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

D3.2

Fault tolerant LDPC encoding and decoding

Editor:	E. Dupraz (ENSEA)
Deliverable nature:	Public
Due date:	October 31, 2014
Delivery date:	November 14, 2014
Version:	2.0
Date of current version:	April 15, 2016
Total number of pages:	71 pages
Reviewed by:	i-RISC members
Keywords:	LDPC codes, noisy density evolution, FAID decoders, stochastic decoders, bit-flipping decoders, fault-tolerant encoding and decoding, decoder design.

Abstract
This deliverable presents the overview of the main activities carried out within the Work-Package 3 (WP3) framework, during the period Month 13 to Month 21 (M13-M21) of the project. The main contributions included in this deliverable are related to the design and analysis of fault tolerant encoder and decoder architectures.

Concerning the fault tolerant LDPC decoders, in this deliverable we build on the results of the first year in order to design Finite Alphabet Iterative Decoders robust to hardware errors. We further investigate the robustness of the Stochastic decoder under faulty hardware, and propose a new Probabilistic Gradient Descent Bit-Flipping decoder, which is shown to provide increased error correction performance compared to standard bit-flipping algorithms, and to be robust to hardware errors.

Concerning the fault tolerant LDPC encoders, we evaluate the robustness of several encoding solutions and particular code constructions proposed in the literature to reduce the encoding complexity. As most of these solutions prove not to be robust to hardware errors, we further propose two new encoding solutions, which present increased robustness to hardware errors.

List of Authors

Participant	Author
CEA	Valentin Savin (valetin.savin@cea.fr)
	Christiane L. Kameni Ngassa (christiane.kameningassa@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)
	Sran Brkić (brka05@gmail.com)
	Omran Al Rasheed (omrano84@hotmail.com)
UCC	Emanuel Popovici (e.popovici@ucc.ie)
	Christian Spagnol (Christian.spagnol@ue.ucc.ie)

Contents

List of Dissemination Activities

Published papers

[P1] S. Brkic, P. Ivaniš, and B. Vasić, "Analysis of one-step majority logic decoding under correlated data-dependent gate failures," *IEEE International Symposium on Information Theory*, Honolulu, Hawaii, June 2014

[P2] E. Dupraz, D. Declercq, B. Vasić, and V. Savin "Finite Alphabet Iterative Decoders Robust to Faulty Hardware: Analysis and Selection", *International Symposium on Turbo Codes and Iterative Information Processing (STIC)*, Bremen, Germany, August 2014.

[P3] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Faulty Stochastic LDPC Decoders Over the Binary Symmetric Channel", *International Symposium on Turbo Codes and Iterative Information Processing (STIC)*, Bremen, Germany, August 2014.

[P4] O.-A. Rasheed, P. Ivanis, and B. Vasić, "Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoders," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487 - 1490, September 2014.

[P5] B. Vasić, and D. V. Nguyen, "Failure analysis of two-bit flipping decoding algorithms", *Signal Processing and Communications (SPCOM 2014)*, Indian Institute of Science, Bangalore, India, July 2014 (invited paper)

Submitted papers

[P6] C. L. Kameni Ngassa, V. Savin, E. Dupraz, and D. Declercq, "Density Evolution and Functional Threshold for the Noisy Min-Sum Decoder", *IEEE Transactions on Communications*, major revision.

[P7] E. Dupraz, D. Declercq, B. Vasić, and V. Savin "Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware", *IEEE Transactions on Communications*, submitted

[P8] P. Ivanis, O. Ras, and B. Vasić, "MUDRI: A fault-tolerant decoding algorithm," *IEEE International Conference on Communications (ICC 2015)*, London, UK, June 2015, submitted

[P9] 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, submitted.

Workshop presentations

[P10] B. Vasić and P. Ivanis, "Fault-Tolerant Decoders," *National Conference on Information Theory and Complex Systems (TINKOS 2014)*, Nis, Serbia, June 2014.

[P11] B. Vasić and P. Ivanis, "Reliable Memories Built from Unreliable Components: Theory and Connections with Codes on Graphs," *National Conference on Information Theory and Complex Systems (TINKOS)*, Nis, Serbia, June 2014.

[P12] V. Savin, "LDPC Codes and Message-Passing Decoders: An Introductory Survey", *National Conference on Information Theory and Complex Systems (TINKOS)*, Nis, Serbia, June 2014.

[P13] D. Declercq, "Recent Advances on Error Correction Coding with non-binary LDPC Codes", *National Conference on Information Theory and Complex Systems (TINKOS)*, Nis, Serbia, June 2014.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BER	Bit Error Rate
BI-AWGN	Binary Input Additive White Gaussian Noise
BSC	Binary Symmetric Channel
BF	Bit-Flipping
BP	Belief-Propagation
CN	Check Node
CNU	Check Node Update
DE	Density Evolution
EM	Edge Memories
FAID	Finite Alphabet Iterative Decoder
FD	Full-Depth
FER	Frame Error Rate
GDBF	Gradient-Descent Bit-Flipping
IRA	Irregular Repeat Accumulate
IVRG	Intrinsic-Value Random Generator
LDGM	Low Density Generator Matrix
LDPC	Low Density Parity Check
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MAJ	Majority-Logic
MS	Min-Sum
MP	Message-Passing
MWBF	Modified Weighted Bit-Flipping
NGDBF	Noisy Gradient-Descent Bit-Flipping
PBF	Probabilistic Bit-Flipping
PMF	Probability Mass Function
QC	Quasi-Cyclic
RBSG	Random-Binary Sequence Generator
RG	Random Generator
SCMS	Self-Corrected Min-Sum
SP	Sum-Product Algorithm
SP	Sign-Preserving
VLSI	Very Large Scale Integration

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 (WP3 – Fault Tolerant Algorithms for Error-Correction) is aimed at investigating error-correcting codes in the context of unreliable hardware. This represents a new paradigm in coding theory, since faulty hardware can potentially induce errors during both the encoding and decoding processes. It is then critical to properly evaluate the robustness of 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 period Month 13 to Month 21 (M13-M21) of the project is presented in the next section.

Executive Summary

In the period Month 13 to Month 21 (M13-M21), Work Package 3 (WP3) activities addressed the achievement of the following objectives[1]:

- Objective 3.1: Design of fault-tolerant LDPC encoders,
- Objective 3.2 – Design iterative LDPC decoders which are tolerant to gate errors,
- Objective 3.3 – Analysis of the asymptotical performance of LDPC decoders, as well as their performance in the low frame error rate region for finite block-lengths.

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, updated according to the recommendations from the First Technical Review Report of the project. It indicates that the tasks addressed during the period M13-M21 are Task 3.1 – MS/FAID decoders under faulty gates, Task 3.2 – Stochastic decoder under faulty gates, Task 3.4 – Practical fault tolerant encoding, and Task 3.5 – Randomized bit-flipping decoders for fault tolerance. The contributions of the first two years prepare the continuation of the work toward Task 3.3 – Quasi-error free protection or long-term protection under faulty hardware setting.

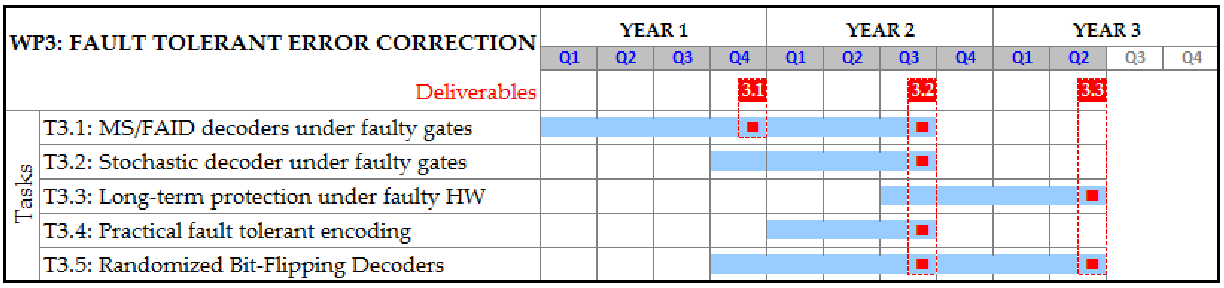


Figure 1: Gantt chart of WP3, updated according to the recommendations from the First Technical Review Report

Note that Task 3.5 was originally planned to cover the fault-tolerant iterative decoding of Reed-Muller and Polar-codes (according to the initial work-plan from the Description of Work document). However, according to the following recommendation from the First Technical Review Report of the project, we decided to refocus efforts on the fault tolerant decoding of LDPC codes, by considering a class of very low-complexity, but possible more robust decoders, namely bit-flipping decoders.
[Recommendation from the First Technical Review Report:] "This objective[2], while interesting, would divert too much effort in this WP away from its core work and may cause the project not to exploit the successful results already obtained on fault tolerant LDPC codes. It is the reviewers' opinion that this objective should be removed from the project and the project participants agreed with this view during the review meeting"

[1]Please note that WP3 has been slightly reorganized at the end of the first reporting period, as Objective 3.3 has been partially achieved – in advance – during the first year of the project, while Objective 3.1 has been postponed for the second deliverable.

[2]Editor's node: Objective 3.4 – Design of fault-tolerant Reed-Muller decoders for the case of very short block-lengths, and application to Polar-codes.

Related to these tasks the main technical contributions 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, we built on the results of the first year in order to design Finite Alphabet Iterative Decoders (FAIDs) robust to hardware errors, and we further investigated the Stochastic decoder and the Bit-Flipping (BF) decoder under faulty hardware settings. The main contributions can be summarized as:

- **Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware (Task 3.1).** As a direct continuation of the work of the first year, we have first provided a more precise analysis of FAIDs running on faulty hardware (Chapter 1). In order to characterize the asymptotic behavior of faulty FAIDs, we have introduced a new threshold definition referred to as the *functional threshold*. Although the functional threshold has its limitations when the hardware noise level is too high, it enables to predict which decoders are robust to hardware errors. Based on the functional threshold, we have proposed a rigorous method for the design of FAIDs robust to hardware errors. Monte-Carlo simulations have validated at finite-length the robustness of the decoders obtained from the proposed design method.
- **Faulty Stochastic Decoder (Task 3.2).** The results of the first year have shown that for both Min-Sum (MS) and FAID decoders, the reliability of the sign bit of the exchanged messages is a critical point in designing robust decoders. In order to deal with this issue, we have investigated stochastic decoders on faulty hardware (Chapter 2). For stochastic decoders, probability beliefs are converted into streams of random bits, referred to as stochastic streams, and complex arithmetic operations are performed by simple bit-wise operations on the streams. Noisy (faulty) operations within the stochastic decoder may only flip a few bits of the stochastic streams, which is expected to have a limited impact on the decoder performance. We have confirmed by Monte-Carlo simulations that stochastic decoders are naturally robust to errors introduced by faulty hardware. We have further highlighted the increased robustness of the stochastic decoder to hardware errors, compared to the MS decoder.
- **Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder (Task 3.5)**. We have proposed a new BF decoding algorithm called Probabilistic Gradient Descent Bit-Flipping (PGDBF) (Chapter 3), which introduces randomness in the bit-flipping rules. In the PGDBF algorithm, code bits with a number of unsatisfied parity checks larger than a fixed threshold are flipped with a given probability. Monte-Carlo simulations have shown that the PGDBF algorithm has better performance than standard BF algorithms, even when no errors are introduced in the decoder. The simulations have also shown that the PGDBF algorithm is robust to hardware errors and, more surprisingly, that hardware errors can sometimes improve the decoder performance. A similar result had already been observed for the MS decoder during the first year of the project. To finish, we have proposed an efficient implementation of the PGDBF decoder (Chapter 4). We have shown that the proposed implementation improves the error-correction performance of the algorithm and constitutes a promising solution for hardware implementation.

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

- **Faulty Encoding of Low Density Parity Check Codes (Task 3.4).** We first investigated several encoding solutions (Lower Triangular, Approximate Lower Triangular encoding) and particular code constructions (Zig-Zag, IRA codes, LDGM codes) proposed in the literature to reduce the encoding complexity. Unfortunately, most of these solutions proved not to be robust to hardware errors. We have then proposed two new encoding solutions, with increased robustness to hardware errors. The first solution performs systematic encoding, followed by a decoder that recovers the correct codeword prior to transmission over the channel. The second solution consists of the use of Low Density Generator Matrix (LDGM) codes. However, the first solution has high encoding complexity, while the second one shows degraded decoding performance.

Chapter 1

Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware

Abstract: *This chapter addresses the problem of designing LDPC decoders robust to transient errors introduced by a faulty hardware. We assume that the faulty hardware introduces errors during the message passing updates and we propose a general framework for the definition of the message update faulty functions. Within this framework, we define symmetry conditions for the faulty functions, and derive two simple error models used in the analysis. With this analysis, we introduce a new Density Evolution threshold definition referred to as the functional threshold. We analyze the behavior of the functional threshold and show its limitations in case of highly unreliable hardware. However, we show that under restricted decoder noise conditions, the functional threshold can be used to predict the convergence behavior of FAIDs under faulty hardware. In particular, we reveal the existence of robust and non-robust FAIDs and propose a framework for the design of robust decoders. We finally illustrate robust and non-robust decoders behaviors of finite length codes using Monte Carlo simulations.*

Part of the work presented in this Chapter has been published or submitted for publication in:

- E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Finite Alphabet Iterative Decoders Robust to Faulty Hardware: Analysis and Selection", *International Symposium on Turbo Codes and Iterative Information Processing (STIC)*, Bremen, Germany, August 2014 [P2]
- E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Analysis and Design of Finite Alphabet Iterative Decoders Robust to Faulty Hardware", *IEEE Transactions on Communications* (submitted) [P7]

1.1 Introduction

Reliability is becoming a major issue in the design of modern electronic devices. The huge increase in integration factors coupled with the important reduction of the chip sizes makes the devices much more sensitive to noise and may induce transient errors. Furthermore, the fabrication process makes hardware components more prone to defects and may also cause permanent computation errors. As a consequence, in the context of communication and storage, errors may not only come from transmission channels, but also from the faulty hardware used in transmitters and receivers.

The general problem of reliable function computation using faulty gates was first addressed by von Neumann in [2] and the notion of redundancy was later considered in [3–5]. Hardware redundancy is defined as the number of noisy gates required for reliable function computation divided by the number of noiseless gates needed for the same function computation. Gács and Gál [3] and Dobrushin and Ortyukov [4], respectively, provided lower and upper bounds on the hardware redundancy for reliable Boolean function computation from faulty gates. Pippenger [5] showed that finite asymptotic redundancy can be achieved when using Low Density Parity Check (LDPC) codes for the reliable computation of linear Boolean functions. Taylor [6] and Kuznetsov [7] considered memories as a particular instance of this problem and provided an analysis of a memory architecture based on LDPC decoders made of faulty components. More recently, an equivalence between the architecture proposed by Taylor and a noisy Gallager-B decoder was identified by Vasic *et al.* [8], while Chilappagari *et al.* [9] analyzed a memory architecture based on one-step majority logic decoders.

As a consequence, there is a need to address the problem of constructing reliable LDPC decoders made of faulty components not only for error correction on faulty hardware, but also as a first step in the context of reliable function computation and storage. Formulating a general method for construction of robust decoders requires understanding whether a particular decoder is inherently robust to errors introduced by the faulty hardware. There is also a need for a rigorous analysis to determine which characteristics of decoders make them robust.

To answer to the first point, Varshney [10] introduced a framework referred to as noisy Density Evolution (noisy-DE) for the performance analysis of noisy LDPC decoders in terms of asymptotic error probability. Based on this framework, the asymptotic performance of a variety of noisy LDPC decoders was analyzed. In [10], infinite precision BP decoders were investigated, which is not useful for actual implementation on faulty hardware. On the contrary, noisy practically important hard-decision decoders, such as noisy Gallager-A [10] and Gallager-E [11] decoders were considered. Gallager-B decoders were analyzed for binary [8, 12, 13] and non-binary [14] alphabets under transient error models, and [13] also considered permanent error models. From the same noisy-DE framework, [15,16] proposed an asymptotic analysis of the behavior of stronger discrete Min-Sum (MS) decoders, for which the exchanged messages are no longer binary but are quantized soft information represented by a finite (and typically small) number of bits.

Recently, a new class of LDPC decoders referred to as Finite Alphabet Iterative Decoders (FAIDs) has been introduced [1]. In these decoders, the messages take their values in small alphabets and the variable node update is derived through a predefined Boolean function. The FAID framework offers the possibility to define a large collection of these functions, each corresponding to a particular decoding algorithm. The FAIDs were originally introduced to address the error floor problem, and designed to correct error events located on specific small topologies of error events referred to as *trapping sets* that usual decoders (MS, BP-based) cannot correct. When operating on faulty hardware, the FAIDs may potentially exhibit very different properties in terms of tolerance to transient errors and we are interested in identifying the robust ones among the large diversity of decoders.

In this chapter, we propose a rigorous method for the analysis and the design of decoding rules robust to transient errors introduced by the hardware. We assume that the faulty hardware introduces transient errors during function computation and propose a general description of faulty functions. We introduce new symmetry conditions for faulty functions that are more general than those in [10]. We discuss possible simplifications of the general description and present two particular error models to represent the faulty hardware effect. The design procedure we propose is based on an asymptotic performance analysis of noisy-FAIDs using noisy-DE. In order to characterize the asymptotic behavior of the FAIDs from the noisy-DE equations, we introduce a new noisy-DE threshold definition referred to as the *functional threshold*. We analyze the behavior of the functional threshold and we observe that if the decoder noise level is too high, the functional threshold fails at predicting the convergence behavior of the faulty decoder. However, under the restricted decoder noise conditions, we show that the functional threshold can be used to predict the behavior of noisy-FAIDs and gives a criterion for the comparison of the asymptotic performance of the decoders. Based on this criterion, we then propose a noisy-DE based framework for the design of decoders inherently robust to errors introduced by the hardware. Finite-length simulations illustrate the gain in performance at considering robust FAIDs on faulty hardware.

The outline of the chapter is as follows. Section 1.2 gives the notations and basic decoder definition. Section 1.3 introduces a general description of faulty functions and presents particular error models. Section 1.4 gives the noisy-DE analysis for particular decoder noise models. Section 1.5 restates the definition of the functional threshold and presents the analysis of its behavior. Section 1.6 presents the method for the design of robust decoders. Section 1.7 gives the finite-length simulation results.

1.2 Notations and Decoders Definition

This section introduces notations and basic definitions of FAIDs introduced in [1]. In the following, we assume that the transmission channel is a Binary Symmetric Channel (BSC) with parameter α. We consider a BSC because on the hardware all the operations are performed at a binary level.

An N_s-level FAID is defined as a 5-tuple given by $\mathrm{D} = (\mathcal{M}, \mathcal{Y}, \Phi^{(v)}, \Phi^{(c)}, \Phi^{(a)})$. 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 $\mathcal{Y} = \{\pm\mathrm{B}\}$. The channel value $y \in \mathcal{Y}$ corresponding to Variable Node (VN) v 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 values of incoming messages to a Check Node (CN) of degree d_c and let $\eta_1, \ldots, \eta_{d_v-1}$ be the values of incoming messages to a VN of degree d_v. Denote $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_{d_c-1}]$ and $\boldsymbol{\eta} = [\eta_1, \ldots, \eta_{d_v-1}]$ the vector representations of the incoming messages to a CN and to a VN, respectively. FAIDs are iterative decoders and as a consequence, messages $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$ are computed at each iteration. However, for simplicity, the current iteration is not specified in the notations of the messages.

At each iteration of the iterative decoding process, the following operations are performed on the messages. The Check Node Update (CNU) function $\Phi^{(c)} : \mathcal{M}^{d_c-1} \to \mathcal{M}$ is used for the message update at a CN of degree d_c. The corresponding outgoing message is computed as

$$\eta_{d_c} = \Phi^{(c)}(\boldsymbol{\mu}). \tag{1.1}$$

In [1], $\Phi^{(c)}$ corresponds to the CNU of the standard MS decoder. The Variable Node Update (VNU) function $\Phi^{(v)} : \mathcal{M}^{d_v-1} \times \mathcal{Y} \to \mathcal{M}$ is used for the update at a VN of degree d_v. The corresponding outgoing message is computed as

$$\mu_{d_v} = \Phi^{(v)}(\boldsymbol{\eta}, y). \tag{1.2}$$

The properties that Φ_v must satisfy are given in [1]. At the end of each decoding iteration, the *A Posteriori* Probability (APP) computation produces messages γ calculated from the function $\Phi^{(a)} : \mathcal{M}^{d_v} \times \mathcal{Y} \to \bar{\mathcal{M}}$, where $\bar{\mathcal{M}} = \{-L_{s'}, \ldots, L_{s'}\}$ is a discrete alphabet of $N_{s'} = 2s' + 1$ levels. Denote $\boldsymbol{\eta}^\star = [\eta_1, \ldots, \eta_{d_v}]$ the vector representation of all the messages incoming to a VN. The APP computation produces

$$\gamma = \Phi^{(a)}(\boldsymbol{\eta}^\star, y). \tag{1.3}$$

The APP is usually computed on a larger alphabet $\bar{\mathcal{M}}$ in order to limit the impact of saturation effects when calculating the APP. The mapping $\Phi^{(a)}$ is given by

$$\Phi^{(a)}(\tilde{\boldsymbol{\eta}}^\star, y) = \sum \tilde{\boldsymbol{\eta}}^\star + y \quad . \tag{1.4}$$

The hard-decision bit corresponding to each variable node v_n is given by the sign of the APP. If $\Phi^{(a)}(\tilde{\boldsymbol{\eta}}^\star, y) = 0$, then the hard-decision bit is selected at random and takes value 0 with probability 1/2.

Alternatively, $\Phi^{(v)}$ can be represented as a Look-Up Table (LUT). For instance, Table 1.1 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. Classical decoders such as the standard MS and the offset MS can also be seen as instances of FAIDs. It indeed suffices to derive the specific LUT from the VNU functions of these decoders. Table 1.2 gives the VNU of the 7-level offset MS decoder. Therefore, the VNU formulation enables to define a large collection of decoders with common characteristics but potentially different robustness to noise. In the following, after introducing error models for the faulty hardware, we describe a method for analyzing the asymptotic behavior of noisy-FAIDs. This method enables us to compare decoder robustness for different mappings $\Phi^{(v)}$ and thus to design decoders robust to faulty hardware.

Table 1.1: LUT $\Phi^{(v)}_{\text{opt}}$ reported in [1] optimized for the error floor

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

Table 1.2: VNU of a 3-bit offset MS represented as a FAID

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

1.3 Error Models for Faulty Hardware

Here, we assume that the faulty hardware introduces transient errors only during function computation. For the performance analysis of faulty decoders, specific error models have been considered in previous works. In [11, 13, 15], transient errors are assumed to appear at a binary level on message wires between VNs and CNs. In [10, 16], the noise effect is represented by a random variable independent of the function inputs and applies only through a deterministic error injection function. Here we propose a more general error model which includes the above cases.

For the noisy-DE analysis, the considered faulty functions have to be symmetric, which implies that the error probability of the decoder does not change when flipping a codeword symbol. As a consequence, the error probability of the decoder does not depend on the transmitted codeword, which greatly simplifies the analysis. Here, we introduce new symmetry conditions for the general error models. We then discuss possible simplifications of the general model and introduce two particular simple error models which allow the asymptotic analysis of the faulty iterative decoding.

1.3.1 General Faulty Functions and Symmetry Conditions

To describe general faulty functions, we replace the deterministic functions $\Phi^{(c)}$, $\Phi^{(v)}$, $\Phi^{(a)}$ introduced in Section 1.2 by the following conditional Probability Mass Functions (PMF). Denote $\tilde{\mu}_{d_v}$, $\tilde{\eta}_{d_c}$, and $\tilde{\gamma}$ the noisy versions of μ_{d_v}, η_{d_c}, γ, and denote $\tilde{\boldsymbol{\mu}} = [\tilde{\mu}_1, \dots, \tilde{\mu}_{d_c-1}]$, $\tilde{\boldsymbol{\eta}} = [\tilde{\eta}_1, \dots, \tilde{\eta}_{d_v-1}]$, $\tilde{\boldsymbol{\eta}}^\star = [\tilde{\eta}_1, \dots, \tilde{\eta}_{d_v}]$ their vector representations. Then a faulty VNU is defined as the conditional PMF

$$\mathrm{P}^{(v)}(\tilde{\mu}_{d_v} | \tilde{\boldsymbol{\eta}}, y), \tag{1.5}$$

a faulty CNU is defined as

$$\mathrm{P}^{(c)}(\tilde{\eta}_{d_c} | \tilde{\boldsymbol{\mu}}), \tag{1.6}$$

and a faulty APP is defined as

$$\mathrm{P}^{(a)}(\tilde{\gamma} | \tilde{\boldsymbol{\eta}}^\star, y). \tag{1.7}$$

The described model is memoryless and takes only into account transient errors in the decoder, but it ignores permanent errors and possible dependencies with previous or future function arguments. However it is general enough to represent any type of memoryless mapping and error model.

For the noisy-DE analysis, the considered faulty functions have to be symmetric. The definitions of symmetry given in [10] only consider the particular case of error injection functions and are not sufficient to characterize the symmetry of the above faulty functions. In the following, we introduce more general definitions of symmetry.

Definition 1. A faulty VNU is said to be symmetric if

$$\mathrm{P}^{(v)}(\tilde{\mu}_{d_v} | \tilde{\boldsymbol{\eta}}, y) = \mathrm{P}^{(v)}(-\tilde{\mu}_{d_v} | -\tilde{\boldsymbol{\eta}}, -y). \tag{1.8}$$

2. A faulty CNU is said to be symmetric if

$$\mathrm{P}^{(c)}(\tilde{\eta}_{d_c} | \mathbf{a}.\tilde{\boldsymbol{\mu}}) = \mathrm{P}^{(c)}\left(\left(\prod \mathbf{a}\right) \tilde{\eta}_{d_c} | \tilde{\boldsymbol{\mu}}\right). \tag{1.9}$$

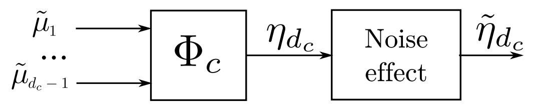


Figure 1.1: Function decomposition for the CNU

where $\mathbf{a} = [a_1, \dots, a_{d_c-1}]$, $a_i \in \{-1, 1\}$, $\mathbf{a}.\tilde{\boldsymbol{\mu}}$ is the component by component product of $\mathbf{a}$ and $\tilde{\boldsymbol{\mu}}$, and $\prod \mathbf{a}$ is the product of all components in vector $\mathbf{a}$.

3. A faulty APP is said to be symmetric if

$$\mathrm{P}^{(a)}(\tilde{\mu}_{d_v}|\tilde{\boldsymbol{\eta}}^{\star}, y) = \mathrm{P}^{(a)}(-\tilde{\mu}_{d_v}| - \tilde{\boldsymbol{\eta}}^{\star}, -y). \tag{1.10}$$

Note that our definitions of symmetry are the same as the ones originally introduced in [17] for deterministic decoders, except that ours apply on conditional PMFs instead of deterministic mappings.

1.3.2 Faulty Function Decomposition

A possible simplification of the general models described in the previous section is to consider that the noise appears only at the output of a function computation. More precisely, we assume that the noisy function can be decomposed as a noiseless function followed by the noise effect, as in Fig. 1.1 for the case of the CNU. In this simplified error model, η_{d_c}, μ_{d_v}, and γ, represent the messages at the output of the noiseless CNU, VNU, and APP computation respectively, and their noisy versions are denoted $\tilde{\eta}_{d_c}$, $\tilde{\mu}_{d_v}$, $\tilde{\gamma}$. The noisy output is assumed to be independent of the inputs conditionally to the noiseless output, *i.e.*, for the case of faulty CNU, this gives $\mathrm{P}^{(c)}(\tilde{\eta}_{d_c}|\eta_{d_c}, \tilde{\boldsymbol{\mu}}) = \mathrm{P}^{(c)}(\tilde{\eta}_{d_c}|\eta_{d_c})$. Furthermore, as the noiseless output is obtained from a deterministic function of the inputs, we get

$$\mathrm{P}^{(c)}(\tilde{\eta}_{d_c}|\tilde{\boldsymbol{\mu}}) = \mathrm{P}^{(c)}(\tilde{\eta}_{d_c}|\Phi^{(c)}(\tilde{\boldsymbol{\mu}})). \tag{1.11}$$

The same conditions hold for the faulty VNU and APP.

The noise effects at the output of $\Phi^{(c)}$ and $\Phi^{(v)}$ are represented by probability transition matrices $\Pi^{(v)}$ and $\Pi^{(c)}$ respectively, with

$$\Pi^{(c)}_{k,m} = \Pr(\tilde{\eta}_{d_c} = m|\eta_{d_c} = k),\ \Pi^{(v)}_{k,m} = \Pr(\tilde{\mu}_{d_v} = m|\mu_{d_v} = k), \quad \forall k, m \in \mathcal{M} \tag{1.12}$$

wherein the matrix entries are indexed by the values in $\mathcal{M}$. This indexing is used for all the vectors and matrices introduced in the remaining of the chapter. The noise effect on $\Phi^{(a)}$ is modeled by the probability transition matrix $\Pi^{(a)}$ with

$$\Pi^{(a)}_{k,m} = \Pr(\tilde{\gamma} = m|\gamma = k), \quad \forall k, m \in \bar{\mathcal{M}}. \tag{1.13}$$

The forms of the probability transition matrices depend on the considered error models. In the next section, two simple examples derived from this simplified model are introduced. They will then be considered in the noisy-DE analysis.

Note that in the above decomposition model the noise is added only at a message level at the output of the noiseless functions. An alternative model would be to consider noise effect introduced *inside* the functions, for example during elementary operations such as the minimum computation between two elements in $\Phi^{(c)}$, as in [16]. While the decomposition model introduced here may not capture all the noise effects, it is sufficient for the analysis of the behavior and robustness of noisy decoders without requiring knowledge of a particular hardware implementation. More accurate models will be considered in future works.

Note that some faulty functions cannot be decomposed as a deterministic mapping followed by the noise effect. For example, it can be verified that the faulty minimum function defined as

$$\tilde{\eta}_3 = \begin{cases} \min(\mu_1, \mu_2) \text{ with probability } 1-p \\ \max(\mu_1, \mu_2) \text{ with probability } p \end{cases} \tag{1.14}$$

does not satisfy (1.11).

1.3.3 Particular Decoder Noise Models

In the following, two particular noise models models that we have proposed in [18] will be considered. They are derived from the above decomposition model by specifying particular transition matrices $\Pi^{(c)}, \Pi^{(v)}, \Pi^{(a)}$ and will be considered for the noisy-DE analysis.

Sign-Preserving error model

The first model is called the Sign-Preserving (SP) model. It has a SP property, meaning that noise is assumed to affect only the message amplitude, but not its sign. Although this model is introduced for the purpose of asymptotic analysis, it is also a practical model, as protecting the sign can be realized at the hardware level by proper circuit design. The probability transition matrices for the SP-Model can be constructed from a SP-transfer matrix defined as follows.

Definition The SP-transfer matrix $\Pi^{(\text{SP})}(p,s)$ is a matrix of size $(2s+1)\times(2s+1)$ such that

$$\begin{aligned}
&\Pi^{(\text{SP})}_{k,k}(p,s) = 1-p, \quad \Pi^{(\text{SP})}_{k,0}(p,s) = \frac{p}{s}, \quad \Pi^{(\text{SP})}_{0,k}(p,s) = \frac{p}{2s} \\
&\Pi^{(\text{SP})}_{k,m}(p,s) = \frac{p}{s}, \text{ for } m \neq k \neq 0, \ \text{sign}(m) = \text{sign}(k) \\
&\Pi^{(\text{SP})}_{k,k}(p,s) = 0, \text{ elsewhere.}
\end{aligned} \tag{1.15}$$

According to this definition, a strictly positive message can be altered to only another positive message and the same holds for strictly negative messages.

The matrices $\Pi^{(c)}$, $\Pi^{(v)}$, and $\Pi^{(a)}$ can be now obtained from $\Pi^{(\text{SP})}$ as a template. The noise level parameter at the output of $\Phi^{(c)}$ is given by the parameter p_c, and the corresponding probability transition matrix is given by $\Pi^{(c)} = \Pi^{(\text{SP})}(p_c, s)$. In the same way, the noise level parameters at the output of $\Phi^{(v)}$ and $\Phi^{(a)}$ are denoted p_v and p_a respectively, and the corresponding probability transition matrices are given by $\Pi^{(v)} = \Pi^{(\text{SP})}(p_v, s)$ and $\Pi^{(a)} = \Pi^{(\text{SP})}(p_a, s')$. In the following, the collection of hardware noise parameters will be denoted $\nu = (p_v, p_c, p_a)$. The probability transition matrix $\Pi^{(a)}$ is of size $(2s'+1)\times(2s'+1)$ because the APP (1.3) is computed on the alphabet $\bar{\mathcal{M}}$ of size $(2s'+1)$. It can be verified that if the deterministic mappings $\Phi^{(v)}$, $\Phi^{(c)}$, $\Phi^{(a)}$, are symmetric in the sense of [17, Definition 1], then the SP-model gives symmetric faulty functions from conditions (1.8), (1.9), (1.10), in Definition 1.3.1.

Full-Depth error model

The second model is called the Full-Depth (FD) model. This model is potentially more harmful than the SP-Model because the noise affects both the amplitude and the sign of the messages. However, it does not require hardware sign-protection any more. The FD-transfer matrix is defined as follows.

Definition The FD-transfer matrix $\Pi^{(\text{FD})}(p,s)$ is a matrix of size $(2s+1)\times(2s+1)$ such that

$$\begin{aligned}
&\Pi^{(\text{FD})}_{k,k}(p,s) = 1-p, \\
&\Pi^{(\text{FD})}_{k,m}(p,s) = \frac{p}{s}, \text{ for } m \neq k.
\end{aligned} \tag{1.16}$$

The FD-transfer Matrix defines a $(2s+1)$-ary symmetric model of parameter p. The noise level parameters at the end of $\Phi^{(c)}$, $\Phi^{(v)}$, $\Phi^{(a)}$, are denoted as before p_c, p_v, p_a, respectively, and $\nu = (p_v, p_c, p_a)$. The corresponding probability transition matrices are given by $\Pi^{(c)} = \Pi^{(\text{FD})}(p_c, s)$, $\Pi^{(v)} = \Pi^{(\text{FD})}(p_v, s)$, and $\Pi^{(a)} = \Pi^{(\text{FD})}(p_a, s')$. It can be verified that if the deterministic mappings $\Phi^{(v)}$, $\Phi^{(c)}$, $\Phi^{(a)}$, are symmetric in the sense of [17, Definition 1], then the FD-model gives symmetric faulty functions from the conditions (1.8), (1.9), (1.10), in Definition 1.3.1.

1.4 Noisy Density Evolution

This section presents the noisy-DE recursion for asymptotic performance analysis of FAIDs on faulty hardware. The DE [10] consists of expressing the Probability Mass Function (PMF) of the messages at successive iterations under the local independence assumption, that is the assumption that the messages coming to a node are independent. As a result, the noisy-DE equations can be used to derive the error probability of the considered decoder as a function of the hardware noise parameters. The noisy-DE analysis is valid on average over all possible LDPC code constructions, when infinite codeword length is considered.

In the following, we first discuss the all-zero codeword assumption which derives from the symmetry conditions of Definition 1.3.1 and greatly simplifies the noisy-DE analysis.

1.4.1 All-zero Codeword Assumption

In [17], it was shown that if the channel is output-symmetric, and the VNU and CNU functions are symmetric functions, the error probability of the decoder does not depend on the transmitted codeword. From this codeword independence, one can compute the PMFs of the messages and the error probability of the decoder assuming that the all-zero codeword was transmitted. The codeword independence was further extended in [10,18] to the case of faulty decoders when the noise is introduced through symmetric error injection functions. Unfortunately, the results of [10,18] do not apply to our more general error models. In particular, the proof technique of [10, 18] cannot be used when the noise is not introduced through deterministic error injection functions. The following theorem thus restates the codeword independence for faulty functions described by the general error introduced in Section 1.3.1 and for the symmetry conditions of Definition 1.3.1.

Theorem 1.1 *Consider a linear code and a faulty decoder defined by a faulty VNU* (1.5)*, a faulty CNU* (1.6)*, and a faulty APP* (1.7)*. Denote $P_e^{(\ell)}(\mathbf{x})$ the probability of error of the decoder at iteration ℓ conditioned on the fact that the codeword $\mathbf{x}$ was transmitted. If the transmission channel is symmetric in the sense of [17, Definition 1] and if the faulty VNU, CNU, and APP are symmetric in the sense of Definition 1.3.1, then $P_e^{(\ell)}(\mathbf{x})$ does not depend on $\mathbf{x}$.*

Theorem 1.1 states that for a symmetric transmission channel and symmetric faulty functions, the error probability of the decoder is independent of the transmitted codeword. All the error models considered in this chapter are symmetric and as a consequence, we will assume that the all-zero codeword was transmitted. Note that when the decoder is not symmetric, DE can be performed from the results of [19, 20]. In this case, it is not possible anymore to assume that the all-zero codeword was transmitted, and the analysis becomes much more complex.

1.4.2 Noisy-DE Equations

In this section, we assume that the all-zero codeword was transmitted, and we express the PMFs of the messages at successive iterations. The error probability of the decoder at a given iteration can then be computed from the PMFs of the messages at the considered iteration. The analysis is presented for regular LDPC codes. However, the generalization to irregular codes is straightforward.

Let the N_s-tuple $\mathbf{q}^{(\ell)}$ denote the PMF of an outgoing message from a VN at ℓ-th iteration. In other words, the μ-th component $q_\mu^{(\ell)}$ of $\mathbf{q}^{(\ell)}$ is the probability that the outgoing message takes the value $\mu \in \mathcal{M}$. Similarly, let $\mathbf{r}^{(\ell)}$ denote the PMF of an outgoing message from a CN. The PMFs of noisy messages are represented by $\tilde{\mathbf{q}}^{(\ell)}$ and $\tilde{\mathbf{r}}^{(\ell)}$, respectively. In the following, the noisy-DE recursion is expressed with respect to general probability transition matrices $\Pi^{(c)}$, $\Pi^{(v)}$, $\Pi^{(a)}$. To obtain the noisy-DE equations for a specific error model, it suffices to replace these general probability transition matrices with the ones corresponding to the considered model.

The density evolution is initialized with the PMF of the channel value

$$q_{-B}^{(0)} = 1-\alpha \quad q_{+B}^{(0)} = \alpha \quad q_k^{(0)} = 0 \text{ elsewhere.}$$

Denote $\tilde{\mathbf{q}}_{\boldsymbol{\mu}}^{(\ell-1)}$ the (d_c-1)-tuple associated to $\boldsymbol{\mu}$. More precisely, if the k-th component of $\boldsymbol{\mu}$ is given by μ_k, then the k-th component of $\tilde{\mathbf{q}}_{\boldsymbol{\mu}}^{(\ell-1)}$ is given by $\tilde{q}_{\mu_k}^{(\ell-1)}$. The PMF $\mathbf{r}^{(\ell)}$ of the output of the CNU is obtained from the expression of Φ_c as $\forall \eta \in \mathcal{M}$,

$$r_\eta^{(\ell)} = \sum_{\boldsymbol{\mu}:\Phi_c(\boldsymbol{\mu})=\eta} \prod \tilde{\mathbf{q}}_{\boldsymbol{\mu}}^{(\ell-1)} \tag{1.17}$$

where the vector product operator is performed componentwise on vector elements. The noisy PMF is then obtained directly in vector form as

$$\tilde{\mathbf{r}}^{(\ell)} = \Pi^{(c)}\mathbf{r}^{(\ell)}. \tag{1.18}$$

Denote $\tilde{\mathbf{r}}_{\boldsymbol{\eta}}^{(\ell)}$ the (d_v-1)-tuple associated to $\boldsymbol{\eta}$. The PMF $\mathbf{q}^{(\ell)}$ of the output of the VNU is obtained from the expression of Φ_v as $\forall \mu \in \mathcal{M}$,

$$q_\mu^{(\ell)} = \sum_{\boldsymbol{\eta}:\Phi_v(\boldsymbol{\eta},-B)=\mu} q_{-B}^{(0)} \prod \tilde{\mathbf{r}}_{\boldsymbol{\eta}}^{(\ell)} + \sum_{\boldsymbol{\eta}:\Phi_v(\boldsymbol{\eta},+B)=\mu} q_{+B}^{(0)} \prod \tilde{\mathbf{r}}_{\boldsymbol{\eta}}^{(\ell)} \tag{1.19}$$

and

$$\tilde{\mathbf{q}}^{(\ell)} = \Pi^{(v)}\mathbf{q}^{(\ell)}. \tag{1.20}$$

Finally, applying the sequence of 4 equations (1.17), (1.18), (1.19) and (1.20) implements one recursion of the noisy-DE over the BSC channel.

The error probability of the decoder can be obtained from the above recursion and from the PMF of the messages at the end of the APP computation. Denote $\tilde{\mathbf{r}}_{\bar{\boldsymbol{\eta}}}^{(\ell)}$ the d_v-tuple associated to $\bar{\boldsymbol{\eta}}$, and denote $\mathbf{q}_{\text{app}}^{(\ell)}$ and $\tilde{\mathbf{q}}_{\text{app}}^{(\ell)}$ the respective noiseless and noisy PMFs of the messages at the output of the APP computation. They can be expressed from (1.3) as $\forall \gamma \in \bar{\mathcal{M}}$,

$$q_{\text{app},\gamma}^{(\ell)} = \sum_{\bar{\boldsymbol{\eta}}:\Phi_a(\bar{\boldsymbol{\eta}}^\star,-B)=\gamma} q_{-B}^{(0)} \prod \tilde{\mathbf{r}}_{\bar{\boldsymbol{\eta}}}^{(\ell)} + \sum_{\bar{\boldsymbol{\eta}}:\Phi_a(\bar{\boldsymbol{\eta}}^\star,+B)=\gamma} q_{+B}^{(0)} \prod \tilde{\mathbf{r}}_{\bar{\boldsymbol{\eta}}}^{(\ell)}$$

and

$$\tilde{\mathbf{q}}_{\text{app}}^{(\ell)} = \Pi^{(a)}\mathbf{q}_{\text{app}}^{(\ell)}. \tag{1.21}$$

Finally, for a given α and hardware noise parameters $\nu = (p_v, p_c, p_a)$, the error probability at each iteration can be computed under the all-zero codeword assumption as

$$P_{e,\nu}^{(\ell)}(\alpha) = \frac{1}{2}\tilde{q}_{\text{app},0}^{(\ell)} + \sum_{k<0} \tilde{q}_{\text{app},k}^{(\ell)}. \tag{1.22}$$

Lower bounds on the error probability can be obtained as follows, see [21].

Proposition 1 *The following lower bounds hold at every iteration ℓ*

1. *For the SP model, $P_{e,\nu}^{(\ell)}(\alpha) \geq \frac{1}{2s'}p_a$*
2. *For the FD model, $P_{e,\nu}^{(\ell)}(\alpha) \geq \frac{1}{2}p_a + \frac{p_a}{4s'}$*

The term s' appears in the two lower bounds because the APP (1.3) is computed on the alphabet $\bar{\mathcal{M}}$ of size $2s'+1$.

The asymptotic error probability of an iterative decoder is the limit of $P_{e,\nu}^{(\ell)}(\alpha)$ when ℓ goes to infinity. If the limit exists, let us denote $P_{e,\nu}^{(+\infty)}(\alpha) = \lim_{\ell\to+\infty} P_{e,\nu}^{(\ell)}(\alpha)$. In the case of noiseless decoders ($p_v = p_c = p_a = 0$), the maximum channel parameter α such that $P_{e,\nu}^{(+\infty)}(\alpha) = 0$ is called the DE *threshold* of the decoder [17]. However, the condition $P_{e,\nu}^{(+\infty)}(\alpha) = 0$ cannot be reached in general for faulty decoders. For instance, from Proposition 1, we see that the noise in the APP computation prevents the decoder from reaching a zero error probability. Thus, the concept of iterative decoding threshold for faulty decoders has to be modified, and adapted to the fact that only very low asymptotic error probabilities, bounded away from zero, are achievable. The following section introduces the definition of the functional threshold to characterize the asymptotic behavior of faulty decoders. We then analyze in details the properties of the functional threshold.

1.5 Analysis of Convergence Behaviors of Faulty Decoders

Varshney in [10] defines the *useful* region as the set of parameters α for which $P_{e,\nu}^{(+\infty)}(\alpha) < \alpha$. The useful region indicates what are the faulty hardware and channel noise conditions that a decoder can tolerate to reduce the level of noise. However, there are situations where the decoder can actually reduce the noise while still experiencing a high level of error probability. As a consequence, the useful region does not predict which channel parameters lead to a low level of error probability. Another threshold characterization has been proposed in [10, 15], where a constant value λ is fixed and the target-BER threshold is defined as the maximum value of the channel parameter α such that $P_{e,\nu}^{(+\infty)}(\alpha) \leq \lambda$. However, the target-BER definition has its limitations. The choice of lambda is arbitrary, and the target-BER threshold does not capture an actual "threshold behavior", defined as a sharp transition between a low level and a high level of error probability.

In this section, we introduce another threshold definition referred to as the functional threshold to detect the sharp transition between the two levels of error probability. In this section, we first recall the functional threshold definition. We then provide a new detailed analysis of the functional threshold behaviors and properties. In particular, we point out the limitations of the functional threshold for the prediction of the asymptotic performance of faulty decoders.

1.5.1 Functional Threshold Definition

Here, we introduce the functional threshold definition. The functional threshold definition uses the Lipschitz constant of the function $\alpha \mapsto P_{e,\nu}^{(+\infty)}(\alpha)$ defined as

Definition Let $P_{e,\nu}^{(+\infty)} : I \to \mathbb{R}$ be a function defined on an interval $I \subseteq \mathbb{R}$. The *Lipschitz constant* of $P_{e,\nu}^{(+\infty)}$ in I is defined as

$$L\left(P_{e,\nu}^{(+\infty)}, I\right) = \sup_{\alpha \neq \beta \in I} \frac{|P_{e,\nu}^{(+\infty)}(\alpha) - P_{e,\nu}^{(+\infty)}(\beta)|}{|\alpha - \beta|} \in \mathbb{R}_+ \cup \{+\infty\} \tag{1.23}$$

For $a \in I$ and $\delta > 0$, let $I_a(\delta) = I \cap (a-\delta, a+\delta)$. The *(local) Lipschitz constant* of $P_{e,\nu}^{(+\infty)}$ in $\alpha \in I$ is defined by:

$$L\left(P_{e,\nu}^{(+\infty)}, \alpha\right) = \inf_{\delta > 0} L\left(P_{e,\nu}^{(+\infty)}, I_\alpha(\delta)\right) \in \mathbb{R}_+ \cup \{+\infty\} \tag{1.24}$$

Note that if α is a discontinuity point of $P_{e,\nu}^{(+\infty)}$, then $L\left(P_{e,\nu}^{(+\infty)}, \alpha\right) = +\infty$. On the opposite, if $P_{e,\nu}^{(+\infty)}$ is differentiable in α, then the Lipschitz constant in α corresponds to the absolute value of the derivative. Furthermore, if $L\left(P_{e,\nu}^{(+\infty)}, I\right) < +\infty$, then $P_{e,\nu}^{(+\infty)}$ is uniformly continuous on I and almost everywhere differentiable. In this case, $P_{e,\nu}^{(+\infty)}$ is said to be *Lipschitz continuous* on I.

The functional threshold is then defined as follows.

Definition For given decoder noise parameters $\nu = (p_v, p_c, p_a)$ and a given channel parameter α, the decoder is said to be *functional* if it satisfies the three conditions below

(a) The function $x \mapsto P_{e,\nu}^{(+\infty)}(x)$ is defined on $[0, \alpha]$,

(b) $P_{e,\nu}^{(+\infty)}$ is Lipschitz continuous on $[0, \alpha]$, and

(c) $L\left(P_{e,\nu}^{(+\infty)}, x\right)$ is an increasing function of $x \in [0, \alpha]$.

Then the functional threshold $\bar{\alpha}$ is defined as

$$\bar{\alpha} = \sup\{\alpha \mid \text{conditions } (a), (b) \text{ and } (c) \text{ above are satisfied}\} \tag{1.25}$$

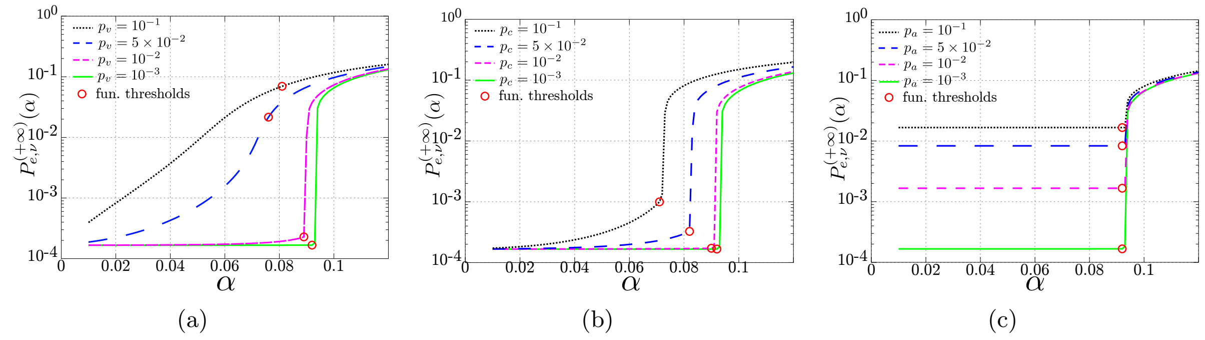


Figure 1.2: Asymptotic error probabilities for $(3,5)$ codes for the offset MS, for $B = 1$, for the SP-Model, with (a) $p_c = 10^{-3}$, $p_a = 10^{-3}$, (b) $p_v = 10^{-3}$, $p_a = 10^{-3}$, (c) $p_v = 10^{-3}$, $p_c = 10^{-3}$

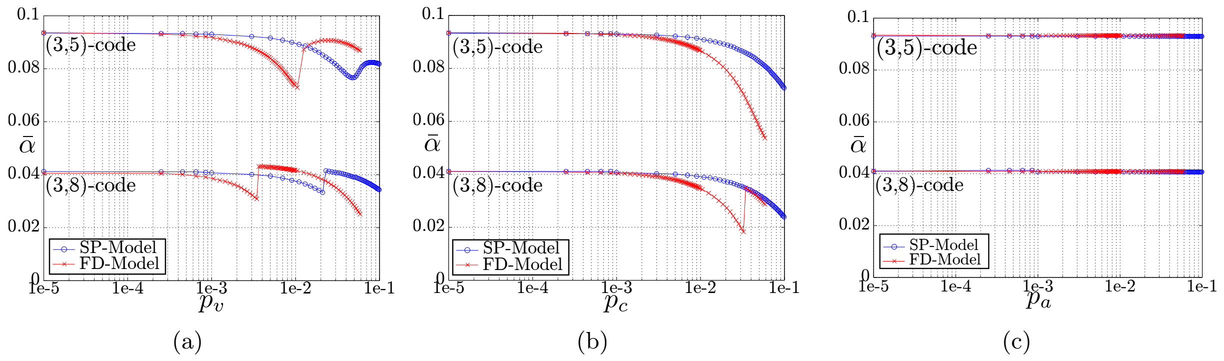


Figure 1.3: Functional regions for the offset MS, for $B = 1$, (a) w.r.t. p_v, with $p_c = p_a = 10^{-3}$ (SP-Model) and $p_c = p_a = 10^{-4}$ (FD-Model), (b) w.r.t. p_c, with $p_v = p_a = 10^{-3}$ (SP-Model) and $p_v = p_a = 10^{-4}$ (FD-Model), (c) w.r.t. p_a, with $p_v = p_c = 10^{-3}$ (SP-Model) and $p_v = p_c = 10^{-4}$ (FD-Model)

The function $P_{e,\nu}^{(+\infty)}(x)$ is defined provided that there exist a limit of $P_{e,\nu}^{(\ell)}(x)$ when ℓ goes to infinity. Condition (a) is required because $P_{e,\nu}^{(\ell)}(x)$ does not converge for some particular decoders and noise conditions, as shown in [18].

The functional threshold is defined as the transition between two parts of the curve representing $P_{e,\nu}^{(\ell)}(\alpha)$ with respect to α. The first part corresponds to the channel parameters leading to a low level of error probability, *i.e.*, for which the decoder can correct most of the errors from the channel. In the second part, the channel parameters lead to a high level of error probability, meaning that the decoder does not operate properly.. Note that there are two possibilities. If $L\left(P_{e,\nu}^{(+\infty)}, \bar{\alpha}\right) = +\infty$, then $\bar{\alpha}$ is a discontinuity point of $P_{e,\nu}^{(+\infty)}$ and the transition between the two levels is sharp. If $L\left(P_{e,\nu}^{(+\infty)}, \bar{\alpha}\right) < +\infty$, then $\bar{\alpha}$ is just an inflection point of $P_{e,\nu}^{(+\infty)}$ and the transition is smooth. Using the Lipschitz constant defined in this section, it is possible to characterize the type of transition for the error probability and discriminate between the two cases. We provide more details on our approach in the next section.

1.5.2 Functional Threshold Interpretation

Our goal is to use the functional threshold as a tool to discriminate between different FAIDs and design faulty decoders which are robust to faulty hardware. In order to do so, we need a precise understanding of the behaviors and the limits of the functional threshold. We present the analysis for regular $d_v = 3$ LDPC codes, and for the offset MS decoder [22] interpreted as a FAID. Table 1.2 gives the LUT of the VNU of the 7-level offset MS decoder considered for the analysis.

Fig. 1.2 (a) represents the asymptotic error probability $P_{e,\nu}^{(+\infty)}(\alpha)$ with respect to α for several values of p_v for the SP-Model with $p_c = p_a = 10^{-3}$. The circled points represent the positions of the functional thresholds obtained from Definition 1.5.1. When p_v is low, the threshold is given by the discontinuity point of the error probability curve. But when p_v becomes too high, there is no discontinuity point anymore, and the functional threshold is given by the inflection point of the curve. However, the inflection point does not predict accurately which channel parameters lead to a low level of error probability. Fig. 1.2 (b) represents $P_{e,\nu}^{(+\infty)}(\alpha)$ for several values of p_c with $p_v = p_a = 10^{-3}$. In all the considered cases, the functional threshold is given by the discontinuity point of the error probability curve. Fig. 1.2 (c) represents $P_{e,\nu}^{(+\infty)}(\alpha)$ for several values of p_a with $p_v = p_c = 10^{-3}$. In this case, not only the functional threshold is always given by the discontinuity point of the error probability curve, but the position of the functional threshold position does not seem to depend on the value of p_a.

Fig. 1.3 (a) shows the functional thresholds $\bar{\alpha}$ as a function of the hardware noise parameter at the VNU, p_v. For the SP-Model, we consider $p_c = p_a = 10^{-3}$, and for the FD-Model, $p_c = p_a = 10^{-4}$. When p_v is small, the value of $\bar{\alpha}$ decreases with increasing p_v. But when p_v becomes too large, we observe an unexpected jump in the $\bar{\alpha}$ values. The curve part at the right of the jump corresponds to the values p_v for which the functional threshold is given by the inflection point of the error probability curve. This confirms that when p_v is too large, the functional threshold does not predict accurately which channel parameters lead to a low level of error probability. Fig. 1.3 (b) shows the $\bar{\alpha}$ values as a function of p_c. For the $(3, 8)$-code and the FD-Model, we observe that when p_c becomes too large, the functional threshold also fails at predicting the convergence behavior of the faulty decoder. Finally, Fig. 1.3 (c) shows the $\bar{\alpha}$ values as a function of p_a. It confirms that the functional threshold value does not depend on p_a. This is expected, because the APP computation does not affect the iterative decoding process. As a consequence, the faulty APP computation only adds noise in the final codeword estimate, but does not make the decoding process fail.

We have seen that when the hardware noise is too high, it leads to a non-standard asymptotic behavior of the decoder in which the functional threshold does not predict accurately the convergence behavior of the faulty decoder. That is why we modify the functional threshold definition as follows.

Definition Denote $\alpha^\star$ the functional threshold value obtained from Definition 1.5.1. The functional threshold value is restated by setting its value to $\bar{\alpha}$ defined as

$$\bar{\alpha} = \begin{cases} \alpha^\star & \text{if } L\left(P_{e,\nu}^{(+\infty)}, \alpha^\star\right) = +\infty, \\ 0 & \text{if } L\left(P_{e,\nu}^{(+\infty)}, \alpha^\star\right) < +\infty. \end{cases} \tag{1.26}$$

Definition 1.5.2 eliminates the decoder noise values which lead to non-desirable behavior of the decoder. The functional threshold of Definition 1.5.2 identifies the channel parameters α which lead to a low level of asymptotic error probability and predicts accurately the convergence behavior of the faulty decoders. In this case, the functional threshold can be used as a criterion for the performance comparison of noisy FAIDs. This criterion will be used in the following for the comparison of FAIDs performance and for the design of robust decoders.

1.6 Design of FAIDs Robust to Faulty Hardware

Based on noisy-DE recursion and on the functional threshold definition, we now propose a method for the design of decoders robust to transient noise introduced by the faulty hardware. In Section 1.2, we have seen that the FAID framework enables to define a large collection of VNU mappings Φ_v and thus a large collection of decoders. The choice of the VNU mapping gives a degree of freedom for optimizing the decoder for a specific constraint. In [1], FAIDs were optimized for low error flor. Here, we want to optimize FAIDs for robustness to noise introduced by the faulty hardware.

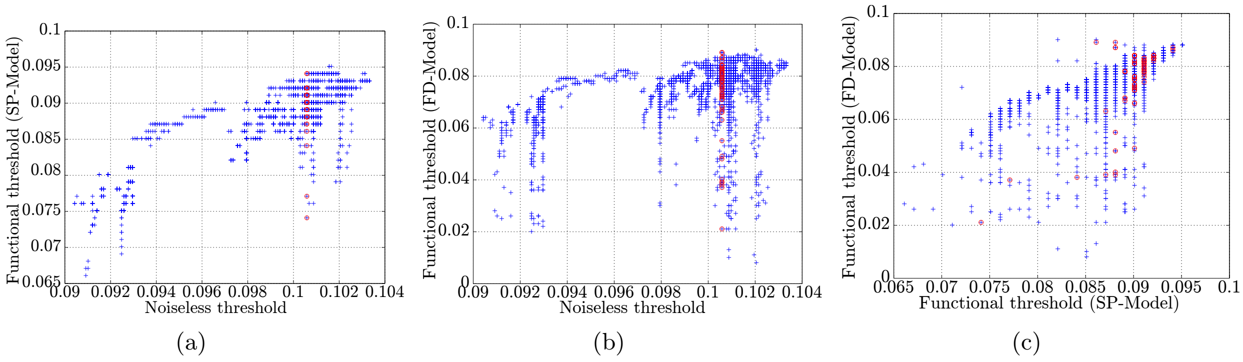


Figure 1.4: (a) Noiseless thresholds vs functional thresholds for the SP-Model ($p_v = p_c = p_a = 10^{-2}$) , (b) Noiseless thresholds vs functional thresholds for the FD-Model ($p_v = p_c = p_a = 5 \times 10^{-3}$) (c) Functional thresholds for the SP-Model ($p_v = p_c = p_a = 10^{-2}$) vs functional thresholds for the FD-Model ($p_v = p_c = p_a = 5 \times 10^{-3}$)

For message alphabet size $N_s = 7$, the number of possible FAIDs is equal to 530 803 988, which is too large for a systematic analysis. Instead, we rely on previous work on FAIDs, and start with a collection of $N_D = 5291$ FAIDs which correspond to column-weight tree codes selected from the trapping sets analysis presented in [1]. As a result of this selection process, each of the N_D FAIDs have both good noiseless threshold, and good performance in the error floor. We now perform a noisy-DE analysis on this set by computing, for each of the N_D FAIDs, the value of their functional threshold.

As an illustration, Fig. 1.4 (a) and (b) represent the functional thresholds with respect to the noiseless thresholds. For the SP-Model, the functional thresholds are computed for $p_v = p_c = p_a = 10^{-2}$, and for the FD-Model, $p_v = p_c = p_a = 5 \times 10^{-3}$. Although all the considered decoders have good noiseless threshold (between 0.09 and 0.104), a wide range of behaviors can be observed when the decoder is faulty. Indeed, for the SP-Model, the functional threshold values are between 0.065 and 0.095, thus illustrating the existence of both robust and non-robust decoders. In particular, even decoders with approximately the same noiseless threshold value (e.g. around 0.101) can exhibit different robustness. This is even more pronounced for the FD-Model, for which the functional threshold values are between 0.01 and 0.085. These observations illustrate the importance of selecting robust decoders to operate on faulty hardware and that a noiseless analysis is not sufficient to reach any useful conclusion.

We did also a performance comparison with noisy-DE and different error models, and Fig. 1.4 (c) represents the functional thresholds obtained for the FD-Model (for $p_v = p_c = p_a = 5 \times 10^{-3}$) with respect to the functional thresholds obtained for the SP-Model (for $p_v = p_c = p_a = 10^{-2}$). In this case also a large variety of behaviors can be observed. Indeed, only a small number of decoders are robust to both error models, while some of them are robust only to the SP-Model, and some others only to the FD-Model. This suggests that robustness to different error models may require different decoders.

Following these observations, we have selected four decoders from the set of N_D FAIDs. The first two ones denoted $\Phi^{(v,\mathrm{SP})}_{\mathrm{robust}}$ and $\Phi^{(v,\mathrm{FD})}_{\mathrm{robust}}$ are the decoderd have been selected such as to minimize discrepancy between noiseless and functional thresholds, for the SP-Model and the FD-Model respectively. Two other FAIDs $\Phi^{(v,\mathrm{SP})}_{\text{non-robust}}$ and $\Phi^{(v,\mathrm{FD})}_{\text{non-robust}}$ are selected to maximize the difference between noiseless and functional thresholds respectively for the SP-Model and for the FD-Model. The LUTs of $\Phi^{(v,\mathrm{SP})}_{\mathrm{robust}}$ and $\Phi^{(v,\mathrm{SP})}_{\text{non-robust}}$ are given in Table 1.3 and Table 1.4, and the LUTs of $\Phi^{(v,\mathrm{FD})}_{\mathrm{robust}}$ and $\Phi^{(\text{non-robust})}_{v,\mathrm{FD}}$ are given in Table 1.5 and Table 1.6. The four decoders will be considered in the following section to validate the asymptotic noisy-DE results with finite-length simulations.

Table 1.3: FAID rule $\Phi_{\text{robust}}^{(v,\text{SP})}$ robust to the faulty Hardware (SP-Model)

$\boldsymbol{m_1/m_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	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 1.4: FAID rule $\Phi_{\text{non-robust}}^{(v,\text{SP})}$ not robust to faulty Hardware (SP-Model)

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

Table 1.5: FAID rule $\Phi_{\text{(robust)}}^{(v,\text{FD})}$ robust to the faulty Hardware (FD-Model)

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

Table 1.6: FAID rule $\Phi_{\text{non-robust}}^{(v,\text{FD})}$ not robust to faulty Hardware (FD-Model)

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

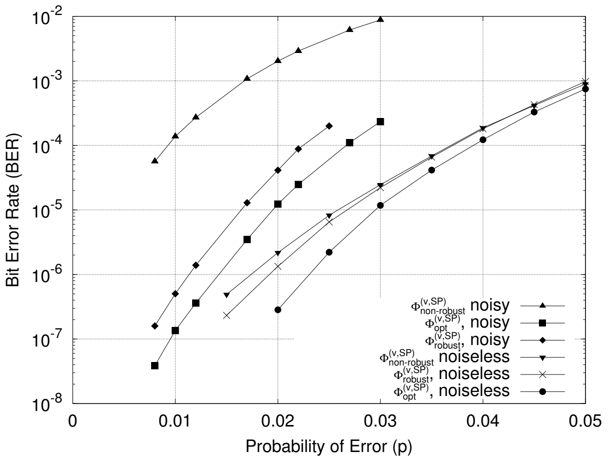
1.7 Finite Length Simulations Results

This section gives finite-length simulation results with the FAIDs $\Phi_{\text{robust}}^{(v,\text{SP})}$, $\Phi_{\text{non-robust}}^{(v,\text{SP})}$, $\Phi_{\text{robust}}^{(v,\text{FD})}$, and $\Phi_{\text{non-robust}}^{(v,\text{FD})}$ that have been identified by the noisy-DE analysis. For the sake of comparison, a fifth decoder denoted $\Phi_{\text{(opt)}}^{(v)}$ (Table 1.1) will also be considered. $\Phi_{\text{opt}}^{(v)}$ has been optimized in [1] for noiseless decoding with low error floor. In our simulations, the number of iterations is set to 100 and we consider the $(155, 93)$ Tanner code with degrees $(d_v = 3, d_c = 5)$ given in [23].

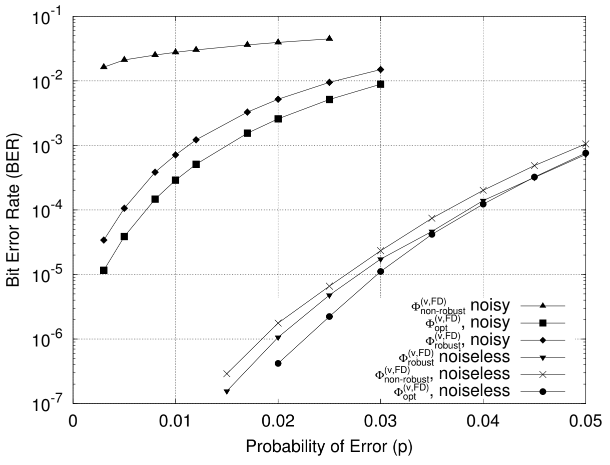
Fig. 1.5 (a) represents the Bit Error Rates (BER) with respect to channel parameter α and for the SP-Model. In the case of noiseless decoding, as $\Phi_{\text{opt}}^{(v)}$ has been optimized for low error floor, it performs better, as expected, than $\Phi_{\text{robust}}^{(v,\text{SP})}$ and $\Phi_{\text{non-robust}}^{(v,\text{SP})}$. But as $\Phi_{\text{robust}}^{(v,\text{SP})}$ and $\Phi_{\text{non-robust}}^{(v,\text{SP})}$ belong to a predetermined set of good FAID decoders, they also have good performance in the noiseless case.

We now discuss the faulty decoding case. For the SP-Model, we fix $p_v = p_c = p_a = 0.05$, and for the FD-Model, $p_v = p_c = p_a = 0.02$. We first see that the lower bound conditions of Proposition 1 are not satisfied here. Indeed, in our simulations, we considered an early stopping criterion, which halts the decoding process when the sequence estimated by the APP block is a codeword, while the results of Proposition 1 consider the averaged error probabilities at a fixed iteration number, and thus do not take into account the stopping criterion. We then see that the results are in compliance with the conclusions of the functional thresholds analysis. Indeed, when the decoder is faulty, $\Phi_{\text{robust}}^{(v,\text{SP})}$ performs better than $\Phi_{\text{opt}}^{(v)}$ while $\Phi_{\text{non-robust}}^{(v,\text{SP})}$ has a significant performance loss compared to the two other decoders. From Fig. 1.5 (b) we see that the same holds for the FD-Model in which case the error correction performance of the faulty decoders are much worse than for the SP-Model. The FD-Model makes decoders less robust to noise than the SP-Model, because with the FD-Model, not only the amplitudes, but also the signs of the messages can be corrupted by the noise. In particular, the non-robust decoder $\Phi_{\text{non-robust}}^{(v,\text{FD})}$ performs extremely poorly.

We now comment the results of Fig. 1.6. The code and decoder noise parameters are the same as before. In Fig. 1.6, the FD-Model with $p_v = p_c = p_a = 0.02$ is applied to $\Phi_{\text{robust}}^{(v,\text{SP})}$ and $\Phi_{\text{robust}}^{(v,\text{FD})}$, and the SP-Model with $p_v = p_c = p_a = 0.05$ is also applied to $\Phi_{\text{robust}}^{(v,\text{SP})}$ and $\Phi_{\text{robust}}^{(v,\text{FD})}$. We see that $\Phi_{\text{robust}}^{(v,\text{SP})}$ is robust for the SP-Model but not-robust for the FD-Model and that $\Phi_{\text{robust}}^{(v,\text{FD})}$ is robust for the FD-Model but not-robust for the SP-Model. These results are in compliance with the asymptotic analysis of Section 1.6 which shows that some decoders that are robust for one model are not necessarily robust for the other one.



(a)



(b)

Figure 1.5: (155, 93) Tanner Code, $d_v = 3$, $d_c = 5$, 100 iterations, (a) BER for the SP-Model, with $p_v = p_c = p_a = 0.05$, (b) BER for the FD-Model, with $p_v = p_c = p_a = 0.02$

To conclude, the finite-length simulations confirm that the functional threshold can be used to predict the performance of faulty decoders. Both the asymptotic analysis and the finite-length results demonstrate the existence of robust and non-robust decoders. They both illustrate the importance of designing robust decoders for faulty hardware and show that the design of robust decoders is dependant on the hardware error model.

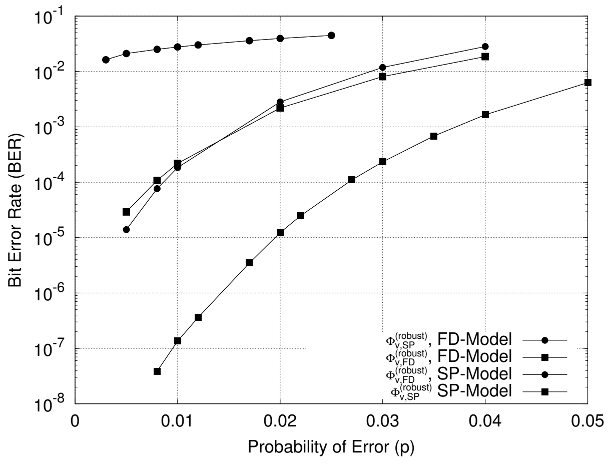


Figure 1.6: (155, 93) Tanner Code, $d_v = 3$, $d_c = 5$, 100 iterations, $p_v = p_c = p_a = 0.05$ (SP-Model) and $p_v = p_c = p_a = 0.02$ (FD-Model) For the legend, *e.g.*, $\Phi_v^{(\text{robust,SP})}$ FD is the decoder robust for the SP-Model applied to the FD-Model

1.8 Conclusion

In this chapter, we performed an asymptotic performance analysis of noisy FAIDs using noisy-DE. We introduced the functional threshold definition to characterize the behavior of noisy decoders. and provided an analysis of the behavior of the functional threshold and showed that under restricted noise conditions, it enables to predict the asymptotic behavior of noisy FAIDs. From this asymptotic analysis, we illustrated the existence of a wide variety of decoders robustness behaviors, and proposed a framework for the design of inherently robust decoders. The finite-length simulations illustrated the gain in performance when considering robust decoders.

Chapter 2

Faulty Stochastic Decoder

Abstract: *In this chapter, we analyse the performance of stochastic decoders on faulty hardware. We introduce two errors models to describe the noisy components of the stochastic decoder. Monte-Carlo simulations show that the two considered versions of the faulty stochastic decoder are inhenrently robust to noise introduced by the hardware. The simulations also show that the faulty stochastic decoder presents an increased robustness to hardware errors compared to the Min-Sum decoder.*

Part of the work presented in this Chapter has been published in C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Faulty Stochastic LDPC Decoders Over the Binary Symmetric Channel", *International Symposium on Turbo Codes and Iterative Information Processing (STIC)*, Bremen, Germany, August 2014 [P3]

2.1 Introduction

As part of the analysis carried out in Deliverable 3.1 for noisy Min-Sum (MS)-based decoders, we showed that the reliability of the most significant bit (sign bit) of exchanged messages has a critical impact on the error-correction performance, while more errors can be tolerated on less significant bits. Hence, protecting the sign bit of the exchanged messages may significantly improve the error correction capability of MS-based decoders. This was confirmed by the performance analysis of Chapter 1 on FAIDs which shows that faulty FAIDs are more robust under the SP-Model than under the FD-Model. However, it might be difficult to properly preserve the reliability of the most significant bit after an addition, since errors occurring on less significant bits can propagate to the next level and may affect the sign bit. It is therefore particularly useful to investigate LDPC decoders for which the information is represented using bits of similar significance. The stochastic decoder investigated in this section has this property.

In the stochastic decoder, probability beliefs are converted into streams of random bits, referred to as stochastic streams, and complex arithmetic operations are performed by simple bit-wise operations on the streams. Since stochastic computing is by its very nature random, if errors occur on a small number of bits of the stochastic stream, the resulting probability value (corresponding to the frequency of 1's in the stochastic stream) will be close to the correct value. Therefore, errors are expected to have a limited impact on the decoder performance. For instance, in [24] it has been shown that timing errors have a limited impact on the stochastic decoder performance. It is then important to study the behavior of the stochastic decoder in a more general hardware error scenario, in order to determine the amount of errors that can be tolerated, and which parts of the decoder can be built out of unreliable components.

In this chapter, we first present an overview of the Stochastic decoder and methods to improve its decoding performance, through the use of edge-memories and noisy-dependent scaling. We then propose several error models for its processing units. The error correction performance of the noisy Stochastic decoder is evaluated and compared to that of its noiseless counterpart. We further compare the performance of the Stochastic and MS decoders under faulty hardware setting, and show that the Stochastic decoder presents an increased robustness to hardware errors compared to the MS decoder.

2.2 Notation

In the following, we consider a codeword $\mathbf{x}$ transmitted over a binary-input memoryless channel, and we denote $\mathbf{y} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ the received sequence, where $\mathcal{Y}$ denotes the output alphabet of the channel. The following summarizes the notation that will be used throughout the chapter:

- N, the number of variable-nodes,
- M, the number of check-nodes,
- $n \in \{1, 2, ..., N\}$, a variable node of H,
- $m \in \{1, 2, ..., M\}$, a check node of H,
- $\mathcal{H}(n)$, the set of check-nodes connected to the variable-node n,
- $\mathcal{H}(m)$, the set of variable-nodes connected to the check-node m.

Two channel models will be considered in this chapter

BSC – the Binary Symmetric Channel – in this case $\mathcal{Y} = \{0, 1\}$ and $\mathbf{y}$ is obtained by flipping each bit of $\mathbf{x}$ with probability ε, referred to as the error or crossover probability of the channel.

BI-AWGN – the Binary-Input Additive White Gaussian Noise channel – in this case $\mathcal{Y} = \mathbb{R}$ and $\mathbf{y}$ is obtained by $y_n = (1 - 2x_n) + z_n$, where $1 - 2x_n \in \{+1, -1\}$ is the binary phase-shift keying (BPSK) modulation of the bit x_n, and z_n is the white Gaussian noise.

We further denote

- $P_n = \Pr(x_n = 1 \mid y_n)$, the probability of the transmitted bit x_n being equal to 1, conditional on the received value y_n.
- $L_n = \log \dfrac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)} = \log \dfrac{1 - P_n}{P_n}$, the LLR of the transmitted bit x_n, conditional on the received value y_n.

The following summarizes the notation that will be used throughout this chapter with respect to BP decoders

- γ_n, the *a priori* information of the decoder concerning variable-node n.
- $\alpha_{m,n}$, the message sent from variable-node n to check-node m.
- $\beta_{m,n}$, the message sent from check-node m to variable-node n.
- $\tilde{\gamma}_n$, *a posteriori* information provided by the decoder, concerning the variable node n.

2.3 Stochastic Decoder

The stochastic decoder is the stochastic implementation of the probability-domain BP decoder. Instead of propagating probability values between the nodes of the factor graph, the stochastic decoder converts these probabilities into *stochastic streams* [25]. A stochastic stream is a Bernoulli sequence of bits, and the probability value encoded in the stochastic stream is the probability of a bit in this sequence being equal to 1. Therefore, if $s(t)$ is a bit of a stochastic stream representing the probability p, one has:

$$\Pr(s(t) = 1) = p \tag{2.1}$$

This representation is not unique, since different stochastic streams may contain the same frequency of 1's, and therefore represent the same probability.

In the context of iterative LDPC decoding, stochastic computing presents two main advantages, as follows: (1) it allows using simple logic gates to perform complex arithmetic operations on continuous (probability) values, such as multiplications or divisions, thus significantly reducing the area of the computational components, and (2) only one bit is exchanged between computational components, hence reducing the number of wires in the circuitry and allowing high clock frequency [26].

Throughout this section, in addition to the notations of Section 2.2, the following notation will be used:

- $\mathbf{c}_n$, stochastic stream representing the probability of the transmitted bit x_n being equal to 1, conditional on the received value y_n. It will be referred to as *input stochastic stream.*
- $\mathbf{a}_{m,n}$, stochastic stream going from variable-node n to check-node m.
- $\mathbf{b}_{m,n}$, stochastic stream going from check-node m to variable-node n.

For stochastic decoding, decoding iterations are usually referred to as *decoding cycles.* At each decoding cycle, only one bit of each of the above stochastic streams is generated. Bits generated at decoding cycle ℓ will be denoted by $\mathbf{c}_n^{(\ell)}$, $\mathbf{a}_{m,n}^{(\ell)}$, $\mathbf{b}_{m,n}^{(\ell)}$. However, by a slight abuse of notation, we shall omit the superscript (ℓ) when no confusion is possible.

2.3.1 Input stochastic stream generation

Following the notation from Section 2.2, let $P_n = \Pr(x_n = 1 \mid y_n)$. In the implementation of stochastic decoders, these probability values are first quantized on q-bits. The quantized value, denoted in the sequel by p_n, is then used to generate the input stochastic stream $\mathbf{c}_n$. To generate a bit of $\mathbf{c}_n$, a q-bit number R is randomly generated and compared to the quantized probability value p_n. This operation is depicted in Figure 2.1.

In the following, the notation:

$$\mathbf{c}_n \leftarrow \mathbf{\Pi}(p_n)$$

will be used each time a new bit of $\mathbf{c}_n$ is generated.

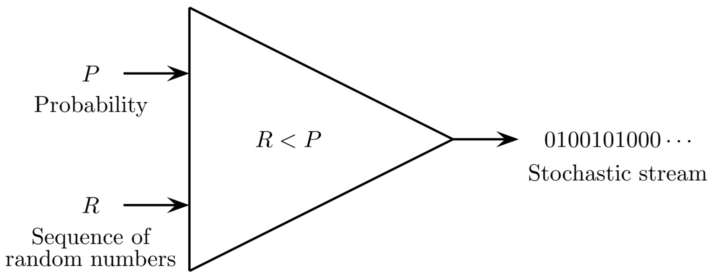


Figure 2.1: Stochastic Stream Generator

2.3.2 Check-node processing

Within the probability-domain BP decoder, messages exchanged over graph edges represent the probability of the incident variable-node being equal to 1. Within the stochastic decoder, these probability values are encoded in the stochastic streams $\mathbf{a}_{m,n}$ and $\mathbf{b}_{m,n}$. Now, consider a check-node m and a variable-node $n \in \mathcal{H}(m)$. Then $x_n = \text{XOR}_{n' \in \mathcal{H}(m) \setminus n} x_{n'}$. Since stochastic streams $\mathbf{a}_{m,n'}$ encode the probability of $x_{n'}$ being equal to 1, taking their XOR will give the probability of x_n being equal to 1. It follows that the check-node stochastic stream $\mathbf{b}_{m,n}$ can be computed by:

$$\mathbf{b}_{m,n} = \underset{n' \in H(m) \setminus n}{\mathbf{XOR}} \left(\mathbf{a}_{m,n'} \right) \tag{2.2}$$

Accordingly, the stochastic check-node processing unit is shown in Figure 2.2.

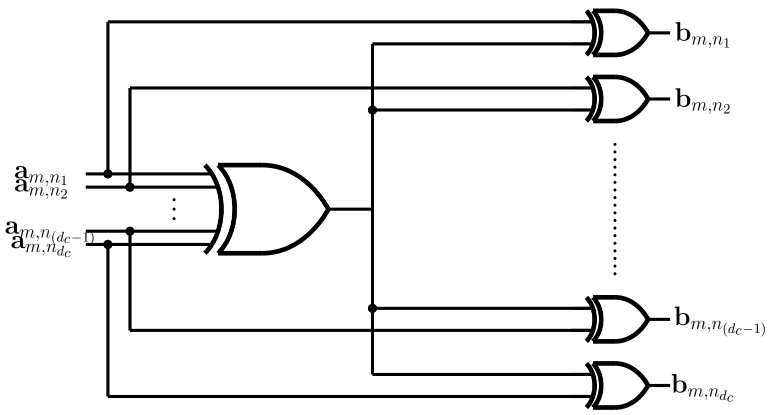


Figure 2.2: Stochastic check-node processing unit

2.3.3 Variable-node processing

In order to define the stochastic-domain implementation of the variable-node processing, we recall below its probability-domain counterpart:

$$\alpha_{m,n} = \frac{p_n \prod_{m' \in H(n) \setminus m} \beta_{m',n}}{\left(p_n \prod_{m' \in H(n) \setminus m} \beta_{m',n} \right) + \left((1 - p_n) \prod_{m' \in H(n) \setminus m} (1 - \beta_{m',n}) \right)} \tag{2.3}$$

In the stochastic-domain, the probability p_n is encoded by the input stochastic stream $\mathbf{c}_n$, while messages $\alpha_{m,n}$ and $\beta_{m,n}$ are encoded by stochastic streams $\mathbf{a}_{m,n}$ and $\mathbf{b}_{m,n}$. Moreover, Eq. (2.3) can be implemented by using inverters and JK flip-flops, as explained below.

Consider a JK flip-flop with inputs $\{\mathbf{j}^{(\ell)}, \mathbf{k}^{(\ell)}\}$ and output $\mathbf{q}^{(\ell)}$, at time ℓ, defined by the truth table given in Table 2.1.

$\mathbf{j}^{(\ell)}$	$\mathbf{k}^{(\ell)}$	$\mathbf{q}^{(\ell)}$
0	0	$\mathbf{q}^{(\ell-1)}$
0	1	0
1	0	1
1	1	$\overline{\mathbf{q}}^{(\ell-1)}$

Table 2.1: JK flip-flop truth table

The JK flip-flop can be viewed as a two states (0 and 1) Markov chain with a transition matrix T, defined by:

$$T = \begin{bmatrix} 1 - P_j & P_j \\ P_k & 1 - P_k \end{bmatrix}, \tag{2.4}$$

where $P_j = \Pr(\mathbf{j}^{(\ell)} = 1)$, $P_k = \Pr(\mathbf{k}^{(\ell)} = 1)$. Its steady state probability is defined by $P_q = \lim_{\ell\to\infty} \Pr(\mathbf{q}^{(\ell)} = 1)$, and can be derived from the eigenvector of T corresponding to the eigenvalue 1. The obtained probability is given by:

$$P_q = \frac{P_j}{P_j + P_k} \tag{2.5}$$

It can be seen that Eq. (2.3) can be obtained from Eq. (2.5), by substituting:

$$P_q = \alpha_{m,n} \tag{2.6}$$

$$P_j = p_n \prod_{m' \in H(n)\setminus m} \beta_{m',n} \tag{2.7}$$

$$P_k = (1 - p_n) \prod_{m' \in H(n)\setminus m} (1 - \beta_{m',n}) \tag{2.8}$$

In stochastic computing, multiplication can be implement by an AND gate; hence, P_j can be computed by taking the AND of stochastic streams $\mathbf{c}_n$, encoding the p_n value, and $\mathbf{b}_{m',n}$, encoding the $\beta_{m',n}$ values. Similarly, P_k can be computed by first inverting the above stochastic streams and then taking their AND. The stochastic variable-node processing unit is shown in Figure 2.3.

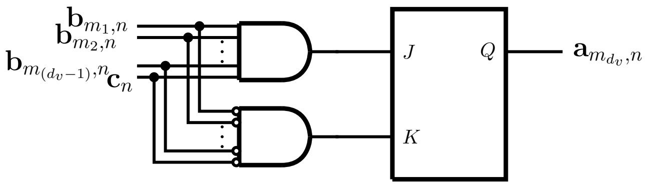


Figure 2.3: Stochastic variable-node processing unit

It can be seen that if all the incoming stochastic streams (including input stochastic stream) agree on the same bit value, the two inputs of the JK flip-flop are different. Hence, the outgoing stream $\mathbf{a}_{m,n}$, given by the output of the JK flip-flop, will take on the $\mathbf{j}$ value, which is equal to the common value of the incoming stochastic streams. In case that incoming stochastic streams do not agree on the same value, the outgoing stream $\mathbf{a}_{m,n}$ will hold the previous value. Hence:

$$\mathbf{a}_{m,n}^{(\ell)} = \begin{cases} \mathbf{c}_n^{(\ell)}, & \text{if } \mathbf{c}_n^{(\ell)} = \mathbf{b}_{m',n}^{(\ell)}, \forall m' \in \mathcal{H}(n) \setminus m \\ \mathbf{a}_{m,n}^{(\ell-1)}, & \text{otherwise} \end{cases} \tag{2.9}$$

In the first case, the variable-node is said to be in a *regular state*, while in the second case, it is said to be in a *hold state.*

2.3.4 A posteriori update

For stochastic decoders, each variable node estimates its value by using an up/down counter. For each variable node n, the corresponding counter θ_n is initialized to 0 at the beginning of the decoding process. During the decoding process, at each decoding cycle, θ_n is incremented for each message $\mathbf{a}_{m,n} = 1$, and decremented for each $\mathbf{a}_{m,n} = 0$. The transmitted bit is then estimated according to θ_n value:

$$\hat{x}_n = \begin{cases} 1, & \text{if } \theta_n > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

Note that the a posteriori update step in stochastic decoding is not equivalent to the a posteriori update step in BP decoding. To compute the stochastic stream encoding the value of the a posteriori information within BP, would require a circuit similar to the variable-node processing unit, with one additional entry (for the last incoming message). However, all the practical implementations reported in the literature make use of up/down counters, in order to reduce the hardware cost.

2.3.5 Stochastic decoding algorithm

The stochastic decoding is presented in Algorithm 1. The main steps of the stochastic algorithm are further detailed below.

1. *Initialization*:
   For each variable node, the probability $P_n = \Pr(x_n = 1|y_n)$ is computed and then quantized to a q-bit value, p_n. One bit of each input stochastic stream is generated by: $\mathbf{c}_n \leftarrow \mathbf{\Pi}(p_n)$. Finally, a-posteriori up/down counters are initialized to 0, and variable-to-check messages are initialized by $\mathbf{a}_{m,n} = \mathbf{c}_n$ (messages represent one bit from the corresponding stochastic streams).

2. *Check node processing*:
   At each cycle, each check node receives 1-bit messages from its neighbor variable-node. Check-to-variable messages $\mathbf{b}_{m,n}^{(\ell)}$ are computed by XOR-ing all the incoming messages, except the message received from variable node n.

3. *Variable node processing*:
   At each cycle, a new bit of the input stochastic stream is generated for each variable-node, which also receives 1-bit messages from neighbor check-nodes. If all the incoming messages agree (including the bit of the input stochastic stream), the node outputs their common value. Otherwise, it holds the previous value.

4. *Counter update*:
   At each cycle, a-posteriori up/down counters are update according to the values of the corresponding variable-to-check messages.

2.3.6 Improving the Stochastic decoder performance

The main issue with the above stochastic decoder is that it assumes that stochastic streams are independent Bernoulli sequences. However this is no longer the case when there are cycles in the factor graph. Besides, a low level of switching activity in the stochastic decoder can also cause groups of nodes to lock into fixed states which prevents proper decoding and leads to poor performance. This issue is called the *latching problem.* In order to increase the switching activity in the circuitry, the noise-dependent scaling method has been introduced in [25]. Moreover, several rerandomization methods have been proposed in the literature to reduce the correlation between stochastic bits. Edge-Memories, commonly used in the literature, will be used as rerandomization units in this work, due to their very good performance.

Noise-dependent scaling

The goal of the noise-dependent scaling is to ensure similar switching activity for different channel noise level.

For the BI-AWGN channel, the method consists of multiplying the LLRs of transmitted bits, denoted by L_n (see Section 2.2), by a factor proportional to the noise power, as follows:

$$L'_n = (2\kappa\sigma^2)L_n = (2\kappa\sigma^2)(\frac{2}{\sigma^2}y_n) = 4\kappa y_n \tag{2.11}$$

Algorithm 1 Stochastic Decoding

```
Input: $\mathbf{y} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet)    ▷ received word
Output: $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{0,1\}^N$    ▷ estimated codeword
Initialization
    for all $n = 1, \ldots, N$ do $P_n = \Pr(x_n = 1|y_n)$
    for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ do $\Gamma_n \leftarrow \mathbf{\Pi}(p_n)$
    for all $n = 1, \ldots, N$ do $\theta_n = 0$
    for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ do $\mathbf{a}_{m,n} = \Gamma_n$
Iteration Loop
    for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ do    ▷ CN-processing
        $\mathbf{b}_{m,n} = \underset{n' \in H(m) \setminus n}{\mathbf{XOR}} \left(\mathbf{a}_{m,n'}\right)$
    for all $n = 1, \ldots, N$ do    ▷ VN-processing
        $\Gamma_n \leftarrow \mathbf{\Pi}(p_n)$
        for all $m \in \mathcal{H}(n)$ do
            $\mathbf{a}_{m,n} = \begin{cases} \Gamma_n, & \text{if } \mathbf{b}_{m',n} = \Gamma_n \ \forall m' \in H(n) \setminus m \\ \mathbf{a}_{m,n}, & \text{otherwise} \end{cases}$
    for all $n = 1, \ldots, N$ do    ▷ Counter-update
        $\theta_n = \theta_n + \sum_{m \in \mathcal{H}(n)} (2\mathbf{a}_{m,n} - 1)$
    for all $n = 1, \ldots, N$ do    ▷ hard decision
        $\hat{x}_n = \begin{cases} 1, & \text{if } \theta_n > 0 \\ 0, & \text{otherwise} \end{cases}$
    if $\hat{\mathbf{x}}$ is a codeword then exit the iteration loop    ▷ syndrome check
End Iteration Loop
```

The probability of P_n, used to generate the input stochastic stream of the decoder, is then computed by:

$$P_n = \frac{1}{1 + \exp(L'_n)} \tag{2.12}$$

For the BSC, making P_n independent of the channel crossover probability ε is tantamount to replacing ε by a constant value μ, and therefore:

$$P_n = \begin{cases} 1 - \mu, & \text{if } y_n = 1 \\ \mu, & \text{if } y_n = 0 \end{cases} \tag{2.13}$$

Edge-memories

Edge Memories (EMs) are memory-based re-randomization units used to decorrelate bits in stochastic streams. Each EM consists of a S-bit shift register and is assigned to one edge of the decoder. EMs are initialized according to the probability p_n of the corresponding variable-node (*i.e.* each bit of an EM adjacent to variable node n is generated by $\mathbf{\Pi}(p_n)$).

Figure 2.4 describes the variable-node processing unit of the stochastic decoder with EMs. If all inputs agree on the same value ($\mathbf{c}_n = \mathbf{b}^{(\ell)}_{m',n}, \forall m' \in \mathcal{H}(n) \setminus m$), the outgoing stream $\mathbf{a}_{m,n}$ takes on their common value. In addition, this value is also stored in the EM. In case that inputs disagree a bit is randomly picked from the EM and sent to the adjacent check-node $\mathbf{a}^{(\ell)}_{m,n} = \mathrm{EM}(i)$, where i denotes a random EM location.

The VN-processing of the Stochastic decoder with EMs in detailed in Algorithm 2.

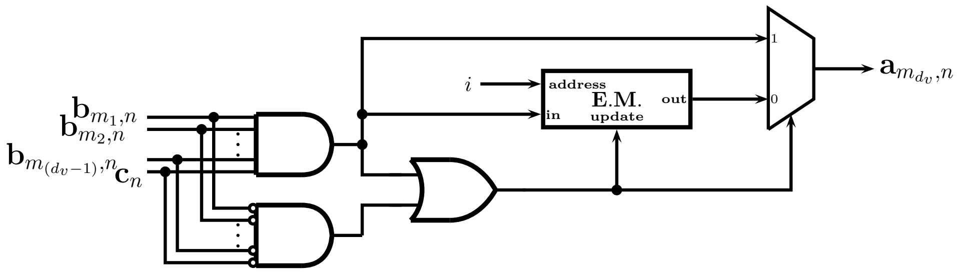


Figure 2.4: Stochastic variable-node processing unit with edge-memory

Algorithm 2 Stochastic Decoding with Edge-Memories

```
...                                                      ▷ same as stochastic decoding
Iteration Loop
    for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ do       ▷ CN-processing
        $\mathbf{b}_{m,n} = \underset{n' \in H(m) \backslash n}{\mathbf{XOR}} \left(\mathbf{a}_{m,n'}\right)$
    for all $n = 1, \ldots, N$ do                          ▷ VN-processing
        $\Gamma_n \leftarrow \mathbf{\Pi}(p_n)$
        for all $m \in \mathcal{H}(n)$ do

            if $\forall m' \in H(n) \backslash m$: $\mathbf{b}_{m',n} = \Gamma_n$ then
                $\mathbf{a}_{m,n} = \Gamma_n$
                $\Gamma_n \rightarrow \mathbf{EM}$
            else
                $\mathbf{a}_{m,n} = \mathbf{EM}(i)$
    ...                                                  ▷ same as stochastic decoding
End Iteration Loop
```

Remark: $\Gamma_n \rightarrow \mathbf{EM}$ means that the bit Γ_n is stored in the Edge-Memory and i is a random position in the Edge-Memory. When a new Γ_n bit is stored in the EM, bits in the EM are shifted first from left to the right, and then Γ_n is stored in the left-most position.

2.4 Error Models for the Faulty Stochastic Decoders

In this section we introduce two error models for the faulty stochastic decoders. In the first model, we suppose that hardware errors occur only in the edge-memories. In the second error model, the input stochastic stream generators, the variable-node processing units and the check-node processing units are also considered to be noisy.

2.4.1 Stochastic decoders with noisy edge-memories

EMs allows rerandomizing the stochastic streams, thus resulting in a significant improvement of the error correction performance. Despite these benefits, the main concerns regarding their use in practical applications are their number (equal to the number of edges in the graph) and size (several tens of bits for each EM). For long codes, the cost – in terms of area or energy consumption – of EMs might be prohibitive. To address this issue, EMs could be built from low-cost components, by trading reliability for area or energy savings. Therefore, the objective of the error model presented in this section is to analyze the impact of unreliable edge-memories on the decoder performance, and to further determine the level of noise in edge-memories that the stochastic decoder can tolerate.

Consider a faulty EM. Errors can occur in the EM when a bit is written in the memory and when a bit is read from the memory. Denote w the probability that an error occurs during the writing and

r the probability that an error occurs during the reading. The output of the EM will be in error if the selected bit was either written with error or read with error. Note that error during the writing and during the reading at the same address in the memory will compensate each other. Therefore, the error probability of the EM output is $p_{em} = w(1-r) + (1-w)r$.

Denote by $\mathbf{EM}(i)$ and $\mathbf{EM}_{\text{pr}}(i)$ the bit read at address i from the noiseless EM and from the noisy EM, respectively. The faulty EM is described as follows:

$$\mathbf{EM}_{\text{pr}}(i) = \begin{cases} \mathbf{EM}(i), & \text{with probability } 1 - p_{em} \\ \overline{\mathbf{EM}(i)}, & \text{with probability } p_{em} \end{cases} \tag{2.14}$$

The stochastic decoder with *noisy* EMs differs from Algorithm 2 only in the VN-processing step, as shown below:

```
for all $n = 1, \dots, N$ do                                    ▷ VN-processing
    $\mathbf{c}_n \leftarrow \mathbf{\Pi}(p_n)$;
    if $\mathbf{c}_n = \mathbf{b}_{m',n},\ \forall m' \in H(n)\backslash m$ then
        $\mathbf{a}_{m,n} = \mathbf{c}_n$;
        $\mathbf{c}_n \rightarrow \mathbf{EM}$;
    else
        $\mathbf{a}_{m,n} = \mathbf{EM}_{\text{pr}}(i)$;
```

2.4.2 Full noisy stochastic decoder

In this section we introduce a more general error model for stochastic decoders. We suppose that the processing units of the stochastic decoder are made of faulty components, except the counter update, hard decision and syndrome check steps. Note that syndrome check step is not a compulsory since the maximum number of cycles ensures the termination of the iteration loop.

In order to encompass all possible errors into a minimal number of error probability parameters, we inject errors at the output of the noisy processing units, as described in the following sections.

Noisy input stochastic stream generator

Recall that the input stochastic stream is generated by $\mathbf{\Pi}(p_n)$. The noisy input stochastic stream generator is defined by:

$$\mathbf{\Pi}_{\text{pr}}(p_n) = \begin{cases} \mathbf{\Pi}(p_n), & \text{with probability } 1 - p_{\mathcal{T}} \\ \overline{\mathbf{\Pi}(p_n)}, & \text{with probability } p_{\mathcal{T}} \end{cases}$$

where $p_{\mathcal{T}}$ is the error probability of the stochastic stream generator. It follows that $\mathbf{\Pi}_{\text{pr}}(p_n)$ behaves like the noiseless $\mathbf{\Pi}(p(1-p_{\mathcal{T}}) + (1-p)p_{\mathcal{T}})$.

Noisy check-node processing unit

Denote $\mathbf{C}$ the output of the noiseless check node unit (hence, $\mathbf{C} = \mathbf{b}_{m,n}$, for some (m,n)). The output of the noisy check node unit, denoted by $\mathbf{C}_{\text{pr}}$, is defined by:

$$\mathbf{C}_{\text{pr}} = \begin{cases} \mathbf{C}, & \text{with probability } 1 - p_c \\ \overline{\mathbf{C}}, & \text{with probability } p_c \end{cases}$$

Parameter p_c is referred to as the check-node error probability.

Noisy variable-node processing unit

Denote $\mathbf{V}$ the output of the noiseless variable-node unit (hence, $\mathbf{V} = \mathbf{a}_{m,n}$, for some (m,n)). The output of the noisy check node unit, denoted by $\mathbf{V}_{\text{pr}}$, is defined by:

$$\mathbf{V}_{\text{pr}} = \begin{cases} \mathbf{V}, & \text{with probability } 1 - p_v \\ \overline{\mathbf{V}}, & \text{with probability } p_v \end{cases}$$

Parameter p_v is referred to as the variable-node error probability.

Since each EM is part of a VN-processing unit, this error model also takes into account errors that occur during access to EMs. Therefore, for this error model, there is no need to use a specific parameter for the EM error probability.

2.5 Robustness Assessment of Noisy Stochastic Decoders

Density-evolution analysis cannot be performed for stochastic decoders, due to the fact that variable-to-check node messages are computed as functions of *dependent* random variables (therefore, the independence assumption does not hold even in the cycle free case). Precisely, in the VN-processing step, it can be seen that $\mathbf{a}^{(\ell)}_{m,n}$ is a function of $\mathbf{c}_n$, $(\mathbf{b}^{(\ell)}_{m',n})_{m' \in H(n)\setminus m}$, and $\mathbf{a}^{(\ell-1)}_{m,n}$. But $(\mathbf{b}^{(\ell)}_{m',n})_{m' \in H(n)\setminus m}$ and $\mathbf{a}^{(\ell-1)}_{m,n}$ are dependent random variables, since $\mathbf{a}^{(\ell-1)}_{m,n}$ depends on $(\mathbf{b}^{(\ell-1)}_{m',n})_{m' \in H(n)\setminus m}$, and the computation tree [27] of $\mathbf{b}^{(\ell-1)}_{m',n}$ is included in the computation tree of $\mathbf{b}^{(\ell)}_{m',n}$.

This dependency between $\mathbf{a}^{(\ell-1)}_{m,n}$ and $(\mathbf{b}^{(\ell)}_{m',n})_{m' \in H(n)\setminus m}$ has not been taken into account in the density evolution approach proposed in [28], which explains why the obtained threshold values in *loc. cit.* are even better than the Shannon limit. The problem remains, and is even compounded, with the use of edge-memories. Indeed when the inputs of the variable node disagree, the bit extracted from EMs is one of the values of the variable-node output at a previous non-hold state. The Markov-chain model for edge-memories proposed in [29] also neglects the dependency relation between messages sent on the same edge of the graph at different decoding iterations.

Therefore, to study the impact of hardware noise on the error correction capability of faulty stochastic decoders, Monte-Carlo simulations have been carried out for the $(3,6)$-regular LDPC code with length $N = 1008$ bits, available in [30]. Decoders have been simulated over the BSC. The following parameters will be used throughout this section:

- channel input probabilities are quantized on $q = 8$ bits,
- the noise dependent scale factor is $\mu = 0.12$,
- all decoders use 48-bit edge-memories,
- the maximum number of decoding cycles is set to 1000.

The performance of the noisy decoders is compared with that of their noiseless version and with the floating-point BP decoding with maximum number of iterations equals to 100 (which serves as reference).

2.5.1 Numerical results for the stochastic decoder with noisy EM

Figure 2.5 shows the BER performance of stochastic decoders with noisy edge-memories for five values of the EM error probability: $p_{em} \in \{0.0001, 0.005, 0.01, 0.02, 0.05\}$. The black curve and the blue curve represent the noiseless Belief Propagation decoder and the noiseless stochastic decoder respectively.

The results show that when $p_{em} = 0.0001$, the noisy stochastic decoder performs very close the noiseless decoder. A slight performance degradation can be observed for $p_{em} = 0.005$, which becomes more pronounced as the level of noise in EMs increases. However, for $0.005 \leq p_{em} \leq 0.01$, this

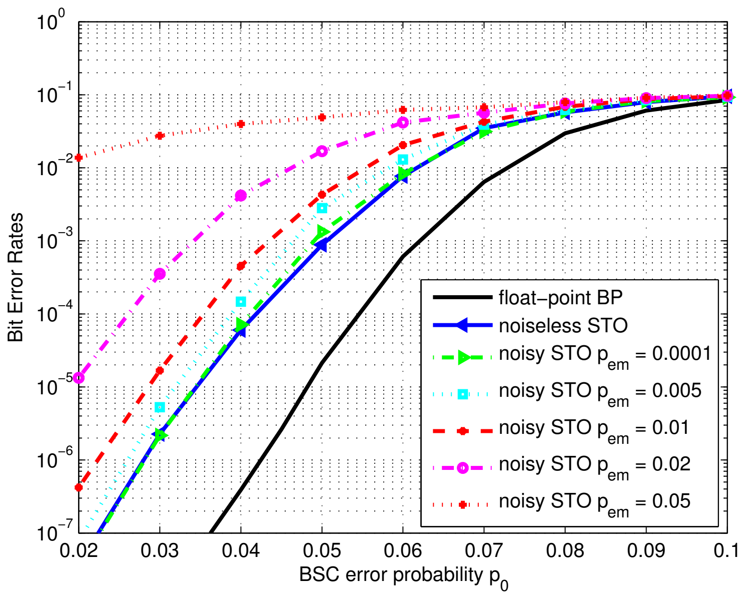


Figure 2.5: BER performance of the stochastic decoder with noisy edge-memories

performance degradation is quite limited, and the noisy stochastic decoder achieves effective error correction. Hence, we can conclude that the stochastic decoder is inherently robust to noise coming from EMs.

2.5.2 Numerical results for full noisy stochastic decoder

Figure 2.6 compares the performance of four full-noisy stochastic decoders, the BP decoder (black curve), and the noiseless stochastic decoder (blue curve). To better understand the impact of each noisy unit on the decoder error correction capability, the following noisy stochastic decoders have been simulated.

- A decoder with hardware noise coming only from input stochastic stream generators ($p_{\mathcal{T}} = 0.01$).
- A decoder with hardware noise coming only from check-node processing ($p_c = 0.01$).
- A decoder with hardware noise coming only from variable-node processing ($p_v = 0.01$). Recall that in this model, the variable-node processing also includes the edge memory.
- A decoder with all the above noisy units ($p_{\mathcal{T}} = p_c = p_v = 0.01$).

According to the results, the hardware noise coming from stochastic stream generators and check-node units does not degrade the performance of the decoder, even if the error probability considered are relatively high. The noise coming from variable node units leads to a perceivable performance loss. When $p_{\mathcal{T}} = p_c = p_v = 0.01$, the decoder exhibits the same performance than the decoder with only $p_v = 0.01$. This proves that noise from variable node units have the most significant impact on the overall decoder performance. Similar to the case of stochastic decoder with noisy EMs, we remark that the performance degradation is quite limited, and then conclude that the stochastic decoder is still very robust, even if all its processing units are noisy.

To determine which value of p_v can lead to results close to the noiseless decoder, further simulations have been carry out with only noisy variable node units and several values of the variable node error probability ($p_v \in \{0.001, 0.01, 0.05\}$). Figure 2.7 shows that when $p_v = 0.001$, the noisy stochastic decoder exhibits the same performance as the noiseless decoder.

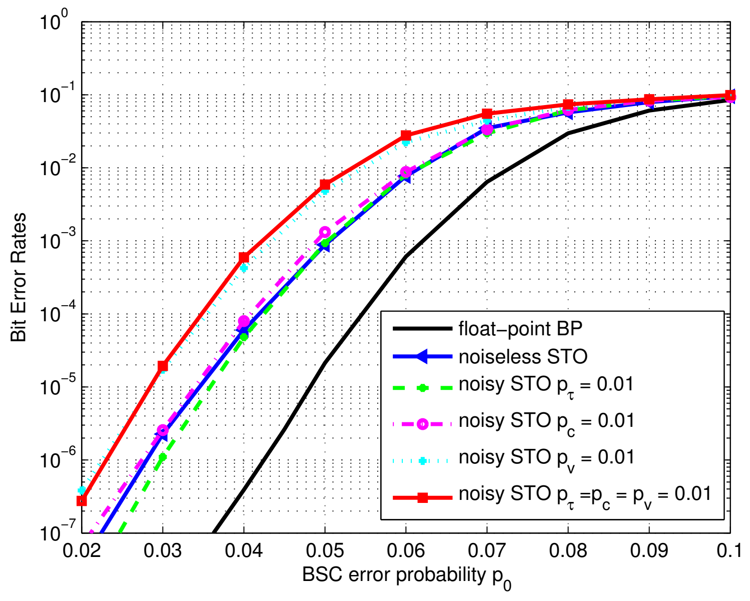


Figure 2.6: BER performance of the full-noisy stochastic decoder

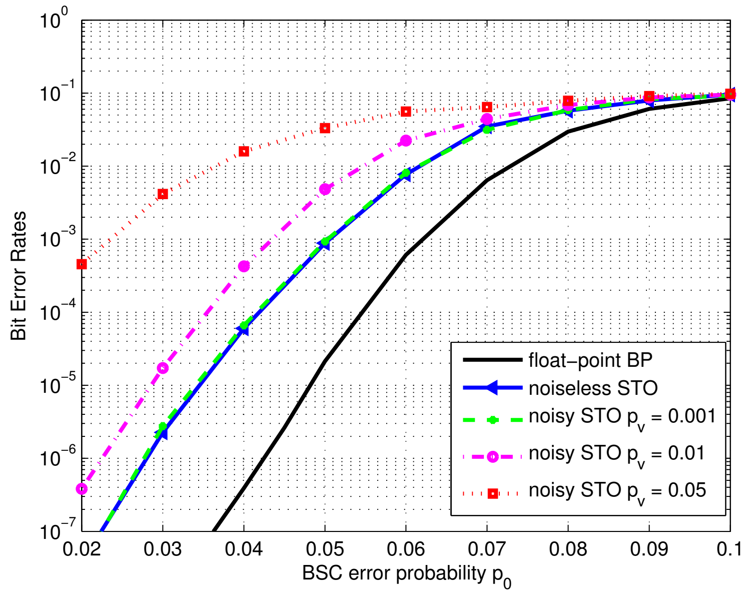


Figure 2.7: BER performance with only noisy Variable Node Units

2.5.3 Comparison between noisy stochastic and MS decoders

It is in general a difficult task to assess the performance of a decoder against another, under noisy hardware settings, except if they are made of similar hardware components. Although stochastic and MS decoders are made of dissimilar components, they also share a common component: the circuitry needed to implement the check-node processing unit within the stochastic decoder is exactly the same as the one needed to compute the sign of check-to-variable messages within the MS decoder.

We consider the MS decoder described in [18] with noiseless adder and comparator, but noisy XOR-operator (used only to compute the sign of check-to-variable messages), with error probability $p_{\text{xor}} = 0.004$. For check-nodes of degree $d_c = 6$, the corresponding error probability on the sign of check-to-variable messages is given $(1 - (1 - p_{\text{xor}})^{d_c - 1})/2 \approx 0.01$. Therefore, we also consider the stochastic decoder with noisy check-node processing unit, with error probability $p_c = 0.01$.

The comparison between the error-correction performance of the stochastic decoder with $p_c = 0.01$ and MS decoder with $p_{\text{xor}} = 0.004$ is shown in Figure 2.8. For comparison purposes, Figure 2.8 also shows the error-correction performance of the SCMS decoder with $p_{\text{xor}} = 0.004$, and full-noisy stochastic decoder with $p_{\mathcal{T}} = p_v = p_c = 0.01$.

It can be seen that the noisy MS decoder is largely outperformed by the two noisy versions of the stochastic decoder. We can therefore conclude the stochastic decoder is more robust to noisy hardware than the MS decoder, particularly since that all the processing units of the full-noisy stochastic decoder are faulty, while for the MS-decoder only the XOR-operator is considered to be noisy.

It can also be seen that the noisy SCMS outperforms the two noisy versions of the stochastic decoder. However this doesn't allow drawing clear conclusions, since the SCMS decoder has an advantage in that only the XOR-operator is considered to be noisy.

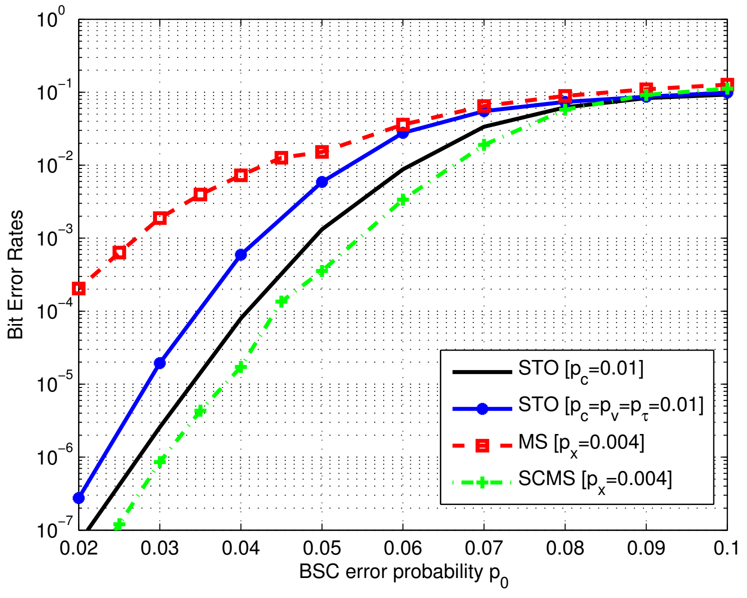


Figure 2.8: Comparison between noisy stochastic and MS decoders

2.6 Conclusion

In this chapter we investigated the performance of stochastic decoders in presence of hardware noise. We proposed error models for different processing units, and investigated two noisy versions of the stochastic decoder: stochastic decoder with noisy edge memories and full-noisy stochastic decoders. Due to the random nature of stochastic computation, stochastic decoders proved to be inherently robust to hardware noise, including noise coming from edge-memories. This is an important result, since the cost of EMs – in terms of area or energy consumption – might be prohibitive in practical implementations. Our result shows that EMs need not be reliable, and thus could be built from low-cost components, so as to trade reliability for area or energy savings. The performance of the noisy stochastic decoder has been also assessed against that of the noisy MS decoder, and we showed that the stochastic decoder presents an increased robustness to hardware errors compared to the MS decoder.

Chapter 3

Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder

Abstract: *We propose a gradient descent type bit flipping algorithm for decoding low density parity check codes on the binary symmetric channel. Randomness introduced in the bit flipping rule makes this class of decoders not only superior to other decoding algorithms of this type, but also robust to logic-gate failures. We report a surprising discovery that for a broad range of gate failure probability our decoders actually benefit from faults in logic gates which serve as an inherent source of randomness and help the decoding algorithm to escape from local minima associated with trapping sets.*

Work presented in this Chapter has been published in O.-A. Rasheed, P. Ivanis, and B. Vasic, "Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoders", *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487 - 1490, September 2014 [P4]

3.1 Introduction

It is now widely accepted that design of low-energy consumption Very Large Scale Integration (VLSI) systems must incorporate the fact that due to lower supply voltages and variations in the technological process, emerging nano-scale devices are inherently unreliable [31]. The reliable storage of data in a memory built of unreliable logic gates with transient failures can be achieved by employing low density parity check (LDPC) codes and simple bit-flipping (BF) decoding [8]. Density evolution of the sum-product algorithm (SPA) by Varshney [10], min-sum algorithm [15], FAID algorithm by Huang *et al.* [11] and Gallager B algorithm [11, 12] demonstrate robustness of these more complex decoding algorithms.

However, when only bit hard decisions are available, the decoder must rely on the BF or Gallager A/B algorithms. Since the performance of the BF decoder is typically inferior when compared to the Gallager-B algorithm it is of importance to try to close this gap by using more powerful variants of the BF decoder yet simpler than Gallager A/B algorithms. Although the previous results suggest a possibility that these decoders might be robust in the presence of gate failures, no analysis exists in the literature to prove or disprove this belief.

In this letter we consider fault-tolerant BF on the binary symmetric channel (BSC). The decoder we present is inspired by two decoders: Wadayama's Gradient Descent Bit Flipping (GDBF) [32] and Miladinovic and Fossorier's Probabilistic Bit Flipping (PBF) [33]. The GDBF was designed for the Additive White Gaussian Noise (AWGN) channel and shown to provide good balance between steady convergence and decoding speed. Inserting noise in all variable nodes in every particular iteration, as a means of improving the GDBF on the AWGN channel was proposed by Sundararajan *et al.* [34], and termed the noisy GDBF (NGDBF).

In PBF, code bits with a number of unsatisfied check sums larger than a fixed threshold are flipped with some probability, which is adapted throughout the iterations. By combining the ideas of [32] and [33] with some critical improvements following from our intuition on fault-tolerant decoders, we design a hard decision decoder, which we call the Probabilistic GDBF (PGDBF) decoder, resilient to logic gate failures.

3.2 Preliminaries

Let G denote the Tanner graph of an (N, K) binary LDPC code $\mathcal{C}$ of rate $R = K/N$, which consists of the set of N variable nodes V and the set of M check nodes C. The parity check matrix H is the bi-adjacency matrix of G. Two nodes in G are neighbors if there is an edge between them. The degree of a node v is the number of its neighbors and is denoted as d_v. Graph G is said to be variable-regular if all variable nodes in V have the same degree. The degree of a check node c is denoted d_c, and check-regular codes are defined analogously. The sets of neighbors of nodes v and c are denoted as $\mathcal{N}_v$ and $\mathcal{N}_c$, respectively.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ denote a codeword of $\mathcal{C}$ that is transmitted over a BSC with crossover probability α, where x_v denotes the value of the bit associated with variable node v and let the vector received by a decoder from the BSC be $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$. $\mathbf{e} = (e_1, e_2, \ldots, e_N)$ denotes the *error pattern* introduced by the BSC such that $\mathbf{y} = \mathbf{x} \oplus \mathbf{e}$, and $\oplus$ is the component-wise modulo-two sum.

We consider iterative decoders which at the ℓ-th iteration ($\ell \in [0, L]$, where L is the maximal number of iterations) produce the estimated codeword $\hat{\mathbf{x}}^{(\ell)}$ as an output. The GDBF algorithm is based on the calculation of the inverse function, defined as [32, Eqn. (6)]

$$\Delta_v^{(\ell)}(\chi, \eta) = \chi_v^{(\ell)}\eta_v + \sum_{c \in \mathcal{N}_v} \prod_{u \in \mathcal{N}_c} \chi_u^{(\ell)}, \tag{3.1}$$

where $\boldsymbol{\eta} = (-1)^{\mathbf{y}}$ and $\boldsymbol{\chi}^{(\ell)} = (-1)^{\hat{\mathbf{x}}^{(\ell)}}$ denote the "bipolar" versions of $\mathbf{y}$ and $\hat{\mathbf{x}}^{(\ell)}$. The estimate of a variable node v is initialized as $\chi_v^{(0)} = \eta_v$, and in the l-th iteration, the values $\Delta_v^{(\ell)}$ are calculated for all variable nodes v, and (in single-bit flipping mode of GDBF [32]) only the symbols with minimum value of the inverse function are inverted to obtain $\chi_v^{(\ell+1)}$.

3.3 GDBF Decoding Algorithm For BSC

The original GDBF was designed for the additive white Gaussian noise (AWGN) channel, where η are real valued vectors. To adapt it to the BSC, we first rewrite the polar-based inverse function in binary form. Since $\chi_v^{(\ell)} = 1 - 2\hat{x}_v^{(\ell)}$ and $\eta_v = 1 - 2y_v$, by using modulo-2 arithmetic, the inverse function can be simplified into

$$\Delta_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = 2 - 2(\hat{x}_v^{(\ell)} \oplus y_v) + d_v - 2 \sum_{c \in \mathcal{N}_v} \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(\ell)}. \tag{3.2}$$

For d_v-variable-regular codes, the above expression is minimized by maximization of the following modified inverse function

$$\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = \hat{x}_v^{(\ell)} \oplus y_v + \sum_{c \in \mathcal{N}_v} \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(\ell)}. \tag{3.3}$$

Let $b^{(\ell)}$ be the largest value of the modified inverse function at the ℓ-th iteration, i.e., $b^{(\ell)} = \max_v(\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}))$. If the flipping decision was wrong, the "flip messages" would propagate through the short cycles in G. To minimize this effect, it is reasonable to flip a small number of bits per iteration. In the case of the AWGN channel, this typically results in flipping of one bit per iteration (with minimum value of $\Delta_v^{(\ell)}(\hat{\chi}, \eta)$). In the BSC the range of the modified inverse function is restricted to the set of integer values $[0, d_v + 1]$, and usually more than one variable node satisfies the relation $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(\ell)}$, which has a negative impact on the algorithm convergence. On the other hand, as we show in the next section, the restrictions of the range of $\Lambda_v^{(\ell)}$ makes BSC version of GDBF decoder much less sensitive to logic gate failures in the decoder.

3.4 Fault-Tolerant PGDBF Decoder For BSC

In this section, we define formally the PGDBF decoder, and suggest a possible implementation in faulty hardware. According to Eqn. (3.3), for a (d_v, d_c)-regular code it is necessary to calculate the parities in the neighboring check nodes, by using d_c-input exclusive or (XOR) gates. An additional two-input XOR gate is required to check if the v-th bit of the current estimate is the same as the bit initially estimated from the channel. The value of $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y})$ is equal to the number of non-zero outputs of the XOR gates. Combinational logic at the variable nodes is based on a set of majority logic (MAJ) gates, each having $d_v + 1$ inputs and adaptable threshold. The output of the v-th MAJ gate in the ℓ-th iteration is non-zero only if the modified inverse function value is equal to the threshold value $b^{(\ell)}$. This threshold is the same for every variable node.

The new decoder is given in Algorithm 3, and Fig. 3.1 shows the hardware structure of a corresponding variable node processor. The main novelty we propose is the following. In the GDBF decoder for BSC $\Lambda_v^{(\ell)} = b^{(\ell)}$ was a sufficient condition for flipping, and the refreshed estimation is calculated as $\hat{x}_v^{(\ell+1)} = 1 \oplus \hat{x}_v^{(\ell)}$ if $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(\ell)}$ and $\hat{x}_v^{(\ell+1)} = \hat{x}_v^{(\ell)}$ if $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) < b^{(\ell)}$. It is stored in the register inside the decoder, and used for calculation of $\Lambda_v^{(\ell+1)}(\hat{\mathbf{x}}, \mathbf{y})$. In our algorithm, even if $\Lambda_v^{(\ell)} = b^{(\ell)}$, the observed bit $\hat{x}_v$ will not be flipped automatically - instead it will be flipped with a predefined probability p. In Algorithm 3 this is done by multiplying the flipping decision with the Bernoulli $B(1,)$ random variable R_v. As we show in the next section, this modification is critical for ensuring the resilience to gate failures. In hardware, it can be realized by adding to each variable node processor one AND gate and a generator of Bernoulli random variables R_v with $\Pr(R_v = 1) = p$.

Algorithm 3 Probabilistic GDBF Algorithm

```
Input: $\mathbf{y}$
$\forall v \in V: \ \hat{x}_v^{(0)} \leftarrow y_v$
$\mathbf{s}^{(0)} \leftarrow \hat{\mathbf{x}}^{(0)} H^T \ (\forall c \in C: \ s_c^{(0)} \leftarrow \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(0)})$
$\ell = 0$
while $\mathbf{s}^{(\ell)} \neq \mathbf{0}$ and $\ell \leq L$ do
    $\forall v \in V$: Compute $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}))$
    $b^{(\ell)} \leftarrow \max_v(\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y})))$
    $v = 1$
    while $v \leq N$ do
        if $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(\ell)}$ then
            $\hat{x}_v^{(\ell+1)} \leftarrow R_v \oplus \hat{x}_v^{(\ell)}$
        else
            $\hat{x}_v^{(\ell+1)} \leftarrow \hat{x}_v^{(\ell)}$
        $v \leftarrow v + 1$
    $\mathbf{s}^{(\ell+1)} \leftarrow \hat{\mathbf{x}}^{(\ell+1)} H^T$
    $\ell \leftarrow \ell + 1$
Output: $\hat{\mathbf{x}}^{(\ell)}$
```

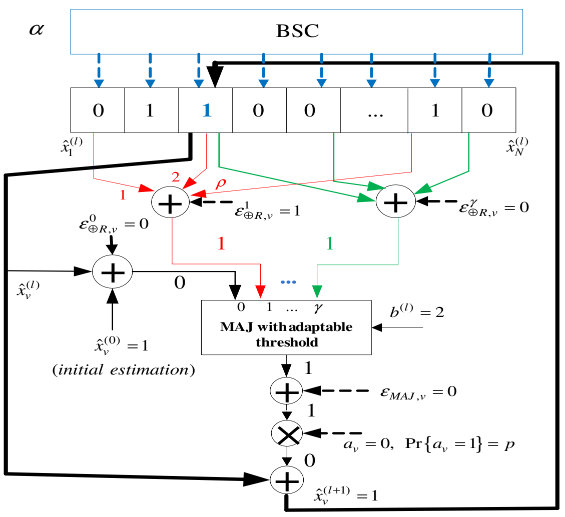

α
BSC
0
1
1
0
0
...
1
0
$\hat{x}_1^{(l)}$
$\hat{x}_N^{(l)}$
1
2
ρ
$\varepsilon^1_{\oplus R,v} = 1$
$\varepsilon^\gamma_{\oplus R,v} = 0$
$\varepsilon^0_{\oplus R,v} = 0$
1
1
$\hat{x}_v^{(l)}$
0
0 1 ... γ
MAJ with adaptable threshold
$b^{(l)} = 2$
$\hat{x}_v^{(0)} = 1$
(initial estimation)
1
$\varepsilon_{MAJ,v} = 0$
1
$a_v = 0,\ \Pr\{a_v = 1\} = p$
0
$\hat{x}_v^{(l+1)} = 1$

Figure 3.1: Illustration of the variable node processing unit, operations performed in the ℓ-th iteration. The randomness in the v-th variable node is modeled by adding a Bernoulli random variable $\varepsilon_{MAJ,v}$, and flipping the output of the c-th XOR is modeled by adding Bernoulli random variable $\varepsilon^c_{\oplus R,v}$.

The threshold $b^{(\ell)}$ may be initialized to the maximum value $(d_v + 1)$, and decremented in those iterations when all MAJ gate outputs are zero. When the output of at least one MAJ gate is not equal to zero, the threshold is set to $b^{(\ell)} = \max_v(\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}))$. The PGDBF results in a low complexity decoder that can be realized by using only XOR and MAJ logic gates, while the NGDBF decoder requires real-valued operations both for the inverse function calculation and for processing the AWGN samples inserted in variable nodes [34]. Moreover, in the NGDBF the threshold is adapted separately for every node, while in the PGDBF the threshold is the same for all variable nodes during the iteration.

3.5 Frame Error Rate Performance Analysis in the Presence of Hardware Failures

To assess the performance of the proposed decoding algorithm, we consider the canonical transient von-Neumann logic gate failure mechanism in which the failures in different gates and different time instants are independent and identically distributed. The failures manifest themselves as random bit flips at the gate outputs. All XOR gates have faulty probability $P_\oplus$, failures in the MAJ gates occur with probability P_{MAJ}, and the probability that a bit written in a register inside the decoder is incorrectly reconstructed is P_R. Computation of all other quantities is performed with perfect hardware.

The probability of flipping the output of a d_c-input XOR depends on the reliability of the memory cells connected to the gate inputs and the failure probability of the gate itself. Therefore, their combined effect can be modeled by flipping the output of the XOR gate with probability $P^c_{\oplus,R}$, which is obtained by using [12, Eqn. (3)] and given by

$$P^c_{\oplus,R} = \frac{1-(1-2P_R)^{q_c}}{2}(1-P_\oplus) + \frac{1+(1-2P_R)^{q_c}}{2}P_\oplus. \tag{3.4}$$

In the above expression, q_c corresponds to the number of inputs of the XOR gate connected with the register ($q_0 = 2$ for the gate that calculates $\hat{x}_v^{(\ell)} \oplus y_v$, and $q_c = d_c$ for parity check gates). The net result of this transformation is that $P^c_{\oplus,R}$ defines the probability of failure in the c-th check node, while P_{MAJ} corresponds to the variable nodes.

Now we present the numerical results of Monte Carlo simulations for a girth-8 regular codes with $d_v = 3$. For a given crossover probability of BSC, α, and the failure probabilities $P_{\oplus,R}$, P_{MAJ}, the frame error rate (FER) of the faulty PGDBF decoder is estimated and compared with the existing decoders.

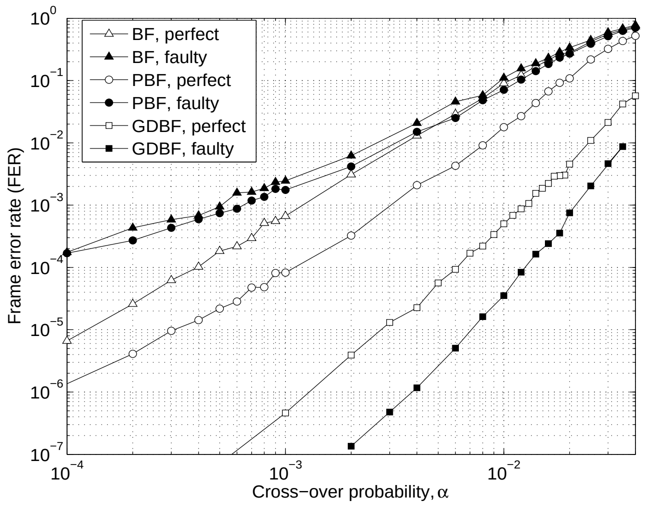


Figure 3.2: FER performance comparison for the $(155, 64)$ Tanner code, perfect: $P_{MAJ} = 0$, $P_{\oplus,R} = 0$, faulty: $P_{MAJ} = 10^{-3}$, $P_{\oplus,R} = 10^{-2}$.

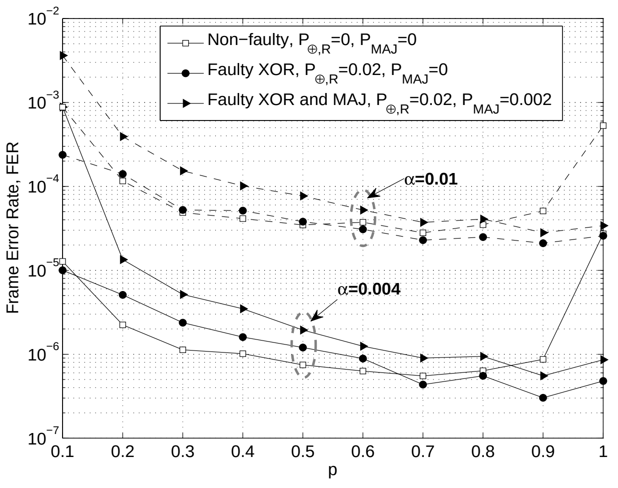


Figure 3.3: Impact of the parameter p on PGDBF optimized for the BSC realized in faulty hardware. The plot is for the $(155, 64)$ Tanner code, $\alpha = 4 \times 10^{-3}$, $\alpha = 10^{-2}$ and $L = 100$.

In Fig. 3.2, the FER performance of the $(155, 64)$ Tanner code is presented for the maximum number of decoding iterations $L = 100$. The results are presented for the BF, PBF and GDBF algorithms. In the case of non-faulty hardware in the decoder, GDBF algorithm results in lower FER values compared to the BF and PBF algorithms. As expected, the performances of faulty BF and PBF decoders are significantly degraded. For the case when $P_{\oplus,R} = 10^{-2}$, $P_{MAJ} = 10^{-3}$, the performance of the two are approximately the same. On the other hand, for the same failure rates, performance of the GDBF are improved compared to the non-faulty decoder case! This surprising effect is related to the finite set of possible values of $\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y})$. In such a case, the hardware failures can change its values in different variable nodes and the decoder could escape from a trapping set, as will be explained in Fig. 3.4. Therefore, the first goal is to optimize the parameter p in PGDBF to reduce the FER values for a wider range of the failure rates.

Fig. 3.3 shows the FER for the same code and fixed α, for various values of the parameter p. Numerical results are presented for the non-faulty case as well as for the case when $P_{\oplus,R} = 2 \times 10^{-2}$ and $P_{MAJ} = 0$ or $P_{MAJ} = 2 \times 10^{-3}$. In the non-faulty case it can be observed that the PGDBF decoder has the best performance for $p \approx 0.7$ while the best performances for the faulty case are

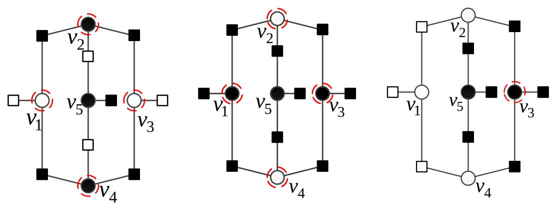


Figure 3.4: Error patterns: (a) $\ell = 1$, GDBF; (b) $\ell = 2$, non-faulty GDBF; (c) $\ell = 2$, faulty GBDF / non-faulty PGBDF.

obtained for $p \approx 0.9$. The optimal value of parameter p does not depend significantly on the BSC crossover probability, in contrast to NGDBF where the variance of the noise inserted to the variable nodes has to be approximately equal to the variance of the noise in the channel. For the case of perfect (reliable) logic gates, the PGDBF significantly reduces the FER compared to the GDBF (where $p = 1$). We have verified by simulation that the PGDBF decoder is almost insensitive to the logic gate failures in the range $P_{\oplus,R} < 3 \times 10^{-2}$ and $P_{MAJ} \leq 3 \times 10^{-3}$ for $p = 0.7$.

In Fig. 3.4 we present one motivating example based on an error pattern for a girth-8 code with $d_v = 3$, known to be uncorrectable by using the parallel BF algorithm, as it corresponds to a trapping set [35]. Fig. 3.4(a) illustrates this error pattern, where variable nodes are denoted by circles, the check nodes with squares, where the full markers correspond to binary one and empty to binary zero. The variable nodes marked with the dashed circles have maximum value $\Lambda_v^{(1)} = 2$ and these nodes are flipped in the GDBF algorithm. In the second iteration, shown in Fig. 3.4(b), we obtain $\Lambda_v^{(2)} = 4$ for the same symbol nodes as in the previous iteration, and further convergence is not possible. However, if v_1 is flipped before the second iteration of the GDBF algorithm, due to a hardware failure, we obtain the pattern in Fig. 3.4(c) and decoding is successful after one additional iteration.

A similar effect can be observed even in the perfect decoder, if we intentionally avoid the flipping of one bit that satisfies the necessary condition. Although four variable nodes in Fig. 3.4(a) satisfy it, due to the probabilistic nature of the algorithm, only v_2, v_3 and v_4 can be actually flipped and the second iteration of non-faulty PGDBF is illustrated in Fig. 3.4(c). In such a case, the probabilistic approach results in the same effect as the hardware failure that inverts one of the variable bits inside the trapping set. If $p = 3/4$ three out of four bits inside the length-8 cycle from the above example should be flipped on average, and if failures are present in the logic gates the value of the parameter p can be increased. Although the exact analysis is nontrivial, this seems to be a good explanation for the results shown in Fig. 3.3. In contrast to the PBF [33], the value of p is constant during the iterations and it is large enough so the decoding process is not slowed down significantly.

In Fig. 3.5 we present the FER performances for a $(732, 551)$ quasi-cyclic (QC) code [35]. In the presence of failures in the decoder, the performance is degraded for the PBF and Gallager-B decoders, but is improved for the GDBF ($p = 1$). For the case of PGDBF with $p = 0.8$, the performances are approximately the same for the faulty and non-faulty cases. Further simulations indicate that PGDBF has approximately the same performance if $P_{MAJ} \leq 1/(2N)$ and $P_{\oplus,R} \leq 5/N$.

3.6 Conclusion

By combining the ideas of GDBF and PBF, we have designed a hard decision decoder resilient to logic gate failures. In our approach, the probabilistic flipping is applied only to the variable nodes that satisfy a necessary condition for flipping. The corresponding flipping probability is fixed during iterations, the tuning of the parameters is simpler compared to previously proposed algorithms, and results in a minor increase of the decoding latency. Furthermore, we have shown that the proposed decoder not only has large immunity to gate failures but, surprisingly, can utilize the hardware failures to improve the decoding performance. We have considered some example QC LDPC codes and shown

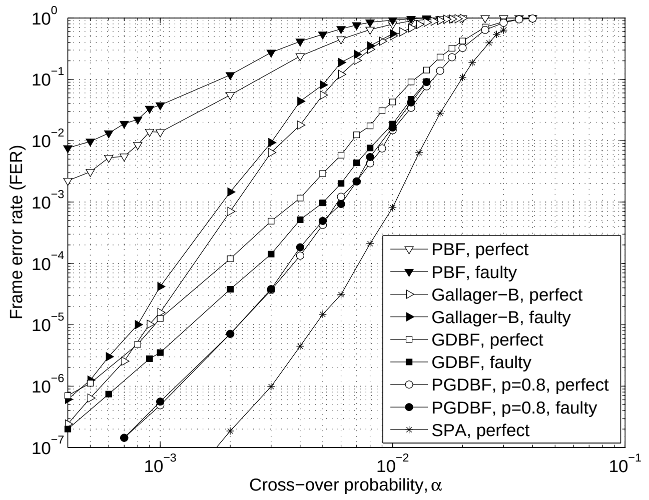


Figure 3.5: FER for PGDBF algorithm, BSC, QC$(732, 551)$, $L = 100$, perfect: $P_{MAJ} = 0$, $P_{\oplus,R} = 0$, faulty: $P_{MAJ} = 2 \times 10^{-4}$, $P_{\oplus,R} = 2 \times 10^{-3}$.

that the proposed algorithm is insensitive to hardware unreliability for a wide range of failure rates in combinational logic and memory cells in the decoder.

Chapter 4

Efficient Realization Of Probabilistic Gradient Descent Bit Flipping Decoders

Abstract: *In this chapter, several implementations of the PGDBF decoder introduced in Chapter 3. In Chapter 3, we have shown that using randomness in bit-flipping decoders can greatly improve the error correction performance. In this chapter, two models of random generators are proposed and compared through hardware implementation and performance simulation. A conventional implementation of the random generator through LFSR as a first design, and a new approach using binary sequences that are produced by the LDPC decoder, named IVRG, as second design. We show that both implementation of the PGDBF improve greatly the error correction performance, while maintaining the same large throughtput. However, the performance gain requires a large hardware overhead in the case of LFSR-PGDBF, while the overhead is limited to only 10% in the case of the IVRG-PGDBF.*

Work presented in this Chapter has been submitted to publication in K. Le, D. Declercq, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasic, "Efficient Realization of Probabilistic Gradient Descent Bit Flipping Decoders", *IEEE International Conference on Electronics Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015 (submitted) [P9]

4.1 Introduction

Low-Density Parity-Check (LDPC) codes have been intensively studied in the past several years due to their excellent performance under iterative decoding. Their practical iterative decoders vary from Belief Propagation (BP) [36] which offers the best error correction performance, but at the cost of intensive computation, to simple hard-decision algorithms such as Bit-Flipping (BF) decoders [37][38][33]. All iterative LDPC decoders share the same general concept of passing the information between Variable Nodes (VNs) and Check Nodes (CNs). The difference between BP-based decoders and BF-based decoders lies in the computation of iteratively passed messages. Due to their simple computation units, BF algorithms significantly reduce the hardware resources needed for implementation. The drawback of this simplification is a non-negligible performance loss compared to BP and its variants Min-Sum (MS), normalized MS [32]. As a consequence, many generalization of BF algorithms have been proposed, with the objective of reducing the performance loss while keeping the hardware complexity low as in weighted BF (WBF) [37], modified weighted BF (MWBF) [38], Gradient-Descent BF [32] algorithms.
Gradient Descent Bit Flipping (GDBF) algorithm for binary LDPC decoders have been first proposed by Wadayama et al [32]. This algorithm is derived from gradient descent formulation and its principle consists in finding the best suitable bit (or group of bits) to be flipped in the VN processing in order to maximize a pre-defined objective function. GDBF algorithm shows error correction performance

far better than other BF variants and very close to normalized MS algorithm [32]. Inspired by GDBF algorithm and the Probabilistic BF algorithm in [33], Chapter 3 proposed another variant of GDBF called Probabilistic GDBF (PGDBF) which has even better performance than the original GDBF. Instead of flipping all bits satisfying the gradient descent condition, PGDBF takes the flipping decision in a *probabilistic* manner. The results of Chapter 3 show that the randomness introduced in PGDBF makes this decoder superior to other BF decoding algorithms.
The performance improvement of PGDBF comes at the cost of extra hardware resources, since hardware-exhausted random generators blocks have to be implemented, for each and every VNs units. In this chapter, we present two different hardware implementations of probabilistic GDBF with a study of their trade-offs in term of performance versus hardware resource required. A comparison with the non-probabilistic GDBF decoder is also presented. The basic difference of the two proposed PGDBF realizations comes from the implementation methods of random binary sequence generators. The first method is conventional, and makes use of linear feedback shift register (LFSR) with a modification in using different length of registers. The second method, called Intrinsic-Value Random Generator (IVRG), uses the value of Check Node (CN) units and interprets these values as a random source of bits. In IVRG, a function G is designed to obtain, from the CN outputs, random binary sequences with a controlled probability distribution. This method, to the best of our knowledge, is original and has not been proposed in the literature. By using the intrinsic values which are already generated by the existing hardware block of the GDBF decoder, the IVRG-PGDBF significantly reduces the hardware resource needed.

The chapter is organized as follows. In section 4.2, general notations on LDPC codes and decoders are recalled and a short description of the PGDBF is made. We also highlight the main difference between the probabilistic GDBF and non-probabilistic GDBF which motivates the requirements of binary random generators in the hardware implementation. In section 4.3, the two methods for the hardware design of the random generators are presented. In section 4.4, our proposed global architecture of the PGDBF is presented and synthesis results are produced, for different cases of the random generator use. Additionally to our IVRG-PGDBF model, we also consider the case of a partial use of random generators in the LFSR-PGDBF approach, in which the random generators are applied to only part of the VN units. Finally, in section 4.5, we plot simulation results of the two PGDBF decoders implementation, and discuss the trade-off between error correction performance and hardware complexity. It is in particular shown that with only 9.7% of slice registers and 12.1% of slice LUTs overhead compared to the non-probabilistic PGDBF, the IVRG-PGDBF has performance results approaching the Min-Sum decoder.

4.2 Probabilistic Gradient Descent Bit flipping

An LDPC code is defined by a sparse parity-check matrix H with size (M,N), where $N > M$. A code word is a vector $\mathbf{x} = (x_1, x_2, ...x_N) \in \{0,1\}^N$ which satisfies $H.\mathbf{x} = 0$. We denote by $\mathbf{y} = \{y_1, y_2, ..., y_N\} \in \{0,1\}^N$ the output of a binary symmetric channel (BSC), in which the bits of the transmitted codeword $\mathbf{x}$ have been flipped with crossover probability α. The decoders presented in this chapter are dedicated to BSC channel. Let $\mathcal{N}_v$ denotes the set of CNs connected to the VN v, with connexion degree d_v. Let also define $\mathcal{N}_c$ as the set of VNs connected to the CN c, with connexion degree d_c.
In BF decoders, the value of variable nodes can change over the iterations, and we denote in this chapter by $\hat{x}_v^{(\ell)}$ the value of the variable node v at the ℓ-th iteration. We correspondly denote by $\delta_c^{(\ell)}$ the value of the parity check c at iteration ℓ.
The CN calculation in BF algorithms is defined by checking whether the parity check is satisfied or not. It can be written as: $\delta_c^{(\ell)} = \bigoplus_{v \in \mathcal{N}_c} \hat{x}_v^{(\ell-1)}$, ($\bigoplus$ is the bit-wise Exclusive-OR operation). In the case of gradient descent BF algorithms, a function called *inversion function*, is defined for each VN unit, and used to evaluate whether the value $\hat{x}_v^{(\ell)}$ should be flipped or not.
The original GDBF is designed for the Additive White Gaussian Noise (AWGN) channel. In GDBF

[32], only the VN having the smallest inversion function's value will be flipped, and sent for the next iteration. In Chapter 3, we introduced an inversion function to apply GDBF algorithm for the Binary Symmetric Channel (BSC). The inversion function for the BSC is given by

$$\Delta_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = 2 - 2(\hat{x}_v^{(\ell)} \oplus y_v) + d_v - 2 \sum_{c \in \mathcal{N}_v} \delta_c^{(\ell)}. \tag{4.1}$$

It can be modified as

$$\Lambda_v^{(\ell)}(\hat{\mathbf{x}}, \mathbf{y}) = \hat{x}_v^{(\ell)} \oplus y_v + \sum_{c \in \mathcal{N}_v} \delta_c^{(\ell)}. \tag{4.2}$$

The bits having the maximum value of $\Lambda_v^{(\ell)}$ in (4.2) are flipped.

In Chapter 3, the inversion function's value is an integer and varies from 0 to $d_v + 1$. Due to the integer representation of inversion function, many bits can be flipped in one iteration. This fact may induce a negative impact to the convergence of the algorithm as the analysis of Chapter 3 shows. To avoid this effect, the PGDBF has been proposed with the idea that, 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 by a pre-defined probability $p_v^{(\ell)}$, which could be different for each VN and each iteration. In this work, we restrict ourself in keeping $p_v^{(\ell)}$ constant for all iteration and all VNs (denoted as p hereafter). The details of the PGDBF are explained in Algorithm 3 in Chapter 3.
For the hardware implementation, it can be seen that the non-probabilistic GDBF and PGDBF have the same structure for the CN units and for the maximum-finder. The maximum-finder is in charge of finding the maximum value of inversion functions. In this work, we follow the conventional method which uses the binary comparator tree to implement the maximum-finder. In VN units of PGDBF, extra blocks which generates sequences of random bits, denoted as $R_v^{(\ell)}$ in Algorithm 3, are needed. Those blocks are the main difference between PGDBF and non-probabilistic GDBF and are required in order to improve the error correction performance. In Chapter 3, it is also shown that the optimum probability mass function for the random binary sequence is $p = 0.9$. Two solutions for the analysis and design of random generators having a fixed value of p are presented in the next section.

4.3 Analysis and design of random binary generators

4.3.1 LFSR random generator

The first design that we study is based on linear feedback shift registers, with controlled probability of getting zero or ones, that we consider for inclusion is each instantiated VNU. The generic architecture for the random binary sequence generator is not presented here, but is briefly described thereafter. We make use of LFSR with maximum length feedback polynomial to generate an integer number, and the generated number is compared with a threshold to decide if the new bit in the random sequence should be a 0 (higher than threshold) or 1 (lower than threshold). Two aspects are of interest when designing a variable threshold random binary sequence generator, first the period of the random sequence, second the granularity with which the threshold can be programmed. In the proposed architecture the period depends on the length of the LFSRs and the granularity from the number of LFSRs (i.e. bits) implemented. The chosen parameters guarantee a granularity of 2^{-8}. Moreover, having different length for each LFSR ensures a higher period than the period of the longest LFSR. In particular the least common multiple of all the periods $lcm([2^l - 1])$ with $l \in \{3 \dots 10\}$ is around 17 billions, which ensures that the binary sequence will appear random to the decoding process. Having different length LFSRs also reduces the total number of register required.

4.3.2 Intrinsic-value random generator

An alternative solution is presented in this section that reduces the cost of generating random binary sequences by means of subtituting all local RBSGs (one per VNU) with a global one. We name this new method *intrinsic-value random generator* (IVRG), which makes use of the value of the CNs inside

decoder as its inputs. In an LDPC iterative decoder, the values of the CNs depend both on the BSC crossover probability of α, the degree of check nodes d_c and the iteration number. 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 $p(\delta_c^{(\ell)} = 1) = F(\alpha, k, d_c)$ the probability that a CN is unsatisfied, as a function of the three mentioned parameters.
In this chapter, we will use only the CN values produced at the first iteration $\ell = 1$ in order to generate sequences of random bits. At the first iteration, the probability mass function is given by $p(\delta_c^{(1)} = 1) = F(\alpha, 1, d_c) = \frac{1}{2} - \frac{1}{2}(1-2\alpha)^{d_c}$. Figure 4.1 shows the probability $p(\delta_c^{(1)} = 1)$ versus α when the CN degree d_c changes. In order to control the random generator probability p, we propose to use a function G of the CN values $G(\delta_{c1}^{(0)}, \delta_{c2}^{(0)}, ..), c1, c2, ... \in [1, M]$ that controls the desired probability p. We briefly describe the function G in the following.
Let δ_{c1} and δ_{c2} be two binary random variables with $p(\delta_{c1} = 1) = p(\delta_{c2} = 1) = p'$, it can be proved that $p(\delta_{c1} \text{ OR } \delta_{c2} = 1) = 2p' + p'^2 > p'$ and $p(\delta_{c1} \text{ AND } \delta_{c2} = 1) = p'^2 < p'$. More specifically, $p(\delta_{c1} = 1) = p(\delta_{c2} = 1) = p' = \frac{1}{2} - \frac{1}{2}(1 - 2\alpha)^{d_c}$, $p(\delta_{c1} \oplus \delta_{c2} = 1) = \frac{1}{2} - \frac{1}{2}(1 - 2\alpha)^{2d_c}$. Using these transformations of probability, and a function G implemented as described in figure 4.2, we can transform the CN output sequence into a longer binary pseudo-random sequence with a desired probability p. 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 input-selectable-OR gates through an interconnexion network in order to ensure randomness of the output sequence. The value of crossover probability triggers the selectable-OR gates, in order to control the value of p. The more precision is put on α, the finer is the control on p, but at the cost of larger hardware resource.
This IVRG has been realized and verified, for the case of α that triggers the IVRG output stored on 2 bits. As a result, in running time of the PGDBF decoder, the probability of the IVRG output is not exactely tuned along the iterations and varies around the target value $0.88 < p < 0.92$.

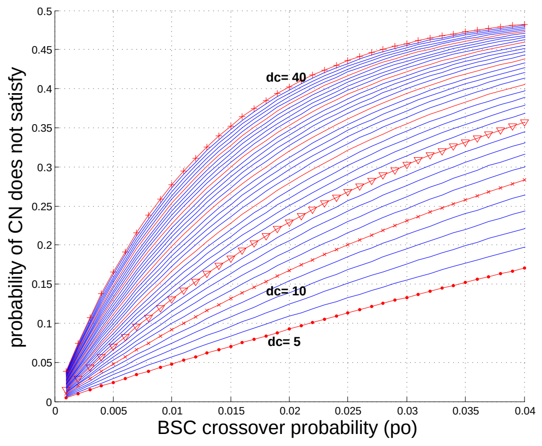


Figure 4.1: Statistics of the CN values as a function of the BSC crossover probability.

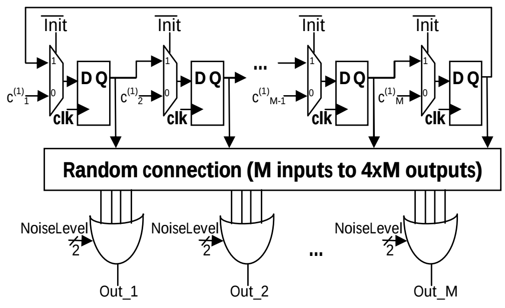


Figure 4.2: Architecture of Intrinsic-valued Random Generator

4.4 Global architecture of Probabilistic GDBF

The top level architecture of the decoder is presented in Figure 4.3. This architecture differs from the generic LDPC decoder architecture in several aspects. First, the presence of a global block that takes inputs from all VNUs (the Λ_v) and computes the maximum, and second the presence of binary random generators. The LFSR approach can be seen as a distributed random generator due to the fact that it is implemented inside every VN unit.
The complexity of the interconnection network depends on the type and size of LDPC code used as well as the chosen level of parallelism. These aspects have been widely discussed in the literature [39] and are not discussed here. Implementation of the RG have been discussed in the section 4.3.

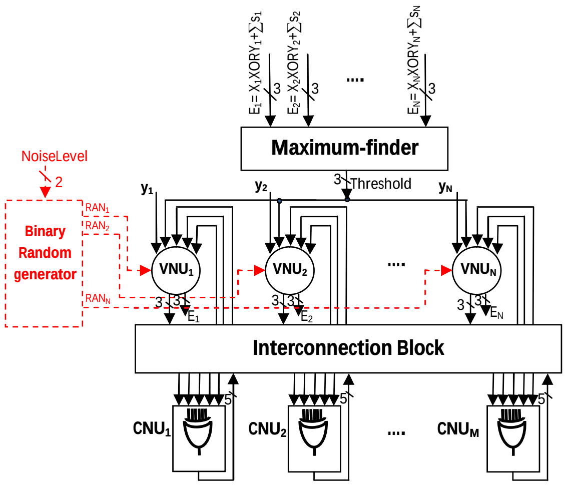


Figure 4.3: Global architecture of PGDBF compared to the original GDBF

We have made the synthesis of the different solutions for the case of a small LDPC code that has been proposed in the literature [40]: a regular, quasi-cyclic LDPC code with regular connexion degrees $d_v = 3$ and $d_c = 5$, with codeword length $N = 155$, called the *Tanner code*. Table 4.1 shows the hardware resources needed to implement the two different PGDBF structures. As benchmarks, the resources for the non-probabilistic GDBF and 6 bits Min-Sum decoder are shown as well. The maximum frequency and the estimated throughput have been obtained from an implementation using FPGA Xilinx virtex 6 of 40nm technology, after place and route. For the throughput calculation, we use the following definition: $Throughput = f_{max} * N/(I_{aver} * S)$ where f_{max}, I_{aver}, S are respectively the maximum frequency, the average iteration number and the number of clock cycles needed for one iteration. We obtained $S = 1$ for the PGDBF algorithms and $S = 10$ for the offset Min-Sum.
The IVRG-PGDBF needs an additional 92 1-bit registers (9.7% overhead) compared to the non-probabilistic while the LFSR-PGDBF needs 8215 1-bit registers overhead (868.4% overhead). This large overhead emphasize the advantage of IVRG over LFSR in terms of implementation. Comparing the Slice LUTs required, the IVRG-PGDBF requires 261 more slices than the non-probabilistic (12.1%) and this number for LFSR-PGDBF is 1394 (64.8%). The extra complexity brought by the RG implementation has moreover a negligible impact on the obtained throughput (less than 2%) in all PGDBF impementations. We can also see that the offset min-sum decoder is far more complex than the BF type decoders, and cannot compete in terms of decoding speed.

In order to reduce the hardware resources used in the LFSR-PGDBF we propose to apply the RG only in a subset of the VN, and not everywhere. The reason for this study is that putting RGs in all the variable units might not be necessary in order to obtain good decoding results, especially in the case of Quasi-cyclic LDPC codes. We report in table 4.2 the synthesis results for different fractions of VNs using LSFR-RG, from 0% (non-probabilistic) to 100%. As expected, the complexity in registers and slices grows linearly with the number of LSFR-RG considered. Even with only 20% of VNs incorporating the LSFR-RG, the complexity is larger than the one of the IVRG approach (see

Table 4.1: Hardware and throughput estimation for PGDBF with different RG implementations and for offset Min-Sum

	1-bit Register	Slice LUTs	F_{max} (MHz)	Throughput (Mbps)
Non-Probabilistic GDBF	946	2151	132.721	4114.3
PGDBF with IVRG	1038	2412	132.721	4114.3
PGDBF with LFSR	9161	3545	135.56	4202.36
offset min-sum (6 bits)	13694	15350	237.185	197.5

table 4.1). The performance results of these different cases are reported in the next section.

Table 4.2: Hardware and throughput estimation for PGDBF with different number of LFSR in PGDBF

	1-bit Register	Slice LUTs	F_{max} (MHz)	Throughput (Mbps)
Non-Probabilistic GDBF	946	2151	132.721	4114.3
PGDBF with LFSR in 20% VNs	2589	2429	133.886	4150.466
PGDBF with LFSR in 40% VNs	4232	2708	134.426	4167.206
PGDBF with LFSR in 60% VNs	5875	2987	132.117	4095.627
PGDBF with LFSR in 80% VNs	7518	3266	134.21	4160.51
PGDBF with LFSR in 100% VNs	9161	3545	135.56	4202.36

4.5 Numerical results

Figure 4.4 shows the frame error rate of the PGDBF algorithms and non-probabilistic GDBF as a function of the channel crossover probability. The two solutions proposed in this chapter for PGDBF algorithms produces a significant gain in performance comparing to non-probabilistic GDBF. The IVRG-PGDBF shows a performance loss in the error floor region (flattening) compared to the LFSR-PGDBF. However, the performance in the waterfall region are strictly similar. This performance loss in the error floor could come from the correlation induced by the imprecise random generation implemented with IVRG. We will continue the characterization of the IVRG approach in future works. We can also notice that the partial use of LFSR (20%) is not sufficient to obtain good results. As a conclusion the IVRG-PGDBF appears as the best performance vs. complexity trade-off for the implementation of the PGDBF decoder. As expected, hard decision decoder are still far from soft-decision ones (min-sum), as can be seen in figure 4.4.

4.6 Conclusion

In this chapter, several implementations of the PGDBF decoder for LDPC codes have been proposed. A conventional implementation of the random generator through LFSR as a first design, and a new approach using binary sequences that are produced by the LDPC decoder, named IVRG, as second design. We showed that both implementation of the PGDBF improve greatly the error correction performance compared to the non-probabilistic version, while maintaining the same large throughtput. However, the performance gain requires a large hardware overhead in the case of LFSR-PGDBF, while the overhead is limited to only 10% in the case of the IVRG-PGDBF, which appears then as a promising solution for very high throughtput application.

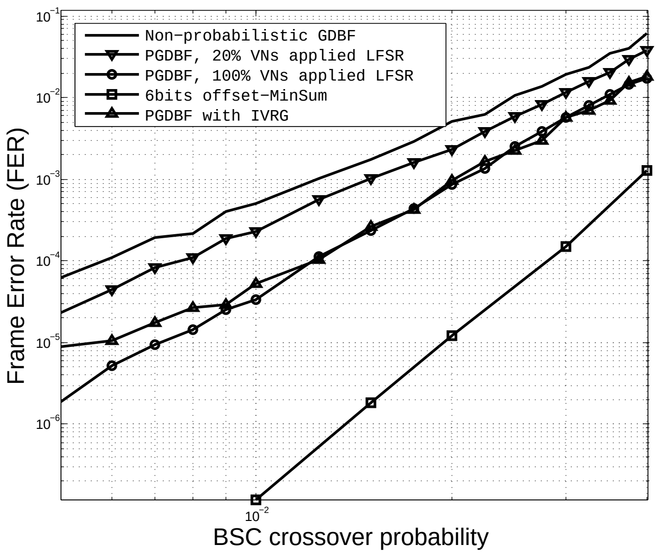


Figure 4.4: FER performance comparison of the different decoders on the (N=155,K=64) Tanner code

Chapter 5

Faulty Encoding of Low Density Parity Check Codes

Abstract: *In the standard non-faulty framework, several encoding solutions and code constructions have been proposed in order to reduce the encoding complexity [41–43]. In this chapter, we review the encoding solutions and code structures proposed in [41–43] and analyze their robustness to hardware errors. We then eliminate the clearly non-robust encoding solutions and evaluate the performance of the remaining solutions with Monte-Carlo simulations.*

5.1 Introduction

Until now, we have focused on the performance of LDPC decoders on faulty hardware. In the previous chapters, we have analyzed the performance of FAIDs, stochastic, and PGDBF decoders on faulty hardware, and we have designed robust LDPC decoders. In this chapter, we investigate the robustness of the LDPC encoding part.

When the hardware is assumed perfect, many efforts have been made for the design of low complexity encoders. Richardson and Urbanke [41] propose a method for the construction of low complexity encoders from any LDPC parity check matrix. Li et al [44] also propose low complexity encoder architectures for Quasi-Cyclic (QC) codes. On the other hand, Zig-Zag codes [42], Irregular Repeat Accumulate (IRA) codes [45], and Low Density Generator Matrix (LDGM) codes [43] are particular code constructions that guarantee low encoding complexity.

When the hardware is faulty, the question that comes is on the robustness of the former encoding solutions to noise introduce by the hardware. This problem has not been studied much so far, and to the best of our knowledge, [46] is the only work considering noisy LDPC encoding. In [46], Hachem et al considered a noiseless transmission channel and determined the number of encoding errors that can be tolerated such that the original codeword can be recovered by a perfect decoder. However, [46] only considers systematic encoding and does not analyze the other existing encoding solution.

In this chapter, we want to evaluate the robustness of existing encoding solutions [41–45] to hardware errors. We also want to determine whether it is possible to construct robust LDPC encoders. To this aim, Section 5.2 introduces the statistical model we consider for the hardware errors. Section 5.3 reviews the encoding solutions proposed in [41–45] and analyze their robustness to hardware errors. To finish, Section 5.4 eliminates the clearly non-robust encoding solutions and evaluates the performance of the remaining encoding solutions with Monte-Carlo simulations.

5.2 LDPC Codes and Encoding Error Models

In this section, after introducing notations for LDPC encoders and decoders, we present the error model we consider for the faulty hardware.

5.2.1 Notations

Denote by G the binary generator matrix of size $n \times (n-m)$ of an LDPC code. Denote $\mathbf{u} = [u_1, \dots, u_{(n-m)}]$ the information sequence of length $(n-m)$. The codeword $\mathbf{x} = [x_1, \dots, x_n] \in \{0,1\}^n$ can be constructed from $\mathbf{u}$ and G as

$$\mathbf{x} = G\mathbf{u}. \tag{5.1}$$

Denote by H the parity check matrix of size $n \times m$ of the code. If $\mathbf{x}$ is a codeword, it verifies

$$H^T\mathbf{x} = \mathbf{0}. \tag{5.2}$$

Equations (5.1) and (5.2) lead to the conditions $H^T G\mathbf{u} = \mathbf{0}$ and

$$H^T G = [0]_{m\times(n-m)} \tag{5.3}$$

where $[0]_{m\times(n-m)}$ is the matrix full of zeros of size $m \times (n-m)$.

Usually, for the construction of LDPC codes with good decoding performance, the effort is on the design of the parity check matrix H which is used at the decoder, see *e.g*, [47,48] . Once the parity check matrix H is obtained, the corresponding encoder has to be constructed. A first option is to construct the generator matrix G of the code from (5.3) and to compute the codeword $\mathbf{x}$ from (5.1). However, the generator matrix G is not sparse in general. As a consequence, the encoding operation (5.1) has very high complexity in $O(n^2)$. That is why several encoding algorithms have been proposed in order to reduce the encoding complexity [36, 41]. In addition, particular code constructions such as Zig-Zag codes [42] and LDGM codes [43] have been proposed to guarantee low encoding complexity.

Here, we assume that the encoding is realized on faulty hardware, which introduces errors during the encoding. In order to evaluate the robustness of the existing encoding solutions [41–45] to hardware errors, we first introduce the error model we consider for the faulty hardware. We then review the existing encoding solutions [41–45] and evaluate their robustness to noise introduced by the hardware.

5.2.2 XOR Error Model

All the encoding techniques that will be considered in this paper can be realized from XOR gates only, see (5.1) for instance. As a consequence, we assume that errors are introduced during elementary XOR operations. Denote p_{xor} the XOR gate error probability. The faulty XOR operator $\tilde{\oplus}$ is defined as

$$a \mathbin{\tilde{\oplus}} b = \begin{cases} a \oplus b & \text{with prob. } 1 - p_{\text{xor}} \\ 1 \oplus (a \oplus b) & \text{with prob. } p_{\text{xor}}. \end{cases} \tag{5.4}$$

where a and b are binary digits and $a \oplus b$ is the (reliable) XOR sum of a and b. Alternatively, the faulty XOR operator can be expressed as

$$a \mathbin{\tilde{\oplus}} b = (a \oplus b) \oplus e \tag{5.5}$$

where e is a binary random variable such that $\Pr(e = 1) = p_{\text{xor}}$.

The expression $(a_1 \mathbin{\tilde{\oplus}} \dots \mathbin{\tilde{\oplus}} a_K)$ computes the faulty XOR sum of K binary digits $(a_1, \dots, a_K)$. As the operation $(a_1 \mathbin{\tilde{\oplus}} \dots \mathbin{\tilde{\oplus}} a_K)$ is realized from $(K-1)$ elementary faulty XOR operators, the error probability

$$P_e^{(K)}(p_{\text{xor}}) = \Pr\left((a_1 \mathbin{\tilde{\oplus}} \dots \mathbin{\tilde{\oplus}} a_K) \neq (a_1 \oplus \dots \oplus a_K)\right) \tag{5.6}$$

can be expressed as

$$P_e^{(K)}(p_{\text{xor}}) = \frac{1}{2} - \frac{1}{2}(1 - 2p_{\text{xor}})^{(K-1)}. \tag{5.7}$$

Indeed, the result of $(a_1 \mathbin{\tilde{\oplus}} \dots \mathbin{\tilde{\oplus}} a_K)$ is in error if and only if an odd number of errors are introduced among the $(K-1)$ involved elementary XOR operations. Equation (5.7) then comes from the Gallager's formula [49, Section 3.8]. Note that the error probability $P_e^{(K)}(p_{\text{xor}})$ depends on the number K of elementary XOR operators and on the XOR error probability p_{xor}. However, $P_e^{(K)}(p_{\text{xor}})$ does not depend on the order the elementary XOR operations are performed.

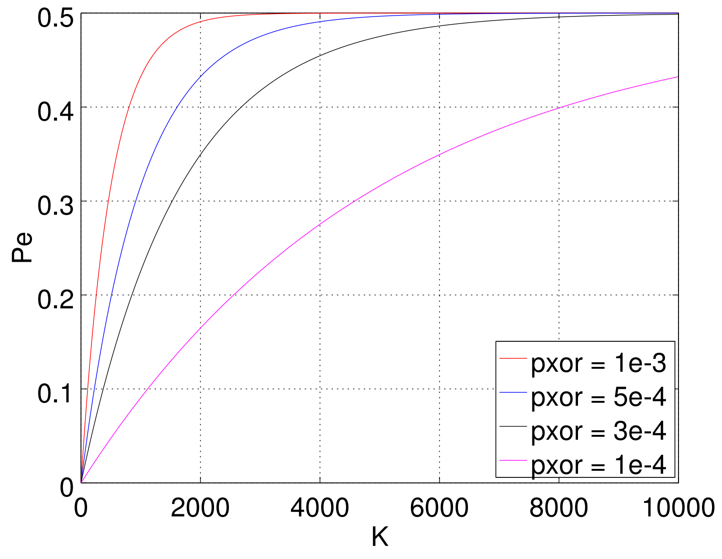


Figure 5.1: Error probability $P_e^{(K)}(p_{\text{xor}})$ with respect to K for various values of p_{xor}

Figure 5.1 represents the error probability $P_e^{(K)}(p_{\text{xor}})$ (5.7) with respect to K for various values of p_{xor}. It shows that the error probability increases with K and converges to 0.5. For high values of p_{xor}, we see that the error probability increase is very fast and that even for relatively small values of K (*e.g.*, $K = 1000$), the error probability is high. This suggests that any encoding operation involving the XOR sum of an important number of digits will lead to a high error probability. To confirm this, we now describe the existing encoding solutions and evaluate their robustness to hardware noise with respect to the XOR error model.

5.3 Existing Encoding Techniques

Here, we describe the existing encoding solutions and evaluate their robustness when the XOR operations are faulty and follow the model of Section 5.2.2. Here, only encoding with codeword in a systematic form will be considered. A codeword $\mathbf{x}$ in systematic form is denoted $\mathbf{x} = [\mathbf{u}, \mathbf{p}]^T$ and is composed by two parts. The first part is the information sequence $\mathbf{u}$ of length $n - m$, and the second part is given by the parity vector $\mathbf{p}$ of length m. In this case, encoding consists of constructing the codeword $\mathbf{x}$ by calculating the parity bits $\mathbf{p}$ from the information sequence $\mathbf{u}$.

In this section, we first analyze the robustness of the general encoding solutions for which an encoder can be obtained from any parity check matrix H. We then present particular code constructions which guarantee low encoding complexity.

5.3.1 General Encoding Solutions

In this section, we describe the solutions to construct an encoder from a given parity check matrix H. We begin with systematic encoding which is the simplest solution to obtain the encoder.

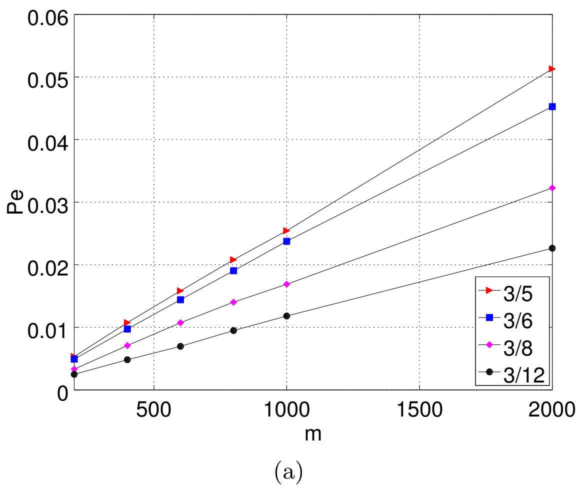
Systematic Encoding

From Gaussian elimination, the parity check matrix H can be put in systematic form

$$H = [P \quad I_m]^T \tag{5.8}$$

where I_m is the identity matrix of size $m \times m$ and P is a matrix of size $m \times (n - m)$. A systematic generator matrix G is then be obtained from (5.8) and (5.3) as

$$G = \left[I_{(n-m)} \quad P^T\right]^T . \tag{5.9}$$



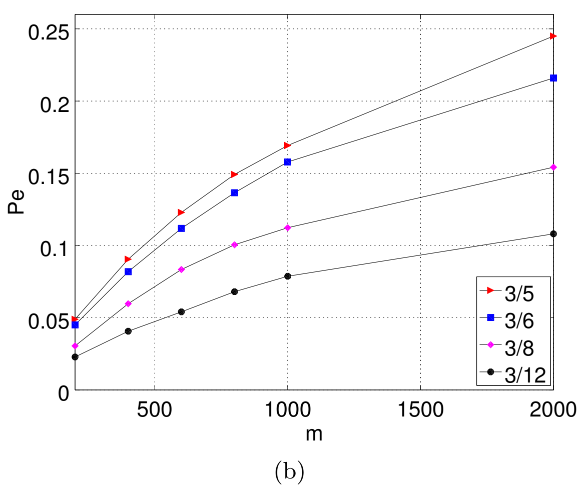


Figure 5.2: Error probability with respect to m for systematic encoding different regular codes, with (a) $p_{\text{xor}} = 10^{-4}$, (b) $p_{\text{xor}} = 10^{-3}$

The codeword $\mathbf{x} = [\mathbf{u}, \mathbf{p}]^T$ in systematic form can then be directly computed from (5.1). Unfortunately, the matrix P obtained from Gaussian elimination is not sparse which induces a high encoding complexity in $O(n^2)$.

The error probability of the systematic encoding operation (5.1) can be calculated as follows. The computation of the parity bit p_j, $j = 1, \dots, m$, requires $N_j - 1$ elementary XOR operations, where N_j is the number of non-zero components in the $j - th$ line of P. Thus from (5.7) the encoding error probability for a given generator matrix G can be expressed as

$$P_e = \frac{1}{n} \sum_{j=1}^{m} \left(\frac{1}{2} - \frac{1}{2} (1 - 2p_{\text{xor}})^{(N_j - 1)} \right). \tag{5.10}$$

In order to evaluate the robustness of systematic encoding, we have constructed a collection of parity check matrices. All the constructed parity check matrices have variable node degree $d_v = 3$ but they have various check node degrees d_c and information sequence length m. For each of the considered parity check matrices H, we have constructed the corresponding generator matrix G and calculated the encoding error probability P_e from (5.10). Figure 5.2 represents the obtained encoding error probabilities with respect to m for $p_{\text{xor}} = 1e - 3$ and $p_{\text{xor}} = 1e - 4$. The encoding error probabilities are high because the matrices P are not sparse. Furthermore, we see that when the code rates decrease, the error probabilities increase. This is expected because the error probability (5.10) increases with the N_j which themselves increase with m and n.

When $p_{\text{xor}} = 10^{-4}$, we observe that the encoding error probability is not too dramatic, even when $m = 2000$. As a consequence, a more robust solution would be to complete the encoding with a robust decoder. The objective of the decoder would be to reconstruct the original codeword before transmission on the channel. However, when $p_{\text{xor}} = 10^{-3}$, the encoding error probability becomes too important, and even the additional decoder may not be able to recover the good codeword.

As a conclusion, systematic encoding not only induces high encoding complexity but also exhibits poor robustness to hardware errors. An encoding solution called Approximate Lower Triangular encoding has been proposed in [41] in order to reduce the encoding complexity. One of the building blocks of Approximate Lower Triangular encoding is Lower Triangular encoding which is described in the next paragraph.

Lower Triangular encoding

From Gaussian elimination, the parity check matrix H can also be put in lower triangular form

$$H = [Q \;\; T]^T . \tag{5.11}$$

In (5.11), T is a lower triangular matrix of size $m \times m$ with ones in the diagonal and non-zero components in the lower part of the matrix only, and Q is a matrix of size $m \times (n-m)$. The parity part $\mathbf{p}$ of the codeword $\mathbf{x}$ can be computed from (5.2) by back-substitution

$$\begin{aligned} p_1 &= \sum_{k=1}^{n-m} H_{j,k} u_k \\ \forall j = 2, \ldots, m, \;\; p_j &= \sum_{k=1}^{n-m} H_{j,k} u_k + \sum_{k=1}^{j-1} H_{j,(n-m)+k} p_k . \end{aligned} \tag{5.12}$$

As the matrices Q and T are not sparse, the encoding complexity is still in $O(n^2)$.

The error probability of Lower Triangular encoding can be evaluated as follows. Denote by N_j the number of non-zero components in the j-th line of Q and denote $\mathcal{T}_j$ the positions of the non-zero components in the j-th line of T, excluding the term in the diagonal. The successive error probabilities $P_{e,j}$ for the parity bits p_j can be calculated recursively as

$$\begin{aligned} P_{e,1} =& \frac{1}{2} - \frac{1}{2}(1 - 2p_{\text{xor}})^{N_1 - 1} \\ \forall j = 2, \ldots, m, \;\; P_{e,j} =& \frac{1}{2} - \frac{1}{2}(1 - 2p_{\text{xor}})^{N_j - 1} \prod_{j \in \mathcal{T}_j} (1 - 2P_{e,k}). \end{aligned} \tag{5.13}$$

The encoding error probability is then given by

$$P_e = \frac{1}{n} \sum_{j=1}^{m} P_{e,j} . \tag{5.14}$$

We now evaluate the error probability (5.14) for the same parity check matrices H considered in the previous paragraph for systematic encoding. Figure 5.3 represents the encoding error probabilities with respect to m for $p_{\text{xor}} = 10^{-4}$. We see that whatever the considered code, the error probability is very high even for small values of m. This is due to the non-sparsity of G and T, and also to error propagation induced by the recursive computation of the parity bits (5.12). In addition, in all cases, the error probability reaches a saturation level. In fact, for a codeword $\mathbf{x} = [\mathbf{u}, \mathbf{p}]$ in systematic form, the encoding errors are only on the parity bits $\mathbf{p}$, and not on the information bits $\mathbf{u}$. The saturation levels correspond to an error probability of $1/2$ over the parity bits. It can indeed be verified that the level of saturation is given by $1/2 \times (1 - r)$ where r in the rate of the considered code.

There was no saturation effect in the systematic encoding described in the previous paragraph. This suggests that the very high encoding error probability induced by Lower Triangular encoding is mainly due to error propagation. As a consequence, any decoder involving recursive computation of the parity bits may be non-robust to hardware errors.

Although having high complexity, Lower Triangular encoding is an building block of a lower complexity encoding solution called Approximate Lower Triangular encoding [41], which we present in the next paragraph.

Approximate Lower Triangular encoding

In [41], it is shown that from line and column permutations, the matrix H can be put in the following form

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix}^T \tag{5.15}$$

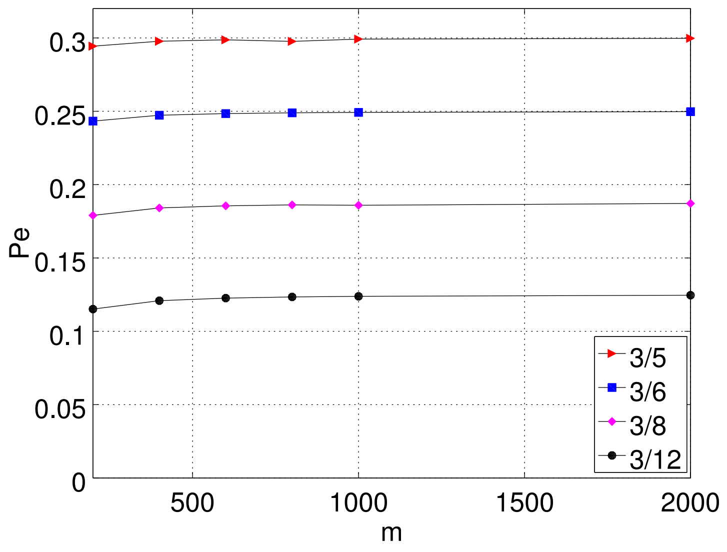


Figure 5.3: Lower triangular encoding, $p_{\text{xor}} = 1e-4$

where A (of size $(m-g)\times(n-m)$), B $((m-g)\times g)$, C $(g\times(n-m))$, D $(g\times g)$, and $E((m-g)\times g)$ are sparse matrices and T $((m-g)\times(m-g))$ is a lower triangular sparse matrix. The parameter g is called the gap of the code. The block matrices $A,\dots,E,T$, are sparse because the form (5.15) is obtained from line and column permutations only. Denote $\mathbf{x} = [\mathbf{u}\ \ \mathbf{p}_1\ \ \mathbf{p}_2]$ where $\mathbf{p}_1$ and $\mathbf{p}_2$ are binary vectors of length g and $(m-g)$, respectively.

From [41], multiplying H in the form of (5.15) by the matrix

$$\begin{bmatrix} I & 0 \\ -ET^{-1} & I \end{bmatrix} \tag{5.16}$$

makes the encoding problem equivalent to solving the system

$$A\mathbf{u} + B\mathbf{p}_1 + T\mathbf{p}_2 = 0 \tag{5.17}$$

$$(-ET^{-1}A + C)\mathbf{u} + (-ET^{-1}B + D)\mathbf{p}_1 = 0 \tag{5.18}$$

with respect to $\mathbf{p}_1$ and $\mathbf{p}_2$. The encoding can thus realized in two steps

1. Solve (5.18) by computing $\mathbf{p}_1 = -\Phi^{-1}(-ET^{-1}A + C)\mathbf{u}$ where $\Phi = -ET^{-1}B + D$. The matrix Φ is sparse, but the matrix Φ^{-1} is not sparse. As a consequence, calculating $\mathbf{p}_1$ has complexity in $O(g^2)$.

2. Solve (5.17) by computing $\mathbf{p}_2$ recursively by back-substitution as in (5.12). This time, the matrices A, B, and T involved in (5.17) are sparse. As a consequence, the complexity of calculating $\mathbf{p}_2$ is in $O(n)$.

Operations 1 and 2 induce a total encoding complexity in $O(n+g^2)$. We thus want to obtain form (5.15) with gap g as small as possible, in order to lower the encoding complexity. In [41] several algorithms were proposed for the construction of (5.15). As a result, the authors of [41] got an encoding complexity in $0.017^2n^2 + O(n)$ for $(3,6)$-codes. The encoding complexity is still in $O(n^2)$, but with a very small constant, which makes the encoding feasible even for large values of n.

Figure 5.4 gives the error probability with respect to m for codes with variable node degree $d_v = 3$ and various check node degrees. Here again, the error probabilities are high. The encoding of $\mathbf{p}_1$ is realized from non-sparse operations, which results in high error probability as for systematic encoding. The encoding of $\mathbf{p}_2$, although sparse, is constructed from $\mathbf{p}_1$ that has important error probability. Furthermore, because of back-substitution, the encoding of $\mathbf{p}_2$ results in a high error propagation,

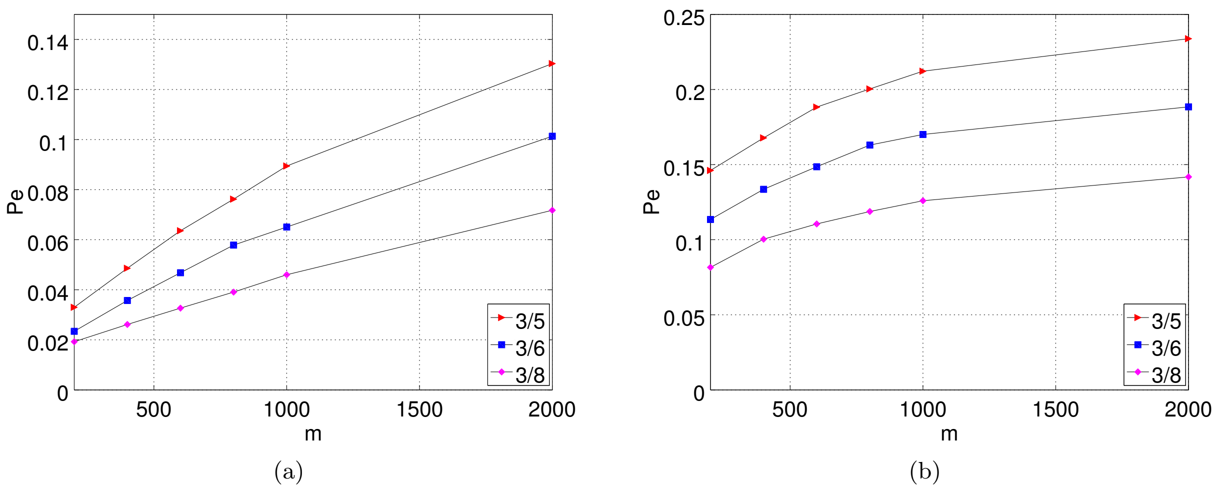


Figure 5.4: Error probability with respect to m for for Approximate Lower Triangular encoding for different regular codes, with (a) $p_{\text{xor}} = 10^{-4}$, (b) $p_{\text{xor}} = 10^{-3}$

as in Lower Triangular encoding. As a consequence, Approximate Lower Triangular encoding is not robust to hardware errors.

To conclude, the general encoding solutions not only have important encoding complexity, but they are also non-robust to hardware errors. To overcome the complexity issue, several particular code constructions have been proposed [42–45]. The next section described these particular code constructions and evaluate their robustness to faulty hardware.

5.3.2 Particular code constructions

In this section, we consider four particular code constructions that are Zig-Zag codes [42], IRA codes [45],QC codes [44], and LDGM codes [43]. We evaluate the encoder robustness of Zig-Zag and LDGM codes, and discuss the cases of IRA and QC codes.

Zig-Zag codes

The encoding structure of Zig-Zag codes [42] is described in Figure 5.5. For Zig-Zag codes, we denote the information bits as $u_{i,j}$ with $i \in \{1, \dots, I\}$, $j \in \{1, \dots, J\}$, $m = I$ and $n = I \times J$. The parity bits are denoted p_i, $i \in \{1, \dots, I\}$, and they are calculated from an accumulator as

$$p_1 = \sum_{j=1}^{J} u_{1,j}$$
$$p_i = p_{i-1} + \sum_{j=1}^{J} u_{i,j}. \tag{5.19}$$

A Zig-Zag code has rate $r = J/(J-1)$ and encoding complexity in $O(n)$.

The successive error probabilities $P_{e,i}$ of the parity bits p_i can be calculated recursively as

$$P_{e,1} = \frac{1}{2} - \frac{1}{2}(1 - 2p_{\text{xor}})^{(J-1)}$$
$$P_{e,i} = \frac{1}{2} - \frac{1}{2}(1 - 2p_{\text{xor}})^{(J-1)}(1 - 2P_{e,i}) \tag{5.20}$$

and the encoding error probability is given by

$$P_e = \frac{1}{I \times (J+1)} \sum_{i=1}^{I} P_{e,i}. \tag{5.21}$$

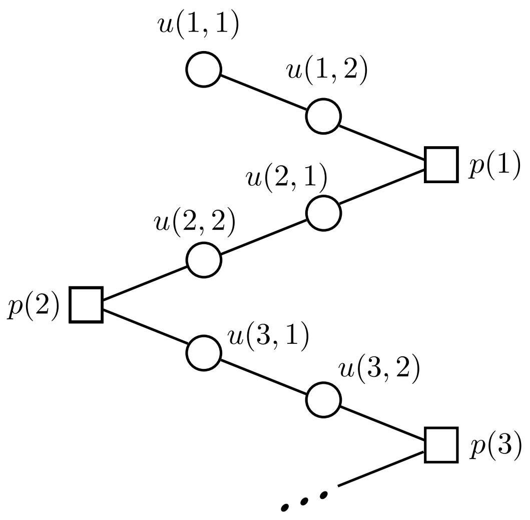


Figure 5.5: Example of Zig-Zag code

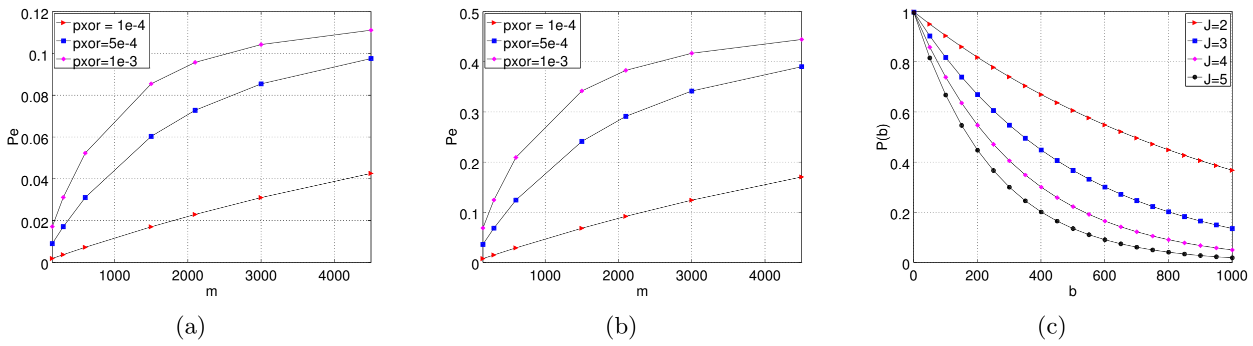


Figure 5.6: Zig-Zag codes, (a) Overall error probability with respect to $m = I \times J$ for $J = 3$, (b) Error probability over the parity bits with respect to $m = I \times J$ for $J = 3$, (c) Probability of a burst of length b for $p_{\text{xor}} = 10^{-3}$

Figure 5.6 (a) represents the encoder error probabilities with respect to m for $J = 3$ and for several values of p_{xor}. The error probability is high, although not too dramatic. However, it concerns codes with high rates $r = J/(J-1)$. The error probability is computed in average over all the $I \times (J+1)$ bits of the codeword, while the $I \times J$ information bits are error-free. As the code rate is high, the codeword contains only a few number of parity bits, which lowers the overall error probability. However, the error probability calculated over the parity bits only is very high, as represented in Figure 5.6 (b). This is mainly due to high error propagation induced by the encoding technique. For instance, if bit p_i is in error, the probability that bit p_{i+1} is in error will be very high. To emphasize that, Figure 5.6 (c) represents the probability $P(b)$ of a burst of errors of length b, that is the probability that if one parity bit is in error, then the b successive parity bits are also in error. We see that $P(b)$ decreases very slowly with b, which confirms the high error propagation.

To conclude, although the Zig-Zag encoder has low complexity in $O(n)$, it is not robust to hardware errors because of the accumulator. Zig-Zag codes are a building block for the construction of IRA codes, as described in the following paragraph.

IRA codes

An IRA code [45] is the concatenation of an outer irregular repetition code and of an inner Zig-Zag code. With the concatenated structures, on can construct codes of any rates. Due to the error model we consider, The hardware noise does not affect the repetition part, but only the Zig-Zag encoding.

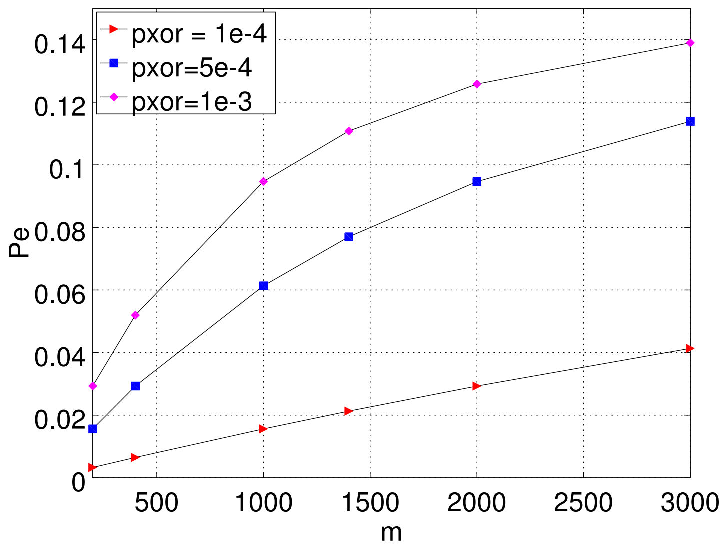


Figure 5.7: Error probability for Zig-Zag encoding

However, as shown previously, the error probability of Zig-Zag encoding is high, and in particular, the error probability in the parity bits is very high because of error propagation. As a consequence, the IRA encoder may not be robust to hardware errors.

QC codes

The parity check matrix H of a QC code is composed by $c \times t$ circulant matrices $A_{i,j}$ ($i = 1, \dots, c$, $j = 1, \dots, t$) of size $b \times b$. It can be expressed as

$$H = \begin{bmatrix} A_{1,1} & \dots & A_{1,t} \\ \dots & \dots & \dots \\ A_{c,1} & \dots & A_{c,t} \end{bmatrix}. \tag{5.22}$$

Three encoding solutions were proposed in [44] for QC codes. They were evaluated in terms of circuit complexity, that is the number of components (XOR and NAND gates, cyclic-shift registers, etc.) required by the encoder. The circuit complexity is different from the computational complexity that is the number of operations that are actually performed to realize the encoding. The first two encoding solutions are systematic encoding technique adapted to QC-codes. The third one is a two stages encoding technique.

The authors in [44] show that, with the use of cyclic-shift registers, the circuit complexity of the encoding solutions they propose is either linear with the number of information bits (serial systematic encoding), with the number of parity bits (parallel systematic encoding), or with the codeword length (two stages encoding). However, the low circuit complexity is due to the fact that each component is used several times, which induces a high computational complexity. Indeed, as the circulant matrices used in the encoding operation are not sparse, this computational complexity is in $O(n^2)$. As a consequence, the encoder error probabilities may be in the same order to magnitude of systematic encoding presented in Section 5.3.1. Thus, encoding for QC-codes may not be robust to hardware errors.

LDGM codes

Consider the generator matrix $G = \left[I_{(n-m)} \;\; P^T\right]^T$ and the parity check matrix $H = [P \;\; I_m]^T$ in systematic forms. With LDGM codes [43], instead of constructing the systematic forms from an already defined parity check matrix, we directly construct a sparse matrix P. As a result, with LDGM codes, both matrices H and G are sparse. The encoding can then be realized from (5.1), which gives a

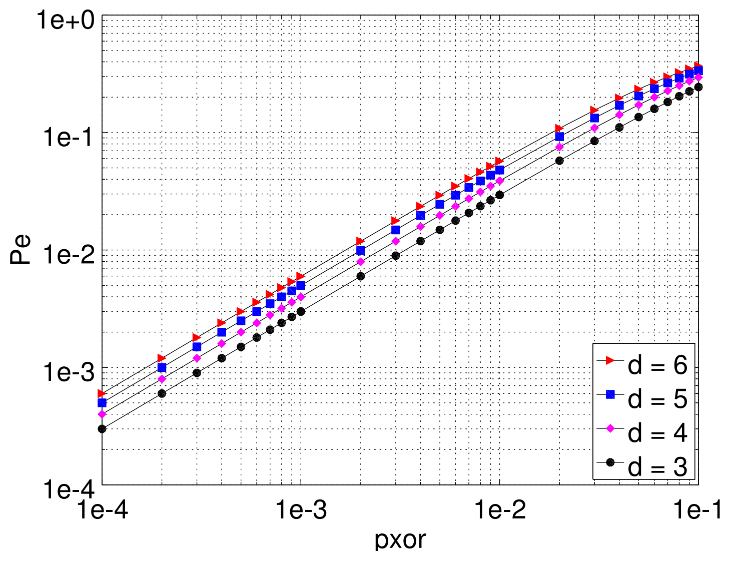


Figure 5.8: Error probability with respect to p_{xor} for LDGM codes

complexity in $O(n)$. The encoding error probability is given by (5.10). Figure 5.8 represents the error probabilities with respect to p_{xor} for various codes with column degrees d for the matrix P. As P is sparse, the encoding error probabilities are very small. In addition, the encoding error probabilities do not vary with m as the column degrees d are fixed.

LDGM codes are thus naturally robust to noise introduced by the hardware, as illustrated in Figure 5.8. However, the decoder performance of LDGM codes is not as good as the performance of LDPC codes, as illustrated from Monte-Carlo simulations in the following section.

5.4 Evaluation of Robust Encoding Solution

In this section, we discard all the encoding solutions with error propagation (Lower Triangular, Approximate Lower Triangular, Zig-Zag codes) and evaluate the performance of the three following encoding solutions.

1. *Systematic encoding, encoder alone* (see Figure 5.9). It corresponds to the standard transmission scheme in which both the encoder and the decoder are faulty.

$$\begin{tikzcd} {} \arrow[r, "\mathbf{u}"] & \boxed{\mathrm{E}} \arrow[r, "\mathbf{x}"] & \boxed{\text{channel}} \arrow[r, "\mathbf{y}"] & \boxed{\mathrm{D}} \arrow[r, "\hat{\mathbf{u}}"] & {} \end{tikzcd}$$

Figure 5.9: Encoder alone

2. *Systematic encoding, encoder + decoder*. In this case, we add a decoder D_1 at the encoder part. The objective of D_1 is to reconstruct the good codeword before transmission on the channel.

$$\begin{tikzcd} {} \arrow[r, "\mathbf{u}"] & \boxed{\mathrm{E}} \arrow[r] & \boxed{\mathrm{D}_1} \arrow[r, "\mathbf{x}"] & \boxed{\text{channel}} \arrow[r, "\mathbf{y}"] & \boxed{\mathrm{D}_2} \arrow[r, "\hat{\mathbf{u}}"] & {} \end{tikzcd}$$

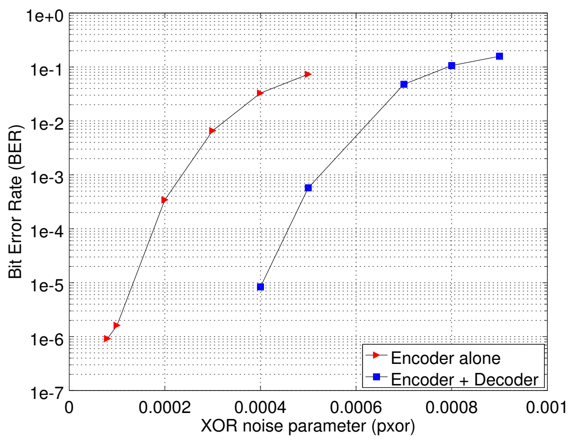
Figure 5.10: Encoder and decoder

3. *LDGM codes, encoder alone*. It also corresponds to the transmission scheme of Figure 5.9, but the encoding and decoding are realized with an LDGM code instead of an LDPC code.

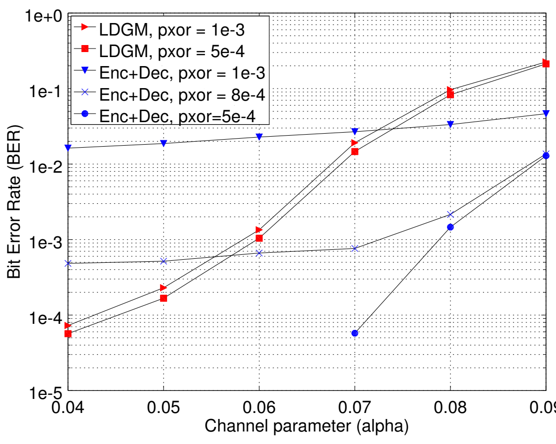
In the following, all the decoders are faulty offset min-sum decoders with FD error models as in Chapter 1 and $p_c = p_v = p_a = 10^{-3}$. We first compare Solutions 1 and 2. We choose a code of rate 1/2, with 100 iterations and $m = 500$. The channel parameter is set to $\alpha = 0.03$. Figure 5.11 (a) represents the Bit Error Rate (BER) with respect to p_{xor} for Solutions 1 and 2. As expected, we observe an important loss in performance when D_1 is absent.

We now discard Solution 1 and compare Solutions 2 and 3. We consider codes of rate 1/4 with $m = 400$. For LDPC codes, we consider a $(3, 4)$ regular codes. For LDGM codes, the matrix P is constructed as a $(4, 6)$ regular codes. The BER of both solutions are represented in Figure 5.11 (b). For LDGM codes, we see that the BER does not vary much with p_{xor}. We also see that despite their robustness to hardware errors, LDGM codes give poor BER performance, and in particular high error floor. On the opposite, we see that for LDPC codes, a small variation of p_{xor} can induce an important loss in BER performance. For $p_{\text{xor}} = 10^{-3}$ and $p_{\text{xor}} = 8 \times 10^{-4}$, D_1 is not able to recover completely the original codeword. As a consequence, the encoder noise combines with the channel noise, which lowers the BER performance. For $p_{\text{xor}} = 5 \times 10^{-4}$, D_1 can correct almost all the noise that was introduced by the faulty encoder and we get better BER performance.

As a conclusion, on one hand, LDGM codes are robust to hardware errors at the price of a high error floor. On the other hand, Solution 2 is less robust to hardware errors but shows better performance when the hardware noise is small enough.



(a)



(b)

Figure 5.11: (a) BER with respect to p_{xor} for solutions 1 and 2, (b) BER with respect to α for solutions 2 and 3.

5.5 Conclusion

In this chapter, we have evaluated the robustness of several encoding solutions to noise introduced by the hardware. We have seen that most of them are not robust to noise introduced by the hardware, especially when the codeword length increases. When the hardware is faulty, we have identified two possible encoding solutions. In the first one, the encoder is composed by noisy systematic encoding followed by a faulty decoder that has to recover the original codeword before transmission on the channel. However, this solution induces high encoding complexity and is not robust when the hardware noise is too high. The second solutions consists of the use of LDGM codes. However, it shows poor decoder performance and in particular low error floor.

General Conclusion and Next Steps

In the period M13-M21 covered by Deliverable 3.2, we have made considerable progress beyond the state of the art on the analysis and design of faulty iterative decoders, with a special focus on FAID decoders, stochastic decoders, and bit-flipping decoders. In parallel, we have also analyzed LDPC encoders on faulty hardware. The main contributions during this period for this WP can be summarized as follows:

– We have introduced the functional threshold definition to characterize the asympotic behavior of noisy decoders. We have shown that under restricted noise conditions, the functional threshold predicts the asymptotic behavior of noisy decoders. Based on the functional threshold analysis, we have proposed a method for the design of robust FAIDs. Monte-Carlo simulations have validated the proposed design methodology.

– We proposed error models for different processing units of the stochastic decoder, and investigated two noisy versions: stochastic decoder with noisy edge memories and full-noisy stochastic decoders. We have shown by Monte-Carlo simulations that the stochastic decoder is naturally robust to hardware errors. In addition, the performance of the noisy Stochastic decoder has been also assessed against that of the noisy MS decoder, and we showed that the Stochastic decoder presents an increased robustness to hardware errors.

– 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 decoder's 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 have reviewed several encoding solutions and analyzed their robustness to hardware errors. We have observed that most of them are not robust to hardware errors. We have further proposed and analyzed the performance of two possible encoding solutions on faulty hardware.

New challenges for the rest of the project in WP3 include:

– In the period M13-M21 we initiated the investigation of a class of low-complexity and possibly more robust bit-flipping decoders. In this context we proposed a new PGDBF decoder, which has been shown to have the ability of utilizing hardware failures in order to improve the decoding performance. The investigation of the PGDBF decoder we will continue during the last year of the project. The goal is twofold: (1) to incorporate more realistic error models, and (2) to further develop the theoretical analysis, such as to allow understanding in which conditions the hardware noise can improve the decoding performance.

– Until now, in WP3 we only considered memoryless, data independent, symmetric error models, which have permitted to develop theoretical tools for the analysis of faulty decoders. They have also given us some insights on the design of robust decoders. However, such models are far from realistic. Therefore, during the last year of the project, one of the main goals will be to incorporate more realistic noise models, at least for some of the decoders investigated in WP3, *e.g.*, the PGDBF decoder mentioned above, or the stochastic decoder.

– While the solutions we proposed for faulty encoding allow increasing the robustness to hardware errors, they are not completely satisfactory, because of either high encoding complexity or low decoding performance. However, this study allowed us to identify new avenues for future research that will be explored in the last year of the project.

If necessary, the WP3 work plan could be slightly modified for the last year, in order to address the main challenges that have been identified during the first two years of the project (*e.g.*, the duration of some tasks may be extended, so as to continue working on some specific decoding or encoding solutions).

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] J. V. Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[3] P. Gács and A. Gál, "Lower bounds for the complexity of reliable Boolean circuits with noisy gates," *IEEE Transactions on Information Theory*, vol. 40, no. 2, pp. 579–583, March 1994.

[4] R. Dobrushin and S. Ortyukov, "Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements," *Problemy Peredachi Informatsii*, vol. 13, no. 3, pp. 56–76, 1977.

[5] N. Pippenger, "On networks of noisy gates," in *26th Annual Symposium on Foundations of Computer Science*, Oct. 1985, pp. 30–38.

[6] M. G. Taylor, "Reliable information storage in memories designed from unreliable components," *Bell System Technical Journal*, vol. 47, pp. 2299–2337, Dec. 1968.

[7] A. V. Kuznetsov, "Information Storage in a Memory Assembled from Unreliable Components," *Problems of Information Transmission*, vol. 9, no. 3, pp. 254–264, 1973.

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

[9] S. K. Chilappagari, M. Ivkovic, and B. Vasic, "Analysis of one step majority logic decoders constructed from faulty gates," in *Proc. of IEEE Int. Symp. on Information Theory*, 2006, pp. 469–473.

[10] L. Varshney, "Performance of LDPC Codes Under Faulty Iterative Decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, July 2011.

[11] C.-H. Huang and L. Dolecek, "Analysis of finite-alphabet iterative decoders under processing errors," in *Proc. 2013 IEEE Int. Conf. Acoustics, Speech, Sig. Proc.*, May 2013, pp. 5085——5089.

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

[13] C.-H. Huang, Y. Li, and L. Dolecek, "Gallager B LDPC decoder with transient and permanent errors," in *IEEE International Symposium on Information Theory Proceedings*, July 2013, pp. 3010–3014.

[14] S. Yazdi, H. Cho, and L. Dolecek, "Gallager B Decoder on Noisy Hardware," *IEEE Transactions on Communications*, vol. 66, no. 5, pp. 1660–1673, May 2013.

[15] A. Balatsoukas-Stimming and A. Burg, "Density Evolution for Min-Sum Decoding of LDPC Codes Under Unreliable Message Storage," *IEEE Communications Letters*, vol. 18, no. 5, pp. 849–852, May 2014.

[16] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Min-Sum-based decoders running on noisy hardware," in *proc. of IEEE Global Communications Conference (GLOBECOM)*, 2013.

[17] T. J. 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.

[18] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Unconventional Behavior of the Noisy Min-Sum Decoder over the Binary Symmetric Channel," in *Information Theory and Applications Workshop (ITA)*, Feb. 2014, pp. 1–10.

[19] A. Bennatan and D. Burshtein, "Design and analysis of nonbinary LDPC codes for arbitrary discrete-memoryless channels," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 549–583, 2006.

[20] C. Wang, S. Kulkarni, and H. Poor, "Density evolution for asymmetric memoryless channels," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4216–4236, Dec 2005.

[21] C. K. Ngassa, V. Savin, E.Dupraz, and D. Declercq, "Density Evolution and Functional Threshold for the Noisy Min-Sum Decoder," *Submitted to IEEE Transactions on Communications*, May 2014.

[22] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[23] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello Jr, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. on Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, 2004.

[24] 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 *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, April 2013, pp. 4293–4298.

[25] S. Sharifi Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

[26] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. on Signal Processing*, vol. 56, no. 11, pp. 5692–5703, 2008.

[27] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Likoping University, Sweden, 1996.

[28] V. Gaudet and W. Gross, "Switching Activity in Stochastic Decoders," in *Multiple-Valued Logic (ISMVL), 2010 40th IEEE International Symposium on*, May 2010, pp. 167–172.

[29] K.-L. Huang, V. Gaudet, and M. Salehi, "A Markov chain model for Edge Memories in stochastic decoding of LDPC codes," in *Information Sciences and Systems (CISS), 45th Annual Conference on*, 2011.

[30] D. J. MacKay. Encyclopedia of Sparse Graph Codes. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[31] 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.

[32] 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.

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

[34] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes," 2014. [Online]. Available: http://arxiv.org/abs/1402.2773.

[35] D. Nguyen, S. Chilappagari, M. Marcellin, and B. Vasic, "On the Construction of Structured LDPC Codes Free of Small Trapping Sets," *IEEE Trans. Inform. Theory*, vol. 58, no. 4, pp. 2280–2302, April 2012.

[36] T. Richardson and R. Urbanke, *Modern coding theory.* Cambridge Univ Press, 2008.

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

[38] 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.

[39] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Block-interlaced ldpc decoders with reduced interconnect complexity," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 1, pp. 74–78, 2008.

[40] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured ldpc codes," *Proc. of ICSTA*, 2001.

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

[42] 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.

[43] J. Garcia-Frias and W. Zhong, "Approaching shannon performance by iterative decoding of linear codes with low-density generator matrix," *IEEE Communications Letters*, vol. 7, no. 6, pp. 266–268, 2003.

[44] 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.

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

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

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

[48] A. Roumy, S. Guemghar, G. Caire, and S. Verdú, "Design methods for irregular repeat-accumulate codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1711–1727, 2004.

[49] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962, research Monograph series.