FP7-ICT / FET-OPEN 309129 / *i*-RISC

D3.3

Fault tolerant LDPC encoding and decoding

Editor:	E. Dupraz, ENSEA
Deliverable nature:	Public
Due date:	July 31, 2015
Delivery date:	August 31, 2015
Version:	2.0
Date of current version:	April 15, 2016
Total number of pages:	54 pages
Reviewed by:	i-RISC members
Keywords:	LDPC codes, FAID decoders, Gallager B decoder, bit-flipping decoders, fault-tolerant encoding and decoding, decoder design.

Abstract

This deliverable presents an overview of the activities carried out by the work package WP3 during the third year of the project. These activities include mainly the design and analysis of fault tolerant encoder and decoder architectures.

In order to protect the different parts of the chips from transient defects, the storage and computation units have to be redundant and incorporate a powerful error-correction technique. In this workpackage, we focus on the class of LDPC codes, decoded with iterative message passing decoders, and our goal is the design of LDPC codes with fault-tolerant encoder and decoder architectures. Our contributions during this third year of the project have been to design to analyze LDPC decoders on faulty hardware under more realistic error models and to design fault-tolerant LDPC encoders. Error correcting codes with fault tolerant encoder and decoder architectures constitute a building block of our approach to fault tolerant chip design.

List of Authors

Participant	Author
CEA	Valentin Savin (valetin.savin@cea.fr)
ENSEA	David Declercq (declercq@ensea.fr)
	Fakhreddine Ghaffari (fakhreddine.ghaffari@ensea.fr)
	Elsa Dupraz (elsa.dupraz@ensea.fr)
	Khoa Le (le.khoa@ensea.fr)
ELFAK	Bane Vasić (vasic@email.arizona.edu)
	Predrag Ivaniš (predrag.ivanis@etf.rs)
	Brkić Srdran (brka05@gmail.com)
	Omran Al Rasheed (omrano84@hotmail.com)
UCC	Emanuel Popovici (e.popovici@ucc.ie)
	Satish Kumar Grandhi (satishkumargrandhi@gmail.com)

Contents

List of Dissemination Activities

Published papers

[**P1**] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware", *To appear in IEEE Transactions on Communications*, 2015

[**P2**] S. Brkic, O. Al Rasheed, P. Ivanis, and B. Vasic, "On Fault-Tolerance of the Gallager B Decoder under Data-Dependent Gate Failures", *IEEE Communications Letters* , no.99, pp.1,1, June 2015

[**P3**] C. Kameni Ngassa, V. Savin, E. Dupraz, and D. Declercq, "Density Evolution and Functional Threshold for the Noisy Min-Sum Decoder", *IEEE Transactions on Communications*, vol.63, no.5, pp.1497,1509, May 2015

[**P4**] K. Le, D. Declercq, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasić, "Efficient Realization Of Probabilistic Gradient Descent Bit Flipping Decoders", *IEEE International Conference on Electronics Circuits and Systems (ISCAS 2015)*, Lisbon, Portugal, May 2015.

[**P5**] E. Dupraz and D. Declercq, "Evaluation of the Robustness of LDPC Encoders to Hardware Noise", *IEEE International BlackSea Conference on Communications and Networking*, Constanta, Romania, May 2015

Workshop presentations

[**P6**] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Finite-Alphabet Iterative Decoders Robust to Faulty Hardware", *Groupement de Recherche Information, Signal, Image et Vision (GdR-ISIS)*, Paris, November 2014.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

APP	*A Posteriori* Probability
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BF	Bit-Flipping
BP	Belief-Propagation
BSC	Binary Symmetric Channel
CN	Check Node
CNU	Check Node Update
CPE	Codeword Prediction Encoder
DE	Density Evolution
FAID	Finite Alphabet Iterative Decoder
FER	Frame Error Rate
GDBF	Gradient-Descent Bit-Flipping
IVRG	Intrinsic-Value Random Generator
LDPC	Low Density Parity Check
LDGM	Low Density Generator Matrix
LFSR	Linear Feedback Shift Register
LFSR-RG	Linear Feedback Shift Register Random Generator
LLR	Log-Likelihood Ratio
LS	Latin Squares
LUT	Look Up Table
MAJ	Majority-Logic
MS	Min-Sum
PEG	Progressive Edge Growth
PGDBF	Probabilistic Gradient-Descent Bit-Flipping
QC	Quasi-Cyclic
RG	Random Generator
SP	Sign-Preserving
VN	Variable Node
VNU	Variable Node Update
WBF	Weighted Bit-Flipping

Introduction

The i-RISC project addresses the problem of reliable computing with unreliable components, which is a crucial issue for the long-term development of computing technology. The novelty of the proposed research comes from the synergistic utilization of information theory and coding techniques, traditionally utilized to improve the reliability of communication systems, and circuit and system theory and design techniques in order to create reliable/predictable hardware.

Within i-RISC, the Work-Package 3 (Fault Tolerant Algorithms for Error-Correction) is aimed at understanding error-correcting codes in the context of unreliable computing systems. This is a new paradigm in coding theory, since faulty hardware can potentially induce errors during the encoding and the decoding process. It is then critical to properly evaluate the robustness of the existing encoders and decoders in the presence of an additional source of noise at the circuit level. The main goal of WP3 is to propose encoding and decoding algorithms that can effectively deal with the probabilistic behavior of the circuit.

We focus on the family of Low-Density Parity Check (LDPC) codes and several candidates for LDPC decoding will be considered during the project: Min-Sum (MS) based decoders, Finite-Alphabet Iterative Decoders (FAIDs), Stochastic decoders, Bit-Flipping (BF) decoders. Error correcting codes with fault tolerant decoder architectures constitute a building block of our approach to fault tolerant chip design. This building block will be used to address the problem of reliable memories and interconnections (WP4), and will be integrated into the fault-tolerant implementations of the logical functionality of the circuit (WP5).

An overview of the activities carried out during the third year of the project is presented in the next section.

Executive Summary

In the period Month 22 to Month 30 (M22-M30), Work Package 3 (WP3) activities addressed the achievement of the following objectives:

- Objective 3.1: Design of fault-tolerant LDPC encoders,
- Objective 3.2: Design of faulty-tolerant LDPC decoders under more realistic error models,
- Objective 3.3: Design of low-complexity fault-tolerant bit-flipping LDPC decoders.

In this line of reasoning, the main contributions of WP3 are aimed at analyzing and designing fault-tolerant LDPC encoders and decoders. Figure 1 presents the WP3 Gantt diagram. In indicates that the tasks addressed during the period M22-M30 are Task 3.1 – MS/FAID decoders under faulty gates, Task 3.4 – Practical fault tolerant encoding, and Task 3.5 – Randomized bit-flipping decoders for fault tolerance.

The main technical contributions related to these tasks and presented in this deliverable are summarized below, first for fault-tolerant LDPC decoders, and then for fault-tolerant LDPC encoders.

Concerning the fault tolerant LDPC decoders, during the first two years of the project, we only considered memoryless, data independent, symmetric error models, which have permitted to develop Density Evolution (DE) tools for the asymptotical analysis of faulty decoders. During the last year of the project, we incorporated more realistic noise models, and we investigated the performance of Gallager B and Finite Alphabet Iterative Decoders (FAIDs) under these models. The main contributions can be summarized as:

- **Faulty Gallager B Decoder under Timing Errors (Task 3.1)**. We characterized the performance of the Gallager B decoder under timing errors (Chapter 1). We showed that the timing error model makes the decoder dependent on a transmitted codeword, thus rendering inapplicable the traditional DE analysis. By using Monte Carlo simulations, we identified two operating regions - one in which hardware unreliability leads to significant performance degradation, and one in which the performance loss is negligible. Based on these results, we proposed a simple modification of the decoder that ensures its fault-tolerance under timing errors.
- **Faulty Finite Alphabet Iterative Decoders under Error Models with Memory (Task 3.2)**. We characterized the performance of FAIDs under error models with memory (Chapter 2). The memory makes it very difficult to perform a DE analysis of the decoders under these models. We reused the FAIDs we designed during the first two years of the project for robustness under the simple memoryless error models. Through Monte-Carlo simulations, we measured the performance of these FAIDs under error models with memory. We observed that the FAIDs designed for robustness in the memoryless case are still robust under error models with memory.
- **Hardware Efficient Implementation of Probabilistic Gradient Descent Bit-Flipping (Task 3.5)**. During the second year of the project, we proposed a new BF decoding algorithm called Probabilistic Gradient Descent Bit-Flipping (PGDBF) which introduces randomness in the bit-flipping rules. We observed that the PGDBF algorithm had better performance than standard BF algorithms. During the third year of the project, we introduced a new method

called Intrinsic-Valued Random Generator (IVRG) for the random generation of bits in the PGDBF. The introduced IVRG avoids generating the random bits required by the PGDBF algorithm but uses sequences of bits from the existing decoder memory. It provides a highly efficient hardware implementation compared to other conventional methods while preserving the outstanding decoding performance of PGDBF. We provided a statistical analysis of the behavior of the PGDBF and identified the important features of the random generators that permit to obtain large coding gains compared to the deterministic GDBF. The statistical analysis showed that the proposed implementation only induces a very small extra complexity, without any loss in performance for the PGDBF decoder.

Concerning the fault tolerant LDPC encoders, the main contribution can be summarized as:

- **Reliable LDPC Encoders built from Unreliable Gates (Task 3.4)**. During the second year of the project, we showed that most of the standard LDPC encoding techniques are non-robust to hardware noise. During the third year, we proposed an LDPC encoding technique robust to hardware errors (Chapter 4). The robust LDPC encoding technique we proposed is based on the Codeword Prediction Approach (CPE) introduced in WP5. It consists of computing an augmented codeword that contains both the codeword to be transmitted on the channel and extra parity bits. Before transmission on the channel, an LDPC decoder is applied to the augmented codeword in order to eliminate the hardware errors from the channel codeword. From Monte Carlo simulations, we showed the robustness of the proposed encoding solution for various codes and decoders.

Chapter 1

Faulty Gallager B Decoder under Timing Errors

Abstract: *In this chapter we characterize the effect of data-dependent gate failures on the performance of the Gallager B decoder of low density parity check codes. We show that this type of failures makes the decoder dependent on a transmitted codeword, thus rendering inapplicable the traditional analysis tools such as density evolution and trapping sets. By using Monte Carlo simulations, we identify two operating regions - one in which hardware unreliability leads to significant performance degradation, and one in which the performance loss is negligible. Based on these results, we propose a simple modification of the decoder that ensures its fault-tolerance.*

Work presented in this Chapter has been published in S. Brkic, O. Al Rasheed, P. Ivanis, and B. Vasic, "On Fault-Tolerance of the Gallager B Decoder under Data-Dependent Gate Failures", *IEEE Communications Letters*, no.99, pp.1,1, June 2015 [P2]

1.1 Introduction

Increasingly stringent requirements for semiconductor device energy-efficiency has lead to a point where computation performed by using these devices is no longer reliable [2]. The need to ensure fault-tolerance on inherently unreliable hardware has resulted in an increased interest in the analysis and design of novel and powerful error control schemes. The main direction of recent research in this area includes investigation of low-density parity-check (LDPC) codes and their decoders implemented on unreliable hardware, which relies on a theoretical framework developed by Vasic and Chilappagari [3]. The large body of knowledge on the analysis of codes on graphs and iterative decoding, has enabled further progress in the analysis of fault-tolerant schemes based on LDPC codes (see [4–8] and references therein).

The majority of the results on decoding by circuits made of unreliable hardware relies on modelling the logic gate unreliability as transient, independent failures - a model proposed by von Neumann [9] in the fifties. In the von Neumann failure model, in each clock cycle, components of a (clocked) Boolean network fail with some known probability. Additionally, failures of a given component are independent of those in previous clock cycles and independent of failures of other components. The simplicity of this model makes it amenable to theoretical analysis, but at same time limits its applicability. In new energy-efficient CMOS technologies, the logic gate failures are highly data-dependent and correlated in time [10]. The main source of incorrect gate outputs in these technologies is timing violations, or *timing errors*. They depend on logic gate inputs, but they are most damaging when the gate output changes its value [10, 11].

In order to characterize the hardware unreliability phenomenon accurately, Brkic *et al.* in [8] used a Markov chain timing error model, which has enabled them to analyze the behavior of one-step majority logic decoders. Timing errors in the context of the stochastic decoders have been considered

recently by Perez-Andrade *et al.* [12] who have shown an inherent tolerance to timing errors of these type of decoders. While the error injection at transistor level was thoroughly analyzed, the decoder data-dependence was not considered.

In this chapter we investigate the effects of timing errors to the performance of the Gallager B decoder, a simple hard decision decoder with a good trade off between the complexity and the error-correcting capability. Based on the Markov model proposed in [8], we evaluate the frame error rate (FER) for several classes of LDPC codes. We first demonstrate the inadequacy of von Neuman error model, and then show that the decoder performance is highly dependent on the sequence of codewords that is being decoded. Finally, we propose a simple modification of the decoder that increases the robustness to timing errors. Due to the lack of analytic techniques suitable to this failure model, we use Monte Carlo simulations.

1.2 Preliminaries

1.2.1 LDPC codes and Decoding Algorithm

Let (N, K) be an LDPC code, with code length N and code rate K/N, represented by a bipartite graph $G = (V \cup C, E)$, where V is the a of N variable nodes, C is a set of check nodes, and E is a set of edges. An edge is an unordered pair (v, c) which connects two nodes $v \in V$ and $c \in C$. The parity check matrix $\mathbf{H}$ is a bi-adjacency matrix of G. Nodes v and c are called *neighbors* iff $\mathbf{H}_{c,v} = 1$, i.e. there is an edge between them. Let $\mathcal{N}_v$ ($\mathcal{N}_c$) be a set of neighbors of a variable node v (check node c). In this chapter we consider (γ, ρ)-regular binary LDPC codes which means that $|\mathcal{N}_v| = \gamma$, $\forall v \in V$ and $|\mathcal{N}_c| = \rho$, $\forall c \in C$, where $|\cdot|$ denotes cardinality.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ denote a codeword of an LDPC code that is transmitted over a Binary Symmetric Channel (BSC) with crossover probability α, where $x_v \in \{\pm 1\}$ is the polar representation of a bit value associated with a variable node v and let a vector received by the Gallager B decoder from the BSC be $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$. Let $\nu_{v,c}^{(\ell)}$ ($\nu_{c,v}^{(\ell)}$) be messages passed on an edge (v, c) from(to) variable node to(from) check node during the ℓ-th decoding iteration, respectively. We next summarize the Gallager B decoder.

- *Variable to check node update*: For each variable node $v \in V$. At iteration $\ell = 0$: $\nu_{v,c}^{(0)} = y_v$, $\forall c \in \mathcal{N}_v$. At iteration $\ell > 0$:

$$\nu_{v,c}^{(\ell)} = \begin{cases} -y_v & \text{if } |\{c' \in \mathcal{N}_v \setminus c : \nu_{c',v}^{(\ell-1)} = -y_v\}| > \lfloor \gamma/2 \rfloor, \\ y_v & \text{otherwise.} \end{cases} \tag{1.1}$$

- *Check to variable node update.* For each check node $c \in C$ and $\forall v \in \mathcal{N}_c$, at iteration $\ell \geq 0$:

$$\nu_{c,v}^{(\ell)} = \prod_{v' \in \mathcal{N}_c \setminus \{v\}} \nu_{v',c}^{(\ell)}. \tag{1.2}$$

The decoding is terminated when all parity-check equations are satisfied or the maximum number of iterations is reached.

The decoder comprises of the *processing units* that correspond to the nodes in the bipartite graph representation of the decoder. Each check node (CN) unit is composed of ρ XOR gates, each with $\rho - 1$ inputs, needed for calculation of the check to variable node messages. A variable node (VN) processing unit calculates the variable to check node messages, and employs γ majority logic (MAJ) gates, each with $\gamma - 1$ inputs [3]. The decoder also requires additional logic gates for the final bit estimations and parity-checks calculation. If we allow these gates to be unreliable, the performance of the decoder would be determined by the failure probabilities of these gates, not by the error control scheme. Thus, it is reasonable to assume that these gates are perfect, and that only gates used to calculate messages that are passed on edges of the bipartite graph are faulty. Such reliable gates in

the final decision circuitry can be realized by using larger transistors, slowing down the clock or using higher voltage supply. Similar assumption was also used in other relevant literature [3, 8].

Note that we considered that the threshold in the variable-to-check-node update operations is fixed, opposed to the original solution proposed by Gallager. Although the threshold adaptation increases the performance, it requires the additional logic which makes it more complex than described algorithm. The use of a fixed threshold value represents a good trade-off between performance and efficiency, especially if the targeted error rates are low.

1.2.2 The Failure Model

The recent work in the area of low-powered combinatorial circuits has identified the increased signal propagation delay as the main cause of the circuits unreliability [11, 13]. Timing violations happen when the propagation delay is longer than what can be tolerated by the circuit design, which results in propagating an outdated output value to the rest of the circuit.

Let $f : \{\pm 1\}^m \to \{\pm 1\}$, $m > 1$, be an m-argument Boolean function, which at time instant ℓ produces the result $z^{(\ell)} = f(y_1^{(\ell)}, y_2^{(\ell)}, \ldots, y_m^{(\ell)})$, where $y_1^{(\ell)}, y_2^{(\ell)}, \ldots, y_m^{(\ell)}$ are input arguments at time ℓ. Due to unreliability of the logic gates used to calculate f, the result is not $z^{(\ell)}$ but $\mu^{(\ell)} = z^{(\ell)} e^{(\ell)}$, where $e^{(\ell)} \in \{\pm 1\}$ is the error at time ℓ. In the von Neumann failure model $e^{(\ell)}$ is a Bernoulli random variable, and does not depend on the gate input arguments [9].

In the timing failure model $e^{(\ell)}$ is data-dependent, and the probability that the logic gate fails to switch is $\Pr\{e^{(\ell)} = -1 | z^{(\ell)} \neq z^{(\ell-1)}\} = \varepsilon$, where $\varepsilon > 0$. On the other hand, when the gate output is unchanged during two consecutive time instants, the function f is always correctly computed as assumed in [10] and [11], i.e. $\Pr\{e^{(\ell)} = -1 | z^{(\ell)} = z^{(\ell-1)}\} = 0$. This corresponds to the gate switching probabilistic model [11], that was shown to have reduced complexity with minor degradation of accuracy when compared to more complex models that take into account that different input patterns can cause gate failures with different probabilities [10, 11].

The value ε depends on the technology parameters and can be obtained experimentally or by the circuit-level simulations. It typically differs from a gate type to a gate type. We denote by $\varepsilon_{\oplus}$ and ε_{MAJ} the failure rates of XOR and MAJ gates, respectively. The gate output at time instant ℓ, $\mu^{(\ell)}$, is obtained by the mapping $\Upsilon : \{\pm 1\}^3 \to \{\pm 1\}$ as follows

$$\mu^{(\ell)} = \Upsilon(z^{(\ell)}, z^{(\ell-1)}, e^{(\ell)}) = z^{(\ell)} (e^{(\ell)})^{(z^{(\ell)} - z^{(\ell-1)})/2}, \tag{1.3}$$

where $\Pr\{e^{(\ell)} = -1\} = \varepsilon$. Note that the failure model is being redefined to take advantage of the fact that $e^{(\ell)} = 1$ with probability 1, when $z^{(\ell)} = z^{(\ell-1)}$.

1.3 Failures of the Gallager B Decoding Algorithm

The gate failure model introduced in Section 1.2-B acts as a binary channel with memory. We next explain how that memory makes the performance of the faulty Gallager B decoder dependent on the transmitted codewords. Consider the received vector $\mathbf{y} = \mathbf{x} \cdot \mathbf{n}$, where "$\cdot$" denotes pointwise multiplication of the codeword vector $\mathbf{x}$ and the noise vector $\mathbf{n}$. Let $\nu_{v,c}^{(-1)}$ and $\nu_{c,v}^{(-1)}$ be output values of the update functions sent between variable node v and check node c in the time instant prior to the initial decoding iteration of a current codeword. These are the messages sent in the last iteration of decoding the previous received word. The following theorem defines the conditions under which the probability of successful decoding is independent of the transmitted codeword.

Theorem 1.1 *The frame error rate of the Gallager B decoder in the presence of timing errors is independent of the transmitted codeword* $\mathbf{x}$ *iff* $\nu_{v,c}^{(-1)} = x_v A_v$ *and* $\nu_{c,v}^{(-1)} = x_v B_v$, $\forall v \in V$ *and* $\forall c \in C$, *where* $A_v, B_v \in \{\pm 1\}$.

Proof: Let $\mu_{c,v}^{(\ell)}(\mathbf{y})$ and $\mu_{v,c}^{(\ell)}(\mathbf{y})$ be, respectively, the messages passed from a check node c to a variable node v and from a variable node v to a check node c, at iteration ℓ, given the received $\mathbf{y}$.

They are obtained from Eq. (1.3) based on the correctly computed messages $\nu_{c,v}^{(\ell)}(\mathbf{y})$ and $\nu_{v,c}^{(\ell)}(\mathbf{y})$ and the corresponding error values $e_{c,v}^{(\ell)}$ and $e_{v,c}^{(\ell)}$.

Let us assume $\nu_{v,c}^{(-1)} = x_v A_v$ and $\nu_{c,v}^{(-1)} = x_v B_v$, for some $A_v, B_v \in \{\pm 1\}$. We use mathematical induction to prove the theorem statement. From the variable node symmetry condition at iteration $\ell = 0$, we have $\nu_{v,c}^{(0)}(\mathbf{y}) = \nu_{v,c}^{(0)}(\mathbf{x} \cdot \mathbf{n}) = x_v \nu_{v,c}^{(0)}(\mathbf{n})$ [4]. As these values are received from the channel (not calculated using the combinatorial circuit), they are passed without errors to the neighboring check nodes. From the check node symmetry follows $\nu_{c,v}^{(0)}(\mathbf{y}) = x_v \nu_{c,v}^{(0)}(\mathbf{n})$ [4], and we have

$$\begin{aligned}\mu_{c,v}^{(0)}(\mathbf{y}) &= \Upsilon(\nu_{c,v}^{(0)}(\mathbf{y}), x_v B_v, e_{c,v}^{(0)}) \\ &= x_v \nu_{c,v}^{(0)}(\mathbf{n}) (e_{c,v}^{(0)})^{x_v(\nu_{c,v}^{(0)}(\mathbf{n}) - B_v)/2} \\ &= x_v \Upsilon(\nu_{c,v}^{(0)}(\mathbf{n}), B_v, e_{c,v}^{(0)}) = x_v \mu_{c,v}^{(0)}(\mathbf{n}). \end{aligned} \tag{1.4}$$

Similarly, we conclude $\mu_{v,c}^{(1)}(\mathbf{y}) = x_v \Upsilon(\nu_{v,c}^{(1)}(\mathbf{n}), A_v, e_{v,c}^{(1)}) = x_v \mu_{v,c}^{(1)}(\mathbf{n})$.

Let us assume $\mu_{v,c}^{(\ell)}(\mathbf{y}) = x_v \mu_{v,c}^{(\ell)}(\mathbf{n})$, $\forall v \in V$, $\forall c \in C$ and $\ell > 1$. From the fact that $\prod_{v:\mathcal{N}(c)} x_v = 1$ and from the check node symmetry defined in [4], it follows that $\nu_{c,v}^{(\ell)}(\mathbf{y}) = x_v \nu_{c,v}^{(\ell)}(\mathbf{n})$. Then, similarly as in Eq. (1.4), we have $\mu_{c,v}^{(\ell)}(\mathbf{y}) = \Upsilon(x_v \nu_{c,v}^{(\ell)}(\mathbf{n}), x_v \nu_{c,v}^{(\ell-1)}(\mathbf{n}), e_{c,v}^{(\ell)}) = x_v \mu_{c,v}^{(\ell)}(\mathbf{n})$. Furthermore, by invoking again the variable node symmetry condition and applying it to Eq. (1.3), we obtain $\mu_{v,c}^{(\ell+1)}(\mathbf{y}) = x_v \mu_{v,c}^{(\ell+1)}(\mathbf{n})$. As all messages passed between a node v and its neighbors are equal to the product of x_v and the corresponding message when $\mathbf{n}$ is received, the decoder performance is independent of the transmitted codeword.

On the other hand, if the theorem conditions were not satisfied, i.e. if $\nu_{v,c}^{(-1)} \neq x_v A_v$ or $\nu_{c,v}^{(-1)} \neq x_v B_v$, then it would follow that $\mu_{v,c}^{(1)}(\mathbf{y}) \neq x_v \mu_{v,c}^{(1)}(\mathbf{n})$. As messages $\mu_{v,c}^{(1)}$ continue to further propagate in the subsequent iterations, the error correction, and thus FER, depends on $\mathbf{x}$. ■

The conditions required for FER independence of $\mathbf{x}$ are highly unrealistic. They can be satisfied only by adjusting the initial logic gate states to a transmitted codeword, which is unknown to the decoder. Thus, in the case of timing errors the decoder error-correction capability is conditional on $\mathbf{x}$.

On the other hand, we have shown that the main reason for such decoder behavior is related to failing to satisfy the check and the variable node symmetry conditions in the first decoding iteration. If the first iteration were free of logic gate failures, the symmetry conditions would be satisfied. The fault-free iteration can be achieved by forcing all transistors in the decoding circuitry to reach a stationary state. Practically, this can be done by slowing down the clock in the first iteration and letting the signal level stabilize. Since the clock is slower, there are no timing errors and the computations are reliable. The decoder in which the first iteration is reliable in the rest of the chapter is called *the modified Gallager B decoder*.

1.4 Numerical Results

The past work related to the faulty Gallager B decoder was mostly dedicated to the infinite code length analysis under independent identically distributed (i.i.d.) failures (von Neumann failure model) [5–7]. Due to the decoder asymmetry, the density evolution technique cannot be applied to the timing error model, when failures are present in the first iteration. Although, the modified decoder satisfy the symmetry conditions, the presence of memory in the VN and CN operations cause the complexity of density evolution computation to grow at least exponentially with the number of iterations [14]. For that reason in this chapter we focus on the finite length simulation analysis, where the maximal number of decoding iterations is limited to 100. We analyze LDPC codes free of small trapping sets. Such codes can be constructed, for example, from Latin Squares (LS) [15], or based on Progressive Edge Growth (PEG) technique [16]. In order to capture the effects of data-dependence three simulation modes are used:

- mode M_0: only the all-zero codewords are transmitted;

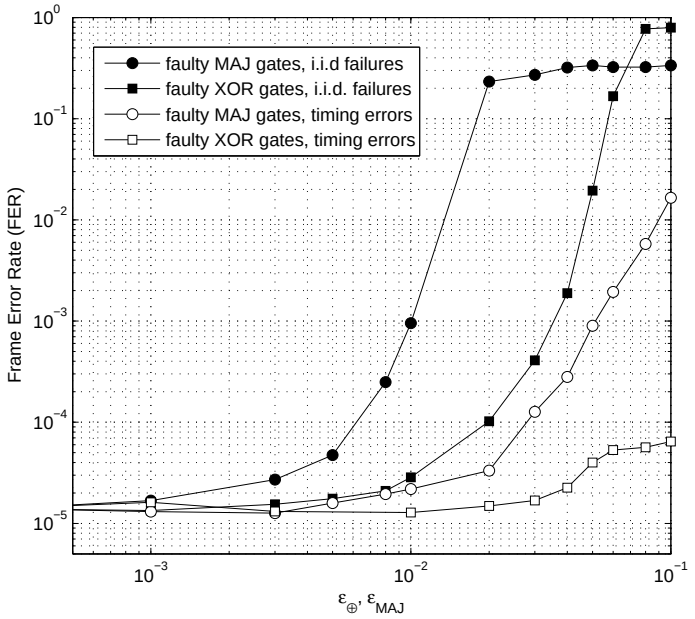


Figure 1.1: The comparison of i.i.d. and timing error models for the LS(155,64) code (α=0.01).

- mode M_H: the two codewords with large Hamming distance are transmitted alternately in a consecutive transmissions;
- mode M_R: randomly chosen codewords are transmitted.

In Fig. 1.1 we illustrate the decoder performance for the case of LS(155,64) code [15] evaluated under two failure models, when the most realistic setup M_R is used. It is a (3,5)-regular code with $|V| = 155$ and $|C| = 93$. We consider two different scenarios: (i) when CN processing is reliable and only MAJ gates are faulty, and (ii) when VN processing is reliable and only XOR gates are faulty. Note that no performance degradation is observed for logic gate failure rates below 10^{-3} regardless of the failure model. When logic gates failures become more frequent, a *threshold* is reached, and the FER rapidly increases. The threshold depends on the logic gate type and has lower values when the MAJ gates are faulty.

The timing error model reduces to the von Neumann model when a gate is constantly in a "bad state", i.e. changes its output value in every clock cycle. In that sense, the von Neumann model can be seen as pessimistic and misleading, which is especially pronounced in a case of highly unreliable XOR logic gates. The use of the timing error model reveals robustness of the Gallager B decoder to XOR gate failures. For example, even for the large bad state failure probability $\varepsilon_{\oplus} = 0.1$ the decoder retains low FER value. On the other hand, according to the von Neumann model, the faulty decoder is in average outperformed by an uncoded system.

For low and moderate channel error rates, in most of the cases only a few of 155 code bits are received incorrectly and the most of $\nu_{v,c}^{(0)}$ messages represent the correct bit estimates. If during the first iteration the number of gate failures is also low, most of the variable to check node messages will remain unchanged. However, if the number of gate failures is not negligible, after the first iteration, the number of incorrect bit estimates can be significantly increased. The effect of gate failures in the first iteration is illustrated in Fig. 1.2. It can be noticed that these failures have dominant influence on the decoder performance. When the same word is transmitted (mode M_0) there is no performance degradation even for high bad state failure rate ($\varepsilon_{\oplus} = \varepsilon_{MAJ} = 0.05$). On the other hand, mode M_H reveals that the worst case FER degradation can be of several orders of magnitude. Significant degradation is also notable in the more realistic scenario, which corresponds to the successive transmission of randomly chosen codewords.

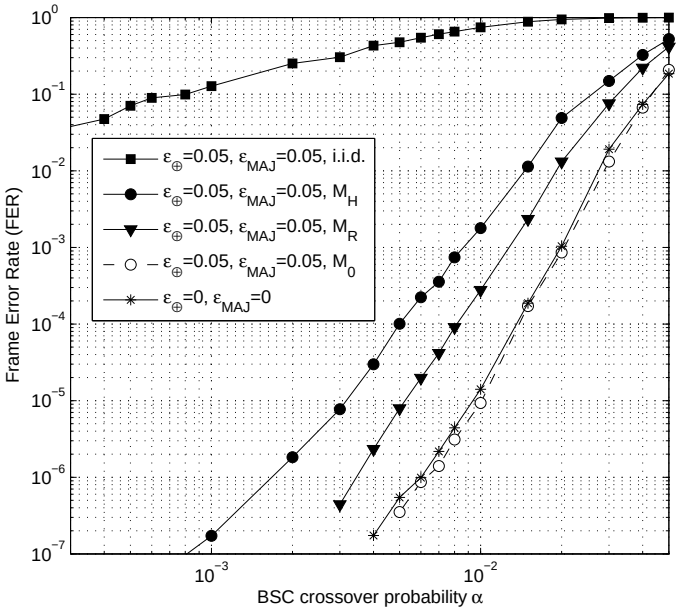


Figure 1.2: The performance dependence on codeword decoding order for LS(155,64) LDPC code.

In Fig. 1.3 we illustrate the improvement achieved by using the modified Gallager B decoder in which the first iteration is fault-free, for several $\gamma = 3$ codes. It can be seen that the modified Gallager B decoder, although built from unreliable components, performs approximately the same as perfectly reliable decoder. On the other hand, the FER of the Gallager B decoder without modification is several magnitude higher. For example, the code LS(2388,1793) is non-operational for wide range of channel error probabilities.

1.5 Conclusion

We have shown that the performance degradation of the Gallager B decoder caused by the timing violations depends of the sequence of transmitted codewords. In addition, we proposed the modification that ensures the decoder robustness to hardware unreliability. This result raises a number of open questions in analysis of faulty decoders. It is known that the capacity of LDPC codes under the von Neumann error model can not be achieved [4, 5]. However, the finite code length analysis presented in this chapter suggests more promising results for the timing error model. Additionally, in our future work we will apply the presented framework to investigation and design of fault-tolerant LDPC code-based memories.

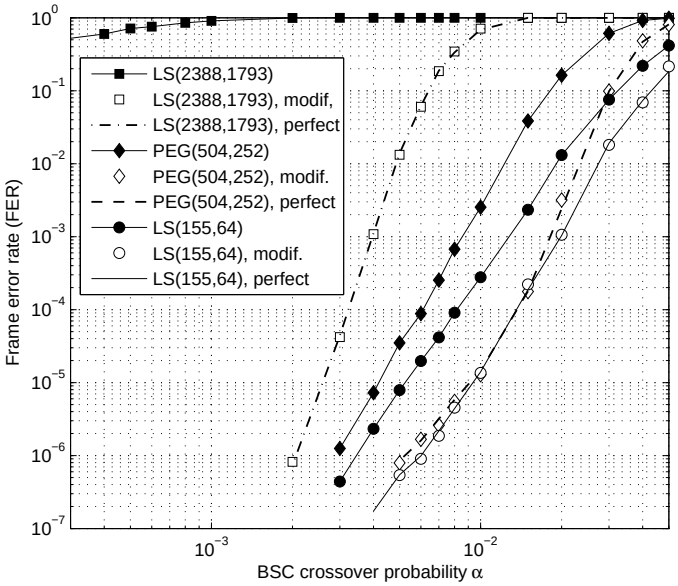


Figure 1.3: The performance comparison of different codes ($\varepsilon_{\oplus} = \varepsilon_{MAJ} = 0.05$) under simulation mode M_R.

Chapter 2

Faulty Finite Alphabet Iterative Decoders under Error Models with Memory

Abstract: *In our previous works, we considered LDPC decoders on faulty hardware under very simple memoryless error models. Based on a theoretical Density Evolution (DE) analysis of the performance of faulty decoders, we proposed a method for the design of Finite Alphabet Iterative Decoders (FAIDs) robust ot hardware errors under the memoryless model. In this chapter, we consider two more realistic error models that take into account temporal and spatial dependencies in hardware errors, and we investigate the robustness of FAIDs under the two models. The memory in the introduced models makes it very difficult to analyze theoretically the performance of the decoders under these models. Hence we analyze the performance of the decoders through Monte-Carlo simulations only. We reuse the FAIDs we designed for robustness under simple memoryless error models. Through Monte-Carlo simulations, we measure the performance of these FAIDs under error models with memory. We observe that the memory in error models degrades the performance of noisy FAIDs compared to the memoryless case. We also observe that the FAIDs designed for robustness in the memoryless case are still robust under error models with memory.*

2.1 Introduction

The asymptotic performance of LDPC decoders under faulty hardware has been widely investigated, based on the noisy Density Evolution (DE) framework introduced in [4]. Hard decoders such as noisy Gallager-A [4] and Gallager-E [17] decoders were considered. Gallager-B decoders were analyzed for binary [3, 6, 7] and non-binary [5] alphabets. In our previous works, we also analyzed stronger soft decoders such as Min-Sum (MS) [18] and Finite Alphabet Iterative Decoders (FAIDs) [19]. Based on the noisy-DE analysis for FAIDs, we proposed a method for the design of LDPC decoders naturally robust to hardware noise [19]. Finite-length simulations confirmed that the FAIDs we obtained from our design method were more robust compared to FAIDs optimized for ideal noise-free decoders.

As a first step of the analysis, most of the above works [3–5, 7, 18, 19] considered very simple transient memoryless models to represent the errors introduced by the hardware. More realistic error models were considered only in a few works. The performance of the Gallager B decoder under timing errors was analyzed in Chapter 1 through Monte-Carlo simulations. The authors in [6] also considered permanent errors in the Gallager B decoder, and did a noisy-DE analysis under simplifying memoryless assumptions. LDPC decoders under more realistic error models can be very difficult to analyze theoretically. In particular, noisy-DE becomes intractable for error models with memory, as shown in [14].

In this Chapter, we introduce two errors models that take into account the memory in hardware errors. The first model takes into account temporal dependencies, while the second one deals with

spatial dependencies. We reuse the FAIDs we designed for robustness under simple memoryless error models. Through Monte-Carlo simulations, we measure the performance of these FAIDs under error models with memory. We show that the FAIDs designed for robustness in the memoryless case are still robust under error models with memory.

The outline of the Chapter is as follows. Section 2.2 introduces our notations for LDPC codes and FAIDs. Section 2.3 defines the two error models with memory we consider for the performance evaluation of FAIDs. Section 2.4 presents the performance evaluation of decoders under the two models with memory.

2.2 Notations and Decoder Definitions

In the following, we assume that the transmission channel is a Binary Symmetric Channel (BSC) with parameter α. We first define FAIDs and introduce our notations for LDPC codes. We then present the decoders we designed for robustness under simple memoryless models.

2.2.1 Definition and Notations

An N_s-level FAID is defined as a 4-tuple given by $\mathrm{D} = (\mathcal{M}, \mathcal{Y}, \Phi^{(v)}, \Phi^{(c)})$. The message alphabet is finite and can be defined as $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$, where $L_i \in \mathbb{R}^+$ and $L_i > L_j$ for any $i > j$. It thus consists of $N_s = 2s + 1$ levels to which the message values belong. For the BSC, the set $\mathcal{Y}$, which denotes the set of possible channel values, is defined as $\mathcal{Y} = \{\pm \mathrm{B}\}$, where $\mathrm{B} \in \{-L_s, \ldots, L_s\}$. For the n-th symbol of the codeword, the channel value $y_n \in \mathcal{Y}$ corresponding to node v_n is determined based on its received value. Here, we use the mapping $0 \to \mathrm{B}$ and $1 \to -\mathrm{B}$. In the following, $\mu_1, \ldots, \mu_{d_c-1}$ denote the extrinsic incoming messages to a Check Node (CN) of degree d_c and let $\eta_1, \ldots, \eta_{d_v-1}$ be the extrinsic incoming messages to a Variable Node (VN) of degree d_v.

At each iteration of the iterative decoding process, the following operations defined in [1] are performed on the messages. The Check Node Update (CNU) function $\Phi^{(c)} : \mathcal{M}^{d_c-1} \to \mathcal{M}$ used for the update at a Check Node (CN) of degree d_c is given by $\Phi^{(c)}(\mu_1, \ldots, \mu_{d_c-1})$ and corresponds to the CNU of the standard min-sum decoding. The Variable Node Update (VNU) function $\Phi^{(v)} : \mathcal{M}^{d_v-1} \times \mathcal{Y} \to \mathcal{M}$ used for the update at a Variable Node (VN) v_n, $n = 0 \ldots N-1$ of degree d_v, is given by $\Phi^{(v)}(\eta_1, \cdots, \eta_{d_v-1}, y_n)$. The properties that $\Phi^{(v)}$ must verify are given in [1]. To finish, at the end of the decoding process, the *A Posteriori* (APP) computation produces messages γ calculated from the function $\Phi^{(a)} : \mathcal{M}^{d_v} \times \mathcal{Y} \to \bar{\mathcal{M}}$, where $\bar{\mathcal{M}} = \{-L_{s'}, \ldots, L_{s'}\}$ and $s' = 2s + 1$. The function $\Phi^{(a)}$ is expressed as

$$\Phi^{(a)}(\eta_1, \ldots, \eta_{d_v}, y_n) = \sum_{j=1}^{d_v} \eta_j + y_n \quad . \tag{2.1}$$

It is computed on a bigger alphabet $\bar{\mathcal{M}}$ in order to limit the influence of saturation effects when calculating the sum. The hard-decision bit corresponding to each variable node v_n is given by the sign of the APP computation. If the output of $\Phi^{(a)}$ is 0, then the hard-decision bit is selected at random and takes value 0 with probability 1/2.

The VNU $\Phi^{(v)}$ can also be represented as a Look-Up Table (LUT) that is defined for a specific channel value. Table 2.1 shows an example of LUT for a 7-level FAID and column-weight three codes when the channel value is $-\mathrm{B}$. The corresponding LUT for the value $+\mathrm{B}$ can be deduced by symmetry. The VNU $\Phi^{(v)}_{\mathrm{opt}}$ given in Table 2.1 has been optimized in the noise-free case for low error floor. VNUs can also be optimized for robustness to hardware noise, as we now describe.

2.2.2 Robust FAIDs under Memoryless Models

The VNU formulation defines a large collection of mappings with common characteristics but potentially different abilities to be robust to noise in the decoder. In [19], we proposed an asymptotic noisy-DE analysis of the performance of noisy FAIDs, under very simple memoryless under models. From the theoretical analysis, we designed VNUs for robustness to hardware noise. Table 2.2 gives

the VNU $\Phi^{(v)}_{\text{robust}}$ of the FAID optimized for robustness to hardware noise, while Table 2.3 gives the VNU $\Phi^{(v)}_{\text{non-robust}}$ of a FAID that is non-robust to hardware noise, under the memoryless model. The two VNUs $\Phi^{(v)}_{\text{robust}}$ and $\Phi^{(v)}_{\text{non-robust}}$ were obtained under the Sign-Preserving (SP) transient memoryless error model defined in [19].

In [19], we also verified through finite length simulations that the FAID defined by $\Phi^{(v)}_{\text{robust}}$ was indeed more robust under the memoryless SP-model compared to $\Phi^{(v)}_{\text{non-robust}}$ and $\Phi^{(v)}_{\text{opt}}$. Here, we introduce two new error models that take into account temporal and spatial dependencies in hardware errors. Under these two models with memory, we compare the Monte-Carlo performance of $\Phi^{(v)}_{\text{robust}}$ with the performance of $\Phi^{(v)}_{\text{opt}}$ and $\Phi^{(v)}_{\text{non-robust}}$ in order to determine whether the VNU optimized under the simple memoryless model is still robust under more realistic models with memory.

Table 2.1: LUT $\Phi^{(v)}_{\text{opt}}$ reported in [1] optimized for the error floor

$\boldsymbol{m_1/m_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_1$
$\boldsymbol{-L_2}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	L_1
$\boldsymbol{-L_1}$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	$-L_1$	L_1
0	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0	0	L_1
$\boldsymbol{L_1}$	$-L_3$	$-L_2$	$-L_1$	0	0	L_1	L_2
$\boldsymbol{L_2}$	$-L_3$	$-L_1$	$-L_1$	0	L_1	L_1	L_3
$\boldsymbol{L_3}$	$-L_1$	L_1	L_1	L_1	L_2	L_3	L_3

2.3 Error Models

In this section, we define the two error models with memory we will consider in our Monte-Carlo simulations. Both models are based on the SP-Model defined in [19] and assume that the APP is error free.

2.3.1 Error Model with Temporal Dependencies

In order to take into account temporal dependencies in hardware errors, we introduce a *global* noise status variable Z which can take only two values 0 and 1. The noise status is initialized as $Z = 0$ and it is updated *each time* a VNU or a CNU computation is performed, just before the computation. The update of Z is done according to the following probability distribution. Denote Z_{old} the previous noise status and Z_{new} the current one. The probability mass function that models the transitions between Z_{old} and Z_{new} is represented by two parameters p_{01} and p_{10} defined as

$$p_{01} = P(Z_{\text{new}} = 1 | Z_{\text{old}} = 0), \quad p_{10} = P(Z_{\text{new}} = 0 | Z_{\text{old}} = 1). \tag{2.2}$$

Then, if $Z = 1$, an error is introduced during the considered computation. The error is introduced according to the Sign-Preserving (SP) error model defined in [19]. Here, we assume that the final APP computation is error free.

Note that setting $p_{10} = 1$ and $p_{01} = 0$ defines a memoryless error model, while choosing parameters $0 < p_{10} < 1$ and $0 < p_{01} < 1$ enables to deal with temporal dependencies in hardware errors. We now proceed to describe the error model that takes into account spatial dependencies.

2.3.2 Error Model with Spatial Dependencies

Spatial dependencies are represented by *local* noise status variables $Z^{(v)}_{i,j}$ and $Z^{(c)}_{i,j}$ which can take only two values 0 and 1. The noise status $Z^{(v)}_{i,j}$ is associated to the VNU computation from VN i to CN j,

Table 2.2: FAID rule $\Phi^{(v)}_{\text{robust}}$ robust to the faulty Hardware (SP-Model). μ_1 and μ_2 are the input values of the VNU.

$\boldsymbol{\mu_1/\mu_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	0
$\boldsymbol{-L_2}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	L_1
$\boldsymbol{-L_1}$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$-L_1$	L_1
$\boldsymbol{0}$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$-L_1$	0	L_1
$\boldsymbol{+L_1}$	$-L_3$	$-L_2$	$-L_1$	$-L_1$	0	L_1	L_2
$\boldsymbol{+L_2}$	$-L_2$	$-L_2$	$-L_1$	0	L_1	L_2	L_2
$\boldsymbol{+L_3}$	0	L_1	L_1	L_1	L_2	L_2	L_3

Table 2.3: FAID rule $\Phi^{(v)}_{\text{non-robust}}$ not robust to faulty Hardware (SP-Model)

$\boldsymbol{\mu_1/\mu_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	0
$\boldsymbol{-L_2}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	0	L_2
$\boldsymbol{-L_1}$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	0	L_2
$\boldsymbol{0}$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0	L_1	L_3
$\boldsymbol{+L_1}$	$-L_3$	$-L_2$	$-L_1$	0	0	L_1	L_3
$\boldsymbol{+L_2}$	$-L_3$	0	0	L_1	L_1	L_1	L_3
$\boldsymbol{+L_3}$	0	L_2	L_2	L_3	L_3	L_3	L_3

while $Z^{(c)}_{i,j}$ is associated to the CNU computation from CN j to VN i. The noise status are initialized as

$$\forall i,j, \quad Z^{(v)}_{i,j} = 0 \text{ and } Z^{(c)}_{i,j} = 0. \tag{2.3}$$

The noise status variable $Z^{(v)}_{i,j}$ (respectively $Z^{(c)}_{i,j}$) is updated at each VNU (respectively CNU) computation from VN i to CN j (respectively from CN j to VN i), just before the computation. The transition probability mass function for $Z^{(v)}_{i,j}$ is represented by two parameters p_{01} and p_{10} defined as

$$p_{01} = P(Z^{(v)}_{i,j,\text{new}} = 1 | Z^{(v)}_{i,j,\text{old}} = 0), \quad p_{10} = P(Z^{(v)}_{i,j,\text{new}} = 0 | Z^{(v)}_{i,j,\text{old}} = 1) \tag{2.4}$$

where $Z^{(v)}_{i,j,\text{old}}$ is the previous noise status and $Z^{(v)}_{i,j,\text{new}}$ is the current one. For simplicity, we also assume that p_{01} and p_{10} define the probability transitions for $Z^{(c)}_{i,j}$. Then, if $Z^{(v)}_{i,j} = 1$, an error is introduced during the considered VNU computation from VN i to CN j. As before, the APP computation is assumed th be error free.

We now proceed to compare the finite-length performance of decoders $\Phi^{(v)}_{\text{robust}}$ with the performance of $\Phi^{(v)}_{\text{opt}}$ and $\Phi^{(v)}_{\text{non robust}}$ (obtained in the memoryless case) under the two defined error models with memory.

2.4 Simulation Results

In this section, we evaluate the performance at finite length of the three decoders defined by $\Phi^{(v)}_{\text{robust}}$, $\Phi^{(v)}_{\text{non robust}}$, $\Phi^{(v)}_{\text{opt}}$. The two decoders $\Phi^{(v)}_{\text{robust}}$, $\Phi^{(v)}_{\text{non robust}}$ were obtained for robustness according to the memoryless SP model defined in [19]. The decoder $\Phi^{(v)}_{\text{opt}}$ was optimized for low error floor in a noise-free setup. Here, we evaluate the performance of $\Phi^{(v)}_{\text{robust}}$, $\Phi^{(v)}_{\text{non robust}}$, $\Phi^{(v)}_{\text{opt}}$. according to the two introduced error models with memory. For the performance evaluation, we consider two different codes. The first code is the Tanner_dv3 code, with regular degrees $d_v = 3$ and $d_c = 5$ and length $n = 155$. The second so called Mat_35 code is also a regular code with $d_v = 3$ and $d_c = 5$, with length $n = 1035$ and girth 12.

2.4.1 Error Model with Temporal Dependencies

We first consider the Tanner_dv3 code. Figure 2.1 (a) represents the Bit Error Rate (BER) with respect to α for the three decoders $\Phi^{(v)}_{\text{robust}}$, $\Phi^{(v)}_{\text{non robust}}$, $\Phi^{(v)}_{\text{opt}}$. The performance of the three decoder is evaluated under the noise-free setup, and also under the error model with temporal dependencies. For the noisy case, we set $p_{01} = 0.05$ and $p_{10} = 0.95$. In the noiseless case, we see that, as expected, $\Phi^{(v)}_{\text{opt}}$ gives the lowest BER performance. In the noisy case, we see that, as under the memoryless model, $\Phi^{(v)}_{\text{robust}}$ performs better than $\Phi^{(v)}_{\text{opt}}$. We also observe that $\Phi^{(v)}_{\text{non robust}}$ shows poor BER performance.

We now consider the Mat_35 code. Figure 2.1 (b) represents the BER for the same three decoders under two different error models. The BER performance of the decoders is evaluated under the memoryless SP-model, setting $p_{01} = 0.05$ and $p_{10} = 1$. The BER performance of the decoders is

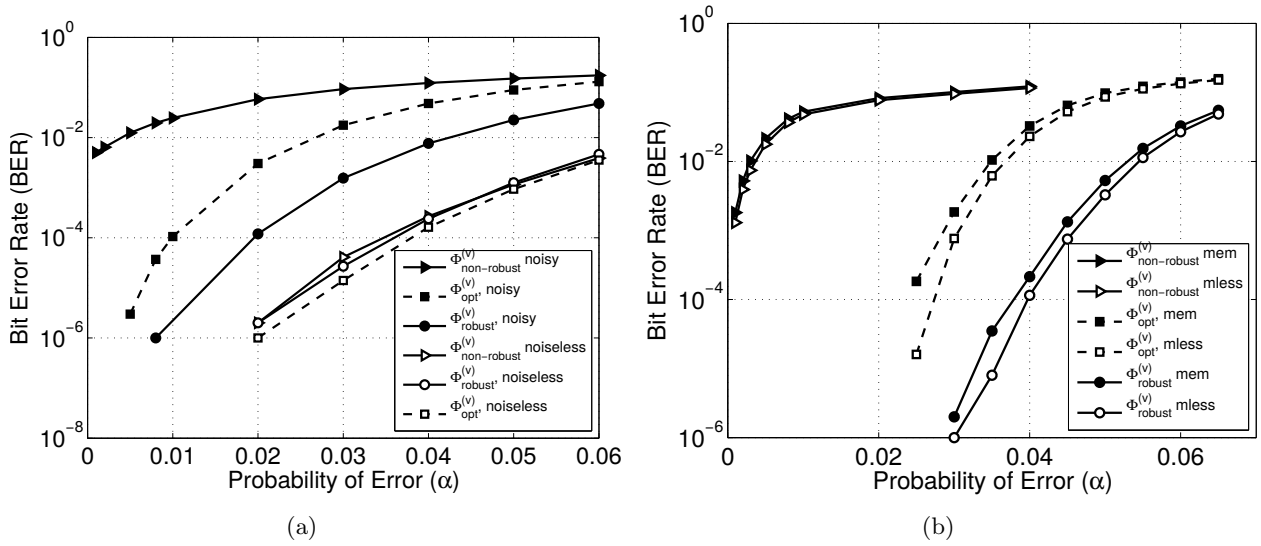


Figure 2.1: BER with respect to α, (a) Tanner_dv3 code, noiseless decoders and noisy decoders under error model with temporal dependencies $p_{01} = 0.05$ and $p_{10} = 0.95$, (b) Mat_35 code, noisy decoders under memoryless (mless) error model, $p_{01} = 0.05$, $p_{10} = 1$, and under error model with temporal dependencies (mem), $p_{01} = 0.026$ and $p_{10} = 0.5$

also evaluated under the error model with temporal dependencies. For the error model with temporal dependencies, we set $p_{01} = 0.026$ and $p_{10} = 0.5$, which corresponds to the same stationary probability $P(Z = 0) = 0.05$ as in the memoryless case, but with an important memory. For both models, $\Phi^{(v)}_{\text{robust}}$ performs better than $\Phi^{(v)}_{\text{opt}}$ and $\Phi^{(v)}_{\text{non robust}}$. We also observe that the performance of the decoders under the model with memory is degraded compared to the memoryless model, which is expected.

The results of Figure 2.1 (a) and Figure 2.1 (b) thus suggest that it is sufficient to optimize the VNU for the simple memoryless error model. The optimized FAID will still be robust under the error model with temporal dependencies.

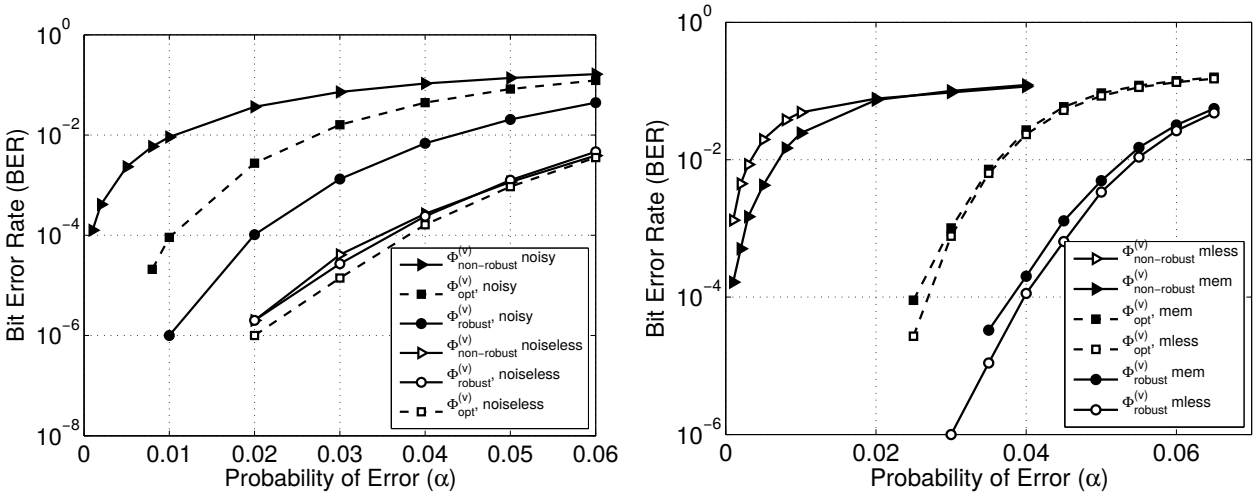
2.4.2 Error Model with Spatial Dependencies

We perform the same set of experiments under the error model with spatial dependencies.

We first consider the Tanner_dv3 code. Figure 2.2 (a) represents the Bit Error Rate (BER) with respect to α for the three decoders, in noiseless and in noisy setups under the error model with spatial dependencies, with $p_{01} = 0.05$ and $p_{10} = 0.95$. With this model as well, the robust decoder $\Phi^{(v)}_{\text{robust}}$ performs better than $\Phi^{(v)}_{\text{opt}}$ and $\Phi^{(v)}_{\text{non robust}}$.

We then consider the Mat_35 code. Figure 2.2 (b) represents the BER performance of noisy decoders, under the memoryless model with $p_{01} = 0.05$, $p_{10} = 1$, and under the model with memory and $p_{01} = 0.026$ and $p_{10} = 0.5$. In both cases, we see that $\Phi^{(v)}_{\text{robust}}$ performs better than $\Phi^{(v)}_{\text{opt}}$, and that $\Phi^{(v)}_{\text{non robust}}$ shows poor performance. For $\Phi^{(v)}_{\text{robust}}$ and $\Phi^{(v)}_{\text{opt}}$, we see that the memory in the error model degrades the performance. For $\Phi^{(v)}_{\text{non robust}}$, the memory improves the performance, but $\Phi^{(v)}_{\text{non robust}}$, which may be explained by the fact that $\Phi^{(v)}_{\text{non robust}}$ is anyway a bad decoder.

At the end, the performance evaluation under the two error models with memory show that FAIDs designed for robustness under the memoryless error model are still robust under error models with memory. This result suggests that it may be sufficient to optimize the VNU for the simple memoryless error model and to apply it for error models with temporal and spatial dependencies.



(a) (b)

Figure 2.2: BER with respect to α, (a) Tanner_dv3 code, noiseless decoders and noisy decoders under error model with spatial dependencies $p_{01} = 0.05$ and $p_{10} = 0.95$, (b) Mat_35 code, noisy decoders under memoryless (mless) error model, $p_{01} = 0.05$, $p_{10} = 1$, and under error model with spatial dependencies (mem), $p_{01} = 0.026$ and $p_{10} = 0.5$

2.5 Conclusion

In this chapter, we considered FAID decoders and introduced two error models with memory to represent more accurately the hardware errors. The first error model takes into account temporal dependencies while the second one considers spatial dependencies. Through Monte-Carlo simulations, we showed that the decoders that were optimized for robustness to noise under the memoryless model are still robust under the two introduced error models with memory.

Chapter 3

Hardware Efficient Implementation of Probabilistic Gradient Descent Bit-Flipping

Abstract: *In this chapter, we introduce a new and original method for the random generation of bits in the probabilistic gradient-descent bit-flipping (PGDBF) decoder, which provides an highly efficient hardware implementation compared to other conventional methods while preserving the outstanding decoding performance of PGDBF. This new random generator method called Intrinsic-valued random generator avoids generating the required random bits but uses sequences of bits from the existing decoder memory which is the source of the simplicity and hardware efficiency. Moreover, we analyze through a statistical analysis the behavior of the PGDBF and identify the important features of the random generators, which are important to obtain large coding gains compared to the deterministic GDBF. Our results show that the good PGDBF performance can be obtained with a very small extra complexity, without any performance loss.*

3.1 Introduction

Low-Density Parity-Check (LDPC) codes have attracted much attention in the past several years due to their excellent performance under iterative decoding, which is defined as an iterative exchange of information between the nodes of the Tanner graph representation of the LDPC code [20]. These studies focus either on further improving the error correction capacity 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) based decoders offer the best error correction performance, close to the theoretical coding bounds, but comes along with an intensive computation cost [21]. On the contrary, the class of hard-decision iterative decoders is often seen as a very low-complexity solution to efficiency decode LDPC codes, with an associated error-correction performance loss. Of special interest is the class of APP-based hard decision decoders, such as Bit-Flipping (BF) or Majority-Logic decoding where, contrary to message-passing decoders, both the extrinsic and the intrinsic information are exchanged between the nodes of the Tanner graph [22, 23].

The BF algorithms significantly reduce the required implementation hardware resource due to their simple computation units, but lead to a non-negligible performance loss compared to BP or MS algorithms. In order to improve the performance while keeping the low complexity, many modifications of BF decoders have been introduced, such as Weighted BF (WBF) [24], Improved WBF [22], Gradient Descent Bit Flipping (GDBF) [25].

All the BF decoders share the same concept of passing *only one bit of information* between the variable nodes (VNs) and check nodes (CNs) of the LDPC Tanner graph. The difference between BF variants lies on the mechanism proposed to select the bits to be flipped, based on the computation of the so-called *inversion function*. Another difference lies in the number of bits that are flipped during

one decoding iteration: the serial BF decoder flips only one single bit at a time, while the parallel or semi-parallel BF flips many bits at the same time. For weighted BF decoders [24,26,27], the inversion function is specifically designed for the AWGN channel through a combination of the CN values and the channel measurement. All these BF variants achieve different level of error correction and different convergence speed, impacting on the performance/throughput trade-off of the LDPC decoder.

More recently, the GDBF algorithm has been introduced by Wadayama et al. in [25]. This GDBF algorithm is derived from a gradient descent formulation and its principle consists in finding the most suitable bits to be flipped in order to maximize a pre-defined objective function. The GDBF algorithm showed an error correction capability superior to most known BF algorithms while still keeping the hardware implementation simplicity. A very promising generalization of the GDBF has been proposed by Rasheed et al. in [28]. The authors proposed to incorporate a probabilistic feature in the flipping step, inspired from the Probabilistic BF algorithms of [22]. In the Probabilistic GDBF (PGDBF) decoder, all the bits that satisfy the gradient descent condition are not systematically flipped, but instead, only a randomly chosen fraction p_0 of them are flipped. Interestingly, this small modification of the GDBF algorithm led to a large performance improvement, with error correction capability approaching the soft-decision message passing decoders [28].

In this chapter, we propose to conduct a statistical analysis of the PGDBF in order to have a precise characterization of its key parameters, and we also propose a modification of the algorithm that avoids the use of a random generators, in order to reduce the hardware complexity of the PGDBF implementation.

In [28], the authors have optimized the parameters of the PGDBF, and especially the value p_0 of the Bernoulli random generator (RG) probability, for some test codes. The first objective of our work is to provide a more precise and more general characterization of the PGDBF features, through a Monte Carlo statistical analysis of both the waterfall and the error floor performance of the decoder. A conclusion of this analysis is that an optimized value of the RG probability p_0 is not crucial to get the performance improvement, since a wide range of values for p_0 can be used to achieve the gain. Based on this observation, we also propose to avoid the RGs in the decoder implementation to reduce the hardware overhead of the PGDBF. Indeed, the advantages of PGDBF come along with the implementation cost of random generators [29]. With a conventional realization of the RGs using Linear Feedback Shift Register (LFSR), the hardware overhead that implements the random steps is greatly increased. In this chapter, we propose a new method to obtain a random binary sequence from the existing decoder memory, called Intrinsic-valued Random Generator (IVRG). This approach to generate a random sequence is based on the CN values at the beginning of the decoding process, and provides an efficient implementation of the PGDBF while keeping the same performance gains compared to the use of LFSR random generators (LSFR-RG).

The rest of the chapter is organized as follows. In section 3.2, the general notations for LDPC codes and BF decoders are recalled and the advantages of PGDBF are presented. Section 3.3 presents the PGDBF's statistical analysis which indicates the requirements for the random binary sequence to obtain good error correction performance. Then, our new algorithm IVRG-PGDBF is presented in section 3.4.1 and the hardware implementation results and performance comparison are made in section3.4.4.

3.2 Advantages of the PGDBF and Implementation Issues

3.2.1 Notations and PGDBF algorithm

An LDPC code is defined by a sparse parity-check matrix H with size (M, N), where $N > M$. A codeword is a vector $x = (x_1, x_2, \dots x_N) \in \{0,1\}^N$ which satisfies $H.x^T = 0$. We denote by $y = \{y_1, y_2, \dots, y_N\} \in \{0,1\}^N$ the output of a binary symmetric channel (BSC), in which the bits of the transmitted codeword x have been flipped with crossover probability α. The decoders presented in this chapter are dedicated to the BSC channel. The classical graphical representation of an LDPC code is a bipartite graph called Tanner graph composed of 2 types of nodes, the VNs $v(i)$, $i = 1 \dots N$ and the CNs $c(j)$, $j = 1 \dots M$. In the Tanner graph, a VN $v(i)$ is connected to a CN $c(j)$

if $H(j,i) = 1$. Let us also denote $\mathcal{N}(v(i))$ the set of CNs connected to the VN $v(i)$, with connection degree $d_{v(i)} = |\mathcal{N}(v(i))|$, and denote $\mathcal{N}(c(i))$ the set of VNs connected to the CN $c(j)$, with connection degree $d_{c(i)} = |\mathcal{N}(c(i))|$.

A BF decoder is defined as a iterative update of the variable node values over the decoding iterations. We denote in this chapter by $v^{(k)}(i)$ the value of the variable node at the k-th iteration. We correspondingly denote by $c^{(k)}(j)$ the binary value of the parity checks at iteration k, which represent the fact that the $j-th$ parity-check equation is satisfied or not. The BF decoding process is terminated when all CNs values are satisfied or a maximum number of iteration is reached. The CN calculation in BF algorithms can be written as $c^{(k)}(j) = \mathbf{XOR}_{v(i)\in\mathcal{N}(c(j))} v^{(k-1)}(i)$, with **XOR** is the bit-wise Exclusive-OR operation.

For the $i-th$ VN calculation, the update rule uses the information on satisfiability of the neighboring check-nodes $\mathcal{N}(v(i))$ to keep or flip the value of $v^{(k)}(i)$. In the case of GDBF algorithms, a function called *inversion function* is defined for each VN, and used to evaluate whether the value $v^{(k)}(i)$ should be flipped or not. The original GDBF has been proposed for the Additive White Gaussian Noise (AWGN) channel [25] and the inversion function was defined as in (3.1) where γ_i is the log-likelihood ratio (LLR) received from AWGN channel.

$$\Lambda_{v(i)}^{(k)} = (1 - 2\, v(i)^{(k)})\gamma_i + \sum_{c(j)\in\mathcal{N}(v(i))} (1 - 2\, c(j)^{(k)}) \tag{3.1}$$

For the GDBF on the AWGN channel, the inversion function is real valued, and has a unique minimum, corresponding to the bit with lowest reliability. In [25], two modes for the bit-flipping rule at iteration k are proposed: either only the bit having smallest inversion function is flipped (single flip), or a group of bits having inversion function lower than a predefined threshold are flipped (multiple flips).

For the BSC channel, the inversion function can be modified with following equation (3.2).

$$E_{v(i)}^{(k)} = v(i)^{(k)} XOR y_i + \sum_{c(j)\in\mathcal{N}(v(i))} c(j)^{(k)} \tag{3.2}$$

In this case, the inversion function is an integer and varies from 0 to $d_{v(i)} + 1$, and the bits which have the maximum value of $E_{v(i)}^{(k)}$ are flipped. Due to the integer representation of inversion function, many bits are likely to have the same maximum inversion function, leading to the multiple flips mode. The fact that the number of bits to be flipped cannot be precisely controlled induces a negative impact to the convergence of the algorithm, as the analysis of [28] shows. To avoid this effect, the PGDBF has been proposed with the following strategy: instead of flipping all the bits with maximum inversion function value, only a random fraction of those bits are flipped. The random fraction is fixed to a pre-defined probability $p_i^{(k)}$, which could be different for each VN and each iteration. In this work, we restrict ourself in keeping $p_i^{(k)}$ constant for all iterations and all VNs, denoted $p_0 \in [0, 1]$ hereafter.

The PGDBF is implemented as follows: a sequence of N random bits is generated with a Bernoulli RG, and triggered to the N VNs at each iteration. We denote this random sequence at the $k-th$ iteration $R^{(k)}$, with $p(R_i^{(k)} = 0) = p_0, i = 1 \ldots N$. The difference in flipping decision between GDBF and PGDBF is described in figure 3.1. In the GDBF algorithm, a VN $v_i^{(k)}$ at iteration k is flipped (is XOR-ed by '1') when its inversion function is a maximum (the variable $I = 1$) while in PGDBF, a VN $v_i^{(k)}$ is flipped if and only if the two conditions $I = 1$ and $R_i^{(k)} = 0$, are both satisfied. The PGDBF algorithm is presented in Algorithm 1.

3.2.2 Importance of the Random Part in PGDBF

Several works showed that the decoding performance of PGDBF is superior to all known BF algorithms [29][28] as illustrated in figure 3.2 for a rate $R = 1/2$ quasi-cyclic LDPC code of length $N = 512$ bits.

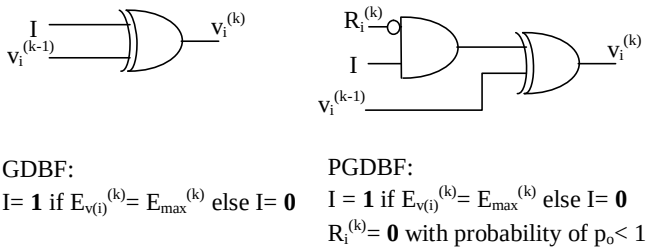


Figure 3.1: The difference between GDBF and PGDBF

Algorithm 1 Probabilistic Gradient Descent Bit-Flipping

Initialization $v_i^{(init)} \leftarrow y_i, i = 1 \dots N$, $k = 1$.
$syndrome = H.(v^{(init)})^T \bmod 2$
while $syndrome \neq 0 \quad and \quad k \leq k_{max}$ **do**
  $\forall i \in [1, N]$
  $E_{v(i)}^{(k)} = v_i^{(k)} XOR y_i + \sum_{c_j \in \mathcal{N}(v(i))} XOR_{v_u \in \mathcal{N}(c(j))} v_u^{(k)}$,
  $E_{max}^{(k)} = max(E_{v(i)}^{(k)}), i = 1 \dots N$.
  if $E_{v(i)}^{(k)} = E_{max}^{(k)}$ **then**
    if $R_i^{(k)} = 0$ **then**
      $v_i^{(k)} = NOT(v_i^{(k)}); \quad \{p(R_i^{(k)} = 0) = p_i^{(k)} = p_0\}$
    end if
  end if
  $syndrome = H.(v^{(k)})^T \bmod 2$
  $k = k + 1$
end while
Outputs : $v^{(k)}$

Note that the difference between GDBF and PGDBF comes from the presence of the random generator described in figure 3.4 which underlines the importance of randomness for PGDBF performance.

A different illustration of this performance gain is shown in figure 3.3. In the GDBF algorithm, most errors come from oscillations in the decoder state space, and prevents the decoder to successfully correct the errors, while only a small number of bits in error remain. The introduction of the random part in PGDBF contributes to breaking these oscillations. In the left part of figure 3.3, the types of oscillations for the GDBF (corresponding to the curves of figure 3.2) have been recorded for several channel crossover probabilities. These error patterns that cannot be corrected by a deterministic GDBF are then re-decoded by the PGDBF, and the statistics are shown on the right part of figure figure 3.3. It can be seen that most of the frame error types of GDBF are corrected by the PGDBF.

3.2.3 Issue of Hardware Implementation Cost for the Random Generators

The random feature of PGDBF plays an important role in improving the decoder's performance as presented in the previous section. However an hardware overhead is unavoidable due to the fact that a binary random generator is required on top of the original GDBF structure described in figure 3.4. This random generator produces a binary sequence of N bits at each decoding iteration k with a probability p_0, $p(R_i^{(k)} = 0) = p_0$.

To emphasize the PGDBF's hardware overhead over the non-probabilistic GDBF, we have implemented the PGDBF for the Tanner code (155,93) [29] on FPGA with a naïve implementation of the binary random generator blocks using LFSR. The generic architecture for this LFSR-RG is described in figure 3.5 and is briefly described thereafter.

We make use of LFSR with the maximum length feedback polynomial to generate an integer

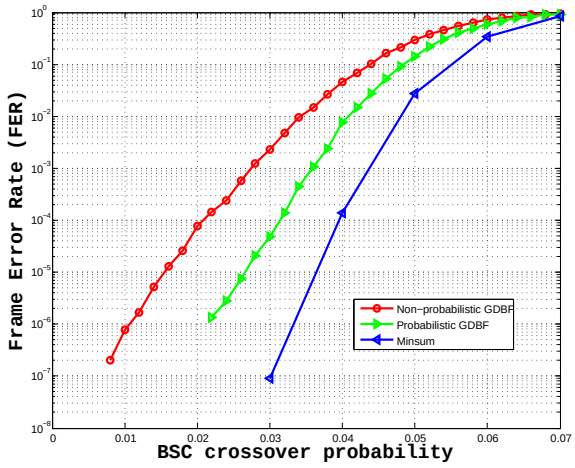


Figure 3.2: Comparison in decoding performance between PGDBF and known BF algorithms

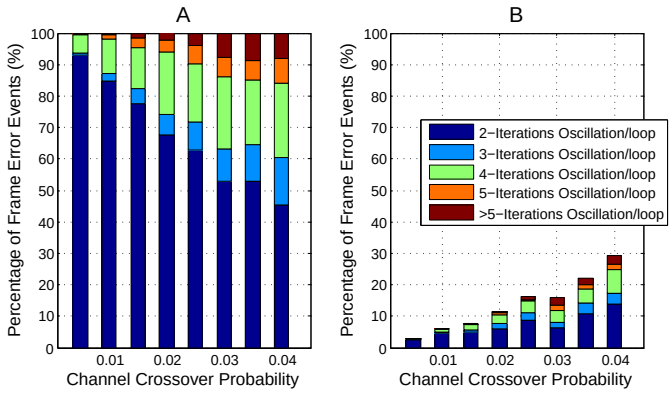


Figure 3.3: A. GDBF's frame error types of the Tanner code (155,93), $d_v = 3$ with maximum iteration: 100. B. The remain after re-decoding all errors in A by using PGDBF

number. This generated number is compared with a threshold to decide whether the new bit generated should be 0 (larger than threshold) or 1 (smaller than threshold). With such a variable threshold binary random generator, the granularity and the period are sufficiently large to ensure that the generated random sequence will appear random to decoding process. The hardware resource usage on the FPGA for both the GDBF and the LSFR-PGDBF are summarized on table 3.1. The LSFR-PGDBF requires 8 times more registers and 1.65 times more slide LUTs (slice Look-up tables) compared to GDBF. Such expensive cost in the hardware implementation can be seen as a major drawback of the PGDBF and motivates us to propose new methods to generate the sequences $R_i^{(k)}, i = 1 \dots N$, not based on classical RGs.

Before presenting our new algorithm, we conduct in the next section a statistical analysis of the PGDBF in order to fully understand what features are especially important in the random sequences $R_i^{(k)}, i = 1 \dots N$, in order to improve the performance.

3.3 Statistical Analysis of PGDBF

3.3.1 Waterfall Analysis

In order to better understand the impact of the randomness of PGDBF on the Frame Error Rate (FER), we have conducted a deep statistical analysis of PGDBF using Monte Carlo simulations. Our objective is to identify which features of the probability density function of the binary random sequence

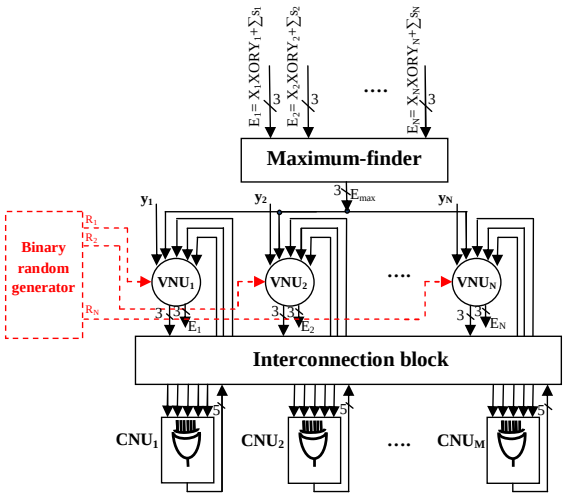


Figure 3.4: The PGDBF global hardware architecture. The difference between PGDBF and GDBF comes from the presence of the Binary Random Generator

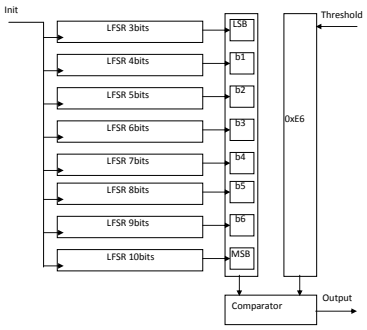


Figure 3.5: The Linear Feedback Shift Register random generator

$R^{(k)} = \left\{ R_i^{(k)}, i = 1 \dots N \right\}$ are the most critical for the performance improvements.

In the remaining of this section, we will focus on the ratio between the number of O's and 1's in the sequence $R^{(k)}$, rather than using the Bernoulli probability p_0. We denote the number of 1's in the binary random sequence $R^{(k)}$ as L. In our analysis, we have randomly allocated a given number of 1's in the sequence and computed the FER as a function of L, $F(L)$ for some values of decoding iteration k. The results presented in figure 3.6 are for the Tanner code $(N = 155, M = 93)$ which is a regular quasi-cyclic LDPC code with $(d_v = 3, d_c = 5)$.

From this figure, two interesting conclusions can be drawn:

1. *Conclusion 1:* In the first decoding iterations ($k < 10$), using randomness does not help. On the contrary, it degrades the decoding performance since The FER of PGDBF is worse than that of GDBF at all values of L (note that GDBF is PGDBF at $L = 0$). 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 inversion function indicates. As a consequence, large weights error patterns are decoded with more iterations with the PGDBF than with the GDBF.

2. *Conclusion 2:* After a subsequent number of iterations, the performance gain flattens for a wide range of L. This means in particular that choosing an optimized value for the RG probability p_0 is not that important to get the performance gain, contrary to what was indicated in [28]. This analysis also indicates that with a fixed number of 1's in $R^{(k)}$, but randomly located, the PGDBF can achieve similar coding gains as with a sequence generated from the Bernoulli distribution. Finally, the number of 1's needs to be sufficiently large $L > 20$ to get the best performance

Table 3.1: Hardware estimation for LFSR-PGDBF with with comparison to GDBF

	Number of 1-bit Register	Number of Slice LUTs
Non-Probabilistic GDBF	946	2151
PGDBF with LFSR	9161	3545

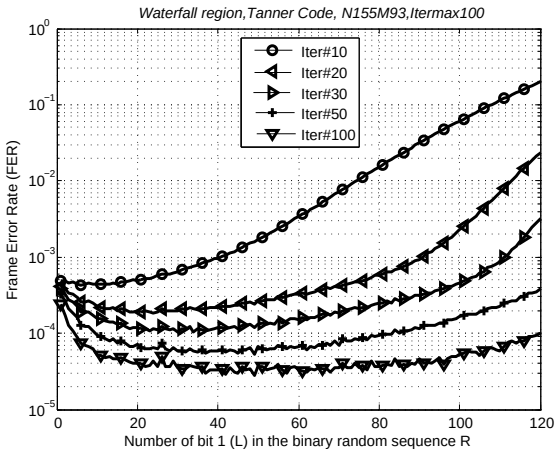


Figure 3.6: Frame Error Rate versus the number of '1' of $R^{(k)}$ in the waterfall region ($\alpha = 0.01$). Tanner code ($d_v = 3, d_c = 5$), $N = 155$.

results.

We have confirmed those conclusions for several regular LDPC codes with different length and VN degrees d_v. Figure 3.7 show the same statistics, for a longer regular LDPC code, with the same conclusions.

3.3.2 Error Floor Analysis

In the error floor region, the dominant uncorrectable error configurations are concentrated on Trapping Sets [21]. The smallest trapping set for the Tanner code is composed of 5 bits formed from three cycles of length 8, and denoted TS$(5, 3)$ (see figure 3.8).

The minimim number of bits that cannot be corrected by the deterministic GDBF is three, and are located in the TS$(5, 3)$ of the LDPC code as indicated in figure 3.8. Note that (v_1, v_2, v_3) and (v_1, v_4, v_3) are also 3-error patterns which cannot be corrected by the GDBF. The probabilistic part of the PGDBF can potentially help to correct these low weight error patterns, resulting in a coding gain in the error floor region.

In order to analyze the PGDBF in the error floor, we fix the channel errors on one of the configurations indicated by figure 3.8, and evaluate the effect the random sequence $R^{(k)}$ with weight L bits, with Monte Carlo simulations. In practice, we change the seed of the RG for each Monte Carlo experiment, in order to generate constructively different random sequences. Results are shown in figure 3.9.

The *conclusion 1* from the waterfall analysis does not hold in the error floor, because the convergence speed matters less for very low weight error patterns. However, an important confirmation is that *conclusion 2* still holds, as the FER curve flattens also for a wide range of L values, similarly to what was observed in the waterfall region.

We further conducted this analysis with weight 4-error patterns. It can be shown that 4 errors located inside a TS$(5, 3)$ are actually corrected by the GDBF algorithm (provided loose conditions on the connexion neighborhood of the TS). On the other hand, 4 errors located as indicated in figure 3.8

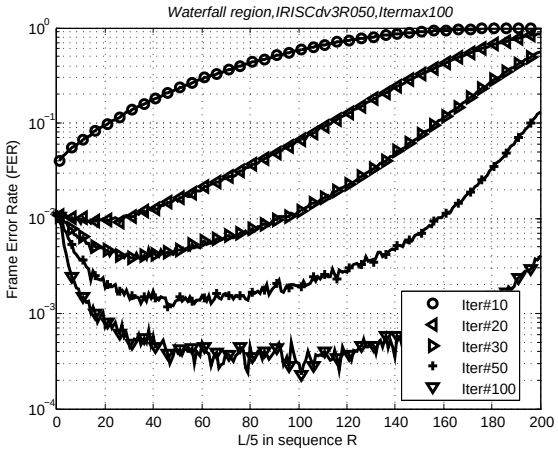


Figure 3.7: Frame Error Rate versus the number of '1' of $R^{(k)}$ in the waterfall region ($\alpha = 0.01$). Regular Quasi-Cyclic LDPC code ($d_v = 3, d_c = 6$), $N = 1296$.

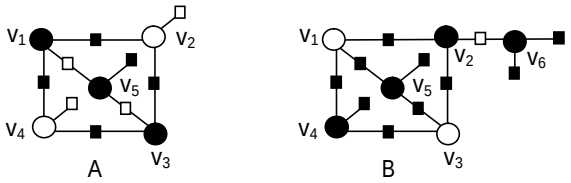


Figure 3.8: Error configurations with 3 and 4 erroneous bits used to analyse in error floor decoding region

cannot be corrected by the GDBF, but could be corrected with the probabilistic version. The same analysis was performed than for the 3-error patterns, and the results are shown in figure 3.10. For 4-error patterns, the performance gain is less impressive, and flattens at FER$= 5\,10^{-1}$. This means that in 50% of the cases, the random sequences $R^{(k)}$ can correct a given 4-error pattern. Figure 3.10 confirms the conclusions drawn for the 3-error patterns.

The statistics of figure 3.10 give rise also to the idea of re-initialization of the decoder. Indeed, those results imply that, PGDBF's performance depends on the random sequence realizations which may either lead the decoder to converge or completely diverge (the same FER for iterations $20 \leq k \leq 100$). Instead of using the PGDBF with many iterations, one could think of stopping the decoder after a small number of iterations and re-run it with the same received codeword from the channel, but with a different random seed for the RG. For weight 4-error patterns, this technique would provably improve the performance to $FER = (\frac{1}{2})^P$ with P being the number of re-initializations of the decoder. Such a decoder with re-initialization could then provably correct all 4-error patterns. This effect is less obvious for the waterfall region or the 3-error patterns, since the PGDBF performance continue to improve after 20 iterations. We will explore this direction of research in future works.

As a conclusion of this section, our statistical analysis reveals that the random generator does not need to be exactly implemented, and especially the probability density function of the number of 1's in $R^{(k)}$ does not seem to matter a lot to get good performance results. In particular, instead of a Bernoulli RG with optimized p_0 as advised in [28], a sequence $R^{(k)}$ with a sufficient number of 1's (e.g. in Tanner code, $5 \leq L \leq 110$), with different random localizations at each iteration, would suffice to improve the performance of the GDBF. This motivated the proposition of a simplified hardware realization for the generation of sequence $R^{(k)}$, which we called Intrinsic-valued Random Generator (IVRG), described in section 3.4.

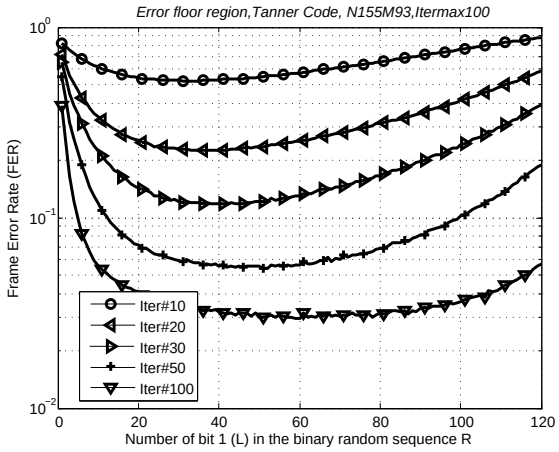


Figure 3.9: Frame Error Rate versus the number of '1' of $R^{(k)}$ in the error floor region with 3 erroneous bits

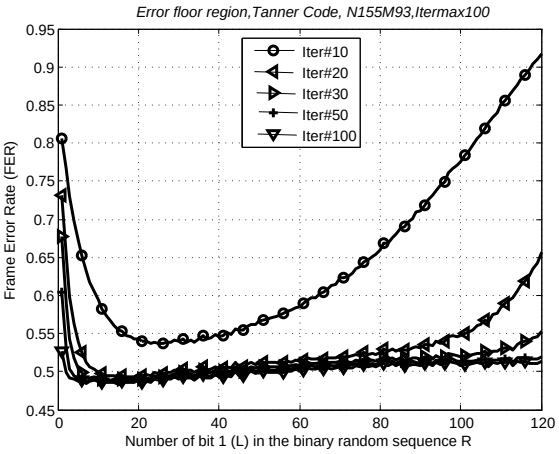


Figure 3.10: Frame Error Rate versus the number of '1' of $R^{(k)}$ in the error floor region with 4 erroneous bits

3.3.3 Partial PGDBF with Truncated Random Sequences

The previous results show that the number of 1's present in the random sequence $R^{(k)}$ does not need to be fixed accurately. In order to push forward this analysis, we have repeated the same procedure, but applying only a truncated version of $R^{(k)}$. Our objective is to identify if the random perturbation of the PGDBF needs to be applied on all the bits of the codeword, or if the coding gains can still be achieved if only a subset of the codeword bits are perturbed by the RGs.

In figures 3.11 and 3.12, we have applied the random bits of $R^{(k)}$ to only $N' = \frac{4}{5} \times N$ of the codeword. Both curves have been produced for the Tanner code, in the waterfall region ($\alpha = 0.01$). Figure 3.11 corresponds to a fixed allocation of the N' bits, while figure 3.11 corresponds to a random allocation of the N' bits, different at each iteration. One can see that although the statistics are slightly different, *Conclusion 1* and *Conclusion 2* from the previous sections still hold. We will use this property to reduce the complexity of the PGDBF algorithm implementation.

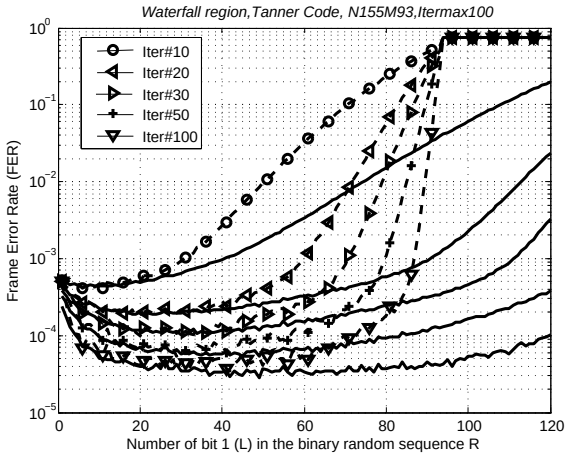


Figure 3.11: Frame Error Rate versus the number of 1's in $R^{(k)}$ in the case that only N' fixed VNs of the LPDC use randomness and the rest (N-N') still use the non-probabilistic GDBF (dashed lines). The solid lines represent the full allocation on N bits.

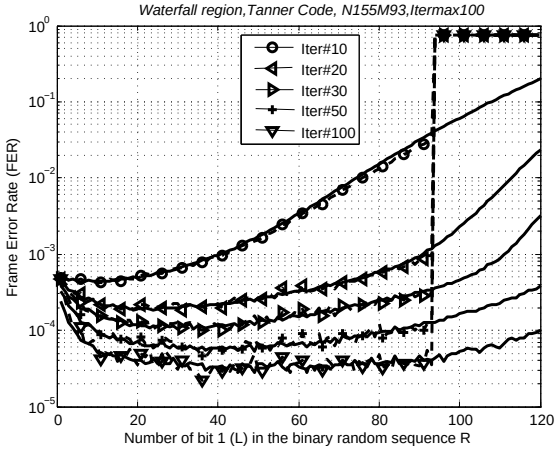


Figure 3.12: Frame Error Rate versus the number of 1's in $R^{(k)}$ in the case that only N' random VNs of the LPDC use randomness and the rest (N-N') still use the non-probabilistic GDBF (dashed lines). The solid lines represent the full allocation on N bits.

3.4 PGDBF with the Intrinsic-Valued Random Generator

3.4.1 Intrinsic-Valued Random Generator (IVRG)

An alternative solution to the Random Generator of PGDBF is presented in this section, which reduces the implementation cost of generating the random binary sequences $R^{(k)}$.

Our approach is based on using the CN values which are already stored inside the decoder to generate the $R^{(k)}$ sequence, and we named this approach *intrinsic-valued random generator* (IVRG). In an BF iterative decoder, the values of the CNs are computed at each iteration, and depend on the BSC crossover probability of α, the degree of check nodes d_c, the iteration number k and the structure of the Tanner graph $\mathcal{C}$ of the LDPC code (such as cycle, trapping sets, ...). Typically, the number of CN which are unsatisfied (value '1') is large during the first iterations, while it becomes smaller as the iteration number increases. We denote by $p(c^{(k)} = 1) = \mathcal{F}(\alpha, k, d_c, \mathcal{C})$ the probability that a CN c is unsatisfied at iteration k, as a function of the above mentioned parameters.

In the proposed IVRG, we will use only the CN values produced at the first iteration $k = 1$ in order to generate the sequences of random bits $R^{(k)}$. Before going into the details of the associated hardware implementation, we present the main statistical properties of the sequence $C^{(1)} =$

$\{c(j)^{(1)}, j = 1 \dots M\}$.

At the first iteration, and without any a priori information about the Tanner graph $\mathcal{C}$, the function $\mathcal{F}(\alpha, k, d_c)$ can be approximated by $\mathcal{F}(\alpha, 1, d_c) = \frac{1}{2} - \frac{1}{2}(1-2\alpha)^{d_c}$. With this approximation, the expected number of 1's in sequence $C^{(1)}$ corresponds to the expected number of unsatisfied CNs, equal to $L' = M * \left(\frac{1}{2} - \frac{1}{2}(1-2\alpha)^{d_c}\right)$. For large values of α (waterfall region), $\mathcal{F}(\alpha, 1, d_c)$ approaches $1/2$ and the expected number of 1's approaches $L' = M/2$. This value is sufficiently large so that the structure of the Tanner graph does not impact a lot on L'. Additionally, $L' = M/2$ always falls in the interesting range of L, i.e. offering the good performance of PGDBF over GDBF, as we analyzed in the previous section.

However, for small values of α (error floor region), the value of $\mathcal{F}(\alpha, k, d_c)$ and the corresponding number of 1's could be very small, and the structure of the graph could impact greatly on L'. In the rest of this section, we show that the bias in L', for small values of α, is actually in favor of the desired properties of the sequence $R^{(k)}$.

We will take into account the graphical structures, such as the cycles or the Trapping Sets (TSs). Trapping Sets $TS(a, b)$ were introduced by Richardson in [30], defined as a small set of a VNs for which the induced subgraph composed of the neighboring CNs, contains exactly b odd degree CNs. It is implied that in the induced subgraph of TS(a,b), there are 2 types of CNs which are *even* and *odd* degree. A cycle of length $2a$ is a particular trapping set TS$(a, a*(dv-1))$., and a weight-a codeword of the LDPC code is a particular trapping set TS$(a, 0)$. Based on these notations, we can bound the number L' of 1's in $C^{(1)}$ with the following theorem.

Theorem 3.1 *Given an LDPC code having only Trapping Set $TS(a, b)$ in its Tanner graph. If there are a erroneous bits in the whole codeword then L' is bounded as*

$$b \leq L' \leq \sum_{i}^{a} (d_{v(i)})$$

Proof Let a be the number of binary errors introduced by the BSC channel. One particular case is when these erroneous bits are located on one of the Trapping Set TS(a, b) of the LDPC code. Therefore, in this case, only the *odd* degree CNs of the TS are unsatisfied, and the corresponding value for L' is b. This case represents clearly the smallest value for L'.
The other extreme case corresponds to when the a erroneous bits do not share any common CN with the other incorrect bits. In this case, $d_{v(i)}$ CNs connected to the VN $v(i)$ are unsatisfied, where $d_{v(i)}$ is defined as the degree of the variable node. Therefore, there are $L'_{max} = \sum_{i}^{a} (d_{v(i)})$ 1's in $C^{(1)}$. In the general case, we finally have $b \leq L' \leq \sum_{i}^{a} (d_{v(i)})$.

Theorem 3.1 is not that helpful in this form, because since minimum codewords TS$(d_{min}, 0)$ are always present in the Tanner graph, it reduces to the trivial lower bound $L' \geq 0$. However, knowing the TS distribution of a particular LDPC code can provide more interesting results. Let us take the example of the Tanner Code, for which the distribution of the smallest TS is known, and reported in [31]. The TS distribution of the Tanner Code is indicated in table 3.2.

The following lemma gives more precise results on the minimum value of L' in the error floor region of the Tanner code.

Lemma 3.2 *If there are $E_r < 8$ erroneous bits in a received codeword the Tanner code, then $L'_{min} = 3$.*

Proof The proof of lemma 3.2 is trivial and follows from theorem 3.1 and the distribution of TSs indicated in table 3.2.

Table 3.2: Trapping Sets of Tanner Code (155,93)

Type of TS	Number of TS
TS(5,3)	155
TS(6,4)	930
TS(7,3)	930
TS(7,5)	13950

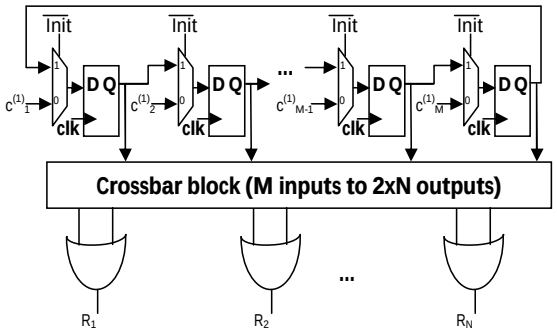


Figure 3.13: A proposed structure (called IVRG-F) of random generator that generates N binary values from M with gain factor $g = 2$.

The CNs values at iteration $k = 1$ generate the sequence $C^{(1)}$, which in turn will be transformed into the sequences $R^{(k)}$ (see next section). We have proved in this section that under mild conditions on the Tanner graph of the code, the minimum number of 1's in sequence $C^{(1)}$ is bounded away from 0. This is an interesting property since we have found in our analysis of section 3.3 that the PGDBF shows improvements compared to GDBF only if there is a sufficient number of 1's in the sequence $R^{(k)}$ (see figures 3.7 and 3.9). Using the IVRG to generate the sequences $R^{(k)}$ will then constructively follow this constraint and contribute to improve the PGDBF performance.

3.4.2 Full IVRG-PGDBF for general LDPC codes

In this section, we present a simple hardware solution to transform the sequence $C^{(1)}$ of size M bits into the sequences $R^{(k)}$ of size N bits.

In order to increase the number of 1's in $R^{(k)}$ of IVRG and guarantee L to fall in the useful range of values, we propose to use the logic gates properties as follows by designing a function $G(C^{(1)}, k)$ of the CN values at iteration 1 and a crossbar block changing from one iteration to the next one. The function $G(C^{(1)}, k)$, outputs N binary values from M inputs, and is designed to control the output probability of 1's in $R^{(k)}$. We briefly describe the function G in the following.

Let c_{j1} and c_{j2} be two binary random variables with $p(c_{j1} = 1) = p(c_{j2} = 1) = p$, it can be proven that $p(c_{j1}\mathbf{OR}c_{j2} = 1) = 2p + p^2 \approx 2p > p$ and $p(c_{j1}\mathbf{AND}c_{j2} = 1) = p^2 < p$. More specifically, $p(c_{j1} = 1) = p(c_{j2} = 1) = p = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{\mathbf{d_c}}$, $p(c_{j1}\mathbf{XOR}c_{j2} = 1) = \frac{1}{2} - \frac{1}{2}(1 - 2p_0)^{\mathbf{2d_c}}$. Using these transformations of probability, and the function $G(c^{(1)}, k)$ implemented as described in figure 3.13, we can transform the CN output sequence into a longer binary pseudo-random sequence with L falling into the desired range.

The CNs values at first iteration are stored in the chain of Flip-Flops and are cyclically shifted at each iteration and assigned to be the inputs of the OR gates through an interconnection network in order to ensure randomness of the output sequence. Using this implementation, the number of 1's in $R^{(k)}$ is approximately doubled compared to the number of 1's in $C^{(1)}$. The PGDBF using this implementation to produce the $R^{(k)}$ sequences will be called IVRG-F (F: Full) in the following. The

algorithmic description of the RG part only in F-IVRG-PGDBF is described in algorithm 2.

Algorithm 2 The IVRG part of the IVRG-F

if $k = 1$ **then**
  $R_j^t = c(j)^{(1)}$, $j = 1 \dots M$
end if
$R^{(k)} = G_F(R^t, k), \quad |G_F(R^t, k)| = N;$

3.4.3 Partial IVRG-PGDBF for QC-LDPC codes

In this section, we will further reduce the complexity of the PGDBF implementation by using the partial random sequence allocation that was presented in section 3.3.3. This method of using only N' out of the N random bits of $R^{(k)}$ is expecially interesting for Quasi-cyclic LDPC codes, and finds its real advantage in terms of hardware implementation when $N' = M$, since it would avoid the use of the function $G(C^{(1)}, k)$ presented in the previous section.

Quasi-Cyclic LDPC codes have a structured parity-check matrix H generated with a $M_b \times N_b$ base matrix expansion. The expansion is performed by replacing each 1 in the base matrix by a $Z \times Z$ circulant matrix (circularly shifted identity matrix), leading to a $(M = M_b Z, N = N_b Z)$ parity check matrix.

We propose here to restrict the use of PGDBF to only M out of the N codeword bits, and use the deterministic GDBF for the rest of the bits. The essential feature of this method lies on avoiding the use of the function $G(C^{(1)}, k)$ which may increase the decoder complexity. This idea is motivated by the statistical analysis results shown in figure 3.12 which predicts that the PGDBF performance in this partial case perfectly match with of full random case. The M bits will be directly taken out of the sequence $C^{(1)}$, and allocated to M_b blocks of Z consecutive columns of the parity check matrix. We call this modified algorithm partial IVRG-PGDBF (P-IVRG-PGDBF). The allocation of the M_b blocks is randomly chosen at each iteration.

Algorithm 3 The IVRG part of the IVRG-P

if $k = 1$ **then**
  $R_j^t = c(j)^{(1)}$, $j = 1 \dots M$
end if
$r = arg\ index_{j \in [1,N]} r^{\frac{M}{Z}}(v^{(k)})$
$R_r^{(k)} = G_P(R^t, k), \quad |G_P(R^t, k)| = M;$
$R_{[1,N]\setminus r}^{(k)} = \underline{0}$
// where $r^{\frac{M}{Z}}(v^{(k)})$ returns all VNs of (M/Z) random blocks out of (N/Z) of $v^{(k)}$ in QC-LDPC, $|r| = M$.

As described in algorithm 3, we randomly choose $M_b = M/Z$ blocks of VNs (M VNs) and assign their random inputs to the M bits generated by the IVRG output $C^{(1)}$. The non-selected VNs still use a non-probabilistic GDBF. From the hardware complexity point of view, the P-IVRG-PGDBF has very low complexity, close to the GDBF implementation. The details on the hardware synthesis results will be reported in future works.

3.4.4 IVRG-PGDBF Performance

The PGDBF decoders with all the RGs proposed in this chapter outperform the GDBF as shown in figure 3.14 for the Tanner code. The performance of the same decoders for a longer quasi-cyclic LDPC code, with parameters $(d_v = 4, d_c = 8)$ and $N = 1296$, are shown on figure 3.15. In all algorithms, a maximum os 100 iterations have been used. As we can see on these curves, the use of the RGs improves significantly the error correction performance of the decoder, as was already reported in [28] and [29]. The simplified implementations of the PGDBF proposed in this chapter, F-IVRG-PGDBF

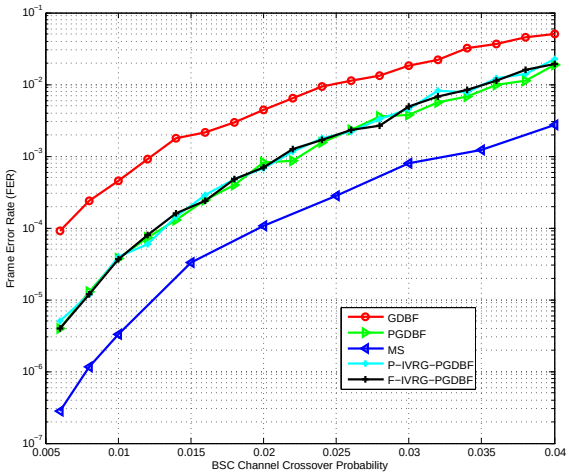


Figure 3.14: Comparison of decoders performance on the Tanner code. PGDBF, F-IVRG-PGDBF and P-IVRG-PGDBF significantly improve decoding performance compared to GDBF.

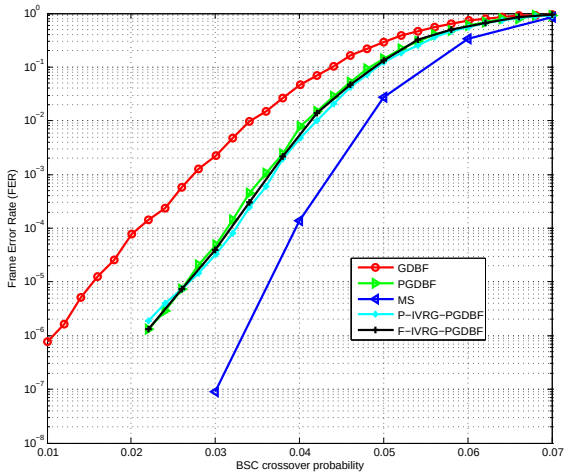


Figure 3.15: Comparison of decoders performance on a quasi-cyclic $(d_v = 4, d_c = 8)$ LDPC code, with codeword length $N = 1296$.

and P-IVRG-PGDBF are very efficient as they do not loose any performance compared to PGDBF, while their computational complexity is greatly reduced. There is still a gap between the Min-Sum decoders and the PGDBF decoders, that is mainly due to the fact that PGDBF converges more slowly than the Min-Sum. Future comparison and improvements of the PGDBF will focus on reducing this performance gap.

3.5 Conclusion

In this chapter, we introduced a new and original method of random generation in PGDBF decoders which provides an highly efficient hardware implementation compared to other conventional method while preserving the outstanding decoding performance of PGDBF. This new random generator method called Intrinsic-valued random generator avoids generating the required random bits but use from the existing decoder memory which is the source of the simplicity and hardware efficiency. Moreover, the generated random sequence of this new method was proven to constructively satisfy the requirements for a good decoding performance of PGDBF, which were characterized by a thorough statistical analysis.

Chapter 4

Reliable LDPC Encoders built from Unreliable Gates

Abstract: *LDPC codes on faulty hardware have been widely investigated recently. While, as a main result, it was shown that LDPC decoders are naturally robust to hardware noise, we also know that most of the standard LDPC encoding techniques completely fail under hardware noise. In this work, we address the problem of constructing LDPC encoders robust to faulty hardware. The encoding solution we propose consists of computing an augmented codeword that contains both the codeword to be transmitted on the channel and extra parity bits. Before transmission on the channel, an LDPC decoder is applied to the augmented codeword in order to eliminate the hardware errors from the channel codeword. The augmented codeword is obtained from a split-extended construction that guarantees good decoding performance both for the codeword transmitted on the channel and for the augmented codeword. From Monte Carlo simulations, we show the robustness of the proposed encoding solution for various codes and decoders.*

4.1 Introduction

Over the past few years, reliability has become a major issue in the design of electronic devices. A huge increase in the integration factors coupled with important chip size reduction will make the next generations of electronic devices much more sensitive to noise [13]. As a consequence, in the future systems of communication and storage, errors may not only come from the transmission channels, but also from the faulty hardware. In this context, there is a need to evaluate the robustness of Low Density Parity Check (LDPC) encoders and decoders running on faulty hardware.

The robustness of LDPC decoders was widely investigated for a large range of decoders. The performance of hard decoders under faulty hardware was analyzed in [4] (Gallager A) and [6] (Gallager B), while soft decoders were considered in [4] (Belief Propagation), and [18] (quantized Min-Sum). As a result, we know that if the decoder parameters (number of quantization levels, etc.) are carefully chosen, most of the above LDPC decoders are naturally robust to hardware noise, with no need for aditional circuitry. Following this idea, [19] proposed to design Finite Alphabet Iterative Decoders (FAIDs) strongly robust to hardware noise.

On the other hand, the problem of LDPC encoding has been considered only very recently. In [32], Hachem et al. evaluated from an information theoretic perspective the level of hardware noise that can be tolerated in the encoder. However, the results of [32] do not indicate how to construct a practical encoder robust to hardware noise. From a practical point of view, [33] shows that most of the standard encoding solutions (Systematic, lower triangular, encoding for Zig-Zag and Quasy-Cyclic codes, etc.) completely fail when some errors are introduced by the faulty hardware.

The objective of this chapter is to propose a practical LDPC encoding solution robust to faulty hardware. A robust LDPC encoding solution was proposed in [34] that consists of embedding several LDPC decoders inside the encoder, in order to correct gradually the hardware errors. In [35], the same authors also proposed to compute additional parity bits in order to better protect the encoder.

However, the solutions proposed in [34,35] are very computationally expensive, due to multiple decodings. Furthermore, only theoretical analysis of the proposed encoding solutions are performed, and no practical implementation or code construction is considered.

The encoding solution we propose consists, as in [35], of computing extra parity bits that will be used in order to correct the hardware errors. We will then rely on the inherent robustness capabilities of LDPC decoders to correct the hardware errors from the parity bits. In this chapter, we use only one LDPC decoder at the end of the encoding stage, and we construct the extra parity bits from split-extended LDPC codes introduced in [36]. We then present experimental results that evaluate the complexity of the proposed encoder and show its robustness to hardware noise.

The outline of the chapter is as follows. Section 4.2 introduces our notations for LDPC codes and the error model we consider to represent the faulty hardware effect on the encoder. Section 4.3 presents the robust LDPC encoding solution. Section 4.4 shows the experimental results.

4.2 LDPC codes and Error Models

In this section, we first introduce our notations for LDPC codes. We then describe the XOR gate error model that represents the faulty hardware effect on the encoder.

4.2.1 LDPC Codes

Denote by H a binary parity check matrix of size $n \times m$. An LDPC code is defined as the null-space of the parity check matrix H. A binary vector $\mathbf{x}$ of length n is a codeword if and only if it verifies

$$H^T \mathbf{x} = \mathbf{0}. \tag{4.1}$$

With LDPC codes, the parity check matrix H is sparse and has to be designed in order to obtain good decoding performance, see [37] for instance. Once H is fixed, the corresponding encoder has to be constructed.

Denote by $\mathbf{u}$ the information sequence of length $k = n - m$. The encoding operation we denote

$$\mathbf{x} = \mathcal{E}(\mathbf{u}) \tag{4.2}$$

has to transform the information sequence $\mathbf{u}$ into a codeword $\mathbf{x}$ that satisfies (4.1). Here, we will consider systematic encoding, for which the codeword $\mathbf{x} = [\mathbf{u}, \mathbf{p}]^{\mathrm{T}}$ contains both the information sequence $\mathbf{u}$ and m parity bits given by $\mathbf{p}$.

A first solution to perform the encoding is to construct a generator matrix G of size $(n \times k)$ that verifies $H^{\mathrm{T}}G = 0$. The generator matrix G can be obtained from H by Gaussian elimination, as described in [37]. The encoding operation is then given by

$$\mathbf{x} = G\mathbf{u}. \tag{4.3}$$

The matrix G is not sparse in general, and the operation (4.3) can be very complex. Several other encoding solutions were proposed in order to reduce the encoding complexity. These solutions are either general, in the sense that they apply to any parity check matrix H [37], or specific to particular code constructions, such as Zig-Zag [38], Quasy-Cyclic (QC) [39], or Low Density Generator Matrix (LDGM) codes [40]. Before discussing the robustness of the encoding solutions [37–40] to hardware errors, we first present the error model we consider to represent the faulty hardware effect on the encoder.

4.2.2 XOR Gate Error Model

The existing encoding solutions [37–40] can be realized from XOR gates only. Consequently, we assume that hardware errors are introduced at the XOR gate level. Denote by p_{xor} the error probability of a 2-inputs XOR gate. The faulty 2-inputs XOR operator $\tilde{\oplus}$ is defined as

$$a \,\tilde{\oplus}\, b = \begin{cases} a \oplus b & \text{with prob. } 1 - p_{\mathrm{xor}}, \\ 1 \oplus (a \oplus b) & \text{with prob. } p_{\mathrm{xor}}, \end{cases} \tag{4.4}$$

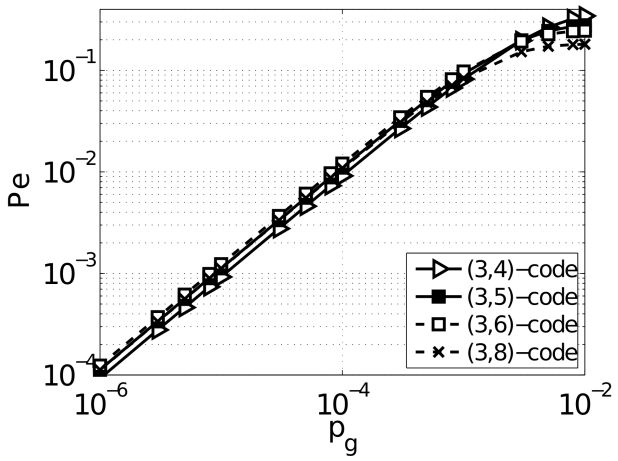


Figure 4.1: Encoding error probability P_e with respect to gate error probability p_g. In the legend, the $(3, x)$-code represents the code with $d_v = 3$ and $d_c = x$

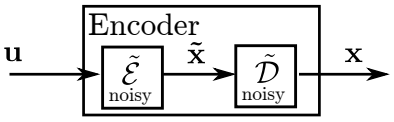


Figure 4.2: First encoding solution

where a and b are binary digits and $a \oplus b$ is the (perfect) XOR sum of a and b. The error model described in (4.4) is memoryless and data-independent. It is considered here as a first step of the analysis, and more accurate models will be considered in future works. At the end, the noisy encoding operation realized from faulty XOR gates is denoted

$$\tilde{\mathbf{x}} = \tilde{\mathcal{E}}(\mathbf{u}) \tag{4.5}$$

where $\tilde{\mathbf{x}}$ is the vector resulting from noisy encoding. Note that due to encoding errors, $\tilde{\mathbf{x}}$ is not necessarily a codeword.

Under the XOR gate error model, it was shown in [33] that almost all the existing encoding solutions [37–40] completely fail under hardware errors. Only encoding for LDGM codes is robust, but LDGM codes have poor decoding performance compared to LDPC codes. Therefore, in the following, we propose an LDPC encoding solution robust to faulty hardware.

4.3 Reliable LDPC encoder

Consider the systematic encoding operation described by (4.3). Fig. 4.1 represents the encoding error probability P_e with respect to the gate error probability p_g for various LDPC codes, with $k = 1000$, $d_v = 3$ and $d_c = 4, 5, 6, 8$, respectively. We observe that the encoding error probability does not depend much on the coding rate. We also see that the encoding error probability is dramatically increased with respect to the gate error probability p_g. For example, a gate error probability $p_g = 10^{-4}$ will give an encoding error probability $P_e = 10^{-2}$, which represents an increase by a factor 100. Furthermore, the encoding error probability is going to combine with the channel noise, and at the end, the decoder will not be able to recover the original information sequence $\mathbf{u}$ from the channel output. Hence there is a need to drastically reduce the encoding error probability before transmission on the channel. For this, a first solution consists of applying a decoder to $\tilde{\mathbf{x}}$ before channel transmission, as we now describe.

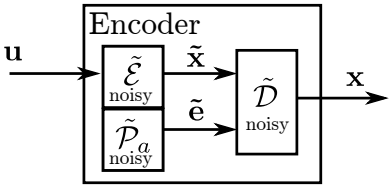


Figure 4.3: The CPE approach

4.3.1 Decoder at the Encoder

As described in Fig. 4.2, a first solution for reliable encoding consists of passing the noisy codeword $\tilde{\mathbf{x}}$ through an LDPC decoder $\tilde{\mathcal{D}}$ before channel transmission, in order to eliminate the encoding errors. As the decoder $\tilde{\mathcal{D}}$ runs on the same hardware as the encoder $\tilde{\mathcal{E}}$, is is assumed noisy as well. As the noise model in a decoder usually depends on the type of decoder, we will specify the decoder error model when we need it in the experimental results.

The LDPC decoder $\tilde{\mathcal{D}}$ will be efficient only if the gate error probability p_g leads to an encoding error probability lower than the correction capability of the LDPC code. For example, from [41], if $P_e > 0.084$, a strong Belief Propagation (BP) decoder will not be able to correct the hardware errors for a $(3, 6)$-code. An error probability $P_e > 0.084$ happens for a value of p_g approximately lower than 10^{-3}. Here, in order to be able to deal with higher values of p_g, we would like to go further than the correction capability of the LDPC code, and we thus propose the following Codeword Prediction Encoder (CPE) approach.

4.3.2 The CPE Approach

As described in Fig 4.3, in addition to the parity bits $\mathbf{p}$ contained in $\mathbf{x}$, we now compute m_a extra parity bits $\mathbf{e}$ from $\mathbf{u}$, as

$$\mathbf{e} = \tilde{\mathcal{P}}_a(\mathbf{u}), \tag{4.6}$$

where $\tilde{\mathcal{P}}_a$ is noisy according to the XOR gate error model. The vector $\mathbf{x}_a = [\mathbf{x}, \mathbf{e}]^T$ is called the augmented codeword. Before channel transmission, the noisy $\tilde{\mathbf{x}}_a$ is passed through a noisy LDPC decoder $\mathcal{D}$ in order to eliminate the encoding errors. The extra parity bits $\mathbf{e}$ serve only to help eliminate the encoding errors, and, after decoding, only $\mathbf{x}$ is transmitted through the channel. Thus, both the LDPC code that produces $\mathbf{x}$ and the one that produces $\mathbf{x}_a$ have to lead to good decoding performance.

The LDPC decoder used at the encoding side makes use of both the original parity bits $\mathbf{p}$ and of the extra parity bits $\mathbf{e}$ to eliminate the encoding errors. If the LDPC codes are constructed carefully, this strategy will result in increased correction capabilities for the augmented codeword $\mathbf{x}_a$ compared to the initial codeword $\mathbf{x}$. For good code construction, an important condition is that the extra parity bits $\mathbf{e}$ are independent of the original parity bits $\mathbf{p}$, which means that $\mathbf{p}$ and $\mathbf{e}$ are computed from different combinations of bits from $\mathbf{u}$. To guarantee the independence while insuring good decoding performance for both $\mathbf{x}_a$ and $\mathbf{x}$, we consider Split-Extended codes introduced in [36], as we now describe.

4.3.3 Code Construction

The basic idea of the split-extend design can be resumed as follows. Let H be the parity-check matrix of the original LDPC code. Hence, the original codeword – constituted of both information and parity bits – satisfies all the parity-check equations defined by the rows of H. Extra parity bits can be computed by splitting these parity-checks, as illustrated in Figure 4.4. The parity-check in the middle corresponds to a row of H. In the left example, a new parity bit e_1 is created by spitting the original parity-check into two sub-checks. Precisely, this means that the set of bits connected to the check-node is partitioned into two subsets, and the parity bit e_1 is generated as the sum of the bits of either one of the two subsets. In the right example, two new parity bits e_1 and e_2 are created by

spitting the original parity-check into three parity-checks. Precisely, the set of bits connected to the check-node is partitioned into three subsets, and e_1 is generated as the sum of the bits in the first subset. Subsequently, e_2 can be generated either as the sum of e_1 and the bits in the second subset, or as the sum of the bits in the third subset. The total number of extra parity bits depends on the number of rows of H and the number of extra bits generated for each row of H (which may vary from one row to another). The matrix H_{ext} obtained by the split-extension of H (i.e. the incidence matrix of the split-extended graph) verifies $H_{\text{ext}}[\mathbf{x}, \mathbf{e}]^T = 0$, where $\mathbf{x}$ denotes the original codeword and $\mathbf{e} = [e_1, e_2, \dots]^T$ is the sequence of all the extra parity bits. Therefore, H_{ext} can be used in order to jointly decode a noisy version of the extended codeword $X_{\text{ext}} = [\mathbf{x}, \mathbf{e}]^T$. More details on the split-extended construction can be found in [36].

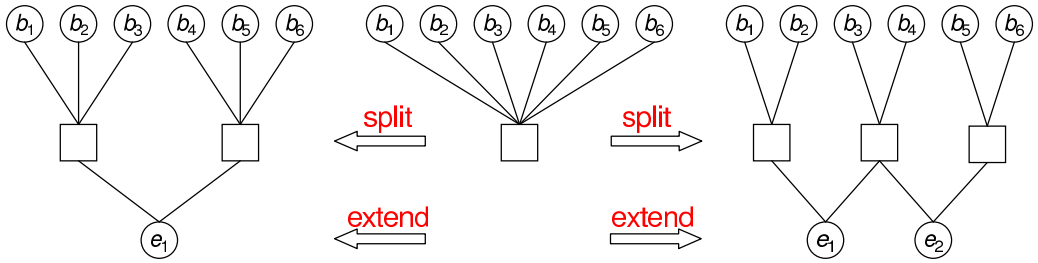


Figure 4.4: Split-Extension examples

The split-extended construction has been applied to the i-RISC QC-LDPC codes with $d_v = 3$ and $d_v = 4$, and coding rates $r = 1/2$ and $r = 2/3$. Each row of the original base matrix has been split into two rows (see Figure 4.4, left example), such that the number of extra parity bits generated by the split-extend construction is equal to the number of original parity bits. Splitting has been performed by a dedicated algorithm that searches for short cycles in the parity check matrix, then splits rows such that to break as many short cycles as possible. The base matrix of the i-RISC QC-LDPC code with $(d_v = 3, r = 1/2)$ and the corresponding split-extended base matrix are shown in Figure 4.5 (-1's entries of the base matrix are represented by a dash sign).

Split-extended matrices can be used to address the fault-tolerant encoding issue as follows: at the transmitter side, both the original codeword $\mathbf{x}$ and the sequence of extra parity-bits $\mathbf{e}$ are computed, using error-prone encoding circuitry. A (possibly error-prone) LDPC decoder, is used to decode the computation errors on the extended codeword $X_{\text{ext}} = [\mathbf{x}, \mathbf{e}]^T$, and then the original codeword $\mathbf{x}$ is sent over the channel. The use of the extra parity bits E allows increasing the error correction capacity, making the overall encoder design more robust to hardware faults. At the receiver end, only the original parity check matrix H is needed, in order to decode the received signal.

The performance of the proposed CPE approach will depend on the encoding technique we consider to construct the functions $\mathcal{E}$ and $\mathcal{P}_a$. If the considered encoding technique is iterative, the errors introduced by the hardware will propagate, which will result in very high error probability [33] at the output of $\tilde{\mathcal{E}}$ and $\tilde{\mathcal{P}}_a$. Some particular XOR gates of $\tilde{\mathcal{E}}$ or $\tilde{\mathcal{P}}_a$ may be critical, in the sense that injecting only one error at the output of such gates may result in a very large number of errors at the output of the circuit. In order to overcome this issue, one can choose to identify and protect the critical XOR gates used in the encoder, as described in the following.

4.3.4 Individual Gate Protection

A XOR gate is critical when the output of the gate is propagated to a large number of outputs of the circuit. Such error event is characterized by a very high number of errors on the output of the circuit and causes decoding failures with very high probability. The critical gates can be identified as follows.

The criticality degree of a gate G, denoted by $\text{cdeg}(G)$, is defined as the number of outputs to which G is connected by at least one path. Thus, injecting an error in node G, may produce at most $\text{cdeg}(G)$ errors on the output. In order to protect the critical XOR gates, we fix a criticality threshold CT. Gates G with $\text{cdeg}(G) > \text{CT}$ are considered to be "protected" (e.g. by increasing area), so as to make then reliable. By convention, setting $\text{CT} = -1$, means that all gates are unreliable.

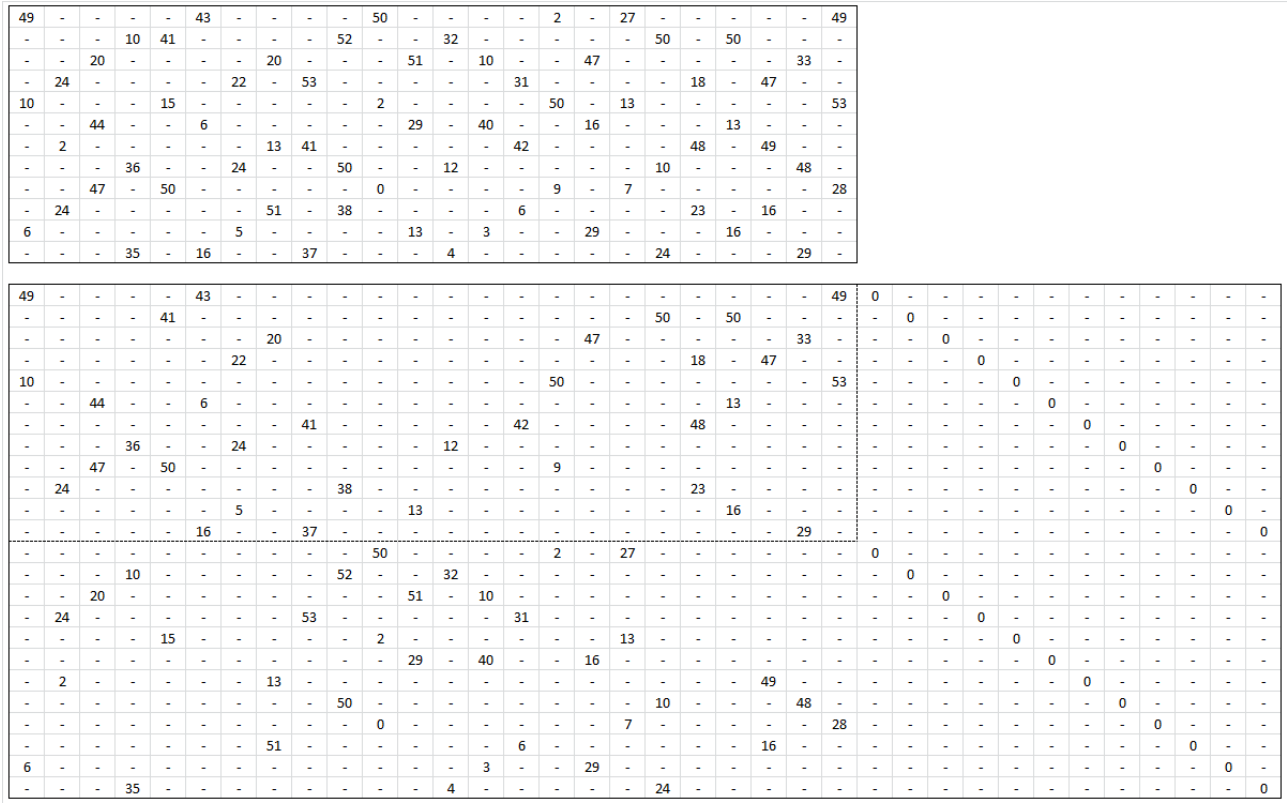


Figure 4.5: Base matrix of the i-RISC QC-LDPC code with $d_v = 3$ and $r = 1/2$ (top), and corresponding split-extended base matrix (bottom)

At the end, the performance of the CPE approach will depend on the choice of the code (rate, degrees, etc.) and of the considered LDPC decoder. In the following, we present the simulations results we obtained for different codes and decoders.

4.4 Experimental Results

In this section we evaluate the reliablity of encoding from CPE for the four iRisc codes iRISC_dv3_r_12, iRISC_dv3_r_34, iRISC_dv4_r_12, iRISC_dv4_r_34. The extra parity bits are obtained from the Split-Extended construction and were consider three different faulty decoders that are Gallager B, Min-Sum, and Self-Corrected Min-Sum. For faulty decoders, we assume that the output of every variable and check node function computation is flipped with a probability $p = 10^{-3}$. For non-binary message alphabets, flipping the output value means that a value different from the correct one is selected uniformly at random from the alphabet.

In the following, we evaluate the performance of the CPE approach in terms of complexity and of error probability.

4.4.1 Complexity Analysis

Consider the encoding operation from the generator matrix (4.3) The generator matrices of the codes can be constructed from the parity check matrices H_{ext} by Gaussian Elimination [37]. In order to reduce the complexity of the encoding operation, we apply a circuit design tool to each generator matrix, which outputs the factorized encoding functions $\mathcal{E}$ and $\mathcal{P}_a$ we use in the simulations.

Table 4.1 shows the number of XOR gates needed by the encoders for the four iRisc codes. It compares the number of XOR gates for encoding from the generator matrix and for encoding from the factorized functions. We see that the factorized functions have much lower complexity.

Table 4.1: Number of XOR gates for generator matrices and for factorized functions

Code	Generator matrix	Factorized function
dv3-r12	296734	44399
dv3-r34	170362	28182
dv4-r12	300205	45175
dv4-r34	303163	27167

Table 4.2: Critical Gate count for different encoding schemes

Encoder	G_{CPE} Node Count	CT=10	CT=20	CT=50
dv3-r12	44399	3373	1844	833
dv3-r34	28182	2288	1240	537
dv4-r12	45175	3424	1851	824
dv4-r34	27167	2112	1183	488

Unfortunately, factorization will also result in error propagation and the critical gates have to be protected. Table 4.2 represents the number of gates to be protected with respect to criticality degree. We see that in each of the considered cases, the number of gates to protect is very small compared to the total number of gates in the circuit. Figure 4.6 also shows the Bit Error Performance (BER) with respect to p_g for the iRISC_dv3_r_34 code with a noisy min-sum decoder, for various CT values. As expected, a reduced value of CT improves the reliability. For the following error probability evaluation of the CPE, we set CT=10.

4.4.2 Error Probability Analysis

For the BER performance evaluation of the CPE approach, we consider the iRISC_dv4_r_12 and the three different noisy decoders. Figure 4.7 represents the BER performance with respect to p_g for the three decoders. We see that there is a significant gain at considering CPE compared to the unprotected case, in particular for the two Min-Sum decoders. In particular, with the Min-Sum decoders, the CPE approach gives a very robust encoder for p_g values up to 10^{-3}. Increasing the value of CT may also enable to get a robust encoder for higher values of CT.

4.5 Conclusion

In this chapter, we proposed an LDPC encoding solution robust to hardware noise. The proposed CPE encoding solution consists of computing extra parity bits and of applying an LDPC decoder before transmitting the codeword on the channel. The extra parity bits used in CPE were obtained from a split-extended code construction. Simulation results show that the proposed CPE approach gives a very robust encoding, with a BER reduced by a factor 100 compared to encoding without CPE.

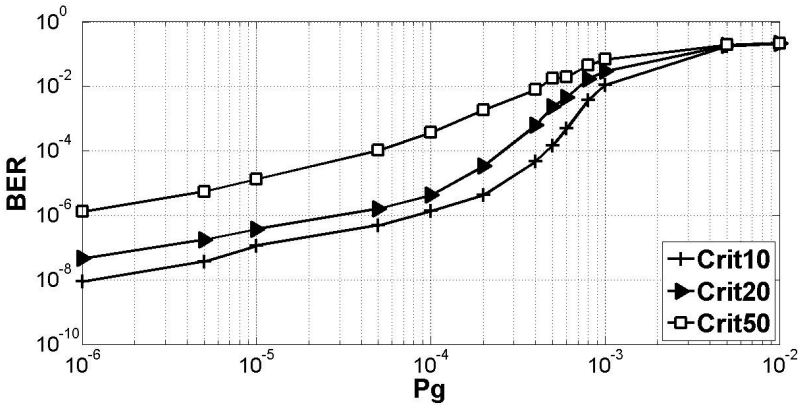


Figure 4.6: Critical Threshold impact on Output BER

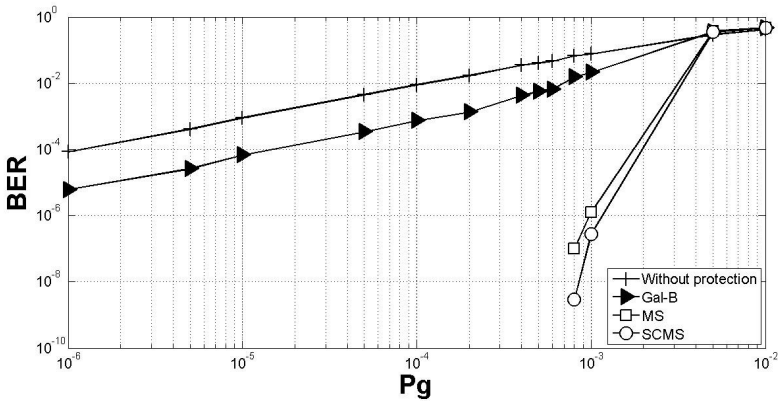


Figure 4.7: CPE error free scenario employing faulty decoder

General Conclusions of WP3

As a general conclusion of the activities in WP3, we have made considerable progress with respect to the initial objectives of the project and to the state-of-the-art related to the design and analysis of faulty iterative decoders. The initial objectives of WP3 were

- Objective 1: Analysis of asymptotical performance of LDPC decoders under faulty hardware,
- Objective 2: Design of fault-tolerant iterative decoders,
- Objective 3: Design of fault-tolerant LDPC encoders.

The initial objective "Design of fault-tolerant Reed-Muller and Polar codes" was replaced by

- Objective 4: Design of low-complexity fault-tolerant LDPC decoders.

Regarding these objectives, we have achieved the following.

Concerning **Objective 1**, we have analyzed the asymptotical performance under faulty hardware of two families of soft decoders with finite alphabets, that are Min-Sum (MS) and Finite Alphabet Iterative Decoders (FAIDs). As a first step, we have performed the asymptotical analysis under very simple memoryless, data-independent, symmetric error models. As a tool for the asymptotical analysis, we have introduced the functional threshold to characterize the asymptotic behavior of noisy decoders. The detailed conclusions of the analysis are as follows.

- MS and FAIDs are implemented with messages in finite precision, while the state of the art usually considers either hard decision decoders (bit-flipping and Gallager-B decoders), or non-realistic infinite precision Belief Propagation. Although we have considered very limited error models, which are far from being realistic, our models are broader and more general than what is usually proposed in the litterature.
- We have progressed toward a deep understanding of the dynamics of faulty decoders, with in particular the new concept of functional threshold, to replace the existing useful region and the Target-BER threshold. We have shown that under restricted noise conditions, the functional threshold predicts the asymptotic behavior of noisy decoders. We have also provided evidence of the compliance between the asymptotical noisy DE analysis and the finite length performance of noisy decoders, so that the noisy DE analysis can be used to predict the finite-length performance of faulty decoders.
- We have identified that the noise can help iterative decoder to avoid unwanted fixed points. This was the case for example for the DE fixed points of the noisy MS decoder.

Concerning **Objective 2**, we first designed fault-tolerant MS, FAIDs, and Stochastic decoders under very simple memoryless, data-independent, symmetric error models, as described in the following.

- Through Monte-Carlo simulations, we have shown first evidence that using memory in faulty decoders can greatly improve their fault-tolerance. In particular, the Self-Corrected MS has been shown to tolerate much more hardware noise compared to the original MS decoder.

- We have shown through Monte-Carlo simulations that the Stochastic decoder is naturally robust to hardware errors. We have also shown that the stochastic decoders presents increased robustness to hardware noise compared to the MS decoder.

- While most of the current State of the Art only settle for the performance analysis of existing LDPC decoders under faulty hardware, we went further and proposed a method for the design of strongly robust decoders. The design method we proposed is based on the asymptotical analysis of FAIDs and in particular on the functional threshold definition we introduced. Monte-Carlo simulations have validated the proposed design methodology, showing that the designed decoders are indeed more robust to hardware errors compared to non-optimized decoders.

In order to validate the above approaches for more realistic error models, we have also analyzed the performance of Gallager B decoders under timming errors, and of FAIDs under error models with memory.

- We characterized the effect of timing errors on the performance of the Gallager B decoder. By using Monte Carlo simulations, we identified two operating regions - one in which hardware unreliability leads to significant performance degradation, and one in which the performance loss is negligible. Based on these results, we proposed a simple modification of the decoder that ensures its fault-tolerance under timing error models.

- We characterized the performance of FAIDs under error models with memory. We reused the FAIDs we previously designed for robustness under simple memoryless error models. Through Monte-Carlo simulations, we measured the performance of these FAIDs under error models with memory. We observed that the FAIDs designed for robustness in the memoryless case are still robust under error models with memory, which validated the approach we developed with memoryless models.

At the end, the contributions realized within Objectives 1 and 2 fully complete Task 3.1 – MS/FAID decoders under faulty gates and Task 3.2 – Stochastic decoder under faulty gates.

Concerning **Objective 3**, while only a few works in the literature consider LDPC encoders on faulty hardware, we have analyzed the performance of existing LDPC encoding solutions under hardware errors, and proposed an LDPC encoding solution robust to hardware errors.

- We have reviewed several encoding solutions and analyzed their robustness to hardware errors. We have observed that most of them are non-robust to hardware errors.

- We have introduced an LDPC encoding technique robust to hardware errors. The LDPC encoding technique we proposed is based on the Codeword Prediction Approach (CPE) introduced in WP5. From Monte Carlo simulations, we showed the robustness of the proposed encoding solution for various codes and decoders.

The contributions proposed with respect to Objective 3 fully complete Task 3.4 – Practical fault-tolerant encoding.

Concerning **Objective 4**,

- We have proposed a new bit-flipping algorithm called PGDBF, which introduces randomness in the bit-flipping decision. We have seen that the PGDBF algorithm has better performance in the noiseless case than other existing bit-flipping algorithms. We have also observed that PGDBF is robust to hardware errors and, more surprisingly, that hardware errors can even improve the decoders performance.

- We have proposed two implementations of the PGDBF algorithm and have shown that both implementations improve greatly the error correction performance compared to the non-probabilistic version.

- We introduced a new and original method for the random bit generation in PGDBF decoders. The introduced Intrinsic-valued random generator (IVRG) provides a highly efficient hardware implementation compared to other conventional method while preserving the outstanding decoding performance of PGDBF. IVRG avoids generating the required random bits but use from the existing decoder memory which is the source of the simplicity and hardware efficiency. We provided a statistical analysis of the behavior of the PGDBF and identified the important features of the random generators that permit to obtain large coding gains compared to the deterministic GDBF. The statistical analysis showed that the proposed implementation only induces a very small extra complexity, without any loss in performance for the PGDBF decoder.

The contributions proposed with respect to Objective 4 fully complete Task 3.5 – Randomized Bit-Flipping decoders.

Bibliography

[1] S. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders–Part I: decoding beyond belief propagation on the binary symmetric channel," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

[2] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.

[3] B. Vasic and S. K. Chilappagari, "An information theoretical framework for analysis and design of nanoscale fault-tolerant memories based on low-density parity-check codes," *IEEE Transactions on Circuits and Systems I, Regular Papers*, vol. 54, no. 11, pp. 2438–2446, Nov. 2007.

[4] L. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, July 2011.

[5] S. M. Yazdi, H. Cho, and L. Dolecek, "Gallager B decoder on noisy hardware," *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1660–1673, May 2013.

[6] C. H. Huang, Y. Li, and L. Dolecek, "Gallager B LDPC decoder with transient and permanent errors," *IEEE Transactions on Communications*, vol. 62, no. 1, pp. 15–28, Jan. 2014.

[7] F. Leduc-Primeau and W. Gross, "Faulty Gallager-B decoding with optimal message repetition," in *Proceedings of 50th Allerton Conference on Communication, Control, and Computing*, Monticello, USA, Oct. 2012, pp. 549–556.

[8] S. Brkic, P. Ivanis, and B. Vasic, "Analysis of one-step majority logic decoding under correlated data-dependent gate failures," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, Honolulu, USA, June 2014, pp. 2599–2603.

[9] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, C.E. Shannon and J. McCarty, eds., Princeton Univ. Press, July 1956, pp. 43–98.

[10] J. Chen, C. Spagnol, S. Grandhi, E. Popovici, S. Cotofana, and A. Amaricai, "Linear compositional delay model for the timing analysis of sub-powered combinational circuits," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, July 2014.

[11] A. Amaricai, S. Nimara, O. Boncalo, J. Chen, and E. Popovici, "Probabilistic gate level fault modeling for near and sub-threshold cmos circuits," in *Proceedings of 17th Euromicro Conference on Digital System Design (DSD)*, Verona, Avg. 2014, pp. 473–479.

[12] I. Perez-Andrade, X. Zuo, R. Maunder, B. Al-Hashimi, and L. Hanzo, "Analysis of voltage- and clock-scaling-induced timing errors in stochastic LDPC decoders," in *Proceedings IEEE Wireless Communications and Networking Conference (WCNC)*, Shanghai, Apr. 2013, pp. 4293–4298.

[13] S. Zaynoun, M. S. Khairy, A. M. Eltawil, F. J. Kurdahi, and A. Khajeh, "Fast error aware model for arithmetic and logic circuits," in *Proceedings of 30th IEEE International Conference on Computer Design (ICCD)*, Montreal, QC, Sept. 2012, pp. 322–328.

[14] E. Janulewicz and A. Banihashemi, "Performance analysis of iterative decoding algorithms with memory over memoryless channels," *IEEE Transactions on Communications*, vol. 60, no. 12, pp. 3556–3566, 2012.

[15] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, "On the construction of structured LDPC codes free of small trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr. 2012.

[16] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2012.

[17] C. H. Huang and L. Dolecek, "Analysis of finite alphabet iterative decoders under processing errors," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, Candada, May 2013, pp. 5085–5089.

[18] C. K. Ngassa, V. Savin, E. Dupraz, and D. Declercq, "Density evolution and functional threshold for the noisy Min-Sum decoder," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1497–1509, May 2015.

[19] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Finite alphabet iterative decoders robust to faulty hardware: Analysis and selection," in *8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2014, pp. 107–111.

[20] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[21] D. Declercq, M. Fossorier, and E. Biglieri, *Channel Coding: Theory, Algorithms, and Applications*. Elsevier, 2014.

[22] N. Miladinovic and M. Fossorier, "Improved Bit-Flipping decoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1594–1606, April 2005.

[23] 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, 2009.

[24] 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, 2005.

[25] 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.

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

[27] T. Ngatched, F. Takawira, and M. Bossert, "A modified bit-flipping decoding algorithm for low-density parity-check codes," in *Proceedings of IEEE International Conference on Communications*, 2007, pp. 24–28.

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

[29] K. Le, D. Declercq, C. Spagnol, P. Ivanis, and B. Vasic, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *Proceedings of IEEE International Conference on Electronics Circuits and Systems (ISCAS)*, 2015, pp. 1494 – 1497.

[30] T. Richardson and R. Urbanke, "Error floors of LDPC codes," in *Proceedings of 41th Annual Allerton Conf on Communications Control and Computing*, 2003, pp. 1426–1435.

[31] D. Declercq, B. Vasic, S. 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, 2013.

[32] J. Hachem, I. Wang, C. Fragouli, S. Diggavi *et al.*, "Coding with encoding uncertainty," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, 2013, pp. 276–280.

[33] E. Dupraz and D. Declercq, "Evaluation of the robustness of LDPC encoders to hardware noise," in *Proccedings of IEEE BlackSeaCom conference*, 2015, pp. 1–5.

[34] Y. Yang, P. Grover, and S. Kar, "Can a noisy encoder be used to communicate reliably?" in *Proceedings of 52nd Annual Allerton Conference on Communication, Control, and Computing*, 2014, pp. 659–666.

[35] ——, "Computing linear transformations with unreliable components," *arXiv preprint arXiv:1506.07234*, 2015.

[36] V. Savin, "Split-extended LDPC codes for coded cooperation," in *Proceedings of International Symposium on Information Theory and its Applications (ISITA)*, 2010, pp. 151–156.

[37] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 638–656, 2001.

[38] L. Ping, X. Huang, and N. Phamdo, "Zigzag codes and concatenated zigzag codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 800–807, 2001.

[39] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1973–1973, 2005.

[40] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proceedings of 2nd International Symposium on Turbo codes and related topics*, 2000, pp. 1–8.

[41] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.