization of FST by analyzing the performance of PGDBF decoder and compare it with other BF decoders. It is firstly shown that the number of uncorrectable error patterns of PGDBF on the tested LDPC code is significantly reduced compared to GDBF and conventional BF decoders. The predictive performance curves on the error floor are then derived, based one the estimated number of uncorrectable and partially correctable error patterns and are finally confirmed by the simulation results. The improvement of PGDBF is clearly explained by showing that many transitions leading to the zero-error state in PGDBF do not appear in deterministic GDBF. Furthermore, FST provably shows that for some given error patterns, the PGDBF can converge to the zero-error state only if some specific transitions occur. Getting into these transitions depends on the realization of the random signal. This explains the decoding gain phenomenon of re-initialization (or restarting, rewinding) of the PGDBF decoding from the beginning with new random realizations.

Chapter 4: Efficient hardware implementation of Probabilistic Gradient Descent Bit Flipping

Chapter 4 presents an efficient hardware (HW) implementation of the PGDBF decoder which minimizes the resource overhead needed to implement the random perturbations and the maximum finder of the PGDBF. In Section 4.2, the conducted statistical analysis in PGDBF is presented in order to show the precise characterization of its key parameters, especially the values of the random generator parameters that lead to the maximum coding gains. This analysis is performed through Monte Carlo simulations in both the waterfall and the error floor regions. Section 4.3 shows the optimized HW architecture for the PGDBF decoder. The proposed architecture is based on the use of a short random signals that are duplicated to fully apply the

PGDBF decoding rules on the whole codeword. Two different initialization solutions are proposed with equivalent HW overheads, but with different behaviors on different LDPC codes. An optimization of the maximum finder unit of the PGDBF algorithm is also presented in order to reduce the critical path and improve the decoding throughput. Finally, Section 4.4 shows the synthesis results on ASIC 65nm technology, and Monte-Carlo simulations with a bit-accurate C implementation of the proposed PGDBF architecture on LDPC codes with various rates and lengths.

Chapter 5: A Quasi-Cyclic friendly architecture for LDPC decoders : the Variable-Node Shift Architecture

Chapter 5 introduces a new decoding architecture for the QC-LDPC codes, called as Variable-Node Shift Architecture (VNSA). The VNSA takes advantage of the structure of the QC-LDPC codes to shift the memory of the decoders while preserving the exact decoding operations. It is shown in this chapter that the VNSA-based decoders significantly reduce the complexity and achieve the better decoding performance compared to the conventional decoder implementations. These advantages come from the fact that by shifting the memory of the decoders, the variable node computations can be processed differently when different types of variable nodes are implemented. This dynamical processing helps the decoder break some trapping states and converge while the decoder with conventional implementation does not. The hardware savings also come from the fact that some variable node implementations in VNSA are simpler than those of the conventional implementation making the global complexity reduced. The chapter is presented as follows. The VNSA principle is firstly presented in the generic form in Section 5.2 since it can be applied to different decoding algorithms. The illustrations of VNSA applications on different types of LDPC decoders are presented in the next sections (Section 5.3 and 5.4) for the edge-type memory and node-type memory decoders in which the VNSA is shown to be well adapted for all of these decoders. The VNSA becomes trully advantageous when different functions are required for the VNU or the CNU implementations. These advantages are either expressed in terms of performance gain, or in HW complexity savings which is discussed in Section 5.5. We illustrate the interest of this new architecture with the implementation of the Probabilistic Gradient Descent Bit Flipping basing on the VNSA (called VNSA-PGDBF) in Section 5.6. It is shown that the outstanding decoding performance of PGDBF is preserved in VNSA-PGDBF while the decoder complexity is significantly reduced and even smaller than the deterministic GDBF. A further simplified version of VNSA-PGDBF is also introduced, called as the imprecise VNSA-PGDBF (VNSA-IM-PGDBF). This VNSA-IM-PGDBF not only reduces the complexity compared to the VNSA-PGDBF but also improves the decoding performance.

Chapter 2

Hard decision decoders

This chapter starts by providing a brief introduction of LDPC codes and LDPC decoding concept (Section 2.1). The channel models, *i.e.* Binary Symmetric Channel and Additive White Gaussian Noise channel, in which the distortion affects on the transmitted signals, are also presented. The Quasi-Cyclic construction of LDPC codes is introduced by highlighting its structure which is used in the proposed architecture called Variable-Node Shift Architecture in Chapter 5. The second part of this chapter provides a literature review on the development of the hard decision decoders. We first discuss the BF-based decoders in Section 2.2. BF decoder family is a type of A Posteriori Probability propagation decoders and several solutions have been proposed in the litterature to modify and improve the original BF decoder [2][8][14][15]. From the recent literature, it is clear that BF decoders made a big evolution from a "toy" algorithm with a very weak in error correction capability to very powerful decoder that can be competitive with the soft decision decoders. *Extrinsic* message passing decoders such as Gallager-A, Gallager-B... are also presented in this chapter (Section 2.3) together with the new type of noise-added BF decoders - Noisy Gradient Descent Bit Flipping decoder applied on AWGN and Probabilistic Gradient Descent Bit Flipping decoder applied on BSC channel (Section 2.4). The noise-added BF decoders share the same principle that adds a random perturbation to the selection of the bits to be flipped, helps the decoders to escape from the trapping points which prevent the convergence of the decoders. Beside the good decoding performance of these noise-added BF decoders, their hardware complexity becomes the emerging issue which comes from the exhaustive implementation of the randomness generation. Efficient implementations are required such that the performance gain is preserved while minimizing the additional hardware overhead. The hardware complexity of BF decoders is reviewed in Section 2.5. Section 2.6 concludes this chapter.

2.1 Low-Density Parity-Check codes and channel models

2.1.1 Low-Density Parity-Check codes

A binary LDPC code is defined by a sparse parity-check matrix H with size $(M \times N)$, where $N > M$. Each row of H represents a parity check function, computed by a Check Node (CN), on the bits so-called Bit Nodes or Variable Nodes (VN) represented by the columns of H. The CN c_m $(1 \leq m \leq M)$ checks the VN v_n $(1 \leq n \leq N)$ if the entry $h_{m,n} = 1$. An example of the parity-check matrix H is illustrated in Figure 2.1.

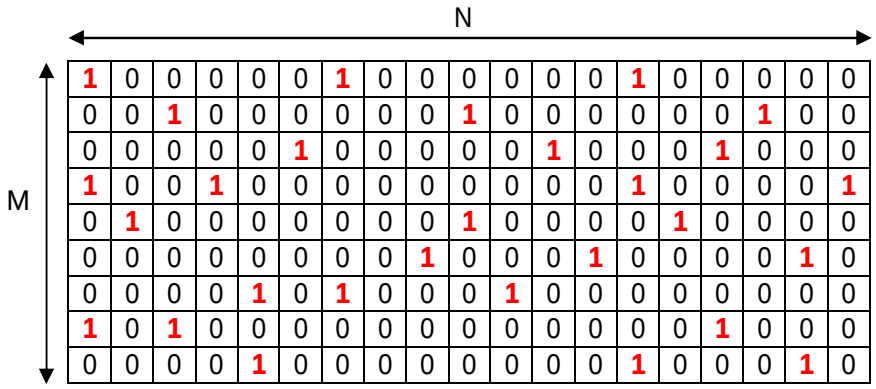


Figure 2.1: An example of the parity check matrix H.

A codeword is a vector $\mathbf{x} = (x_1, x_2, \ldots x_N) \in \{0,1\}^N$ which satisfies $H\mathbf{x}^T = \mathbf{0}$. This codeword $\mathbf{x}$ is produced by the LDPC encoder (see Figure 1.1) and is sent through a transmission channel. The output of the channel is denoted by $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$. Depending on the channel model, the value of y_n could be only a binary value (0 or 1) in Binary Symmetric Channel or a higher level of precision as in the Additive White Gaussian Noise (AWGN) channel. An introduction of the channel models is presented in the next section.

Another alternative representation of an LDPC code is a bipartite graph called Tanner graph composed of two types of nodes, the VNs v_n, $n = 1, \ldots, N$ and the CNs c_m, $m = 1, \ldots, M$ as in Figure 2.2 where the cycles represent the VNs and squares represent the CNs. In the Tanner graph, a VN v_n is connected by an edge to a CN c_m if the entry $h_{m,n} = 1$.

The set of CNs connected to the VN v_n is called the neighbor set of this VN and denoted as $\mathcal{N}(n)$. Similarly, the set of VNs connected to the CN c_m is the neighbor set of this CN and referred as $\mathcal{N}(m)$. The connection degree is defined by the size of the neighbor set, i.e. the VN degree $d_{v_n} = |\mathcal{N}(n)|$ and the CN degree $d_{c_m} = |\mathcal{N}(m)|$. An LDPC code is called as regular code when its connection degrees are equal for all nodes, i.e. $d_{c_m} = d_c, \forall m$ and $d_{v_n} = d_v, \forall n$.

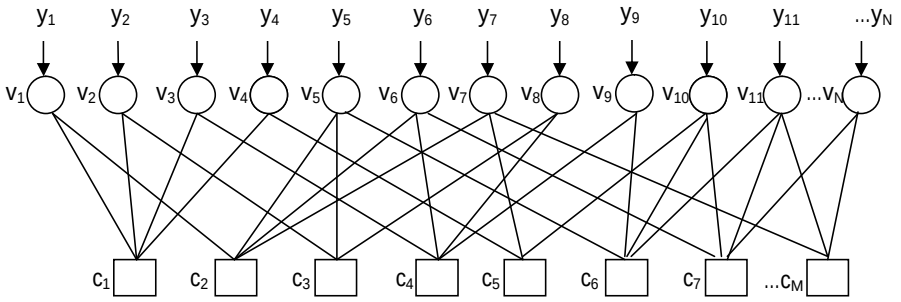


Figure 2.2: The Tanner graph presentation of a parity matrix H.

2.1.2 LDPC decoding concepts

LDPC decoding process is defined by the iterative updating and passing the computed messages between the VNs and CNs on the edges of the Tanner graph through decoding iteration. A decoding iteration is typically composed by the following steps:

- 1. **VNs computation**: The VNs compute the messages to send to the CNs basing on the channel received values and the messages from the CNs.
- 2. **VNs to CNs**: Computed messages are sent to the CNs.
- 3. **CNs computation**: The CNs compute the new messages basing on their incoming messages.
- 4. **CNs to VNs**: The new CNs messages are sent to the VNs.

During the decoding process, the codeword validity is always verified basing on the intermediate values by the syndrome check computation. The decoding process is terminated when either a valid codeword is found, in which case the decoding success is declared or the number of iterations reach to the maximum allowed value, It_{max}, in which case the decoding failure is declared.

The nature of passed messages defines different types of LDPC decoding algorithms. In soft decision message passing algorithms such as Belief Propagation (BP) or Min-Sum (MS) [27], the probability of a given bit to be 0 or 1 is passed back and forth along the edges of the Tanner graph. These soft messages require a very complex computations in the VNs and CNs and the large interleaver (or interconnection) network to exchange them. However, they provide the best decoding performance approaching the decoding bounds. Another type of LDPC decoding algorithms is hard decision decoders in which only binary hard decision values are exchanged between VNs and CNs. The computations in VNs and CNs of hard decision decoders are, therefore, usually simple and the interconnection network is small. This makes the hard decision decoders usually lower in complexity, faster in computation compared to soft decision decoders. However, a non-negligible performance loss is observed in hard decision decoders resulting in not being the first option for practical applications.

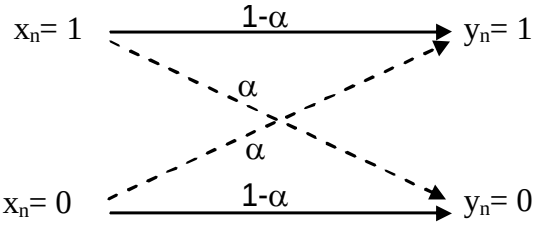


Figure 2.3: The Binary Symmetric Channel model.

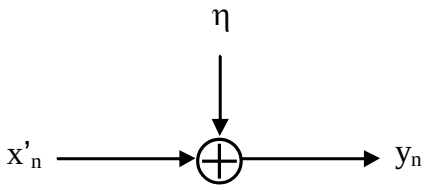


Figure 2.4: The Additive White Gaussian Noise channel model.

2.1.3 The channel models of LDPC decoding

Depending on the transmission channel model, the decoder input value, y_n, can be differently defined. In Binary Symmetric Channel (BSC) model, each bit x_n in $\mathbf{x}$ is flipped with a probability α called channel crossover probability as described in Figure 2.3 forming y_n, *i.e.* $y_n = x_n$ with $p = 1 - \alpha$ and $y_n = not(x_n)$ with $p = \alpha$.

In Additive White Gaussian Noise (AWGN) channel, x_n is polarized ($x'_n = 1 - 2 * x_n$) before being transmitted and y_n is a real number defined by $y_= x'_n + \eta$ where η is a zero-mean Gaussian noise. Typically, in the hardware realization of LDPC decoder for this type of channel model, y_n is usually quantized by a quantizing function Q, $y_n = Q(x'_n + \eta_n)$ to $q_t > 1$ bits precision.

2.1.4 Quasi-cyclic Low-Density Parity-Check codes

The LDPC code is conventionally designed by a random locating the entries 1 in the initial all-zero matrix H forming a sparse matrix. With this conventional design, the hardware implementation is shown to be complex and less flexible. The QC-LDPC code was proposed to facilitate the decoder implementation. A QC-LDPC code is a specifically construction method of H in which each vertex in a small *base matrix* H_B ($n_r \times n_c$) is replaced by either a circulant shift of main diagonal $Z \times Z$ matrix or $Z \times Z$ *all-zero* matrix, ($N = n_c * Z$, $M = n_r * Z$). An example of a main diagonal $Z \times Z$ matrix and one of its circulant shift version are illustrated in Figure 2.5. An extended matrix H from H_B is shown in Figure 2.6.

A well-known LDPC code which is used in several works, is the Tanner ($N = 155$, $M = 93$), ($d_v = 3, d_c = 5$) LDPC code [30] presented below in which $n_r = 3$, $n_c = 5$ and circulant size $Z = 31$. A vertex in this base matrix indicates the cyclic shift factor of 31x31 diagonal matrix.

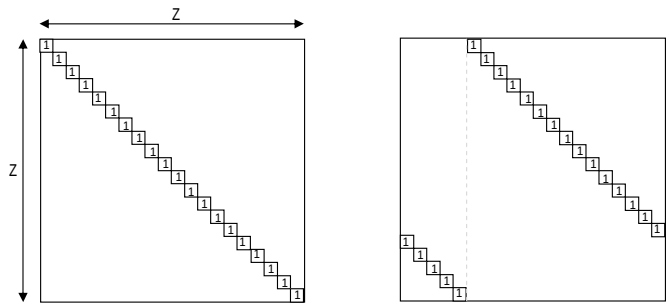


Figure 2.5: A main diagonal $Z \times Z$ matrix and one of its circulant shift version used in the construction of QC-LDPC code. The zero entries are not shown for the sake of simplicity.

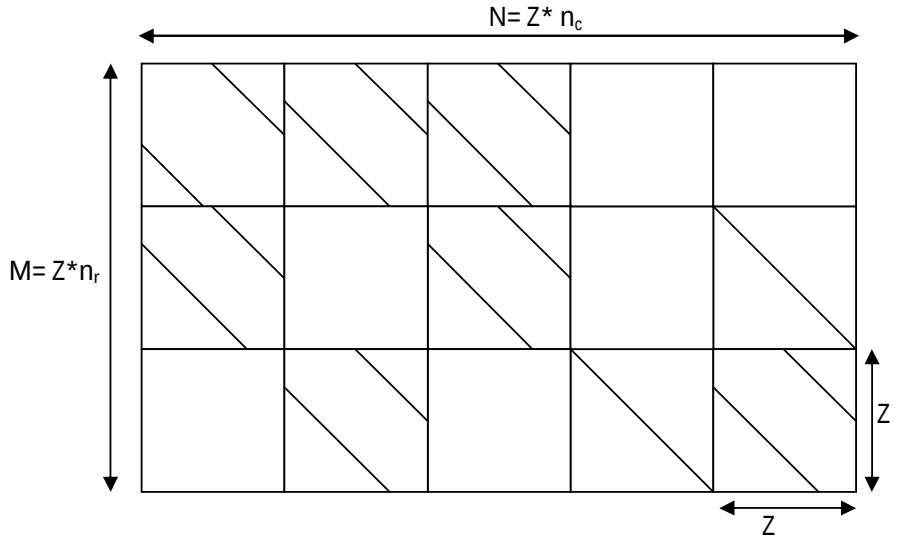


Figure 2.6: An example of the parity check matrix H of a QC-LDPC code.

$$H_B = \begin{pmatrix} 1 & 2 & 4 & 8 & 16 \\ 5 & 10 & 20 & 9 & 18 \\ 25 & 19 & 7 & 14 & 28 \end{pmatrix}$$

By design, the connections between the VNs and CNs in QC-LDPC codes are very constructive as illustrated in Figure 2.7. Indeed, the CNs connected to VNs are arranged in a constructive order. The Z consecutive VNs in a column of the base matrix will connect to Z consecutive CNs in a row. The only difference is on the connecting order. Z CNs are connected straightly to Z VNs when shift weight is 0 (Figure 2.7a) while they are are cyclically shifted by the cyclic-shift-weight before being connected (Figure 2.7b). Another example of the connections between Z VNs in a base column with $d_v = 3$ to all neighbor CNs is illustrated in Figure 2.8.

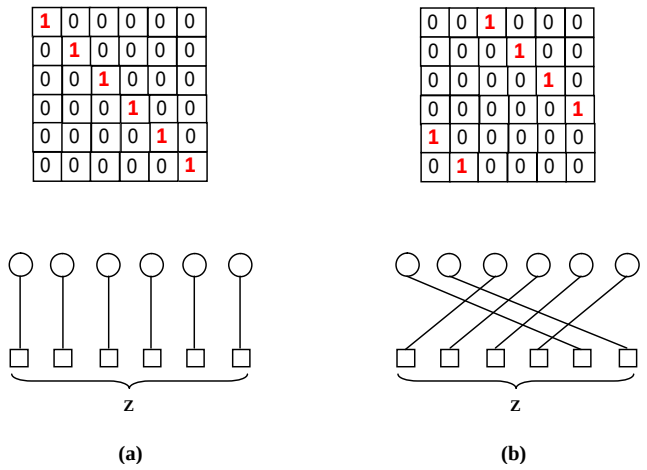


Figure 2.7: The cyclic shift of 6x6 diagonal matrix with shift factor is 0 (a) and 2 (b) and the corresponding connections between the VNs and CNs.

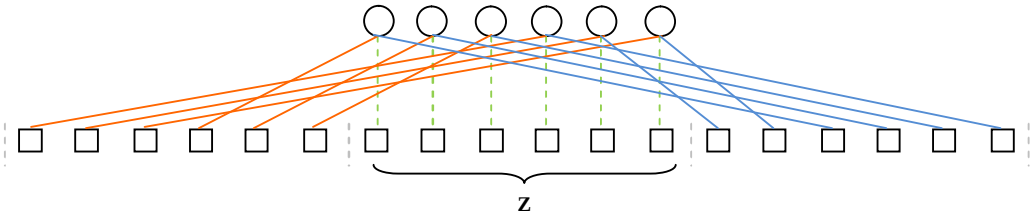


Figure 2.8: A full connection of Z VNs in a column of the base matrix with $d_v = 3$ to their neighbor CNs.

2.2 Bit-Flipping-based Decoders

The first hard decision decoding skim, referred as *bit flipping decoding*, was introduced by Gallager in [1] in 1963 and works as follows:

- 1. All the parity checks are computed and all the VN that have more than a fix number of unsatisfied CNs will be flipped.
- 2. The new values of VNs are used to recompute the parity checks, the process is repeated until all parity checks are satisfied or the maximum number of iterations is reached.

This hard decoding skim was left unexplored for around 4 decades until the rediscovery introduced in 2001 by Y. Kou *et al.* [2] with the Weighted Bit Flipping (WBF) decoding algorithm. From the WBF introduction to today, a long list of BF decoders was introduced which either improved the decoding performance or accelerated the decoding speed while keeping the decoding principle introduced by Gallager. The BF decoders development is summarized in Figure 2.9.

The introduced BF decoders share the same principle that exchanging only 1 bit information between CNs and VNs and that the CNs compute the parity check on

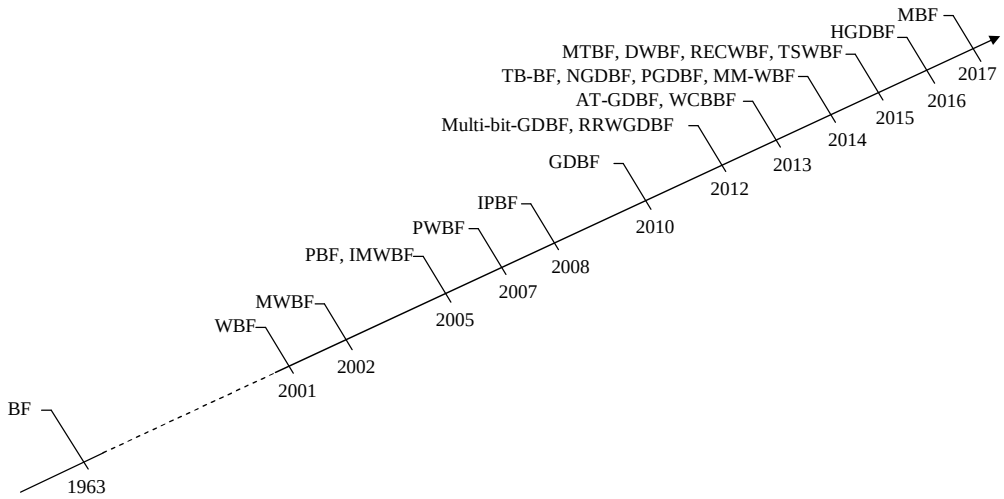


Figure 2.9: The BF decoders development since the bit flipping decoding skim introduced by Gallager in 1963. These BF decoders can be found at the following references: BF[1], WBF[2], MWBF[3], PBF[4], IMWBF[5], PWBF[6], IPBF[7], GDBF[8], Multi-bit-GDBF[9], RRWGDBF[10], AT-GDBF[11], WCBBF[12], TB-BF[13], NGDBF[14], PGDBF[15], MM-WBF[16], MTBF[17], DWBF[18], RECWBF[19], TSWBF[20], HGDBF[21], MBF[22].

all the connected VNs as described in Equ. 2.1 then send the result equally to all the connected VNs. In the equations in this chapter, we denote by $v_n^{(k)}$ the hard decision value of the VN v_n at the k-th iteration. We denote correspondingly by $c_m^{(k)}$ the binary value of the CN c_m at iteration k, which indicates whether the m-th parity-check equation is satisfied (being 0) or not (being 1). The main differences between these BF decoders are on the formulation of the computed value so-called *inversion function* or *energy function* in each VN and on the mechanism to choose the VN to flip.

$$c_m^{(k)} = \bigoplus_{n \in \mathcal{N}(m)} v_n^{(k)} \tag{2.1}$$

2.2.1 Energy computation in BF decoders

Each VN in BF decoders computes the energy value during the decoding process and basing on this value, a VN is decided to be flipped or not. Gallager introduced a simple sum of binary checksums in the energy computation in VN n at the iteration k as in Equ. 2.2 in which an unsatisfied neighbor CN contributes -1 to the energy value while a satisfied one adds a $+1$.

$$E_n^{(k)} = \sum_{m \in \mathcal{N}(n)} (1 - 2\, c_m^{(k)}) \tag{2.2}$$

By taking into account the soft value of channel information in AWGN channel, WBF and its variants then modified the energy formulation as following. The WBF in [2] added the weights, $\omega_{m,n}^{WBF}$, to the checksum in energy function described in Equ. 2.3. The weight value of CN m to compute energy of VN n, $\omega_{m,n}^{WBF}$, was defined as in Equ. 2.4.

$$E_n^{(k)} = \sum_{m \in \mathcal{N}(n)} \omega_{m,n}^{WBF}(1 - 2\, c_m^{(k)}) \tag{2.3}$$

$$\omega_{m,n}^{WBF} = \omega_{m,n}^{MWBF} = \min_{n' \in \mathcal{N}(m)} |y_{n'}| \tag{2.4}$$

The modified WBF (MWBF) in [3] followed the same manner of energy computation of WBF and further incorporated the channel output value of the VN scaled by a empirical optimized factor α (Equ. 2.5).

$$E_n^{(k)} = \sum_{m \in \mathcal{N}(n)} \omega_{m,n}^{MWBF}(1 - 2\, c_m^{(k)}) + \alpha |y_n| \tag{2.5}$$

The improved MWBF (IMWBF) in [5] modified the checksum weights computation by excluding the channel value of a VN to compute the weight of a CN for this VN as in Equ. 2.6.

$$\omega_{m,n}^{IMWBF} = \min_{n' \in \mathcal{N}(m) \setminus n} |y_{n'}| \tag{2.6}$$

The authors of the proposed modified WBF decoders above suggested that the computations of the checksum weights in Equ. 2.4 and 2.6 are proceeded before the decoders actually start (in the initialization phase). The multiplication element, $\alpha * y_n$, in the energy equation of MWBF and IMWB can also be computed at that initialization phase. The computations of these BF decoders, therefore, were shown to be simple and fast since they are only the additions on the real values.

Many other BF-based decoders have been proposed recently such as Mixed Modified WBF (MM-WBF) [16] in 2014, Two State WBF (TSWBF) [20], Multi-threshold BF (MTBF) [17], Dynamic Weighted Bit Flipping (DWBF) [18], Recursive WBF (RECWBF) [19] in 2015 and Multi-Bit Flipping (MBF) [22] at the beginning of 2017. These proposed decoders either continued to modify the computed weight values such as DWBF [18], RECWBF [19] or proposed the new approach MM-WBF [16], TSWBF [20], MTBF [17], MBF [22]. The modification on energy computation, in general, improved the decoding performance at the cost of an increment of decoder complexity. In MM-WBF and TSWBF, the authors suggested to mix the formulation of energy computations of different BF-based decoders into a single decoder and the decoding process is divided into different phases with respect to the number of iterations. One phase uses an energy formulation while another

phase uses another formulation. MTBF [17], MBF [22] changed the bit flip selection mechanisms, provided a better converging speed and will be discussed in the next section.

The Gradient Descent Bit Flipping (GDBF) algorithm introduced by Wadayama *et al.* in [8] proposed another approach. The authors considered the decoding as an optimization process by defining an objective function described in Equ. 2.7. The energy computation function of GDBF algorithm was derived from a gradient descent formulation and GDBF principle consisted of finding the most suitable bits to be flipped in order to maximize this pre-defined objective function. The derived energy computation function in GDBF is described in Equ. 2.8.

$$f^{(k)} = \sum_{n=1}^{N}(1 - 2v_n^{(k)})y_n + \sum_{m=1}^{M}(1 - 2\,c_m^{(k)}) \tag{2.7}$$

$$E_n^{(k)} = (1 - 2v_n^{(k)})y_n + \sum_{m\in\mathcal{N}(n)}(1 - 2\,c_m^{(k)}) \tag{2.8}$$

The sign correlation between the tentative hard decision of a VN at iteration k, $v_n^{(k)}$, to its channel value, y_n, contributes to the VN energy by the first term of Equ. 2.8. The bit at iteration k having the same sign between $v_n^{(k)}$ and y_n tends to have the larger energy $E_n^{(k)}$. The second term in Equ. 2.8 is only the sum of all parity check values from all neighbor CN (with weights $\omega_{m,n}^{GDBF} = 1$). The GDBF energy computation was even simpler than the WBF-based decoders by adding a real value to the binaries. GDBF provided the best decoding performance to all known BF decoders at the introducing time.

GDBF algorithm was firstly modified to apply on the BSC channel by Rasheed *et al.* in 2014 in [15] . Since there is no soft information in BSC, the bit energy can be seen as the discrete energy and computed in Equ. 2.9 where $\oplus$ is the Exclusive-OR operation. The GDBF on BSC provided also better performance compared to other BF decoders.

$$E_n^{(k)} = v_n^{(k)} \oplus y_n + \sum_{m\in\mathcal{N}(n)} c_m^{(k)} \tag{2.9}$$

Several modifications of GDBF decoders were then introduced with even better in decoding performance and some of them approached soft-decision decoders. Others modifications are at the target of improving the converging speed. These typical GDBF variants can be selectively listed as Multi-Bit type GDBF [9] and Reliability Ratio Weight GDBF [10] in 2012, Adaptive Threshold BF [11] in 2013, Noisy GDBF [14] and Probabilistic GDBF [15] in 2014, Hibrid GDBF [21] in 2016.

The Reliability Ratio Weight GDBF (RRWGDBF) in [10] added the weight $\omega_{m,n}^{RRWGDBF}$ on the neighbor CN values as in Equ. 2.10 and 2.11. The RRWGDBF was shown to have better in converging speed with the equivalent decoding performance to GDBF.

$$E_n^{(k)} = (1 - 2v_n^{(k)})y_n + \sum_{m \in \mathcal{N}(n)} \omega_{m,n}^{RRWGDBF}(1 - 2\,c_m^{(k)}) \tag{2.10}$$

$$\omega_{m,n}^{RRWGDBF} = \frac{1}{|y_n|} \sum_{n' \in \mathcal{N}(m)} |y_{n'}| \tag{2.11}$$

The HGDBF in [21] proposed in 2016 recommended not to keep the channel value y_n constantly but update it during the decoding process. HGDBF used the same energy function of GDBF decoder while the channel reliability y_n at the k-th iteration is updated as in Equ. 2.12 where α_1 is a simulation optimized factor. The HGDBF offered 0.4 dB better in decoding gain compared to GDBF with an additional complexity overhead for updating the y_n.

$$y_n^{(k+1)} = y_n^{(k)} + v_n^{(k)}.\alpha_1.\frac{N_0}{E_b}E_n^{(k)} \tag{2.12}$$

The Multi-Bit type GDBF [9], Adaptive Threshold BF (ATBF) were introduced by modifying the bit flip selection and is described in the next section. The Noisy GDBF [14] and Probabilistic GDBF [15] also modified on the bit flip selection by a very particular method that incorporated randomness in the selection. This random incorporation turned the GDBF to the most powerful decoder compared to all introduced BF decoders, approached to soft decision decoders such as Min-Sum [14]. These two decoders are discussed in the noise-aided decoders section (Section 2.4).

2.2.2 Flipping strategies

Different BF decoders also differ on the mechanism of choosing the flipped bits in each iteration. It can be classified into the single flip type where only 1 bit is flipped in an iteration *i.e.* the flipping set $\mathcal{B}$: $|\mathcal{B}| = 1$ and the multiple flip type where multiple bits are flipped in an iteration *i.e.* $|\mathcal{B}| > 1$.

In the serial flip decoders, only the bit that has the smallest energy in N energy values will be flipped as described in Equ. 2.13, *i.e.* $|\mathcal{B}| = 1$. The single flip concept comes from the fact that the bit that receives the most unsatisfied CNs is likely to be the erroneous bit and should be flipped. When the large number of iterations is allowed, the single flip decoders offers the very good decoding performance, while the number of iteration is limited, single flip decoders perform worse than the multiple flip decoders. This comes from the fact that they flip only 1 VN at an iteration and therefore, converge very slow to the correct codeword. WBF, MWBF, IMWBF and decoders with the prefix "S-" such as S-RRWGDBF in this work, are the single flip BF decoders. Another limitation of single BF decoders is that they require a global sorting over all energy values which is high complexity and slow especially in the case of large N.

$$\mathcal{B}^{(k)} = \{v_n | n = \underset{n' \in [1...N]}{argmin}(E_{n'}^{(k)})\} \tag{2.13}$$

In the multiple flip decoders, all the bits that have the energy smaller than a threshold $\tau < 0$, are flipped in parallel. The flipped set is determined by Equ. 2.14. Typically, the multiple flip BF decoders provided a better convergence speed compared to the serial. Note that, when $|\tau|$ is set too large, few VNs are flipped in an iteration and decoders tend to behave as serial flip mode. When $|\tau|$ is set too small, many bits are likely to be flipped in an iteration, the decoders may oscillate and fail to converge to a correct codeword as observed the performance loss in [8].

$$\mathcal{B}^{(k)} = \{v_n | E_n^{(k)} < \tau\} \tag{2.14}$$

Several solutions were proposed in order to compensate this performance loss such as the adaptive threshold procedure as in ATBF [11], the fixing of number flipped bits in an iteration in MBF [22]. In adaptive threshold BF (ATBF) decoder, at the initialization stage, each VN is set with a threshold $\tau_n^{(0)} = \tau^{(0)} \forall n$. An adaptive factor θ which is a simulation optimized factor is also set. During the decoding process, if a VN has energy lower than threshold, it will be flipped, otherwise, the threshold is updated by $\tau_n^{(k+1)} = \theta.\tau_n^{(k)}$. The beneficial consequence of threshold scaling at the VN level is that the decoders start with multiple bit flipped and progressively limit the fewer flipped bit as most CN satisfied. The adaptive threshold moves decoders from parallel flip to serial flip intrinsically [11]. The threshold adaptive process maintains the good decoding performance while improving significantly the converging speed which is interpreted as improving the decoding throughput.

The MBF in [22] proposed a multiple flip process by limiting to a fixed number of flipped bits, ϑ, in each iteration for all decoding iteration. A global sorting block indicates ϑ VNs which have the smallest energy and flip them in parallel. This method offered a good decoding performance. However the complexity of the global sorting is high especially with large value of N.

The GDBF [8] decoding algorithm proposed a method to choose the bit to flip in order to maximize the objective function described in Equ. 2.7. GDBF also came up with a single flip (Equ. 2.13) which provided a good decoding performance but slow in converging speed and the multiple flip (Equ. 2.14) with higher converging speed.

The authors in [8] proposed a hybrid method by using the multiple flip mode for fast error correction at the beginning and switching to the single flip mode to avoid the oscillation in the later phase of decoding process. The objective function is evaluated at each iteration and the mode switching is proceeded whenever the objective function does not increases. This GDBF decoder is denoted as M-GDBF. The authors further proposed a modification called as *"M-GDBF with escape"* denoted as M-esc-GDBF. Two thresholds τ_1 and τ_2 are used where τ_1 is used for the multiple mode at the beginning of decoding process. After several iterations, the multiple

mode is changed to the single mode, and the search point may eventually arrive to a local-maximum which is not a codeword. In this case, the decoder changes to the multiple mode again with the threshold τ_2 in only the first iteration and then τ_1 is used. The M-esc-GDBF is comparable to the MS decoder as described in Figure 2.10.

For the GDBF decoder on BSC channel [15], the bits that has the maximum energy, are flipped (see energy in Equ. 2.9). Due to the integer energy representation, many bits are likely to have the maximum energy and the flipping set is defined in Equ. 2.15

$$\mathcal{B}_{GDBF}^{(k)} = \{v_n | E_n^{(k)} = E_{max}^{(k)}\} \qquad [BSC] \tag{2.15}$$

2.2.3 Probabilistic Bit Flipping

Another interesting bit flip selection mechanism is the probabilistic flip in the multiple flip mode. Its principle is that, instead of flipping all the bits selected in the deterministic decoders, the probabilistic decoders flip these bits with some probability $p_0 < 1$. The probabilistic bit flip selection was firstly proposed by Miladinovic *et al.* in 2005 in [4] . The proposed decoder was called probabilistic BF (PBF). PBF follows precisely the decoding steps of BF proposed by Gallager. However, PBF flips only a random part of the flipping candidates indicated by conventional BF decoders as described in Equ. 2.16 where $p_n^{(k)}$ is the realization of a uniform random variable over the interval [0,1].

$$\mathcal{B}_{PBF}^{(k)} = \{n \in \mathcal{B}_{BF}^{(k)} | p_n^{(k)} < p_0\} \tag{2.16}$$

The probabilistic flip manner helped PBF decoder improve the error correction capability compared to conventional BF decoder as shown in [4]. This probabilistic flip also inspired to Rasheed *et al.* to propose the Probabilistic Gradient Descent Bit Flipping in [15] which was known as the best BF decoder compared to all introduced BF decoders on BSC channel. The Noisy GDBF [14] is also a type of probabilistic flip selection which offers good decoding performance, and it is competitive to soft decoding algorithms in AWGN channel.

2.2.4 Performance comparisons

We make the comparison between different BF decoders in term of decoding performance both in AWGN (Figure 2.10) and BSC channel (Figure 2.11). It can be seen that the BF decoders progressively improved the error correction ability and some decoders are comparable or even surpass the soft decision decoders. Among these decoders, it is also noticed that the noise-aided decoders (N-GDBF in AWGN and PGDBF in BSC, described in Section 2.4) have best error correction gain.

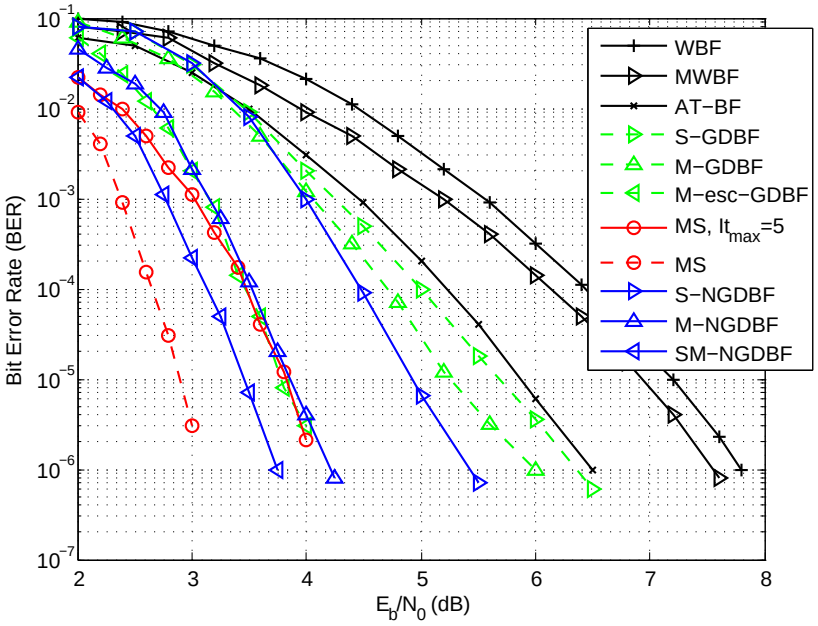


Figure 2.10: Performance comparison between hard decision BF decoders of the regular LDPC code $(d_v = 3, d_c = 6)$, $(N = 1008, M = 504)$ (PEGReg504x1008), the maximum iteration: M-esc-GDBF, $It_{max} = 300$; for MS decoder, $It_{max} = 100$ and $It_{max} = 5$ are used; $It_{max} = 100$ for all other decoders.

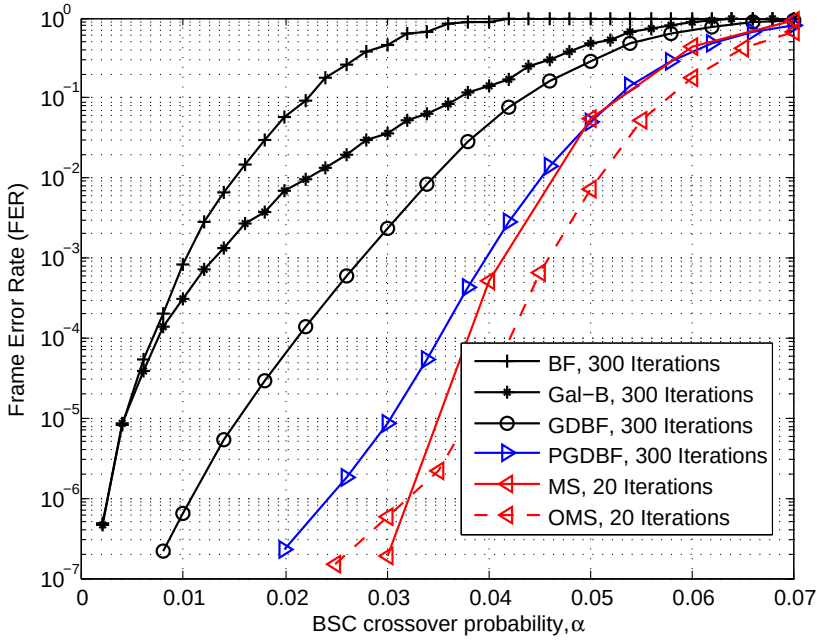


Figure 2.11: Performance comparison between LDPC decoders: BF, Gallager-B (Gal-B), GDBF, PGDBF ($p_0 = 0.9$), Quantized MS, Quantized Offset Min-Sum (OMS) with offset factor of 1 of the regular QC-LDPC code $(d_v = 4, d_c = 8, Z = 54)$, $(N = 1296, M = 648)$.

2.3 Other Diversities of Hard decision Decoders

2.3.1 Gallager-A/Gallager-B decoders

The Gallager-A, Gallager-B are the type of hard decision *extrinsic* message passing decoders in which the message sent to a node (VN or CN) by its neighbor is computed based on all neighbor incoming messages except the one comes from that node. In other words, different neighbors of a node may receive different messages. We denote the message sent from VN n to CN m at iteration k as $v_{n,m}^{(k)}$ and the message sent from a CN m to VN n at iteration k as $c_{m,n}^{(k)}$. The hard decision of VN n at iteration k is denoted as $v_n^{(k)}$. We also denote γ_n as the *a priori* information of VN n which is computed from the channel output y_n. The *a posteriori* information of the VN n at iteration is referred as $\gamma_n^{(k)}$ which is computed as a function of the γ_n and messages received from neighbors. $\gamma_n^{(k)}$ is used to make the hard decision $v_n^{(k)}$.

In order to ease the discussion, it is more convenient to consider the exchanged messages taking values as {+1, -1} rather than {0,1}. The VN hard decision value at iteration k, $v_n^{(k)}$, takes the value 1 when a posteriori information, $\tilde{\gamma}_n$, is -1 and 0 when $\tilde{\gamma}_n = +1$. The sum modulo 2 of binary values in {0,1} in the CN operation is substituted by the product of the corresponding values in {+1, -1}.

The detail of Gallager-B decoder is presented in Algorithm 1. In the initialization step, the priori information, γ_n, is obtained from the binary channel output. The initialized values of VN n to all CN messages are set as the priori information γ_n. The decoder starts to decode as the following steps.

- 1. CN to VN messages: for any CN m and VN $n \in \mathcal{N}(m)$, the messages $c_{m,n}^{(k)}$ are the producta of the incoming messages $v_{n',m}^{(k)}$, $\forall n' \in \mathcal{N}(m) \setminus n$.

- 2. VN to CN messages: for any VN n and CN $m \in \mathcal{N}(n)$, the decoder computes the $E_{n,m}^{(k)}$ of the priori γ_n and the incoming messages $c_{m',n}^{(k)}$, $\forall m' \in \mathcal{N}(n) \setminus m$. $E_{n,m}^{(k)}$ represents the difference between two possible values +1 and -1 among the group $\{\gamma_n, c_{m',n}^{(k)}, \forall m' \in \mathcal{N}(n) \setminus m\}$. If the absolute value of $E_{n,m}^{(k)}$ is less than a threshold value τ, the messages $v_{n,m}^{(k)}$ are set as the priori information γ_n, otherwise, $v_{n,m}^{(k)}$ is set as the sign of $E_{n,m}^{(k)}$.

- 3. A posteriori update: for any VN n, the sum $E_n^{(k)}$ of γ_n and all incoming messages is computed. When $E_n^{(k)} = 0$ in the case when the number of vote for +1 and -1 is equal, $\tilde{\gamma}_n$ is set to the initial value γ_n, otherwise, $\tilde{\gamma}_n$ is set to the value of majority voted.

- 4. Hard decision: for any VN n, the hard decision $v_n^{(k)}$ is computed basing on the posteriori information. The decoder stops when the codeword is found indicated by the syndrome check or the maximum number of iteration is reached.

Algorithm 1 Gallager-B (Ga-B) decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in 0, 1^N$ ▷ received word
Output: $\underline{\boldsymbol{v}} = (v_1, \ldots, v_N) \in \{0, 1\}^N$ ▷ estimated codeword
Initialization
for all $n = 1, \ldots, N$ **do** $\gamma_n = 1 - 2y_n$;
for all $n = 1, \ldots, N$ and $m \in \mathcal{N}(n)$ **do** $v_{n,m}^{(0)} = \gamma_n$;
Iteration Loop
for all $m = 1, \ldots, M$ and $n \in \mathcal{N}(m)$ **do** ▷ **CN-processing**

$$c_{m,n}^{(k)} = \prod_{n' \in \mathcal{N}(m) \backslash n} v_{n',m}^{(k)}$$

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

$$E_{n,m}^{(k)} = \gamma_n + \sum_{m' \in \mathcal{N}(n) \backslash m} c_{m',n}^{(k)};$$

$$v_{n,m}^{(k)} = \begin{cases} \gamma_n, & \text{if} \quad |E_{n,m}^{(k)}| < \tau \\ \text{sign}(E_{n,m}^{(k)}), & \text{otherwise.} \end{cases}$$

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

$$E_n^{(k)} = \gamma_n + \sum_{m' \in \mathcal{N}(n)} c_{m',n}^{(k)};$$

for all $n = 1, \ldots, N$ **do**

$$\tilde{\gamma}_n^{(k)} = \begin{cases} \gamma_n, & \text{if} \quad E_n^{(k)} = 0 \\ \text{sign}(E_n^{(k)}), & \text{otherwise.} \end{cases}$$

for all $n = 1, \ldots, N$ **do** ▷ **hard decision**

$$v_n^{(k)} = \frac{(1 - \tilde{\gamma}_n^{(k)})}{2};$$

if $\underline{\boldsymbol{v}}^{(k)} = \{v_1^{(k)}, v_2^{(k)} \ldots v_N^{(k)}\}$ is a codeword **then** exit the iteration loop ▷ syndrome check
End Iteration Loop

The threshold value τ is the optimizing factor. It may vary and depend on iteration, on VN or even be different for each CN neighbor in the same VN. Several works proposed to optimize τ by tracking the error probability of $v_{n,m}$ messages throughout the decoding process such as [1][31]. Gallager-A is a particular case when Gallager-B process with τ set as $\tau = d_v - 2$ all during decoding process.

The Gallager-A,B are shown to provide a good decoding performance. However, these decoder may require higher complexity due to the extrinsic computations.

2.3.2 Majority voting decoder

In the case that the Gallager-B decoder uses the threshold $\tau = 1$ all during the decoding process, the VN to CN message computation is only the majority voting rule and the initial value from channel is only used when $E_{n,m}^{(k)} = 0$. This type

of decoding algorithm is referred as Majority-Voting decoding. Majority-Voting decoder requires a lower complexity than Gallager-B since only the simple majority block is implemented in each VN. However, a small performance lost is observed.

2.3.3 Differential Decoders

Differential decoder with binary message passing (DD-BMP) was proposed by Mobini *et al.* in [32] and implemented by Cushon *et al.* in 2014 in [33]. DD-BMP is an extrinsic message passing decoder deploying memories to update the log-likelihood ratios (LLRs). Each edge in the Tanner graph is allocated a memory element. The DD-BMP algorithm operates as in algorithm 2. In the initialization, the LLR memory is initialized by the LLR value, γ_n, received from channel. The binary sent from VN n to CN m, $v_{n,m}^{(k)}$, in the initialization, $k = 0$, as well as during decoding process, $k > 0$, is the sign of the corresponding LLR memory $\tilde{\gamma}_n^{(k)}$ derived by the signum function, $sgn_r(.)$ where $sgn_r(0) = 1$. In each decoding iteration, similar to the Gallager-A, Gallager-B decoders, CN value $c_{m,n}^{(k)}$ is the extrinsic product of the incoming messages $v_{n',m}^{(k)}$, $\forall n' \in \mathcal{N}(m) \setminus n$. In each VN, the LLR memory value used for the next iteration, $\tilde{\gamma}_{n,m}^{(k+1)}$, is updated by the sum of the current value, $\tilde{\gamma}_{n,m}^{(k)}$, and the CN values extrinsic sum weighted by factor ω, $\omega.\sum_{m' \in \mathcal{N}(n) \setminus m} c_{m',n}^{(k)}$. The hard decision of each VN is defined by the majority vote of sign of its LLR memories and channel value. DD-BMP is shown to have a good decoding performance, comparable to Min-Sum algorithm in some particular LDPC codes. However, the DD-BMP complexity is dramatically increased compared to BF-based decoders due to the large allocated memory blocks. In each VN, d_v memory elements are required and are updated independently by the implemented adders. Also, in each CN, the extrinsic message is computed for each connected edge leading to higher complexity than CNs in the BF decoders.

In order to reduce the decoder complexity, authors in [33] proposed to use only 1 memory element in each VN and to send the same message to connected CNs. This modification eliminates the extrinsic feature in VN computation. The LLR memory update rule is modified as in Equ. 2.17 and the proposed decoder is called as modified differential decoding with binary message passing (MDD-BMP). The decoder complexity is shown to be significantly reduced with a small performance loss observed.

$$\tilde{\gamma}_n^{(k+1)} = \tilde{\gamma}_n^{(k)} + \omega \sum_{m \in \mathcal{N}(n)} c_{m,n}^{(k)} \tag{2.17}$$

Authors further proposed a modification in the updating function of MDD-BMP by adding/subtracting an adjusting factor d in the updated value of $\tilde{\gamma}_n^{(k+1)}$ as in Equ. 2.18 and named the decoder as improved differential binary (IDB). IDP added small complexity with gain in performance. In general, the DD-BMP and its variants provided a good decoding performance but are high in complexity [33].

$$\tilde{\gamma}_n^{(k+1)} = \tilde{\gamma}_n^{(k)} + \omega \sum_{m \in \mathcal{N}(n)} c_{m,n}^{(k)} - d * sgn_r(\tilde{\gamma}_n^{(k)}) \tag{2.18}$$

Algorithm 2 Differential Decoding with Binary Message Passing algorithm - (DD-BMP)

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet)▷ received word
Output: $\underline{\boldsymbol{v}} = (v_1, \ldots, v_N) \in \{0,1\}^N$ ▷ estimated codeword

Initialization

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

for all $n = 1, \ldots, N$ **do** $\tilde{\gamma}_n^{(0)} = \gamma_n$;

for all $n = 1, \ldots, N$ and $m \in \mathcal{N}(n)$ **do** $v_{n,m}^{(0)} = sgn_r(\tilde{\gamma}_n^{(0)})$;

Iteration Loop

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

$$c_{m,n}^{(k)} = \prod_{n' \in \mathcal{N}(m) \setminus n} v_{n',m}^{(k)}$$

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

$$\tilde{\gamma}_{n,m}^{(k+1)} = \tilde{\gamma}_{n,m}^{(k)} + \omega. \sum_{m' \in \mathcal{N}(n) \setminus m} c_{m',n}^{(k)};$$

$$v_{n,m}^{(k)} = sgn_r(\tilde{\gamma}_{n,m}^{(k)})$$

for all $n = 1, \ldots, N$ **do** ▷ **hard decision**

$$v_n^{(k)} = \begin{cases} 0, & \text{if} \quad sgn_r(\gamma_n) + \sum_{m \in \mathcal{N}(n)} sgn_r(\tilde{\gamma}_{m,n}^{(k)}) \geq 0 \\ 1, & \text{otherwise.} \end{cases};$$

if $\underline{\boldsymbol{v}}^{(k)} = \{v_1^{(k)}, v_2^{(k)} \ldots v_N^{(k)}\}$ is a codeword **then** exit the iteration loop ▷ syndrome check

End Iteration Loop

2.4 The noise-aided BF decoders

In 2014, two modifications of GDBF decoder were proposed which provided the best error correction ever of hard decision decoders and were shown to approach the soft decision decoders. The two proposed method shared the same concept that introduces the perturbation in VN computation. The Noisy GDBF (NGDBF) [14] decoder is applied on AWGN channel and Probabilistic GDBF (PGDBF) [15] is applied on BSC. We denote these modifications noise aided GDBF decoders as NA-GDBF.

$sgn_r(0) = 1$.

2.4.1 Noisy Gradient Descent Bit-Flipping decoding algorithm

The NGDBF modified the energy function described in Equ. 2.19. In NGDBF, all parity check values are weighted by the same weight factor ω^{NGDBF} which is a real number and is optimized empirically. The speciality of NGDBF comes from the random-added value ς which is the normal-distributed random value with equal variant of channel noise σ. Although the NGDBF requires a channel noise estimator (to evaluate σ) which required an additional complexity, its decoding performance is considerably improved and is comparable to MS decoder [14].

$$E_n^{(k)} = (1 - 2v_n^{(k)})y_n + \omega^{NGDBF}. \sum_{m \in \mathcal{N}(n)} (1 - 2\, c_m^{(k)}) + \varsigma \tag{2.19}$$

NGDBF also comes up with the single mode in which the bit that has the smallest energy will be flipped (denoted as S-NGDBF). The multiple mode of NGDBF is denoted as M-NGDBF in which the threshold τ is adapted from one iteration to another as in ATBF [11]. In M-NGDBF, there is possibility that a bit has a small value of energy even if its connected CNs are satisfied due to the random added value, ς, to the energy value. This bit is flipped unintendedly. To avoid this negative effect, authors of [14] proposed to proceed a majority vote on the value of VN in decoding iterations at the end of decoding process whenever the decoder reaches the maximum iteration without finding the correct codeword. This operation is called as *smoothing operation* and this NGDBF decoder is denoted as SM-NGDBF.

It can be seen in Figure 2.10 that NGDBF decoders provided a very promising results. M-NGDBF was equivalent to M-esc-GDBF and MS decoder with $It_{max} = 5$. SM-NGDBF was even better than M-NGDBF and became the best BF-based decoders in AWGN.

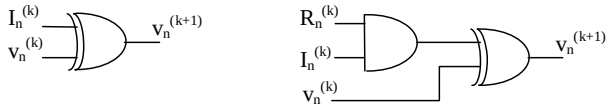
2.4.2 Probabilistic Gradient Descent Bit-Flipping decoding algorithm

In GDBF decoder working on BSC channel, the energy value of each VN is computed as in Equ. 2.9 and is integer-valued, varies from 0 to $(d_v + 1)$, and the bits which have the maximum value $E_{max}^{(k)} = \max_n(E_n^{(k)})$ are flipped. Due to the integer representation of the energy function, many bits are likely to have the maximum energy, leading to the parallel (multiple) flip mode. Let us use an indicator variable to indicate the VNs which have the maximum energy at iteration k, i.e. $I_n^{(k)} = 1$ if $E_n^{(k)} = E_{max}^{(k)}$, and $I_n^{(k)} = 0$ otherwise. The fact that the number of bits to be flipped cannot be precisely controlled, induces a negative impact to the convergence of the GDBF, as the analysis of [15] shows. To avoid this effect, the authors in [15] proposed the PGDBF algorithm in which, instead of flipping all the bits with maximum energy function value, only a random fraction of those bits are flipped. The random fraction is fixed to a pre-defined value $p_n^{(k)}$, which could be different for each VN and each iteration. In this work, we restrict ourself to the case for which $p_n^{(k)}$ are constant for all iterations and all VNs, denoted $p_0 \in [0, 1]$ hereafter.

The flipping set of PGDBF is defined as in Equ. 2.20 where $p_n^{(k)}$ is the realization of a uniform random variable over the interval [0,1]. PGDBF offers the best error correction compared to all BF-based decoders on BSC channel (see Figure 2.11). However, the required additional complexity for generating the randomness is too large (see Figure) and becomes the bottleneck of PGDBF decoder.

$$\mathcal{B}_{PGDBF}^{(k)} = \{n \in \mathcal{B}_{GDBF}^{(k)} | p_n^{(k)} < p_0\} \tag{2.20}$$

The large additional complexity of PGDBF comes from the probabilistic signal generator. Indeed, the main difference between GDBF and PGDBF is only the probabilistic signal block. PGDBF can be implemented using a sequence of N random bits, generated following a Bernoulli distribution with parameter p_0, with different realizations at each iteration. We denote the random signal (RS) sequence at the k-th iteration by $R^{(k)} = \{R_n^{(k)} | 1 \leq n \leq N\}$ in which the random signal $R_n^{(k)}$ is triggered correspondingly to VN n-th. Other small difference in flipping implementation between GDBF and PGDBF is the added AND gate described in Figure 2.12. In the GDBF algorithm, a VN $v_n^{(k)}$ at iteration k is flipped (is XOR-ed by '1') when its energy function is a maximum (the indicator variable is $I_n^{(k)} = 1$) while in PGDBF, a VN $v_n^{(k)}$ is flipped *if and only if* the two conditions $I_n^{(k)} = 1$ and $R_n^{(k)} = 1$, are both satisfied. One of our naïve implementation of PGDBF [29] shows that PGDBF requires more than 8 times the complexity of GDBF decoder.



GDBF:
$I_n^{(k)} = 1$ if $E_n^{(k)} = E_{max}^{(k)}$ else $I_n^{(k)} = 0$

PGDBF:
$I_n^{(k)} = 1$ if $E_n^{(k)} = E_{max}^{(k)}$ else $I_n^{(k)} = 0$.
$R_n^{(k)} = 1$ with probability of p_0

Figure 2.12: The difference in the flipping operator between GDBF and PGDBF algorithms.

It is remarked that the perturbation in BF decoding process helps greatly to improve the decoding performance, approach or even surpass soft decision decoders. Indeed, in AWGN, with a random variable added in computed energy value, the versions of NGDBF provided the best error correction compared to all introduced BF decoders and are even comparable to MS algorithm. In BSC, PGDBF decoder is also the best BF decoder ever in BSC channel. All the advantages in error correction were observed through simulation. **The need of understanding the effect of perturbation on decoding performance is critical** due to twofold. First, the understanding reveals the perturbation optimization strategy as well as direct to optimized realization architecture which guarantees the maximum decoding gain at the minimum hardware cost. Second, the understanding suggest to design the faulty tolerant decoders. However, **there was no analytic method to analyse the perturbation effect on BF decoders.**

2.5 Hardware complexity of BF-based decoders

In order to make more comparison of the proposed BF decoders, we present in Tabble 2.1 the computation operations of some typical BF decoders. For being fair in comparison, we assume all BF decoders perform 1 iteration in 1 clock cycles, the memories are only to store the channel soft value, the adaptive threshold and the tentative hard decision codeword, the rest of the decoders is combinatory logic. We count all addition (ADD), multiplication (MUL) and comparison (COM) modules needed to implement the VN operations for each type of BF decoders. We also count the number of comparison module for the global sort in single flip decoders. Some BF-based decoders which require the initialization computation block, *i.e.* for the channel soft information computation before the BF decoders start decoding, we count, therefore, the number of addition (ADD), multiplication (MUL) and comparison (COM) modules need for this block. Note that, since the multiplication implementation is more complex compared to the addition, in MWBF and IMWBF, the element $\alpha|y_n|$ should be proceeded at the channel output scale processing step to avoid multiplication implementation. The results are shown in the Table 2.1. It can be seen that the adaptive threshold decoder such as ATBF, M-NGDBF, SM-NGDBF have no operation for the global sorting but the require N multiplication blocks to update the adaptive thresholds. Also, these decoders need $2N$ real memory elements to store the channel soft values and adaptive thresholds while other decoders require only N.

	VN computation			Comparison for	Start-up initialization			Memory		Additional block
	ADD	MUL	COM	the global sorting	ADD	MUL	COM	real-value	1-bit	
WBF [2]	$N(d_v-1)$	0	N	$N-1$	0	0	$M(d_c-1)$	N	N	-
MWBF [3]	$N.d_v$	0	N	$N-1$	0	0	$M(d_c-1)$	N	N	-
IMWBF [5]	$N.d_v$	0	N	$N-1$	0	0	$M(d_c-2).d_c$	N	N	-
S-GDBF [8]	$N^{(1)}$	0	N	$N-1$	0	0	0	N	N	-
M-GDBF [8]	$N^{(1)}$	0	$2N^{(2)}$	$N-1$	0	0	0	N	N	A global objective function computation
M-GDBF with escape [8]	$N^{(1)}$	0	$2N^{(3)}$	$N-1$	0	0	0	N	N	A global objective function computation
S-RRWGDBF [10]	$N.d_v$	0	N	$N-1$	$M(d_c-1)$	$M.d_v$	0	N	N	-
M-RRWGDBF [10]	$N.d_v$	0	$2N^{(2)}$	$N-1$	$M(d_c-1)$	$M.d_v$	0	N	N	A global objective function computation
ATBF [11]	$N.d_v$	N	N	0	0	0	0	$2N$	N	-
S-NGDBF [14]	$2N$	0	N	$N-1$	0	0	0	2N	N	a $RRG^{(4)}$
M-NGDBF [14]	$2N$	N	N	0	0	0	0	2N	N	a RRG
SM-NGDBF [14]	$2N$	N	N	0	0	0	0	2N	N	a RRG and N 0-to-$(log_2(It_{max})+1)$ counters

Table 2.1: The complexity of some typical hard decision decoders.

By taking into account the decoding performance, we plot in Figure 2.13 the performance-complexity comparison between some typical BF decoders on the PE-

(1) Summation between a real value with an integer.

(2) Comparison between the computed energy with either parallel flip threshold $\tau_n^{(k)}$ in parallel flip mode or the minimum value sorted by the minimum sorter in serial flip mode (equivalent to 2 single comparators and a 2-to-1 binary multiplexer in each VN).

(3) Comparison between the computed energy with either parallel flip threshold $\tau_n^{(k)}$ and $\tilde{\tau}_n^{(k)}$ in parallel flip mode or the minimum value sorted by the minimum sorter in serial flip mode (equivalent to 2 single comparators, a 2-to-1 real multiplexer and a 2-to-1 binary multiplexer in each VN).

(4) *RRG*: Real-value random generator.

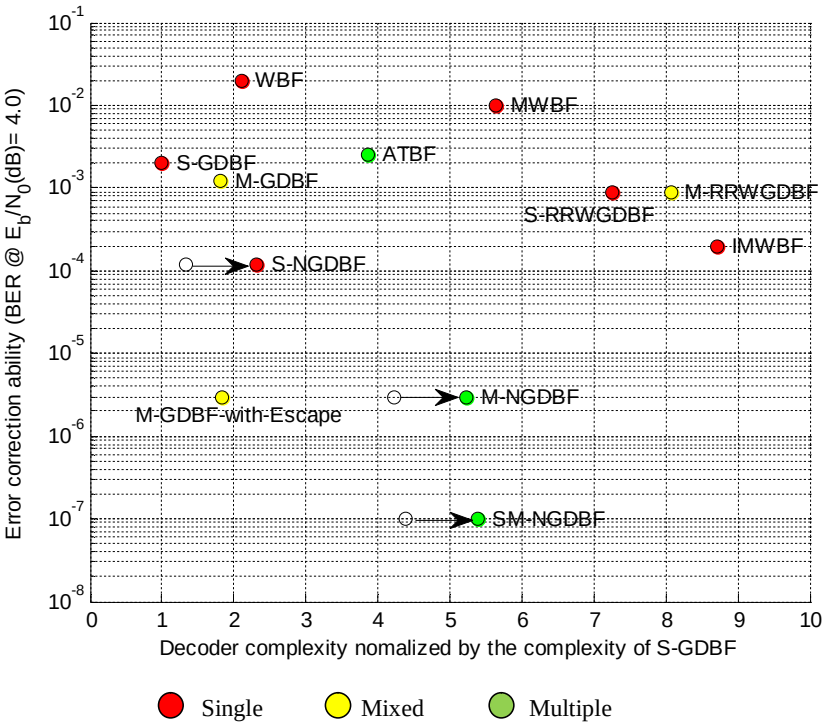
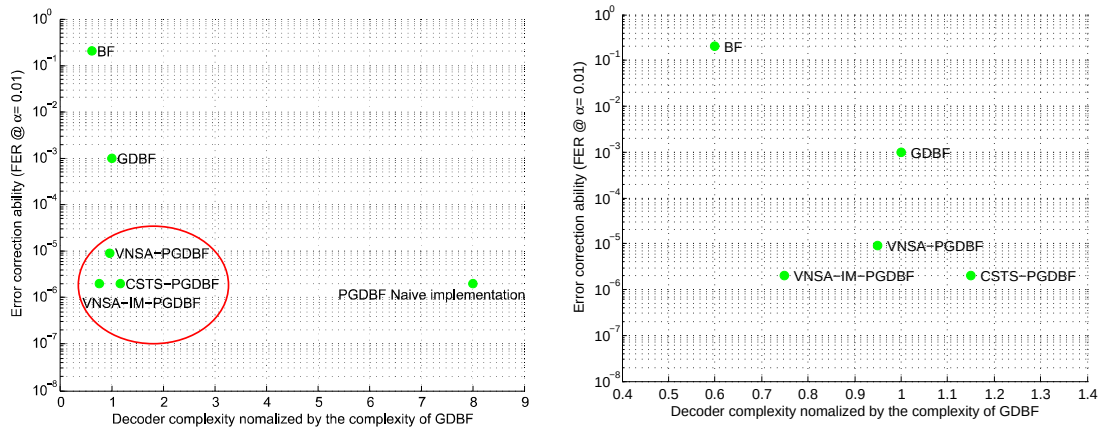


Figure 2.13: Performance-complexity comparison of some typical BF-based decoders on the PEGReg504x1008, regular $d_v = 3$, $d_c = 6$ LDPC code.

GReg504x1008, regular $d_v = 3$, $d_c = 6$ LDPC code. Note that, because the cost of the real-number random generator (RG) was not evaluated, we plot the complexity of NGDBF without RG implemented and relatively shift to the right by a period representing the additional hardware of RG (see the arrows in Figure 2.13). It can be seen that the S-GDBF requires the smallest hardware to implement while the type of RRWGDBF and IMWBF are most heavy decoders with relatively equivalent performance. The SM-NGDBF provides the best in error correction with 4 decades gain compared to S-GDBF but is 4.3 times more in decoder complexity. It is interesting that, with a small modification between M-GDBF and M-GDBF with Escape, a negligible complexity required but a large gain in performance obtained. This interesting remark is also observed in M-NGDBF and SM-NGDBF.

On performance-complexity comparison of the BF-based decoders on BSC channel (Figure 2.14a), GDBF provides 2 dB in correction gain compared to BF decoders while requiring 40% additional hardware complexity. Especially, the PGDBF with naïve implementation provides 3 dB gain in error correction compared to GDBF while it needs 8 times complexity of GDBF to implement. We propose several implementations in this work (marked in the red circle) which provide an equivalent error correction ability of PGDBF with naïve implementation while requiring the equivalent complexity of GDBF. More specially, some of our architecture have even smaller complexity than GDBF decoder (see Figure 2.14(b)).



(a) (b)

Figure 2.14: Performance-complexity comparison of some typical BF-based decoders on BSC channel for the regular $d_v = 3$, $d_c = 6$, $M = 648$, $N = 1296$ QC-LDPC code. The PGDBF implementations in the red cycle are the one proposed in this thesis.

2.6 Conclusion

In the next chapters, we propose a novel method called Finite State Tracking (FST) method to analyse the hard decision decoders on the BSC channel model. This proposed method is, in general, applicable for all BF decoders and more efficient to analyse the decoder with probabilistic feature. With this new analytic method, one can formulate and compute the decoding error probability. Furthermore, many decoding behavior of PGDBF such as the improvement of re-initialization can be clearly explained by FST. For the implementation of PGDBF, we propose an optimized architecture to implement the probabilistic signal generator as well as the PGDBF decoder called Cyclic Shift Truncated Sequence PGDBF (CSTS-PGDBF). The proposed CSTS-PGDBF requires a negligible additional hardware compared to GDBF while preserves the decoding as good as the theoretical PGDBF. We further propose an efficient architecture for LDPC decoding on QC-LDPC code called Variable Node Shift Architecture (VNSA). Applying VNSA for PGDBF implementation called as VNSA-PGDBF, the decoding performance is maintained as theoretical PGDBF while the complexity is even smaller than GDBF. A further simplified version of VNSA-PGDBF called VNSA-IM-PGDBF reduces even more in complexity and approaches to the complexity of the simplest BF decoder. These advantages in area, throughput and decoding performance make our PGDBF decoders a competitive hard-decision LDPC decoding solution for current and future standards.

Chapter 3

Theoretical analysis of Probabilistic Gradient Descent Bit Flipping

3.1 Introduction

In this chapter, we introduce an analysis method for hard decision decoders denoted as Finite State Tracking (FST). Although FST is shown to be able to apply on different type of hard decision decoders, we limit the FST presentation in this chapter only on PGDBF decoder. FST represents the PGDBF decoding process from an iteration to another as a state transition in a Markov Chain (MC). A state of PGDBF decoding is represented as a state in this MC and by analyzing the MC, the closed-form expression of Frame Error Rate as a function of number of iteration can be derived. The uncorrectable and partial-uncorrectable error pattern definitions are newly introduced in this chapter in order to indicate correspondingly the error patterns that can not be corrected and those which can be corrected with some probability by PGDBF. The uncorrectable and partial-uncorrectable error pattern can be also determined by analyzing the MC. We illustrate the utilization of FST by analyzing the performance of PGDBF decoder and compare it with other BF decoders. It is firstly shown that the number of uncorrectable error patterns of PGDBF on the tested LDPC code is significantly reduced compared to GDBF and conventional BF decoders. The predictive performance curves on the error floor are then derived, based one the estimated number of uncorrectable and partially uncorrectable error patterns and are finally confirmed by the simulation results. The improvement of PGDBF is clearly explained by showing that many transitions leading to the zero-error state in PGDBF do not appear in deterministic GDBF. Furthermore, FST provably shows that for some given error patterns, the PGDBF can converge to the zero-error state only if some specific transition occur. Getting into these transitions depends on the realization of the random signal. This explains the decoding gain phenomenon of re-initialization (or restarting) of the PGDBF decoding from the beginning with new random realizations.

3.2 Markov Chain representation of the decoding process

3.2.1 Markov Chain of hard decision decoding process

The principle behind the PGDBF improvement is to perturb the dynamics of the decoder by randomly modifying the flipping sets as in Equ. 2.20. Also, PGDBF is shown to be able to correct the error patterns where error bits are allocated on support of a subgraph called trapping set while the deterministic GDBF could not correct. The proposed method will consider and analyse the behavior of PGDBF decoder on this subgraph.

Let consider a subgraph g composed by L VNs and $\boldsymbol{v}^{(k)} = \{v_\ell^{(k)}\}_{[\ell=1\ldots L]}$ be the ordered set of all values of VNs at iteration k and define a dynamical system expressed by Equ. 3.1 where function Υ is the updating function of deterministic GDBF decoding algorithm and the binary vector $\mathbf{r}^{(k)}$ of length L is the realization of the probabilistic part affecting on the values of $\boldsymbol{v}^{(k)}$.

$$\boldsymbol{v}^{(k)} = \Upsilon(\boldsymbol{v}^{(k-1)}) \oplus \mathbf{r}^{(k)} = \tilde{\Upsilon}(\boldsymbol{v}^{(k-1)}) \tag{3.1}$$

At iteration k, all elements ℓ of $\mathbf{r}^{(k)}$, $(\ell = 1 \ldots L)$, that $E_\ell^{(k)} = E_{max}^{(k)}$, are deterministic functions of Bernoulli distribution random variables such that, if $E_\ell^{(k)} = E_{max}^{(k)}$, $r_\ell^{(k)} = 0$ with $p = p_0$ and $r_\ell^{(k)} = 1$ with $p = 1 - p_0$, otherwise $r_\ell^{(k)} = 0$. The random process $\{\boldsymbol{v}^{(\ell)}\}_{\ell \geq 0}$ is a Markov chains in finite state spaces $\{0,1\}^L$.

We define $\boldsymbol{e} = \{e_\ell\}_{[\ell=1\ldots L]}$ as the error pattern affecting on the part of the codeword $\boldsymbol{x} = \{x_\ell\}_{[\ell=1\ldots L]}$ induced by subgraph g to form $\boldsymbol{v}^{(0)}$ such that $\boldsymbol{v}^{(0)} = \boldsymbol{x} \oplus \boldsymbol{e}$. When all zero codeword is sent, $\boldsymbol{v}^{(0)} = \boldsymbol{e}$.

For a given subgraph g and error pattern $\boldsymbol{e}$, the formed Markov chain is denoted as $\mathcal{M}_{\boldsymbol{e}}$ in the state space $S = \{0,1\}^L$, and corresponds to the transition probability matrix $P = (p_{\varepsilon,\delta})_{\varepsilon,\delta \in S}$ where the transition probabilities $p_{\varepsilon,\delta} = \mathtt{Pr}\{\boldsymbol{v}^{(k)} = \delta | \boldsymbol{v}^{(k-1)} = \varepsilon\}$.

Given a subgraph g, an error pattern $\boldsymbol{e}$ and a state $\varepsilon \in S$ (corresponding to the iteration k), we define the function $d'(\varepsilon, \boldsymbol{e}, g) = \sum_{\ell=1\ldots L} \mathbb{1}(E_\ell^{(k)} = E_{max}^{(k)})$. $d'(\varepsilon, \boldsymbol{e}, g)$ counts the number of bits that have the maximum energy and are the flip candidates. These flip candidates are flipped automatically in GDBF while only a random part are flipped in PGDBF. In PGDBF, given $d'(\varepsilon, \boldsymbol{e}, g)$ flip candidates, there are $2^{d'(\varepsilon,\boldsymbol{e},g)}$ flip possibilities corresponding to $2^{d'(\varepsilon,\boldsymbol{e},g)}$ transitions from a state ε to $2^{d'(\varepsilon,\boldsymbol{e},g)}$ "next states" in $\mathcal{M}_{\boldsymbol{e}}$. These next state set is denoted as $S^{(k+1)}$, $|S^{(k+1)}| = 2^{d'(\varepsilon,\boldsymbol{e},g)}$. Let δ be the state of these next states reached from the state ε ($\delta \in S^{(k+1)}$), then

$$p_{\varepsilon,\delta} = p_0^{d_{\varepsilon,\delta}} (1 - p_0)^{d'(\varepsilon,\boldsymbol{e},g) - d_{\varepsilon,\delta}} \tag{3.2}$$

where $d_{\varepsilon,\delta}$ is the Hamming distance between the binary vectors ε and δ. For any state ε, $\sum_{\forall \delta \in S^{(k+1)}} p_{\varepsilon,\delta} = 1$. The state space S of Markov Chain $\mathcal{M}_{\boldsymbol{e}}$ is composed as $S = \{S^{(k)}\}_{k \geq 0}$. In GDBF, $S^{(k+1)} = 1$ since from a given state ε, there is only the possibility to flip all candidates to form next state δ, $p_{\varepsilon,\delta} = 1$. For graphical

presentation in the next section, we describe the state $\boldsymbol{v}^{(k)}$ in the form of S_i such that $i = \sum_{\ell=1}^{L} 2^{\ell-1} v_\ell$ and the initial state as S_e, $S_e = S_j$ where $j = \sum_{\ell=1}^{L} 2^{\ell-1} e_\ell$.

3.2.2 Markov chain representation: GDBF and PGDBF illustrations

This section aims to illustrate our representation of decoding process by the Markov chain. The GDBF, PGDBF decoding process are used to serve in the examples. These examples are also selectively chosen to highlight the property of PGDBF over GDBF such as (1). Both PGDBF and GDBF correct the errors but the probabilistic part slows the converging speed of PGDBF by visiting the intermediate states (with weight-1 or weight-2 error patterns, weight-1 error pattern has 1 bit in error). (2). GDBF fails to correct while PGDBF corrects all errors (with weight-3 error pattern) and (3). GDBF again fails to correct while PGDBF can correct errors with some probability (with weight-3 error pattern). We use the Tanner code ($N = 155, M = 93$) and ($d_v = 3, d_c = 5$) and the error patterns used are the one that error bits are located in a subgraph called Trapping Set TS(5,3) as in figure 3.1. We assume the all-zero codeword is sent. We represent the white (black) circles as the correct (erroneous) VNs and white (black) squares as the satisfied (unsatisfied) CNs in the subgraphs. For the comparison on the Markov chain of PGDBF and GDBF, the state transitions in Markov chain of PGDBF decoder are shown as solid black arrows along with those of GDBF as dashed red arrows respectively.

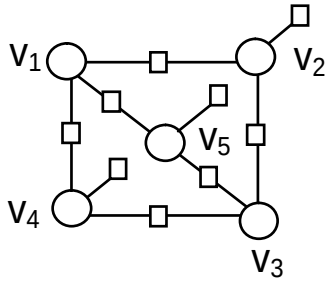


Figure 3.1: Trapping Set TS(5,3) in the Tanner Code.

3.2.2.1 Error patterns weight-1 and weight-2

When there is 1 error happened, there are 3 unsatisfied CNs which connect to the error VN. This erroneous VN then has the energy 3 while all VN which connect to the unsatisfied CNs have energy 1 and all the other VNs are 0. The girth of Tanner code (the minimum cycle in the Tanner graph) is $g = 8$, so there is no VN connecting to more than 1 unsatisfied CNs. Therefore, the erroneous VN is the energy maximum VN and is the only flipping candidate. Let assume VN v_1 in the TS(5,3) is in error shown in Figure 3.2a, forming the error pattern $\boldsymbol{e} = e_5 e_4 e_3 e_2 e_1 = 00001$ (the initial state $S_{\boldsymbol{e}} = S_1$). The GDBF will flip this flipping candidate and decoding process

stop while PGDBF either flips this flipping candidate with probability p_0 or does not flip with $1 - p_0$. The PGDBF eventually flips this VN since the probability of not flip v_1 is $(1 - p_0)^\ell \to 0$ when $\ell \to \infty$.

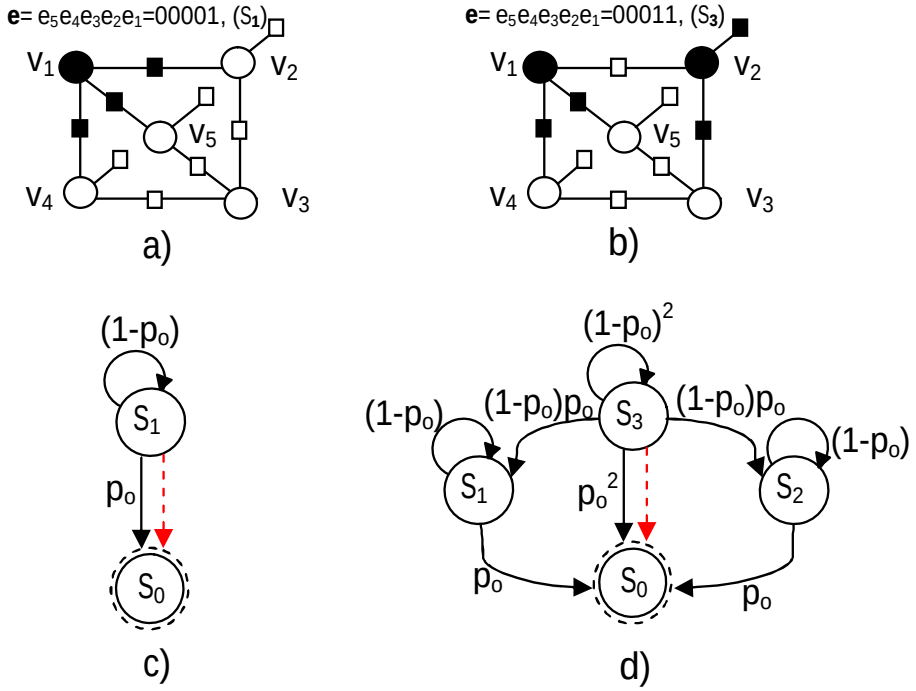


Figure 3.2: The weight-1 (a), weight-2 (b) error patterns and the corresponding Markov chain representation of GDBF and PGDBF decoders. The dashed red arrows are the transitions of GDBF, the solid red arrows are the transitions of PGDBF.

The Markov chain of PGDBF and GDBF decoders for this error pattern are shown in Figure 3.2c. The error correction behavior of GDBF is described as the state transition in the Markov chain as following. The next state of the initial state S_e is determined by applying the deterministic function $\Upsilon(S_e) = 00000 = S_0$. The Markov chain of GDBF decoder moves from the initial state $S_e = S_1$ to the next state S_0. The stopping condition of the Markov chain is corresponding to the satisfaction of syndrome check. Since the state S_0 in the Markov chain corresponds to value $'0'$ to all VNs which lead to satisfied syndrome check, the state transition is halted, therefore, there is no transition from S_0 to other states in the Markov chain. S_0 is called *converging state* and marked by the second dashed circles in figure 3.2c. In the Markov chain of PGDBF decoder, $S^{(1)} = \tilde{\Upsilon}(S_1) = \{S_0, S_1\}$, there are two possibilities of transitions from the initial state S_1: to move to converging state S_0 with probability p_0 or to stay in S_1 with probability $(1 - p_0)$. The Markov chain stops when it is on the state S_0 and continues to move when it is in S_1. It eventually stops due to the fact that probability to be at S_1 of the Markov chain, $(1-p)^\ell \to 0$, $\ell \to \infty$.

When there are 2 erroneous bits happened, these 2 bits could either share 1 CN (Figure 3.3a) or separate (do not share any common CN, Figure 3.3b).

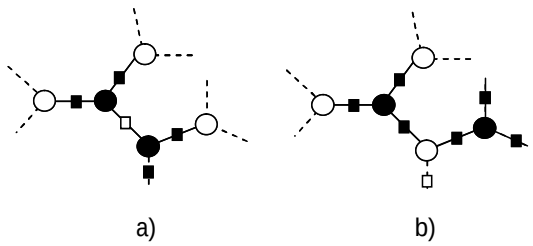


Figure 3.3: 2 erroneous bits located in Tanner graph of Tanner code. The 2 erroneous bits are either a). sharing a CN or b). separating from each other

It can be seen that in both cases, the 2 bits are the energy maximum VNs and are flipping candidates. Flipping any 1 of 2 erroneous bits does not create any new flipping candidate. Without loss of generality, let assume 2 erroneous bits be v_1 and v_2 in the TS(5,3) in Figure 3.2b ($S_e = S_3$). With these 2 flipping candidates, the GDBF decoder certainly flips both of them, leading to all-zero VNs. The GDBF decoder stops after 1 iteration. In the corresponding Markov chain (Figure 3.2d), the 2 erroneous bits form initial state S_3. The chain moves from S_3 to $\Upsilon(S_3) = S_0$ (the dashed red transition) then stops at this converging state. The PGDBF decoder, with 2 flipping candidates and due to the flipping randomness, has $2^2 = 4$ possibilities: 1. flips all of the candidates with probability of p_0^2, then has all-zero VNs and stops, 2. does not flip any bit with probability of $(1-p_0)^2$, stays with the same 2 erroneous bits and in the next iteration, the decoder has again 4 possibilities to proceed, 3. flips v_1; and v_2 is still in error with probability of $p_0(1-p_0)$, 4. flips v_2; and v_1 is still in error with probability of $(1-p_0)p_0$. In the 2 later cases, the decoder eventually converges similarly to the previous example with 1 error happened. This PGDBF decoder behavior can be expressed by the Markov chain in figure 3.2d (with solid back transitions). The next states from the initial state S_3 can be determined by the probabilistic function $\tilde{\Upsilon}(S_3) = \{S_0, S_1, S_2, S_3\}$. Thus, there are 4 possibilities with the corresponding probability$\{p_0^2, p_0(1-p_0), (1-p_0)p_0, (1-p_0)^2\}$. Similarly, by expressing all states $\tilde{\Upsilon}(S_0) = \{\emptyset\}$, $\tilde{\Upsilon}(S_1) = \{S_0, S_1\}$, $\tilde{\Upsilon}(S_2) = \{S_0, S_2\}$, $\tilde{\Upsilon}(S_3) = \{S_0, S_1, S_2, S_3\}$, it forms the Markov chain with determinable transitions probabilities s in Figure 3.2d.

It can be seen that in weight-1 and weight-2 error patterns, both GDBF and PGDBF will finally converge. However, PGDBF has more states leading to slower convergence as shown in [15].

3.2.2.2 Weight-3 error pattern

Next, we show an example that PGDBF can correct the error pattern with 3 bits in error while GDBF could not. We consider the example shown in Figure 3.4a in which $\{v_1, v_3, v_5\}$ are in error ($\{v_1, v_2, v_3\}$ and $\{v_1, v_3, v_4\}$ produce the same behavior). The 3 erroneous bits located in Figure 3.4 form the initial state $S_e = S_{21}$.

For GDBF decoding process, the fact that $\Upsilon(S_{21}) = S_{26}$ and $\Upsilon(S_{26}) = S_{21}$ shows the oscillation in the Markov chain between these 2 states as in Figure 3.4c (the dashed red arrows). There is no (dashed red) transition to the converging state S_0,

the GDBF decoder, therefore, fails to correct the error pattern regardless of number of iteration.

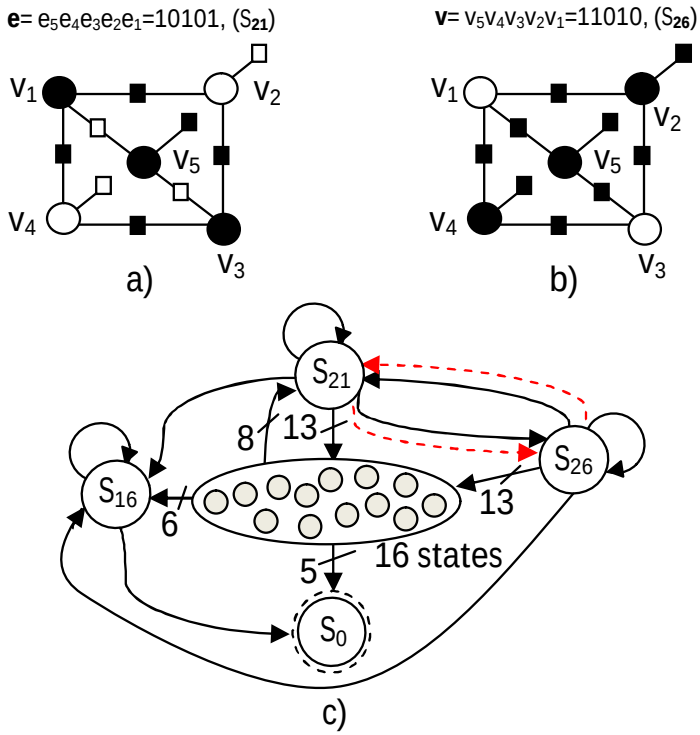


Figure 3.4: The weight-3 error pattern and the corresponding Markov chain representation of GDBF and PGDBF decoders. The dashed red arrows are the transitions of GDBF, the solid red arrows are the transitions of PGDBF.

For PGDBF, there are 16 possibilities of transitions from the initial state S_{21} due to $|\tilde{\Upsilon}(S_{21})| = 16$. The Markov chain can follow $S_{21} \to S_{16} \to S_0$ as the shortest converging path as described in Figure 3.4c. Also, in order to be at S_0, there are 5 others states in the set of 20 total states of S in the Markov chain leading to S_0. Despite non detailed transition shown in Figure 3.4, the fact that from any state $S_i \in S, S_i \neq S_0$, $S_i \to S_0$ and the Markov chain stops only at S_0, the Markov chain is eventually at the converging state. The speed to S_0 obviously depends on the transition paths. This example shows that, by deliberating randomness in transitions, it offers the converging possibilities which is not the case of deterministic decoder.

3.2.2.3 Weight-4 error pattern

We illustrate another example of Markov chain representation of PGDBF decoding with weight-4 error pattern with which GDBF fails to correct the errors while PGDBF can correct with a probability. The state transitions of PGDBF decoding are illustrated in Figure 3.5 (some states are removed to ease the discussion). It can be seen firstly that from the initial state, S_{58}, there are the transitions leading to the

converging state S_0. PGDBF, therefore, can correct the given error pattern. There are, however, 3 special states, marked by the dashed cycles, that whenever the state get into this group, it never get out of them. In other words, the transitions will be locally in these 3 states and will never reach to converging state S_0. We name this group as the *isolated group* or *absorbing group*. With $p_0 = 0.7$, the probability of getting into this isolated group, p_e, is evaluated by FST at $p_e = 0.485$ which means PGDBF fails to correct the error pattern at around 50% the cases.

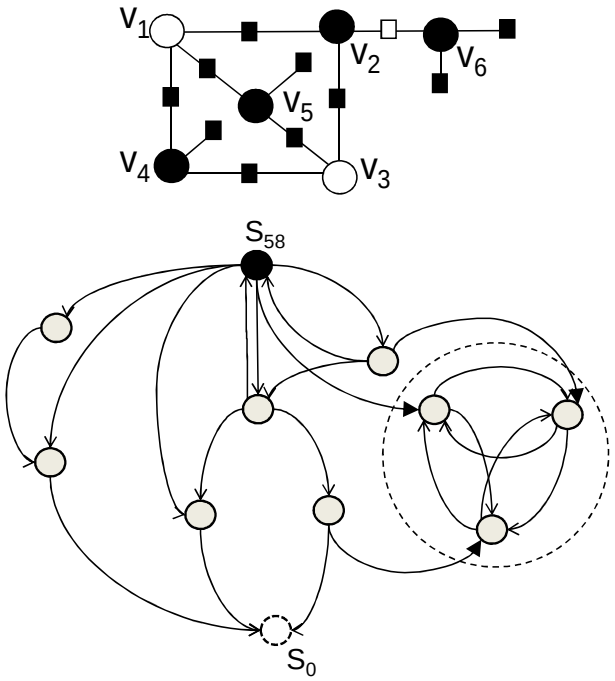


Figure 3.5: The weight-4 error pattern and the corresponding Markov chain representation of PGDBF decoder.

3.3 Frame Error Rate Evaluation

3.3.1 Markov Chain, algebraic and graph-theoretic considers

Suppose that $\mathcal{M}$ is a homogenous finite-state Markov chain with the state space S and the transition probability matrix $P = p_{\varepsilon,\delta}$. The probability that $\mathcal{M}$ proceeds from state ε to state δ after k transitions is denoted by $p^k_{\varepsilon,\delta}$ and the k steps transitions probability matrix $P^{(k)} = [p^k_{\varepsilon,\delta}]$ is computed as $P^{(k)} = P^k$.

Let $G_{\mathcal{M}} = (S, E)$ be a directed graph associated with the Markov chain $\mathcal{M}$ having the set of nodes S and a set of edges E. Each node corresponds to a state in S and $G_{\mathcal{M}}$ contains edge $(\varepsilon, \delta) \in E$ if and only if $p_{\varepsilon,\delta} > 0$. The quantity $p^k_{\varepsilon,\delta} > 0$ means there is a directed path Q of length $l(Q) = k$ from node ε to node δ in $G_{\mathcal{M}}$. The node δ is said to be reachable from ε if $p^k_{\varepsilon,\delta} > 0$ holds for some $k \geq 0$ (written as $\varepsilon \to \delta$). If there is no path in $G_{\mathcal{M}}$ between ε and δ then $\varepsilon \not\to \delta$. If both $\varepsilon \to \delta$ and $\delta \to \varepsilon$ then it is said that ε and δ *communicate* (written as $\varepsilon \leftrightarrow \delta$). A *circuit* is a path joining a node to itself and if there is no repeat node in this circuit, it forms a *cycle*.

A *strongly connected subgraph* (or a strong component) of $G_{\mathcal{M}}$ is induced by a subset of nodes S_s such that for all $\varepsilon, \delta \in S_s$ then $\varepsilon \leftrightarrow \delta$. A *condensed* or *reduced* graph $\hat{G}_{\mathcal{M}}$ is formed by grouping all of nodes in a strong connected subgraph into 1 new "supernode". It can be seen that there is no path connecting two arbitrary nodes in a strong component in which there exists a node not including in this strong connected subgraph. For this reason, there is no cycle in the condensed graph $\hat{G}$ which governs the supernodes of a graph $G_{\mathcal{M}}$. Identifying the strong connected subgraph and creating the condensed $\hat{G}$ of G can done efficiently by Depth-First Search Algorithm [34].

3.3.2 Classification of the states

We denote above the converging state S_0 which is the state where all parity check are satisfied and all VN decisions form all-zero codeword. S_e is the initial state formed by the error pattern. We denote heres $S_{\sim 0}$ as the set of states for which all parity checks are satisfied, and the variable node decisions form a non-zero codeword. The set $S_{\sim \mathcal{C}}$ includes all states for which the variable node decisions are not codewords. Thus, the above three disjoint sets partition the set of states $S = S_0 \cup S_{\sim 0} \cup S_{\sim \mathcal{C}}$.

In general, given a BF decoder with an error vector $\boldsymbol{e}$, the associate Markov chain $\mathcal{M}_e$ contains all the possible states could be of the decoder induced from the initial state S_e. More precisely, all states in $\mathcal{M}_e$ satisfy that $\forall S_i \in S$ then $S_e \rightarrow S_i$. The cardinality of S could be smaller than 2^L due to the fact that there exists the state (or states) in $\{0,1\}^L$ but never appears in S. We show in the section 3.4 that by considering the presence or absence of some states in S, it can reveal the error correction properties of the decoder.

We further categorize the states in $S_{\sim \mathcal{C}}$ as either *transient* or *recurrent*. A state is a recurrent state once start from it, will return to that state with probability of 1. On the contrary, for a transient state, there exists a positive probability that the chain will never to it. By using the graph-theoretic concepts, ε is a transient state if there exists at least one state δ for which $\varepsilon \rightarrow \delta$ but $\delta \not\rightarrow \varepsilon$, otherwise, ε is a recurrent state. For an example, in the weight-4 error pattern illustration in Section 3.2, 3 states marked in dashed cycle are the recurrent states, all other states are transient states. It can be shown that the descendants of a recurrent state are also the recurrent states and all together, they form a strongly connected component in $\mathcal{M}_e$. More precisely, all states in this strong component do not reach to any other state outside (strong *close* component). In fact, if they did, they were not recurrent states.

The fact that a recurrent state and its descendants form a strong close component attracts our consideration. When the Markov chain is in a state of this strong close component, it continues to move to the next state (since the syndrome is unsatisfied). However, the Markov chain is always in the states of strong component due to the closeness and never to the converging state. It is a kind of absorbing in the Markov chain and by tracking this strong close component appearance and/or probability being in its of the Markov chain, it reveals the the decoding failure and/or failure

probability.

We denote T as the set of all transient states and $R = S_{\sim\mathcal{C}} \setminus T$ as the set of recurrent nodes. Moreover, let $R_1, R_2, ..., R_r$ ($R_1 \cup R_2 \cup ... \cup R_r = R$) are the subsets of R which form r strong components in $\mathcal{M}_e$ then,

$$S = S_0 \cup S_{\sim 0} \cup S_T \cup S_{R_1} \cup ... \cup S_{R_r}. \tag{3.3}$$

During the decoding process, the syndrome check is operated after each iteration, and if the Markov chain is in the state $\beta \in S_0 \cup S_{\sim 0}$, the decoding is terminated, and the Markov chain stays in β. Thus, the states in S_0 and $S_{\sim 0}$ are absorbing. Also, if $\beta \in S_{R_i}$ ($i = 1...r$) the decoder is allowed to keep running due to the unsatisfied syndrome check. However, β is only the states of the strong component S_{R_i}, for that reason, we also consider that strong components S_{R_i}, $i = 1...r$, are absorbing (the state transition diagram is shown in Figure 3.6).

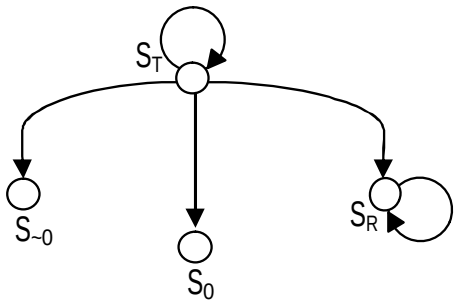


Figure 3.6: .

Probability of Absorption Define now the matrices $P_{T,0}$, $P_{T,\sim 0}$, $P_{T,T}$ with dimensions $|S_T| \times 1$, $|S_T| \times 1$, $|S_T| \times |S_T|$ respectively, and r matrices P_{T,R_i}, $i = 1...r$, with dimensions $|S_T| \times 1$, as follows:

$$\begin{aligned}
P_{T,0} &= \left(\Pr\{\boldsymbol{v}^{(\ell)} = S_0 | \boldsymbol{v}^{(\ell-1)} = \beta\}\right)_{\beta \in S_T} \\
P_{T,\sim 0} &= \left(\sum_{\delta \in S_{\sim 0}} \Pr\{\boldsymbol{v}^{(\ell)} = \delta | \boldsymbol{v}^{(\ell-1)} = \beta\}\right)_{\beta \in S_T} \\
P_{T,T} &= \left(\Pr\{\boldsymbol{v}^{(\ell)} = \varepsilon | \boldsymbol{v}^{(\ell-1)} = \beta\}\right)_{\beta, \varepsilon \in S_T} \\
P_{T,R_i} &= \left(\sum_{\xi \in S_{R_i}} \Pr\{\boldsymbol{v}^{(\ell)} = \xi | \boldsymbol{v}^{(\ell-1)} = \beta\}\right)_{\beta \in S_T}, i = 1...r.
\end{aligned}$$

The matrix

$$P = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \mathbf{0} \\ 0 & 1 & 0 & \dots & 0 & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & \mathbf{0} \\ P_{T,0} & P_{T,\sim 0} & P_{T,S_{R_1}} & \dots & P_{T,S_{R_r}} & P_{T,T} \end{pmatrix} = \begin{pmatrix} I_{r+2} & \mathbf{0} \\ J & Q \end{pmatrix} \tag{3.4}$$

defines the transition probability matrix of the Markov chain $\mathcal{M}_e$. In $\mathcal{M}_e$, S_0 is an absorbing state. All the states in $S_{\sim 0}$ are lumped into a single state. With a moderate abuse of notation, this new state is labeled as $S_{\sim 0}$. The r absorbing states (with lumped states from S_{R_i}, $(i = 1...r)$, is labeled as S_{R_i}, $(i = 1...r)$. The matrix Q in Eq. 3.4 is a transition probability matrix between the transient states in S_T, and I_{r+2} is the $(r+2) \times (r+2)$ identity matrix.

By separating into r absorbing states R_i, $(i = 1...r)$, it allows to compute the probability in which absorbing state that the Markow chain is absorbed. However, if one only consider the probability of absorbing, it is more compact by lumping all the absorbing states S_{R_i}, $i = 1...r$, into only 1 new absorbing state S_R with probability matrix computed as in Equ. 3.5 where

$$P_{T,R} = \left(\sum_{i=1}^{r} \{P_{T,R_i}\} \right).$$

$$P = \begin{pmatrix} 1 & 0 & 0 & \mathbf{0} \\ 0 & 1 & 0 & \mathbf{0} \\ 0 & 0 & 1 & \mathbf{0} \\ P_{T,0} & P_{T,\sim 0} & P_{T,R} & P_{T,T} \end{pmatrix} = \begin{pmatrix} I_3 & \mathbf{0} \\ J' & Q \end{pmatrix} \tag{3.5}$$

The probability distribution of $\mathcal{M}_e$ at iteration k can be written as

$$\boldsymbol{\pi}^{(k)} = (\pi_0^{(k)}, \pi_{\sim 0}^{(k)}, \pi_R^{(k)}, \boldsymbol{\pi}_T^{(k)}), \tag{3.6}$$

where $\pi_0^{(k)} = \Pr\{\beta^{(k)} = 0\}$, $\pi_R^{(k)} = \Pr\{\beta^{(k)} = S_R\}$. $\boldsymbol{\pi}_T^{(k)} = (\pi_\beta^{(k)})_{\beta \in S_T}$ is the probability vector of transient states, and the probability $\pi_{\sim 0}^{(k)}$ is obtained by summing up the probabilities of the corresponding states: $\pi_{\sim 0}^{(k)} = \sum_{\beta \in S_{\sim 0}} \pi_\beta$. The initial distribution $\boldsymbol{\pi}^{(0)}$ is determined as: $\pi_\beta^{(0)} = 1$ if $\beta = S_e$ and $\pi_\beta^{(0)} = 0$ if $\beta \neq S_e, \forall \beta \in S$.

The transition probabilities from transient to absorbing states S_0, $S_{\sim 0}$ and S_R are given by the matrix $J' = (P_{T,0}, P_{T,\sim 0}, P_{T,R})$. The transition probabilities from transient to transient states is determined by $Q = P_{T,T}$.

The transition probabilities between states in k iterations are given by

$$P^k = \begin{pmatrix} I_3 & \mathbf{0} \\ B^{(k)} & Q^k \end{pmatrix}, \tag{3.7}$$

where

$$B^{(k)} = \left(\sum_{i=0}^{k-1} Q^i \right) J' = (I - Q)^{-1}(I - Q^{k+1})J'. \tag{3.8}$$

When the number of iterations is very large

$$\lim_{k \to \infty} B^{(k)} = (I - Q)^{-1} J'. \tag{3.9}$$

The fundamental matrix of the absorbing chain $N = (I - Q)^{-1}$ determines the average times to absorption from different transient states. More specifically, if $S_e \notin \{S_0 \cup S_{\sim 0} \cup S_R\}$ then $\sum(\boldsymbol{\pi}_T^{(0)} N)$ is the average time to absorption from the initial state S_e.

3.3.3 Frame Error Rate Computation

For a given decoder and error pattern e, the frame error rate (FER), the miscorrection probability, (*i.e.* miscorrection error rate (MER)) and correction failure probability (FaER) in the iteration k can be defined as:

$$FER_e^{(k)} = \Pr\{\boldsymbol{v}^{(k)} \in S_{\sim \mathbf{0}} \cup S_R \cup S_T\} \tag{3.10}$$

$$MER_e^{(k)} = \Pr\{\boldsymbol{v}^{(k)} \in S_{\sim \mathbf{0}}\} = \pi_{\sim 0}^{(k)} \tag{3.11}$$

$$FaER_e^{(k)} = \Pr\{\boldsymbol{v}^{(k)} = S_R\} = \pi_R^k \tag{3.12}$$

Theorem 1. *For PGDBF on a subgraph g of any LDPC code $\mathcal{C}$, $\exists L^*$ and Δ, which depends on L^*, such that $\forall L > L^*$*

$$FER_e^{(L)} - MER_e^{(L)} - FaER_e^{(L)} < \Delta. \tag{3.13}$$

Proof. When the Markov chain is in the states of the three disjoint sets $S_{\sim 0}, S_R, S_T$, *i.e.* $\beta \in \{S_{\sim 0} \cup S_R \cup S_T\}$, at iteration k as in Equ. 3.10, it forms the decoding frame error. We can write the FER at iteration k as: $FER_e^{(k)} = \pi_{\sim 0}^{(k)} + \pi_R^{(k)} + \sum \boldsymbol{\pi}_T^{(k)} = \pi_{\sim 0}^{(k)} + \pi_R^{(k)} + \sum \boldsymbol{\pi}_T^{(0)} . Q^k$.

When k becomes larger (let say $k \to \infty$), $Q^k \to \mathbf{0}$, and $\boldsymbol{\pi}_T^{(k)} = \mathbf{0}$, which leads to $FER_e^{(k)} = MER_e^{(k)} + FaER_e^{(k)}$.

□

Average FER Performance The Markov chain model allows to compute the FER associating with an error pattern e located in a subgraph g at iteration k, $FER_e^{(k)}$. By averaging all the error patterns and the graph g in the whole LDPC graph, we obtain the average FER of the decoder at k as

$$FER^{(k)} = \sum_{e \in \{0,1\}^N} \mathtt{Pr}\{e\} \times FER_e^{(k)}. \tag{3.14}$$

The probability of the error pattern e, $\mathtt{Pr}\{e\}$, depends on the error weight $w(e)$ and the number of graph g, N_g, in the whole graph. For the Binary Symmetric Channel (BSC) with crossover probability α, the probability of e, $\mathtt{Pr}\{e\}$ can be expressed as

$$\mathtt{Pr}\{e\} = N_g.\alpha^{w(e)}(1-\alpha)^{N-w(e)}. \tag{3.15}$$

In general, FST completely determines the FER on an entire code graph. However, the state space becomes very large and is the drawback of this method. In this work, the decoding behavior is analysed on the subgraphs of an LDPC code and the appeared limitation is known to come from the isolation assumption [35]. We partially diminish the effect of this isolation assumption on the performance formulation by analyzing on the subgraphs which are a codeword and have the size L much larger than the error weight $w(e)$.

3.4 Performance of Probabilistic Gradient Descent Bit Flipping Decoder

We apply the FST for analysing the performance of PGDBF in 2 cases: the asymptotic decoding performance and the finite number of iteration decoding performance. We first consider the decoding performance of PGDBF in which the number of iteration is assumed to infinity. In the asymptotic decoding performance, we only consider the presence of states in the Markov chain rather than the quantity value of transition probability. In the finite number of iteration decoding process, we show that the average number of iteration needed to converge is finite and determinable as shown in previous section.

We present the analysis results on the 3 subgraphs induced by the minimal codewords of weight-20 of the Tanner code [36][37] and put on the comparison with other BF decoders as well as Min-Sum. As shown in [36][37], by using the results on these subgraphs, the results are predictive when running on the whole graph. We present the 3 subgraphs in the appendix of the thesis.

3.4.1 The asymptotic decoding performance of PGDBF

We classify an error pattern $\boldsymbol{e}$ into 3 categories: *uncorrectable*, *partial-uncorrectable* and *correctable*. An *uncorrectable* error pattern $\boldsymbol{e}$ is the error pattern which can not be corrected by the decoder regardless of number of iterations. More precisely, $FER_{\boldsymbol{e}}^{(k)} = 1, k \to \infty$. Similarly, an *partial-uncorrectable* $\boldsymbol{e}$ is the one that has

$0 < FER_{\boldsymbol{e}}^{(k)} < 1, k \to \infty$ and if $FER_{\boldsymbol{e}}^{(k)} = 0, k \to \infty$ then $\boldsymbol{e}$ is said as *correctable* to the decoder.

We use the following propositions in order to determine the error correction capability of the PGDBF decoder.

Proposition 1. *The error pattern $\boldsymbol{e}$ is uncorrectable to decoder $\mathcal{D}$ in the subgraph g of LDPC code $\mathcal{C}$ if there is no state S_0 in its induced Markov chain $\mathcal{M}_{\boldsymbol{e}}$.*

Proof. This proposition can be proven by using the result of the theorem 1 when $\ell \to \infty$. Due to the absence of S_0, $FER_e^{(\infty)} = MER_e^{(\infty)} + FaER_e^{(\infty)} = 1 - \pi_0^{(\infty)} = 1$. □

The error pattern and the induced Markov chain in figure 3.7 illustrate an uncorrectable error pattern $\boldsymbol{e}$ of PGDBF decoder. In this case, there are only 2 states in S ($S = \{S_{15}, S_{31}\}$), $S_e = S_{15}$ and $S_0 \notin S$, the PGDBF decoder then cycles between these two states S_{15} and S_{31} and has no chance to converge (to be in state S_0).

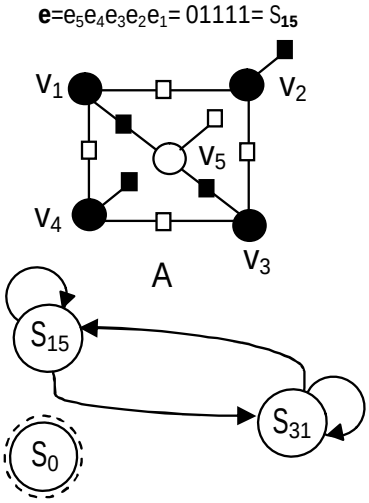


Figure 3.7: An uncorrectable error pattern of PGDBF since the converging state S_0 is not in the induced Markov chain of e.

Proposition 2. *An error pattern $\boldsymbol{e}$ is said as a partial-uncorrectable to decoder $\mathcal{D}$ in the subgraph g of LDPC code $\mathcal{C}$ when there exist S_0 and state (or states) $S_i \in S$ where $S_i \not\to S_0$ in the induced Markov chain $\mathcal{M}_{\boldsymbol{e}}$.*

Proof. Let separate S into 3 subsets: S_0, $\hat{S}_1, (\forall S_i \in \hat{S}_1, S_i \to S_0)$ and $\hat{S}_2, (\forall S_j \in \hat{S}_2, |\hat{S}_2| \geq 1, S_j \not\to S_0)$. Due to the fact that, all states in $\hat{S}_2$ do not communicate to S_0 so they do not communicate to any states in $\hat{S}_1$. In other words, the states in $\hat{S}_2$ communicate only to state in $\hat{S}_2$. Thus, they form an absorbing state in the reduced

Markov chain $\mathcal{M}_e$. Therefore, when $k \to \infty$, $0 < MER_{\boldsymbol{e}}^{(\infty)} + FaER_{\boldsymbol{e}}^{(\infty)} < 1$ so $0 < FER_{\boldsymbol{e}}^{(\infty)} < 1$. □

An example of partial-uncorrectable error pattern of PGDBF is the weight-4 error pattern illustration of Section 3.2. It can be seen in the Markov chain representation that the converging state, S_0, exists leading the possibility to correct all the errors. There are also 3 states (in the dashed cycle) with no transitions to S_0 with which the PGDBF fails to correct the errors.

We apply the FST on the 3 subgraphs induced by the minimal codewords of weight-20 of the Tanner code for BF, GDBF and PGDBF decoders and compare to the quantized Min-Sum (3 bits for LLR and 4 bits for APP information). The numerical quantity in the bracket of tables 3.2, 3.3, 3.4 is number of uncorrectable followed by the partial-uncorrectable error patterns of decoding algorithms corresponding to 3 subgraphs. We choose to evaluate we (the number of erroneous bits) from $we = 2$ to $we = 5$. The number of tested error patterns are computed as the total combinations of $w(e)$ in the subgraph length $L = 20$. It is shown that PGDBF is obviously better than BF and GDBF in term of error correction. PGDBF can correct all weight-3 error patterns while GDBF fails to correct 3 weight-3 error patterns in Type I and 3 weight-3 error patterns in Type II. With weight-4 error patterns, GDBF fails to correct correspondingly 75, 77 and 20 in 3 types of the subgraph while PGDBF fails to correct only 3 weight-4 error patterns in Type I and 3 weight-4 error patterns in Type II. PGDBF also partially fails to correct 9 in Type I and 5 in Type II weight-4 error patterns. Min-Sum decoder is the best error correction by correcting all weight-4 error patterns and fails only with 3 in total weight-5 error patterns.

Table 3.1: Number of codeword weight-20 and TS(5,3) in Tanner code [23].

Type I	Type II	Type III	TS(5,3)
465	465	93	155

By using the number of uncorrectable, partial-uncorrectable error patterns and the codeword weight-20 distribution in Tanner code in Table 3.1, one can produce the predictive decoding performance of GDBF, PGDBF decoders on the error floor using Equ. 3.14. For Tanner code, it is particularly noted that the failure error patterns of GDBF and PGDBF on Tanner code concentrate on the TS(5,3). GDBF fails to correct the weight-3 error patterns located in TS(5,3) as in Figure 3.4a and for each TS(5,3), there are 3 compositions of error bits in the TS(5,3) that GDBF fails ($\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}$ and $\{v_1, v_3, v_5\}$). The performance of GDBF in error floor is, therefore, lower-bounded by: $FER_{GDBF} = N_{TS(3,5)} * 3 * \alpha^3 (1-\alpha)^{(N-3)}$ where $N_{TS(3,5)}$ denotes the number of TS(5,3) in Tanner code.

Table 3.2: Error correction ability of LDPC decoders on the Type I codeword of Tanner code. The numbers in the brackets are (number of uncorrectable - number of partial-uncorrectable) error patterns.

Nb of erroneous bits - Nb of tested error patterns	BF	GDBF	PGDBF	MS(3,4)
2 - 190	(6,0)	(0,0)	(0,0)	(0,0)
3 - 1140	(124,0)	(3,0)	(0,0)	(0,0)
4 - 4845	(1166,0)	(75,0)	(3,9)	(0,0)
5 - 15504	(6554,0)	(846,0)	(67,189)	(2,0)

Table 3.3: Error correction ability of LDPC decoders on the Type II codeword of Tanner code. The numbers in the brackets are (number of uncorrectable - number of partial-uncorrectable) error patterns.

Nb of erroneous bits - Nb of tested error patterns	BF	GDBF	PGDBF	MS(3,4)
2 - 190	(10,0)	(0,0)	(0,0)	(0,0)
3 - 1140	(185,0)	(3,0)	(0,0)	(0,0)
4 - 4845	(1560,0)	(77,0)	(3,5)	(0,0)
5 - 15504	(7980,0)	(925,0)	(73,97)	(1,0)

Table 3.4: Error correction ability of LDPC decoders on the Type III codeword of Tanner code. The numbers in the brackets are (number of uncorrectable - number of partial-uncorrectable) error patterns.

Nb of erroneous bits - Nb of tested error patterns	BF	GDBF	PGDBF	MS(3,4)
2 - 190	(10,0)	(0,0)	(0,0)	(0,0)
3 - 1140	(210,0)	(0,0)	(0,0)	(0,0)
4 - 4845	(1886,0)	(20,0)	(0,0)	(0,0)
5 - 15504	(9704,0)	(480,0)	(20,0)	(0,0)

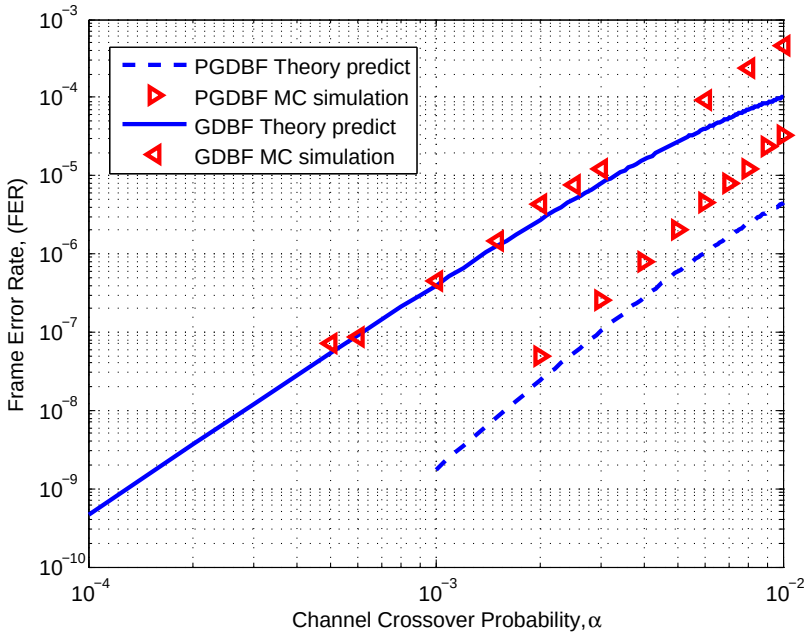


Figure 3.8: Performance of PGDBF ($p_0 = 0.7$) and GDBF by simulation and theoretical prediction on the Tanner code.

3.4.2 The decoding performance of PGDBF in finite number of iteration

The FER performance of the decoder can be computed as a function of iterations by using Equ. 3.14. Also, the average number of iteration is determined by the average absorbing time shown in previous section.

We show in Figure 3.9 tracked FER to follow the decoding iterations of the weight-3 bit error pattern (Figure 3.4) and the 2 weight-4 error patterns with which PGDBF only partially corrects (Figure 3.5 and Figure 3.10) in Tanner code. It can be seen in Figure 3.9 that PGDBF can correct error pattern in Figure 3.5 with $FER = 0.5$. This means that half of the generated random sequences $R^{(k)}$ can correct this weight-4 error pattern, while the other half does not improve the performance. PGDBF performance gains depend on the realizations of $R^{(k)}$, and in some situations requires the concept of PGDBF decoder restarting (or rewinding) [38], that is to start again the decoder from the initial values, but with different random sequences $R^{(k)}$. For the example of the weight-4 error patterns of Figure 3.5, a PGDBF with r_e rewinding stages would correct the error patterns with probability $1 - (0.5)^{r_e}$.

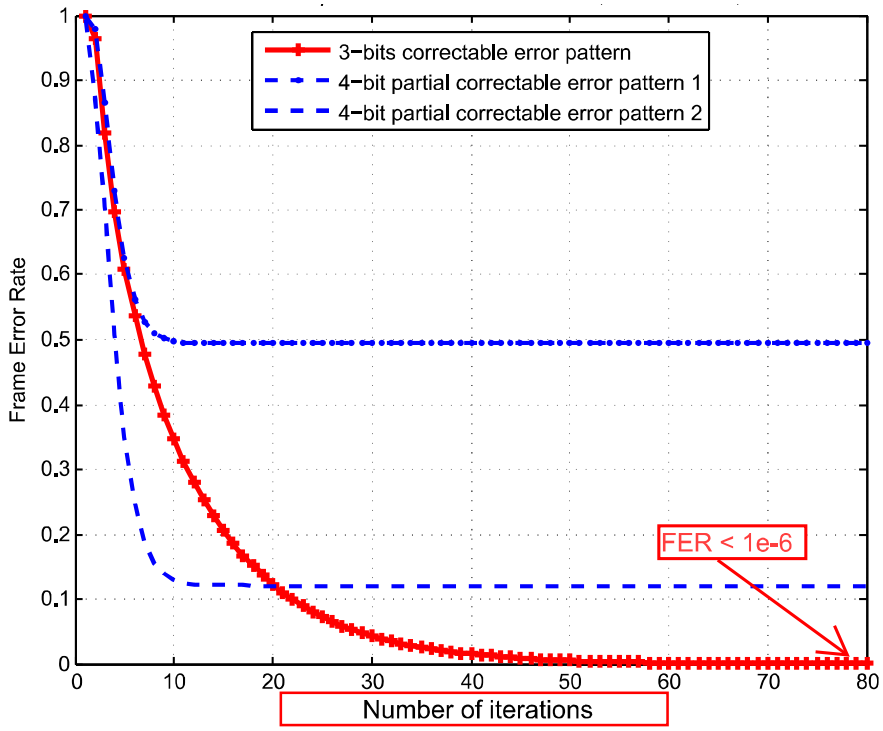


Figure 3.9: Performance of PGDBF as a function of the number of iterations in 3-bits error pattern in Figure 3.4 and 4-bits error patterns in Figure 3.5 and 3.10.

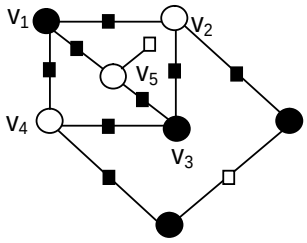


Figure 3.10: An weight-4 partial-uncorrectable error pattern of PGDBF.

3.5 Conclusion

In this chapter, we introduce an analysis method for hard decision decoders denoted as Finite State Tracking. FST represents the decoding process from an iteration to another as a state transition in a Markov Chain (MC). A state of the decoder is represented as a state in this MC and by analyzing the MC, the closed-form expression of Frame Error Rate as a function of number of iteration can be derived. We illustrate the utilization of FST by analyzing the performance of PGDBF decoder and compare it with other BF decoders.

Chapter 4

Efficient hardware implementation of Probabilistic Gradient Descent Bit Flipping

4.1 Introduction

An efficient hardware (HW) implementation of the PGDBF decoder is proposed in this chapter, which minimizes the resource overhead needed to implement the random perturbations and the maximum finder of the PGDBF. The chapter is organized as follows. In Section 4.2, the conducted statistical analysis in PGDBF is presented in order to show the precise characterization of its key parameters, especially the values of p_0 that lead to the maximum coding gains. This analysis is performed through Monte Carlo simulations in both the waterfall and the error floor regions. Section 4.3 shows the optimized HW architecture for the PGDBF decoder. The proposed architecture is based on the use of a short random signals that is duplicated to fully apply the PGDBF decoding rules on the whole codeword. Two different initialization solutions are proposed with equivalent HW overheads, but with different behaviors on different LDPC codes. An optimization of the maximum finder unit of the PGDBF algorithm is also presented in order to reduce the critical path and improve the decoding throughput. Finally, Section 4.4 shows the synthesis results on ASIC 65nm technology, and Monte-Carlo simulations with a bit-accurate C implementation of the proposed PGDBF architecture on LDPC codes with various rates and lengths.

4.2 The statistical analysis of PGDBF decoder

Several recent works showed that the decoding performance of PGDBF is superior to all known BF algorithms [15] as illustrated in Figure 4.1 for a regular $(d_v, d_c) = (4, 8)$ QC-LDPC code of length $N = 1296$ bits. It can be seen in this figure that the PGDBF performs halfway between the GDBF and the Min-Sum

decoder, which is promising in terms of error correction, provided that the extra hardware complexity to implement the generation of the random sequences $R^{(k)}$ is small enough.

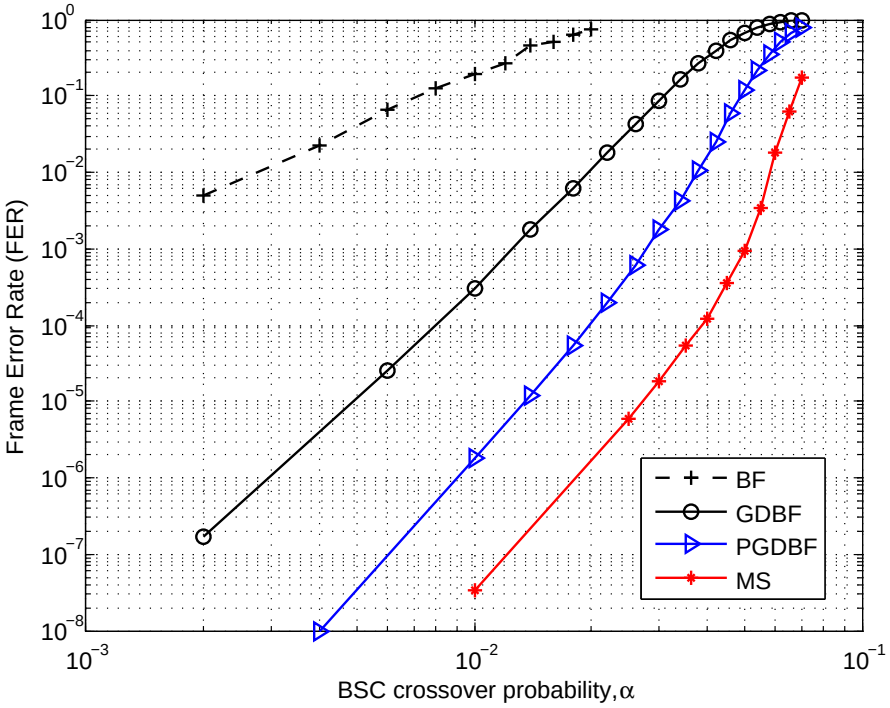


Figure 4.1: Performance comparison between LDPC decoders: BF, GDBF, PGDBF ($p_0 = 0.7$), Quantized MS of the regular QC-LDPC code ($d_v = 3, d_c = 6, Z = 54$), ($N = 1296, M = 648$).

In order to better understand the impact of the randomness of PGDBF on the Frame Error Rate (FER), a statistical analysis of PGDBF is conducted using Monte Carlo simulations. The objective is to identify which features of the probability density function of the binary random sequence $R^{(k)}$ are the most critical for the performance improvements.

4.2.1 Waterfall analysis

The statistical analysis focuses on the effect of the relative occurrence of zeros and ones in the sequence $R^{(k)}$ in the waterfall region of the decoder. The simulations are performed on the BSC channel with cross-over probability α. For each noisy codeword, a fraction of $p_0 N$ ones are put in the random sequence $R^{(k)}$, and the FER of the PGDBF decoder is drawn as a function of p_0, for different iteration and a value of α corresponding to the waterfall region. The results are presented in Figure 4.2 for the ($N = 155, M = 93$) Tanner code [30] which is a regular QC-LDPC code with ($d_v = 3, d_c = 5, Z = 31$).

Based on these results, two interesting conclusions can be drawn. First, during the first decoding iterations ($k \leq 10$), using randomness does not help. On the

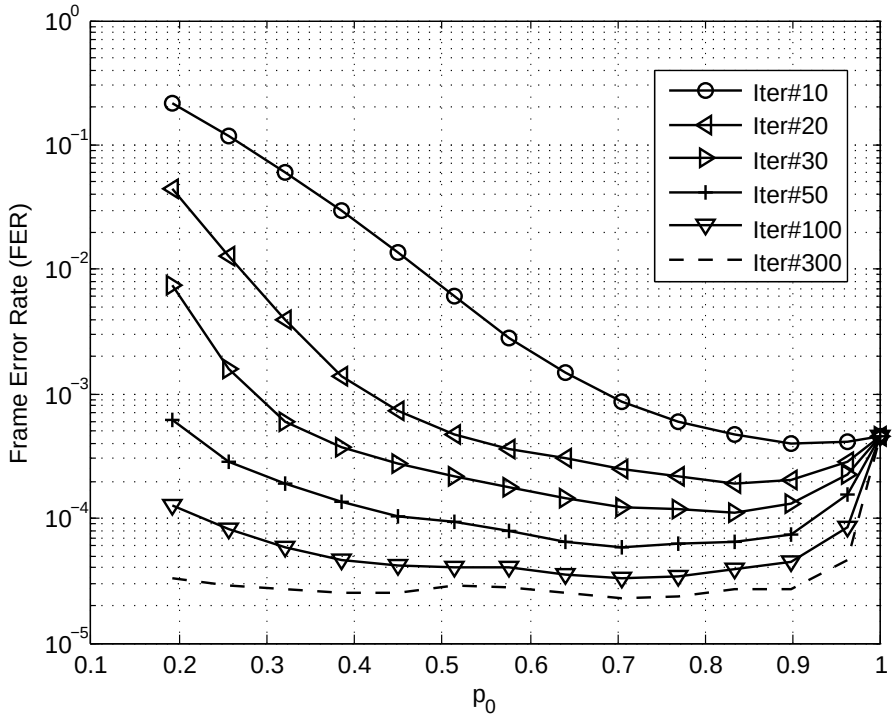


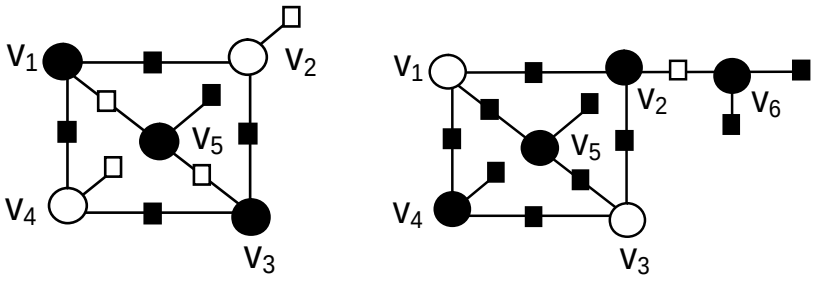
Figure 4.2: Frame Error Rate versus p_0 in the waterfall region ($\alpha = 0.01$) of Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$).

contrary, it degrades the decoding performance since the FER of PGDBF is worse than that of GDBF for almost all values of p_0 (note that GDBF corresponds to PGDBF for $p_0 = 1$). This comes from the fact that the random part of the PGDBF slows down the convergence speed, since fewer bits are flipped than what the energy function indicates. Second, after a sufficient number of iterations, the performance gain is substantial and does not depend much on p_0. This means in particular that optimizing RS sequence probability p_0 does not impact significantly the performance gain, as was already observed in [15]. Those conclusions have been confirmed for several regular LDPC codes with different lengths and values of d_v.

4.2.2 Error-floor analysis

Similar to other iterative LDPC decoders, the GDBF algorithm fails to correct some low-weight error patterns concentrated on trapping sets (TS) [27, 39], giving rise to the so-called error floor region. The PGDBF has been introduced to overcome the attraction of TS. In this section, a same experiment is conducted as in Section 4.2.1 for the Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$), but in the error floor region. The smallest trapping set for this code is composed of 5 VNs and 3 odd-degree CNs, denoted TS$(5, 3)$ (see Figure 4.3).

The minimum number of bits that cannot be corrected by the deterministic GDBF is three [38], and they are located in the TS$(5, 3)$ of the LDPC code as indicated in Figure 4.3(a) with black circles. Note that (v_1, v_2, v_3) and (v_1, v_4, v_3) are



(a) (b)

Figure 4.3: Error configurations with (a) 3 erroneous bits and (b) 4 erroneous bits located on a TS(5, 3). Black/white circles denote erroneous/correct variable nodes, and black/white squares denote unsatisfied/satisfied check nodes.

also weight-3 error patterns which cannot be corrected by the GDBF. The PGDBF can potentially help correct these low weight error patterns, resulting in a coding gain in the error floor region. Figure 4.3(b) shows also a weight-4 error pattern which is uncorrectable by the GDBF. In order to analyze the PGDBF in the error floor, the channel errors on the positions indicated in Figure 4.3 is fixed, and is evaluated whether a random sequence $R^{(k)}$ with probability p_0 can correct these error patterns. The results are shown in Figures 4.4 and 4.5.

The first remark that can be made is that, contrary to the conclusion of the waterfall analysis, it is useful to use the random feature of the PGDBF, even during the first decoding iterations. This is verified for both the weight-3 error and the weight-4 error patterns. The weight-3 error pattern is eventually corrected when the number of iterations increases, for all values of $p_0 \in [0.3, 0.9]$. The weight-4 error patterns can also be corrected by the PGDBF for a wide range of p_0 values, but flattens at a FER equal to 0.5. This means that half of the generated random sequences $R^{(k)}$ can correct the weight-4 error pattern, while the other half does not improve the performance. PGDBF performance gains depend on the realizations of $R^{(k)}$, and in some situations requires the concept of PGDBF decoder rewinding [38], that is to start again the decoder from the initial values, but with different random sequences $R^{(k)}$. For the example of the weight-4 error patterns of Figure 4.3(b), a PGDBF with k rewinding stages would correct the error patterns with probability $1-(0.5)^k$. Since the work in this chapter deals with low complexity implementation of the PGDBF decoder and because the PGDBF gain is already significant without rewinding, decoder rewinding is not considered in this chapter.

As a conclusion of this section, the statistical analysis reveals that the random generator does not need to have a specific value for p_0 in order to provide the performance gains of the PGDBF, as long as it is bounded away from $p_0 = 1$. This motivated the proposition of a simplified, low complexity hardware realization for the generation of sequence $R^{(k)}$, which is described in the next section.

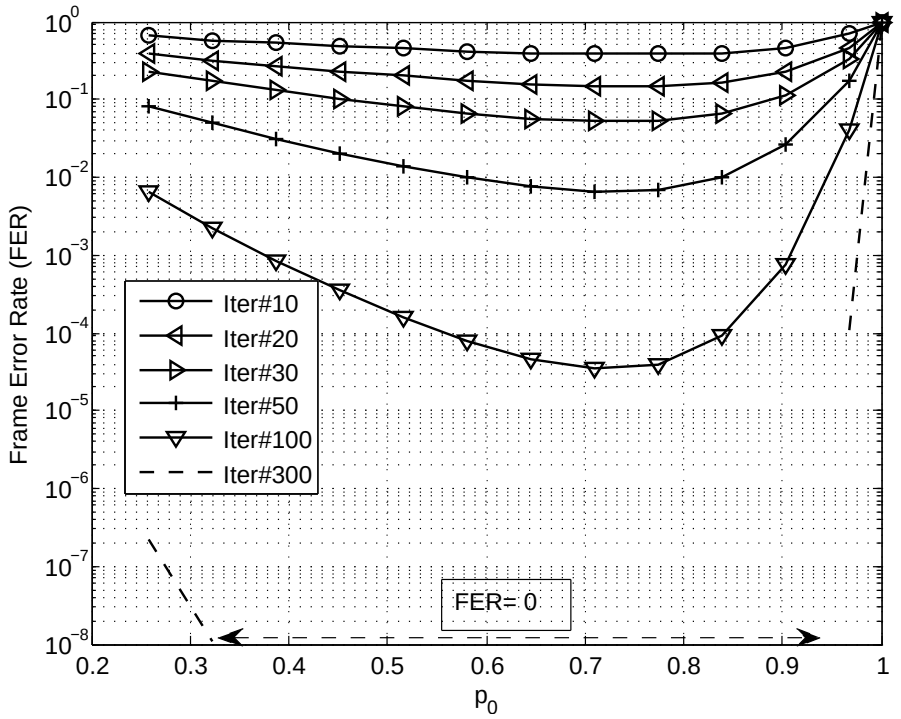


Figure 4.4: Frame Error Rate versus p_0 in the error floor region with 3 erroneous bits of Tanner code $(d_v = 3, d_c = 5, Z = 31)$, $(N = 155, M = 93)$.

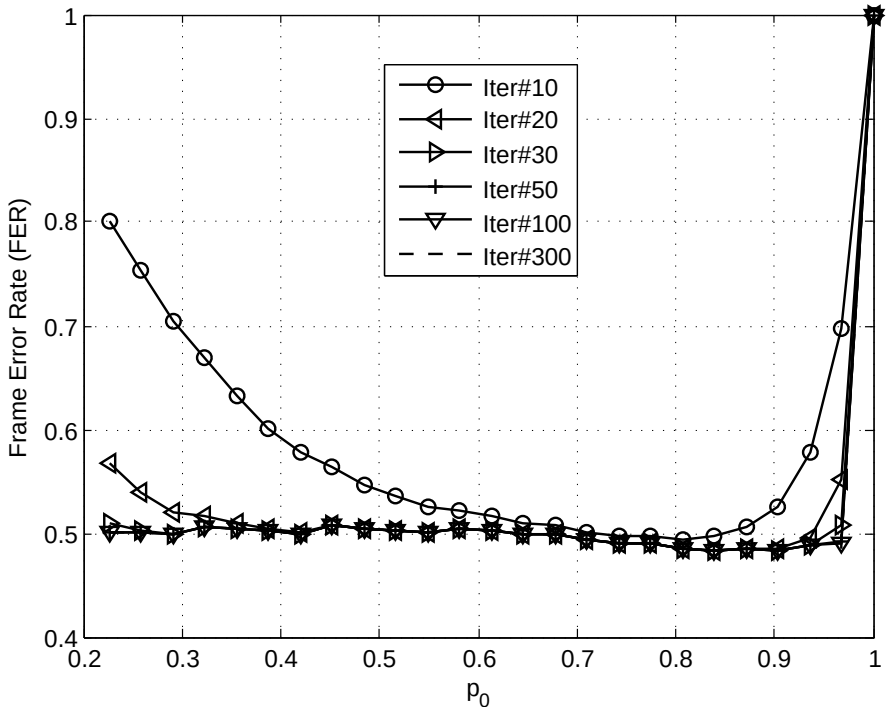


Figure 4.5: Frame Error Rate versus the p_0 in the error floor region with 4 erroneous bits of Tanner code $(d_v = 3, d_c = 5, Z = 31)$, $(N = 155, M = 93)$.

4.3 The optimized hardware implementation

The random feature of PGDBF plays an important role in improving the decoder performance as presented in the previous section. However, a hardware overhead is unavoidable due to the fact that a binary random generator is required on top of the original GDBF structure. An optimized hardware implementation of the PGDBF is presented in this Section, with the objective of keeping the coding gain at a minimum hardware cost.

4.3.1 PGDBF global architecture

The global architecture of the PGDBF decoder is shown in Figure 4.6. The organization of the blocks and the data flow follows precisely the organization of the GDBF decoder, with the difference being in the additional block which produces the random signals $R^{(k)}$.

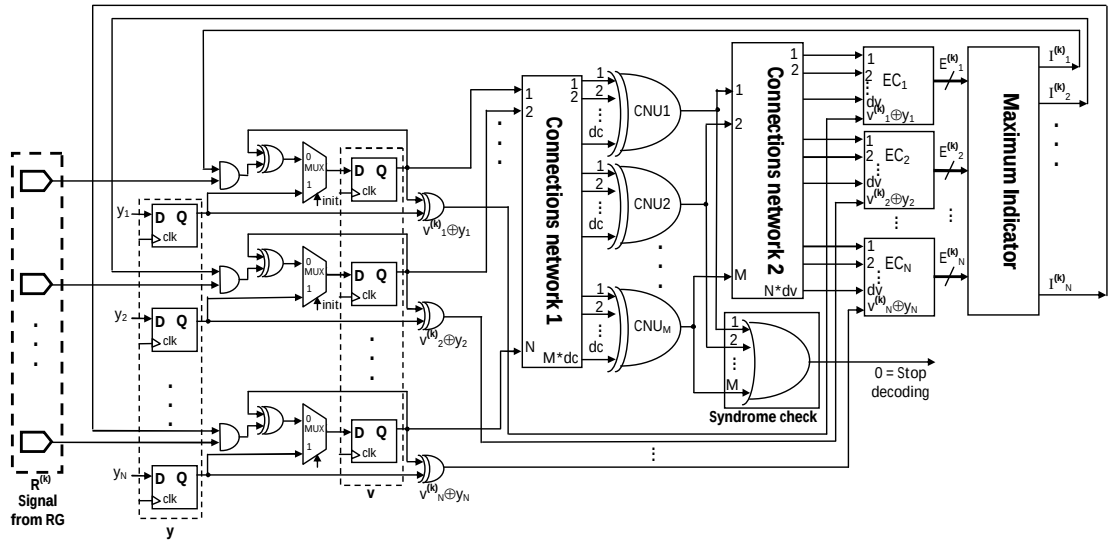


Figure 4.6: The global architecture of the PGDBF. The PGDBF follow precisely the data flow of GDBF with difference coming from the random generator and the AND-gates.

The operation principle of PGDBF algorithm on this hardware architecture is briefly presented as following. Since the PGDBF is introduced in this work for the BSC channel, two register arrays represented as two sequences of D-Flip Flops (D-FFs) are required to store the noisy codeword $\mathbf{y}$ and the estimated codeword at the current iteration $\mathbf{v}^{(k)}$. At the initialization of the decoder, the signal *init* triggers the copy of $\mathbf{y}$ into $\mathbf{v}^{(0)}$. Then, the CNUs compute the parity of their neighboring bits in $\mathbf{v}^{(k)}$, after properly driven by a first connection network. The second connection network drives the CN values to the energy computation blocks, for each VN. The maximum indicator module is composed of a maximum finder component and comparators which outputs $I_n^{(k)} = 1$ whenever the corresponding energy is equal to the maximum, and $I_n^{(k)} = 0$ otherwise. Indicator values $I_n^{(k)}$ are propagated to the AND

gates, and combined with the RS sequence. This series of AND gates highlights the difference between PGDBF and GDBF. In the GDBF decoder, all bits with $I_n^{(k)} = 1$ are flipped, while in the PGDBF algorithm, only the bits with $I_n^{(k)} = 1$ and $R_n^{(k)} = 1$ are flipped. New values of the bits stored in $\mathbf{v}^{(k)}$ are used for the next decoding iteration.

At each iteration, the syndrome check module performs an OR operation on the CNs values to verify whether the intermediate sequence $\mathbf{v}^{(k)}$ is a codeword, in which case the decoding process is halted. Another instance when the decoding halts is when no codeword $\mathbf{v}^{(k)}$ has been found, and a predetermined maximum number of iterations It_{max} has been reached, in which case a decoding failure is declared (the iteration counter is implemented in the controller which controls the system and is not shown in the above architecture). Note that all components in the decoder architecture are combinational circuits except the registers $\mathbf{v}^{(k)}$ and $\mathbf{y}$. Therefore, new values of the bits in $\mathbf{v}^{(k)}$ are updated after each clock cycle. In order to optimize the proposed architecture, the following two important issues are focused on. First, as we identified and published in [29], the hardware overhead induced by the RS is not negligible with a naïve implementation, and different low complexity methods to generate the sequences $R^{(k)}$ are proposed in Section 4.3.2.1. Second, the optimized architecture of the maximum indicator module is presented in order to maximize the decoding throughput, as explained in Section 4.3.3.

4.3.2 Implementation of the perturbation block

4.3.2.1 Cyclically-shift truncated sequences

It is shown in Section 4.2 that the performance gain in the PGDBF algorithm comes from the introduction of the random sequence $R^{(k)}$ which makes a perturbation in bit flips. In the theoretical description of the PGDBF, for each and every codeword, the sequences at different iterations $\left(R^{(0)}, \ldots, R^{(It_{max})}\right)$ are independent and identically distributed. However, a direct and naïve generation of the sequences $R^{(k)}$ with linear feedback shift register (LFSR) random generators is costly, and requires many times more registers than the non-probabilistic GDBF [29]. An approach to reduce the hardware overhead required to generate the RS sequences is introduced in this Section.

The first modification proposed is to reduce the register requirements by storing only $S \leq N$ random values in a shorter sequence denoted as $R_t^{(k)}$, and produce the RS sequence $R^{(k)}$ with a hard-wired duplication network. The duplication network can be restricted to a simple *copy and concatenate*, as illustrated by the example in Figure 4.7-a, or can be implemented as a more complex connection network, allocating S register outputs to N signals. When $S = N$, the duplication network is a simple copy of the register outputs, i.e. $R^{(k)} = R_t^{(k)}$ as shown in Figure 4.7-b.

Although the length S of the truncated sequence can take in principle any value, in this implementation the possible values of S are restricted to integer multiples of the QC-LDPC circulant size Z. The duplication network is also reinforced to copy the register outputs to distinct circulant blocks, such that two copies of the same

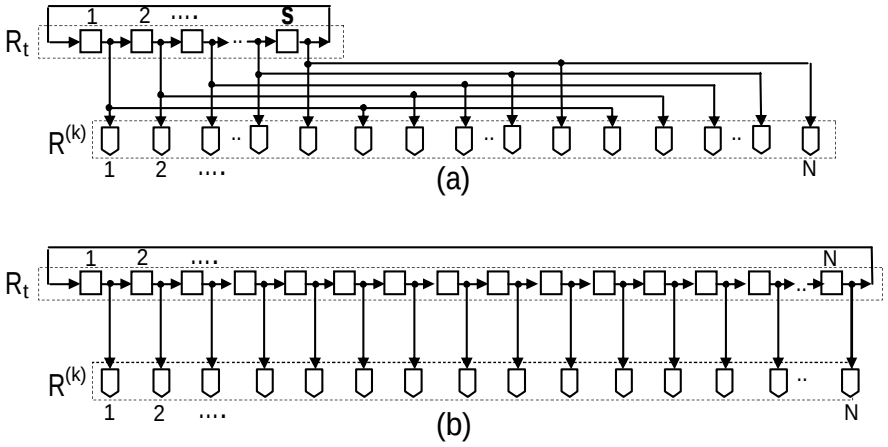


Figure 4.7: Generation of the random signals, (a) corresponds to the use of truncated sequences, and (b) to the use of full sequences.

random bit do not belong to the same circulant block. The duplication network comes at very little cost in hardware implementation.

The second modification proposed is to re-use the same sequence of random bits for all the decoding iterations, instead of generating a new random sequence at each iteration. However, in order to preserve the performance gains of the PGDBF, the sequences $R_t^{(k)}$ need to be distinct from one iteration to another. The proposed solution is to simply rotate cyclically the sequence by one position, as shown in Figure 4.7, such that:

$$R_t^{(k+1)}\left((n+1)\ mod\ S\right) = R_t^{(k)}(n) \quad n = 1, \ldots, S \tag{4.1}$$

With these two modifications, the objective is to reduce to the maximum the hardware overhead induced by the RS sequence generation, while maintaining the necessary randomness of the PGDBF algorithm. The proposed solution, named Cyclically Shifted Truncated Sequences (CSTS), has also the following two advantages with respect to the desired statistical properties of the random sequence. First, the initialization step starts with a random truncated sequence $R_t^{(0)}$ having a given number $n_0 = p_0\, S$ of 1's, the number of 1's in the complete sequence $R^{(0)}$ will be in the range $\{\lfloor N/S \rfloor\, n_0, \ldots, (\lfloor N/S \rfloor + 1)\, n_0\}$. This means that the value of p_0 fixed by the statistical analysis in Section 4.2 would be the same for $R_t^{(0)}$ and $R^{(0)}$. Second, since a cyclically shift on the sequences $R_t^{(k)}$ is proceeded from one iteration to another, the number of 1's is kept constant throughout the decoding iterations if N/S is an integer, and almost constant if N/S is not an integer.

One of the drawbacks of this reduced complexity method for generating the random signals is the induced correlation. In the theoretical PGDBF, the assumption is that the sequences $R^{(k)}$ should be independent from each others. However, it is obvious that by using the duplication from $R_t^{(k)}$ to $R^{(k)}$, and by shifting the sequences from one iteration to another, the independence assumption is no longer valid. It is also clear that the sequences will be more correlated as S becomes smaller. However,

it have been verified through Monte Carlo simulations that the induced correlation has actually very little impact on the decoder performance. Figure 4.8 shows the FER performance of the PGDBF with CSTS and the PGDBF with perfect RG. The simulations are performed on a rate R= 1/2 regular $(d_v = 3, d_c = 6)$ QC-LDPC code, with circulant size $Z = 54$ and length $N = 1296$ and the BSC channel. It can be seen in these results that the use of CSTS does not degrade the error correction performance, except a very small values of S. This behavior has been also observed for other code rates and lengths.

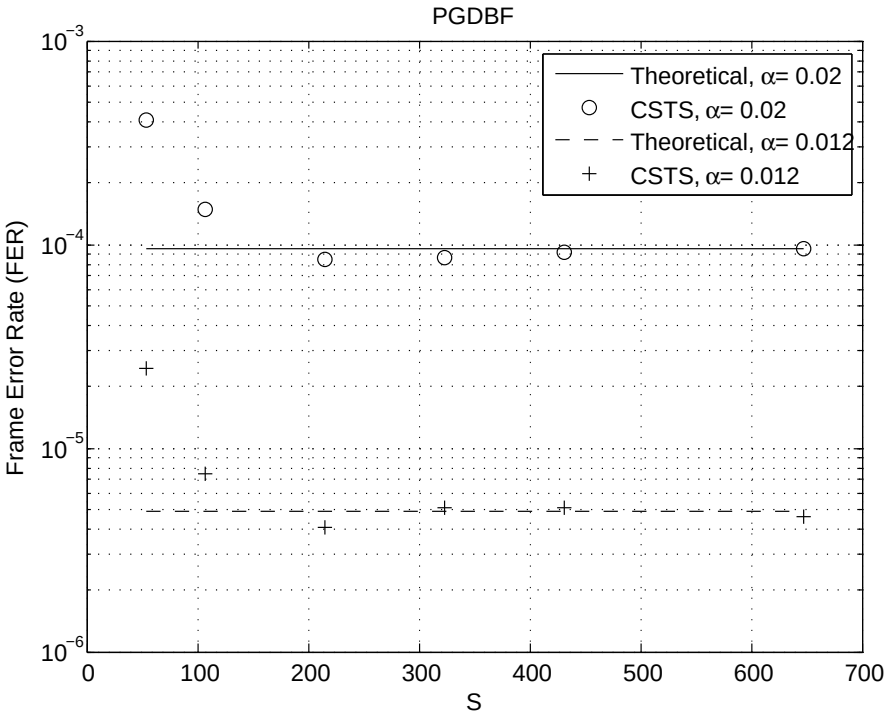


Figure 4.8: Decoding performance of CSTS-PGDBF as a function of the size S of $R_t^{(0)}$.

4.3.2.2 Initialization with Linear Feedback Shift Register

It is shown in the previous section that once the initial truncated sequence $R_t^{(0)}$ has been generated, the proposed simplified CSTS architecture can build efficiently the RS sequences $R^{(k)}$, $k = 0, \ldots, It_{max}$, with no performance loss. In this section, two solutions to initialize $R_t^{(0)}$ with S random binary values are described, which will be compared in terms of hardware resource in Section 4.4. The proposed solutions are based on two random generating methods, namely linear feedback shift register (LFSR) and intrinsic valued random generator (IVRG). For simplicity, the CSTS notation in the algorithm names is dropped and these names are then referred as LFSR-PGDBF and IVRG-PGDBF.

A conventional method to produce pseudo-random bits is by making use of the LFSR. The outputs of the LFSR registers define an integer number, which is com-

pared to a pre-determined threshold in order to finally produce a random bit as in Figure 4.9. The value of the threshold is tuned so that the appearance probability of 1's at the comparator output is equal to the desired value, i.e. p_0 in our case. For large enough LFSR memory η, the correlation of the output bit sequence could be made negligible [40]. An $\eta = 32$-LFSR pseudo-random generator is used, to initialize the truncated sequence $R_t^{(0)}$ for each noisy codeword, described in Figure 4.10. Note that the latency to produce the S random values can be neglected since the LFSR can produce those values while the previous noisy codeword is decoded.

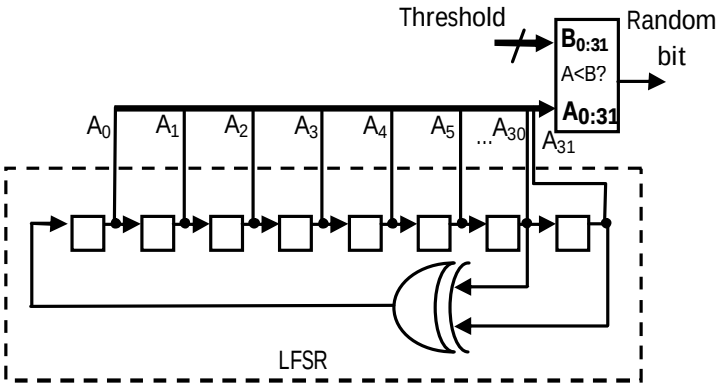


Figure 4.9: A LFSR unit to generate 1 random bit.

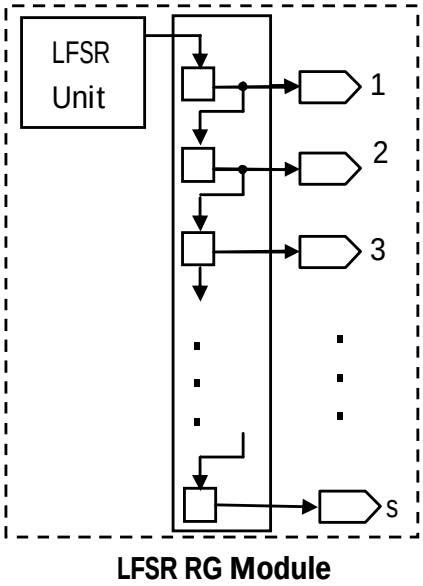


Figure 4.10: The LFSR RG module

One drawback of this approach corresponds to the event that $R_t^{(0)}$ contains only binary ones, in which case all sequences $R^{(k)}$ are also filled with binary ones, thus

reducing the PGDBF to deterministic GDBF. The probability of having $R_t^{(0)} = \mathbf{1}$ is equal to $(p_0)^S$, which can be considered negligible only if it is smaller than the target FER in the error-floor region. For example, with $p_0 = 0.75$ and $S = 64$, $p(R_t^{(0)} = \mathbf{1}) = 1e-8$. The LFSR method therefore requires an extra control module that tests if $R_t^{(0)} \neq \mathbf{0}$, or by forbidding too large values of p_0 and too small values of S.

4.3.2.3 Initialization with The Intrinsic-Valued Random Generator

In this section, we introduce a novel and specific way for generating a sequence of random bits in an LDPC decoder without relying on an actual random generator. Our Intrinsic-Valued Random Generator (IVRG) approach was based on the use of the CN values which are already computed at the output of the CNU (see Figure 4.6). The CNU blocks compute the parity-check node values, and if the estimated codeword contains errors, they typically contain a fraction of ones. We make use of the CN values at the first iteration $k = 0$, which depend on the noisy codeword generated by a random realization of the BSC channel. As a result, the sequence of CN values will appear as a random-like sequence that can be used to initialize $R_t^{(0)}$.

To simplify the discussion, let us consider the special case of $S = M$, the number of parity-check equations in the LDPC code, such that $R_t^{(0)}$ can be filled directly with the complemented outputs of the CNUs at the first iteration, as described in Figure 4.11. With the IVRG approach, we need to complement the CN values since the number of unsatisfied check nodes is usually smaller than the number of satisfied CNs, and we saw in the statistical analysis of Section 4.2 that the interesting range of p_0 is greater than 0.5.

Also, with the IVRG approach, the number of ones in $R_t^{(0)}$ cannot be controlled and depends on the channel noise realization. Let us denote by $p(c^{(0)} = 1)$ (respectively $p(c^{(0)} = 0)$), the probability that a CN c is unsatisfied (respectively satisfied), at the initialization $k = 0$. Because of the complement after the CN computation, the probability of the binary ones in the RS sequence in the IVRG approach is equal to $p_0 = p(c^{(0)} = 0)$. For a BSC with crossover probability α, we can approximate this probability by $p_0 = p(c^{(0)} = 0) \simeq \frac{1}{2} + \frac{1}{2}(1-2\alpha)^{d_c}$. For large values of α, as in the waterfall region of the LDPC decoder, p_0 converges to 1/2, and for small values of α, in the error-floor region, it converges to $p_0 = 1$, which corresponds to the GDBF.

Contrary to the LFSR method, the case where $R_t^{(0)} = \mathbf{1}$ is less problematic since it happens only when the decision at the channel output is already a codeword, and then no decoding iteration is required. Figure 4.12 shows the distribution of p_0 for a $(d_v = 3, d_c = 6, N = 1296)$ QC-LDPC code and two different channel error probabilities α. It can seen that p_0, although not fixed to a constant value with the IVRG, falls in the range of interest, predicted by the statistical analysis of Section 4.2.

Another feature of the IVRG initialization is that copying the CN values in the $R_t^{(0)}$ registers can be done on-the-fly, and does not induce any extra latency. Note finally that, although the discussion in this section has been made for $S = M = N\,d_v/d_c$, the IVRG technique is also applicable for any value of $S \leq M$.

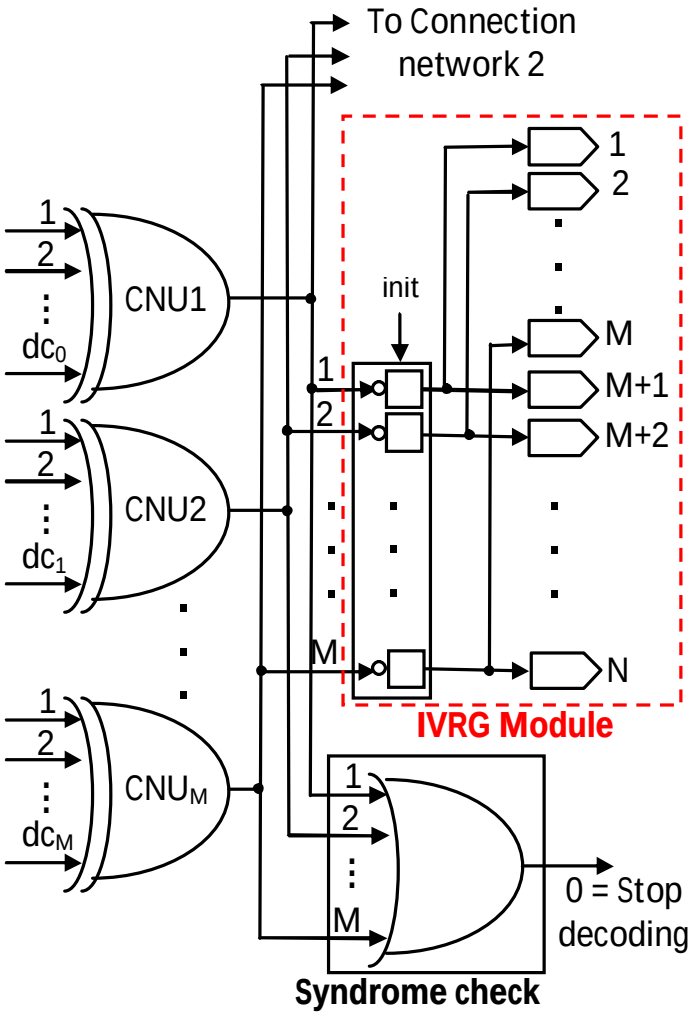


Figure 4.11: A block diagram of the Intrinsic-Valued Random Generator module for $S = M = N/2$. The CNs values are copied into the $R_t^{(0)}$ at the first iteration, then cyclically shifted at each iteration.

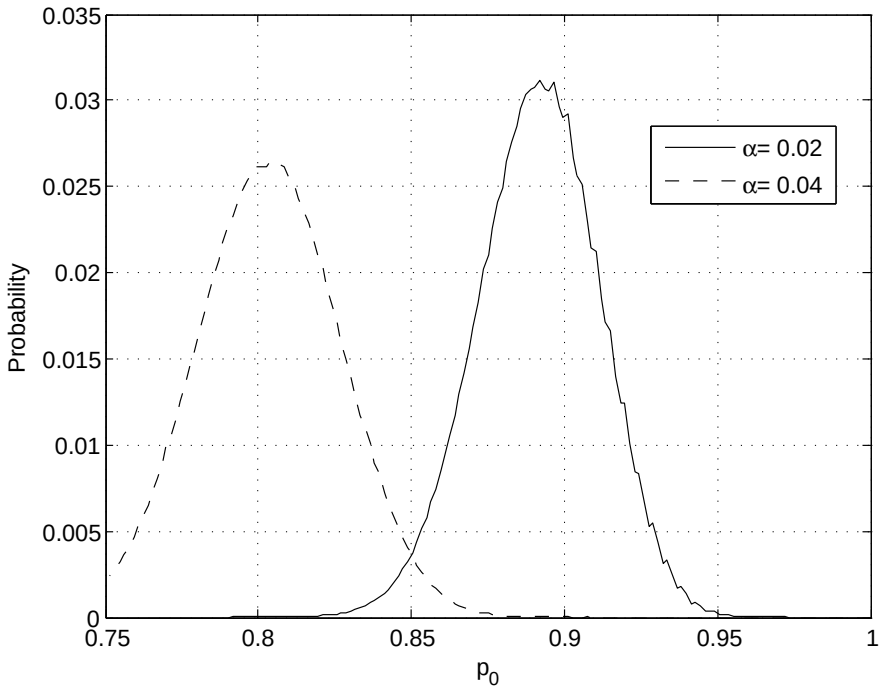


Figure 4.12: The distribution of p_0 for the IVRG-PGDBF and a $(d_v = 3, d_c = 6, N = 1296)$ QC-LDPC code, for $\alpha = 0.02$ and $\alpha = 0.04$.

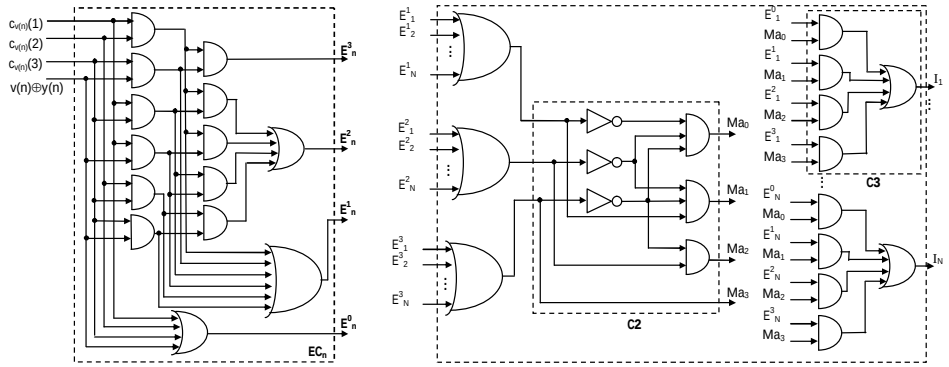
4.3.3 The optimized architecture of the maximum finder

Another important module of the PGDBF decoder is the maximum indicator (MI) module, which contains the critical path of the global architecture (Figure 4.6), and therefore limits the achievable working frequency and decoding throughput [41]. A rigorous survey on the maximum finder architectures can be found in [41], in which the authors confirmed that the implementation of finding the maximum value out of a list of N numbers is always a trade off between computation cost and area/energy consumption. Since the goal is to get an optimized decoder with respect to multiple objectives, including hardware cost minimization, but also fast decoding throughput, the minimization of the critical path for the MI module is focused on, even if it comes at the cost of an increased hardware resource. This work, therefore, leaves aside the solutions which minimize the area/consumption, such as the Traditional Binary Tree (TBT) method, and focuses on the method which maximizes the decoding speed, i.e. the Leading-zero Counting Topology (LCT) [41]. For simplicity, the iteration index (k) is omitted in this section.

The timing efficiency of LCT comes from the principle of processing the bits of an operand independently from other bits. In order to do this, the LCT method requires the operand to be represented in a special format called *one-hot format* with $q = \delta + 1$ quantization bits, where δ is the maximum value of the operand. In order to find the maximum in a set of N values by LCT, a q-bits vector is produced by applying N-inputs bit-wise OR operations on the N one-hot-format values. The maximum value is then sorted out by using a priority encoder on the results of

these OR operations [41]. The complexity of LCT maximum finder comes from the N-inputs OR gates, and a reduction of the number of N-inputs OR gates would reduce significantly the complexity of decoders. In the proposed LCT-based MI for PGDBF ($\delta = d_v + 1$), since E_{max} can never be equal to 0 except when $\mathbf{y}$ is already a codeword or $\mathbf{v}^{(k)}$ is converging to a codeword, the optimized implementation of the LCT and reduction of its complexity are obtained by representing the energy function values in a format with the length of $q-1$ bits (instead of q bits) such that $E_n = E_n^{q-2}...E_n^1E_n^0$, $E_n^j = 1$ if $E_n = j+1$, $E_n^i = 0$, $\forall i > j$ and E_n^i are don't-care otherwise. By doing this, all the operation of LCT is preserved while we can reduce by 1 N-inputs OR-gate.

We can further reduce by 1 the N-inputs OR-gates by applying logic minimization. Indeed, it can easily ignore the case of $E_{max} = 1$ and consider only the remaining possibilities (i.e $E_n = j$, where $j = 2, \ldots, q-1$). The case of $E_{max} = 1$ will be automatically considered when all the other possibilities are discarded. Thus, the number of N-inputs OR-gate can be finally reduced to only $q-2$ (originally q). Figure 4.13 shows the detailed circuits for energy computation and MI blocks of $d_v = 3$ LDPC codes. The energy computation block (Figure 4.13(a)) produces the energy value, E_n, for VN v_n in the required format by using the neighbor CN values and XOR-ing results of v_n to y_n as its inputs. The MI block (Figure 4.13(b)) requires only 3 N-inputs OR gates (instead of 5) and block $C2$ realizes the logic minimization process as mentioned above. The $I_n, n = 1, \ldots, N$, is produced by the circuit as in Figure 4.13(b) (block $C3$). In the $C3$ circuit, $I_n = 1, n = 1 \ldots N$ if and only if $Ma_j = 1$ and $E_n^j = 1$, $0 \leq j \leq q-1$, concurrently, otherwise $I_n = 0$.



(a) (b)

Figure 4.13: Detailed circuits of implemented Energy Computation and Maximum Indicator blocks for $d_v = 3$ LDPC codes, (a) Energy Computation block, (b) Maximum Indicator block.

4.4 Synthesis results

4.4.1 PGDBF Synthesis Results

In this section, we report the ASIC results at post-synthesis level, of the proposed PGDBF implementations. The synthesis has been done targeting a 65nm CMOS

process technology, using Synopsys tools.

For the first synthesis comparison, our goal is to demonstrate the area gains that one can achieve using the CSTS-PGDBF approach, *i.e.* using short size S for the random sequence $R_t^{(0)}$. The results are reported in Table 4.1 for a QC-LDPC code, with parameters $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$ and circulant size $Z = 54$ (denoted as dv3R050N1296). In this Table, we have constrained the implementations to run at the same clock frequency, by setting the timing constraint identical for all decoders, fixed to 8 ns. We choose this strategy to measure precisely the impact of S on the hardware cost, even if the working frequency is not maximized. We indicate in brackets the additional cost in percentage compared to the deterministic GDBF implementation. As a first remark, we can see that the LFSR-based and the IVRG-based implementations have similar costs. So, both solutions for the implementation of the RS sequence are equally competitive with respect to the hardware resource usage. As expected, the overhead is roughly proportional to the size of the RS memory S, and becomes very small ($6\% - 7\%$) for small values of $S \leq 4\,Z$. As we demonstrate in the next section, this small value of S is sufficient to obtain important error correction gains.

We have verified that the same conclusions could be drawn for different LDPC codes, with various lengths, rates and values of d_v. The synthesis results are reported in table 4.2. For the same codeword length N and both for $d_v = 3$ and $d_v = 4$, the overhead is lesser for rate $R = 3/4$, than for rate $R = 1/2$. This comes from the fact that the value of $S = M$ is smaller for larger rates. For the same rate and length, we can see that the overhead is slightly smaller for $d_v = 4$ than for $d_v = 3$: $10\% - 11\%$ against $12\% - 13\%$ for $R = 1/2$, and $6\% - 7\%$ against $8\% - 9\%$ for $R = 3/4$. Our PGDBF implementations are therefore more advantageous as the rate increases. Finally, for a very long codeword and high rate with parameters $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, the extra overhead needed to implement the PGDBF becomes almost negligible, as low as $4\% - 5\%$.

dv3R050N1296 - AREA (μm^2)				
	S = Z = 54	**S = 4Z = 216**	**S = 12Z = 648**	**S = 24Z = 1296**
GDBF	53692 (+0%)			
LFSR-PGDBF	55837 (+4.00%)	57342 (+6.80%)	60360 (+12.42%)	64897 (+20.87%)
IVRG-PGDBF	55586 (+3.53%)	57295 (+6.71%)	60815 (+13.27%)	N/A

Table 4.1: Hardware resource used to implement the PGDBF decoders as a function of S. The percentages in brackets indicate the additional hardware compared to the GDBF.

In the second synthesis comparison, we report maximum working frequency and throughput in Tables 4.3 and 4.4. Table 4.3 shows the results for the GDBF, PGDBF and two MS decoders taken from the literature [24, 25]. The code used for the GDBF, PGDBF and [24] is the $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$, $Z = 54$ LDPC code, while [25] considers the IEEE 802.11n standard codes with various lengths and

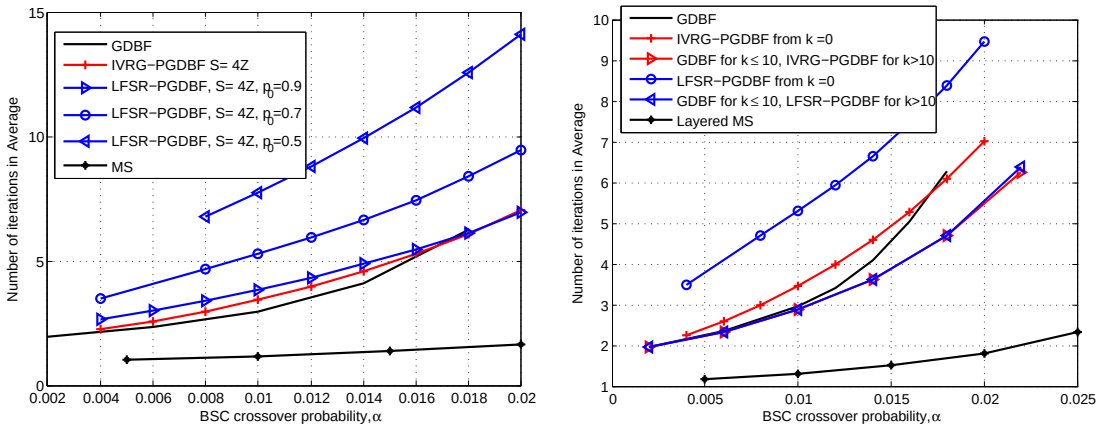
AREA (μm^2)						
	dv= 3			dv= 4		
	$d_c = 5$	$d_c = 6$	$d_c = 9$	$d_c = 8$	$d_c = 12$	$d_c = 34$
	R040N155	**R050N1296**	**R075N1296**	**R050N1296**	**R075N1296**	**R088N9520**
GDBF	6601 (+0.00%)	53692 (+0.00%)	54131 (+0.00%)	66367 (+0.00%)	67170 (+0.00%)	493408 (+0.00%)
LFSR-PGDBF (S = M)	7777 (17.82%)	60360 (12.42%)	58526 (8.12%)	73042 (10.06%)	71561 (6.54%)	515061 (4.39%)
IVRG-PGDBF (S = M)	7599 (+15.12%)	60815 (+13.27%)	58605 (+8.27%)	73480 (+10.72%)	71644 (+6.66%)	516216 (+4.62%)

Table 4.2: Hardware resource used to implement GDBF and PGDBF decoders for different LDPC codes from short to very long codeword lengths and different code rates. The values in brackets are percentage of additional hardware compared to GDBF

rates. In Table 4.4, a large length and high rate LDPC code is considered, and our decoders are compared with the MS implementation of [26].

For the GDBF and the PGDBF decoders, we performed the synthesis with the objective of optimizing the timing constraint, which results in the maximum frequency at which the decoder can operate.

We furthermore considered the following two scenarios: one scenario when the RS sequence is applied from the beginning of the decoding, and the other scenario when the deterministic GDBF is used for the first 10 iterations, and the RS sequence used for the remaining iterations. The reason of this later scenario is twofold. First, our statistical analysis from Figure 4.2 showed that using the random sequences during the first iterations could be detrimental to the decoding performance, and that it is better to trigger the random feature of the PGDBF after a small number of iterations. Second, many of the noisy codewords do not require the strength of the PGDBF, and a simple GDBF could already correct them in a few iterations. Since the GDBF converges faster than the PGDBF, the average throughput is larger if we use GDBF, instead of the PGDBF, for the first 10 iterations. To support this claim, we plotted on figure 4.14 the average number of iterations as a function of α, for the $d_v = 3$, $d_c = 6$, $N = 1296$, QC-LDPC code (Note that in Figure 4.14(a), the randomness is applied at the beginning of decoding process in PGDBF decoders). It can be seen that when we start using the PGDBF after 10 iterations (see Figure 4.14(b)), the average number of iterations is always lower or equal than the deterministic GDBF. We will use this scenario for the average throughput computations. The average throughput (θ) of the decoders is computed as $\theta = \frac{f_{max} * N}{It_{ave} * N_c}$ where f_{max} is the maximum working frequency provided by the synthesizer, It_{ave} is the average number of iterations and N_c is the number of clock cycles needed for one decoding iteration.



(a) (b)

Figure 4.14: Average number of iterations for GDBF, PGDBF and MS decoders on the dv3R050N1296 regular LDPC code. In Figure (a), randomness is applied at the beginning of decoding process in PGDBF decoders. In Figure (b). PGDBF decoders are with $S = 4Z$ and $p_0 = 0.7$ in LFSR-PGDBF

	Code length	Code rate	AREA (μm^2)	kGE	f_{max} (MHz)	N_c	$FER = 1e-5$		$\alpha = 0.01$	
							It_{ave}	θ (Gbit/s)	It_{ave}	θ (Gbit/s)
GDBF	1296	1/2	87810	75	222	1	2.00 (@$\alpha = 0.005$)	144.00	2.95 ($FER = 3e^{-4}$)	97.63
LFSR-PGDBF (S = 4Z = 216)	1296	1/2	100521 (+14.5%)	86	232	1	3.50 (@$\alpha = 0.012$)	86.11	2.88 ($FER = 4e^{-6}$)	104.65
IVRG-PGDBF (S = 4Z = 216)	1296	1/2	92645 (+5.5%)	79	232	1	3.50 (@$\alpha = 0.012$)	86.11	2.88 ($FER = 5e^{-6}$)	104.65
MS [24]	1296	1/2	720000	615	250	6	2.34 (@$\alpha = 0.025$)	23.08	1.29 ($FER = 1e^{-7}$)	41.86
MS [25]	648 - 1944	1/2 - 5/6	1023000	-	400	-	108 - 337 (Mbps) at $It_{max} = 20 - 25$ iterations 1.39 - 4.34 (Gbps) at $It_{ave} = 1.94$			

Table 4.3: Frequency and throughput comparison between GDBF decoder, PGDBF decoders, and MS decoders [24, 25]. QC-LDPC $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$, $Z = 54$.

As can be seen in Table 4.3, the hardware cost of the MS decoders is a lot larger than the BF-based decoders, and requires 7 to 10 times more area than the PGDBF. Our implementation of PGDBF allows to perform one iteration in $N_c = 1$ clock cycle, which results in a very important throughput gain of GDBF and PGDBF decoders over MS. We compared the average throughput of the decoders under two settings. For the same target performance, i.e. at FER= $1e{-}5$, the PGDBFs have a throughput 4 times larger than the MS of [24], and only 67% lower than the deterministic GDBF. The second setting corresponds to all decoders working at the same level of channel noise, i.e. $\alpha = 0.01$. In this case, the PGDBFs have a larger throughput than GDBF and a gain of two orders of magnitude in FER. Compared to MS, the PGDBF have poorer performance (FER= $5e^{-6}$ compared to FER= $1e^{-7}$), but with 2 times faster throughput. The conclusions are confirmed when comparing with the MS architecture of [25], in which the authors report an architecture for which the area is 10 times larger than our PGDBF solutions, and with a throughput smaller than the one in [24].

	Code length	Code rate	AREA (μm^2)	kGE	f_{max} (MHz)	N_c	$FER = 5e-7$		$\alpha = 0.03$	
							It_{ave}	θ (Gbit/s)	It_{ave}	θ (Gbit/s)
GDBF	9520	0.88	514760	173	100	1	2.1 ($@\alpha = 0.001$)	453.33	4.73 ($FER = 2e^{-4}$)	201.27
LFSR-PGDBF (S = M = 8Z = 1120)	9520	0.88	533394 (+3.61%)	182	100	1	3.95 ($@\alpha = 0.0025$)	241.01	4.68 ($FER = 2e^{-6}$)	203.42
IVRG-PGDBF (S = M = 8Z = 1120)	9520	0.88	533574 (+3.66%)	121	100	1	3.95 ($@\alpha = 0.0025$)	241.01	4.68($FER = 2e^{-6}$)	203.42
MS [26] 180nm	8192	0.875	11300000	-	317	-	$It_{max} = 15$, $\theta = 5.1$			
scaled to 65nm	8192	0.875	1470000	-	877.8	-	$It_{max} = 15$, $\theta = 14.12$			

Table 4.4: Frequency and throughput comparison between GDBF, PGDBF decoders and MS decoder from [26]. QC-LDPC $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$.

Another demonstration of the advantages of our PGDBF implementations is presented in Table 4.4, in which we considered LDPC code parameters that are especially interesting for storage applications. The code used for GDBF and PGDBF decoders is a QC-LDPC with parameters $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$. The average number of iterations is considered for the GDBF and PGDBF decoders, while we only have access to the maximum number of iterations for the MS architecture of [26]. For this long, high rate code, and $S = M$, the area overhead of our PGDBF compared to GDBF is slightly smaller than for the shorter code of Table 4.3, which means that our implementations are more and more efficient as the code rate and codeword length increase. Compared with the MS implementation of [26], we can see that our PGDBF implementations occupy 3 times less area, while having a throughput at least 4 times faster than the MS (taking into account the difference between It_{max} and It_{ave}).

Overall, we can conclude that only with a few percents of hardware overhead compared to the classical GDBF, our proposed implementations of the PGDBF represent an interesting and competitive solution for very high throughput decoders.

4.4.2 PGDBF Performance

In this section, we illustrate the difference in decoding performance of PGDBF decoders in comparison with MS and GDBF decoders, on the BSC channel. The MS decoder is a layered version with 6 quantization bits for the APP-LLR and 4 quantization bits for the extrinsic messages, with a maximum of $It_{max} = 20$ decoding iterations. We consider the regular LDPC codes for the simulations: a QC-LDPC code with parameters $d_v = 3$, $d_c = 6$, rate 0.5, $N = 1296$ (dv3R050N1296), a QC-LDPC code with $d_v = 4$, $d_c = 8$, rate 0.5, $N = 1296$ (dv4R050N1296) and a QC-LDPC code with $d_v = 4$, $d_c = 12$, rate 0.75, $N = 1296$ (dv4R075N1296). For the GDBF and the PGDBF, the maximum number of iterations is set to $It_{max} = 300$.

In Figure 4.15, we show the result of testing different implementations of the PGDBF decoder, i.e. the IVRG-PGDBF and the LFSR-PGDBF with different RS probabilities p_0, on the dv3R050N1296 code. We have also simulated the LFSR-PGDBF with varying values of the RS probabilities $p_0^{(k)}$ along the iterations, under the following setting: $p_0 = 0.9$ for the first 100 iterations, $p_0 = 0.7$ if $100 < k \leq 200$, and $p_0 = 0.5$ if $200 < k \leq 300$. We labeled this decoder V-LFSR-PGDBF. It can be seen that all PGDBF decoders have roughly the same performance, as well

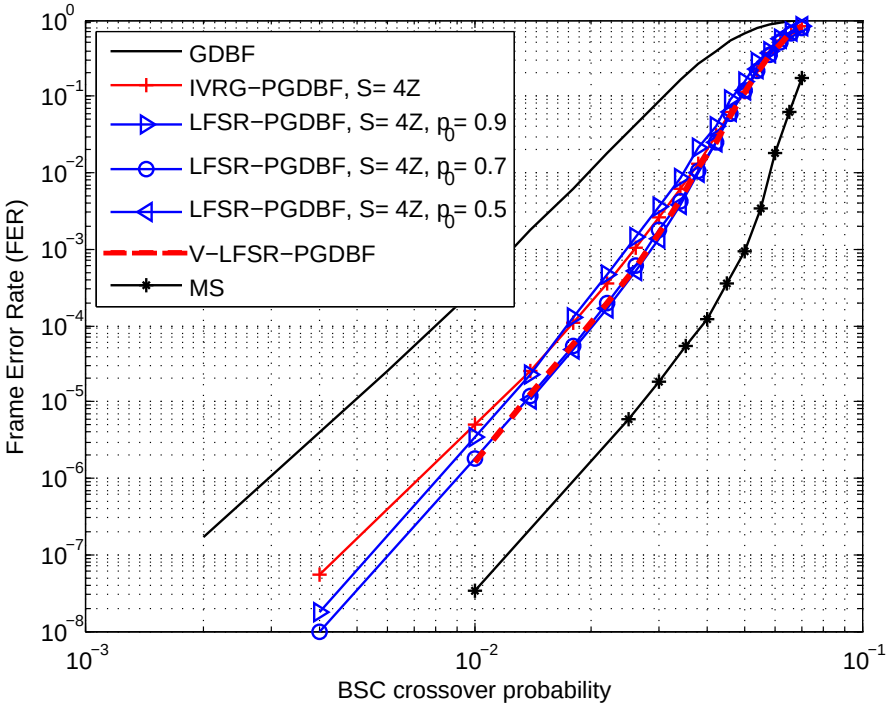
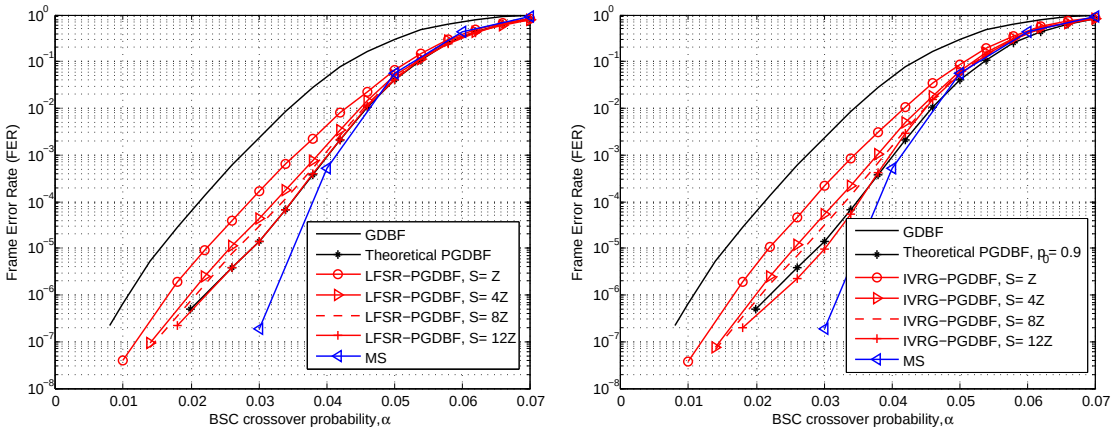


Figure 4.15: Decoding performance of GDBF, LFSR-PGDBF, IVRG-PGDBF and MS decoders on the LDPC code $d_v = 3$, $d_c = 6$, $N = 1296$ (dv3R050N1296).

as the V-LFSR-PGDBF, which confirms that a wide range of value of p_0 achieve approximately the same coding gain, and also that LFSR and IVRG versions are both competitive in terms of performance.

The PGDBF decoders perform halfway between the GDBF and the MS decoder. For the IVRG-PGDBF, the gain compared to GDBF decoder comes at only 6% chip area overhead and no throughput degradation as shown in Table 4.3. The performance loss compared to MS was expected, and could be acceptable for applications that are very demanding in energy/throughput.

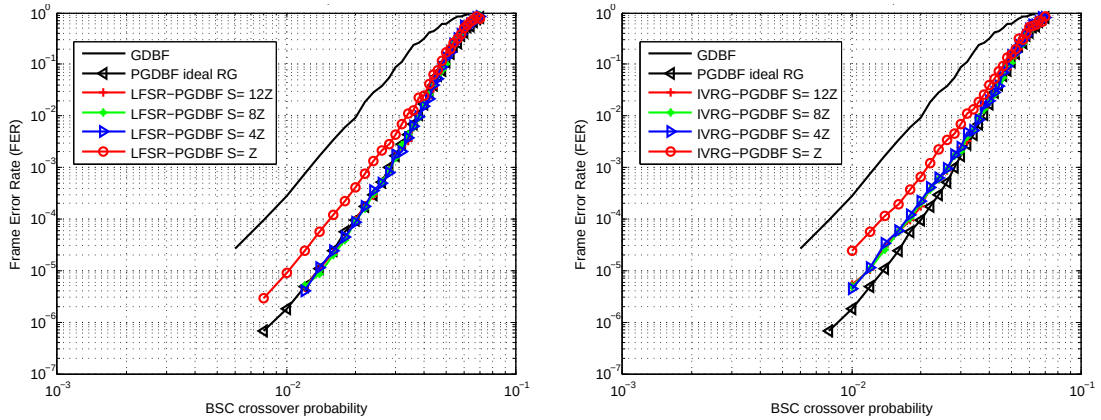
Fig. 4.16(a) and 4.16(b) show the decoding performance of PGDBF decoders for increasing values of the random sequence length S, for the dv4R050N1296 code. As expected, the greater value of S, the better decoding performance and $S = 12\,Z$ is sufficient to reach the performance of the theoretical PGDBF. The IVRG-PGDBF with $S = 12Z$ performs even better compared to the theoretical PGDBF at small α. For example FER $= 2e^{-6}$ for the IVRG-PGDBF compared to FER $= 4e^{-6}$ for the theoretical PGDBF, at $\alpha = 0.026$. This interesting behavior can be explained by the fact that IVRG-PGDBF has the feature of *adapting* the number of ones in the CN value sequence to the channel noise realization, by producing more ones when the channel contains many errors, and less ones when it contains only a few errors. This feature of the IVRG-PGDBF can be interpreted as a PGDBF with adaptive p_0 parameter, the adaptation coming from the channel quality. Another interesting result is that the PGDBF decoders approach closely the performance of the MS decoder especially in the waterfall region. This means that for $d_v = 4$ codes, the



(a) (b)

Figure 4.16: Effect of the RS length S on the decoding performance for the dv4R050N1296 code: (a). LFSR-PGDBF decoders ($p_0 = 0.9$), (b). IVRG-PGDBF

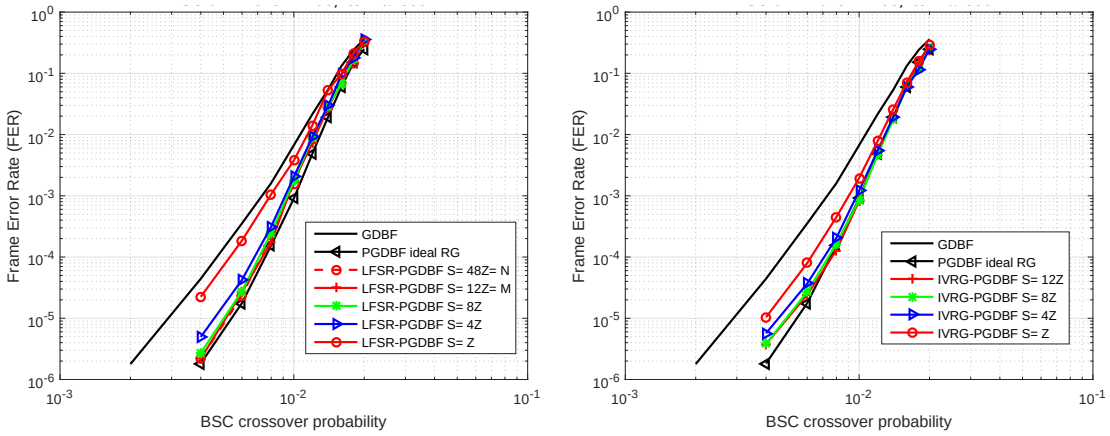
PGDBF solution is even more competitive than for $d_v = 3$ codes. We show more simulation results in Figure 4.17 of dv3R050N1296 and 4.18 for dv4R075N1296 to reconfirm all above conclusions.



(a) (b)

Figure 4.17: Effect of the RS length S on the decoding performance for the dv3R050N1296 code: (a). LFSR-PGDBF decoders ($p_0 = 0.7$), (b). IVRG-PGDBF.

Finally, we present on figure 4.19 the performance of our decoders for the $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$ of table 4.4. For this code, the PGDBF is especially very good as it close to the MS results in the waterfall region, and starts showing an error floor at $FER < 10e^{-5}$. This very good performance results compared to the deterministic GDBF comes at only 3.5% extra hardware cost, as indicated in table 4.4.



(a) (b)

Figure 4.18: Effect of the RS length S on the decoding performance for the dv4R075N1296 code: (a). LFSR-PGDBF decoders ($p_0 = 0.9$), (b). IVRG-PGDBF.

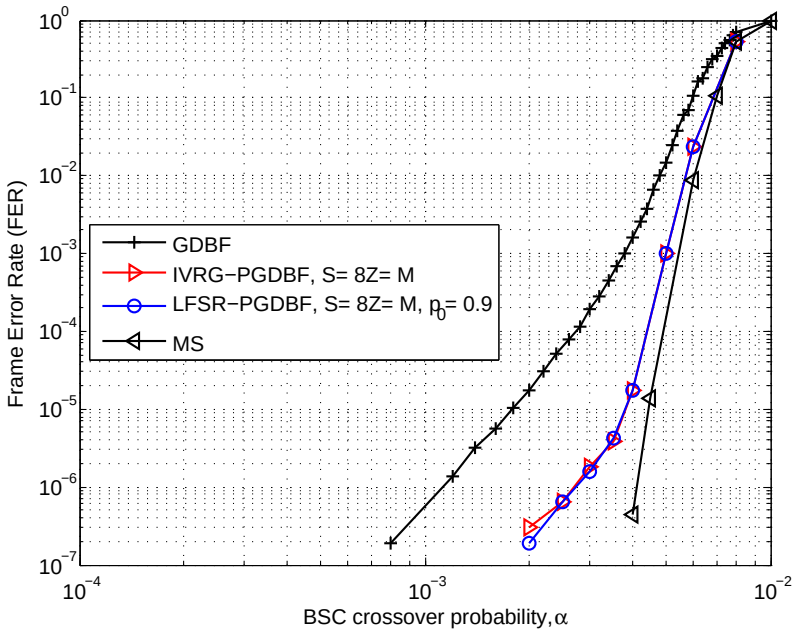


Figure 4.19: Decoding performance of GDBF, PGDBF ($It_{max} = 300$) and MS ($It_{max} = 20$) decoders on a QC-LDPC code with $dv = 4$, $Rate = 0.88$, $Z = 140$, $M = 1120$ and $N = 9520$.

4.5 Conclusion

We proposed an efficient hardware architecture to implement the PGDBF in this chapter. We have focused on minimizing the resource overhead needed to implement the random perturbations of the PGDBF and on the optimization of the maximum indicator unit. Our random perturbation block is based on the use of a short random sequence that is duplicated to fully apply the PGDBF decoding rules. We also propose two different methods to initialize the short RS, LFSR-based and IVRG-based, with equivalent hardware overheads but with different behaviours on different LDPC codes. We showed, by implementing the LFSR-PGDBF and IVRG-PGDBF decoders on ASIC, that the proposed random perturbations require a very small extra complexity compared to the GDBF. We further improve the decoding throughput of our BF decoders by optimizing the Maximum Indicator using the LCT maximum finder in order to shorten the critical path. Compared to the MS decoder, the proposed PGDBF implementation offer 5 to 7 times faster throughput and requires 7 to 10 times less chip area, at the cost of a performance degradation, which is in all our simulations smaller than all the known hard decision decoders. These advantages in throughput and area make our PGDBF decoders a competitive hard-decision LDPC decoding solution for current and future standards.

Chapter 5

A Quasi-Cyclic friendly architecture for LDPC decoders : the Variable-Node Shift Architecture

5.1 Introduction

This chapter presents a new decoding architecture for the QC-LDPC codes, called as Variable-Node Shift Architecture (VNSA). The VNSA deploys the homogeneous construction property of the QC-LDPC codes to shift the memory of the decoders while preserving the decoding operations properly as the conventional implementation architecture. It is shown in this chapter that the VNSA-based decoders significantly reduce the complexity and achieve the better decoding performance compared to the conventional decoder implementations. These advantages come from the fact that by shifting the memory of the decoders, the variable node computation can be processed differently when different types of variable nodes implemented. This dynamical processing helps the decoder break some trapping states and converges while the decoder with conventional implementation does not. The advantages also come when some variable node implementations in VNSA are simpler than those of the conventional implementation making the general complexity reduced. The chapter is presented as following. The VNSA principle is firstly presented in the generic form in Section 5.2 since it can be applied to different decoding algorithms. The major modifications of VNSA are highlighted by putting on comparison to the conventional implementation. The illustrations of VNSA applications on different types of LDPC decoders are presented in the next 2 sections (Section 5.3 and 5.4) for the edge-type memory and node-type memory decoders in which the VNSA is shown to be well adapted for all of these decoding types. Although the VNSA is an alternative method beside the conventional implementation, its advantages come when different functions are implemented in different hardware for VNs or CNs. These advantages are either gain in decoding performance and/or in

the complexity which is briefly discussed in Section 5.5. In order to more emphasize the advantages of the VNSA, an interesting example of the VNSA are shown which is the implementation of the Probabilistic Gradient Descent Bit Flipping basing on the VNSA (called VNSA-PGDBF) in Section 5.6. It is shown that the outstanding decoding performance of PGDBF is preserved in VNSA-PGDBF while the decoder complexity is significantly reduced and even smaller than the deterministic GDBF. A further simplified version of VNSA-PGDBF is also introduced, called as the imprecise VNSA-PGDBF (VNSA-IM-PGDBF). This VNSA-IM-PGDBF not only reduces importantly the complexity compared to the VNSA-PGDBF but also improves the decoding performance in some testing case. The synthesis results and decoding performance are presented in Section 5.7. Section 5.8 concludes the chapter.

5.2 The Variable-Node Shift Architecture

Since this chapter focuses on the decoders for QC-LDPC codes, some modifications on the indexing notations are newly introduced in order to ease the discussion. In QC-LDPC code, N VNs can be split into n_c groups of Z corresponding to n_c columns of H_B and similarly, M CNs can also be grouped into n_r groups of Z corresponding to n_r rows of H_B. We denote the j-th VN ($1 \leq j \leq Z$) belonging to the i-th column ($1 \leq i \leq n_c$) of the base matrix H_B as $v_{i,j}$, similarly the b-th CN ($1 \leq b \leq Z$) belonging to the a-th row ($1 \leq a \leq n_r$) as $c_{a,b}$. In Tanner graph, a VN $v_{i,j}$ connects to a CN $c_{a,b}$ when $H(a*Z+b, i*Z+j) = 1$. The set of CNs connected to the VN $v_{i,j}$ are denoted as $\mathcal{N}(v_{i,j})$ and similarly, $\mathcal{N}(c_{a,b})$ as a set of VNs connected to CN $c_{a,b}$. The VN and CN degree are defined as $|\mathcal{N}(v_{i,j})| = d_v$ and $|\mathcal{N}(c_{a,b})| = d_c$ $\forall i, j, a, b$. A vector $\mathbf{x} = \{x_{i,j} | 1 \leq i \leq n_c, 1 \leq j \leq Z\} = \{x_{1,1}, x_{1,2}, \ldots, x_{n_c,Z-1}, x_{n_c,Z}\} \in \{0,1\}^N$ is called a codeword if and only if $H\mathbf{x}^T = 0$. $\mathbf{x}$ is sent through a BSC channel and $\mathbf{y} = \{y_{i,j} | 1 \leq i \leq n_c, 1 \leq j \leq Z\} = \{y_{1,1}, y_{1,2}, \ldots, y_{n_c,Z-1}, y_{n_c,Z}\}$ denotes the output of this channel.

In order to highlight the novelty of VNSA, in this section, the conventional architecture of QC-LDPC decoders are firstly presented. On top of this conventional architecture, the VNSA is described and analyzed.

5.2.1 The Conventional Architecture of QC-LDPC decoders

LDPC decoders are generally implemented by the connection network blocks connecting two groups of implemented processing units, the variable node processing units (VNUs) and the check node processing units (CNUs). The computed messages are iteratively passed between the VNUs and CNUs through these connection networks during the decoding process. The generic and conventional architecture of LDPC decoders is presented in Figure 5.1 in which the circles represent the VNUs and squares represent the CNUs. The memory elements are signified by the clock signals *(clk)*. The main difference between QC-LDPC and the general LDPC implementations is that the interconnections are very constructive in QC-LDPC such that two consecutive VNUs connect to two consecutive CNUs (see in definition 1) as illustrated in the Figure 5.2.

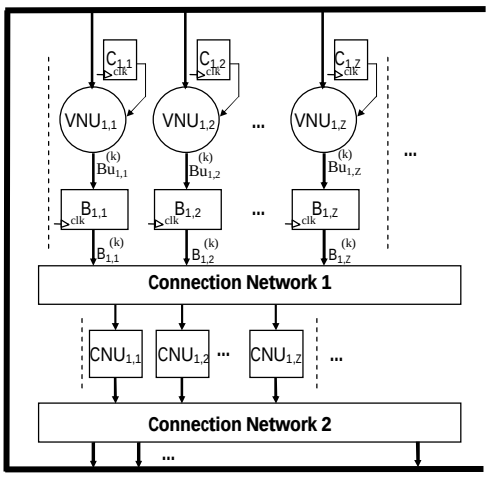


Figure 5.1: The conventional architecture of QC-LDPC.

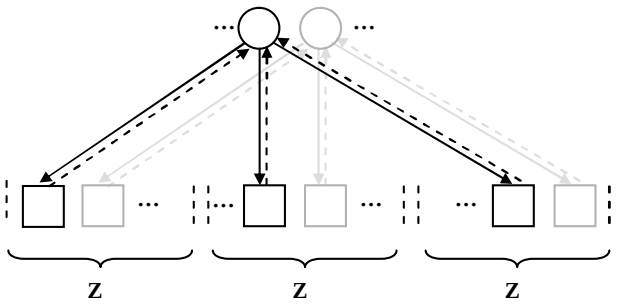


Figure 5.2: The Tanner graph of a QC-LDPC code.

Definition 1. *Two VNUs* v_{i_1,j_1} *and* v_{i_2,j_2}, $1 \leq i_1, i_2 \leq n_c, 1 \leq j_1, j_2 \leq Z$ *(or two CNUs* c_{a_1,b_1} *and* c_{a_2,b_2}, $1 \leq a_1, a_2 \leq n_r, 1 \leq b_1, b_2 \leq Z$*), are called to be consecutive if they are in the same column (row) of the base matrix* H_B, $i_1 = i_2$ ($a_1 = a_2$), *and are in the positions* j_1, j_2 (b_1, b_2) *such that* $j_2 = j_1 \% Z + 1$ ($b_2 = b_1 \% Z + 1$) *where* $\%$ *is the modulus operation.*

The memory elements B is implemented to store the intermediate passing messages and C are allocated to store the channel estimation. The size of these memory elements and their organization strongly depend on the decoding algorithms as well as the channel models. For the hard decision decoding algorithms such as GDBF, PGDBF on the BSC channel [15], B and C are 2 1-bit registers. For the soft decision decoding algorithms such as quantized MS, Layered MS on the AWGN channel [25][42], C is a register bank of size q where q is the quantization length while B contains multiple register banks of size q to store different extrinsic messages. We index B and C (as in Figure 5.1) to follow the indexing of VNUs and CNUs in QC-LDPC decoder above. Also, the superscript (k) indicates the values at iteration k. The organization of B is affected by different decoding algorithms and decoder scheduling can be found more detail in the Section 5.3 and 5.4.

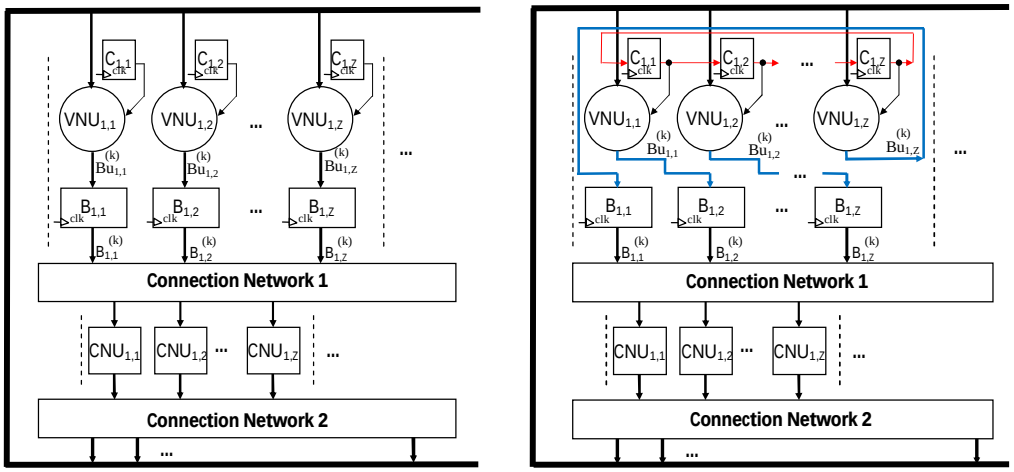
The operations of LDPC decoding algorithms on the generic architecture are briefly described as following. At the decoding initialization phase, B and C are initialized by the channel estimation. A decoding iteration can be roughly divided into 2 steps. In the first step, the messages stored in B are transmitted to the CNUs by the connection network 1. The updated messages are computed by these CNUs. In the second step, the updated messages from CNU are propagated by the connected network 2 to the corresponding VNU in which the new messages Bu are produced based on the CNU messages and the channel estimation from C memory. This updated messages Bu are stored into B memory when clock event occurs such that $B_{i,j}^{(k+1)} = Bu_{i,j}^{(k)}$, and the decoder starts a new decoding iteration. Since the memory B is updated after each clock cycle, one decoding iteration is finished by only one clock cycle. Although there are differences in the detail of decoder implementations such as the place to allocate the memory, the nature of stored messages..., this generic architecture can fully describe the operations of a generic LDPC decoder and is the generalization of many LDPC decoding implementations [24][25][42].

To conclude this section, it should be noted that in the generic architecture, the updated messages $Bu_{i,j}^{(k)}$ is stored in the same memory location $B_{i,j}$ for all iterations such that $B_{i,j}^{(k+1)} = Bu_{i,j}^{(k)}$. This makes each VN (and also, each CN) processed on the same VNU (CNU) all during the decoding process. The proposed VNSA, presented in the next section, makes change the storing location of Bu leading the VNs (and also the CNs) processed on different implemented hardware.

5.2.2 The Variable-Node Shift Architecture for QC-LDPC decoders

The key feature of the VNSA is that the consecutive memory elements are made connected to cyclically store the updated messages from one iteration to another during the decoding process. As described in Figure 5.3(b), the major change of VNSA compared to the conventional implementation (Figure 5.3(a)) is that the updated messages Bu of a VNU is stored in the memory element B of the consecutive VNU instead of its own memory. In order to do that, the connection doing $B_{i,j}^{(k+1)} = Bu_{i,j}^{(k)}$ is replaced by another connection doing $B_{i,(j\%Z+1)}^{(k+1)} = Bu_{i,j}^{(k)}$. After each iteration, all the intermediate messages of the VNs (in all column of the base-matrix) are cyclically shifted by 1. The same modification is applied on the channel estimation storing memory C, i.e. the memory sequence C is also cyclically shifted after each iteration, $C_{i,(j\%Z+1)}^{(k+1)} = C_{i,j}^{(k)}$. The global VNSA architecture is described in Figure 5.3(b).

In VNSA, the computation of VNs and CNs are proceeded in different VNUs and CNUs while preserving the decoding behavior identically to the conventional implementation. This can be clarified as following. In general, a VN is unique to another by its channel estimation value and its neighbor set. The computation of a VN is preserved, even in different computing VNU, only when the VNU receives correctly the value of the VN channel estimation and messages from its own CN

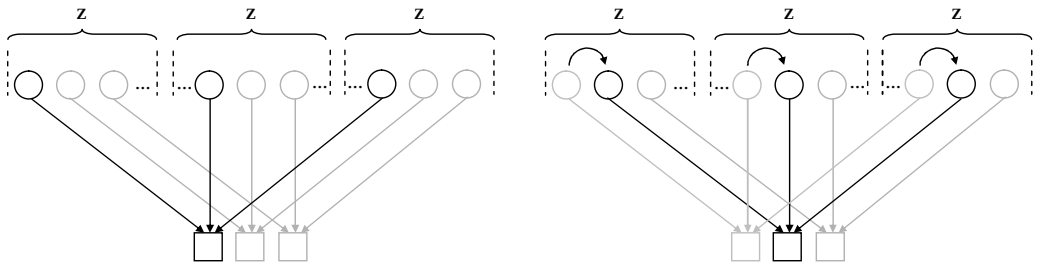


(a) (b)

Figure 5.3: The generic QC-LDPC decoder architectures: Figure 5.3(a) the conventional architecture. Figure 5.3(b) the proposed Variable-Node Shift Architecture (VNSA).

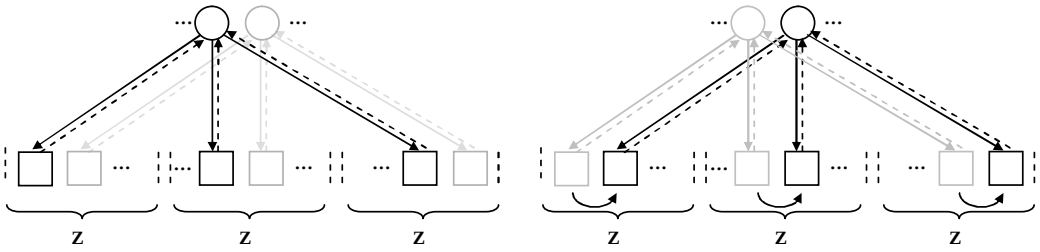
neighbors. Similarly, the computation of a CN can be proceeded on different CNU preserving the same results as the conventional implementation (which does not change the CNU) provided that the CNU receives accurately the messages from its own VN neighbors. The VNSA on QC-LDPC satisfies all of these conditions that, by the cyclic shift of the memory B and C after an iteration, the computation of VNs and CNs results are identical to the conventional implementation. Indeed, Figure 5.4 illustrates the arriving of messages to a CNU when the memory is cyclically shifted on a QC-LDPC decoder. The cycles represent the VNUs with memory, the squares represent the CNUs and the connection edges connects the neighbor nodes where the neighbor messages are conveyed on. It can be seen that due to the constructive connection network of QC-LDPC decoder, all the messages arrived to a CNU before the cyclic shift of the memory (Figure 5.4(a)) will be transmitted to a common, consecutive CNU after the cyclic shift (Figure 5.4(b)). This common CNU produces the same results (as the none cyclic shift) given that all CNU are identically implemented, which is the case of VNSA. For the VN computation, first, by the construction of VNSA, after each iteration, the channel estimation memory C is cyclically shifted and so, the channel value of a VN is conveyed to the input of the consecutive VNU. Second, due to the CNU shift in the CNU computation, Figure 5.5 shows that the CNU computed messages are also conveyed to the consecutive VNU. The VN computation results are therefore similar to those of the conventional implementation, and differ only on the computing location.

Along with the conventional implementation, the VNSA is an alternative implementation method with a negligible extra cost. Indeed, compared to the conventional implementation, the additional connections are required to cyclically shift the channel memory C which increase the overall complexity. However, the additional cost is negligible since this extra connections is much smaller than the interconnection



(a) (b)

Figure 5.4: With non-memory cyclically shift (a) and with memory cyclically shift (b), the messages are both well conveyed to a common CNU thank to the constructive implemented connections in QC-LDPC decoders.



(a) (b)

Figure 5.5: When a cyclic shift is applied on the memory of VNU, the messages from CNUs are also sent to the corresponding cyclic shift VNU.

network of the decoder. It is confirmed by the synthesis results in the Section 5.7.

5.3 The Variable-Node Shift Architecture for edge-type memory LDPC decoders: flooding MS and layered MS implementation illustrations

We show in this section that the VNSA can be applied widely on different type of LDPC decoding algorithms. Basing on the memory types in the decoder implementations, we classify these decoding algorithms into two types, the edge-type memory and the node-type memory. In the edge-type memory, the messages sent to an edge of VNU are different to another edge. Therefore, it is required to have the memory elements to store these messages between iterations. MS and its variants are the examples of edge-type memory decoders. In the node-type memory, all messages sent from a VNU to all connected CNUs are the same. For this reason, only one memory element is required to store the message for each VN. GDBF and PGDBF are the examples of node-type memory decoders. We illustrate that VNSA can be applied on edge-type memory decoders through 2 examples of MS decoder on different scheduling, flooding and layered scheduling. All the notations as well as algorithmic descriptions of MS decoder can be found in the appendix.

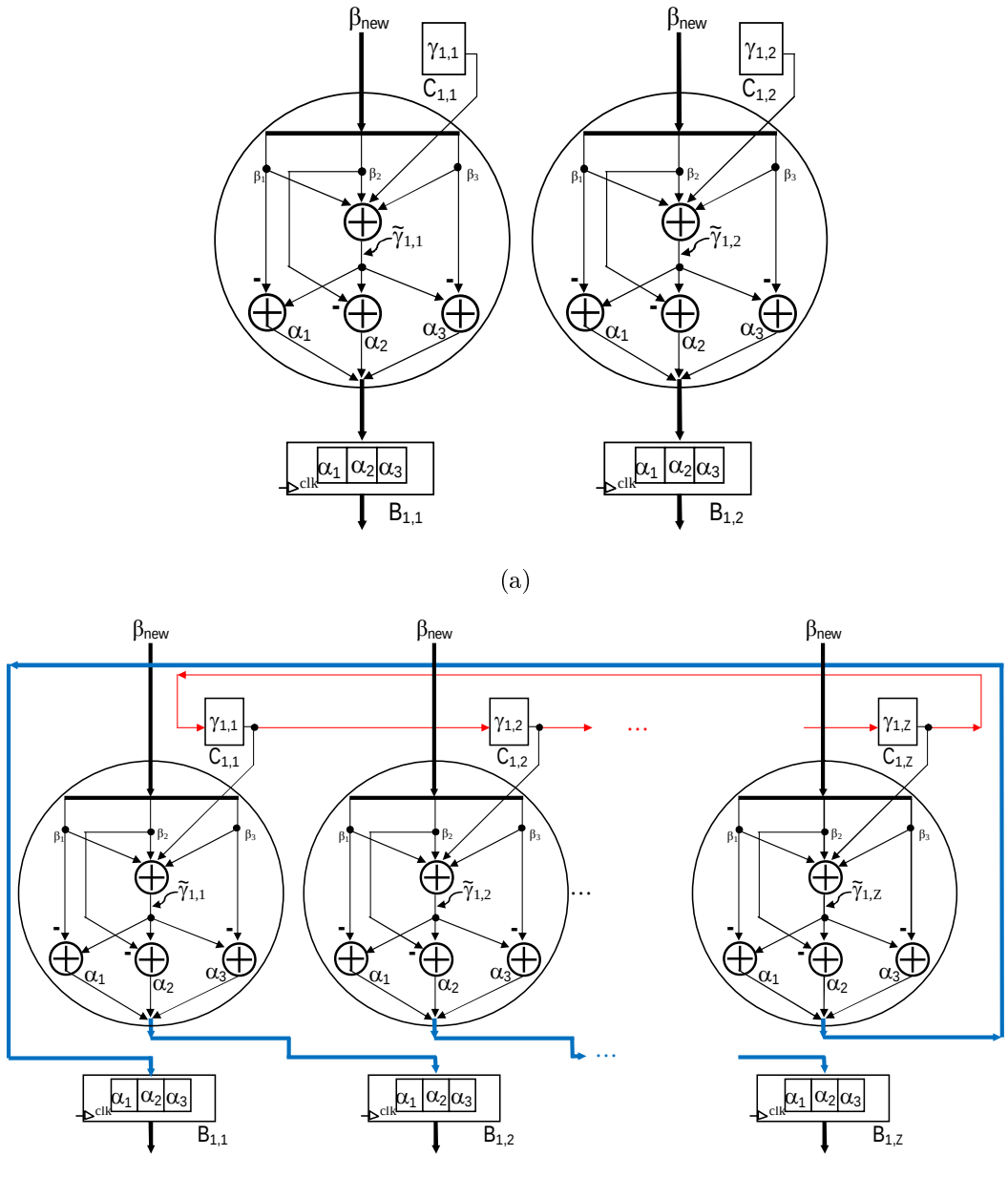


Figure 5.6: VNSA application on Flooding MS. Figure 5.6(a): 2 consecutive VNUs in the conventional flooding MS implementation. Figure 5.6(b): Z consecutive VNUs in a base-column of base matrix in VNSA-based implementation where the memory elements are cyclically shifted.

In flooding scheduling MS decoder, each VNU receives at the same time all updated messages from its connected CNs, β_{new} (The bolded lines, such as β_{new}..., in Figure 5.6 refer the grouped values from and to neighbor nodes). The VNU computes the new messages α concurrently and stores all into the message memory when clock event occurs . In the conventional implementation (see Figure 5.6(a)), these newly computed α messages are stored to the corresponding VNU messages memory elements, *i.e.* α messages computed by $VNU_{i,j}$ are stored in $B_{i,j}$, while in VNSA (see Figure 5.6(b)), they are stored to the consecutive message elements, *i.e.* α messages computed by $VNU_{i,j}$ are stored in $B_{i,j\%Z+1}$. The channel estimation memory, which stores the channel estimation values $\gamma_{i,j}$, is also cyclically shifted such that $\gamma_{i,j\%Z+1}^{(k+1)} = \gamma_{i,j}^{(k)}$. Since memory is assumed to implement only in B and C, for both implementations, the conventional and VNSA-based MS implementations, one iteration is finished in each clock cycle.

Another example of VNSA application on edge-type memory decoders is on the layered scheduling MS described in Figure 5.7. Different to the flooding scheduling, the VNU in layered MS receives only one β message from one connected CNU at a time. In the layered MS conventional implementation (the VNU architecture is described in Figure 5.7(a)), in each VNU and each clock cycle, this newly received β message is stored into the message memory, B, and at the same time, it is also used to compute the new APP value, $\tilde{\gamma}_{new}$. The computed $\tilde{\gamma}_{new}$ is stored to the $\tilde{\gamma}$ memory, C. Note that, at the decoding initialization phase, channel estimation value γ is initialized in $\tilde{\gamma}$ memory and β memory elements are reset. In the same clock cycle, the new α message, α_{new} - used for the next layer computation, is computed using the new computed $\tilde{\gamma}$ and β of the next layer read from B. A decoding iteration is finished after all elements in β memory are read and updated by the new values. This conventional implementation can be found in several works in literature such as [24][42].

The major change in layered MS VNSA-based implementation is the location to store the new received β. The new β is redirected to the consecutive memory (see Figure 5.7(b)) and at the same time, it is also used to produce the $\tilde{\gamma}_{new}$ in the current VNU. The β values are stored and $\tilde{\gamma}$ are updated continuously until the β value of the last layer read. When the value β_{new} of the last layer is stored in the consecutive memory, the computed $\tilde{\gamma}_{new}$ is also triggered to store into the consecutive $\tilde{\gamma}$ memory by the 2 to 1 multiplexer. The identical decoding behavior between VNSA-based and conventional implementations is confirmed by simulations results in the Figure 5.8.

5.4 The Variable-Node Shift Architecture for node-type memory LDPC decoders: GDBF implementation illustration

As mentioned above, in the node-type memory decoders, the messages are sent equally to all connected neighbors from the VNUs (as well as the CNUs). Therefore, only one memory element is implemented along to each VNU to store the intermediate message during the decoding process. This memory element is updated after

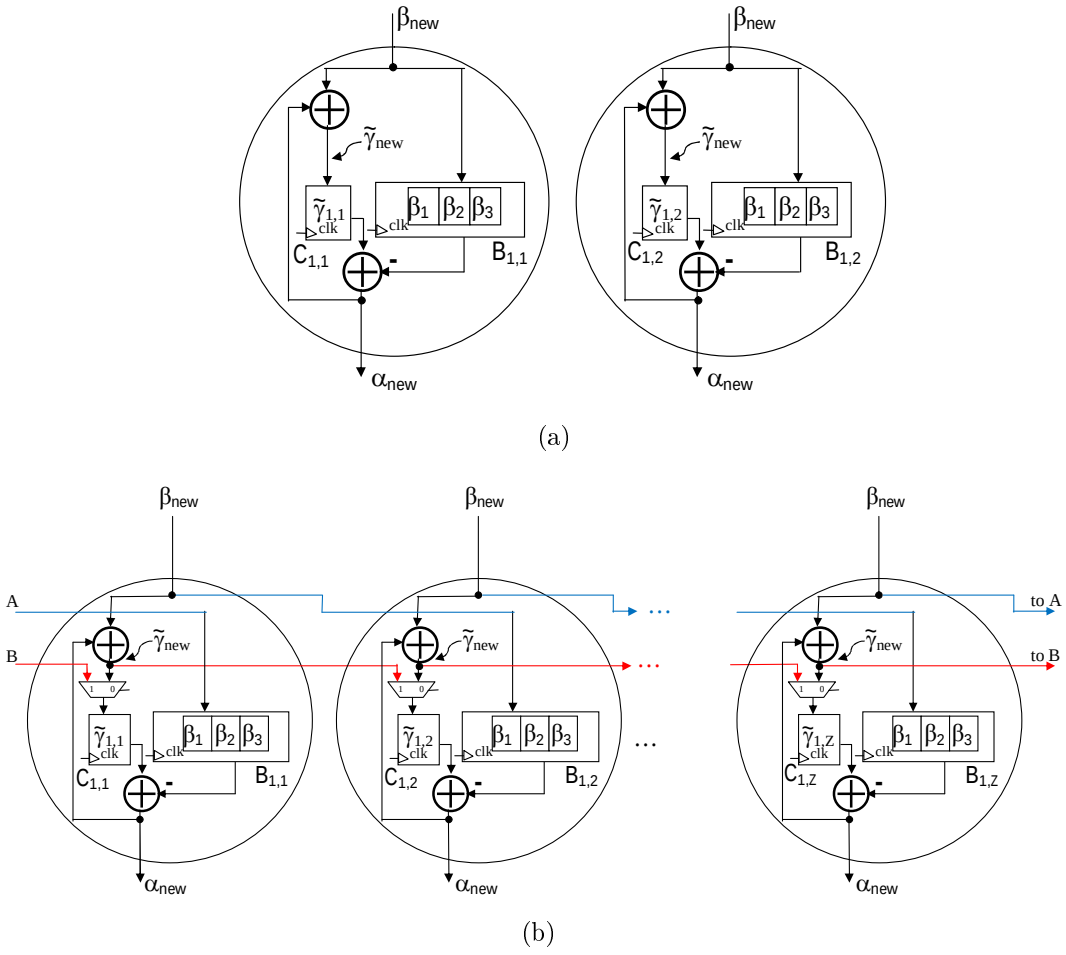


Figure 5.7: VNSA application on Layered MS. Figure 5.7(a): 2 consecutive VNUs in the conventional layered MS implementation. Figure 5.7(b): Z consecutive VNUs in a base-column of base matrix in VNSA-based implementation where the memory elements are cyclically shifted.

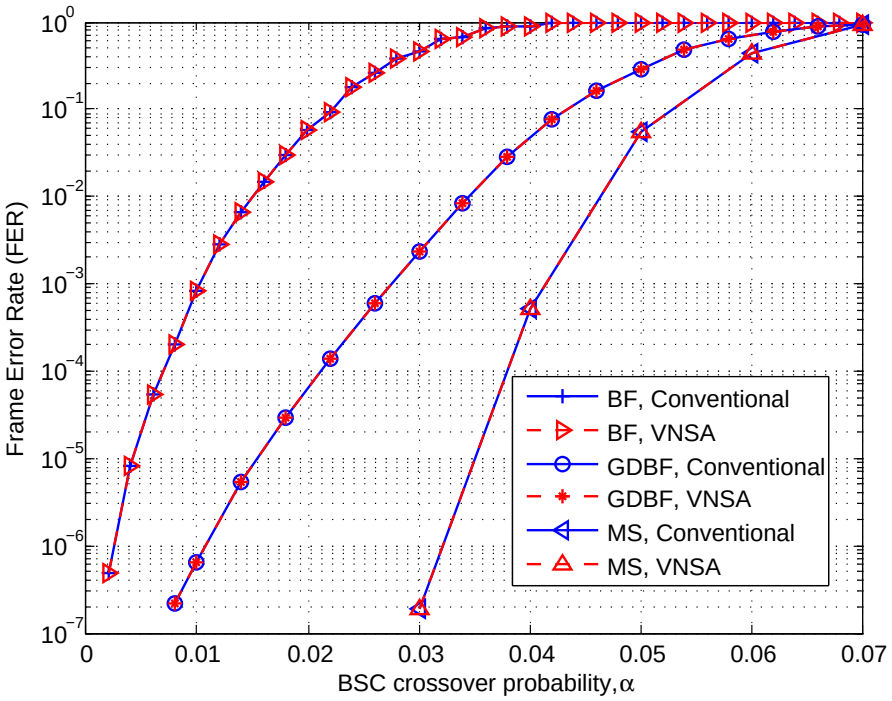


Figure 5.8: Performance comparison between BF, GDBF, Quantized flooding MS LDPC decoders both conventional and VNSA-based implementations for the regular QC-LDPC code ($d_v = 4, d_c = 8, Z = 54$), ($N = 1296, M = 648$).

each iteration.

We illustrate in the Figure 5.9 that VNSA can also be applied on the node-type memory LDPC decoder by the implementation of hard-decision GDBF decoder. In VNU of the conventional implementation of GDBF as in Figure 5.9(a), a register is implemented as the node memory to store the intermediate decision ($v_{i,j}$) and a register is allocated to store the channel estimation ($y_{i,j}$). The value of $v_{i,j}$ register is sent to the connected CNs and it is also used to update the new value for the next iteration. The updated value is computed by XOR-ing the current value to the equality block output as in Figure 5.9(a) (the detailed operations can be found in Chapter 4). Note that, this computed value is propagated to store back in the same ($v_{i,j}$) register. In VNSA-based implementation (Figure 5.9(b)), the updated value is directed to the consecutive memory element, ($v_{i,(j\%Z+1)}$). Also, a connection is built to convey the channel estimation register, ($y_{i,j}$), to the consecutive register, ($y_{i,(j\%Z+1)}$).

The decoding behavior of VNSA-based GDBF is reconfirmed to be identical to the conventional GDBF implementation by simulation as shown in Figure 5.8. We also show another node-type memory hard decision decoder, the Bit Flipping, which is also implemented by VNSA and exhibits the similar behavior with the conventional BF implementation in the same figure.

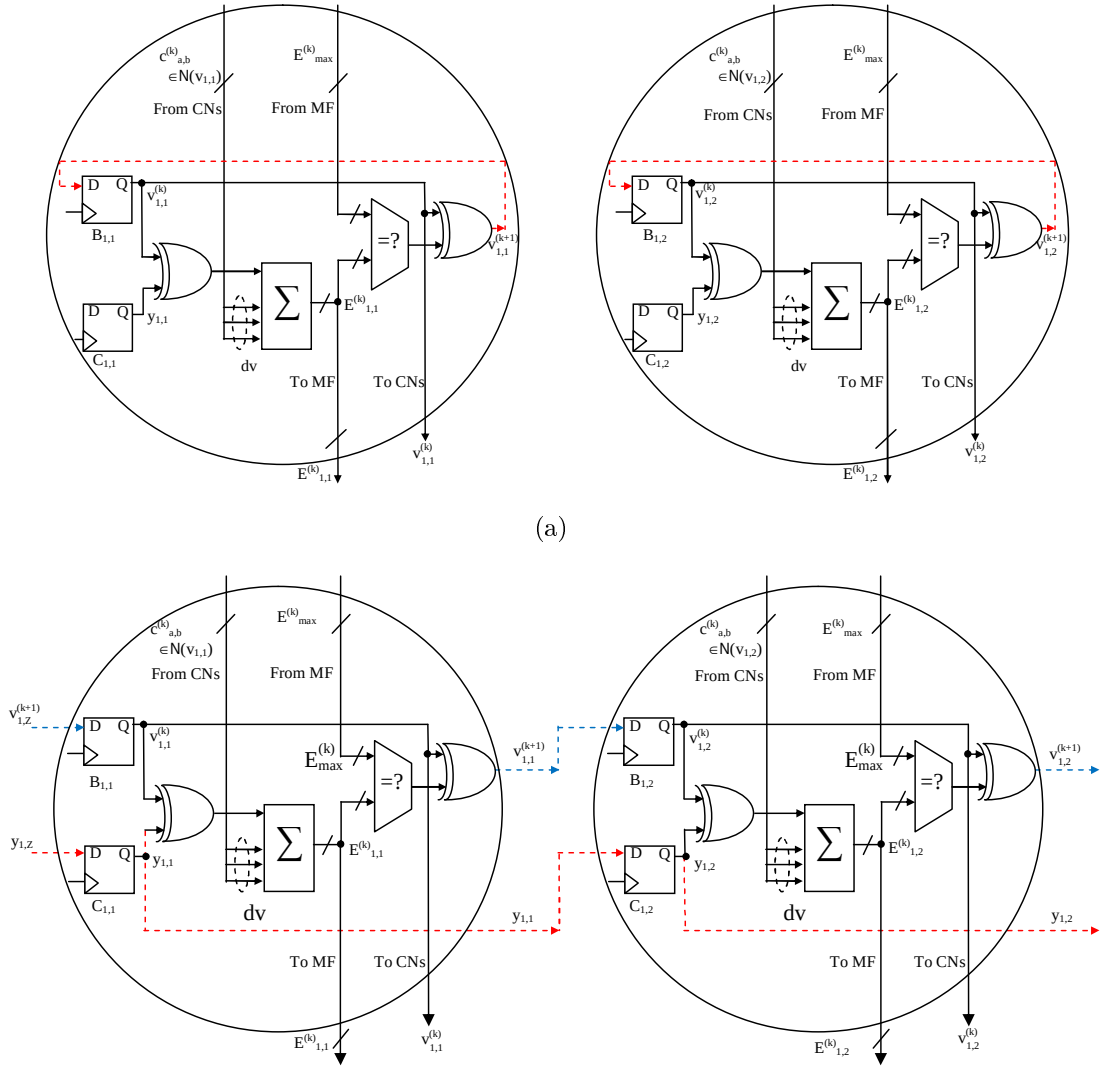


Figure 5.9: VNSA application on GDBF decoder. Figure 5.9(a): 2 consecutive VNUs in the conventional GDBF implementation. Figure 5.9(b): 2 consecutive VNUs in a base-column of base matrix where the node memory elements are cyclically shifted.

5.5 The advantages of VNSA-based LDPC decoders with different type of VNUs

Although the decoder with VNSA can be seen as an alternative implementation method for LDPC decoders, the advantages of using VNSA, in term of decoder complexity and decoding performance, may come when different types of VNUs (and/or CNUs) are implemented. The VNSA can be seen as a hardware distribution solution in some decoding methods from which the gain on performance and complexity can be obtained. Indeed, in order to improve the decoding performance, some decoding algorithms require the multiple functions implemented in each VNU and during the decoding process, the decoder switches from one function to another. An motivating example is the FAID diversity in [36] where the authors applied multiple FAID functions in each VNU and approached the Maximum Likelihood decoding. The drawback of this algorithm is that the VNU is a lot bigger since many functions are implemented while only one of them is used at a time. The VNSA solves this problem by distributing only 1 function in a VNU and cyclic shift the VNs through different implemented VNUs. A VN can still be processed by different functions during the decoding process while the total complexity can be significantly reduced. An example is illustrated in Figure 5.10. For the sake of simplicity, only 2 functions (marked as 1 and 2 and in 2 different colors) are illustrated. Figure 5.10(a) shows the conventional implementation with 2 functions implemented in each VNU leading to a higher complexity. The VNSA is applied in Figure 5.10(b) where only the function 1 or 2 implemented in each VNU and the VNs are cyclically shifted through these implemented VNUs.

The VNUs complexity of the VNSA implementation compared to those of the conventional is expressed as following where p_0 is the ratio of function 1 implemented over all VNUs implemented in VNSA (assuming $p_0 \geq 0.5$), $\mathcal{C}_{12}$ refers the hardware complexity to implement the two functions in a VNU (as VNUs in Figure 5.10(a)), $\mathcal{C}_1$ and $\mathcal{C}_2$ are the complexity to implement function 1 and 2 separately (as VNUs in Figure 5.10(b)):

$$\xi = \frac{\mathcal{C}_{VNSA}}{\mathcal{C}_{conventional}} = \frac{p_0\mathcal{C}_1 + (1-p_0)\mathcal{C}_2}{\mathcal{C}_{12}} \tag{5.1}$$

Due to the hardware reusing, $\mathcal{C}_1 + \mathcal{C}_2 \geq \mathcal{C}_{12}$ then

$$\xi \geq \frac{p_0\mathcal{C}_1 + (1-p_0)\mathcal{C}_2}{\mathcal{C}_1 + \mathcal{C}_2} = p_0 + \frac{1-2p_0}{1+\dfrac{\mathcal{C}_1}{\mathcal{C}_2}} \tag{5.2}$$

Depending on the complexity ratio $\dfrac{\mathcal{C}_1}{\mathcal{C}_2}$ and the hardware reutilisation (ratio $\mathcal{C}_1 + \mathcal{C}_2$ over $\mathcal{C}_{12}$), the useful region of VNSA complexity efficiency is plotted as the shaded region in Figure 5.11. It can be seen that with the complexity $\mathcal{C}_1$, $\mathcal{C}_2$ fixed, the

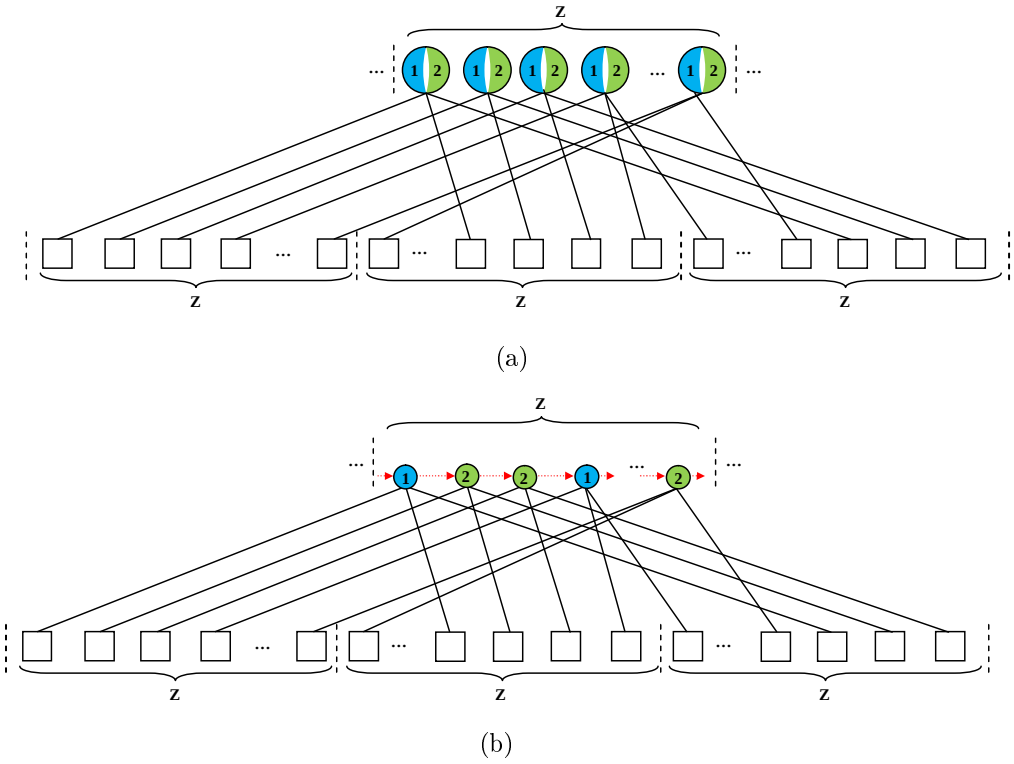


Figure 5.10: An implementation example of LDPC decoding algorithms where multiple functions are implemented in each VNU (Figure 5.10(a)) and an application of VNSA by distributing required functions in different VNUs and cyclically shift the VNs through these implemented VNU (Figure 5.10(b)).

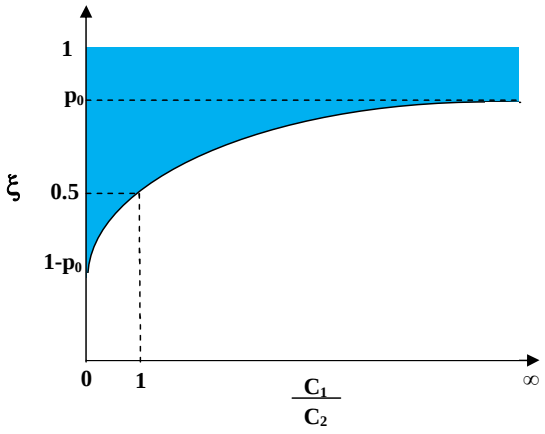


Figure 5.11: The hardware efficiency of VNSA over the conventional implementation.

complexity gain ξ may be controlled by p_0. When there is no hardware reutilisation in $\mathcal{C}_{12}$, *i.e.* $\mathcal{C}_1+\mathcal{C}_2=\mathcal{C}_{12}$, ξ is lower pounded. In the particular case where complexity of function 1 and 2 are equivalent, *i.e.* $\frac{\mathcal{C}_1}{\mathcal{C}_2}=1$, then $\xi=0.5$. It is interesting to consider the case where a VNU type in VNSA is simpler than the other, *i.e.* $\mathcal{C}_1 >> \mathcal{C}_2$, which is the case when VNSA applied in the PGDBF implementation presented in the next section.

5.6 Implementations of PGDBF with Variable-Node Shift Architecture

The key feature of VNSA-based PGDBF implementations is that 2 types of VNUs are implemented and located in an arbitrary order. These 2 types of VNUs are designed by mimicking the operations of VNUs in the conventional PGDBF corresponding to 2 possible values of random signal (0 or 1). With the cyclic shift property of VNSA, each VN can meet one of two VNU types with arbitrary order during the decoding process which imitates the PGDBF operations. The decoding behavior is shown to be similar to CSTS-PGDBF in previous chapter while no random generator is required. Furthermore, the 2 implemented VNUs types are simpler than the conventional VNU which reduces the decoder complexity even smaller than the deterministic GDBF.

5.6.1 The implementation of PGDBF with Variable-Node Shift Architecture

In the implementation of PGDBF using the VNSA (denoted as VNSA-PGDBF), 2 types of VNU are introduced which are illustrated by 2 types of cycles in Figure 5.12. To ease the discussion, we also reproduce the conventional PGDBF implementation with CSTS RG in previous chapter in Figure 5.13 with the VNU detailed circuit. In the this VNU, an AND gate is required to incorporate the random signal to the result of the equality comparator. When the random signal is 1's, this AND gate passes the value of the equality comparator to its output since $X\mathbf{and}1=X$. When the random signal is 0's, this AND gate reset its output to 0 regardless the equality comparator results since $X\mathbf{and}0=0, \forall X$.

The VNU type 1 of the VNSA-PGDBF is illustrated by the black solid cycle in Figure 5.12. The type 1 VNU in VNSA-PGDBF mimics the operation of VNU in PGDBF with the random signal fixed to 1's. With the fixed by 1's at the random signal input, $R_{i,j}^{(k)}$, the equality comparator output is propagated directly to the XOR2 input and the AND gate can be removed without changing the VNU behavior compared to conventional VNU (see Figure 5.12).

The second type (type 2) in VNSA-PGDBF, denoted as the red dashed cycle in Figure 5.12, is introduced by imitating the operation of the VNU in PGDBF with the random signal is equal to 0's. In the VNU of conventional PGDBF implementation (Figure 5.13), the AND gate produces 0's regardless the equality comparison results when the random input, $R_{i,j}^{(k)}$, is 0's ($X\mathbf{and}0=0, \forall X$). The VN updated value for the next iteration, $v_{i,j}^{(k+1)}$, is the one of current iteration, $v_{i,j}^{(k)}$, since $v_{i,j}^{(k+1)}=$

$v_{i,j}^{(k)}\mathbf{XOR}0 = v_{i,j}^{(k)}$. The type 2 simply propagates directly the current value $v_{i,j}^{(k)}$ to the output by removing the XOR2 gate compared to conventional VNU. More specially, the AND gates and equality comparator can also be removed without affecting to VNU operation (see Figure 5.12).

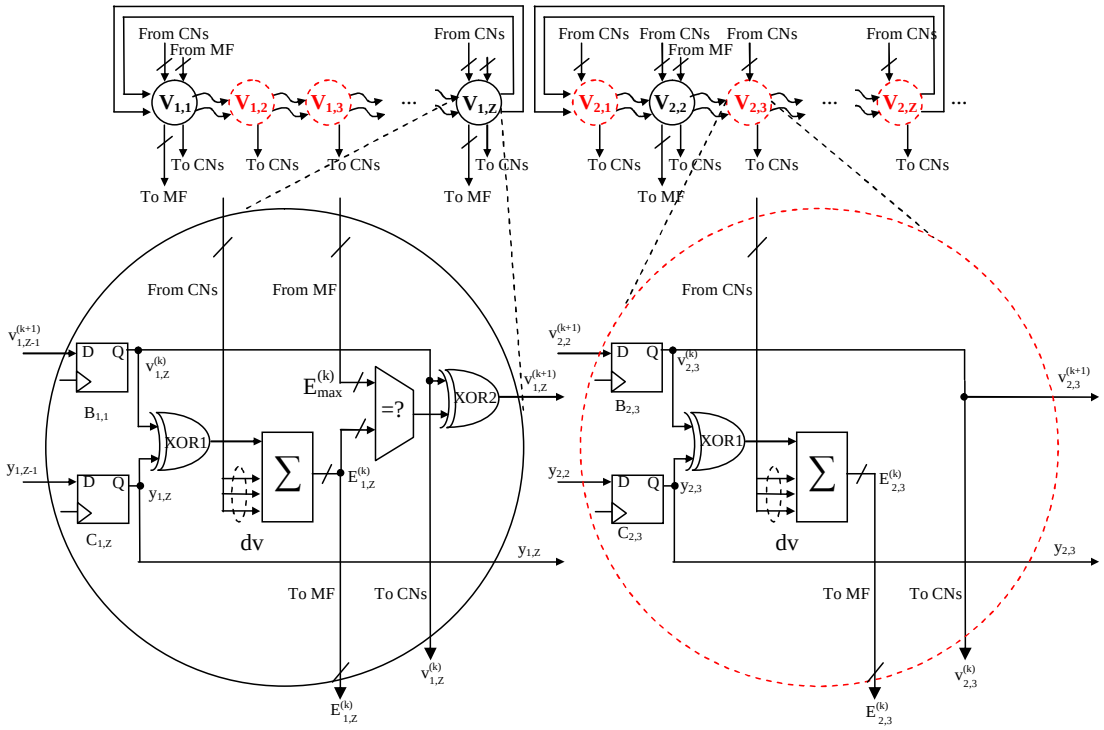


Figure 5.12: The implementation of VNSA-PGDBF decoder.

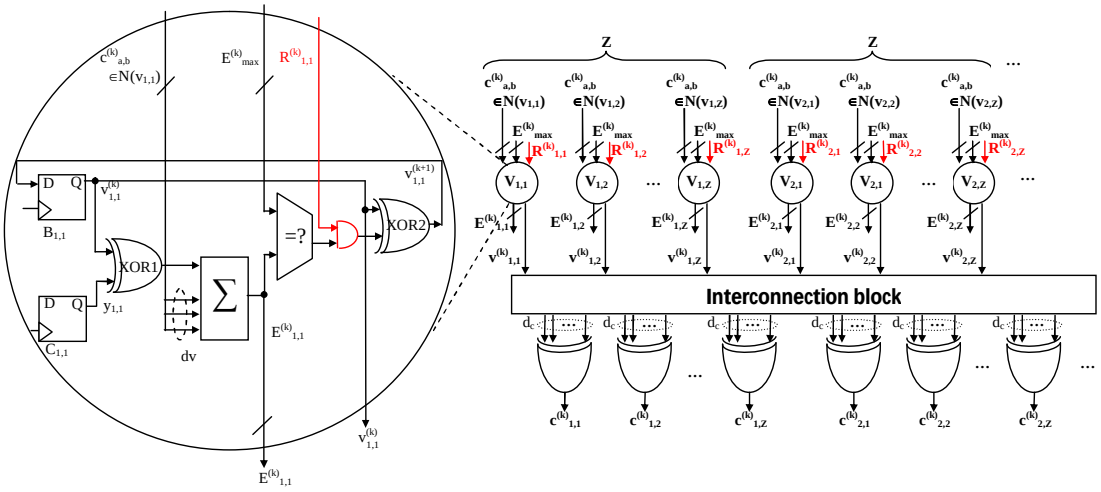


Figure 5.13: The conventional implementation of PGDBF.

With the two types of implemented VNUs accompanied by the VNSA principle

- cyclically shift the VN through different VNUs from iteration to another, the operations of PGDBF decoding algorithm can be performed. Indeed, we denote the ratio of type 1 VNUs over all VNUs as p_0 and of type 2 as $(1 - p_0)$. The cyclic shift of VNSA makes each VN seeing VNU type 1 with ratio p_0 and type 2 of ratio $(1 - p_0)$. Although the distribution of type 1 and type 2 VNUs could be arbitrary, we restrict in our VNSA-PGDBF by having exactly $Z * p_0$ VNUs type 1 and $Z * (1 - p_0)$ VNUs type 2 in the total of Z VNUs of a base column. It is more special that the operation of VNSA-PGDBF is identical to the version of Cyclically Shift Truncated Sequence PGDBF (CSTS-PGDBF) decoder introduced in Chapter 4 with the truncated sequence R_t of size $S = Z$ in Figure 5.14. The speciality is that instead of cyclic shift the random sequence in CSTS-PGDBF, in VNSA-PGDBF, the VNs are shifted to the VNUs and are processed by the same operating function as in CSTS-PGDBF while it does not require the random generator. The VNSA-PGDBF is even more improving in decoding performance by shuffling the type 1 and type 2 VNUs in a base column which operates similarly to the random generator in CSTS-PGDBF decoder with the the hard-wires shuffled in Figure 5.15. Indeed, as shown in Figure 5.16(a), a performance loss is observed when the CSTS size $S = Z$ with only *copy and concatenate* (see Figure 5.14) while with the same sequence but the *hardwire shuffled* (see Figure 5.15), the decoding performance is improved, approaching the theoretical decoder.

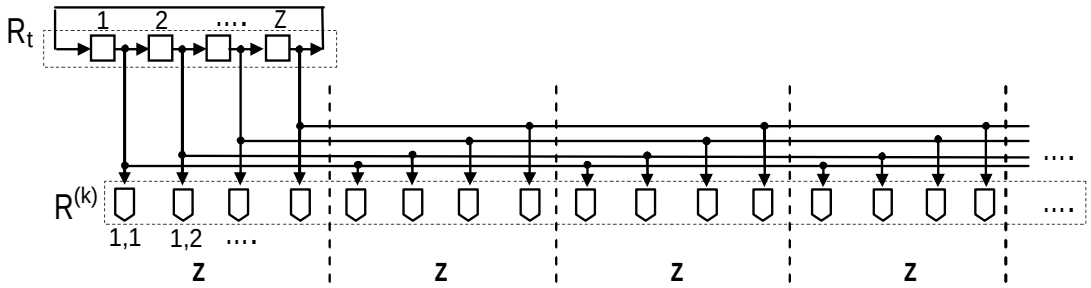


Figure 5.14: The optimized probabilistic signals generator proposed in Chapter 4.

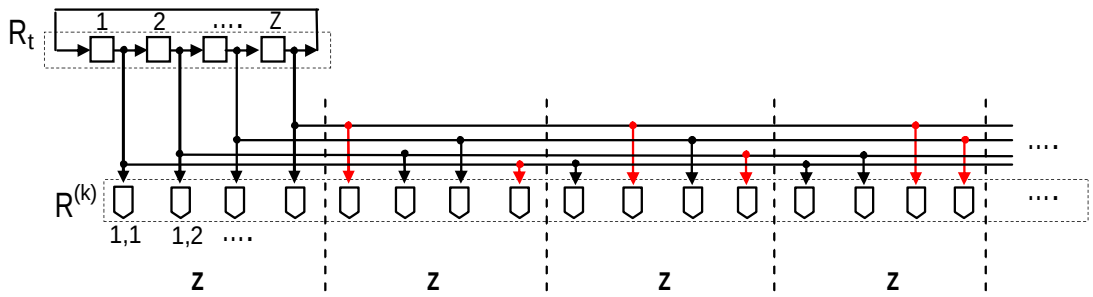
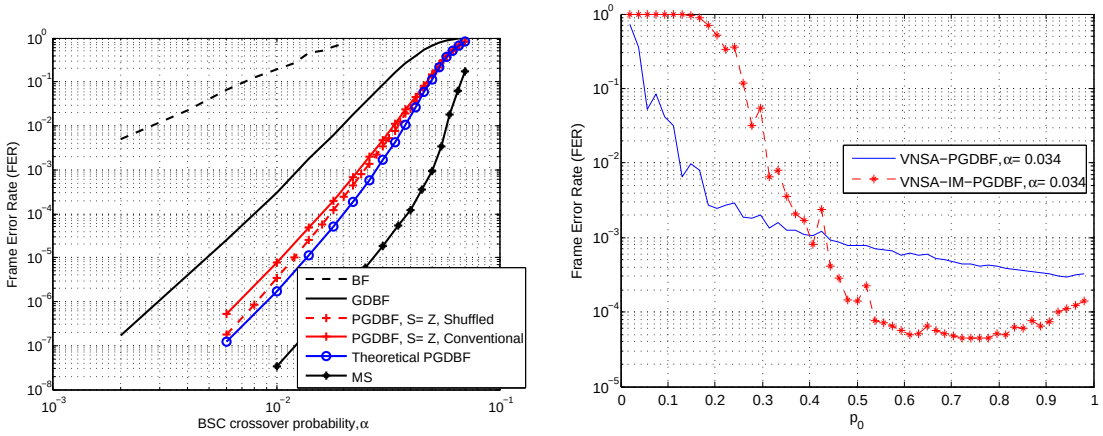
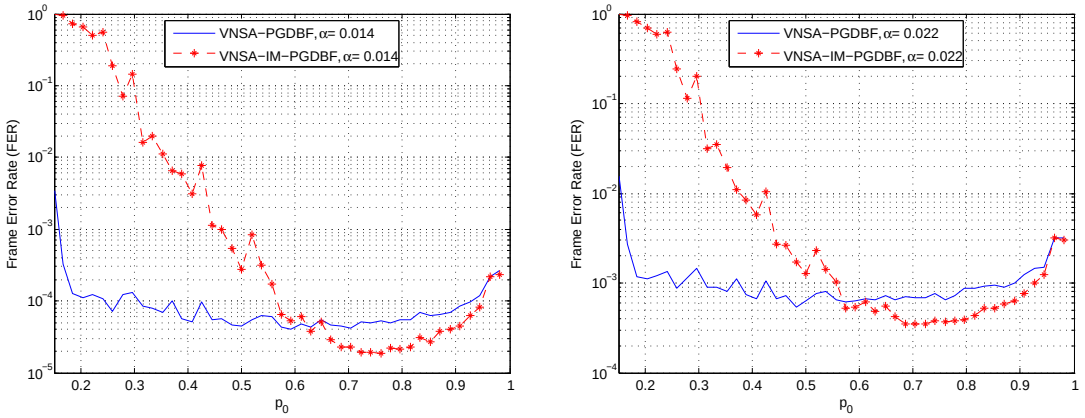


Figure 5.15: The probabilistic signals generator proposed in Chapter 4 with the hardwire shuffled.



(a) (b)

Figure 5.16: Figure 5.16(a) Decoding performance comparison of PGDBF decoder implemented in Chapter 4 with the hardwire connections in random generator shuffled. Figure 5.16(b) The statistical on decoding performance of VNSA-PGDBF and VNSA-IM-PGDBF as a function of p_0 on the $((d_v, d_c) = (4, 8)$, $Z = 54$, $N = 1296$ and $M = 648)$ LDPC code.



(a) (b)

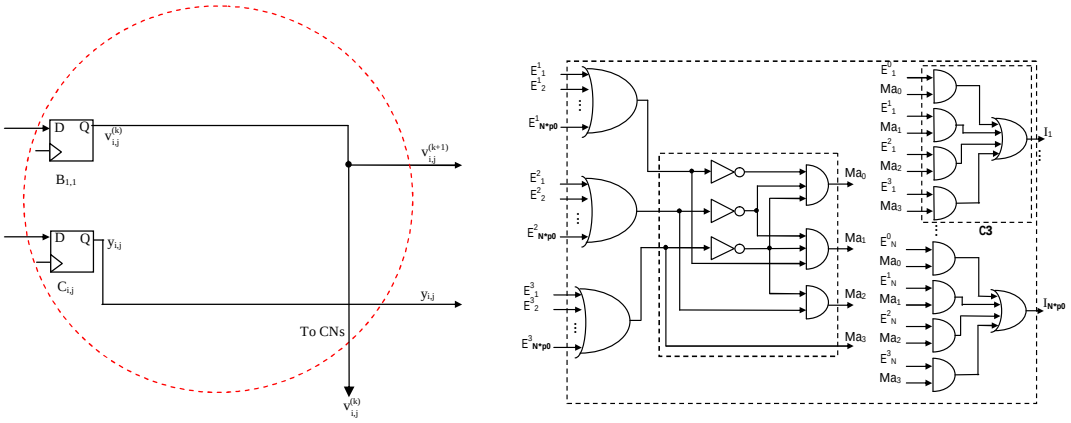
Figure 5.17: The statistical on decoding performance of VNSA-PGDBF and VNSA-IM-PGDBF as a function of p_0 on the $((d_v, d_c) = (3, 6)$, $Z = 54$, $N = 1296$ and $M = 648)$ LDPC code.

5.6.2 An imprecise implementation of PGDBF with Variable-Node Shift Architecture

We push further the simplification of VNSA-PGDBF by proposing another trivial VNU type (type 3) in Figure 5.18(a). The global decoder architecture is similar to the one in Figure 5.12 where the VNU type 2 are replaced by the VNU type 3. We refer this implementation as VNSA-IM-PGDBF. In the type 3 VNU, all the computation circuits are all omitted, the VN $v_{i,j}^{(k)}$ is preserved to the next iteration ,$v_{i,j}^{(k+1)} = v_{i,j}^{(k)}$, and its behavior is similar to the VNU in conventional PGDBF having $R_{i,j}^{(k)} = 0$. The difference comes from the fact that, this VNU type does not compute the energy value and makes the MI even simpler. The MI in VNSA-IM-PGDBF needs only to find the maximum energy in the list of $N * p_0$ input values compared to N conventionally. We apply the same technique in chapter 4 to implement the MI for the VNSA-IM-PGDBF decoder. This new MI presented in Figure 5.18(b) has only $N*p_0$ input values which come from $N*p_0$ VNU type 1. The energy values are represented in a *one-hot format*, d_v $(N * p_0)$-inputs (instead of d_v (N)-inputs) OR-gates are used. By eliminating all computation circuits and by having the lower complexity MI, the decoder complexity is more reduced shown in Section 5.7.

The issue of VNSA-IM-PGDBF decoder is the fact that the founded maximum value of energy may not the true maximum. However, we have conducted a statistical analysis on the effect of p_0 to the decoding performance in Figure 5.16(b) and Figure 5.17. It is surprisingly that for the 2 tested codes, the impreciseness introduced by the imprecise MI provides a better decoding performance compared to those of the precise MI implementation. Although the range of p_0 maintaining the good decoding performance of VNSA-IM-PGDBF is narrower to the one of VNSA-PGDBF, the decoding performance of VNSA-IM-PGDBF on the range of $p_0 \geq 0.6$ is always better than VNSA-PGDBF for the $((d_v, d_c) = (3, 6)$, $Z = 54$, $N = 1296$ and $M = 648)$ test code (Figure 5.17). In the test code $((d_v, d_c) = (4, 8)$, $Z = 54$, $N = 1296$ and $M = 648)$ (Figure 5.16(b)), the performance gap is even more significant and is on the wider range of p_0. Since the value of p_0 affects to the decoding complexity reduction as analyzed in Section 5.5, the statistics also reveal that the optimal value of $p_0 \approx 0.7$ for the 2 test codes which maximize the decoding performance gain and the reduced complexity.

Although the theoretical explanation on the superiority of VNSA-IM-PGDBF over VNSA-PGDBF is still not available, it is coherent to the proposition of decoder-dynamic shift PGDBF (DDS-PGDBF) with previously computed threshold in [43]. The authors in [43] proposed to used the threshold to flip a bit which is the maximum energy of the previous iteration. Despite the "untrue" maximum value threshold used, DDS-PGDBF surpasses the soft decision decoders in performance and approached the Maximum Likelihood Decoding at the cost of a very large number of iterations. The VNSA-IM-PGDBF similarly in some case does not use the true maximum energy value as the threshold since the MI sorts only a part of N energy values. The VNSA-IM-PGDBF coherently provides the better decoding performance in some testing LDPC codes.



(a) (b)

Figure 5.18: Figure 5.18(a): the trivial VNU type (type 3 VNU) proposed for VNSA-IM-PGDBF. Figure 5.18(b): the Maximum Indicator in VNSA-IM-PGDBF.

5.7 The synthesis results and decoding performance

5.7.1 Synthesis results

In this section, we report the ASIC results at post-synthesis level, of the proposed PGDBF implementations. The synthesis has been done targeting a 65nm CMOS technology, using Synopsys tools.

For the first synthesis comparison, our goal is to demonstrate the area gains that one can achieve using the VNSA approach. The results are reported in Table 5.1 for a QC-LDPC code, with parameters $(d_v, d_c) = (3, 6)$, $Z = 54$, $M = 648$, $N = 1296$. In this Table, we have constrained the implementations to run at the same clock frequency, by setting the timing constraint identical for all decoders, fixed to 8 ns. We choose this strategy to measure precisely the impact of VNSA on the hardware cost, even if the working frequency is not maximized. We indicate in brackets the additional cost in percentage compared to the deterministic GDBF implementation. As a first remark, we can see that the VNSA is an alternative implementation solution as no extra complexity required to implement VNSA-based GDBF decoder compared to the conventional GDBF implementation. The second remark is that VNSA in general, reduces the decoder complexity as expected. The VNSA-PGDBF requires less than the conventional GDBF implementation, around 5% while the conventional PGDBF implementation needs 4% extra cost. As also expected, the VNSA-IM-PGDBF largely reduces the complexity with 15% lower than the GDBF decoder. We have verified that similar conclusions could be made for LDPC codes with different parameters, with various lengths, rates and values of d_v.

In the second synthesis comparison, we report working frequency and throughput in Table 5.2. Table 5.2 shows the results for the GDBF, PGDBF and two MS decoders taken from the literature [24, 25]. The code used for the GDBF, PGDBF and [24] is the $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$, $Z = 54$ LDPC code, while [25] considers the IEEE 802.11n standard codes with various lengths and rates.

dv3R050N1296 - AREA (μm^2)			
	Conventional implementation	**VNSA-based**	**Imprecise VNSA-based**
GDBF	53692 (+0%)	53692 (+0%)	N/A
PGDBF	55837 (+4.00%)	51091 (-4.8%)	45487 (-15.3%)

Table 5.1: Comparison on hardware resource used to implement the GDBF and PGDBF decoders by using the conventional and the VNSA architectures. The percentages in brackets indicate the additional/saving hardware compared to the GDBF.

For the GDBF and the PGDBF decoders, we performed the synthesis with the objective of optimizing the timing constraint, which results in the maximum frequency at which the decoder can operate.

It can be seen that the conclusions drawn from the Table 5.1 are maintained in which the VNSA reduces the complexity in PGDBF implementation even compared to the GDBF. The reduction is more significant in the case of VNSA-IM-PGDBF in Table 5.2 with around 22% complexity reduced. the hardware cost of the MS decoders is a lot larger than the BF-based decoders, and requires 8.5 to 15 times more area than the VNSA-based PGDBF. Our implementation of BF decoders allows to perform one iteration in $N_c = 1$ clock cycle, which results in a very important throughput gain of GDBF and PGDBF decoders over MS. The average throughput is compared with 2 settings. At the same channel noise level, *i.e.* $\alpha = 0.01$, the VNSA-PGDBF is around 1.4 times faster than MS decoder, but in 1.6 times slower than the GDBF with 2 decades gain in decoding performance. In order to maintain a target performance, *i.e.* $FER = 1e^{-5}$, the VNSA-PGDBF is 2 time faster than the MS decoder and 2.5 times slower than GDBF. The MS decoder is, of course, the best in error correction capability.

	Code length	Code rate	AREA (μm^2)	kGE	f_{max} (MHz)	N_c	$FER = 1e-5$		$\alpha = 0.01$	
							It_{ave}	θ (Gbit/s)	It_{ave}	θ (Gbit/s)
GDBF	1296	1/2	87810 (+0.00%)	75	222	1	2.00 ($@\alpha = 0.005$)	144.00	2.95 ($FER = 3e^{-4}$)	97.63
LFSR-PGDBF(S = Z = 54), $p_0 = 0.7$	1296	1/2	90589 (+3.3%)	77	232	1	4.83 ($@\alpha \approx 0.01$)	62.3	4.83 ($FER = 8e^{-6}$)	62.3
VNSA-PGDBF, $p_0 = 0.7$	1296	1/2	84045 (-4.2%)	72	222	1	4.83 ($@\alpha \approx 0.01$)	59.6	4.83 ($FER = 8e^{-6}$)	59.6
VNSA-IM-PGDBF, $p_0 = 0.7$	1296	1/2	67917 (-22.7%)	58	222	1	6.30 ($@\alpha \approx 0.013$)	45.7	5.32 ($FER = 2.6e^{-6}$)	54.1
MS [24]	1296	1/2	720000	615	250	6	2.34 ($@\alpha = 0.025$)	23.08	1.29 ($FER = 1e^{-7}$)	41.86
MS [25]	648 - 1944	1/2 - 5/6	1023000	-	400	-	108 - 337 (Mbps) at $It_{max} = 20 - 25$ iterations 1.39 - 4.34 (Gbps) at $It_{ave} = 1.94$			

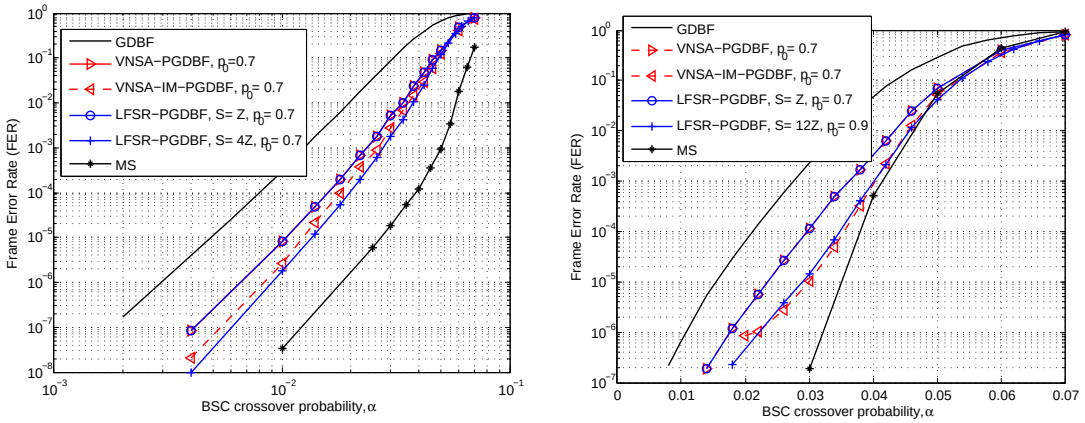
Table 5.2: Frequency and throughput comparison between GDBF decoder, PGDBF decoders, and MS decoders [24, 25].

5.7.2 Decoding performance

In this section, we illustrate the advantage of the implemented PGDBF decoders in decoding performance in comparison with GDBF decoders on the BSC channel. The PGDBF decoders are also put in comparison with the quantized MS decoder.

The compared MS decoder is a layered version with 6 quantization bits for the APP-LLR and 4 quantization bits for the extrinsic messages, with a maximum of $It_{max} = 20$ decoding iterations. We consider firstly two regular LDPC codes for the simulations: a QC-LDPC code with parameters $d_v = 3$, $d_c = 6$, rate 0.5, $Z = 54$, $M = 648$, $N = 1296$ (dv3R050N1296), and a QC-LDPC code with $d_v = 4$, $d_c = 8$, rate 0.5, $M = 648$, $N = 1296$ (dv4R050N1296). For the GDBF and the PGDBF, the maximum number of iterations is set to $It_{max} = 300$.

Figure 5.19 shows the simulation performance of VNSA-PGDBF and VNSA-IM-PGDBF in comparison with GDBF and MS decoders. It can be seen that the VNSA-based PGDBF decoders are half way to the MS performance compared to GDBF. VNSA-IM-PGDBF is considerable better than the VNSA-PGDBF for both tested LDPC codes. Especially, the VNSA-IM-PGDBF is almost equal to the theoretical PGDBF performance (represented by the LFSR-PGDBF with optimal $S = 4Z$ for dv3R050N1296 and $S = 12Z$ for dv4R050N1296, shown in previous chapter). For the dv3R050N1296 code, the FER of VNSA-IM-PGDBF maintains closely to theoretical PGDBF decoder to very low error rate ($1e^{-8}$) while for the dv4R050N1296, the error floor appears from the $FER = 1e^{-6}$. The MS decoder is the best in error correction at the cost of complexity and throughput as shown in previous section.

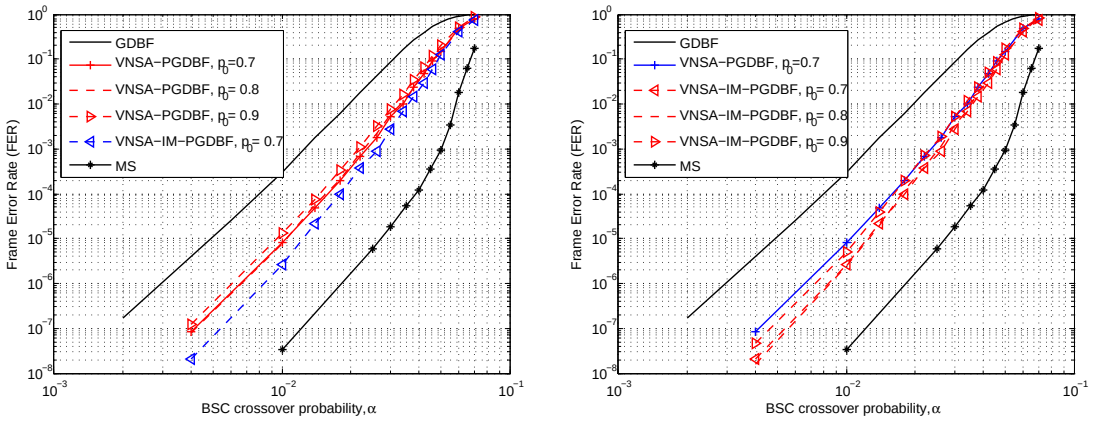


(a) (b)

Figure 5.19: The decoding performance of the VNSA-PGDBF and VNSA-IM-PGDBF on different LDPC code. Figure 5.19(a) for the $((d_v, d_c) = (3, 6)$, $Z = 54$, $N = 1296$ and $M = 648)$ LDPC code. Figure 5.19(b) for the $((d_v, d_c) = (4, 8)$, $Z = 54$, $N = 1296$ and $M = 648)$ LDPC code.

Figure 5.20 shows the decoding performance of VNSA-based PGDBF decoders on dv3R050N1296 when the value of p_0 is varied. It reconfirms that $p_0 = 0.7$ is the optimal values in term of decoding performance. Changing the value of p_0 slightly degrade the error correction ability as seen in the figure.

Finally, we present in Figure 5.21 the decoding performance of the implemented decoders on another long code in the storage applications, $((d_v, d_c) = (4, 34)$, $Z = 140$, $N = 9520$ and $M = 1120)$ LDPC code. Coherently with the above remarks,



(a) (b)

Figure 5.20: The decoding performance of the VNSA-PGDBF and VNSA-IM-PGDBF with the variation of p_0 on the $((d_v, d_c) = (3, 6)$, $Z = 54$, $N = 1296$ and $M = 648)$ LDPC code.

VNSA-based PGDBF provides a very good decoding gain especially the VNSA-PGDBF which is close to the theoretical PGDBF (presented here as LFSR-PGDBF, $S = M = 8Z$). The only remark is that the VNSA-IM-PGDBF starts the error floor very early ($FER = 1e^{-3}$) which makes VNSA-IM-PGDBF being worse than VNSA-PGDBF, but the gain over GDBF is maintained.

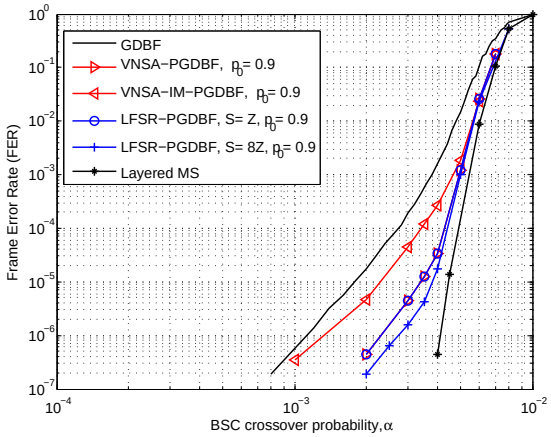


Figure 5.21: The decoding performance comparison on the $((d_v, d_c) = (4, 34)$, $Z = 140$, $N = 9520$ and $M = 1120)$ LDPC code.

5.8 Conclusion

This chapter describes a new and original architecture for QC-LDPC decoders named as the Variable-Node Shift Architecture (VNSA). The VNSA deploys the homogeneous construction property of the QC-LDPC codes to shift the memory of the decoders while preserving the decoding operations properly as the conventional implementation architecture. VNSA is shown to be applicable in different types of

LDPC decoders such as edge-type memory decoders, *i.e.* MS and its variants, and node-type memory decoders *i.e.* GDBF, PGDBF. The efficiency of the proposed VNSA, in term of complexity as well as in decoding performance, is confirmed by the example of PGDBF implementation.

Chapter 6

Conclusion and perspectives

In this thesis, we investigate on the analysis and implementation of the recently proposed Probabilistic Gradient Descent Bit Flipping decoding algorithm. We provide a method to analyse PGDBF and show the principle that random perturbation helps improving decoding performance in Chapter 3. With proposed analysis method, we can formulate the Frame Error Rate of PGDBF as a function of iterations and evaluate the asymptotic decoding performance of PGDBF decoder (infinite number of iteration). In chapter 4 we propose an efficient hardware implementation of PGDBF in which a simplification of the probabilistic signal generator is proposed and an optimization on the Maximum Indicator is presented. The implementation synthesis results reveal that PGDBF is a very low-complexity, high decoding throughput decoder while offering a powerful decoding performance to be competitive to soft decision decoders. More specially, in chapter 5 we propose a novel hardware architecture for Quasi-Cyclic LDPC decoding algorithms called Variable-Node Shift Architecture. VNSA is shown to be able to apply on different types of LDPC decoding algorithms in which the advantages can be expressed as the hardware complexity reduction and/or decoding performance. By implementing PGDBF with VNSA, it is shown that the decoder complexity is even smaller than the deterministic GDBF while preserving the decoding performance as good as the theoretical PGDBF.

In the remainder of the chapter, we describe some interesting open problems as future research directions that we identify for each of the topics that are investigated in this thesis.

Chapter 3: Theoretical analysis of Probabilistic Gradient Descent Bit Flipping

The FST method is proposed for analyzing the hard decision decoders and is illustrated in this thesis by analyzing the PGDBF decoder. FST is more meaningful for analyzing the decoder with "probabilistic" such as the decoder with faulty hardware or decoder with deliberate randomness... since the analysis results are drawn by

considering all possible state of decoders. The FST can be extended to analyse different types of LDPC decoding such as the soft decision decoders or LDPC decoders with memory, *i.e.* DD-BMP, IDP... and provides more practical results. This comes from the fact that FST analyses on the subgraphs of the practical LDPC code with no "impractical" requirements such as infinitive code length or number of iterations as in Density Evolution method. However, several issues need to be considered such as the isolation issue, the size state space... FST analyses the decoding behavior on with the error bits located on the support of a subgraph which is assumed to be isolated. This may lead to the difference results when running on the whole codeword. Furthermore, the size of state space when applying FST on some decoding algorithm such as soft decision decoders may be very large leading to the difficulty on analyzing.

Chapter 4: Efficient hardware implementation of Probabilistic Gradient Descent Bit Flipping

The statistical analysis and implementation of PGDBF decoder are restricted in this thesis only on BSC channel and on regular LDPC code in which PGDBF is shown to be very efficient in complexity and throughput and the FER approaches MS decoder. It would be useful to explore the ability of PGDBF on AWGN in term of error correction performance and its complexity. The conclusions on property of PGDBF on AWGN will give further information on the PGDBF real applications.

Several standards adopt irregular LDPC codes which outperform the regular architectures. Another topic which could be interesting to study is the modification of PGDBF to work on irregular LDPC code. Due to the fact that the VNs have different degrees leading to different achievable maximum energies, a new flip bit selection mechanism need to be proposed to cope with this different maximum energy level.

Chapter 5: A Quasi-Cyclic friendly architecture for LDPC decoders : the Variable-Node Shift Architecture

The VNSA architecture is proposed for the QC-LDPC decoding which is shown to be applicable for different types of decoding algorithms. It would be beneficial to deeper explore the VNSA ability in improving decoding performance and/or reducing decoder complexity. Indeed, the VNSA-PGDBF is an example of using 2 VN functions with arbitrary distribution in the sequence of implemented VNUs. In different decoders (such as FAID...), the VN functions may be optimized by different methods providing better error correction. Also, the VNU distribution may follow some dedicated rules such that different "uncorrectable" error patterns become correctable. For better performance, there may have more than 2 functions implemented with some optimal distributions such that the uncorrectable error patterns is minimized. Although the decoder global complexity depends on different parameters such as the chosen functions, the hardware reusing ratio..., it is expected that the VNSA-based decoder complexity is not increased or even reduced compared to the conventional implementation. The conventional implementation in which the

decoder is equipped the ability to process a VN with different functions in different iteration, requires all functions implemented in each VNU and a mechanism to choose function. This leads to an inevitable increment in hardware complexity as analysed in the thesis.

Appendix A

A.1 Some LDPC codes used in the thesis

A.1.1 The Tanner QC-LDPC code $(d_v, d_c)= (3, 6)$, $R = 0.4$, $M = 93$, $N = 155$ and $Z = 31$

$$H_B = \begin{pmatrix} 1 & 2 & 4 & 8 & 16 \\ 5 & 10 & 20 & 9 & 18 \\ 25 & 19 & 7 & 14 & 28 \end{pmatrix}$$

A.1.2 The QC-LDPC code $(d_v, d_c)= (3, 6)$, $R = 0.5$, $M = 648$, $N = 1296$ and $Z = 54$

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

A.1.3 The QC-LDPC code $(d_v, d_c)= (4, 8)$, $R = 0.5$, $M = 648$, $N = 1296$ and $Z = 54$

$$H_B = \begin{pmatrix}
11 & -1 & -1 & -1 & 27 & -1 & -1 & -1 & 33 & 16 & -1 & -1 & -1 & 44 & -1 & -1 & 44 & -1 & 8 & -1 & -1 & -1 & -1 & 0 \\
-1 & 25 & -1 & -1 & -1 & 31 & 29 & -1 & -1 & -1 & 29 & -1 & -1 & -1 & 36 & -1 & -1 & 34 & -1 & 15 & -1 & -1 & 17 & -1 \\
-1 & -1 & 44 & 4 & -1 & -1 & -1 & 11 & -1 & -1 & -1 & 2 & 50 & -1 & -1 & 52 & -1 & -1 & -1 & -1 & 30 & 33 & -1 & -1 \\
27 & -1 & -1 & -1 & 34 & -1 & 20 & -1 & -1 & 20 & -1 & -1 & -1 & 13 & -1 & -1 & 27 & -1 & 4 & -1 & -1 & -1 & -1 & 27 \\
-1 & 42 & -1 & 22 & -1 & -1 & -1 & 11 & -1 & -1 & -1 & 44 & -1 & -1 & 4 & 14 & -1 & -1 & -1 & -1 & 45 & 17 & -1 & -1 \\
-1 & -1 & 24 & -1 & -1 & 10 & -1 & -1 & 10 & -1 & 18 & -1 & 2 & -1 & -1 & -1 & -1 & 19 & -1 & 38 & -1 & -1 & 31 & -1 \\
-1 & -1 & 40 & -1 & -1 & 35 & -1 & -1 & 31 & 19 & -1 & -1 & 3 & -1 & -1 & 42 & -1 & -1 & -1 & 42 & -1 & -1 & 39 & -1 \\
-1 & 29 & -1 & 0 & -1 & -1 & -1 & 29 & -1 & -1 & 5 & -1 & -1 & -1 & 47 & -1 & -1 & 28 & -1 & -1 & 28 & 41 & -1 & -1 \\
9 & -1 & -1 & -1 & 7 & -1 & 20 & -1 & -1 & -1 & -1 & 1 & -1 & 19 & -1 & -1 & 5 & -1 & 25 & -1 & -1 & -1 & -1 & 41 \\
-1 & -1 & 53 & -1 & -1 & 3 & -1 & -1 & 26 & -1 & 3 & -1 & -1 & -1 & 30 & -1 & -1 & 5 & -1 & 35 & -1 & -1 & 44 & -1 \\
-1 & 4 & -1 & -1 & 4 & -1 & -1 & 5 & -1 & -1 & -1 & 13 & 42 & -1 & -1 & 50 & -1 & -1 & -1 & -1 & 36 & 38 & -1 & -1 \\
39 & -1 & -1 & 17 & -1 & -1 & 36 & -1 & -1 & 34 & -1 & -1 & -1 & 46 & -1 & -1 & 12 & -1 & 8 & -1 & -1 & -1 & -1 & 15
\end{pmatrix}$$

A.2 Min Sum decoding algorithms in flooding and layered scheduling

A.2.1 Flooding Min Sum decoding algorithm

The flooding MS decoding algorithm is described in Algorithm 3. We denote all messages produced by VNs which are sent to the input of the CNs as α. We also denote all messages produced by CNs which are sent to the input of VNs as β. We group all values at the inputs and outputs of VNs and CNs as the vector of values as following. The $VN_{i,j}$ process on the input vector $\underline{\boldsymbol{\beta}}_{new}^{i,j}$ and the produced results are in the vector $\underline{\boldsymbol{\alpha}}_{i,j}$. The $CN_{a,b}$ process on the input vector $\underline{\boldsymbol{\alpha}}_{new}^{a,b}$ and the produced results are in the vector $\underline{\boldsymbol{\beta}}_{a,b}$. 2 interleaving steps are proceeded during decoding iterations to produce $\underline{\boldsymbol{\beta}}_{a,b}$ at CN output to $\underline{\boldsymbol{\beta}}_{new}^{i,j}$ at the VN input and $\underline{\boldsymbol{\alpha}}_{i,j}$ at the VN output to $\underline{\boldsymbol{\alpha}}_{new}^{a,b}$ at the CN input. At the initialization steps, all the output vector of VNs ,$\underline{\boldsymbol{\alpha}}_{i,j}, \forall 1 \leq i \leq n_c, 1 \leq j \leq Z$, are set by the priori information received from the channel. The flooding MS operates as following.

- 1. VN to CN interleaving: for any $CN_{a,b}, 1 \leq a \leq n_r, 1 \leq b \leq Z$, $\underline{\boldsymbol{\alpha}}_{new}^{a,b}$ are formed from $\underline{\boldsymbol{\alpha}}_{i,j}$
- 2. CN computation: For any $CN_{a,b}$, the extrinsic messages are computed by the formed $\underline{\boldsymbol{\alpha}}_{new}^{a,b}$.
- 3. CN to VN interleaving: for any $VN_{i,j}, 1 \leq i \leq n_c, 1 \leq j \leq Z$, $\underline{\boldsymbol{\beta}}_{new}^{i,j}$ are formed from $\underline{\boldsymbol{\beta}}_{a,b}$
- 4. VN computation: For any $VN_{i,j}$, the A Posteriori information, $\tilde{\gamma}_{i,j}$, are computed first by summing all the value in $\underline{\boldsymbol{\beta}}_{new}^{i,j}$ and $\gamma_{i,j}$. The extrinsic messages are computed by eliminating the corresponding value β message in $\tilde{\gamma}_{i,j}$ forming $\underline{\boldsymbol{\alpha}}_{i,j}$.

- 5. The hard decision vector $\underline{\boldsymbol{v}}$ is formed by the signs of all $\tilde{\gamma}_{i,j}$.

Algorithm 3 The flooding Min-Sum (MS) decoding algorithm

Input: $\mathbf{y} = (y_{1,1}, y_{1,2}, \ldots, y_{n_c,Z-1}, y_{n_c,Z})$, $1 \le i \le n_c, 1 \le j \le Z$ ▷ received word
Output: $\underline{\boldsymbol{v}} = (v_{1,1}, v_{1,2}, \ldots, v_{n_c,Z-1}, v_{n_c,Z}) \in \{0,1\}^N$ ▷ estimated codeword
Initialization
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do** $\gamma_{i,j} = \mathbf{q}(y_{i,j})$; ▷ quantization process
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do**
    for all $n = 1, \ldots, d_v$ **do** $\alpha_n = \gamma_{i,j}$; ▷ n: local indexing
    $\underline{\boldsymbol{\alpha}}_{i,j} = \{\alpha_n\}_{n=1\ldots d_v}$;
Iteration Loop
  for all $a = 1, \ldots, n_r$ and $b = 1, \ldots, Z$ **do** ▷ interleaving
    $\underline{\boldsymbol{\alpha}}_{new}^{a,b} = \{\alpha_m\}_{m=1\ldots d_c}$:$\alpha_m \in \underline{\boldsymbol{\alpha}}_{i,j}$ if $H(a*Z+b, i*Z+j) = 1$;
    ▷ **CN-processing**
    for all $m = 1, \ldots, d_c$ **do**
      $\beta_m = \prod_{\alpha_{m'} \in \underline{\boldsymbol{\alpha}}_{new}^{a,b} \setminus \alpha_m} \operatorname{sgn}(\alpha_{m'}) * \min_{\alpha_{m'} \in \underline{\boldsymbol{\alpha}}_{new}^{a,b} \setminus \alpha_m} |\alpha_{m'}|$;
    $\underline{\boldsymbol{\beta}}_{a,b} = \{\beta_m\}_{m=1\ldots d_c}$
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do** ▷ interleaving
    $\underline{\boldsymbol{\beta}}_{new}^{i,j} = \{\beta_n\}_{n=1\ldots d_v}$:$\beta_n \in \underline{\boldsymbol{\beta}}_{a,b}$ if $H(a*Z+b, i*Z+j) = 1$;
    ▷ **VN-processing**
    $\tilde{\gamma}_{i,j} = \gamma_{i,j} + \sum_{\beta_n \in \underline{\boldsymbol{\beta}}_{new}^{i,j}} \beta_n$;
    for all $n = 1, \ldots, d_v$ and $\beta_n \in \underline{\boldsymbol{\beta}}_{new}^{i,j}$ **do**
      $\alpha_n = \tilde{\gamma}_{i,j} - \beta_n$
    $\underline{\boldsymbol{\alpha}}_{i,j} = \{\alpha_n\}_{n=1\ldots d_v}$
  $\underline{\boldsymbol{v}} = \{v_{i,j}\}$: $v_{i,j} = \operatorname{sgn}(\tilde{\gamma}_{i,j})$; ▷ hard decision
  if $\underline{\boldsymbol{v}}$ is a codeword **then** exit the iteration loop ▷ syndrome check
End Iteration Loop

A.2.2 Layered Min Sum decoding algorithm

We denote l ($l = 1, \ldots, L$) as layer index. The set of VNs belonging to layer l-th is denoted as $\mathcal{M}_{VN}(l)$ and the set of CNs belonging to layer l-th is denoted as $\mathcal{M}_{CN}(l)$. In the initialization step, the $\tilde{\gamma}_{i,j}$ is set by the received information from channel, *i.e.* $\tilde{\gamma}_{i,j} = \gamma_{i,j}$ and the message memory is reset, *i.e.* $\underline{\boldsymbol{\beta}}_{i,j}(n) = 0, \forall n = 1; \ldots, d_v, |\underline{\boldsymbol{\beta}}_{i,j}(.)| = d_v$. We introduce a variable $t_{i,j}$ as a counter for each $VN_{i,j}$. The layered MS decoder at layer l-th operates as following and an iteration includes L layers computations.

- 1. VN computation: for any $VN_{i,j} \in \mathcal{M}_{VN}(l)$ compute the messages $\alpha_{i,j}$. Note that, at 1 layer, each VN produces only 1 value of $\alpha_{i,j}$.

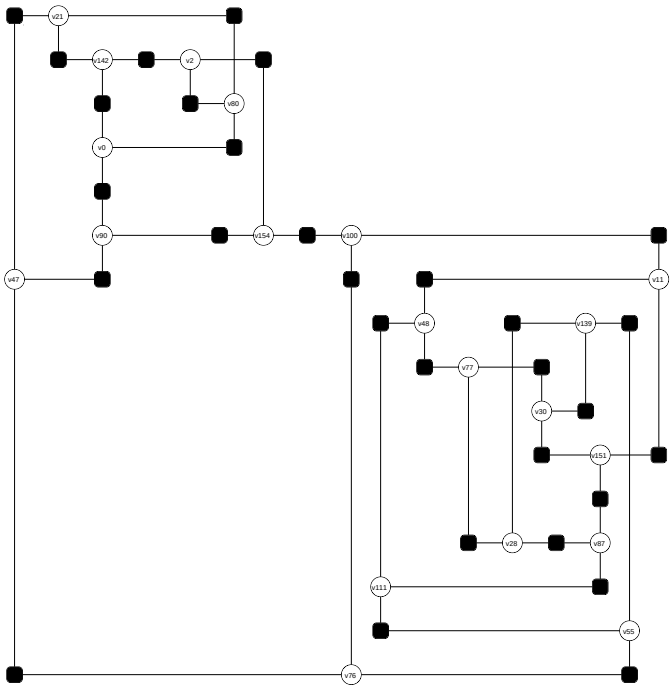
- 2. VN to CN interleaving: for any $CN_{a,b} \in \mathcal{M}_{CN}(l)$, the input vector, $\underline{\boldsymbol{\alpha}}_{new}^{a,b}$, is formed from all values of $\alpha_{i,j}$ by the interleaver.
- 3. CN computation: for any $CN_{a,b} \in \mathcal{M}_{CN}(l)$, the extrinsic vector $\underline{\boldsymbol{\beta}}_{a,b}$ is computed
- 4. VN to CN interleaving: for any $VN_{i,j} \in \mathcal{M}_{VN}(l)$, the value $\beta_{new}^{i,j}$ is identified.
- 5. APP and memory update: for any $VN_{i,j} \in \mathcal{M}_{VN}(l)$, the value $\tilde{\gamma}_{i,j}$ is updated and $\beta_{new}^{i,j}$ is stored into $\underline{\boldsymbol{\beta}}_{i,j}(.)$ in the elements $t_{i,j}$.
- 6. The hard decision vector $\underline{\boldsymbol{v}}$ is formed by the signs of all $\tilde{\gamma}_{i,j}$.

Algorithm 4 Min-Sum (MS) Decoding with Layered Scheduling

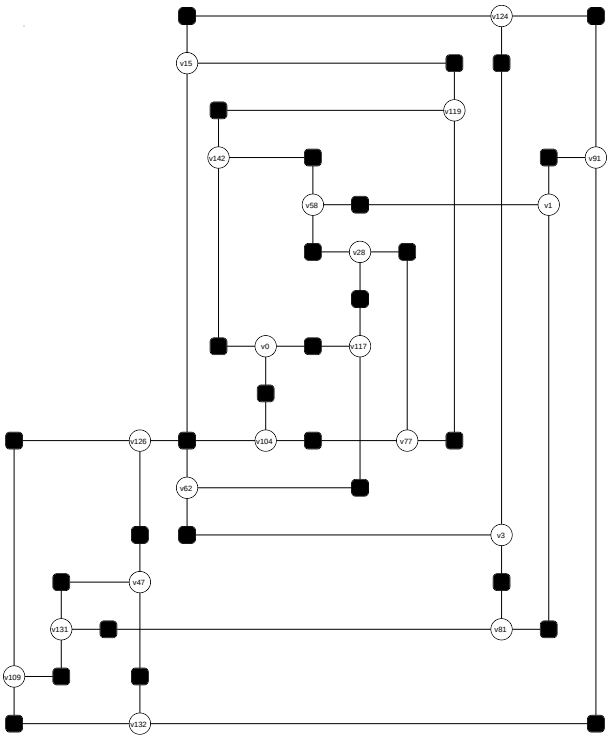
Input: $\mathbf{y} = (y_{1,1}, y_{1,2}, \ldots, y_{n_c,Z-1}, y_{n_c,Z})$, $1 \le i \le n_c, 1 \le j \le Z$ ▷ received word
Output: $\underline{\boldsymbol{v}} = (v_{1,1}, v_{1,2}, \ldots, v_{n_c,Z-1}, v_{n_c,Z}) \in \{0,1\}^N$ ▷ estimated codeword
Initialization
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do** $\gamma_{i,j} = \mathbf{q}(y_{i,j})$; ▷ quantization process
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do** $\tilde{\gamma}_{i,j} = \gamma_{i,j}$;
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do**
    for all $n = 1, \ldots, d_v$ **do** $\underline{\boldsymbol{\beta}}_{i,j}(n) = 0$; ▷ n: local indexing
  for all $i = 1, \ldots, n_c$ and $j = 1, \ldots, Z$ **do** $t_{i,j} = 1$; ▷ memory counter
Iteration Loop
  for all $l = 1, \ldots, L$ **do** ▷ **Layer (check-group) loop**
    for all $VN_{i,j} \in \mathcal{M}_{VN}(l)$ **do** ▷ **VN-processing**
      $\alpha_{i,j} = \tilde{\gamma}_{i,j} - \underline{\boldsymbol{\beta}}_{i,j}(t_{i,j})$;
    for all $CN_{a,b} \in \mathcal{M}_{CN}(l)$ **do** ▷ interleaving
      for all $n = 1, \ldots, d_c$ **do**
        for all $VN_{i,j} \in \mathcal{M}_{VN}(l)$ **do**
          $\alpha_n = \alpha_{i,j}$ if $h(a * Z + b, i * Z + j) = 1$;
        $\underline{\boldsymbol{\alpha}}^{a,b}_{new} = \{\alpha_m\}_{n=1\ldots d_c}$;
    for all $CN_{a,b} \in \mathcal{M}_{CN}(l)$ **do** ▷ **CN-processing**
      for all $n = 1, \ldots, d_c$ **do**
        $\beta_n = \prod_{\alpha_{n'} \in \underline{\boldsymbol{\alpha}}^{a,b}_{new} \setminus \alpha_n} \operatorname{sgn}(\alpha_{n'}) * \min_{\alpha_{n'} \in \underline{\boldsymbol{\alpha}}^{a,b}_{new} \setminus \alpha_n} |\alpha_{n'}|$;
      $\underline{\boldsymbol{\beta}}_{a,b} = \{\beta_n\}_{n=1\ldots d_c}$;
    for all $CN_{a,b} \in \mathcal{M}_{CN}(l)$ **do** ▷ interleaving
      for all $n = 1, \ldots, d_c$ **do**
        for all $VN_{i,j} \in \mathcal{M}_{VN}(l)$ **do**
          $\beta^{i,j}_{new} = \underline{\boldsymbol{\beta}}_{a,b}(n)$ if $h(a * Z + b, i * Z + j) = 1$;
    for all $VN_{i,j} \in \mathcal{M}_{VN}(l)$ **do** ▷ **AP and memory update**
      $\tilde{\gamma}_{i,j} = \alpha_{i,j} + \beta^{i,j}_{new}$;
      $\underline{\boldsymbol{\beta}}_{i,j}(t_{i,j}) = \beta^{i,j}_{new}$;
      $t_{i,j} = t_{i,j} + 1$;
  end (layer loop)
  $\underline{\boldsymbol{v}} = \{v_{i,j}\}$: $v_{i,j} = \operatorname{sgn}(\tilde{\gamma}_{i,j})$; ▷ hard decision
  if $\underline{\boldsymbol{v}}$ is a codeword **then** exit the iteration loop ▷ syndrome check
End Iteration Loop

A.3 3 weight-20 codewords in Tanner code

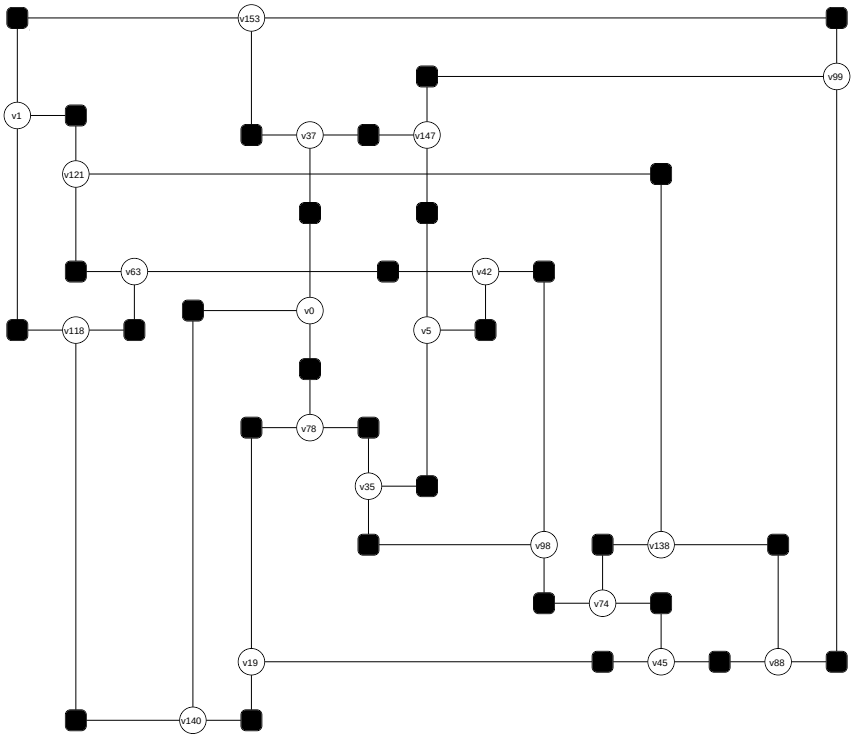
A.3.1 Type I



A.3.2 Type II



A.3.3 Type III


v153
v99
v1
v37
v147
v121
v63
v42
v0
v118
v5
v78
v35
v98
v138
v74
v19
v45
v88
v140

Bibliography

[1] R. G. Gallager, *Low density parity check codes.* MIT Press, Cambridge, 1963, Research Monograph series, 1963.

[2] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711–2736, Nov 2001.

[3] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Communications Letters*, vol. 8, no. 3, pp. 165–167, March 2004.

[4] N. Miladinovic and M. Fossorier, "A improved bit-flipping decoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005.

[5] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for ldpc codes," *IEEE Communications Letters*, vol. 9, no. 9, pp. 814–816, Sep 2005.

[6] X. Wu, C. Zhao, and X. You, "Parallel weighted bit-flipping decoding," *IEEE Communications Letters*, vol. 11, no. 8, pp. 671–673, August 2007.

[7] G. Li and G. Feng, "Improved parallel weighted bit-flipping decoding algorithm for ldpc codes," *IET Communications*, vol. 3, no. 1, pp. 91–99, January 2009.

[8] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding ldpc codes," *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[9] R. Haga and S. Usami, "Multi-bit flip type gradient descent bit flipping decoding using no thresholds," in *2012 International Symposium on Information Theory and its Applications*, Oct 2012, pp. 6–10.

[10] T. Phromsa-ard, J. Arpornsiripat, J. Wetcharungsri, P. Sangwongngam, K. Sripimanwat, and P. Vanichchanunt, "Improved gradient descent bit flipping algorithms for ldpc decoding," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, May 2012, pp. 324–328.

[11] M. Ismail, I. Ahmed, , and J. Coon, "Low power decoding of ldpc codes," *ISRN Sensor Networks*, vol. 2013, no. 650740, 2013.

[12] Q. Zhu and L. n. Wu, "Weighted candidate bit based bit-flipping decoding algorithms for ldpc codes," in *2013 3rd International Conference on Consumer Electronics, Communications and Networks*, Nov 2013, pp. 731–734.

[13] D. V. Nguyen and B. Vasic, "Two-bit bit flipping algorithms for ldpc codes and collective error correction," *IEEE Transactions on Communications*, vol. 62, no. 4, pp. 1153–1163, April 2014.

[14] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for ldpc codes," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3385–3400, Oct 2014.

[15] O. A. Rasheed, P. Ivanis, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, Sept 2014.

[16] H. Huang, Y. Wang, and G. Wei, "Mixed modified weighted bit-flipping decoding of low-density parity-check codes," *IET Communications*, vol. 9, no. 2, pp. 283–290, 2015.

[17] Y. h. Liu, X. l. Niu, and M. l. Zhang, "Multi-threshold bit flipping algorithm for decoding structured ldpc codes," *IEEE Communications Letters*, vol. 19, no. 2, pp. 127–130, Feb 2015.

[18] T. C. Y. Chang and Y. T. Su, "Dynamic weighted bit-flipping decoding algorithms for ldpc codes," *IEEE Transactions on Communications*, vol. 63, no. 11, pp. 3950–3963, Nov 2015.

[19] S. Imani, R. Shahbazian, and S. A. Ghorashi, "An iterative bit flipping based decoding algorithm for ldpc codes," in *2015 Iran Workshop on Communication and Information Theory (IWCIT)*, May 2015, pp. 1–3.

[20] K. Ma, J. Jin, W. Li, and P. Zhang, "Two-staged weighted bit flipping (wbf) decoding algorithm for ldpc codes," in *2015 IEEE 9th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, Sept 2015, pp. 141–144.

[21] H. Li, H. Ding, and L. Zheng, "Hybrid iterative decoding for ldpc codes based on gradient descent bit-flipping algorithm," in *2016 8th International Conference on Wireless Communications Signal Processing (WCSP)*, Oct 2016, pp. 1–3.

[22] J. Jung and I. C. Park, "Multi-bit flipping decoding of ldpc codes for nand storage systems," *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–1, 2017.

[23] D. Declercq, B. Vasic, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders - part ii: Towards guaranteed error correction of ldpc codes via iterative decoder diversity," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4046–4057, October 2013.

[24] 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," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 230–237.

[25] 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 *2007 Design, Automation Test in Europe Conference Exhibition*, April 2007, pp. 1–6.

[26] 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*, vol. 17, no. 2, pp. 262–268, Feb 2009.

[27] D. Declercq, M. Fossorier, and E. Biglieri, *Channel Coding: Theory, Algorithms, And Applications.* Academic Press Library in Mobile and Wireless Communications, Elsevier, ISBN: 978-0-12-396499-1, 2014.

[28] Q. Huang, J. Kang, L. Zhang, S. Lin, and K. Abdel-Ghaffar, "Two reliability-based iterative majority-logic decoding algorithms for ldpc codes," *IEEE Transactions on Communications*, vol. 57, no. 12, pp. 3597–3606, December 2009.

[29] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasíc, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1494–1497.

[30] M. Tanner, D. Srkdhara, and T. Fuja, "A class of group-structured ldpc codes," in *Proc. 5th Int. Symp. Commun. Theory App.*, July 2001.

[31] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb 2001.

[32] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary message-passing ldpc decoder," *IEEE Transactions on Communications*, vol. 57, no. 9, pp. 2518–2523, September 2009.

[33] 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*, vol. 62, no. 3, pp. 619–631, Feb 2014.

[34] A. V. Aho, H. J. E., and U. J. D., *The Design and Analysis of Computer Algorithms*. Cambridge: Addison-Wesley, Reading, MA, 1974.

[35] B. VasiÄĞ, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sept 2009, pp. 1–7.

[36] D. Declercq, E. Li, B. Vasić, and S. K. Planjery, "Approaching maximum likelihood decoding of finite length ldpc codes via faid diversity," in *2012 IEEE Information Theory Workshop*, Sept 2012, pp. 487–491.

[37] D. Declercq, L. Danjean, E. Li, S. K. Planjery, and B. Vasić, "Finite alphabet iterative decoding (faid) of the (155,64,20) tanner code," in *2010 6th International Symposium on Turbo Codes Iterative Information Processing*, Sept 2010, pp. 11–15.

[38] B. Vasić, P. Ivanis, and D. Declercq, "Approaching maximum likelihood performance of ldpc codes by stochastic resonance in noisy iterative decoders," in *Information Theory and Applications Workshop*, Feb. 2016.

[39] T. Richardson, "Error floors of ldpc codes," in *41st Annual Allerton Conf on Communications Control and Computing*, Oct. 2003, pp. 1426–1435.

[40] A. K. Panda, P. Rajput, and B. Shukla, "Fpga implementation of 8, 16 and 32 bit lfsr with maximum length feedback polynomial using vhdl," in *2012 International Conference on Communication Systems and Network Technologies*, May 2012, pp. 769–773.

[41] B. Yuce, H. F. Ugurdag, S. Goren, and G. Dundar, "Fast and efficient circuit topologies forfinding the maximum of n k-bit numbers," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1868–1881, Aug 2014.

[42] 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," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.

[43] D. Declercq, C. Winstead, B. Vasic, F. Ghaffari, P. Ivanis, and E. Boutillon, "Noise-aided gradient descent bit-flipping decoders approaching maximum likelihood decoding," in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Sept 2016, pp. 300–304.