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

D3.1

Fault tolerant LDPC encoding and decoding

Editor:	D. Declercq, ENSEA
Deliverable nature:	Public
Due date:	January 31, 2014
Delivery date:	February 3, 2014
Version:	1.0
Total number of pages:	81 pages
Reviewed by:	*i*-RISC partners
Keywords:	LDPC codes, faulty Density Evolution, Min-Sum and self-corrected Min-Sum, FAID decoders, robustness to transient errors.

Abstract

This deliverable presents an overview of the activities carried out by the work package WP3 during the first year of the project. These activities include mainly the study of iterative decoding under faulty hardware, in order to understand the limits of Fault tolerance techniques based on LDPC forward error correction.

In order to protect the different parts of the chips from transient defects, the storage and computation units have to be redundant and incorporate a powerful error-correction technique. In this workpackage, we focus on the class of LDPC codes, decoded with iterative message passing decoders, and our goal is the design of LDPC codes with fault-tolerant encoder and decoder architectures. Our contribution during this first period of the project has been to develop theoretical analysis and to propose practical decoding algorithms, which are tolerant to transient errors, coming from the faulty hardware. Error correcting codes with fault tolerant encoder and decoder architectures constitute a building block of our approach to fault tolerant chip design.

List of Authors

Participant	Author
CEA	Valentin Savin (valetin.savin@cea.fr)
	Christiane L. Kameni Ngassa (christiane.kameningassa@cea.fr)
ENSEA	David Declercq (declercq@ensea.fr)
	Erbao Li (erbao.li@ensea.fr)
ELFAK	Bane Vasic (vasic@email.arizona.edu)

Contents

List of Dissemination Activities

Published papers

[P1] C. L. Kameni Ngassa, V. Savin, D. and Declercq, "Design of Min-Sum-based LDPC decoders using imprecise arithmetic", *IEEE Int. Conference on Computer as a tool (EUROCON)*, Zagreb, Croatia, July 2013.

[P2] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Analysis of Min-Sum based decoders implemented on noisy hardware", *Asilomar Conference on Signals, Systems and Computers*, Asilomar, CA, USA, November 2013.

[P3] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Min-Sum-based decoders running on noisy hardware", *IEEE Global Communications Conference (GLOBECOM)*, Atlanta, GA, USA, December 2013.

[P4] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Unconventional behavior of the noisy Min-Sum decoder over the Binary Symmetric Channel", *IEEE Information Theory and Applications Workshop (ITA)*, San Diego, CA, USA, February 2014.

Submitted papers

[P5] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Asymptotical and finite length analysis of the Min-Sum Decoder on noisy hardware", *IEEE Transactions on Communications*, submitted.

[P6] S. Planjery, D. Declercq, and B. Vasić, "Guaranteed error correction with finite number of iterations and FAID decoders", *IEEE Transactions on Information Theory*, submitted.

Workshop presentations

[P7] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Min-Sum-based decoders running on noisy hardware", Groupement de Recherche Information, Signal, Image et Vision (GdR-ISIS), Telecom ParisTech, Paris, July 2, 2013.

[P8] C. L. Kameni Ngassa, V. Savin, and D. Declercq, "Analysis and design of Min-Sum-based decoders running on noisy hardware", i-RISC Workshop, European Solid-State Circuits Conference (ESSCIRC), Bucharest, September 20, 2013.

[P9] Bane Vasić and David Declercq, "Bit-flipping decoders for fault-tolerant memories", i-RISC Workshop, European Solid-State Circuits Conference (ESSCIRC), Bucharest, September 20, 2013.

[P10] David Declercq, S. Planjery and B.Vasić, "Low complexity Finite Alphabet Iterative Decoders tolerant to faulty hardware", i-RISC Workshop, European Solid-State Circuits Conference (ESSCIRC), Bucharest, September 20, 2013.

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
BP	Belief-Propagation
CN	Check Node
FER	Frame Error Rate
LDPC	Low Density Parity Check
MS	Min-Sum
MP	Message-Passing
SCMS	Self-Corrected Min-Sum
VN	Variable Node

Introduction

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

Within i-RISC, the Work-Package 3 (Fault Tolerant Algorithms for Error-Correction) is aimed at understanding error-correcting codes in the context of unreliable computing systems. This is a new paradigm in coding theory, since faulty hardware can potentially induce errors during the decoding process. It is then critical to properly evaluate the robustness of the existing decoders in the presence of an additional source of noise at the circuit level. The main goal is to propose 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-based decoders, Finite-Alphabet Iterative Decoders (FAIDs), and stochastic decoder. For all considered decoders, both the asymptotic theoretical analysis and the practical robustness of the algorithms will be studied and benchmarked during the project. As an important outcome of WP3, we aim at providing a ranking of the considered decoding algorithms in terms of their ability to be tolerant to the faulty hardware setting. 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).

The Gantt chart of WP3 is represented in Figure 1. An overview of the activities carried out during the first year of the project is presented in the next section.

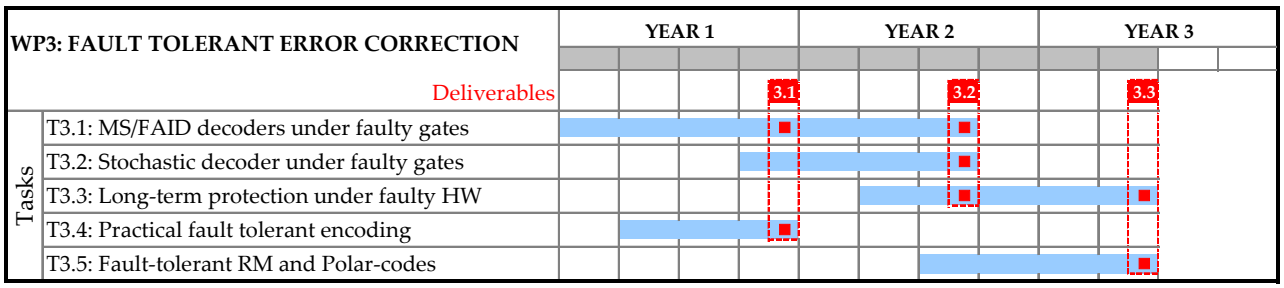


Figure 1: Gantt chart of Work-Package 3

Executive Summary

This deliverable presents an overview of the activities carried out by the work-package WP3 during the first year of the project. These activities include mainly the study of iterative decoding under faulty hardware, in order to understand the limits of fault tolerance techniques based on LDPC forward error correction.

In order to protect the different parts of the chips from transient defects, the storage and computation units have to be redundant and incorporate a powerful error-correction technique. In this work-package, we focus on the class of LDPC codes, decoded with iterative message passing decoders, and our goal is the design of LDPC codes with fault-tolerant encoder and decoder architectures. Our contribution during this first period of the project has been to develop theoretical analysis and to propose practical decoding algorithms, which are tolerant to transient errors, coming from the faulty hardware. Error correcting codes with fault tolerant decoder architectures constitute a building block of our approach to fault tolerant chip design.

We have conducted study and analysis for 3 types of LDPC decoders, and two different models of transient errors. The three types of decoders are:

- The *finite precision Min-Sum* decoder (**MS**),
- A modified version of the Min-Sum decoder incorporating a dynamical correction of mis-convergence, called *self-corrected Min-Sum* (**SC-MS**),
- More general decoders based on non-linear Boolean function for the message passing updates, called *Finite Alphabet Iterative Decoders* (**FAID**).

In the i-RISC project, we have first focused on the theoretical analysis and understanding of the noisy versions of these decoders, by conducting a density evolution (DE) study of these decoders. This will help us to understand, by other means than just Monte Carlo simulations, the limits of iterative decoding with faulty hardware. A density evolution approach has however some limitations as it can be defined only for symmetric decoding functions without memory. Therefore, only the MS and the FAIDs were analyzed using DE. Additionally, the transient error model has also to be symmetric in order to guarantee the independence of the DE dynamical system with respect to the transmitted codeword. We have therefore restricted the theoretical study of the MS and the FAIDs to symmetric error models, which can be far from the real noise present in the low-powered circuits. The analysis of MS and FAID under symmetric faulty hardware is presented in chapters 1 and 2, respectively.

The main conclusions of this study is that iterative message passing decoders under noisy hardware can be very robust to transient errors, as they continue to be able to operate when the different computing units are in defect. This result was already known for theoretical infinite precision decoders [1], and also for hard-decision decoders (bit-flipping and Gal-B), [2, 3] and our analysis confirms this property for practical finite-precision iterative decoders. Our study has also focused on means to improve the decoder reliability, by protecting critical bits of the finite-precision computation flow within the decoding process. To this end, we defined and investigated sign-preserving error models for primary arithmetic operations (e.g. adders), or for larger processing units of the decoder (e.g. variable-node or check-node processing unit). For such models, the sign of the operation (or processing unit) result is always computed accurately. The benefits of sign-preserving computational models have

been demonstrated asymptotically through DE analysis, and verified at finite-lengths by Monte Carlo simulations.

Along the lines of this study, we have also identified some peculiar and very interesting behaviors of the MS and FAID decoders under noisy hardware: in some cases, the noisy version of a decoder can have better asymptotic performance (namely can correct a larger fraction of errors) than its noiseless version. The reasons of this behavior will be further investigated in the second period of the i-RISC project. Another conclusion of this study is that using the FAID decoders framework, one can identify message passing update rules which are naturally more robust to transient errors, and more importantly, that the best update rules for the noisy case are not the same as the best update rules for the noiseless case. This opens one direction of research as to propose message passing decoders which are robust to transient errors and should be used in the context of computing in the presence of errors. We will continue this direction in the second period of the i-RISC project.

Additionally, the practical performance of the considered decoders has also to be verified for the non-asymptotical case, and for more practical error configurations. We have then also conducted a finite length statistical analysis of the different decoders to validate the theoretical analysis. In particular, we have verified in chapters 1 and 2 that the observations made with the DE analysis were still true for the finite length case. We have then confirmed that the noisy message passing decoders have error correction performance very close to their noiseless performance, and sometimes can even surpass the performance of the noiseless case. We have also verified that for sign preserving computational models, the level of noise that can be tolerated with only negligible performance degradation is far superior than in the full-depth noise case. Finally, we evaluated the finite-length performance of the noisy SC-MS decoder, and showed that it provides nearly the same performance as the noiseless decoder, for a wide range of values of the hardware noise parameters.

We have also conducted a study on a more realistic error scenario, consisting of the *imprecise arithmetic* computation framework. In this case, the source of errors comes from the fact that arithmetic units in the decoders are implemented with a smaller number of logic gates than what is actually needed, which may result in significant savings in energy. Under this error model, we have shown in particular that the SC-MS could reach the same error correction performance as the full-precision decoders. This work is reported in chapter 3. The SC-MS decoder seems to be naturally robust to hardware errors and we will continue its development and analysis for the more general types of transient errors in the second phase of the i-RISC project.

The work achieved in WP3 is in line with the work-plan and especially answers fully the objectives of *Task 3.1 "Design and analysis of Min-Sum and FAID decoders under faulty gates models"*, and prepares the continuation of the work toward *Task 3.3 "Quasi-error free protection or long-term protection under faulty hardware setting"* and *Task 3.4 "Practical encoding of sparse-graph codes under faulty hardware setting"*. We have postponed the study of fault-tolerant encoding of the codewords to the next stages of the project, as in the noisy computation framework, the only way of having a fault-tolerant encoder is to rely on the decoder itself. In parallel, the development of *Task 3.2 "Design and Analysis of Stochastic Decoding under faulty gates models"* has started and will be reported in future deliverables.

Let us now describe how this work is related to the development of the other work-packages:

- As said in this introduction and more precisely presented in the deliverable, the error models that we can use for the asymptotical analysis are limited and could be very far from the more realistic noises. We will take the outputs of the error models characterization from WP2 and explore ways to generalize the formal mathematical error models to refine our DE analysis. The accurate error models will also be used with Monte Carlo estimations and simulations to verify the validity of our approaches.

- Contrary to the hard-decision decoders, like the Gal-B decoder, which loses error correction performance under faulty hardware, we have shown in this deliverable that when using more

elaborate message passing decoders, one can still recover the same fraction of errors than in the noiseless case. Related to the work planned in WP4, important gains for Taylor-Kuznetzov reliable memories are then foreseen using either SC-MS or optimized FAIDs, and we will explore this direction in the second period of the project.

- Finally, the development of VHDL models for SC-MS and FAIDs has started, and we will provide them to the i-RISC partners when finalized. These hardware models will be incorporated in the demonstrators in WP6.

Chapter 1

Min-Sum-based decoders running on noisy hardware

Abstract: *This chapter deals with Low-Density Parity-Check decoders running on noisy hardware. The goal is to properly evaluate the robustness of existing decoders in the presence of an additional source of noise at the circuit level. To this end, we first introduce a new error model approach and carry out the "noisy" density evolution analysis of the fixed-point Min-Sum decoder. Then, for different parameters of the noisy components of the decoder, we determine the range of the signal-to-noise ratio values for which the decoder is able to achieve a target bit error rate performance. Finally, we evaluate the finite-length performance of the Min-Sum and the Self-Corrected Min-Sum decoders running on noisy hardware.*

1.1 Introduction

In traditional models of communication or storage systems with error correction coding, it is assumed that the operations of an error correction encoder and decoder are deterministic and that the randomness exists only in the transmission or storage channel. However, with the advent of nanoelectronics, the reliability of the forthcoming circuits and computation devices is becoming questionable. Indeed, due to huge increases in density integration, lower supply voltages, and variations in the technological process, MOS and emerging nanoelectronic devices will be inherently unreliable. Besides, a significant challenge to current CMOS design is to lower the energy consumption by several factors of magnitude, with the obvious goal of energy preservation. Diminishing the energy consumption can be addressed by *aggressive supply voltage scaling*, with the drawback that bringing the signal level closer to the noise level reduces noise immunity and leads to unreliable computing. It is then becoming crucial to design and analyze error correcting decoders able to provide reliable error correction even if they are made of unreliable components.

Except the pioneered works by Taylor and Kuznetsov on reliable memories [2,4,5], later generalized in [3,6] to the case of hard-decision decoders, this new paradigm of noisy decoders has merely not been addressed until recently in the coding literature. However, over the last years, the study of error correcting decoders, especially Low-Density Parity-Check (LDPC) decoders, running on noisy hardware attracted more and more interest in the coding community. In [7] and [8] hardware redundancy is used to develop fault-compensation techniques, able to protect the decoder against the errors induced by the noisy components of the circuit. In [9], a class of modified Turbo and LDPC decoders has been proposed, able to deal with the noise induced by the failures of a low-power buffering memory that stores the input soft bits of the decoder. Very recently, the characterization of the effect of noisy processing on message-passing iterative LDPC decoders has been proposed. In [1], the concentration and convergence properties were proved for the asymptotic performance of noisy message-passing decoders, and density evolution equations were derived for the noisy Gallager-A and Belief-Propagation decoders. In [10–12], the authors investigated the asymptotic behavior of the noisy Gallager-B decoder defined over binary and non-binary alphabets. However, all these papers deal with very simple error

models, which emulate the noisy implementation of the decoder, by passing each of the exchanged messages through a noisy channel.

In this work we focus on the Min-Sum decoder, which is widely implemented in real communication systems. In order to emulate the noisy implementation of the decoder, probabilistic error models are proposed for its arithmetic components (adders and comparators). The proposed probabilistic components are used to build the noisy fixed-point decoders. We further analyze the asymptotic performance of the noisy Min-Sum decoder, and provide useful regions and target-BER-thresholds [1] for a wide range of parameters of the proposed error models. Finally, we investigate the finite-length performance of the noisy Min-Sum and Self-Corrected Min-Sum decoders.

The remainder of the chapter is organized as follows. Section 1.2 gives a brief introduction to LDPC codes and iterative decoding. Section 1.3 presents the error models for the arithmetic components. The density evolution and asymptotic analysis for the noisy quantized Min-Sum algorithm are presented in Section 1.5 and Section 1.6 respectively. Section 1.7 provides the finite-length performance and Section 1.8 concludes the chapter.

1.2 LDPC Codes and the Min-Sum Algorithm

1.2.1 LDPC Codes

LDPC codes [13] are linear block codes defined by sparse parity-check matrices. They can be advantageously represented by bipartite (Tanner) graphs [14] and decoded by message-passing (MP) iterative algorithms. The Tanner graph of an LDPC code is a bipartite graph $\mathcal{H}$, whose adjacency matrix is the parity-check matrix H of the code. Accordingly, $\mathcal{H}$ contains two types of nodes:

- *variable-nodes*, corresponding to the columns of H, or equivalently to the codeword bits, and
- *check-nodes*, corresponding to the rows of H, or equivalently to the parity equations the codeword bits are checked by.

We consider an LDPC code defined by a Tanner graph $\mathcal{H}$, with N variable-nodes and M check-nodes. Variable-nodes are denoted by $n \in \{1, 2, ..., N\}$, and check-nodes by $m \in \{1, 2, ..., M\}$. We denote by $\mathcal{H}(n)$ and $\mathcal{H}(m)$ the *set of neighbor nodes* of the variable-node n and of the check-node m, respectively. The number of elements of $\mathcal{H}(n)$ (or $\mathcal{H}(m)$) is referred to as the *node-degree*.

The Tanner graph representation allows reformulating the *probabilistic decoding* initially proposed by Gallager [13] in terms of Belief-Propagation[1] (BP): an MP algorithm proposed by J. Pearl in 1982 [15] to perform Bayesian inference on trees, but also successfully used on general graphical models [16]. The BP decoding is known to be optimal on cycle-free graphs (in the sense that it outputs the maximum a posteriori estimates of the coded bits), but can also be successfully applied to decode linear codes defined by graphs with cycles, which is actually the case of all practical codes. However, in practical applications, the BP algorithm might be disadvantaged by its computational complexity and its sensitivity to the channel noise density estimation (inaccurate estimation of the channel noisy density may cause significant degradation of the BP performance).

1.2.2 Min-Sum Decoding

One way to deal with complexity and numerical instability issues is to simplify the computation of messages exchanged within the BP decoding. The most complex step of the BP decoding is the computation of check-to-variable messages, which makes use of computationally intensive hyperbolic tangent functions. The Min-Sum (MS) algorithm is aimed at reducing the computational complexity of the BP, by using max-log approximations of the parity-check to coded-bit messages [17–19]. The only operations required by the MS decoding are additions, comparisons, and sign (± 1) products, which solves the complexity and numerical instability problems. The performance of the MS decoding

[1] Also referred to as Sum-Product (SP)

is also known to be independent of the channel noise density estimation, for most of the usual channel models.

For the sake of simplicity, we only consider transmissions over *binary-input* memoryless noisy channels, and assume that the channel input alphabet is $\{-1,+1\}$, with the usual convention that $+1$ corresponds to the 0-bit, and -1 corresponds to the 1-bit. We further consider a codeword $\underline{\boldsymbol{x}} = (x_1,\ldots,x_N) \in \{-1,+1\}^N$ and denote by $\underline{\boldsymbol{y}} = (y_1,\ldots,y_N)$ the received word. The following notations will be used throughout the chapter, with respect to message passing decoders:

- γ_n is the log-likelihood ratio (LLR) value of x_n according to the received y_n value; it is also referred to as the *a priori information* of the decoder concerning the variable-node n;
- $\tilde{\gamma}_n$ is the *a posteriori information* (LLR value) of the decoder concerning the variable-node n;
- $\alpha_{m,n}$ is the variable-to-check message sent from variable-node n to check-node m;
- $\beta_{m,n}$ is the check-to-variable message sent from check-node m to variable-node n.

The (infinite precision) MS decoding is described in Algorithm 1. It consists of an initialization step (in which variable-to-check messages are initialized according to the a priori information of the decoder), followed by an iteration loop, where each iteration comprises three main steps as follows:

- **CN-processing** (check-node processing step): computes the check-to-variable messages $\beta_{m,n}$;
- **VN-processing** (variable-node processing step): computes the variable-to-check messages $\alpha_{m,n}$;
- **AP-update** (a posteriori information update step): computes the a posteriori information $\tilde{\gamma}_n$.

Moreover, each iteration also comprises a *hard decision* step, in which each transmitted bit is estimated according to the sign of the a posteriori information, and a *syndrome check* step, in which the syndrome of the estimated word is computed. The MS decoding stops when whether the syndrome is $+1$ (the estimated word is a codeword) or a maximum number of iterations is reached.

The a priori information (LLR) of the decoder is defined by $\gamma_n = \log \dfrac{\Pr(x_n = +1 \mid y_n)}{\Pr(x_n = -1 \mid y_n)}$, and for the two following channel models (predominantly used in this work), it can be computed as follows:

- For the Binary Symmetric Channel (BSC), $\underline{\boldsymbol{y}} \in \{-1,+1\}^N$ is obtained by flipping each entry of $\underline{\boldsymbol{x}}$ with some probability ε, referred to as the channel's crossover probability. Consequently:

$$\gamma_n = \log\left(\frac{1-\varepsilon}{\varepsilon}\right) y_n \tag{1.1}$$

- For the Binary-Input Additive White Gaussian Noise (BI-AWGN) channel, $\underline{\boldsymbol{y}} \in \mathbb{R}^N$ is obtained by $y_n = x_n + z_n$, where z_n is the white Gaussian noise with variance σ^2. It follows that:

$$\gamma_n = \frac{2}{\sigma^2} y_n \tag{1.2}$$

Remark: It can be easily seen that if the a priori information vector $\underline{\boldsymbol{\gamma}} = (\gamma_1,\ldots,\gamma_N)$ is multiplied by a constant value, this value will factor out from all the processing steps in Algorithm 1 (throughout the decoding iterations), and therefore it will not affect in any way the decoding process. It follows that for both the BSC and BI-AWGN channel models, one can simply define the a priori information of the decoder by $\gamma_n = y_n$, $\forall n = 1,\ldots,N$.

Algorithm 1 Min-Sum (MS) decoding

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

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

Initialization

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

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

Iteration Loop

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

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

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

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

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

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

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

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

End Iteration Loop

1.3 Error Injection and Probabilistic Models for Noisy Computing

1.3.1 Noisy Message-Passing decoders

The model for noisy MP decoders proposed in [1] incorporates two different sources of noise: *computation noise* due to noisy logic in the processing units, and *message-passing noise* due to noisy wires (or noisy memories) used to exchange messages between neighbor nodes.

- The computation noise is modeled as a random variable, which the variable-node or the check-node processing depends on. Put differently, an outgoing message from a (variable or check) node depends not only on the incoming messages to that node (including the a priori information for the variable-node processing), but also on the realization of a random variable which is assumed to be independent of the incoming messages.
- The message-passing noise is simply modeled as a noisy channel. Hence, transmitting a message over a noisy wire is emulated by passing that message through the corresponding noisy channel.

However, in [1] it has been noted that *"there is no essential loss of generality by combining computation noise and message-passing noise into a single form of noise"* (see also [20, Lemma 3.1]). Consequently, the approach adopted has been to merge noisy computation into message-passing noise, and to emulate noisy decoders by passing the exchanged messages through different noisy channel models. Thus, the noisy Gallager-A decoder has been emulated by passing the exchanged messages over independent and identical BSC wires, while the noisy BP decoder has been emulated by corrupting the exchanged messages with bounded and symmetrically distributed additive noise (*e.g.* uniform noise or truncated Gaussian noise).

The approach we follow in this work differs from the one in [1] in that the computation noise is modeled at the lower level of *arithmetic and logic operations* that compose the variable-node and check-node processing units. This finer-grained noise modeling is aimed at determining the level of noise that can be tolerated in each type of operation. As the main focus of this work is on computation noise, we shall consider that messages are exchanged between neighbor nodes through error-free wires

(or memories). However, we note that *this work can readily be extended to include different error models for the message-passing noise* (as defined in [1]). Alternatively, we may assume that the message-passing noise is merged into the computation noise, in the sense that adding noise in wires would modify the probabilistic model of the noisy logic or arithmetic operations.

1.3.2 Error Injection Models

We only consider the case of finite-precision operations, meaning that the inputs (operands) and the output of the operator are assumed to be bounded integer numbers. We simulate a noisy operator by injecting errors into the output of the noiseless one. In the following, $\mathcal{V} \subset \mathbb{Z}$ denotes a finite set consisting of all the possible outputs of the noiseless operator.

Definition 1 *An* error injection model *on $\mathcal{V}$, denoted by $(\mathcal{E}, p_{\mathcal{E}}, \imath \mid \mathcal{V})$, is given by:*

- *A finite* error set $\mathcal{E} \subset \mathbb{Z}$ *together with a probability mass function $p_{\mathcal{E}} : \mathcal{E} \to [0, 1]$, referred to as the* error distribution*;*
- *A function $\imath : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$, referred to as the* error injection function*.*

For a given set of inputs, the output of the noisy operator is the random variable defined by $\imath(v, e)$, where $v \in \mathcal{V}$ is the corresponding output of the noiseless operator, and e is drawn randomly from $\mathcal{E}$ according to the probability distribution $p_{\mathcal{E}}$.

The error injection probability is defined by

$$p_0 = \frac{1}{|\mathcal{V}|} \sum_v \sum_e \bar{\delta}^v_{\imath(v,e)} p_{\mathcal{E}}(e), \tag{1.3}$$

where $\bar{\delta}^v_{\imath(v,e)} = 0$ if $v = \imath(v, e)$, and $\bar{\delta}^v_{\imath(v,e)} = 1$ if $v \neq \imath(v, e)$. In other word, $p_0 = \Pr(v \neq \imath(v, e))$, assuming that v is drawn uniformly from $\mathcal{V}$ and e is drawn from $\mathcal{E}$ according to $p_{\mathcal{E}}$.

The above definition makes some implicit assumptions which are discussed below.

- The set of possible outputs of the noisy operator is the same as the set of possible outputs of the noiseless operator ($\mathcal{V}$). This is justified by the fact that, in most common cases, $\mathcal{V}$ is the set of all (signed or unsigned) integers that can be represented by a given number of bits. Thus, error injection will usually alter the bit values, but not the number of bits.
- The injected error does not depend on the output value of the noiseless operator and, consequently, neither on the given set of inputs. In other words, the injected error is independent on the data processed by the noiseless operator. The validity of this assumption does actually depend on the size of the circuit implementing the operator. Indeed, this assumption tends to hold fairly well for large circuits [21], but becomes more tenuous as the circuit size decreases.

Obviously, it would be possible to define more general error injection models, in which the injected error would depend on the data (currently and/or previously) processed by the operator. Such an error injection model would certainly be more realistic, but it would also make it very difficult to analytically characterize the behavior of noisy MP decoders. As a side effect, the decoding error probability would be dependent on the transmitted codeword, which would prevent the use of the *density evolution* technique for the analysis of the asymptotic decoding performance (since the density evolution technique relies on the all-zero codeword assumption).

However, the fact that the error injection model is data independent does not guarantee that the decoding error probability is independent of the transmitted codeword. In order for this to happen, the error injection model must also satisfy a *symmetry condition* that can be stated as follows.

Definition 2 *An error injection model $(\mathcal{E}, p_{\mathcal{E}}, \imath \mid \mathcal{V})$ is said to be* symmetric *if $\mathcal{V}$ is symmetric around the origin (meaning that $v \in \mathcal{V} \Leftrightarrow -v \in \mathcal{V}$, but 0 does not necessarily belong to $\mathcal{V}$), and the following equality holds*

$$\sum_{\{e \mid \imath(v,e)=w\}} p_{\mathcal{E}}(e) \quad = \sum_{\{e \mid \imath(-v,e)=-w\}} p_{\mathcal{E}}(e), \quad \forall v, w \in \mathcal{V} \tag{1.4}$$

The meaning of the symmetry condition is as follows. Let V be a random variable on $\mathcal{V}$. Let $\phi_V^{(\imath)}$ and $\phi_{-V}^{(\imath)}$ denote the probability mass functions of the random variables obtained by injecting errors in the output of V and $-V$, respectively. Then the above symmetry condition is satisfied if and only if for any V the following equality holds

$$\phi_V^{(\imath)}(w) = \phi_{-V}^{(\imath)}(-w), \ \forall w \in \mathcal{V} \tag{1.5}$$

A particular case in which the symmetry condition is fulfilled is when $\imath(-v, e) = -\imath(v, e)$, for all $v \in \mathcal{V}$ and $e \in \mathcal{E}$. In this case, the error injection model is said to be *highly symmetric*.

Messages exchanged within message-passing decoders are generally in *belief-format*, meaning that the sign of the message indicates the bit estimate and the magnitude of the message the confidence level. As a consequence, errors occurring on the sign of the exchanged messages are expected to be more harmful than those occurring on their magnitude. This motivates the following definition, which will be used in the following section (see also the discussion in Section 1.4.3).

Definition 3 *An error injection model $(\mathcal{E}, p_{\mathcal{E}}, \imath \mid \mathcal{V})$ is said to be* sign-preserving *if for any $v \in \mathcal{V}$ and $e \in \mathcal{E}$, v and $\imath(v, e)$ are either both non-negative (≥ 0) or both non-positive (≤ 0).*

1.3.3 Bitwise-XOR Error Injection

We focus now on the two main symmetric error injection models that will be used in this work. Both models are based on a bitwise XOR operation between the noiseless output v and the error e. The two models differ in the definition of the error set $\mathcal{E}$, which is chosen such that the bitwise XOR operation may or may not affect the sign of the noiseless output. In the first case, the bitwise XOR error injection model is said to be *full-depth*, while in the second it is said to be *sign-preserving*. These error injection models are rigorously defined below.

In the following, we fix $\theta \geq 2$ and set $\mathcal{V} = \{-\Theta, \ldots, -1, 0, +1, \ldots, +\Theta\}$, where $\Theta = 2^{\theta-1} - 1 \geq 1$. We also fix a *signed number binary representation*, which can be any of the *sign-magnitude*, *one's complement*, or *two's complement* representation. There are exactly 2^θ signed numbers that can be represented by θ bits in any of the above formats, one of which does not belong to $\mathcal{V}$ (note that $\mathcal{V}$ contains only $2\Theta + 1 = 2^\theta - 1$ elements for symmetry reasons!). We denote this element by ζ. Hence:

- In sign-magnitude format, $\zeta = -0$, with binary representation $10 \cdots 0$;
- In one's complement format, $\zeta = -0$, with binary representation $11 \cdots 1$;
- In two's complement format, $\zeta = -(\Theta + 1)$, with binary representation $10 \cdots 0$.

For any $u, v \in \mathcal{V}$, we denote by $u \wedge v$ the bitwise XOR operation between u and v. From the above discussion, it follows that $u \wedge v \in \mathcal{V} \cup \{\zeta\}$.

Full-depth error injection

For this error model, the error set is $\mathcal{E} = \mathcal{V}$. The error injection probability is denoted by p_0, and all the possible error values $e \neq 0$ are assumed to occur with the same probability (for symmetry reasons). It follows that the error distribution function is given by $p_{\mathcal{E}}(0) = 1 - p_0$ and $p_{\mathcal{E}}(e) = \frac{p_0}{2\Theta}$, $\forall e \neq 0$. Finally, the error injection function is defined by:

$$\imath(v, e) = \begin{cases} v \wedge e, & \text{if } v \wedge e \in \mathcal{V} \\ e, & \text{if } v \wedge e = \zeta \end{cases} \tag{1.6}$$

Table 1.1: Example of sign-preserving bitwise-xor error injection

	integer	2's complement binary representation				
noiseless output: v	-11	1	0	1	0	1
error: e	6		0	1	1	0
noisy output: $\imath(v,e)$	-13	1	0	0	1	1
bit position		$\theta=5$	4	3	2	1

Sign-preserving error injection

For this error model, the error set is $\mathcal{E} = \{0, +1, \ldots, +\Theta\}$. The error injection probability is denoted by p_0, and all the possible error values $e \neq 0$ are assumed to occur with the same probability (for symmetry reasons). It follows that the error distribution function is given by $p_{\mathcal{E}}(0) = 1 - p_0$ and $p_{\mathcal{E}}(e) = \frac{p_0}{\Theta}$, $\forall e \neq 0$. Finally, the error injection function is defined by:

$$\imath(v,e) = \begin{cases} v \wedge e, & \text{if } v \neq 0 \text{ and } v \wedge e \in \mathcal{V} \\ \pm e, & \text{if } v = 0 \\ 0, & \text{if } v \wedge e = \zeta \end{cases} \tag{1.7}$$

In the above definition, $\imath(0,e)$ is randomly set to either $-e$ or $+e$, with equal probability (this is due once again to symmetry reasons). Note also that the last two conditions, namely $v = 0$ and $v \wedge e = \zeta$, cannot hold simultaneously (since $e \neq \zeta$).

Finally, we note that both of the above models are highly symmetric, if one of the sign-magnitude or the one's complement representation is used. In case that the two's complement representation is used, they are both symmetric, but not highly symmetric.

An example of sign-preserving bitwise-XOR error injection is given in Table 1.1. The number of bits is $\theta = 5$ and two's complement binary representation is used. The sign bit of the error is not displayed, as it is equal to zero for any $e \in \mathcal{E}$. The positions of 1's in the binary representation of e correspond to the positions of the erroneous bits in the noisy output.

Remark: It is also possible to define a *variable depth error injection* model, in which errors are injected in only the λ least significant bits, with $\lambda \leq \theta$. Hence, $\lambda = \theta$ corresponds to the above full-depth model, while $\lambda = \theta - 1$ corresponds to the sign-preserving model. However, for $\lambda < \theta - 1$ such a model will **not** be symmetric, if the the two's complement representation is used.

1.3.4 Output-Switching Error Injection

A particular case is represented by error injection on binary output. Assuming that $\mathcal{V} = \{0, 1\}$, the *bit-flipping* error injection model is defined as follows. The error set is $\mathcal{E} = \{0, 1\}$, with error distribution function given by $p_{\mathcal{E}}(0) = 1 - p_0$ and $p_{\mathcal{E}}(1) = p_0$, where p_0 is the error injection probability, and the error injection function is given by $\imath(v,e) = v \wedge e$. Put differently, the error injection model flips the value of a bit in $\mathcal{V}$ with probability p_0.

Clearly, the above error injection model can be applied on any set $\mathcal{V}$ with two elements, by switching one value to another with probability p_0. In this case, we shall refer to this error injection model as *output-switching*, rather than bit-flipping.

Moreover, if one takes $\mathcal{V} = \{-1, +1\}$ (with the usual 0,1 to ± 1 conversion), it can be easily verified that this error injection model is highly symmetric.

1.3.5 Probabilistic models for noisy adders, comparators and XOR-gates

In this section we describe the probabilistic models for noisy adders, comparators and xor-gates, built upon the above error injection models. These probabilistic models will be used in the next section, in order to emulate the noisy implementation of the quantized (finite-precision) MS decoder.

Noisy adder model

We consider a θ-bit adder, with $\theta \geq 2$. The inputs and the output of the adder are assumed to be in $\mathcal{V} = \{-\Theta, \ldots, -1, 0, +1, \ldots, +\Theta\}$, where $\Theta = 2^{\theta-1} - 1$.

We denote by $\mathbf{s}_{\mathcal{V}} : \mathbb{Z} \to \mathcal{V}$, the θ-bit saturation map, defined by:

$$\mathbf{s}_{\mathcal{V}}(v) = \begin{cases} -\Theta, & \text{if } v < -\Theta \\ v, & \text{if } v \in \mathcal{V} \\ +\Theta, & \text{if } v > +\Theta \end{cases} \tag{1.8}$$

For inputs $(x, y) \in \mathcal{V}$, the output of the noiseless adder is defined as $s_{\mathcal{V}}(x + y)$. Hence, for a given error injection model $(\mathcal{E}, p_{\mathcal{E}}, \imath \mid \mathcal{V})$, the output of the noisy adder is given by:

$$\mathbf{a}_{\text{pr}}(x, y) = \imath\left(\mathbf{s}_{\mathcal{V}}(x + y), e\right), \tag{1.9}$$

where e is drawn randomly from $\mathcal{E}$ according to the probability distribution $p_{\mathcal{E}}$. The *error probability of the noisy adder*, assuming uniformly distributed inputs, is equal to the error injection probability (parameter p_0 defined in (1.3)), and will be denoted in the sequel by p_a.

Noisy comparator model

Let $\mathbf{lt}$ denote the noiseless *less than* operator, defined by $\mathbf{lt}(x, y) = 1$ if $x < y$, and $\mathbf{lt}(x, y) = 0$ otherwise. The *noisy less than* operator, denoted by $\mathbf{lt}_{\text{pr}}$, is defined by injecting errors on the output of the noiseless one, according to the bit-flipping model defined in Section 1.3.4. In other words, the output of the noiseless $\mathbf{lt}$ operator is flipped with some probability value, which will be denoted in the sequel by p_c.

Finally, the *noisy minimum* operator is defined by:

$$\mathbf{m}_{\text{pr}}(x, y) = \begin{cases} x, & \text{if } \mathbf{lt}_{\text{pr}}(x, y) = 1 \\ y, & \text{if } \mathbf{lt}_{\text{pr}}(x, y) = 0 \end{cases} \tag{1.10}$$

Noisy XOR model

The noisy XOR operator, denoted by $\mathbf{x}_{\text{pr}}$ is defined by flipping the output of the noiseless operator with some probability value, which will be denoted in the sequel by p_x (according to the bit-flipping error injection model in Section 1.3.4). It follows that:

$$\mathbf{x}_{\text{pr}}(x, y) = \begin{cases} x \wedge y, & \text{with probability } 1 - p_x \\ \overline{x \wedge y}, & \text{with probability } p_x \end{cases} \tag{1.11}$$

Assumption: We further assume that the inputs and the output of the XOR operator may take values in either $\{0, 1\}$ or $\{-1, +1\}$ (using the usual 0,1 to ± 1 conversion). This assumption will be implicitly made throughout the chapter.

Remark: As a general rule, we shall refer to a noisy operator according to its underlying error injection model. For instance, a sign-preserving (resp. full-depth or sign-preserving bitwise-XORed) noisy adder, is a noisy adder whose underlying error injection model is sign-preserving (resp. one of the bitwise-XOR error injection models defined in Section 1.3.3). We shall also say that a noisy operator is *(highly) symmetric* if its underlying error injection model is so.

1.3.6 Nested Operators

As it can be observed from Algorithm 1, several arithmetic/logic operations must be nested[2] in order to compute the exchanged messages. Since all these operations (additions, comparisons, XOR) are

[2]For instance, $(d_n - 1)$ additions – where d_n denotes the degree of the variable-node n – are required in order to compute each $\alpha_{m,n}$ message. Similarly, each $\beta_{m,n}$ message requires $(d_m - 2)$ XOR operations and $(d_m - 2)$ comparisons.

commutative, the way they are nested does not have any impact on the infinite-precision MS decoding. However, this is no longer true for finite-precision decoding, especially in case of noisy operations. Therefore, one needs an assumption about how operators extend from two to more inputs.

Our assumption is the following. For $n \geq 2$ inputs, we randomly pick any two inputs and apply the operator on this pair. Then we replace the pair by the obtained output, and repeat the above procedure until there is only one output (and no more inputs) left.

The formal definition goes as follows. Let $\Omega \subset \mathbb{Z}$ and $\omega : \Omega \times \Omega \to \Omega$ be a noiseless or noisy operator with two operands. Let $\{x_i\}_{i=1:n} \subset \Omega$ be an unordered set of n operands. We define:

$$\omega\left(\{x_i\}_{i=1:n}\right) = \omega(\cdots(\omega(x_{\pi(1)}, x_{\pi(2)}), \cdots), x_{\pi(n)}),$$

where π is a random permutation of $1, \ldots, n$.

1.4 Noisy Min-Sum Decoding

1.4.1 Finite-Precision Min-Sum Decoder

We consider a finite-precision MS decoder, in which the a priori information (γ_n) and the exchanged messages ($\alpha_{m,n}$ and $\beta_{m,n}$) are quantized on q bits. The a posteriori information ($\tilde{\gamma}_n$) is quantized on $\tilde{q}$ bits, with $\tilde{q} > q$ (usually $\tilde{q} = q + 1$, or $\tilde{q} = q + 2$). We further denote:

- $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, Q\}$, where $Q = 2^{q-1} - 1$, the alphabet of both the a priori information and the exchanged messages;
- $\widetilde{\mathcal{M}} = \{-\widetilde{Q}, \ldots, -1, 0, +1, \ldots, \widetilde{Q}\}$, where $\widetilde{Q} = 2^{\tilde{q}-1} - 1$, the alphabet of the a posteriori information;
- $\mathbf{q} : \mathcal{Y} \to \mathcal{M}$, a quantization map, where $\mathcal{Y}$ denotes the channel output alphabet;
- $\mathbf{s}_{\mathcal{M}} : \mathbb{Z} \to \mathcal{M}$, the q-bit saturation map (defined in a similar manner as in (1.8));
- $\mathbf{s}_{\widetilde{\mathcal{M}}} : \mathbb{Z} \to \widetilde{\mathcal{M}}$, the $\tilde{q}$-bit saturation map

Remark: The quantization map $\mathbf{q}$ determines the q-bit quantization of the decoder soft input. Since $\mathbf{q}$ is defined on the channel input (*i.e.* y_n values), it must also encompass the computation of the corresponding LLR values, whenever is necessary (see also the Remark at the end of Section 1.2.2).

Saturation maps $\mathbf{s}_{\mathcal{M}}$ and $\mathbf{s}_{\widetilde{\mathcal{M}}}$ define the finite-precision saturation of the exchanged messages and of the a posteriori information, respectively.

1.4.2 Noisy Min-Sum Decoder

The noisy (finite-precision) MS decoding is presented in Algorithm 2. We assume that $\tilde{q}$-bit adders are used to compute both $\alpha_{m,n}$ messages in the **VN-processing** step, and $\tilde{\gamma}_n$ values in the **AP-update** processing step. This is usually the case in practical implementations[3], and allows us to use the same type of adder in both processing steps. This assumption explains as well the q-bit saturation of $\alpha_{m,n}$ messages in the **VN-processing** step. Note also that the saturation of $\tilde{\gamma}_n$ values is actually done within the adder (see Equation (1.9)).

Finally, we note that the *hard decision* and the *syndrome check* steps in Algorithm 2 are assumed to be *noiseless*. We note however that the syndrome check step is optional, and if missing, the decoder stops when the maximum number of iterations is reached.

[3]In practical implementation, the $\tilde{\gamma}_n$ is computed first, and then $\alpha_{m,n}$ is obtained from $\tilde{\gamma}_n$ by subtracting the incoming $\beta_{m,n}$ message

Algorithm 2 Noisy Min-Sum (Noisy-MS) decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word
Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{-1, +1\}^N$ ▷ estimated codeword
Initialization
  for all $n = 1, \ldots, N$ **do** $\gamma_n = \mathbf{q}(y_n)$;
  for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\alpha_{m,n} = \gamma_n$;
Iteration Loop
  for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**
    $\beta_{m,n} = \mathbf{x}_{\text{pr}}\left(\{\text{sgn}(\alpha_{m,n'})\}_{n' \in \mathcal{H}(m)\setminus n}\right) \mathbf{m}_{\text{pr}}\left(\{|\alpha_{m,n'}|\}_{n' \in \mathcal{H}(m)\setminus n}\right)$;
  for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**
    $\alpha_{m,n} = \mathbf{a}_{\text{pr}}\left(\{\gamma_n\} \cup \{\beta_{m',n}\}_{m' \in \mathcal{H}(n)\setminus m}\right)$;
    $\alpha_{m,n} = \mathbf{s}_{\mathcal{M}}\left(\alpha_{m,n}\right)$;
  for all $n = 1, \ldots, N$ **do** ▷ **AP-update**
    $\tilde{\gamma}_n = \mathbf{a}_{\text{pr}}\left(\{\gamma_n\} \cup \{\beta_{m,n}\}_{m \in \mathcal{H}(n)}\right)$;
  for all $\{v_n\}_{n=1,\ldots,N}$ **do** $\hat{x}_n = \text{sgn}(\tilde{\gamma}_n)$; ▷ hard decision
  if $\underline{\hat{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check
End Iteration Loop

1.4.3 Sign-Preserving Properties

Let $\mathbf{U}$ denote any of the VN-processing or CN-processing units of the noiseless MS decoder. We denote by $\mathbf{U}_{\text{pr}}$ the corresponding unit of the noisy MS decoder. We say that $\mathbf{U}_{\text{pr}}$ is *sign-preserving* if for any incoming messages and any noise realization, the outgoing message is of the same sign as the message obtained when the same incoming messages are supplied to $\mathbf{U}$.

Clearly, CN_{pr} is sign-preserving if and only if the XOR-operator is noiseless ($p_x = 0$). In case that the noisy XOR-operator severely degrades the decoder performance, it is possible to increase its reliability by using classical fault-tolerant techniques (as for instance modular redundancy, or multi-voltage design by increasing the supply voltage of the corresponding XOR-gate). The price to pay, when compared to the size or the energy consumption of the whole circuit, would be reasonable.

Concerning the VN-processing, it is worth noting that the VN_{pr} is **not** sign-preserving, even if the noisy adder is. This is due to the fact that multiple adders must be nested in order to complete the VN-processing. However, a sign-preserving adder might have several benefits. First, the error probability of the sign of variable-node messages would be lowered, which would certainly help the decoder. Second, if the noisy adder is sign-preserving and all the variable-node incoming messages have the same sign, then the VN_{pr} does preserve the sign of the outgoing message. Put differently, in case that all the incoming messages agree on the same hard decision, the noisy VN-processing may change the confidence level, but cannot change the decision. This may be particularly useful, especially during the last decoding iterations.

Finally, the motivation behind the sign-preserving noisy adder model is to investigate its possible benefits on the decoder performance. If the benefits are worth it (*e.g.* one can ensure a target performance of the decoder), the sign-bit of the adder could be protected by using classical fault-tolerant solutions.

1.5 Density Evolution

1.5.1 Concentration and Convergence Properties

First, we note that our definition of *symmetry* is slightly more general than the one used in [1]. Indeed, even if all the error injection models used within the noisy MS decoder are *symmetric*, the noisy MS decoder does not necessarily verify the ***symmetry*** property from [1]. However, this property is verified in case of *highly symmetric* fault injection[4]. Nevertheless, the concentration and convergence properties proved in [1] for ***symmetric*** noisy message-passing decoders, can easily be generalized to our definition of *symmetry*.

We summarize below the most important results; the proof relies essentially on the same arguments as in [1]. We consider an *ensemble* of LDPC codes, with length N and fixed degree distribution polynomials [22]. We choose a random code $\mathbf{C}$ from this ensemble and assume that a random codeword $\underline{\boldsymbol{x}} \in \{-1,+1\}^N$ is sent over a binary-input memoryless symmetric channel. We fix some number of decoding iterations $\ell > 0$, and denote by $E_{\mathbf{C}}^{(\ell)}(\underline{\boldsymbol{x}})$ the expected fraction of incorrect *messages*[5] at iteration ℓ.

Theorem 1 *Assume that all the error injection models used within the MS decoder are symmetric. Then, the following properties hold:*

1. *[*Conditional Independence of Error*] For any decoding iteration $\ell > 0$, the expected fraction of incorrect messages $E_{\mathbf{C}}^{(\ell)}(\underline{\boldsymbol{x}})$ does not depend on $\underline{\boldsymbol{x}}$. Therefore, we may define $E_{\mathbf{C}}^{(\ell)} := E_{\mathbf{C}}^{(\ell)}(\underline{\boldsymbol{x}})$.*
2. *[*Cycle-Free Case*] If the graph of $\mathbf{C}$ contains no cycles of length 2ℓ or less, the expected fraction of incorrect messages $E_{\mathbf{C}}^{(\ell)}$ does not depend on the code $\mathbf{C}$ or the code-length N, but only on the degree distribution polynomials; in this case, it will be further denoted by $E_{\infty}^{(\ell)}(\underline{\boldsymbol{x}})$.*
3. *[*Concentration Around the Cycle-Free Case*] For any $\delta > 0$, the probability that $E_{\mathbf{C}}^{(\ell)}$ lies outside the interval $\left(E_{\infty}^{(\ell)}(\underline{\boldsymbol{x}}) - \delta, E_{\infty}^{(\ell)}(\underline{\boldsymbol{x}}) + \delta\right)$ converges to zero exponentially fast in N.*

1.5.2 Density Evolution Equations

In this section we derive density evolution equations for the noisy finite-precision MS decoding for a regular (d_v, d_c) LDPC code. The study can be easily generalized to irregular LDPC codes, simply by averaging according to the degree distribution polynomials.

The objective of the density evolution technique is to recursively compute the probability mass functions of exchanged messages, through the iterative decoding process. This is done under the independence assumption of exchanged messages, holding in the asymptotic limit of the code length, in which case the decoding performance converges to the cycle-free case. Due to the symmetry of the decoder, the analysis can be further simplified by assuming that the all-zero codeword is transmitted through the channel. We note that our analysis applies to any memoryless symmetric channel.

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

$$\begin{aligned} C(z) &= \Pr(\gamma = z), && \forall z \in \mathcal{M} \\ \widetilde{C}^{(\ell)}(\tilde{z}) &= \Pr(\tilde{\gamma}^{(\ell)} = \tilde{z}), && \forall \tilde{z} \in \widetilde{\mathcal{M}} \\ A^{(\ell)}(z) &= \Pr\left(\alpha^{(\ell)} = z\right), && \forall z \in \mathcal{M} \\ B^{(\ell)}(z) &= \Pr\left(\beta^{(\ell)} = z\right), && \forall z \in \mathcal{M} \end{aligned}$$

[4] According to the probabilistic models introduced in Section 1.3.5, the noisy comparator and the noisy XOR-operator are highly symmetric, but the noisy adder does not necessarily be so!

[5] Here, *"messages"* may have any one of the three following meanings: "variable-node messages", or "check-node messages", or "a posteriori information values".

Expression of the input probability mass function C

The probability mass function C depends only on the channel and the quantization map $\mathbf{q} : \mathcal{Y} \to \mathcal{M}$, where $\mathcal{Y}$ denotes the channel output alphabet (Section 1.4.1). We also note that for $\ell = 0$, we have $A^{(0)} = C$.

We give below the expression of C for the BSC and the BI-AWGN channel models (see Section 1.2.2). For the BSC, the channel output alphabet is $\mathcal{Y} = \{-1, +1\}$, while for the BI-AWGN channel, $\mathcal{Y} = \mathbb{R}$.

Let μ be a positive number, such that $\mu \leq Q$. The quantization map $\mathbf{q}_\mu$ is defined as follows:

$$\mathbf{q}_\mu : \mathcal{Y} \to \mathcal{M}, \quad \mathbf{q}_\mu(y) = \mathbf{s}_\mathcal{M}([\mu{\cdot}y]), \tag{1.12}$$

where $[\mu{\cdot}y]$ denotes the nearest integer to $\mu{\cdot}y$, and $\mathbf{s}_\mathcal{M}$ is the saturation map (Section 1.4.1). For the BSC, we will further assume that μ is an integer. It follows that $\mathbf{q}_\mu(y) = \mu{\cdot}y$, $\forall y \in \mathcal{Y} = \{-1, +1\}$.

Considering the all-zero $(+1)$ codeword assumption, the probability mass function C can be computed as follows.

- For the BSC with crossover probability ε:

$$C(z) = \begin{cases} 1-\varepsilon, & \text{if } z = \mu \\ \varepsilon, & \text{if } z = -\mu \\ 0, & \text{otherwise} \end{cases} \tag{1.13}$$

- For the BI-AWGN channel with noise variance σ^2:

$$C(z) = \begin{cases} 1 - q\left(\frac{-Q+0.5-\mu}{\mu\sigma}\right), & \text{if } z = -Q \\ q\left(\frac{z-0.5-\mu}{\mu\sigma}\right) - q\left(\frac{z+0.5-\mu}{\mu\sigma}\right), & \text{if } -Q < z < +Q \\ q\left(\frac{Q-0.5-\mu}{\mu\sigma}\right), & \text{if } z = +Q \end{cases} \tag{1.14}$$

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

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

In the sequel, we make the convention that $\Pr(\mathrm{sgn}(0) = 1) = \Pr(\mathrm{sgn}(0) = -1) = 1/2$. The following notation will be used:

- $A_{[x,y]} = \displaystyle\sum_{z=x}^{y} A(z)$, for $x \leq y \in \mathcal{M}$
- $A_{[0^+,y]} = \dfrac{1}{2}A(0) + \displaystyle\sum_{z=1}^{y} A(z)$, for $y \in \mathcal{M},\ y > 0$
- $A_{[x,0^-]} = \dfrac{1}{2}A(0) + \displaystyle\sum_{z=x}^{-1} A(z)$, for $x \in \mathcal{M},\ x < 0$

For the sake of simplicity, we drop the iteration index, thus $B := B^{(\ell)}$ and $A := A^{(\ell-1)}$. We proceed by recursion on $i = 2, \ldots, d_c - 1$, where d_c denotes the check-node degree.
Let $\beta_1 := \alpha_1$, and for $i = 2, \ldots, d_c - 1$ define:

$$\beta_i = \mathbf{x}_{\mathrm{pr}}(\mathrm{sgn}(\beta_{i-1}), \mathrm{sgn}(\alpha_i))\mathbf{m}_{\mathrm{pr}}(|\beta_{i-1}|, |\alpha_i|)$$

Let also B_{i-1} and B_i denote the probability mass functions of β_{i-1} and β_i, respectively (hence, $B_1 = A$).

First of all, for $z = 0$, we have:
$B_i(0) = \Pr(\beta_i = 0) = A(0)B_{i-1}(0) + [B_{i-1}(0)(1 - A(0)) + A(0)(1 - B_{i-1}(0))]\,(1 - p_c)$.

For $z \neq 0$, we proceed in several steps as follows:

For $z > 0$:

$$\begin{aligned} F'_i(z) &\stackrel{\text{def}}{=} \Pr(\beta_i \geq z \mid p_x = 0) \\ &= \left[B_{i-1[0^+,z-1]}A_{[z,Q-1]} + A_{[0^+,z-1]}B_{i-1[z,Q-1]}\right] p_c \\ &+ \left[B_{i-1[1-z,0^-]}A_{[-Q,-z]} + A_{[1-z,0^-]}B_{i-1[-Q,-z]}\right] p_c \\ &+ B_{i-1[z,Q-1]}A_{[z,Q-1]} + B_{i-1[-Q,-z]}A_{[-Q,-z]} \end{aligned}$$

$$\begin{aligned} F_i(z) &\stackrel{\text{def}}{=} \Pr(\beta_i \geq z) \\ &= (1 - p_x).F'_i(z) + p_x.G'_i(-z) \\ B_i(z) &= \Pr(\beta_i = z) = F_i(z) - F_i(z+1) \end{aligned}$$

For $z < 0$:

$$\begin{aligned} G'_i(z) &\stackrel{\text{def}}{=} \Pr(\beta_i \leq z \mid p_x = 0) \\ &= \left[B_{i-1[0^+,-z-1]}A_{[-Q,z]} + A_{[0^+,-z-1]}B_{i-1[-Q,z]}\right] p_c \\ &+ \left[B_{i-1[-z,Q-1]}A_{[z+1,0^-]} + A_{[-z,Q-1]}B_{i-1[z+1,0^-]}\right] p_c \\ &+ B_{i-1[-z,Q-1]}A_{[-Q,z]} + A_{[-z,Q-1]}B_{i-1[-Q,z]} \end{aligned}$$

$$\begin{aligned} G_i(z) &\stackrel{\text{def}}{=} \Pr(\beta_i \geq z) \\ &= (1 - p_x).G'_i(z) + p_x.F'_i(-z) \\ B_i(z) &= \Pr(\beta_i = z) = G_i(z) - G_i(z+1) \end{aligned}$$

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

Expression of $A^{(\ell)}$ as a function of $B^{(\ell)}$ and C

We derive at the same time the *expression of $\widetilde{C}^{(\ell)}$ as a function of $B^{(\ell)}$ and C*.

For simplicity, we drop the iteration index, so $A := A^{(\ell)}$, $B := B^{(\ell)}$, and $\widetilde{C} := \widetilde{C}^{(\ell)}$. We denote by $\left(\mathcal{E}, p_{\mathcal{E}}, \imath \mid \widetilde{\mathcal{M}}\right)$ the error injection model used to define the noisy adder. We decompose each noisy addition into three steps (noiseless infinite-precision addition, saturation, and error injection), and proceed by recursion on $i = 0, 1, \ldots, d_v$, where d_v denotes the variable-node degree:

- For $i = 0$:
  $\Omega_0 \stackrel{\text{def}}{=} \gamma \in \mathcal{M} \subseteq \widetilde{\mathcal{M}}$, $\quad \widetilde{C}_0(\tilde{z}) \stackrel{\text{def}}{=} \Pr(\Omega_0 = \tilde{z}) = \begin{cases} C(\tilde{z}), & \text{if } \tilde{z} \in \mathcal{M} \\ 0, & \text{if } \tilde{z} \in \widetilde{\mathcal{M}} \setminus \mathcal{M} \end{cases}$
- For $i = 1, \ldots, d_v$:
  $\omega_i \stackrel{\text{def}}{=} \Omega_{i-1} + \beta_{m_i,n} \in \mathbb{Z}$, $\quad c_i(w) \stackrel{\text{def}}{=} \Pr(\omega_i = w) = \sum_u \widetilde{C}_{i-1}(u)B(w-u), \forall w \in \mathbb{Z}$

  $\tilde{\omega}_i \stackrel{\text{def}}{=} \mathbf{s}_{\widetilde{\mathcal{M}}}(\omega_i) \in \widetilde{\mathcal{M}}$, $\quad \tilde{c}_i(\tilde{w}) \stackrel{\text{def}}{=} \Pr(\tilde{\omega}_i = \tilde{w}) = \begin{cases} c_i(\tilde{w}), & \text{if } \tilde{w} \in \widetilde{\mathcal{M}} \setminus \{\pm\widetilde{Q}\} \\ \sum_{w \leq -\widetilde{Q}} c_i(w), & \text{if } \tilde{w} = -\widetilde{Q} \\ \sum_{w \geq +\widetilde{Q}} c_i(w), & \text{if } \tilde{w} = +\widetilde{Q} \end{cases}$

  $\Omega_i \stackrel{\text{def}}{=} \imath(\tilde{\omega}_i, e) \in \widetilde{\mathcal{M}}$, $\quad \widetilde{C}_i(\tilde{z}) \stackrel{\text{def}}{=} \Pr(\Omega_i = \tilde{z}) = \sum_{\tilde{\omega}} \sum_e \delta^{\tilde{z}}_{\imath(\tilde{\omega},e)} p_{\mathcal{E}}(e) \tilde{c}_i(\tilde{\omega}), \forall \tilde{z} \in \widetilde{\mathcal{M}}$
  where $\delta^y_x = 1$ if $x = y$, and $\delta^y_x = 0$ if $x \neq y$.

Note that in the definition of Ω_i above, e denotes an error drown from the error set $\mathcal{E}$ according to the error probability distribution $p_{\mathcal{E}}$.

Finally, we have:

- $A = \mathbf{s}_{\mathcal{M}}\left(\widetilde{C}_{d_v-1}\right)$
- $\widetilde{C} = \widetilde{C}_{d_v}$

In the first equation above, applying the saturation operator $\mathbf{s}_{\mathcal{M}}$ on the probability mass function $\widetilde{C}_{d_v-1}$ means that all the probability weights corresponding to values $\tilde{w}$ outside $\mathcal{M}$ must be accumulated to the probability of the corresponding boundary value of $\mathcal{M}$ (that is, either $-Q$ or $+Q$, according to whether $\tilde{w} < -Q$ or $\tilde{w} < +Q$).

Remark: If the noisy adder is defined by one of the bitwise-XOR error injection models (Section 1.3.3), then the third equation from the above recursion (expression of $\widetilde{C}_i$ as a function of $\tilde{c}_i$) may be rewritten as follows:

- Sign-preserving bitwise-XORed noisy adder

$$\widetilde{C}_i(\tilde{z}) = \begin{cases} (1-p_a)\tilde{c}_i(\tilde{z}) + \frac{1}{\widetilde{Q}} p_a \left(\tilde{c}_{i\,[\leq 0^-]} - \tilde{c}_i(z)\right), & \text{if } \tilde{z} < 0 \\ (1-p_a)\tilde{c}_i(0) + \frac{1}{\widetilde{Q}} p_a \left(1 - \tilde{c}_i(0)\right), & \text{if } \tilde{z} = 0 \\ (1-p_a)\tilde{c}_i(\tilde{z}) + \frac{1}{\widetilde{Q}} p_a \left(\tilde{c}_{i\,[\geq 0^+]} - \tilde{c}_i(z)\right), & \text{if } \tilde{z} > 0 \end{cases} \tag{1.15}$$

  where $\tilde{c}_{i\,[\leq 0^-]} = \sum_{\tilde{\omega}<0} \tilde{c}_i(\tilde{\omega}) + \frac{1}{2}\tilde{c}_i(0)$, and $\tilde{c}_{i\,[\geq 0^+]} = \frac{1}{2}\tilde{c}_i(0) + \sum_{\tilde{\omega}>0} \tilde{c}_i(\tilde{\omega})$.

- Full-depth bitwise-XORed noisy adder

$$\widetilde{C}_i(\tilde{z}) = (1-p_a)\tilde{c}_i(\tilde{z}) + \frac{1}{2\widetilde{Q}} p_a \left(1 - \tilde{c}_i(\tilde{z})\right) \tag{1.16}$$

Finally, we note that the density evolution equations for the noiseless finite-precision MS decoder can be obtained by setting $p_a = p_c = p_x = 0$.

1.5.3 Error Probability and Useful and Target-BER regions

Decoding Error Probability

The error probability at decoding iteration ℓ, is defined by:

$$P_e^{(\ell)} = \sum_{\tilde{z}=-\widetilde{Q}}^{-1} \widetilde{C}^{(\ell)}(\tilde{z}) + \frac{\widetilde{C}^{(\ell)}(0)}{2} \tag{1.17}$$

Proposition 1 *The error probability at decoding iteration ℓ is lower-bounded as follows:*

(a) For the sign-preserving bitwise-XORed noisy adder: $P_e^{(\ell)} \geq \frac{1}{2\widetilde{Q}} p_a$.

(b) For the full-depth bitwise-XORed noisy adder: $P_e^{(\ell)} \geq \frac{1}{2} p_a + \frac{1}{4\widetilde{Q}} p_a$.

Proof. (a) Using $\widetilde{C} = \widetilde{C}_{d_v}$ and equations (1.17) and (1.15), it follows that $P_e^{(\ell)} = (1-p_a)\tilde{c}_{d_v\,[\leq 0^-]} + \frac{1}{2\widetilde{Q}} p_a \left(1 - 2\tilde{c}_{d_v\,[\leq 0^-]}\right) + p_a \left(\tilde{c}_{d_v\,[\leq 0^-]} - \frac{1}{2}\tilde{c}_{d_v}(0)\right) \geq (1-p_a)\tilde{c}_{d_v\,[\leq 0^-]} + \frac{1}{2\widetilde{Q}} p_a \left(1 - 2\tilde{c}_{d_v\,[\leq 0^-]}\right) \geq \frac{1}{2\widetilde{Q}} p_a$, since the function $(1-p_a)x + \frac{1}{2\widetilde{Q}} p_a (1-2x)$ is an increasing function of $x \in [0,1]$.
(b) Equations (1.17) and (1.16) imply that $P_e^{(\ell)} = \frac{1}{2} p_a + (1-p_a)\tilde{c}_{d_v\,[\leq 0^-]} + \frac{1}{4\widetilde{Q}} p_a \left(1 - 2\tilde{c}_{d_v\,[\leq 0^-]}\right) \geq \frac{1}{2} p_a + \frac{1}{4\widetilde{Q}} p_a$ □

Note that the above lower bounds are actually inferred from the error injection in the *last (the d_v-th) addition* performed when computing the a posteriori information value. Therefore, these lower bounds are not expected to be tight. However, if the channel error probability is small enough, the sign-preserving lower bound proves to be tight in the asymptotic limit of ℓ (this will be discussed in more details in Section 1.6). Note also that by protecting the sign of the noisy adder, the bound is lowered by a factor of roughly $\widetilde{Q}$, which represents an exponential improvement with respect to the number of bits of the adder.

In the asymptotic limit of the code-length, $P_e^{(\ell)}$ gives the probability of the hard bit estimates being in error at decoding iteration ℓ. For the (noiseless, infinite-precision) BP decoder, the error probability is usually a decreasing function of ℓ. This is no longer true for the noiseless, infinite-precision MS decoder, for which the error probability may increase with ℓ. However, both decoders exhibit a *threshold phenomenon*, separating the region where error probability goes to zero (as the number of decoding iterations goes to infinity), from that where it is bounded above zero [22].

Things get more complicated for the noisy (finite-precision) MS decoder. First, the error probability have a more unpredictable behavior. It does not always converge and it may become periodic[6] when the number of iterations goes to infinity. Second, the error probability is always bounded above zero (Proposition 1), since there is a non-zero probability of fault injection at any decoding iteration. Hence, a decoding threshold, similar to the noiseless case, cannot longer be defined.

Following [1], we define below the notions of useful decoder and target error rate threshold. We consider a channel model depending on a channel parameter χ, such that the channel is degraded by increasing χ (for example, the crossover probability for the BSC, or the noise variance for the BI-AWGN channel). We will use subscript χ to indicate a quantity that depends on χ. Hence, in order to account that $P_e^{(\ell)}$ depends also on the value of the channel parameter, it will be denoted in the following by $P_{e,\chi}^{(\ell)}$.

Useful Region

The first step is to evaluate the channel and hardware parameters yielding a final probability of error (in the asymptotic limit of the number of iterations) less than the *input error probability*. The latter probability is given by $P_{e,\chi}^{(0)} = \sum_{z=-Q}^{-1} C(z) + \frac{1}{2}C(0)$, where C is the probability mass function of the quantized a priori information of the decoder (see Section 1.5.2).

Following [1], the decoder is said to be *useful* if $\left(P_{e,\chi}^{(\ell)}\right)_{\ell>0}$ is convergent, and:

$$P_{e,\chi}^{(\infty)} \stackrel{\text{def}}{=} \lim_{\ell\to\infty} P_{e,\chi}^{(\ell)} < P_{e,\chi}^{(0)} \tag{1.18}$$

The ensemble of the parameters that satisfy this condition constitutes the *useful region* of the decoder.

Target Error Rate Threshold

For noiseless-decoders, the decoding threshold is defined as the supremum channel noise, such that the error probability converges to zero as the number of decoding iterations goes to infinity. However, for noisy decoders this error probability does not converge to zero, and an alternative definition of the decoding threshold has been introduced in [1]. Accordingly, for a target bit-error rate η, the η-threshold is defined[7] by:

$$\chi^*(\eta) = \sup\left\{\chi \mid P_{e,\chi'}^{(\infty)} \text{ exists and } P_{e,\chi'}^{(\infty)} < \eta,\ \forall \chi' \in [0,\chi]\right\} \tag{1.19}$$

[6] In fact, for both BSC and BI-AWGN channels, the only cases we observed, in which the sequence $\left(P_e^{(\ell)}\right)_{\ell>0}$ does not converge, are those cases in which this sequence becomes periodic for ℓ large enough.

[7] In [1], the η-threshold is defined by $\chi^*(\eta) = \sup\left\{\chi \mid P_{e,\chi}^{(\infty)} \text{ exists and } P_{e,\chi}^{(\infty)} < \eta\right\}$, and consequently, there might exist a channel parameter value $\chi' < \chi^*(\eta)$, for which $P_{e,\chi'}^{(\infty)}$ does not exist. In order to avoid this happening, our definition is slightly different from the one in [1].

1.6 Asymptotic analysis of the noisy Min-sum decoder

In this section, the density evolution equations derived previously are used to analyze the asymptotic performance (*i.e.* in the asymptotic limit of both the code length and number of iterations) of the noisy MS decoder.

Unless specified otherwise, the following parameters are used throughout this section:

Code parameters:

- We consider the ensemble of regular LDPC codes with variable-node degree $d_v = 3$ and check-node degree $d_c = 6$

Quantization parameters:

- The a priori information and exchanged messages are quantized on $q = 4$ bits; hence, $Q = 7$ and $\mathcal{M} = \{-7, \ldots, +7\}$.
- The a posteriori information is quantized on $\tilde{q} = 5$ bits; hence, $\widetilde{Q} = 15$ and $\widetilde{\mathcal{M}} = \{-15, \ldots, +15\}$.

We analyze the decoding performance depending on:

- The quantization map $\mathbf{q}_\mu : \mathcal{Y} \to \mathcal{M}$, defined in Equation (1.12). The factor μ will be referred to as the *channel-output scale factor*, or simply the *channel scale factor*.
- The parameters of the noisy adder, comparator, and XOR-operator, defined respectively in Equations (1.9), (1.10), and (1.11).

1.6.1 Numerical results for the BSC

For the BSC, the channel output alphabet is $\mathcal{Y} = \{-1, +1\}$ and the quantization map is defined by $\mathbf{q}_\mu(-1) = -\mu$ and $\mathbf{q}_\mu(+1) = +\mu$, with $\mu \in \{1, \ldots, Q\}$.

The infinite-precision MS decoder (Algorithm 1), is known to be independent of the scale factor μ. This is because μ factors out from all the processing steps in Algorithm 1, and therefore does not affect in any way the decoding process. This is no longer true for the finite precision decoder (due to saturation effects), and we will show in this section that, even in the noiseless case, the scale factor μ can significantly impact the performance of the finite precision MS decoder.

We start by analyzing the performance of the MS decoder with quantization map $\mathbf{q}_1$, and then we will analyze its performance with an optimized quantization map $\mathbf{q}_\mu$.

Min-Sum decoder with quantization map $\mathbf{q}_1$

The case $\mu = 1$ leads to an "unconventional" behavior, as in some particular cases the noise introduced by the device can help the MS decoder to escape from fixed points attractors, and may actually result in an increased correction capacity with respect to the noiseless decoder. This behavior will be discussed in more details in this section.

We start with the noiseless decoder case. Figure 1.1 shows the asymptotic error probability $P_e^{(\infty)}$ as a function of p_0. It can be seen that $P_e^{(\infty)}$ decreases slightly with p_0, until p_0 reaches a threshold value $p_{\text{th}} = 0.039$, where $P_e^{(\infty)}$ drops to zero. This is the *classical* threshold phenomenon mentioned in Section 1.5.3: for $p_0 > p_{\text{th}}$, the decoding error probability is bounded far above zero ($P_e^{(\infty)} > 0.31$), while for $p_0 < p_{\text{th}}$, one has $P_e^{(\infty)} = 0$.

Now, we consider a p_0 value slightly greater than the threshold of the noiseless decoder, and investigate the effect of the noisy adder on the decoder performance. Let us fix $p_0 = 0.06$. Figure 1.2(a) shows the decoding error probability at iteration ℓ, for different parameters $p_a \in \{10^{-30}, 10^{-15}, 10^{-5}\}$ of the noisy adder. For each p_a value, there are two superimposed curves, corresponding to the full-depth ("fd", solid curve) and sign-preserving ("sp", dashed curve) error models of the noisy adder.

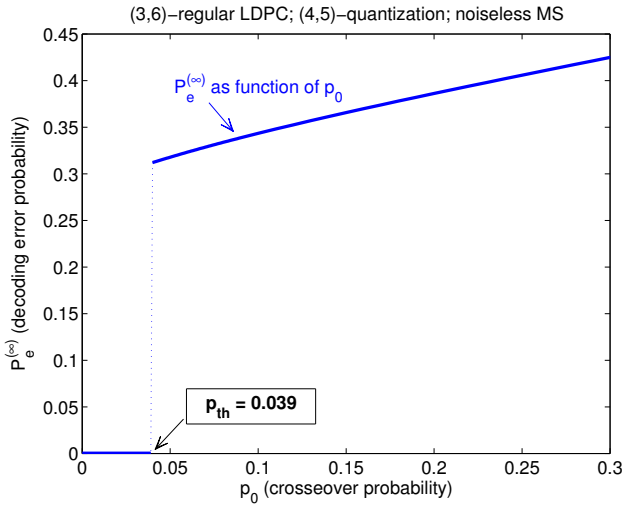
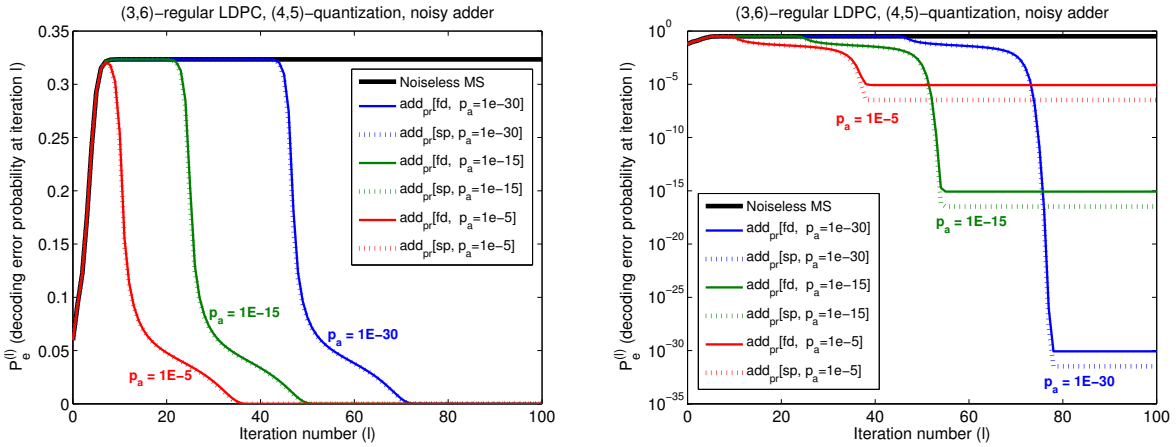


Figure 1.1: Asymptotic error probability $P_e^{(\infty)}$ of the noiseless MS decoder as a function of p_0



(a) $P_e^{(\ell)}$ plotted in linear scale

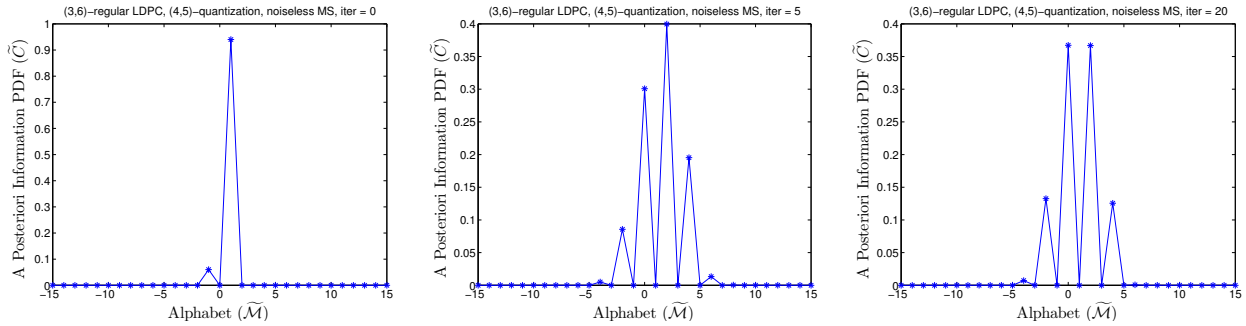
(b) $P_e^{(\ell)}$ plotted in logarithmic scale

Figure 1.2: Effect of the noisy adder on the asymptotic performance of the MS decoder ($p_0 = 0.06$)

Table 1.2: Asymptotic error probability of the MS decoding with noisy adder ($p_0 = 0.06$)

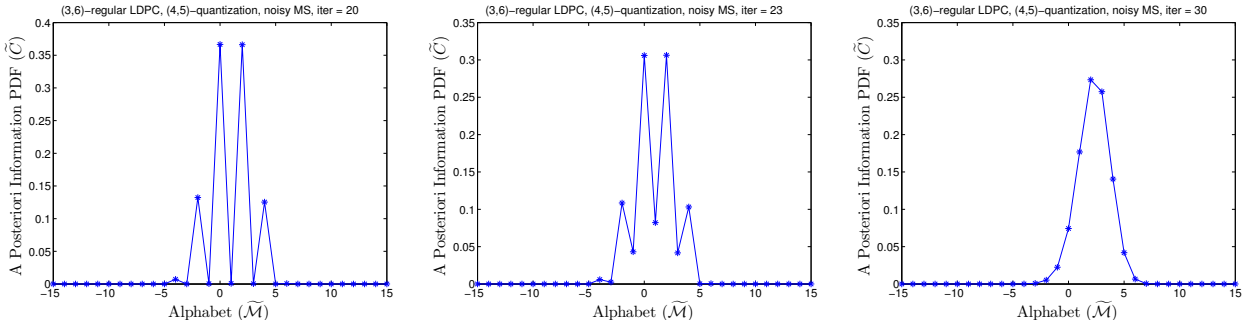
	p_a	10^{-30}	10^{-15}	10^{-5}
full depth	$P_e^{(\infty)}$	8.500×10^{-31}	8.500×10^{-16}	8.507×10^{-6}
	lower-bound	5.167×10^{-31}	5.167×10^{-16}	5.167×10^{-6}
sign protected	$P_e^{(\infty)}$	3.333×10^{-32}	3.333×10^{-17}	3.333×10^{-7}
	lower-bound	3.333×10^{-32}	3.333×10^{-17}	3.333×10^{-7}

The error probability of the noiseless decoder is also plotted (solid black curve): it can be seen that it increases rapidly from the initial value $P_e^{(0)} = p_0$ and closely approaches the limit value $P_e^{(\infty)} = 0.323$ after a few number of iterations. When the adder is noisy, the error probability increases during the first decoding iterations, and behaves similarly as in the noiseless case. It may approach the limit value from the noiseless case, but starts decreasing after some number of decoding iterations. However, it remains bounded above zero, according to the lower bounds from Proposition 1. This can be seen in Figure 1.2(b), where $P_e^{(\ell)}$ plotted in logarithmic scale. The asymptotic values $P_e^{(\infty)}$ and the corresponding lower-bounds values from Proposition 1 are shown in Table 1.2. It can be seen that these bounds are tight, especially in the sign-preserving case.



(a) Iteration $\ell = 0$ (b) Iteration $\ell = 5$ (c) Iteration $\ell = 20$

Figure 1.3: Probability mass function of the a posteriori information $\widetilde{C}^{(\ell)}$ (noiseless MS decoder)



(a) Iteration $\ell = 20$ (b) Iteration $\ell = 23$ (c) Iteration $\ell = 30$

Figure 1.4: Probability mass function of the a posteriori information $\widetilde{C}^{(\ell)}$ (MS decoder with full-depth noisy adder, $p_a = 10^{-15}$)

The above behavior of the MS decoder is explained by the fact that the noise present in the adder helps the MS decoder to escape from fixed points attractors. Figure 1.3 illustrates the evolution of the probability mass function $\widetilde{C}^{(\ell)}$ for the noiseless decoder. At iteration $\ell = 0$, $\widetilde{C}^{(0)}$ is supported in ± 1, with $\widetilde{C}^{(0)}(-1) = p_0$ and $\widetilde{C}^{(0)}(+1) = 1 - p_0$. It evolves during the iterative decoding, and reaches a fixed point of the density evolution for $\ell = 20$. Note that since all variable-nodes are of degree $d_v = 3$, it can be easily seen that, for $\ell \geq 1$, $\widetilde{C}^{(\ell)}$ is supported only on even values. These "gaps" in the probability mass function seem lead to favorable conditions for the occurrence of density-evolution fixed-points.

Figure 1.4 illustrates the evolution of the probability mass function $\widetilde{C}^{(\ell)}$ when the full-depth noisy adder with $p_a = 10^{-15}$ is used within the MS decoder. At iteration $\ell = 20$, $\widetilde{C}^{(\ell)}$ is virtually the same as in the noiseless case. However, the noisy adder allows the decoder to escape from this fixed-point, as it can be seen for iterations $\ell = 23$ and $\ell = 30$. For $\ell > 30$, the $\widetilde{C}^{(\ell)}$ moves further on the right, until the corresponding error probability $P_e^{(\ell)}$ reaches the limit value $P_e^{(\infty)} = 8.5 \times 10^{-16}$.

It is worth noting that neither the noisy comparator nor the XOR-operator can help the decoder to escape from fixed-point distributions, as they do not allow "filling the gaps" in the support of $\widetilde{C}^{(\ell)}$.

We focus now on the useful region of the noisy MS decoder. We assume that only the adder is noisy, while the comparator and the XOR-operator are noiseless.

The useful region for the sign-protected noisy adder model is shown in Figure 1.5. The useful region is shaded in gray and delimited by either a solid black curve or a dashed red curve. Although one would expect that $P_e^{(\infty)} = p_0$ on the border of the useful region, this equality only holds on the solid black border. On the dashed red border, one has $P_e^{(\infty)} < p_0$. The reason why the useful region does not extend beyond the dashed red border is that for points located on the other side of this border the sequence $(P_e^{(\ell)})_{\ell > 0}$ is periodic, and hence it does not converge! The region shaded in brown

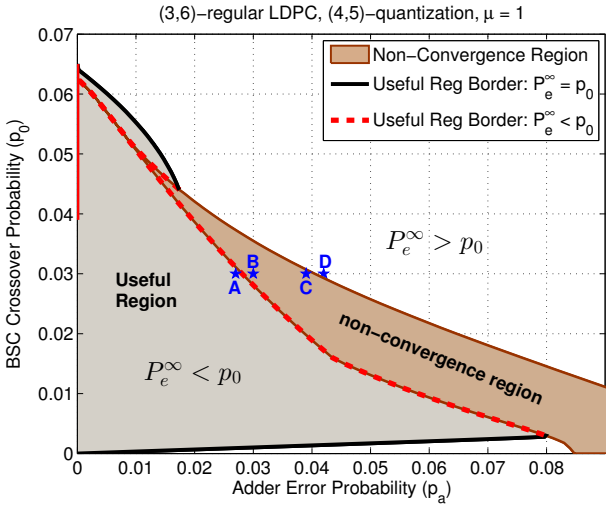
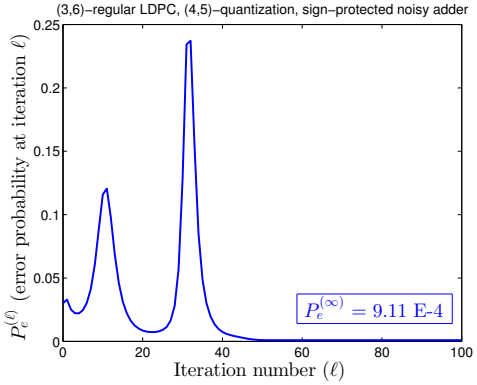
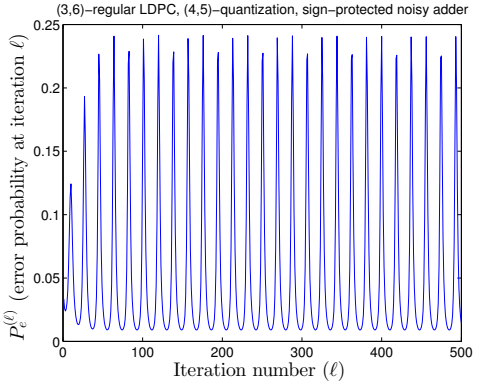


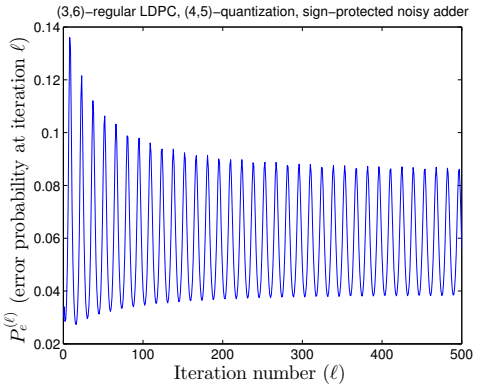
Figure 1.5: Useful and non-convergence regions of the MS decoder with sign-preserving noisy adder



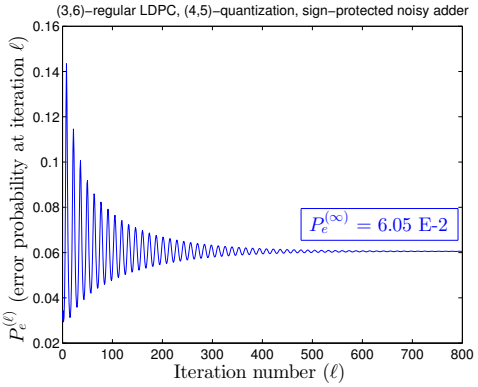
(a) Point $A(p_0 = 0.03, p_a = 0.027)$



(b) Point $B(p_0 = 0.03, p_a = 0.03)$



(c) Point $C(p_0 = 0.03, p_a = 0.039)$



(d) Point $D(p_0 = 0.03, p_a = 0.042)$

Figure 1.6: Decoding error probability $P_e^{(\ell)}$ of the noisy MS decoder, for $p_0 = 0.03$ and sign-preserving noisy adder with various p_a values

in Figure 1.5 is the *non-convergence region* of the decoder. Note that the non-convergence region gradually narrows in the upper part, and there is a small portion of the useful region delimited by the non-convergence region on the left and the black border on the right. Finally, we note that points with $p_a = 0$ (noiseless decoder) and $p_0 > 0.039$ (threshold of the noiseless decoder) – represented by the solid red line superimposed on the vertical axis in Figure 1.5 – are excluded from the useful region. Indeed, for such points $P_e^{(\infty)} > p_0$; however, for p_a greater than but close to zero, we have $P_e^{(\infty)} \approx \frac{p_a}{2Q}$ (see Figure 1.2 and related discussion).

We exemplify the decoder behavior on four points located on one side and the other of the left and right boundaries of the non-convergence region. These points are indicated in Figure 1.5 by A, B, C, and D. For all the four points $p_0 = 0.03$, while $p_a = 0.027, 0.03, 0.039$, and 0.042, respectively. The error probability $(P_e^{(\ell)})_{\ell>0}$ is plotted for each one of these points in Figure 1.6. The point A belongs to the useful region, and it can be seen from Figure 1.6(a) that $(P_e^{(\ell)})_{\ell>0}$ converges to $P_e^{(\infty)} = 9.11\times10^{-4} < p_0$. For the point B, located just on the other side of the dashed red border of the useful region, $(P_e^{(\ell)})_{\ell>0}$ exhibits a periodic behavior (although we only plotted the first 500 iterations, we verified the periodic behavior on the first 5×10^4 iterations). Crossing the non-convergence region from left to the right, the amplitude between the inferior and superior limits of $(P_e^{(\ell)})_{\ell>0}$ decreases (point C), until it reaches again a convergent behavior (point D). Note that D is outside the useful region, as $(P_e^{(\ell)})_{\ell>0}$ converges to $P_e^{(\infty)} = 0.0605 > p_0$.

The non-convergence region gradually narrows in the upper part, and for $0 \leq p_a < 0.01$ it takes the form of a *discontinuity line*: $P_e^{(\infty)}$ takes values close to 10^{-4} just below this line, and values greater than 0.05 above this line.

Note that points (p_a, p_0) with $p_0 < \frac{p_a}{2\widetilde{Q}} = \frac{p_a}{30}$ cannot belong to the useful region, since from Proposition 1 we have $P_e^{(\infty)} \geq \frac{p_a}{2\widetilde{Q}} > p_0$. Moreover, we note that the bottom border of the useful region (solid black curve) is virtually identical to, but slightly above, the line defined by $p_0 = \frac{p_a}{2\widetilde{Q}}$.

Optimization of the quantization map

In this section we show that the decoder performance can be significantly improved by using an appropriate choice of the channel scale factor μ. Figure 1.7 shows the threshold values for the noiseless and several noisy decoders with channel scale factors $\mu \in \{1, 2, \ldots, 7\}$. For the noisy decoders, the threshold values are computed for a target error probability $\eta = 10^{-5}$ (see Equation (1.19)).

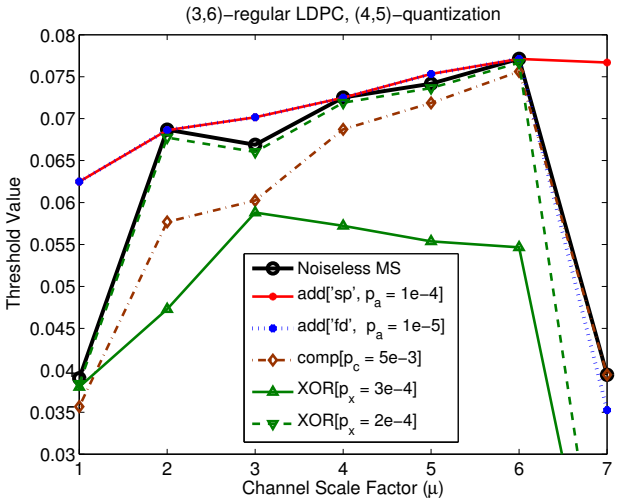


Figure 1.7: Threshold values of noiseless and noisy MS decoders with various channel scale factors (for noisy decoders, threshold values correspond to a target error probability $\eta = 10^{-5}$)

The solid black curve in Figure 1.7 correspond to the noiseless decoder. The solid red curve and the dotted blue curve correspond to the MS decoder with sign-protected noisy adder and full-depth noisy adder, respectively. The adder error probability is $p_a = 10^{-4}$ for the sign-protected noisy adder, and $p_a = 10^{-5}$ for the full-depth adder[8]. The two curves are superimposed for $1 \leq \mu \leq 6$, and differ only for $\mu = 7$. The corresponding threshold values are equal to those obtained in the noiseless case for $\mu \in \{2, 4, 6\}$. For $\mu \in \{1, 3, 5\}$, the MS decoders with noisy-adders exhibit better thresholds than

[8]Note that according to Proposition 1, a necessary condition to achieve a target error probability $P_e^{(\infty)} \leq \eta = 10^{-5}$ is $p_a \leq 2\widetilde{Q}\eta = 3 \times 10^{-4}$ for the signed-protected adder, and $p_a \leq 2\eta\frac{2\widetilde{Q}+1}{2\widetilde{Q}} = 2.07 \times 10^{-5}$ for the full-depth adder.

the noiseless decoder. This is due to the fact that the messages alphabet $\mathcal{M}$ is underused by the noiseless decoder, since all the exchanged messages are necessarily odd (recall that all variable-nodes are of degree $d_v = 3$). For the MS decoders with noisy adders, the noise present in the adders leads to a more efficient use of the messages alphabet, which allows the decoder to escape from fixed-point attractors and hence results in better thresholds (Section 1.6.1).

Figure 1.7 also shows a curve corresponding to the MS decoder with a noisy comparator having $p_c = 0.005$, and two curves for the MS decoder with noisy XOR-operators, having respectively $p_x = 2 \times 10^{-4}$ and $p_x = 3 \times 10^{-4}$.

Concerning the noisy XOR-operator, it can be seen that the threshold values corresponding to $p_x = 2 \times 10^{-4}$ are very close to those obtained in the noiseless case, except for $\mu = 7$ (the same holds for values $p_x < 2 \times 10^{-4}$). However, a significant degradation of the threshold can be observed when slightly increasing the XOR error probability to $p_x = 3 \times 10^{-4}$. Moreover, although not shown in the figure, it is worth mentioning that for $p_x \geq 5 \times 10^{-4}$, the target error probability $\eta = 10^{-5}$ can no longer be reached (thus, all threshold values are equal to zero).

Finally, we note that except for the noisy XOR-operator with $p_x = 3 \times 10^{-4}$, the best choice of the channel scale factor is $\mu = 6$. For the noisy XOR-operator with $p_x = 3 \times 10^{-4}$, the best choice of the channel scale factor is $\mu = 3$. This is rather surprising, as in this case the messages alphabet is underused by the decoder: all the exchanged messages are odd, and the fact that the XOR-operator is noisy does not change their parity.

Assumption: In the following sections, we will investigate the impact of the noisy adder, comparator and XOR-operator on the MS decoder performance, assuming that the channel scale factor is $\mu = 6$.

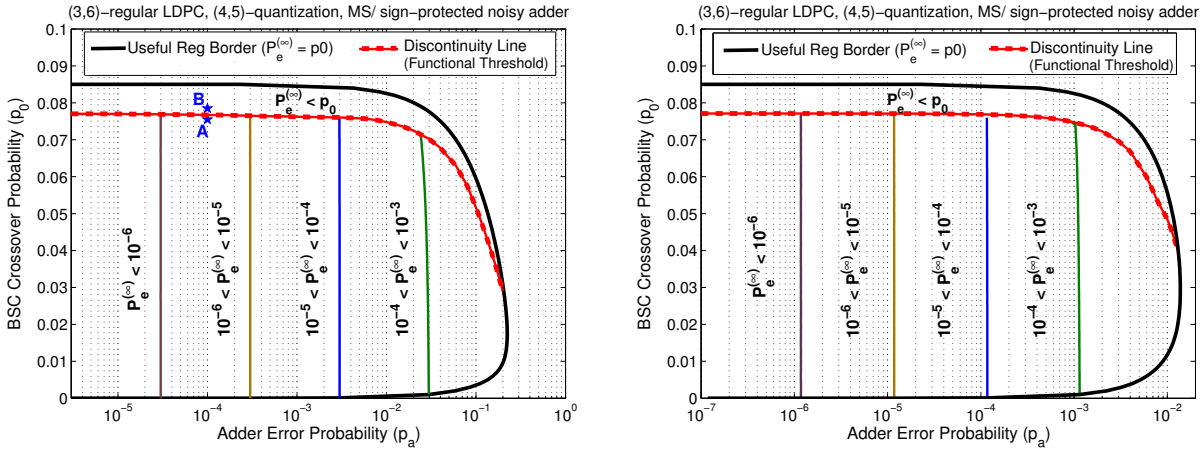
Study of the impact of the noisy adder (quantization map $\mathbf{q}_6$)

In order to evaluate the impact of the noisy adder on the MS decoder performance, the useful region and the η-threshold regions have been computed, assuming that only the adders within the VN-processing step are noisy ($p_a > 0$), while the CN-processing step is noiseless ($p_x = p_c = 0$). This regions are represented in Figure 1.8, for both sign-protected and full-depth noisy adder models.

The useful region is delimited by the solid black curve. The vertical lines delimit the η-threshold regions, for $\eta = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$ (from right to the left).

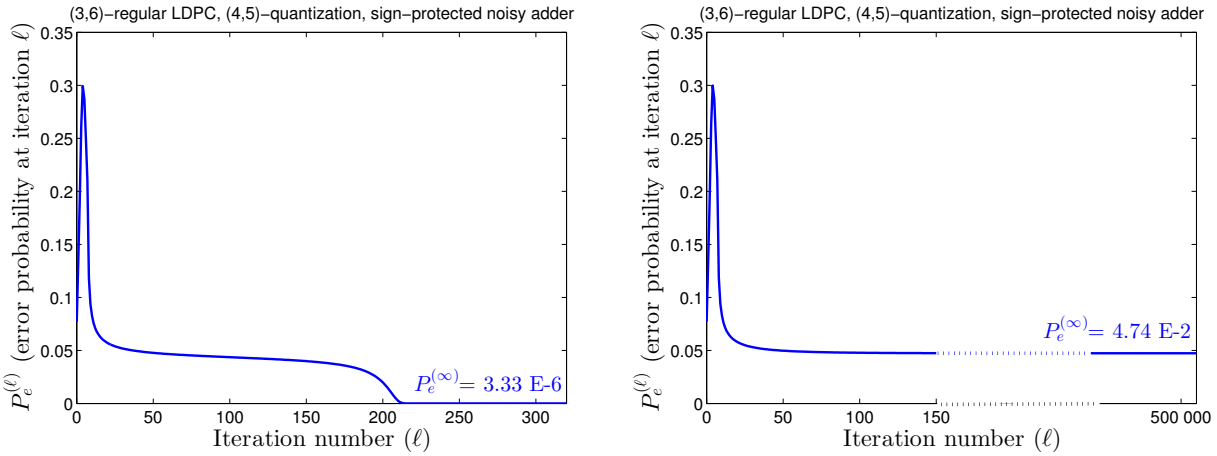
Note that unlike the case $\mu = 1$ (Section 1.6.1), there is no non-convergence region when the channel scale factor is set to $\mu = 6$. Hence, the border of the useful region corresponds to points (p_a, p_0) for which $P_e^{(\infty)} = p_0$. However, it can be observed that there is still a *discontinuity line* (dashed red curve) inside the useful region. This discontinuity line does not hide a periodic (non-convergent) behavior, but it is due to the occurrence of an *early plateau phenomenon* in the convergence of $(P_e^{(\ell)})_\ell$. This phenomenon is illustrated in Figure 1.9, where the error probability $(P_e^{(\ell)})_\ell$ is plotted as a function of the iteration number ℓ, for the two points A and B from Figure 1.8(a). For point A, it can be observed that the error probability $P_e^{(\ell)}$ reaches a first plateau for $\ell \approx 50$, then drops to 3.33×10^{-6} for $\ell \geq 250$. For point B, $P_e^{(\ell)}$ behaves in a similar manner during the first iterations, but it does not decrease below the plateau value as ℓ goes to infinity. Although we have no analytic proof of this fact, it was numerically verified for $\ell \leq 5 \times 10^5$.

In Figure 1.10, we plotted the asymptotic error probability $P_e^{(\infty)}$ as a function of p_0, for the noiseless decoder ($p_a = 0$), and for the sign-protected noisy adder with error probability values $p_a = 10^{-4}$ and $p_a = 0.05$. In each plot we have also represented two points $p_0^{(\mathrm{U})}$ and $p_0^{(\mathrm{DL})}$, corresponding respectively to the values of p_0 on the upper-border of the useful region, and on the discontinuity line. Hence, $p_0^{(\mathrm{DL})}$ coincides with the classical threshold of the MS decoder in the noiseless case, and it can be seen as an appropriate generalization of the classical threshold to the case of noisy decoders. In the following, $p_0^{(\mathrm{DL})}$ will be referred to as the ***functional threshold*** of the noisy decoder, and the sub-region of the useful region located below the discontinuity line will be referred to as the ***functional region***. Within this region, if the adder error probability is small enough, it can be observed that:



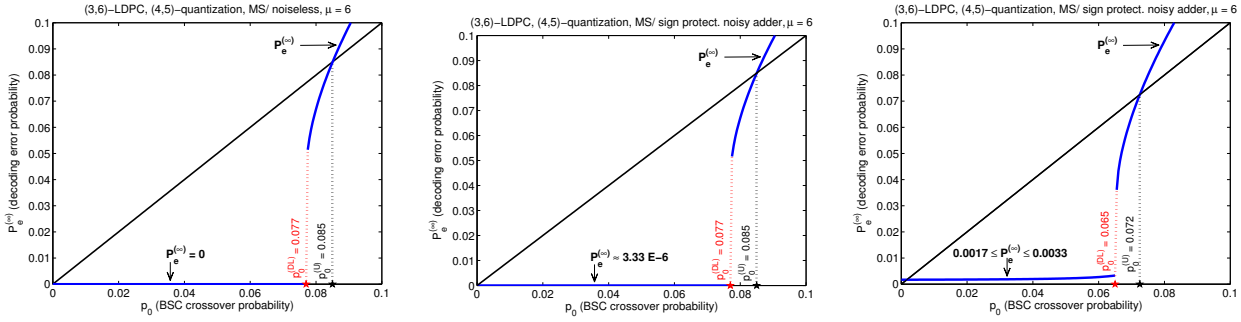
(a) Sign-protected noisy adder (b) Full-depth noisy adder

Figure 1.8: Useful and η-threshold regions of the MS decoder with noisy adder



(a) Point $A(p_0 = 0.0770, p_a = 10^{-4})$ (b) Point $B(p_0 = 0.0772, p_a = 10^{-4})$

Figure 1.9: Illustration of the early plateau phenomenon (points A and B from Figure 1.8(a))



(a) $p_a = 0$ (noiseless decoder) (b) $p_a = 10^{-4}$ (c) $p_a = 0.05$

Figure 1.10: Asymptotic error probability $P_e^{(\infty)}$ as a function of p_0; noiseless and noisy MS decoder with sign-protected noisy adder

(a) For the sign-protected adder: $P_e^{(\infty)} \approx \frac{p_a}{30}$, for $p_a \lessapprox 3 \times 10^{-2}$, which corresponds to the value given by the lower-bound ($\frac{1}{2\widetilde{Q}} p_a = \frac{1}{30} p_a$) from Proposition 1.

(*b*) For the full-depth adder: $P_e^{(\infty)} \approx 1.17 p_a$, for $p_a \lessapprox 10^{-3}$, which is about twice higher than the value given by the lower-bound ($\frac{1}{2}p_a + \frac{1}{4\widetilde{Q}}p_a = 0.52 p_a$) from Proposition 1.

Finally, we note that by protecting the sign of the noisy adder, the useful region is expanded by a factor of roughly $2\widetilde{Q}$, representing an exponential improvement with respect to the number of bits of the adder (see also the discussion following the proof of Proposition 1).

Study of the impact of the noisy XOR-operator (quantization map $\mathbf{q}_6$)

The useful region and the η-threshold regions of the decoder, assuming that only the XOR-operator used within the CN-processing step is noisy, are plotted in Fig. 1.11. Similar to the noisy-adder case, a discontinuity line can be observed inside the useful region, which delimits the *functional region* of the decoder.

Comparing the η-threshold regions from Figure 1.8 and Figure 1.11, it can be observed that in order to achieve a target error probability $P_e^{(\infty)} \leq 10^{-6}$, the error probability parameters of the noisy adder and of the noisy XOR-operator must satisfy:

- $p_a < 1.17 \times 10^{-6}$, for the full-depth noisy-adder;
- $p_a < 3 \times 10^{-5}$, for the sign-protected noisy-adder;
- $p_x < 7 \times 10^{-5}$, for the noisy XOR-operator.
  (moreover, values of p_x up to 1.4×10^{-4} are tolerable if p_0 is sufficiently small)

The most stringent requirement concerns the error probability of the full-depth noisy-adder, thus we may consider that it has the most negative impact on the decoder performance. On the other hand, the less stringent requirement concerns the error probability of the noisy XOR-operator.

Finally, it is worth noting that in practical cases the value of p_x should be significantly lower than the value of p_a (given the high number of elementary gates contained in the adder). Moreover, since the XOR-operators used to compute the signs of CN messages represent only a small part of the decoder, this part of the circuit could be made reliable by using classical fault-tolerant methods, with a limited impact on the overall decoder design.

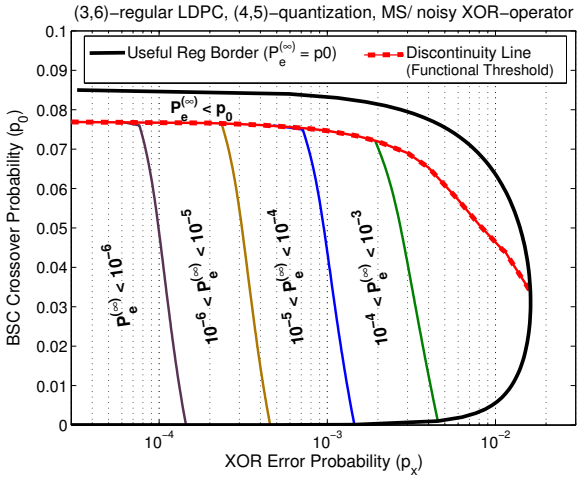


Figure 1.11: Useful and η-threshold regions of the MS decoder with noisy XOR-operator

Study of the impact of the noisy comparator (quantization map $\mathbf{q}_6$)

This section investigates the case when comparators used within the CN-processing step are noisy ($p_c > 0$), but $p_a = p_x = 0$. Contrary to the previous cases, this case exhibits a "classical" threshold phenomenon, similar to the noiseless case: for a given $p_c > 0$, there exists a p_0-threshold value, denoted by $p_0^{(\mathrm{TH})}$, such that $P_e^{(\infty)} = 0$ for any $p_0 < p_0^{(\mathrm{TH})}$.

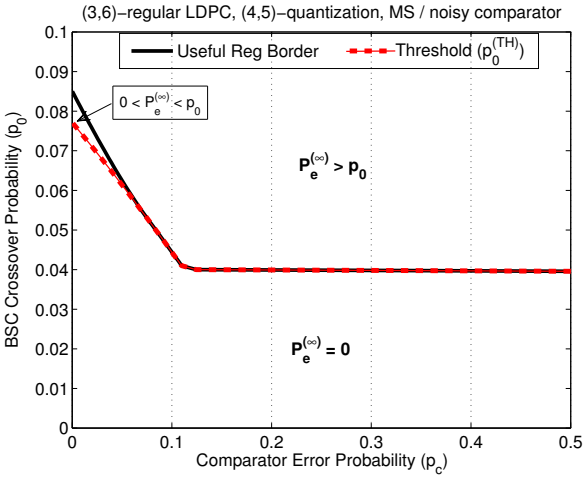


Figure 1.12: Useful region and threshold curve of the MS decoder with noisy comparator

The threshold value $p_0^{(\mathrm{TH})}$ is plotted as a function of p_c in Figure 1.12. The functional region of the decoder is located below the threshold curve, and $P_e^{(\infty)} = 0$ for any point within this region. In particular, it can be seen that $P_e^{(\infty)} = 0$ for any $p_0 \lessapprox 0.039$ and any $p_c > 0$. Although such a threshold phenomenon might seem surprising for a noisy decoder, it can be easily explained. The idea behind is that in this case the crossover probability of the channel is small enough, so that in the CN-processing step only the sign of check-to-variable messages is important, but not their amplitudes. In other words a decoder that only computes (reliably) the signs of check-node messages and randomly chooses their amplitudes, would be able to perfectly decode the received word.

Finally, we note that the useful region of the decoder extends slightly above the threshold curve: for p_c close to 0, there exists a small region above the threshold curve, within which $0 < P_e^{(\infty)} < p_0$.

1.6.2 Numerical results for the BI-AWGN channel

For the BI-AWGN, the channel output is given by $y = x + z$, where $x \in \{\pm 1\}$ is the channel input and z is the additive white Gaussian noise with variance σ^2. Threshold values and useful regions of the decoder will be described in terms of Signal to Noise Ratio (SNR), defined by $\mathrm{SNR} = -10\log_{10}(\sigma^2)$.

For a given channel scale factor μ, the quantization map $\mathbf{q}_\mu$ is defined by $\mathbf{q}_\mu(y) = \mathbf{s}_{\mathcal{M}}([\mu{\cdot}y])$, where $[\mu{\cdot}y]$ denotes the nearest integer to $\mu{\cdot}y$, and $\mathbf{s}_{\mathcal{M}}$ is the saturation map (see also Equation (1.12)).

Similar to the BSC case, the choice of the channel scale factor μ may significantly impact the decoder performance. Hence, we start first by optimizing the channel scale factor value, and then we investigate the impact of the different noisy components on the decoder performance.

Remark: For the BI-AWGN channel we denote by $p_0 \stackrel{\mathrm{def}}{=} P_e^{(0)}$ the error probability at iteration 0, which is, by definition, the probability of the *a priori information* $\gamma = \mathbf{q}_\mu(y)$ being in error. Hence, $p_0 = \sum_{z=-Q}^{-1} C(z) + \frac{1}{2}C(0)$. Using Equation (1.14) it follows that:

$$p_0 = 1 - \frac{1}{2}\left[q\left(\frac{-0.5-\mu}{\mu\sigma}\right) + q\left(\frac{0.5-\mu}{\mu\sigma}\right)\right] \tag{1.20}$$

Optimization of the quantization map

The goal of this section is to provide an optimal choice of the channel scale factor μ. Figure 1.13 shows the threshold SNR values for the noiseless and several noisy decoders for channel scale factors μ varying within the interval $[1, 7]$. For the noisy decoders, the threshold values are computed for a target error probability $\eta = 10^{-5}$ (see Equation (1.19)).

The solid black curve in Figure 1.13 correspond to the noiseless decoder. The dashed red curve and the dotted blue curve correspond to the MS decoder with sign-protected noisy adder and full-depth

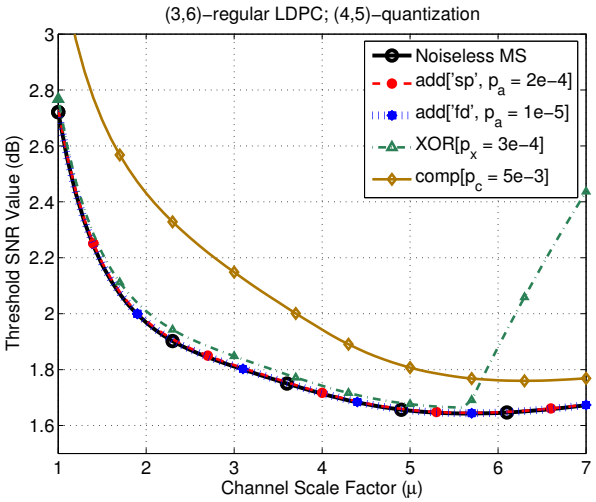


Figure 1.13: Threshold SNR values of noiseless and noisy decoders with various channel scale factors (for noisy decoders, threshold values correspond to a target error probability $\eta = 10^{-5}$)

noisy adder, respectively. The adder error probability is $p_a = 2 \times 10^{-4}$ for the sign-protected noisy adder, and $p_a = 10^{-5}$ for the full-depth adder[9]. These three curves are virtually indistinguishable.

Figure 1.13 also shows two curves corresponding respectively to the MS decoder with a noisy XOR-operator ($p_x = 2 \times 10^{-4}$) and to the MS decoder with a noisy comparator ($p_c = 0.005$). Finally, we note that in all cases the best choice of the channel scale factor is $\mu \approx 5.5$.

Assumption: In the following sections, we will investigate the impact of the noisy adder, comparator and XOR-operator on the MS decoder performance, assuming that the channel scale factor is $\mu = 5.5$.

Study of the impact of the noisy adder

Useful and η-regions of the MS decoder with noisy adders are represented in Figure 1.14, for both sign-protected and full-depth noisy adder models. The useful region is delimited by the solid black curve, while vertical lines delimit the η-threshold regions, for $\eta = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$ (from right to the left). The *functional threshold* of the decoder is also displayed by a red dashed curve.

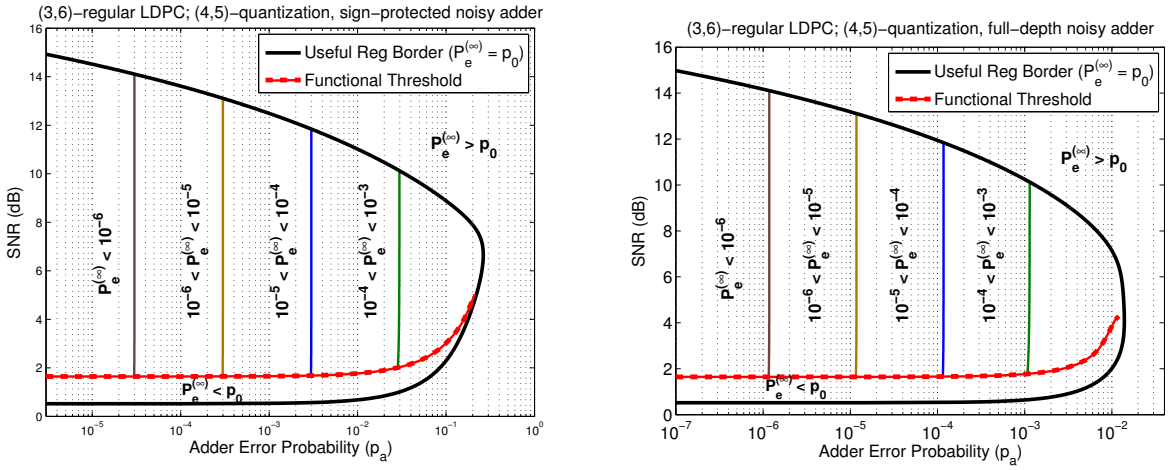
Figure 1.15 shows the input and output error probabilities of the decoder (p_0 and $P_e^{(\infty)}$) as functions of the SNR value, for the sign-protected and full-depth noisy adder models with $p_a = 10^{-4}$. The two intersection points between the two curves correspond to the points on the lower and upper borders of the useful region in Figure 1.14, for $p_a = 10^{-4}$. The discontinuity point of the $P_e^{(\infty)}$ curve corresponds to the functional threshold value in Figure 1.14, for $p_a = 10^{-4}$.

Study of the impact of the noisy XOR-operator and noisy comparator

The useful region and the η-threshold regions of the MS decoder, assuming that only the XOR-operator used within the CN-processing step is noisy, are plotted in Fig. 1.16. The *functional threshold* of the decoder is also displayed by a red dashed curve.

The case of a noisy comparator is illustrated in Figure 1.17. Similar to the BSC channel, this case exhibits a "classical" threshold phenomenon: for any SNR value above the functional threshold curve, one has $P_e^{(\infty)} = 0$.

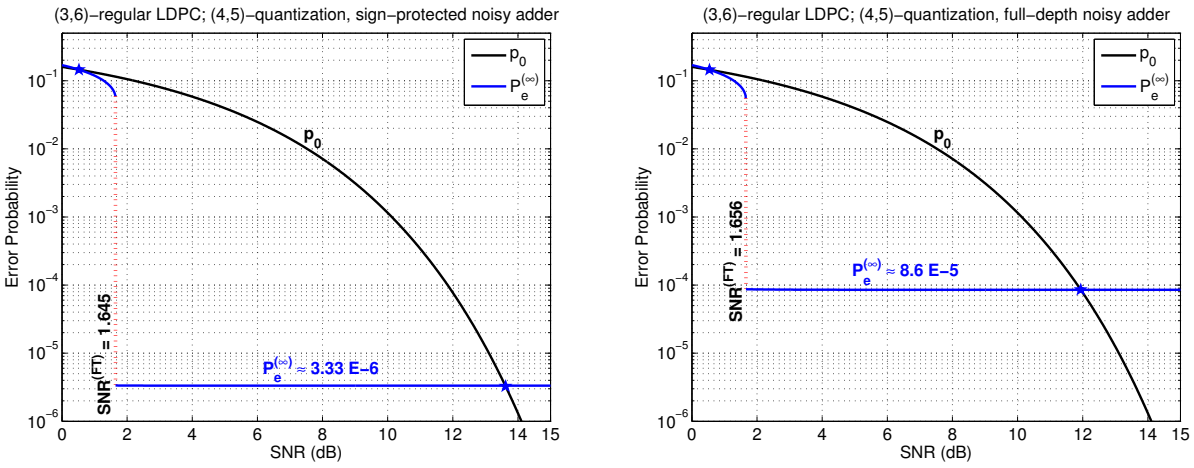
[9]Note that according to Proposition 1, a necessary condition to achieve a target error probability $P_e^{(\infty)} \leq \eta = 10^{-5}$ is $p_a \leq 2\widetilde{Q}\eta = 3 \times 10^{-4}$ for the signed-protected adder, and $p_a \leq 2\eta\frac{2\widetilde{Q}+1}{2\widetilde{Q}} = 2.07 \times 10^{-5}$ for the full-depth adder.



(a) Sign-protected noisy adder

(b) Full-depth noisy adder

Figure 1.14: Useful and η-threshold regions of the MS decoder with noisy adder (BI-AWGN)



(a) sign-protected noisy adder, $p_a = 10^{-4}$

(b) full-depth noisy adder, $p_a = 10^{-4}$

Figure 1.15: Asymptotic error probability $P_e^{(\infty)}$ of the MS decoder with noisy-adder as a function of the SNR

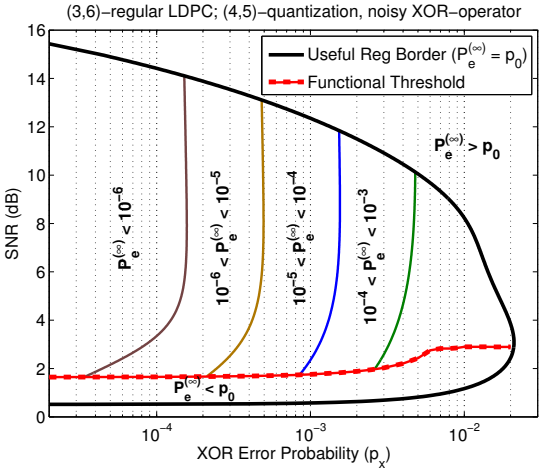


Figure 1.16: Useful and η-threshold regions of the MS decoder with noisy XOR-operator (BI-AWGN)

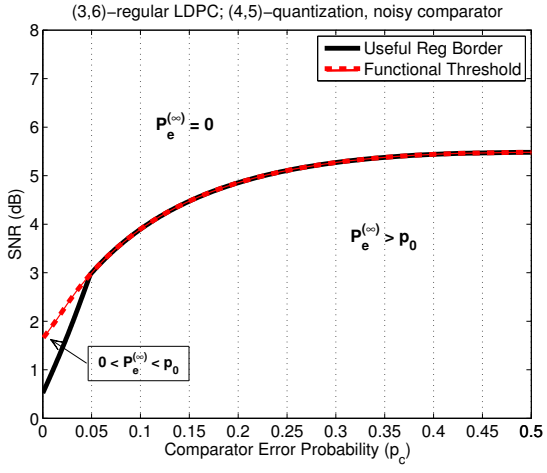


Figure 1.17: Useful region and threshold curve of the MS decoder with noisy comparator (BI-AWGN)

1.7 Finite Length Performance of Min-Sum based decoders

The goal of this section is twofold:

(1) To corroborate the asymptotic analysis through finite-length simulations;

(2) To investigate ways of increasing the robustness of the MS decoder to hardware noise.

Assumption: Unless otherwise stated, the $(3, 6)$-regular LDPC code with length $N = 1008$ bits from [23] will be used for finite length simulations throughout this section.

1.7.1 Practical implementation and early stopping criterion

First of all, we note that the practical implementation of the noisy MS decoder differs slightly from the one presented in Algorithm 2:

- The order of the **VN-processing** and **AP-update** steps is inverted;
- The variable-to-check node messages are computed by subtracting the incoming check-to-variable message from the corresponding a posteriori information value:

for all $n = 1, \ldots, N$ **do** ▷ **AP-update**
$\tilde{\gamma}_n = \mathbf{a}_{\text{pr}}\left(\{\gamma_n\} \cup \{\beta_{m,n}\}_{m \in \mathcal{H}(n)}\right)$;

for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**
$\alpha_{m,n} = \mathbf{a}_{\text{pr}}\left(\tilde{\gamma}_n, -\beta_{m,n}\right)$;
$\alpha_{m,n} = \mathbf{s}_{\mathcal{M}}\left(\alpha_{m,n}\right)$;

For floating-point noiseless decoders, the two ways of computing the variable-to-check messages are completely equivalent. However, this equivalence does not hold anymore for finite-precision (noisy or noiseless) decoders, because of saturation effects and, in case of noisy decoders, of probabilistic computations. We note that the practical implementation might result in a degradation of the decoder performance compared to the "Density-Evolution like" implementation (Algorithm 2), since each variable-to-check node message *encompasses* $d_v + 1$ additions (d_v additions to compute $\tilde{\gamma}_n$ and one subtraction).

Finally, it is worth noting that the density-evolution analysis cannot be applied to the practical implementation, due to the fact that in the VN-processing step, the computation of variable-to-check messages $\alpha_{m,n} = \mathbf{a}_{\text{pr}}\left(\{\gamma_n\}, -\beta_{m,n}\right)$ involves two correlated variables, namely γ_n and $\beta_{m,n}$.

Early stopping criterion (syndrome check)

As described in Algorithm 2, each decoding iteration also comprises a *hard decision* step, in which each transmitted bit is estimated according to the sign of the a posteriori information, and a *syndrome check* step, in which the syndrome of the estimated word is computed.

Both steps are assumed to be *noiseless*, and the syndrome check step acts as an *early stopping criterion*: the decoder stops when whether the syndrome is $+1$ (the estimated word is a codeword) or a maximum number of iterations is reached. We note however that the syndrome check step is optional and, if missing, the decoder stops when the maximum number of iterations is reached.

Remark: The reason why we stress the difference between the MS decoder with and without the syndrome check step is because, as we will see shortly, the *noiseless* early stopping criterion may significantly improve the bit error rate performance of the *noisy* decoder in the error floor region.

Assumptions:

- Unless otherwise stated, the MS decoder is assumed to implement the *noiseless* stopping criterion (syndrome check step).
- The maximum number of decoding iterations is fixed to 100 throughout this section.

1.7.2 Corroboration of the asymptotic analysis through finite-length simulations

We start by analyzing the finite-length decoder performance over the BSC channel. Figure 1.18 shows the bit error rate (BER) performance of the finite-precision MS decoder (both noiseless and noisy) with various channel scale factors. For comparison purposes, we also included the BER performance of the Belief-Propagation decoder (solid black curve, no markers) and of the infinite-precision MS decoder (dashed blue curve, no markers).

It can be observed that the worst performance is achieved by the infinite-precision MS decoder (!) and the finite-precision noiseless MS decoder with channel scale factor $\mu = 1$ (both curves are virtually indistinguishable). The BER performance of the latter improves significantly when using a sign-preserving noisy adder with error probability $p_a = 0.001$ (dashed red curve with empty circles).

For a channel scale factor $\mu = 6$, both noiseless and noisy decoders have almost the same performance (solid and dashed green curves, with triangular markers). Remarkably, the achieved BER is very close to the one achieved by the Belief-Propagation decoder!

These results corroborate the asymptotic analysis from Section 1.6.1 concerning the channel scale factor optimization.

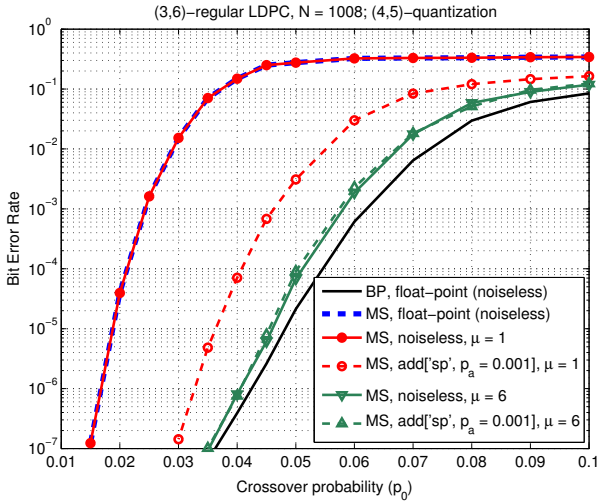


Figure 1.18: BER performance of noiseless and noisy MS decoders with various channel scale factors

Error floor performance

Surprisingly, the BER curves of the noisy decoders from Figure 1.18 do not show any error floor down to 10^{-7}. However, according to Proposition 1, the decoding error probability should be lower-bounded by $P_e^{(\ell)} \geq \frac{1}{2\widetilde{Q}} p_a = 3.33 \times 10^{-5}$ (see also the η-threshold regions in Figure 1.8(a)).

The fact that the observed decoding error probability may decrease below the above lower-bound is due to the early stopping criterion (syndrome check step) implemented within the MS decoder. Indeed, as we observed in the previous section, the above lower-bound is tight, when ℓ (the iteration number) is sufficiently large. Therefore, as the iteration number increases, the expected number of erroneous bits gets closer and closer to $\frac{1}{2\widetilde{Q}} p_a N = 0.034$, and the probability of not having any erroneous bit within one iteration approaches $\left(1 - \frac{1}{2\widetilde{Q}} p_a\right)^N = 0.967$. As the decoder performs more and more iterations, it will eventually reach an error free iteration. The absence of errors is at once detected by the noiseless syndrome check step, and the decoder stops.

To illustrate this behavior, we plotted the Figure 1.19 the BER performance of the noisy MS decoder, with and without early stopping criterion. The noisy MS decoder comprises a sign-preserving noisy adder with $p_a = 0.001$, while the comparator and the XOR-operator are assumed to be noiseless ($p_c = p_x = 0$). Two codes are simulated, the first with length $N = 1008$ bits, and the second with

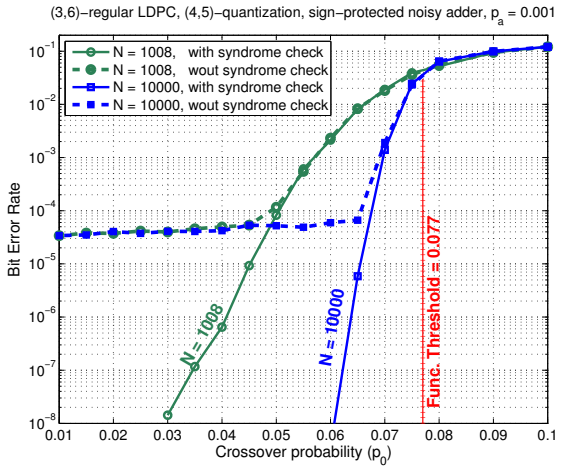


Figure 1.19: BER performance with and without early stopping criterion (MS decoder with sign-preserving noisy adder, $p_a = 0.001$)

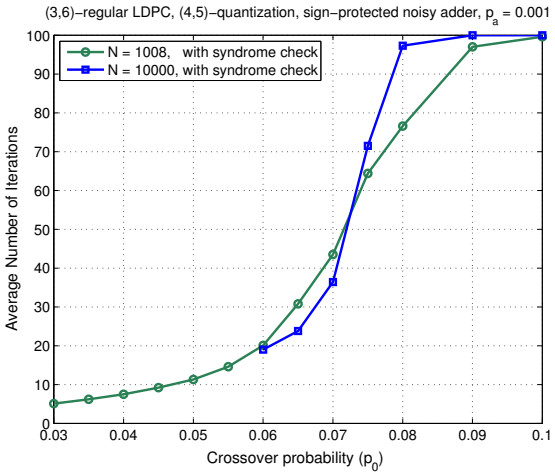


Figure 1.20: Average number of decoding iterations with early stopping criterion (MS decoder with sign-preserving noisy adder, $p_a = 0.001$)

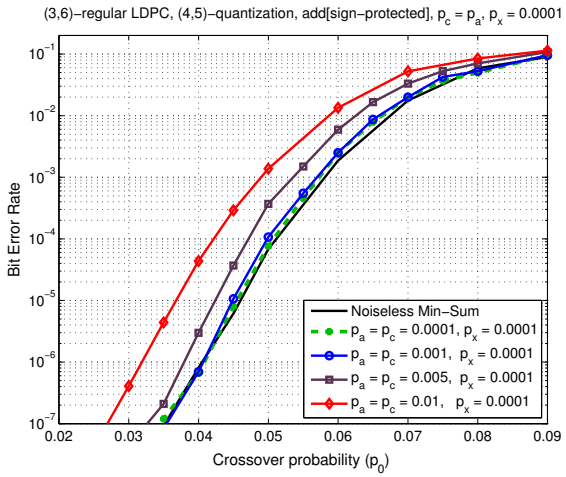
length $N = 10000$ bits. In case that the noiseless early stopping criterion is implemented (solid curves), it can be seen that none of the BER curves show any error floor down to 10^{-8}. However, if the early stopping criterion is not implemented (dashed curves), corresponding BER curves exhibit an error floor at $\approx 3.33 \times 10^{-5}$, as predicted by Proposition 1.

In Figure 1.20 we plotted the average number of decoding iterations in case that the early stopping criterion is implemented. It can be seen that the average number of decoding iterations decreases with the channel crossover probability p_0, or equivalently, with the achieved bit error rate. However, for a fixed BER – say BER $= 10^{-6}$, achieved either at $p_0 \approx 0.04$ for the code with $N = 1008$, or at $p_0 \approx 0.063$ for the code with $N = 10000$ – the average number of iterations is about 8 for the first code and about 21 for the second. Note that in case the early stopping criterion is not implemented, both codes have nearly the same performance for the above p_0 values. Thus, when the early stopping criterion is implemented, the decoder needs to perform more iterations to eventually reach an error free iteration when $N = 10000$, which explains the increased average number of decoding iterations.

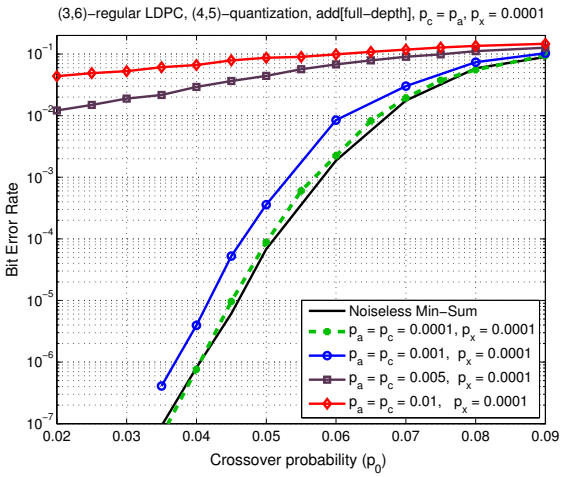
Further results on the finite-length performance

In this section we investigate the finite-length performance when all the MS components (adder, comparator, and XOR-operator) are noisy. In order to reduce the number of simulations, we assume that $p_a = p_c \geq p_x$. Concerning the noisy adder, we evaluate the BER performance for both the sign-preserving and the full-depth error models. Simulation results are presented in Figure 1.21. The error probability of the XOR-operator is $p_x = 0.0001$ in sub-figures 1.21(a) and 1.21(b), and $p_x = 0.001$ in sub-figures 1.21(c) and 1.21(d). The noisy adder is sign-preserving in sub-figures 1.21(a) and 1.21(c), and full-depth in sub-figures 1.21(b) and 1.21(d).

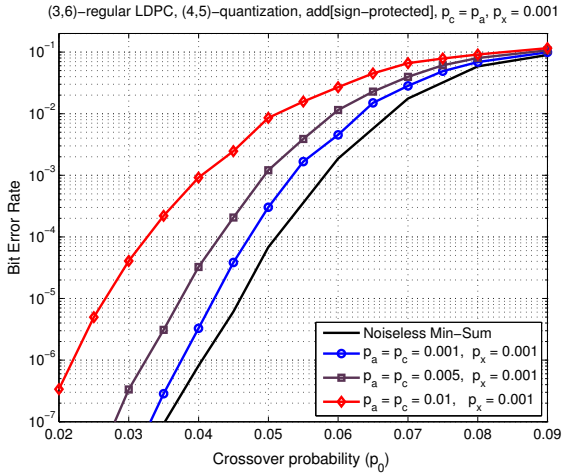
In case the noisy-adder is sign-preserving, it can be seen that the MS decoder can provide reliable error protection for all the noise parameters that have been simulated. Of course, depending on the error probability parameters of the noisy components, there is a more or less important degradation of the achieved BER with respect to the noiseless case. But in all cases the noisy decoder can achieve a BER less than 10^{-7}. This is no longer true for the full-depth noisy adder: it can be seen that for $p_c = p_a \geq 0.005$, the noisy decoder cannot achieve bit error rates below 10^{-2}.



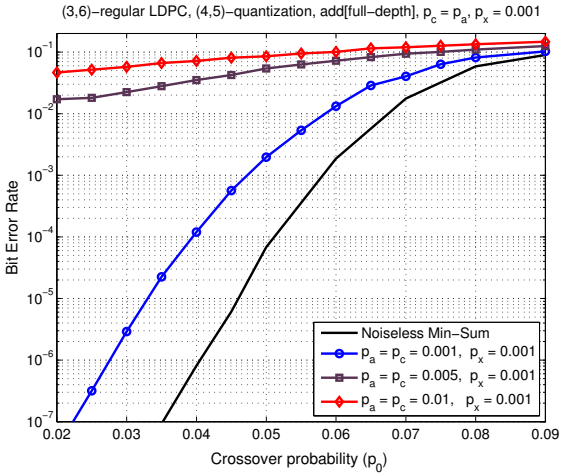
(a) sign-preserving noisy adder, $p_c = p_a$, $p_x = 0.0001$



(b) full-depth noisy adder, $p_c = p_a$, $p_x = 0.0001$



(c) sign-preserving noisy adder, $p_c = p_a$, $p_x = 0.001$



(d) full-depth noisy adder, $p_c = p_a$, $p_x = 0.001$

Figure 1.21: BER performance of the noisy MS decoder with various noise parameters

1.7.3 Noisy Self-Corrected Min-Sum decoder

In this section we investigate the finite-length performance of the Self-Corrected Min-Sum (SCMS) decoder [24]. The objective is to determine if a correction circuit "plugged into" the noisy MS decoder can improve the robustness of the decoder to hardware noise.

The specificity of the SCMS decoder is to *erase* (*i.e.* set to zero) any variable-to-check message that changes its sign between two consecutive iterations. However, in order to avoid erasures propagation, a message cannot be erased if it has also been erased at the previous iteration. Hence, the SCMS decoder performs the same computations as the noisy MS, except that the **VN processing** step further includes a *correction step*, as follows[10]:

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

$$\alpha_{m,n}^{(\ell)} = \mathbf{s}_{\mathcal{M}} \left(\mathbf{a}_{\text{pr}} \left(\tilde{\gamma}_n^{(\ell)}, -\beta_{m,n}^{(\ell)} \right) \right);$$

if $\operatorname{sgn}\left(\alpha_{m,n}^{(\ell)}\right) \neq \operatorname{sgn}\left(\alpha_{m,n}^{(\ell-1)}\right)$ and $\alpha_{m,n}^{(\ell-1)} \neq 0$

$$\alpha_{m,n}^{(\ell)} = 0;$$

end

[10] Superscript (ℓ) used to denote the iteration number

The body enclosed between the **if** condition and the matching **end** is referred to as the *correction step*. In practical implementations, one needs to store the signs of the variable-to-check node messages and to keep a record of messages that have been erased by the self-correction step. We use the following notation:

- $s_{m,n}^{(\ell)} = \operatorname{sgn}\left(\alpha_{m,n}^{(\ell)}\right)$, the sign of the message $\alpha_{m,n}^{(\ell)}$;
- $e_{m,n}^{(\ell)} \in \{0,1\}$, with $e_{m,n}^{(\ell)} = 1$ if and only if the corresponding variable-to-check message has been erased at iteration ℓ; for $\ell = 0$, these values are all initialized as zero.
- $\text{SCU}(s_1, s_2, e) \stackrel{\text{def}}{=} (s_1 \oplus s_2) \otimes (1 \oplus e)$, for any $s_1, s_2, e \in \{0,1\}$, where $\oplus$ denotes the XOR operation (sum modulo 2) and $\otimes$ denotes the AND operation (product). Clearly $\text{SCU}(s_1, s_2, e) = 1$ if and only if $s_1 \neq s_2$ and $e = 0$.

Therefore, the VN-processing step of the SCMS decoder can be rewritten as follows:

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

$$\alpha_{m,n}^{(\ell)} = \mathbf{s}_{\mathcal{M}}\left(\mathbf{a}_{\text{pr}}\left(\tilde{\gamma}_n^{(\ell)}, -\beta_{m,n}^{(\ell)}\right)\right);$$

$$e_{m,n}^{(\ell)} = \text{SCU}\left(s_{m,n}^{(\ell)}, s_{m,n}^{(\ell-1)}, e_{m,n}^{(\ell-1)}\right);$$

if $e_{m,n}^{(\ell)} = 1$ **then** $\alpha_{m,n}^{(\ell)} = 0$; **end**

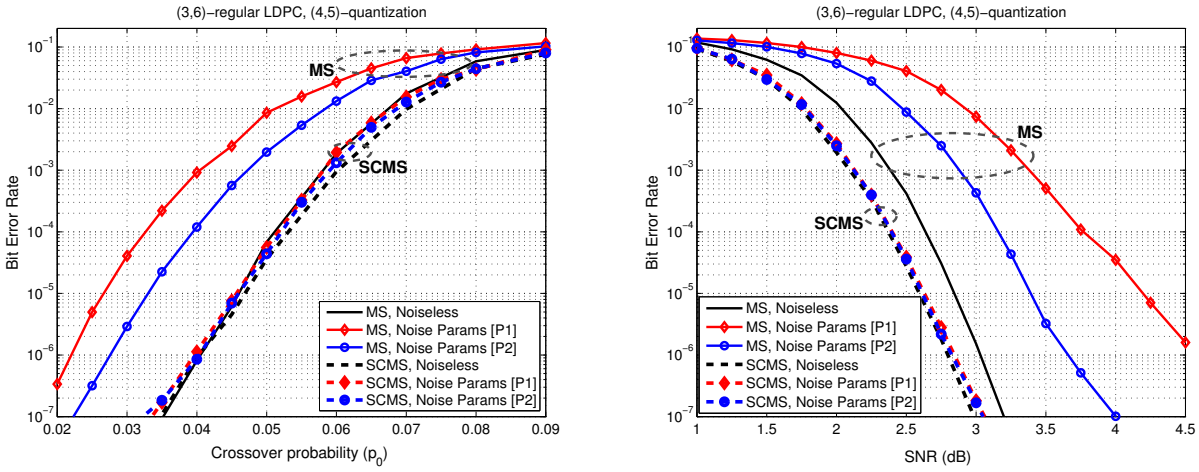
This reformulation of the VN-processing step allows defining a *noisy self-correction* step, by injecting errors in the output of the SCU operator. The noisy SCU operator with error probability p_{scu} is defined by:

$$\text{SCU}_{\text{pr}}(s_1, s_2, e) = \begin{cases} \text{SCU}(s_1, s_2, e), & \text{with probability } 1 - p_{\text{scu}} \\ 1 - \text{SCU}(s_1, s_2, e), & \text{with probability } p_{\text{scu}} \end{cases} \tag{1.21}$$

This error model captures the effect of the noisy logic or of the noisy storage of $s_{m,n}$ and $e_{m,n}$ values on the SCU operator. The SCMS decoder with noisy self-correction step is detailed in Algorithm 3.

The finite length performance of the noisy SCMS decoder is presented in Figure 1.22, for both BSC and BI-AWGN channels. For comparison purposes, Figure 1.22 also shows the performance of the noisy MS decoder. The parameters of the different noisy components are as follows:

[P1] sign-preserving adder with $p_a = 0.01$, $p_c = 0.01$, $p_x = p_{\text{scu}} = 0.001$ (red curves, diamond markers);
[P2] full-depth adder with $p_a = 0.001$, $p_c = 0.001$, $p_x = p_{\text{scu}} = 0.001$ (blue curves, circle markers).



(a) BSC channel ($\mu = 6$)

(b) BI-AWGN channel ($\mu = 5.5$)

Figure 1.22: BER performance comparison between noisy MS and noisy SCMS decoders

Algorithm 3 Noisy Self-Corrected Min-Sum (Noisy-SCMS) decoding

Input: $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ ($\mathcal{Y}$ is the channel output alphabet) ▷ received word
Output: $\underline{\hat{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N) \in \{-1, +1\}^N$ ▷ estimated codeword
Initialization
for all $n = 1, \ldots, N$ **do** $\gamma_n = \mathbf{q}(y_n)$;
for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** $\{\ \alpha_{m,n} = \gamma_n;\ \ s_{m,n} = \mathrm{sgn}(\gamma_n);\ \ e_{m,n} = 0;\}$
Iteration Loop
for all $m = 1, \ldots, M$ and $n \in \mathcal{H}(m)$ **do** ▷ **CN-processing**
$\beta_{m,n} = \mathbf{x}_{\mathrm{pr}}\left(\{\mathrm{sgn}(\alpha_{m,n'})\}_{n' \in \mathcal{H}(m)\setminus n}\right) \mathbf{m}_{\mathrm{pr}}\left(\{|\alpha_{m,n'}|\}_{n' \in \mathcal{H}(m)\setminus n}\right)$;
for all $n = 1, \ldots, N$ **do** ▷ **AP-update**
$\tilde{\gamma}_n = \mathbf{a}_{\mathrm{pr}}\left(\{\gamma_n\} \cup \{\beta_{m,n}\}_{m \in \mathcal{H}(n)}\right)$;
for all $n = 1, \ldots, N$ and $m \in \mathcal{H}(n)$ **do** ▷ **VN-processing**
$\alpha_{m,n} = \mathbf{s}_{\mathcal{M}}\left(\mathbf{a}_{\mathrm{pr}}\left(\tilde{\gamma}_n, -\beta_{m,n}\right)\right)$;
$e_{m,n} = \mathrm{SCU}_{\mathrm{pr}}\left(\mathrm{sgn}(\alpha_{m,n}), s_{m,n}, e_{m,n}\right)$;
$s_{m,n} = \mathrm{sgn}(\alpha_{m,n})$;
if $e_{m,n} = 1$ **then** $\alpha_{m,n} = 0$;
for all $\{v_n\}_{n=1,\ldots,N}$ **do** $\hat{x}_n = \mathrm{sgn}(\tilde{\gamma}_n)$; ▷ hard decision
if $\underline{\hat{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check
End Iteration Loop

Solid and dashed curves correspond respectively to the MS and SCMS performance. While the hardware noise alters the performance of the MS decoder, it can be seen that the noisy SCMS decoder exhibits very good performance, very close to that of the noiseless decoder. Therefore, one can think of the self-correction circuit as a *noisy patch* applied to the noisy MS decoder, in order to improve its robustness to hardware noise. The robustness of the SCMS decoder to hardware noise is explained by the fact that it has an intrinsic capability to detect unreliable messages, and discards them from the iterative decoding process [24].

1.8 Conclusion

In this chapter we investigated the performance of MS-based decoders on noisy hardware. We derived density evolution equations for the noisy MS decoder, and analyzed the decoder performance in terms of useful regions and target-BER thresholds. We also revealed the existence of a different threshold phenomenon, which was referred to as functional threshold. We further evaluated the finite-length performance of the MS and SCMS decoders, for various parameters of the hardware noise models. We highlighted the excellent performance of the noisy SCMS decoder, which provides virtually the same performance as the noiseless decoder, for a wide range of values of the hardware noise parameters. Finally, the results of this work may serve as guidelines for the design of noisy arithmetic components for Min-Sum-based decoders.

Chapter 2

The Finite-Alphabet Iterative Decoding Framework for Faulty Hardware

Abstract: *Recently, a new type of decoders referred to as finite alphabet iterative decoders (FAIDs), were introduced for LDPC codes [43, 45]. In these decoders, the messages are represented by alphabets with a very small number of levels, and the variable-to-check (v-to-c) messages are derived from the check-to-variable (c-to-v) messages and channel information through a predefined Boolean map. Although originally introduced to specifically address the error floor problem and designed to correct error events located on Trapping sets that usual decoders (Min-Sum, BP-based) cannot correct, the FAIDs offer also an excellent framework for the questions raised in the i-RISC project about fault-tolerant iterative decoding and long-term storage on reliable memories. One aspect of the advantages offered by the FAID framework is the ability to define a large collection of boolean maps, each defining a different decoding algorithm, but with potentially different behaviors in terms of tolerance to transient errors. In this deliverable, we will present the current stage of development of fault-tolerance for noisy-FAIDs, based on the techniques developed for the noisy Min-Sum decoders presented in the previous chapter. We show in particular that the behavior of noisy FAIDs is typically the same as the noisy Min-Sum with optimized channel amplitude. We also make the analogy between FAIDs and the offset-corrected Min-Sum and show similar behaviors. Finally, we show that within the diversity of multiple FAIDs, there are boolean maps which are naturally more robust to the hardware errors.*

2.1 Brief introduction of FAID decoders

It has been shown in [43, 45] that with an alphabet size of only seven levels in the message quantization, which translates to messages of 3-bit word length, the FAIDs can outperform floating-point BP decoders in the error-floor region over the binary symmetric channel (BSC). Although the current stage of development of FAIDs is restricted to column-weight-three LDPC codes and the BSC channel, this particular case is important for the i-RISC project since the errors in hardware typically occur as hard-decision errors, although certainly more complex than just a BSC model.

Let us give a brief presentation of the FAID decoders in this section.

2.1.1 Definitions

We now describe the general framework of FAIDs that was introduced in [45] for LDPC codes. An N_s-level FAID denoted by D is defined as a 4-tuple given by $\mathrm{D} = (\mathcal{M}, \mathcal{Y}, \Phi_v, \Phi_c)$. The message alphabet is finite and can be defined as $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$, where $L_i \in \mathbb{R}^+$ and $L_i > L_j$ for any $i > j$. It thus consists of $N_s = 2s + 1$ levels to which the message values are confined to. The sign of a message $x \in \mathcal{M}$ can be interpreted as the estimate of the bit associated with the variable node for which x is being passed to or from (positive for zero and negative for one), and the magnitude $|x|$

as a measure of how reliable this value is. In the case of the BSC, the set $\mathcal{Y}$, which denotes the set of possible channel values, is defined as $\mathcal{Y} = \{\pm C\}$. For the n-th symbol of the codeword, the channel value $y_n \in \mathcal{Y}$ corresponding to node v_n is determined based on its received value. Here, we use the mapping $0 \to C$ and $1 \to -C$.

Let $m_1, \cdots, m_{l-1}$ denote the extrinsic incoming messages to a node of degree l. The CNU function $\Phi_c : \mathcal{M}^{d_c-1} \to \mathcal{M}$ used for the update at a check node of degree d_c is given by

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \text{sgn}(m_j) \right) \min_{j \in \{1,\ldots,d_c-1\}} (|m_j|), \tag{2.1}$$

where sgn denotes the sign function. The VNU function $\Phi_v : \mathcal{Y} \times \mathcal{M}^{d_v-1} \to \mathcal{M}$ used for the update at a variable node v_n, $n = 0 \ldots N-1$ of degree d_v, can be described as a closed-form function given by

$$\Phi_v(m_1, m_2, \cdots, m_{d_v-1}, y_i) = Q\left(\sum_{j=1}^{d_v-1} m_j + \omega_n \cdot y_n \right), \tag{2.2}$$

where the function $Q(.)$ is defined based on a threshold set $\mathscr{T} = \{T_i : 1 \leq i \leq s+1\}$ such that $T_i \in \mathbb{R}^+$ and $T_i > T_j$ if $i > j$, and $T_{s+1} = \infty$.

$$Q(x) = \begin{cases} \text{sgn}(x) L_i, & \text{if } \; T_i \leq |x| < T_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

The weight ω_i assigned to the channel value in Eq. (2.2) is one of the main differences of FAIDs compared to the state-of-the-art decoders. It is computed from a symmetric function $\Omega : \mathcal{M}^{d_v-1} \to \mathbb{R}^{\geq 0}$ whose input arguments are the $d_v - 1$ incoming messages of a VNU. The function Ω could be linear or non-linear, and its purpose is to modify the output of the VNU update in order to prevent the failure of the message passing decoder on specific small topologies of error events referred to as *trapping sets*. The careful design of Ω through a systematic analysis of the dominant Trapping sets of regular $d_v = 3$ LDPC codes is the key feature of the FAID framework (see next section). As a result, the Ω function, and the corresponding FAID, are designed to improve the error-rate performance in the error floor region.

Note that a particular FAID is uniquely specified by the choice of the map for Φ_v as the function Φ_c is the same for all the considered FAIDs. Furthermore, the function Φ_v must satisfy the following two properties.

Property 1 (Property of symmetry)
$\Phi_v(y_n, m_1, ..., m_{d_v-1}) = -\Phi_v(-y_n, -m_1, ..., -m_{d_v-1})$

Property 2 (Property of monotonicity)
$\Phi_v(y_n, m_1, \ldots, m_{d_v-1}) \geq \Phi_v(y_n, m'_1, \ldots, m'_{d_v-1})$ *when* $m_j \geq m'_j \; \forall \; j \in \{1, \ldots, d_v - 1\}$.

Alternatively, for column-weight-three codes, the VNU function Φ_v can be represented as a simple two-dimensional Boolean map or look-up table (LUT) that is defined for a specific channel value. Table 2.1 shows an example of a Boolean map defining the function Φ_v of a 7-level FAID when the channel value is $-C$. The corresponding map for $+C$ can be deduced by symmetry.

At the end of each decoding iteration, the hard-decision bit corresponding to each variable node v_i is determined as the sign of $y_i + \sum_{j=1}^{d_v} m_j$.

2.1.2 Min-Sum-Based Decoders: Instances of FAIDs

Here, for performance comparisons with FAIDs, we consider two quantized Min-Sum-based decoders: the standard Min-Sum decoder, and the offset Min-Sum decoder. The VNU function Φ'_v used in these two decoders is given by

$$\Phi'_v(y_n, m_1, \ldots, m_{d_v-1}) = y_n + \sum_{j=1}^{d_v-1} m_j. \tag{2.3}$$

Table 2.1: Boolean map defining the VNU of a 7-level FAID when $y_n = -\mathrm{C}$

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

For the BSC channel considered here, the value y_n can take 2 values $\{-C, +C\}$.

For the standard min-sum decoder, the CNU is the same as the function (2.1) used in FAIDs. From (2.1), it is clear that the c-to-v messages can only have two possible magnitudes, which are the minimum and the second minimum among the magnitudes of all incoming v-to-c messages. Let the two magnitudes be denoted by Min1 and Min2 respectively. As a result, only four values need to be recorded for each check node: Min1, Min2, $S = \prod_{j=1}^{d_c} \mathrm{sgn}(m_j)$, and the index of the variable node, I, that provides Min1. Then the message to the variable node with index I has magnitude Min2 and the message to all other variable nodes have magnitude Min1. The sign of each c-to-v message can be computed by multiplying S with the sign of the corresponding v-to-c message.

For the offset Min-Sum decoder [44], the CNU involves the introduction of an offset factor γ, and is given by

$$\Phi'_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \mathrm{sgn}(m_j) \right) \max \left(\min_{j \in \{1,\ldots,d_c-1\}} (|m_j|) - \gamma, 0 \right) \tag{2.4}$$

It is evident from (2.4) that in addition to the calculation of the four values Min1, Min2, S, and the index I, the offset factor γ must be subtracted from each magnitude in order to determine the outgoing message. The offset factor γ serves at reducing the overestimate of the outgoing message produced by a check node using (2.1), especially when the magnitudes of the incoming messages are small. Therefore, the offset factor is also often referred to as a *correction factor*. An appropriate choice of the offset factor enables the decoder to achieve a performance approaching the performance of BP in the waterfall region while also offering possible improvement in the error floor region.

Note that for the offset Min-Sum decoder, an equivalent representation can be obtained by introducing the offset factor at the VNU instead. Using this equivalent representation, it can be shown that both the standard Min-Sum and the offset Min-Sum decoders are instances of FAIDs. This is due to the properties of symmetry and monotonicity that the VNU function Φ_v must satisfy. As an example, Table 2.2 represents the VNU function of a 3-bit offset Min-Sum decoder as a Boolean map which can also be treated as an instance of a 7-level FAID.

As can be seen in Table 2.2, the offset corrected Min-Sum has a very regular organization in its LUT representation, and the only non-linearity can be seen in the sequence of three zeros in each column of the LUT. This non-linearity is due to the offset correction and is already of great help in the iterative decoding process since the offset corrected Min-Sum provide much greater error correction capability than the simple Min-Sum. The LUT of the FAID decoder in Table 2.1 shows even more non-linearities than the offset corrected Min-Sum. One example is the fact that the amplitude gaps between two adjacent squares could be greater than one, with the extreme case that $l_{1,6} = -L_3$ and $l_{1,7} = 0$. Although it is difficult to directly relate those non-linearities to a particular effect in terms of improved error correction, the FAID analysis that we provide in [45, 46] and in this report, shows

that a specific organization of these non-linearities in the LUT can lead to better iterative decoders than the classical ones, especially in the error floor region.

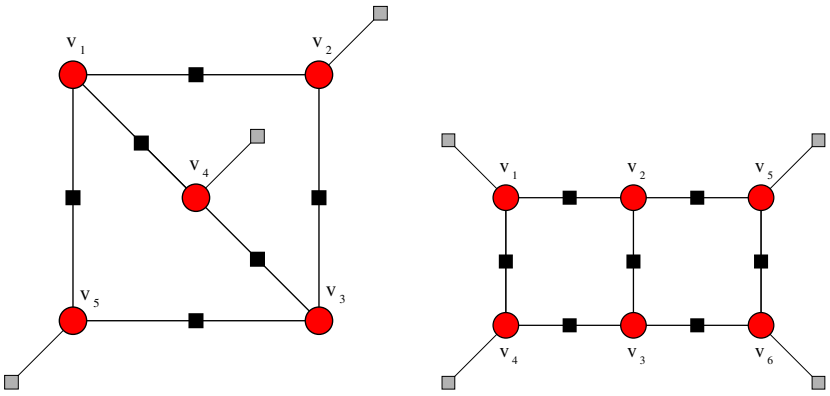
Table 2.2: VNU of a 3-bit offset Min-Sum represented as a FAID

m_1/m_2	$-L_3$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$	$+L_3$
$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$
$-L_2$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0
$-L_1$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0	0
0	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0	0	0
L_1	$-L_3$	$-L_2$	$-L_1$	0	0	0	L_1
L_2	$-L_2$	$-L_1$	0	0	0	L_1	L_2
L_3	$-L_1$	0	0	0	L_1	L_2	L_3

2.1.3 Discussion on the Design of FAIDs

The methodology used for designing FAIDs relies on the knowledge of potentially harmful structures called *trapping sets* [47] that can be present in the Tanner graph of the code and usually cause conventional iterative decoders to fail for certain low-weight error patterns. A notation typically used to denote a trapping set (TS) is (a, b) where a is the number of variable nodes and b is the number of odd-degree check nodes in the subgraph induced by the variable nodes [47].

Figure 2.1 shows the example of two trapping sets that are known to be dominant in the error floor region for regular $d_v = 3$ LDPC codes. The $(5, 3)$ TS is a subgraph consisting of 5 variable nodes and 3 odd-degree check nodes while the $(6, 4)$ TS is a subgraph consisting of 6 variable nodes and 4 odd-degree check nodes. If such structures are contained in the Tanner graph of the code, they cause iterative decoders (BP-based as well as Min-Sum-based) to fail when errors are located in the variable nodes of the trapping sets. The presence of these structures is the source of the error floors for LDPC decoders.



(a) $(5, 3)$ (b) $(6, 4)$

Figure 2.1: Examples of trapping sets for regular $d_v = 3$ LDPC codes.

The trapping set ontology is a database of trapping sets with a hierarchical organization established by the topological relations between trapping sets [48]. The selection method for FAIDs begins by identifying such harmful trapping sets from the trapping set ontology, and then analyzing the message passing algorithm of a given FAID on each of these isolated structures in order to examine its error correction capability. Moreover, the influence of an arbitrary neighborhood of the harmful structure during message passing is partially captured by considering different possible messages that enter the trapping set. Based on this analysis on each trapping set, the FAID with the best error correction capability is chosen. For the design of the FAID defined by Table 2.1, the analysis of the message

passing was done on the $(5, 3)$ TS as well as on the $(6, 4)$ TS. More details on the strategies and algorithms used to design good FAIDs can be found in [45].

Using the above approach, it is possible to design 7-level FAIDs that are capable of outperforming the floating-point BP as well as the quantized Min-Sum-based decoders in the error floor on a given code. Moreover, a single particularly good 7-level FAID identified from the selection methodology is capable of surpassing BP on several codes.

2.1.4 Simulation Results

In order to illustrate the efficiency of FAIDs, Figure 2.2 shows the frame-error-rate (FER) performance comparisons between the BP, the Min-Sum-based decoders, and the 7-level FAID as a function of the cross-over probability α over the BSC on a $(7807, 7177)$ QC-LDPC code with $L = 211$, $r = d_c = 37$, and $t = d_v = 3$. Table 2.1 was used as the VNU function Φ_v for the 7-level FAID.

The parameters for the 3-bit offset Min-Sum decoder and the 6-bit offset Min-Sum decoders are $(C = 2, \gamma = 1)$ and $(C = 10, \gamma = 3)$ respectively. Note that the offset correction values and the channel value of the Min-Sum decoders have been adapted to our particular simulation settings, and optimized through a density-evolution analysis by maximizing their decoding threshold for regular $d_v = 3$ codes on the BSC channel, combined with the selection procedure that was proposed for FAIDs based on trapping sets in order to obtain a better performance in the error floor region. To the best of our knowledge, we are not aware of any other decoder designs in the literature that specifically optimize the performance on the BSC.

From Figure 2.2, it is evident that the 7-level FAID clearly outperforms the 5-bit Min-Sum (which exhibits very poor performance), the 3-bit offset Min-Sum, and the floating-point BP decoder in the error-floor region. Remark that the 6-bit offset Min-Sum decoder approaches the performance of BP in the waterfall while achieving significantly improved performance in the error floor. For this code, the 7-level FAID with only 3 bits of precision is able to perform close to the 6-bit offset Min-Sum decoder in the error floor region.

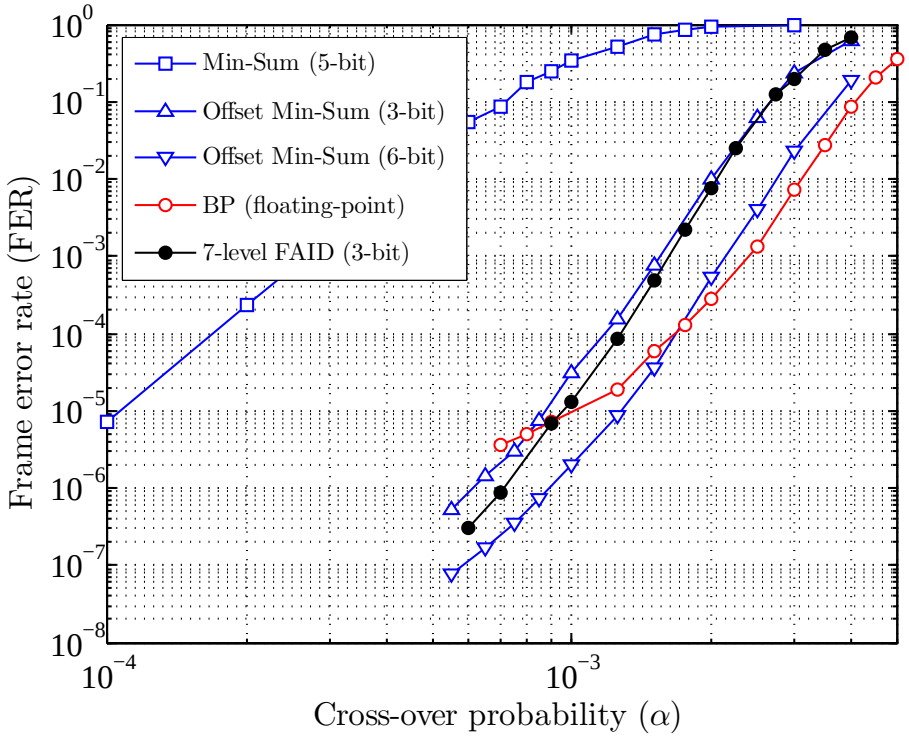


Figure 2.2: Performance comparisons between various LDPC decoders for a $(7807, 7177)$ QC-LDPC code

Additional simulation results are provided in Figure 2.3 to show that the good 7-level FAID depicted in Table 2.1 is capable of surpassing BP on other codes. These results were obtained on a $(2388, 1793)$ QC-LDPC code that was designed for the best BP performance (for that code rate and length) by avoiding certain harmful trapping sets during the code construction [49]. Even for such a code, the 7-level FAID provides superior FER performance in the error floor region compared to the floating-point BP. Note that for this code, the 7-level FAID surpasses the 6-bit offset Min-Sum decoder in the error-floor regions. Hence 7-level FAIDs, which are 3-bit decoders, are capable of outperforming not only the floating-point BP decoders but also 6-bit Min-Sum-based decoders. The number of maximum decoding iterations used for all the decoders is set to 100 in our simulations.

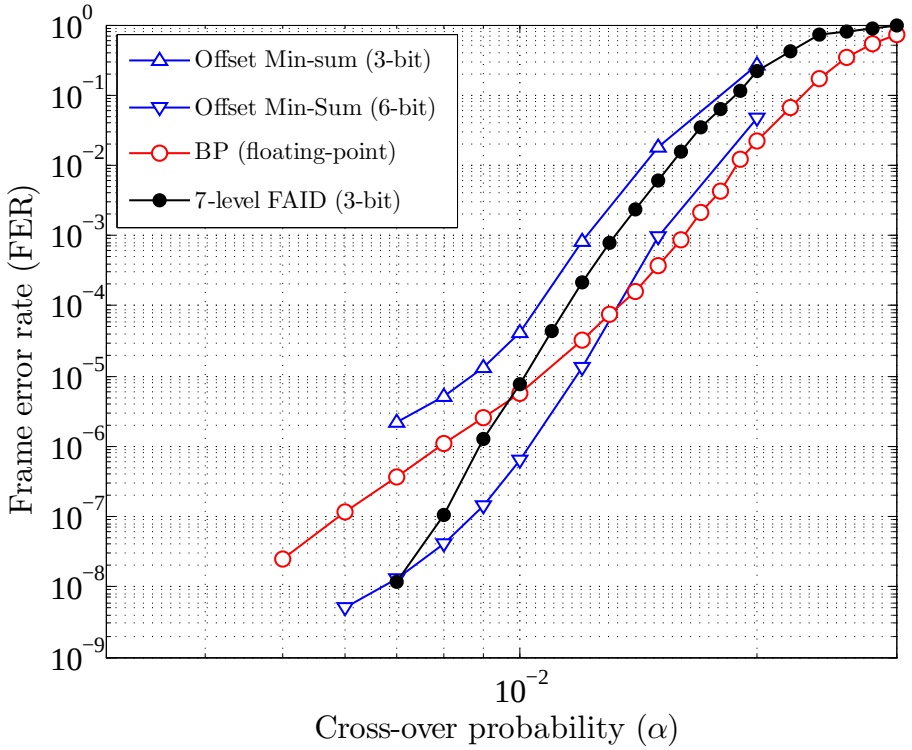


Figure 2.3: Performance comparisons between various LDPC decoders for a $(2388, 1793)$ QC-LDPC code

2.2 Faulty FAID Decoding and Analysis

2.2.1 Definition of Faulty FAID decoders

The introduction of error models in FAID decoders is different than for the Min-Sum based decoders, as the FAID rely on Boolean maps update rules and not arithmetic units. The question whether the FAID update rules should be implemented using local memories (with Look-Up tables) or with a circuit implementation of the boolean maps is still under investigation, and of course has an impact on the error model which should be used.

As a first step in this deliverable, we have decided to put the errors due to faulty hardware at the *"message level"* at the output of each elementary operation. With this assumption, the output of functions Φ_v, Φ_c and the decision step will be corrupted by a transient noise, following the models presented in the first chapter of this deliverable. We restrict the discussion and the study in the i-RISC project to FAID implemented in $N_s = 7$ states, with messages and error events stored on 3 quantization bits. Following the description of the full-depth and sign-preserving error models presented in chapter 1, the error models can be interpreted and represented as the concatenation of the update functions

Φ_v and Φ_c with a N_s-ary symmetric channel. The full-depth and the sign-preserving error models are presented as noisy channels on Figures 2.4 and 2.5.

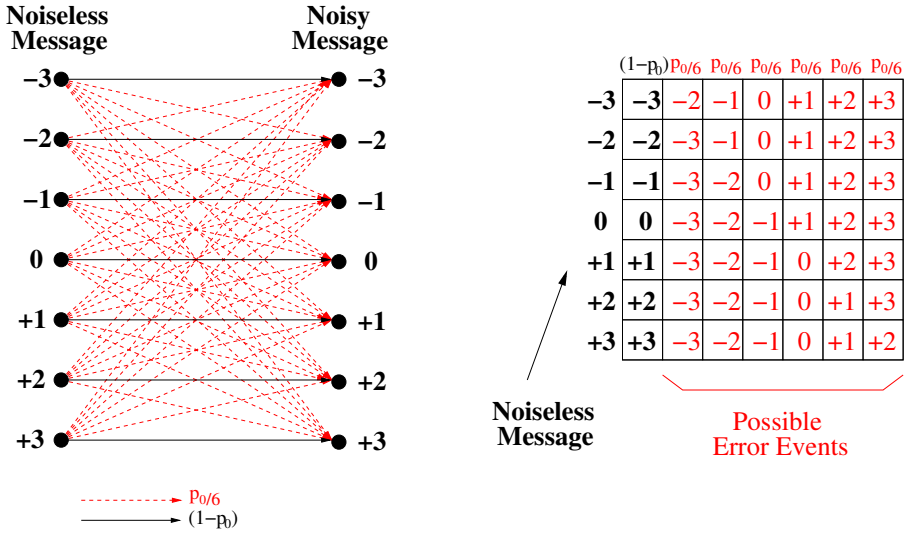


Figure 2.4: Full-Depth Error model seen as a N_s-ary symetric channel.

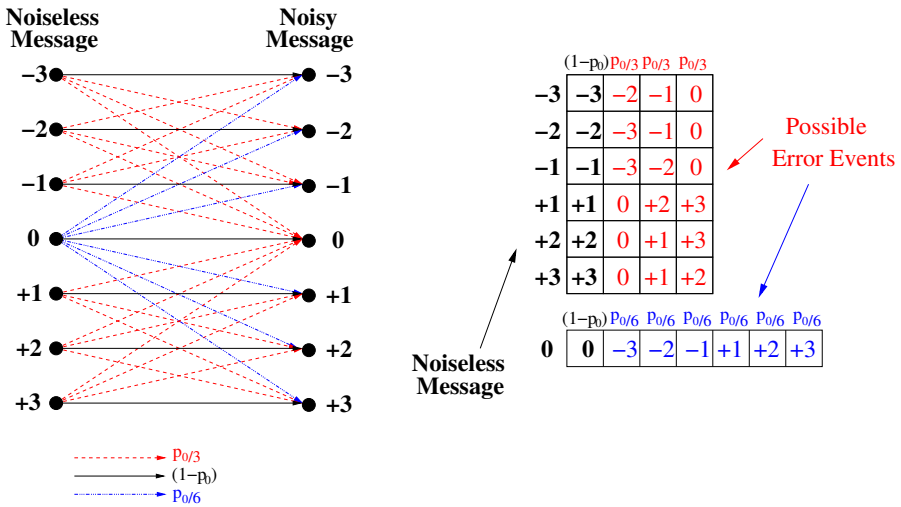


Figure 2.5: Sign-Preserving Error model seen as a N_s-ary symetric channel.

In the case of FAIDs, only the output of the Φ_v, Φ_c and the APP computation are supposed to be noisy, while the decision step is supposed to be noiseless and implemented in perfect hardware. As can be seen From figures 2.4 and 2.5, the error model can be interpreted itself as an additional Boolean map that is put on top of the maps Φ_v and Φ_c. Let us denote by $\mathcal{E}^{p_0}_{FD}(.)$ and $\mathcal{E}^{p_0}_{SP}(.)$ the error models expressed as Boolean functions for the full-depth and the sign-preserving cases, respectively. The noisy versions of the FAID update rules obtained from equations (2.1) and (2.2) are:

$$\tilde{\Phi}_c(m_1,\ldots,m_{d_c-1}) = \mathcal{E}^{p_c}\left(\left(\prod_{j=1}^{d_c-1}\operatorname{sgn}(m_j)\right)\min_{j\in\{1,\ldots,d_c-1\}}(|m_j|)\right) \tag{2.5}$$

$$\tilde{\Phi}_v(m_1,m_2,\cdots,m_{d_v-1},y_i) = \mathcal{E}^{p_v}\left(Q\left(\sum_{j=1}^{d_v-1} m_j + \omega_n\cdot y_n\right)\right) \tag{2.6}$$

where $\mathcal{E}$ can be any of the error model considered, and different error probabilities can be used for the two functions. Note that here p_v stands for the variable node update, and p_c for the check node update. Note that since the APP calculation is performed at the variable node in a message passing decoder, $\mathcal{E}^{p_v}$ will be used as a model to corrupt the APP computation, prior to the decision step.

Finally, and as already explained in Chapter 1, we stress the fact that the error models presented in this deliverables are far from being realistic and are not based on outputs of the WP2. In fact, in order to perform a theoretical and asymptotical analysis of iterative decoders, the local update functions and their noisy versions need to be symmetric, in the sense that they treat the binary values "0" and "1" equally. From the first analysis of WP2, it turns out that the error models that effectively come from faulty hardware are both non-symmetric and data dependent, which prevents any theoretical analysis based on the usual tools of Density Evolution or EXIT charts. However, in the rest of the i-RISC project, we will continue to explore means of incorporating the non-symmetries of the error models into our analysis.

2.2.2 Density Evolution for Faulty FAID

The main tool that we will use for the analysis of Faulty FAID is the noisy version of Density Evolution (DE) over the BSC. The DE is the description of the behavior of a decoding algorithm as a dynamical system representing the probability density function of the messages in the Tanner graph. This analysis is performed under the local independence assumption. The advantage of a DE analysis is that it both alleviates the dependence of the analysis from a particular LDPC code structure, and is valid on average over all possible LDPC code constructions, when infinite-lengths LDPC graphs are considered.

In the case of full precision messages, the DE is initialized with the probability density function of the channel noise, and has two typical dynamical behaviors: *(i)* either the recursion converges to a fixed point that is a Dirac mass on $+\infty$, in which case the decoder has successfully removed all the noise and converged to the correct stable solution, *(ii)* or the recursion converges to a stable fixed point that is a distribution different from a Dirac mass at $+\infty$, in which case the decoder failed to converge and to remove all the noise from the codeword. More details on DE convergence and the conditions under which the fixed point of DE can be stable or not can be found in [22].

For message passing decoders analyzed with DE, the threshold is defined as the level δ of channel noise which separates the two typical behaviors mentioned above. The parameter δ can represent either an SNR or a channel error probability indicating the noise level on the channel. The threshold characterizes the fact that the transition between the two different behaviors of DE is not smooth: for all the channels with noise level greater than δ, the DE does not converge, and for all the channels with noise level lower than δ, the DE successfully converges to the correct fixed point. The parameter δ is usually called DE threshold in the literature.

The DE recursion is usually not defined in a closed-form equation and requires heavy numerical evaluation techniques to be approximated. The only known exceptions are for the erasure channel model, for which DE can be expressed in closed-form, and for the BSC channel and hard-decision decoders (Gal-A and Gal-B decoders). For all other channels or more complex decoders, the DE thresholds has to be be numerically *approximated* using Monte Carlo or quadrature methods.

However, for quantized message passing decoders, for which the update rules are described as simple LUT operations, writing a program which computes *exactly* the DE recursion is quite simple and straightforward, especially for the case of $d_v = 3$ LDPC codes, where the variable node update is defined only by a 2D rule (see section 2.1). Since the error models that we consider are also described as LUT (see figures 2.4 and 2.5), the DE recursion for Faulty FAID is also simple to implement. In this section, we present a short description on the way to implement DE for Faulty FAIDs.

The density evolution is initialized with the density p_0 of the channel likelihood, that is, if $C = \pm L_1$

$$p_0(-L_1) = 1 - \alpha \quad p_0(+L_1) = \alpha \quad p_0(i) = 0 \text{ elsewhere}$$

The density of the output of a check node is obtained by recursions of an elementary step with only 2 incoming messages having the same density. Let $\tilde{p}_{vtoc}^{(\ell)}$ represent the density of the noisy v-to-c messages in the graph at iteration ℓ. The density $p_{vtoc}^{(\ell)}$ of the noiseless v-to-c messages is composed by N_s values in $[0,1]$ which sum up to 1, meaning $\sum_{i=-L_s}^{+L_s} p_{vtoc}^{(\ell)}(i) = 1$. The output density $p_{ctov}^{(\ell)}$ of an elementary update from the look-up table definition of Φ_c is:

$$p_{ctov}^{(\ell)}(k) = \sum_{(i,j):\Phi_c(i,j)=k} p_{vtoc}^{(\ell)}(i)\, p_{vtoc}^{(\ell)}(j) \qquad \forall k \in \{-L_s, \ldots, +L_s\} \tag{2.7}$$

This equation holds for the check node recursive implementation with $d_c - 2$ steps for any check-node of degree d_c.

In the case of a faulty decoder, the function Φ_c in equation (2.7) needs to be replaced by the function $\tilde{\Phi}_c$ of equation (2.5). As $\tilde{\Phi}_c$ is defined as the concatenation of Φ_c and of the error model $\mathcal{E}^{p_c}$ defined as a Boolean map, we proceed the following way. Let $\tilde{p}_{ctov}^{(\ell)}$ be the density of the output messages of a *noisy* parity-check node update, we have:

$$\tilde{p}_{ctov}^{(\ell)}(k) = \sum_{(i):\mathcal{E}^{p_c}(p_{ctov}^{(\ell)}(i))=k} p_{ctov}^{(\ell)}(i)\, p_{\mathcal{E}^{p_c}}(i,k) \qquad \forall k \in \{-L_s, \ldots, +L_s\} \tag{2.8}$$

where $p_{\mathcal{E}^{p_c}}(i,k)$ is the error transition probability between the noiseless message value i and the noisy message value k. For example, in the case of the sign-preserving model of Figure 2.5, $p_{\mathcal{E}^{p_c}}(i,k)$ can take only 3 values that are $p_c/3$, $p_c/6$ or $(1-p_c)$.

We proceed the same way to obtain the DE equation through a variable node. Let $\tilde{p}_{ctov}^{(\ell)}$ be the density of the noisy inputs of a $d_v = 3$ variable node, and $p_{vtoc}^{(\ell+1)}$ be the density of its noiseless output. From equation (2.2) we get:

$$\begin{aligned} p_{vtoc}^{(\ell+1)}(k) \;=\; & \sum_{(i,j):\Phi_v(i,j,C=-L_1)=k} \tilde{p}_{ctov}^{(\ell)}(i)\, \tilde{p}_{ctov}^{(\ell)}(j)\, p_0(-L_1) + \\ & \sum_{(i,j):\Phi_v(i,j,C=+L_1)=k} \tilde{p}_{ctov}^{(\ell)}(i)\, \tilde{p}_{ctov}^{(\ell)}(j)\, p_0(+L_1) \qquad \forall k \in \{-L_s, \ldots, +L_s\} \end{aligned} \tag{2.9}$$

As for the check-node update, the error model $\mathcal{E}^{p_v}$ is then applied at the output of the variable node update:

$$\tilde{p}_{vtoc}^{(\ell+1)}(k) = \sum_{(i):\mathcal{E}^{p_v}(p_{vtoc}^{(\ell+1)}(i))=k} p_{vtoc}^{(\ell)}(i)\, p_{\mathcal{E}^{p_v}}(i,k) \qquad \forall k \in \{-L_s, \ldots, +L_s\} \tag{2.10}$$

Applying recursively the sequence of 4 equations (2.7), (2.8), (2.9) and (2.10) implements one recursion of the *exact* Noisy Density Evolution for FAIDs over the BSC channel.

2.2.3 Noisy Density Evolution Analysis

As said in Chapter 1 for the Min-Sum based decoders, noisy DE is very different from the noiseless DE in the sense that the noise cannot be totally removed by the decoder due to the faulty hardware that is used even at the final decision step. As a result, the threshold of noisy DE cannot be defined as the limit channel parameter for which the zero-error probability is reached by the decoder. Similarly to the definitions given in chapter 1, we will propose two kinds of noisy DE thresholds.

Following [1], we define first the *useful* thresholds as the limit channel parameter for which the error probability at the end of the DE equation is lower than the initial error probability from the channel initialization. Let $P_e^{(\ell)}$ be the probability of error deduced from the distribution of the APPs at the end of ℓ DE recursions. As $P_e^{(\ell)}$ is a function of the decoder parameters, and in the case of FAIDs, of

the particular FAID update rule Φ_v, we will denote the probability of error $P_e^{(\ell)}(p_v, p_c, \alpha, \Phi_v)$. The useful threshold is thus defined as:

$$\delta_{UR} = max(\alpha) \text{ such that } P_e^{(\infty)}(p_v, p_c, \alpha, \Phi_v) < \alpha \tag{2.11}$$

The region of channel values $\alpha < \delta_{UR}$ and decoder parameters (p_v, p_c) that satisfy this condition constitutes the *useful region* of the decoder.

For the BSC channel, finding the useful decoding threshold corresponds to finding the maximum value of α such that equation (2.11) is verified in less than a given number of iterations. The threshold value of α can be found quickly by dichotomic search.

Although the useful region is a good indicator which tells what are the faulty hardware conditions (error probabilities (p_v, p_c)), and the maximum channel noise that a noisy decoder can tolerate to *reduce* the level of noise, it might not be sufficient to define and identify strong faulty decoders, that is faulty decoders which on top of reducing the channel noise, can correct the maximum number of errors.

In this deliverable, as an alternative to the useful threshold and region, we introduce another threshold definition, which relies on more stringent convergence conditions of the noisy DE recursion. Following the analysis proposed in Chapter 1 (see Proposition 1), the error probability at decoding iteration ℓ is lower-bounded as follows:

Proposition 2 *(a) For the sign-preserving error model:* $P_e^{(\ell)} \geq \frac{1}{N_s - 1} p_v = P_{e,SP}^{+\infty}$.

(b) For the full-depth error model: $P_e^{(\ell)} \geq \frac{1}{2} p_v + \frac{1}{2(N_s - 1)} p_v = P_{e,FD}^{+\infty}$.

We define the *"Functional Threshold"*, with respect to these lower bounds as the limit channel parameter at which the probability of error converges in the vicinity of these lower bounds, *i.e.*:

$$\delta_{FR} = max(\alpha) \text{ such that } P_{e,\cdot}^{+\infty} < P_e^{(\infty)}(p_v, p_c, \alpha, \Phi_v) < a\ P_{e,\cdot}^{+\infty} \tag{2.12}$$

The region of channel values $\alpha < \delta_{FR}$ and decoder parameters (p_v, p_c) that satisfy this condition constitutes the *functional region* of the decoder. The value of the parameter $a > 1$ can be adapted to the other parameters (p_v, p_c) such that the range between $P_{e,\cdot}^{+\infty}$ and $a\ P_{e,\cdot}^{+\infty}$ is small enough to reflect the dynamical behavior described in the rest of this section. Throughout this deliverable, we have chosen a value of $a = 2$.

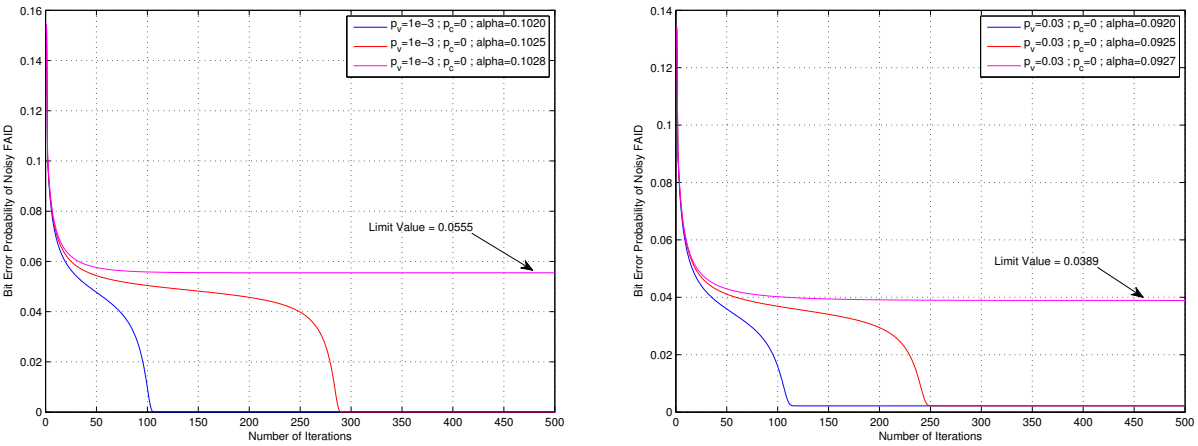
In the rest of this section, only the analysis using the sign-preserving error model is presented, but this study can be generalized to any other symmetric error model, including the full-depth error model, with possibly different conclusions.

The difference between the useful and the functional region might not be clear from these definitions and we provide now an explanation why this distinction is important. We show on Figures 2.6(a)-2.7(a) the sharp transition effect that appears around the functional threshold value. Figure 2.7(a) draws the evolution of $P_e^{(\ell)}(p_v, p_c, \alpha, \Phi_v)$ under DE for different parameter values and Φ_v being defined as the offset-corrected min-sum on 3 quantization bits (see Table 2.2). Figures 2.6(a) and 2.6(b) represent also the probability of error for various parameters, but for the FAID decoder defined in Table 2.1. As can be noticed, for a given choice of p_v and p_c, the values of α have been carefully chosen so as to be close to δ_{FR} and reflect the sharp transition behavior of faulty decoders around the functional threshold.

Let us first discuss the figures 2.6(a) and 2.6(b) on FAID. For the case of $p_v = 1e^{-3}$ and $p_c = 0.0$, the value of the functional threshold is $\delta_{FR} = 0.1026$ while the value of the useful threshold is larger, equal to $\delta_{UR} = 0.1149$. However, when the channel error probability α is slightly below the functional threshold $\delta_{FR} = 0.1026$, the error probability has the *early plateau phenomenon* mentioned in Chapter 1 for the Min-Sum decoder. The error probability flattens around 0.0500 for some iterations before eventually converging to a very small error probability, close to $P_{e,\cdot}^{+\infty}$. In this case, the decoder behavior

is in accordance with what we can expect from an error correcting decoder, that is converging to the minimum error probability after successful decoding, that is 0 in the case of noiseless decoders, and $P_{e,\cdot}^{+\infty}$ in the case of noisy decoders. On the contrary, when α is slightly larger than the functional threshold $\delta_{FR} = 0.1026$, the error probability flattens indefinitely and never reaches the minimum achievable error probability. Although $\alpha = 0.1028$ is a value which belongs to the useful region, as after decoding the error probability is reduced to 0.0555, we can argue the fact that this is not a good situation for the decoder, as it converges to a fixed point which is clearly not the desired one. Note that this is indeed a threshold behavior as the difference in α values to switch from one behavior to another is very small. This threshold behavior around δ_{FR} is also observed for the case of a more noisy FAID decoder, with $p_v = 0.03$.

In our opinion, for these reasons, the functional threshold δ_{FR} represents a better measure of robustness of faulty decoders than the useful threshold δ_{UR}. This observation is new and has not been reported in the literature. We do not understand yet how $P_{e,\cdot}^{+\infty}$ and the limiting error probability value in the neighborhood of δ_{FR} are linked to the other parameters or the type of decoder. Note that for the case of the offset-corrected Min-Sum that is drawn in figure 2.7(a), the same threshold behavior is observed, but in this case, the value of $\delta_{FR} = 0.0997$ and the limit error probability 0.0995 are very close, which in turns indicates that both thresholds δ_{FR} and δ_{UR} are almost equal. When this difference is large, as for the FAID case, it means that the difference between δ_{FR} and δ_{UR} can be large.



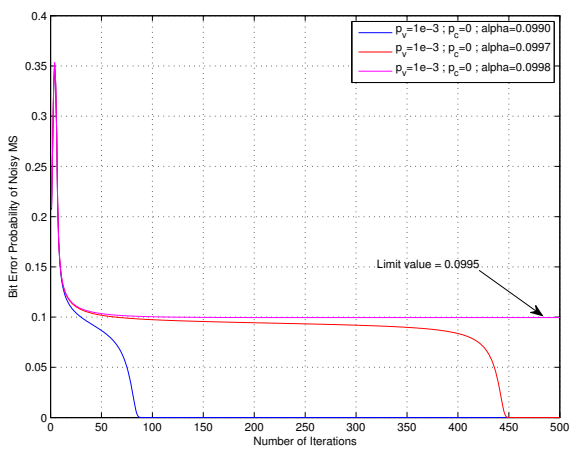
(a) $p_v = 1e^{-3}$ (b) $p_v = 0.03$

Figure 2.6: Threshold behavior of a FAID decoder around the functional threshold.

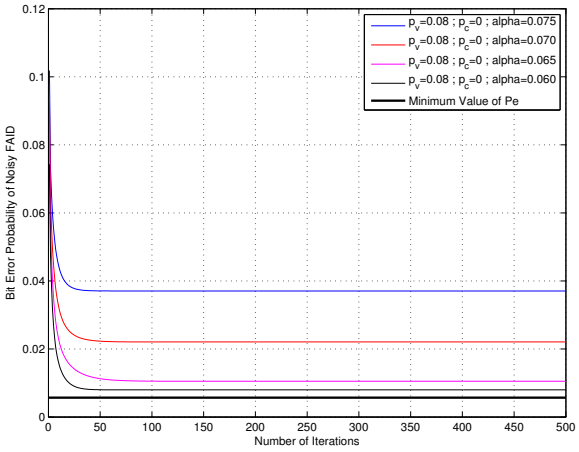
Finally, another kind of dynamical behavior has been observed when the error probability of the faulty hardware at the variable node update p_v is very large, as indicated on figure 2.7(b) for a FAID with $p_v = 0.08$. In this case the plateau phenomenon does not appear, and there is a smooth transition from the functional region to the useful region.

Considering the analysis and the different definitions of thresholds proposed in this section, we will now draw some useful and functional region for the two different decoders that we have compared in this section, *i.e.* the offset corrected Min-Sum of table 2.2, and the FAID decoder of table 2.1. On figure 2.8(a), we show the two regions for both decoders, when the check-node update is noiseless, $p_c = 0.0$. The useful region of the FAID is greater than the one of the offset Min-Sum, which seems to indicate that the FAID is more robust to faulty hardware than the Min-Sum. Note however that we have the opposite conclusion with respect to the functional regions, which would then indicate that the offset Min-Sum is more robust. As we can see, relying only on the useful region might be problematic when we wish to compare different decoders together.

This is confirmed in the next figure 2.8(b), where the useful and functional regions are now drawn for a value of $p_c = 0.1$. Although $p_c = 0.1$ might appear as an extremal case since 10% of error at the check-node update can seem too large, it clearly shows that the FAID regions are very much



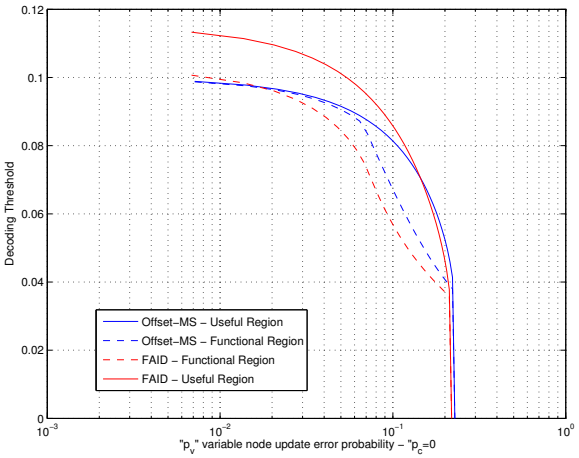
(a) Threshold behavior of the 3-bit Offset Min-Sum Decoder around the functional threshold



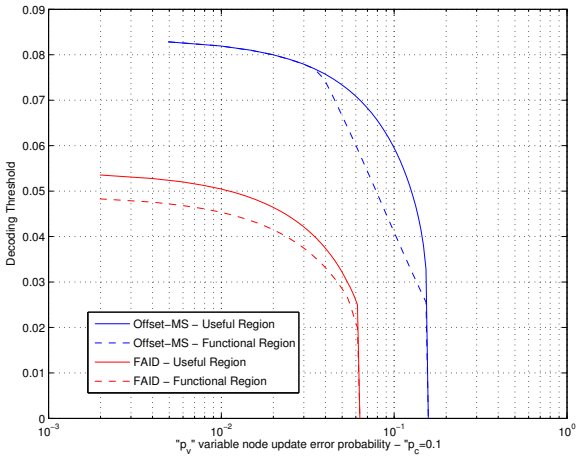
(b) Dynamical behavior of FAID without the plateau phenomenon - $p_v = 0.08$

Figure 2.7: Different Dynamical behaviors of noisy DE.

worse (smaller) than the Min-Sum ones, showing that the FAID is probably less robust to faulty hardware than the offset Min-Sum. This observation clearly demonstrates that the useful region does not characterize nor predict the fault-tolerance of decoders, and that relying on the functional region (or the functional thresholds in general) might be more predictive. This is however a work in progress and we do not claim that all issues are solved in this deliverable. We plan to continue in this direction during the second period of the i-RISC project.



(a) case of $p_c = 0.0$



(b) case of $p_c = 0.1$

Figure 2.8: Useful and Functional Regions of the Offset-corrected Min-Sum and FAID

2.2.4 Selection of FAIDs based on the Functional Region

As shown in the previous section, the useful and functional regions for FAIDs can vary and have different shapes and areas. As a result, the FAID framework allows us to look for specific Boolean update rules that are less sensitive than other to faulty-hardware. From our first analysis of this problem, we have identified several possible ways of defining FAID robust to faulty hardware, and we present a first set of results and conclusions in this section, based on the computation of the functional thresholds δ_{FR}.

like in the previous section, only the analysis using the sign-preserving error model is presented, but this study can be generalized to any other symmetric error model, with possible different conclusions.

One of the core idea of our previous works on FAIDs was to rely on the huge multiplicity of potentially good iterative decoders, with the aim of identifying the best ones, which are not necessarily deduced from Belief-Propagation equations or Min-Sum equations. In [45], our goal was to identify FAIDs which are good decoders in the error floor, by searching for the update rules Φ_v which correct the largest number of errors on the problematic topologies of the Tanner graph, the trapping sets. In the i-RISC project, we still want to capitalize on the diversity of FAID update rules and behavior to identify if there are iterative decoders which are naturally more robust than others under faulty-hardware implementation.

The total number of all possible FAIDs with N_s levels can be deduced from the following theorem [50, 51].

Theorem 2.1 (Number of FAIDs) *The total number $K_A(N_s)$ of symmetric lexicographically orders N_s-level FAIDs is given by*

$$K_A(N_s) = \frac{H_2(3N_s)H_1(N_s)H_2(N_s-1)}{H_2(2N_s+1)H_1(2N_s-1)} \tag{2.13}$$

where $H_k(n) = (n-k)!\,(n-2k)!\,(n-3k)!\ldots$ is the staggered hyper-factorial function.

Based on theorem 2.1, the total numbers of FAIDs for $N_s = 5$, $N_s = 7$, and $N_s = 9$ levels are shown in Table 2.3.

Table 2.3: Number of FAID Decoders

Total number of variable node LUTs ($N_s = 5$)	28 314
Total number of variable node LUTs ($N_s = 7$)	530 803 988
Total number of variable node LUTs ($N_s = 9$)	230 316 871 499 560

Even by restricting the message alphabet size to $N_s = 7$, the number of possible FAIDs is too large for a systematic analysis. Instead, we will rely on our previous work on FAID, and start with a collection of $N_D = 5291$ FAIDs which have been selected by the analysis on noisy-trapping sets presented in [45]. As a result of this selection process, all of these N_D FAIDs have both good DE thresholds (noiseless case), and good performance in the error floor. From this collection of FAIDs, we select one of the best in the error floor, which is described in the following Table 2.4, and that we will denote $\Phi_v^{(\mathrm{opt})}$ in the rest of this chapter.

Table 2.4: FAID rule $\Phi_v^{(\mathrm{opt})}$ reported in [45] optimized for the error floor

m_1/m_2	$-L_3$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$	$+L_3$
$-L_3$	-3	-3	-3	-3	-3	-3	-1
$-L_2$		-3	-3	-3	-2	-1	$+1$
$-L_1$			-2	-2	-1	-1	$+1$
0				-1	0	0	$+1$
$+L_1$					0	$+1$	$+2$
$+L_2$						$+1$	$+3$
$+L_3$							$+3$

We now conduct a faulty DE analysis on this set of $N_D = 5291$ FAIDs by computing, for each of them, the values of their functional threshold under different error model parameters (p_v, p_c). For

the N_D FAIDs, we have drawn on Figure 2.9 the distribution of their functional thresholds, computed with noisy DE for the ensemble of $(d_v = 3, d_c = 5)$ LDPC codes. The right distribution in blue shows the distribution of the noiseless DE, and it can be seen that almost all decoders have good decoding thresholds concentrated around $\delta = 0.1$. As we introduce more and more noise in the hardware, both at the check node and at the variable node updates, the shape of the distribution moves to the left, and eventually will reach $\delta = 0.0$, which means that the decoder is too noisy to correct even a small fraction of errors. More importantly, we can notice that the shapes of these distribution differ, and in particular the one corresponding to $(p_v = 0.05, p_c = 0.03)$ is very wide, ranging from $\delta = 0.02$ to $\delta = 0.08$. This is a first good sign of the interest of our approach, as it shows that different FAIDs have different behaviors when hardware noise is introduced. Otherwise the blue right distribution would have only shifted to the left, but without change in its shape.

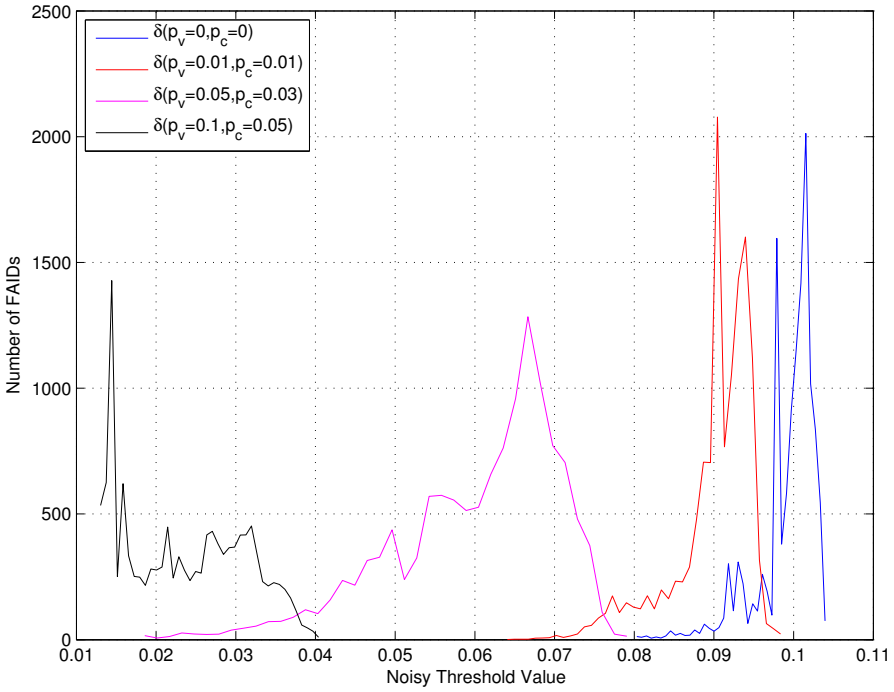


Figure 2.9: Distribution of the functional noisy DE threshold for 5291 FAIDs.

In principle, the FAIDs in the right tail of the distribution are the best decoders, as they have the highest noisy decoding threshold. However, a decoder which is in the right tail of the noiseless distribution may not be in the tail of the noisy distribution, and vice-versa. In order to figure out the decoders which are good in all noise situation (noiseless or noisy decoders), we have plotted on Figure 2.10 the distribution of the difference between the noiseless DE threshold and the noisy DE functional threshold with $(p_v = 0.05, p_c = 0.03)$. The result presented on this figure is highly interesting. The distribution ranges from 0.005 to 0.08, which is almost the same range as the difference between the right tail of the blue (noiseless) distribution and the left tail of the magenta (noisy) distribution on Figure 2.9.

This shows the following unexpected and promising result, and is a first step toward a clean definition of the robustness of FAID to faulty-hardware:

- the FAIDs for which the difference $\delta_{FR}(p_v = 0.0, p_c = 0.0) - \delta_{FR}(p_v = 0.05, p_c = 0.03)$ is the minimum corresponds to decoders which are in the right tail of both distributions, and is then expected to be a good decoder in both situations when the decoder is noiseless or noisy. We will qualify those decoders as *robust*, and we show on Table 2.5 one of the most robust FAID, combining good thresholds for the noiseless and noisy DE. It will be denoted $\Phi_v^{(\text{robust})}$ in the rest of the chapter.

- more surprinsingly, the FAID for which the difference is the maximum corresponds to a decoder in the right tail of the noiseless thresholds, but in the left tail of the noisy thresholds. In other words, this would be a good decoder on noiseless hardware, but a catastrophic decoder on noisy hardware. Such an extreme behavior was unexpected, and we plan to continue the characterization and analysis of these cases. We will qualify such decoders as *non-robust*. One example, which will be denoted $\Phi_v^{(\text{non-robust})}$ in the rest of the chapter, is given in Table 2.6.

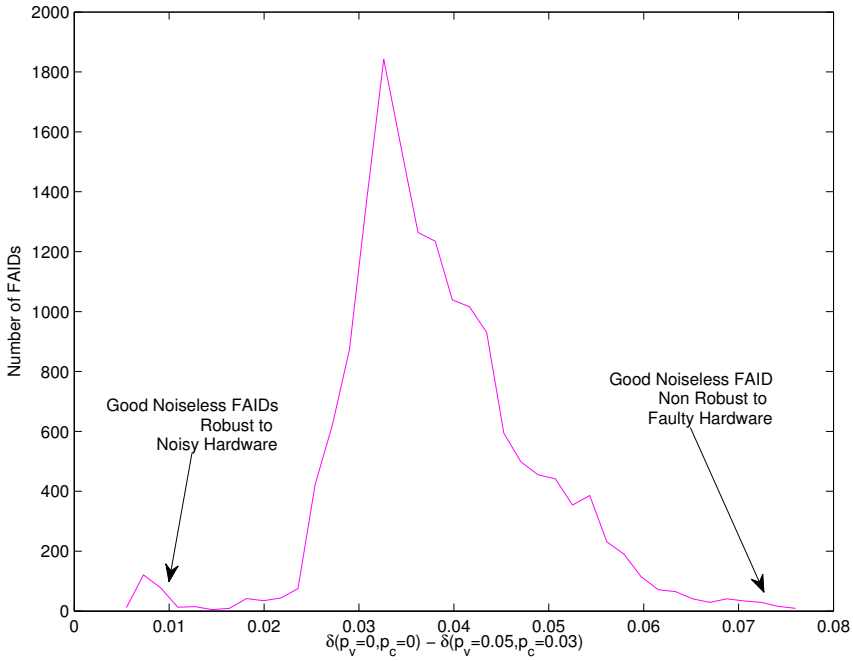


Figure 2.10: Distribution of the difference between Noiseless Thresholds and Noisy Thresholds for 5291 FAIDs. This plot, ranging from values close to 0 to values close to the maximum noiseless thresholds clearly shows that there are robust and non-robust FAIDs.

Table 2.5: FAID rule $\Phi_v^{(\text{robust})}$ optimized for the Robustness to Faulty Hardware (minimum difference between Noiseless and Noisy DE thresholds)

m_1/m_2	$-L_3$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$	$+L_3$
$-L_3$	-3	-3	-3	-3	-3	-2	0
$-L_2$		-3	-3	-3	-2	-2	$+1$
$-L_1$			-3	-2	-1	-1	$+1$
0				-1	-1	0	$+1$
$+L_1$					0	$+1$	$+2$
$+L_2$						$+2$	$+2$
$+L_3$							$+3$

Table 2.6: FAID rule $\Phi_v^{(\text{non-robust})}$ not robust to faulty Hardware (maximum difference between Noiseless and Noisy DE thresholds)

m_1/m_2	$-L_3$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$	$+L_3$
$-L_3$	-3	-3	-3	-3	-3	-3	0
$-L_2$		-3	-3	-3	-2	0	$+2$
$-L_1$			-2	-2	-1	0	$+2$
0				-1	0	$+1$	$+3$
$+L_1$					0	$+1$	$+3$
$+L_2$						$+1$	$+3$
$+L_3$							$+3$

2.3 Finite length Simulation Results

Finally, in this section, we give some simulation results with the best noisy FAIDs that have been identified by the noisy DE analysis. This finite length study will also confirm our interpretation from the outcome of the noisy DE analysis. Our main purpose is to compare the following three FAID decoders:

- $\Phi_v^{(\text{opt})}$, which has been optimized for noiseless decoding with low error floor,
- $\Phi_v^{(\text{robust})}$, which has been optimized for robustness to faulty hardware,
- $\Phi_v^{(\text{non-robust})}$, which has been selected with the maximum discrepancy between noiseless and noisy decoding.

All results in this section have iteration number of 100. The considered channel is a BSC. We have compared the three defined decoders on two different codes, the $(155, 93)$ Tanner code from [25], with degrees $(d_v = 3, d_c = 5)$, and a quasi-cyclic LDPC code $(444, 111)$ with degrees $(d_v = 3, d_c = 12)$. The simulation results are presented on Figures 2.11 and 2.12.

Let us first discuss the noiseless simulations first. Since $\Phi_v^{(\text{opt})}$ has been optimized for low error floor, it seems natural to see that it performs better on the two codes compared to the two other FAIDs. Also, since the two FAIDs $\Phi_v^{(\text{robust})}$ and $\Phi_v^{(\text{non-robust})}$ belong to a pre-determined set of good FAID decoders, they have reasonable performance in the noiseless case, $\Phi_v^{(\text{non-robust})}$ being even very close to $\Phi_v^{(\text{opt})}$ for the $(444, 111)$ QC-LDPC code.

Now, taking a look at the noisy curve, one can see that the results are in compliance with the conclusions from the noisy functional thresholds analysis. Indeed, $\Phi_v^{(\text{robust})}$ has been chosen for minimum difference between the noiseless case and the noisy case, while $\Phi_v^{(\text{non-robust})}$ has been selected for the maximum difference. We observe the same behaviors on both codes for finite length simulations. Another interesting remark is that it seems that the error floor of $\Phi_v^{(\text{robust})}$ under noiseless and noisy cases will eventually coïncide as the FER slope seems better for the $\Phi_v^{(\text{robust})}$ under faulty hardware. This raises the question of the behavior of these decoders in the error floor region and the possibility that noisy decoders could even outperform their noiseless version in the deep error floor. This study is planned in the DoW of the i-RISC project. Finally, we note that the question that we addressed at the beginning of this chapter, that is whether it makes sense to search for iterative decoders which are naturally robust to faulty hardware and could be specifically designed for fault-tolerance, is positively answered by both the noisy DE analysis and the finite length simulations. As a matter of fact, $\Phi_v^{(\text{opt})}$ would be the natural first choice for the implementation of a powerful FAID on a device, but we can see here that there also exists other FAIDs update rules, which have a slight performance degradation in

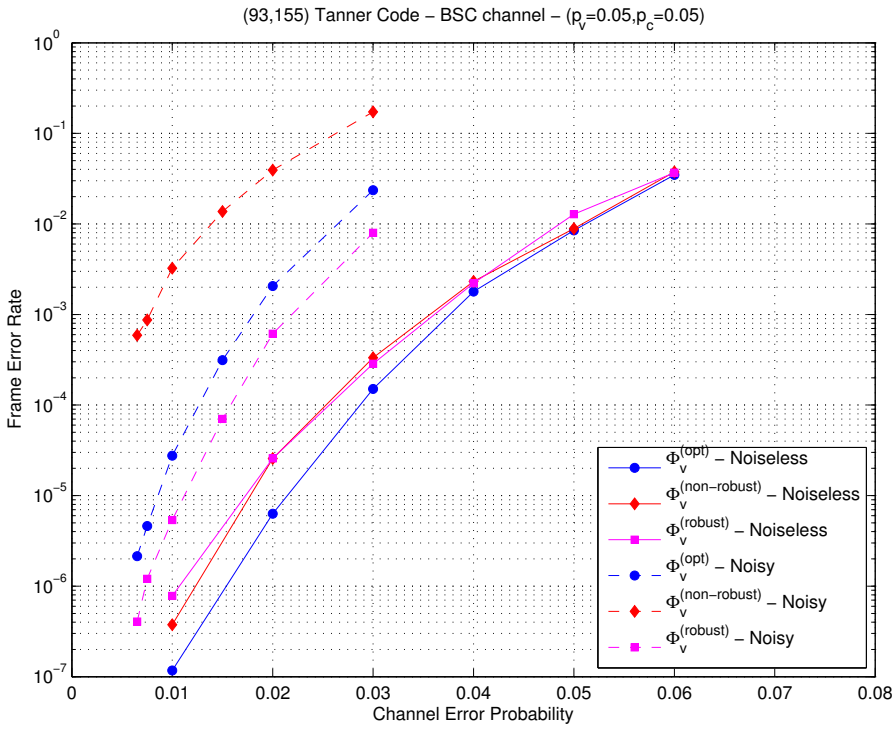


Figure 2.11: Performance of Noiseless and Noisy FAIDs on the $(93, 155)$ Tanner code, with $(d_v = 3, d_c = 5)$ and $(p_v = 0.05, p_c = 0.05)$.

the noiseless case, but can outperform $\Phi_v^{(\mathrm{opt})}$ in the noisy case. In this way, $\Phi_v^{(\mathrm{robust})}$ is a demonstrative example of the existence of such decoder.

2.4 Conclusion and future developments in the i-RISC project

To conclude this chapter, we confirm that with respect to the theoretical analysis of FAID iterative decoding with simple symmetric error models, the FAID framework provides a useful tool for selecting iterative decoders with maximum fault-tolerance features. In particular, we have already shown that some FAIDs are naturally more robust than others to transient noise, and therefore more adapted to be implemented on faulty-hardware. Although at its preliminary stage, this study raised several interesting issues that we plan to explore in the rest of the i-RISC project. In particular:

- the introduction of non-symmetric error models in our analysis, or at least in the Monte-Carlo simulation. The added complexity is that asymptotical analysis with non-symmetric decoding rule is much more problematic (and not solved in the literature to the best of our knowledge). A coset-like analysis would maybe be helpful for the characterization, but not necessarily the design of the decoders,
- include memory in the FAIDs to help combating the circuit noise, and especially study the *self-corrected* FAIDs (SC-FAIDs),
- include in our analysis real hardware implementations of FAIDs, starting from the promising work in [52], and including error models from WP2 that fits to the particular hardware implementation. A VHDL model of the FAID implementation will be developed and tested under faulty hardware.

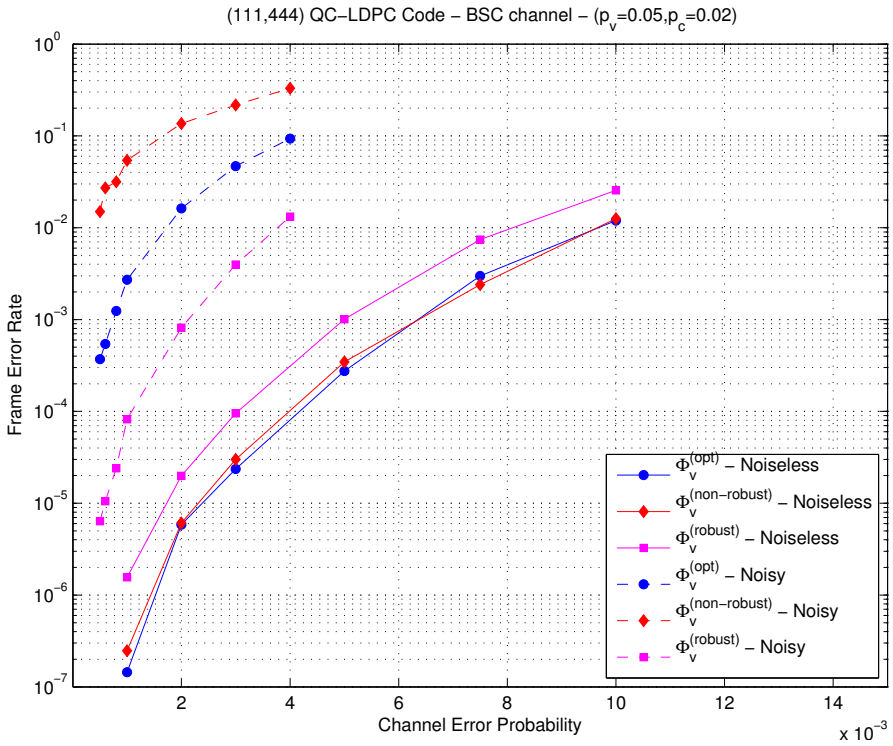


Figure 2.12: Performance of Noiseless and Noisy FAIDs on the $(444, 111)$ QC-LDPC code, with $(d_v = 3, d_c = 12)$ and $(p_v = 0.05, p_c = 0.02)$.

Chapter 3

Design of Min-Sum-based LDPC decoders using imprecise arithmetic

Abstract: *This chapter evaluates the robustness of Low-Density Parity-Check decoders against errors due to imprecise arithmetic. While the use of imprecise arithmetic is motivated by savings in energy, delay and area, it also causes errors during the decoding process. We design imprecise arithmetic operators and investigate their use within several Min-Sum-based decoders. We show that all decoders are able to provide error protection, but most of them suffer a performance penalty compared to the exact arithmetic implementation. Remarkably, the Self-Corrected Min-Sum decoder incurs no performance penalty when imprecise arithmetic is used.*

3.1 Introduction

In order to support the sustainable development of future communication systems, energy efficiency became one of the most important issues to be addressed. Traditionally, the power consumption has been interpreted as the transmit power. This is mainly due to long-range communications, which greatly contributed to the development of information and coding theory, for which the transmit power dominates the total power consumed by the system. Forward error correction (FEC) codes have been key components to design reliable communication systems, while limiting the transmit power to acceptable low levels. Moreover, spectacular advances in the domain of graph-based codes and iterative decoding techniques, made possible the development of new families of error correcting codes ensuring reliable communication at transmit powers closer and closer to the theoretical Shannon limit [22, 26].

Nowadays, there is an increased interest in shorter range communications, from a few meters (femto-cells, wireless sensor networks, etc.) to a few millimeters (inter-chip and on-chip communications). For such applications, it is commonly accepted that the processing power (the power consumed in processing the signals) might represent a substantial fraction of the total power [27, 28]. In this context, power consumption of the FEC decoder module is often a bottleneck, as it can require an important part of the processing power, or even more power than the receiver can supply (*e.g.* in case of low-power systems).

For applications that can trade the accuracy of the circuit for the power consumption, two main approaches are currently investigated. The first approach consists in using aggressive voltage scaling as the basis for reliability and energy tradeoffs. This brings the signal level close to the noise level, which reduces the noise immunity of the circuit and leads to probabilistic computational models [29, 30]. Such models were used to derive low-energy computational platforms for probabilistic algorithms of for applications from the domain of image and video processing, which tolerate probabilistic behavior at the circuit level [31].

The second approach consists in using *imprecise* (also referred to as inexact) circuits, obtained by pruning the exact circuit. This amounts to removing a certain number of logic gates from the circuit (depending on the application's tolerance to errors), which may result in significant savings in energy,

delay and area [32]. Imprecise arithmetic (*e.g.* adders, multipliers) proved to be particularly useful for applications from the domain of image and video processing [33–35].

This chapter investigates the robustness of FEC decoders against errors due to imprecise arithmetic. This is a new paradigm in coding theory, which traditionally assumes that the operations of a FEC decoders are exact, and errors can only be introduced by the channel. While the use of imprecise arithmetic is motivated by savings in energy, delay and area, *the first question we have to answer is whether or not FEC decoders are able to provide reliable error protection when they operate on imprecise hardware.*

We focus on Low-Density Parity-Check (LDPC) codes [13], a class of error correcting codes that feature low complexity message-passing (MP) iterative decoding and can be optimized for a broad class of channels, with performance approaching the theoretical Shannon limit [26]. We evaluate the performance of several LDPC decoders using imprecise arithmetic operators. While the performance penalty due to imprecise arithmetic depends on the considered decoder, we show that the Self-Corrected Min-Sum decoder is inherently robust, and does not suffer any performance penalty due to imprecise arithmetic.

The remainder of the chapter is organized as follows. Related works are discussed in Section 3.2. Section 3.3 gives a brief introduction to LDPC codes and iterative decoding algorithms. Section 3.4 is concerned with the design of imprecise Min-Sum-based decoders and their imprecise arithmetic components. Simulation results are provided in Section 3.5 and Section 3.6 concludes the chapter.

3.2 Related works

Over the last few years, there has been an increased interest in investigating the behavior of LDPC decoders operating on circuits built from probabilistic components. The motivation is two-fold. On the one hand, as mentioned in the Introduction, aggressive voltage scaling can be used to reduce the power consumption of the circuit, leading to probabilistic computational models. On the other hand, it is now widely accepted that emerging nano-electronic devices will be inherently unreliable, due to ineluctable increases in density integration and imperative requirements of low-energy consumption. Recent works studied the performance of Gallager A and Gallager B decoders on faulty hardware [1, 11, 12]. Moreover, in Chapter 1 of this Deliverable, we investigated the performance of Min-Sum-based decoders running on noisy hardware.

The focus of this chapter differs from the above researches in several ways:

1. In this chapter, LDPC decoders are implemented on circuits built from imprecise components. Both imprecise and probabilistic components causes errors during the decoding process, but imprecise behavior is deterministic and can be "tuned" to a desired level of errors.

2. In [1, 11, 12], the authors investigate the asymptotic behavior of the decoder, while we are concerned with finite-length performance.

3. Previous works were concerned with Gallager A and Gallager B decoders, while we are interested in Min-Sum-based decoders, which are widely implemented in real communication systems.

3.3 LDPC codes and iterative decoding

Low-Density Parity-Check (LDPC) codes, have been introduced by Gallager in the early 60's [13], as a class of linear block codes defined by sparse parity-check matrices, suitable for decoding by message-passing (MP) iterative algorithms. Tanner described LDPC codes in terms of sparse bipartite graphs [14], containing two types of nodes: variable-nodes corresponding to coded bits and check-nodes corresponding to parity checks. Equivalently, variable and check nodes correspond respectively to columns and rows of the parity check matrix H, while edges connecting variable and check nodes correspond to the non-zero entries of H.

The graphical representation proposed by Tanner proved to be particularly suitable for MP decoding algorithms. Such a decoding algorithm consists of an exchange of messages along the edges of the bipartite graph. Each message provides an estimation of either the sender or the recipient variable-node (the variable-node incident to the edge), and the exchange of messages takes place in several rounds, or iterations. At each new iteration, new messages are computed in an *extrinsic manner*, meaning that a message that is sent on an edge does not depend on the message just received on the same edge. Consequently, variable-nodes collect more and more information with each new decoding iteration, which gradually improves the estimation of the sent codeword.

The use of Tanner graphs allowed reformulating the probabilistic decoding initially proposed by Gallager in terms of Belief-Propagation (BP) – also referred to as Sum-Product (SP) – a MP algorithm that performs Bayesian inference on graphical models [15, 16]. The BP decoding is known to be optimal for codes defined by cycle-free bipartite graphs, in the sens it outputs the Maximum A Posteriori (MAP) estimates of the coded bits. It is also known to achieve an excellent performance on general sparse bipartite graphs, even if it deviates from the MAP in practical cases (because bipartite graphs associated with practical codes contain cycles). However, the decoding performance is not the only relevant criterion when it comes to practical system implementation, and the BP algorithm is disadvantaged by its complexity, numerical instability, and the fact that it requires the perfect knowledge of the channel parameter (*e.g.* SNR), which may be imprecisely estimated in practical situations.

One way to deal with complexity and numerical instability issues is to simplify the computation of messages exchanged within the BP decoding. The most complex step of the BP decoding is the computation of check-to-variable node messages, which makes use of computationally intensive hyperbolic tangent functions. The Min-Sum (MS) algorithm is aimed at reducing the computational complexity of the BP, by using max-log approximations of the parity check to coded bit messages [17, 19]. The only computations required by the MS decoding are additions and comparisons, which solves the complexity and numerical instability problems. The performance of the MS decoding is also known to be independent of the knowledge of the channel parameter, for most of the usual channel models.

However, the max-log approximation used in the MS decoding leads to a performance degradation with respect to the BP decoding. Several "correction" methods were proposed in the literature in order to mitigate this performance degradation [19, 24, 36–39]. These decoding algorithms are referred to as MS-based algorithms: they are improved versions of the MS algorithm, with only a very limited increase of complexity.

In this chapter we investigate the use of imprecise arithmetic circuits for the following decoders: MS decoder, Normalized-MS (NMS) decoder [36], Offset-MS decoder [36], Self-Corrected-MS (SCMS) decoder [24]. They will be described in the following paragraphs.

3.3.1 Notation

The following notation will be used throughout the chapter:

- H, Tanner graph of an LDPC code,
- N, number of variable-nodes,
- M, number of check-nodes,
- d_n, degree of the variable-node n,
- d_m, degree of the check-node m,
- $n \in \{1, 2, ..., N\}$, a variable node of H,
- $m \in \{1, 2, ..., M\}$, a check node of H,
- $H(n)$, set of check-nodes connected to the variable-node n,

- $H(m)$, set of variable-nodes connected to the check-node m,
- γ_n, a priori log-likelihood ratio (LLR) of variable-node n,
- $\tilde{\gamma}_n$, a posteriori LLR of variable-node n,
- $\alpha_{m,n}$, variable-to-check message sent from n to m,
- $\beta_{m,n}$, check-to-variable message sent from m to n.

3.3.2 Min-Sum decoding

Assume that a codeword $(x_n)_{n=1,\dots,N}$ is sent over a memoryless noisy channel, and let $(y_n)_{n=1,\dots,N}$ denote the received word. The MS decoding algorithm works as follows.
Initialization

- A priori LLRs
$$\gamma_n = \log \frac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}$$
- Variable-to-check messages initialization
$$\alpha_{m,n} = \gamma_n$$

Iterations

- Check-node processing
$$\beta_{m,n} = \left(\prod_{n' \in H(m) \setminus n} \operatorname{sgn}(\alpha_{m,n'}) \right) \min_{n' \in H(m) \setminus n} \left(|\alpha_{m,n'}| \right)$$
- A posteriori LLRs
$$\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$$
- Variable-node processing
$$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$$

In the above description, γ_n and $\tilde{\gamma}_n$ are computed for each variable-node n, while messages $\alpha_{m,n}$ and $\beta_{m,n}$ are computed for each graph edge (m, n). Finally, at each iteration, coded bit estimates are computed by $\hat{x}_n = (1 - \operatorname{sgn}(\tilde{\gamma}_n))/2$, and decoding stops when whether $(\hat{x}_n)_{n=1,\dots,N}$ is a codeword or a maximum number of iterations has been reached.

3.3.3 Normalized Min-Sum decoding

As discussed in Introduction, MS decoding can be seen as a low-complex approximate version of the BP decoding. This approximation is known to result in an overestimation of check-to-variable messages. The aim of the NMS decoding is to compensate this overestimation, by introducing a normalization (scaling) factor $\lambda \in]0, 1[$ within the check-node processing step. Hence, all the other decoding steps remain unchanged, and only the check-node processing step is modified as follows:

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

3.3.4 Offset Min-Sum decoding

Similar to the the NMS decoding, the OMS attempts to compensate the overestimation of the check-to-variable messages. This time an offset factor $\delta > 0$ is used, and the check-node processing step is modified as follows:

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

3.3.5 Self-Corrected Min Sum decoding

The SCMS addresses the overestimation issue at the variable-node processing side of the algorithm. The rationale behind the SCMS is that the overestimation of check-to-variable messages is not critical, unless any given variable-to-check message is updated to map a different bit state. In the log likelihood ratio domain, this corresponds to a sign change. In the SCMS, any variable-to-check message that would experience a sign change is erased (that is, it is set to zero). Hence, check-node processing step is modified as shown below, while all the decoding steps are the same as for MS decoding.

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

So, the variable-to-check message is first stored in a temporary value $\alpha_{m,n}^{\text{tmp}}$, and its sign is compared against the sign of the variable-to-check messages from the previous iteration (stored in $\alpha_{m,n}$). If a sign change is detected and $\alpha_{m,n} \neq 0$, the value of the new variable-to-check message is set to zero. Otherwise, the value the new variable-to-check message is set (as it would usually be) to $\alpha_{m,n}^{\text{tmp}}$.

In [24], the author pointed out that a variable-to-check message changes its sign between two consecutive iterations if and only if its computation tree [40] contains unreliable information. As a consequence, it has been shown that the SCMS decoding behaves as the MS decoding on a computation tree that has been pruned of its unreliable branches. The SCMS decoding ability to detect unreliable messages will prove to be particularly useful when the decoder is implemented on imprecise circuits.

3.4 Imprecise Min-Sum-based decoders

In order to evaluate the impact of the imprecise arithmetic components on the performances of MS-based LDPC decoders, all the messages in the decoders must be quantized. Since the a posteriori LLR is computed as the sum of the a priori LLR and the incoming check-to-variable messages, more quantization bits have to be used to represent its value. Hence, Q bits are used for the quantization of the a priori LLRs (γ_n), as well as exchanged messages ($\alpha_{m,n}$, $\beta_{m,n}$), while $Q+1$ bits are used for the quantization of the a posteriori LLRs ($\tilde{\gamma}_n$). The imprecise arithmetic components used within the MS-based LDPC decoders are:

- Q-bit comparators, used for the implementation of the check-node processing step.
- $(Q+2)$-bit adders used for the implementation of the a posteriori LLRs update and the variable-node processing step.

Note that an extra bit is used for the adder in order to detect the overflow. At the entry of the adder, the $(Q+1)$-bit input operands get an extra bit by repeating their most significant (sign) bit. Throughout this chapter, we shall use $Q = 6$. The design of imprecise 6-bit comparator and 8-bit adder is addressed in the following sections.

3.4.1 Kogge-Stone Adder

Consider two integers A and B, and let S denote their sum and C the corresponding carry. The simplest type of adder that can be used to compute S is the ripple carry adder which computes successively the sum and the carry for each bit starting from the least significant bit (LSB). However, this type of adder is very slow and the Carry Lookahead adders have been developed to reduce the computation time. In order to compute the sum S, they introduce two binary parameters G and P. For each bit position i, G_i is 1 if A_i and B_i are both equal to 1. The bit position i is then said to generate a carry. If only one of A_i or B_i is equal to 1, P_i is equal to 1 and the bit position i is said to propagate a carry.

All carry-lookahead adders (CLA) perform binary addition in three steps: precomputation, prefix and postcomputation. For all adder architectures, the precomputation and postcomputation steps are similar: P_i and G_i are computed for each position bit in the precomputation, while the sum S is computed in the postcomputation step. However, the prefix step is different for each adder architecture and those architectures can be classified into three categories: serial-prefix, group-prefix and parallel-prefix. Fig. 3.1 represents a 4-bits parallel-prefix CLA architecture.

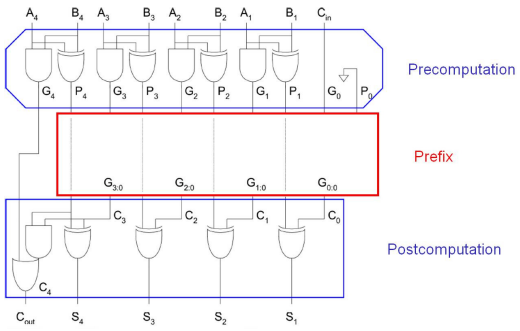


Figure 3.1: 4-bit parallel-prefix CLA architecture

Kogge-Stone adder is a parallel-prefix CLA architecture [41], which generates the carry in a $O(\log n)$ time. It is one of the fastest existing adder architectures and provides the highest performances [32]. The Prefix Diagram of a 8-bit Kogge-Stone adder is represented in Fig. 3.2. As shown in the top of the figure, a grey cell corresponds to one OR and one AND gate, while a black cell corresponds to one OR and two AND gates.

Kogge-Stone adders have been selected as a basis for the implementation of imprecise adders. Imprecise comparators are also derived from Kogee-Stone adders, but only the path that computes the most significant (sign) bit will be used.

3.4.2 The imprecise adder

In order to design inexact arithmetic components, we suppress several logic gates in all the adders and in the comparators of the circuit. However, the objective is to control the errors introduced by pruning the circuit, hence some constraints might be defined according to the desired level of errors. For example, the magnitude of the errors can be bounded or the paths of the circuit with lower probability of being used can be deleted [32]. In this work the main constraints are as follows. First, we require that the changes to the exact circuit do not impact the most significant (sign) bit. Protecting the sign of each addition from errors reduces the impact of the inexact circuit. Secondly, we require that for operands x and y, such that the value of the exact addition $x+y$ is small, the error made by the imprecise adder when computing $x+y$ must also be small (close to 0). The reason is that we do not want to give any extra confidence to the decoder, when its degree of confidence should be low. Note however that if the exact value of $x+y$ is relatively large, the error made by the imprecise adder when computing $x+y$ may also be large (but the sign is always correct).

In order to meet the above requirements, the path that computes the sign bit is kept unchanged and logic gates are suppressed starting from the least significant bit (LSB)to the most significant bit

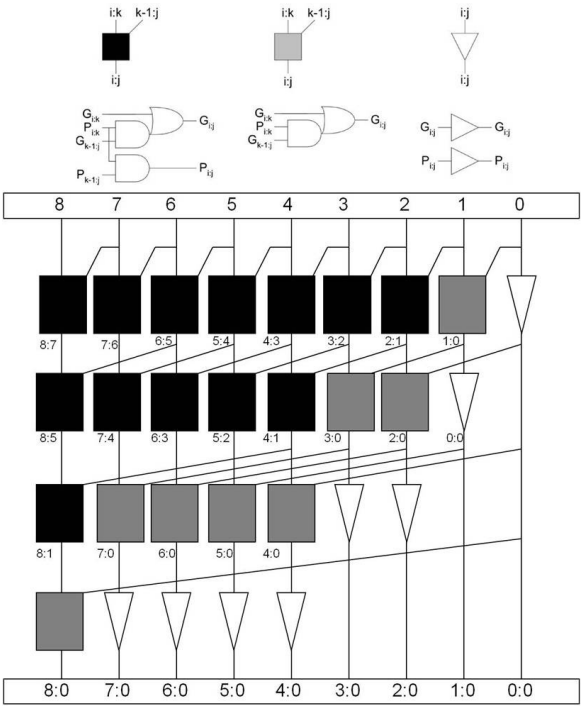


Figure 3.2: 8 bits Kogge-Stone diagram

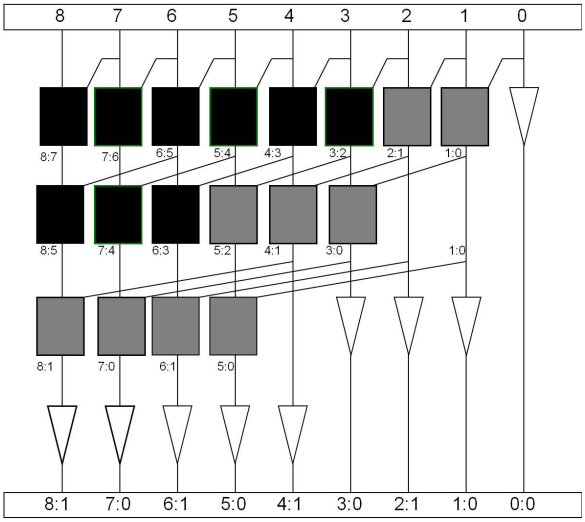


Figure 3.3: Imprecise adder

(MSB). The adder is simulated to check if all the requirements are met or not. The procedure is repeated until all the requirements are met and the suppression of any of the remainder logic gates will fail to comply with the requirements. The designed imprecise adder is shown in Fig. 3.3, and it has the following characteristics:

- 3 grey cells have been deleted.
- 4 black cells have been replaced by 4 grey cells.
- 288 erroneous outputs are generated out of 16129 possible additions.
- The sign bit of any output is always correct.
- For a given x, the adder always correctly computes $x - x$.

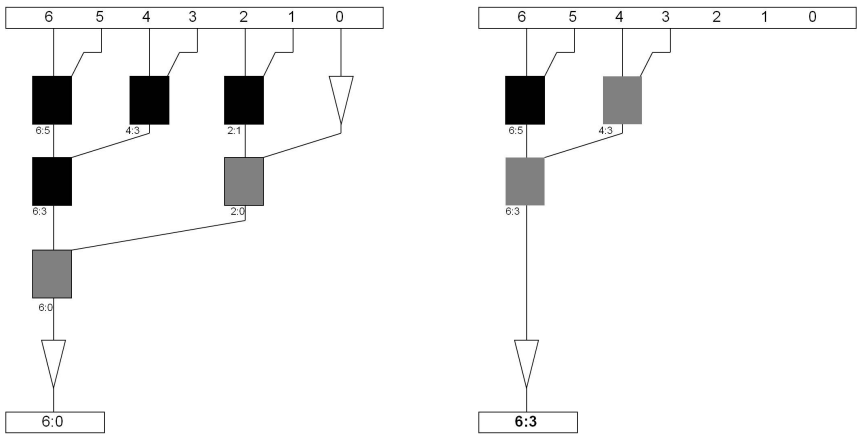
Some examples of imprecise additions are given in Table 3.1.

Table 3.1: Example of inexact additions

A	-29	-25	-21	3	7	11	-31	-27	-23	-29
B	-27	-23	-19	5	9	13	7	11	15	13
Output	-24	-16	-8	40	48	56	-56	-48	-40	-48

3.4.3 The imprecise comparator

Comparators are used to implement the check-node processing step. The sign path of a 6-bit Kogge-Stone adder is used to design the imprecise comparator. For any two operands x and y, we allowed the output of the imprecise comparator to be in error only if x and y have relatively close values. As for the imprecise adder, logic gates are suppressed starting from the LSB to the MSB until the suppression of any of the remainder logic gates will fail to comply with the requirements. The designed imprecise comparator is shown in Fig. 3.4(b), and it has the following characteristics:



(a) Ideal 6 bits comparator (b) Imprecise Comparator

Figure 3.4: 6 bits comparator architecture

- 1 black cell and 2 grey cells have been deleted.
- 2 black cells have been replaced by 2 grey cells.
- 96 erroneous outputs are generated out of 961 possible comparisons.
- Errors happen only when $\mid x - y \mid < 2^2$.

Some examples of imprecise comparisons are given in Table 3.2. The comparator's output is 1 iff $x <_{\text{imprecise}} y$.

Table 3.2: Example of inexact comparisons

A	7	-29	-21	-9	-5	-1	-5	2	3	6
B	-7	-31	-23	-11	-7	-3	-6	1	2	5
Output	1	1	1	1	1	1	1	1	1	1

3.4.4 Implementation of imprecise Min-Sum-based decoders

Let $\mathbf{q}$ denote the Q-bit quantizer used at the decoder input, and $\mathbf{s}$ denote the Q-bit saturation operation ($\mathbf{s}(x) = 1 - 2^{(Q-1)}$ if $x < 1 - 2^{(Q-1)}$, $\mathbf{s}(x) = 2^{(Q-1)} - 1$ if $x > 2^{(Q-1)} - 1$, and $\mathbf{s}(x) = x$ otherwise). We also denote by $\mathbf{m}_{\text{imp}}$ the imprecise minimum computation, and by $\mathbf{a}_{\text{imp}}$ the imprecise adder. The Imprecise-MS decoder is implemented as follows:

Initialization

- A priori LLRs
$$\gamma_n = \mathbf{q}\left(\log \frac{\Pr(x_n = 0 \mid y_n)}{\Pr(x_n = 1 \mid y_n)}\right)$$
- Variable-to-check messages initialization
$$\alpha_{m,n} = \gamma_n$$

Iterations

- Check-node processing
Let $H(m)\backslash\{n\} = \{n_1, \ldots, n_{d_m - 1}\}$
$$\beta_{m,n} = \left(\prod_{i=1,\ldots,d_c-1} \operatorname{sgn}(\alpha_{m,n_i})\right) \cdot \mathbf{m}_{\text{imp}}(\cdots(\mathbf{m}_{\text{imp}}(|\alpha_{m,n_1}|, |\alpha_{m,n_2}|), \cdots, |\alpha_{m,n_{d_m-1}}|)$$

- A posteriori LLRs
  Let $H(n) = \{m_1, \ldots, m_{d_n-1}\}$

$$\tilde{\gamma}_n = \mathbf{a}_{\text{imp}}(\cdots(\mathbf{a}_{\text{imp}}(\gamma_n, \beta_{m_1,n}), \cdots, \beta_{m_{d_n-1},n})$$

- Variable-node processing

$$\alpha_{m,n} = \mathbf{s}\left(\mathbf{a}_{\text{imp}}(\tilde{\gamma}_n, -\beta_{m,n})\right)$$

We note that in order to implement the check-node processing step it is actually sufficient to compute the first and the second minima of $|\alpha_{m,n_i}|$, for all $i = 1, \ldots, d_m$.

For the OMS decoder (Section 3.3.4), the offset factor δ is subtracted from $|\beta_{m,n}|$ by using the imprecise adder $\mathbf{a}_{\text{imp}}$. We use $\delta = 1$, which is the optimal value for the exact OMS decoding.

For the NMS decoder (Section 3.3.3), we use $\lambda = 0.75$ and the multiplication between λ and $\beta_{m,n}$ is implemented as the sum $0.75 \cdot \beta_{m,n} = \mathbf{a}_{\text{imp}}(\beta_{m,n}, \beta_{m,n}/2)$, where the exact value of $\beta_{m,n}/2$ is obtained by a right-shift of bits of $\beta_{m,n}$.

Finally, we note that the SCMS decoder does not use any extra arithmetic operation compared to the MS decoder.

3.5 Simulation Results

The performance of Min-Sum-based decoders has been evaluated for two different codes.

- The first code is a short $(504, 252)$ LDPC code, constructed by Mackay and available online at [23]. It is a regular code, with all variable-nodes of degree 3 and all check-nodes of degree 6.

- The second code is a longer $(2304, 1152)$ and irregular quasi-cyclic LDPC code, specified by the IEEE 802.16e (WiMAX) standard [42].

Both codes have been simulated over the Additive White Gaussian Noise (AWGN) channel with Quadrature Phase-Shift Keying (QPSK) modulation. The above MS-based decoders have been simulated for both exact and imprecise arithmetic components, and the maximum number of decoding iterations was fixed to 100. Imprecise arithmetic components have been simulated through lookup tables.

3.5.1 Decoders' performance

The Frame Error Rate (FER) performance of MacKay and WiMAX LDPC codes are shown respectively in Fig. 3.5 and Fig. 3.6. The FER curves of the MS, NMS, OMS and SCMS decoders are plotted respectively in red, green, blue and black. In addition, for each decoder, the dashed curve (empty markers) was obtained by using exact arithmetic components, and the solid curve (full markers) was obtained by using imprecise arithmetic components.

Analysis of results for the Mackay code

At FER $= 10^{-4}$ the use of the imprecise arithmetic results in a loss of about 1.3 dB for the MS decoder and 0.6 dB for the OMS. The imprecise SCMS provides almost the same performance as the exact SCMS, and even outperforms it in the error floor region, while the imprecise NMS outperforms the exact NMS in the waterfall region. Both SCMS and NMS can be considered robust to imprecise arithmetic components.

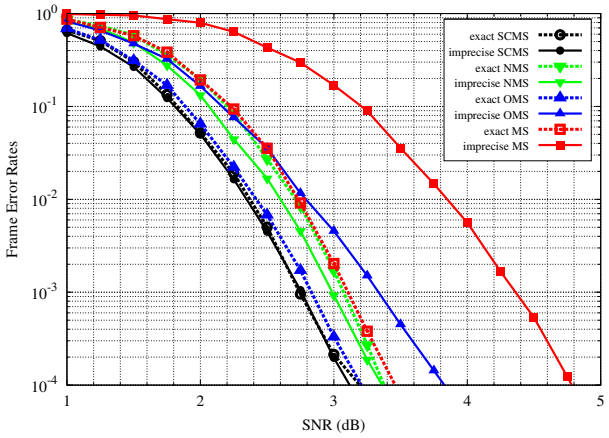


Figure 3.5: FER for the (504, 252) regular code

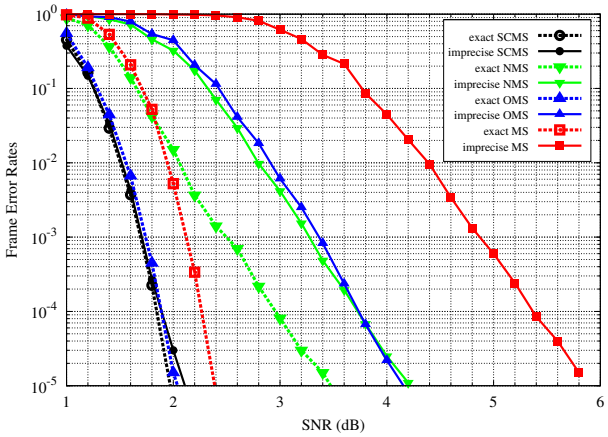


Figure 3.6: FER for the IEEE 802.16e code

Analysis of results for the WiMAX code

At FER $= 10^{-4}$ the use of the imprecise arithmetic results in a loss of about 3.3 dB for the MS algorithm, 1.8 dB for the OMS algorithm, and only 0.8 dB for the NMS algorithm. As previously, the performance of the imprecise SCMS is almost identical to the exact SCMS.

The excellent performance of the SCMS decoder under imprecise arithmetic settings is explained by its inherent ability to detect unreliable messages during the iterative decoding process. These results bring empirical evidence that the SCMS decoder is able to provide efficient error protection even if it operates on imprecise hardware, which represents the main contribution of this work.

3.5.2 Complexity analysis

The exact evaluation of the savings in energy, delay and area is out the scope of this work (but will be addressed in future works). However, we include a complexity analysis of the different decoders, where the complexity is expressed in terms of the number of logic gates of the circuit.

Let C_a be the relative complexity of the imprecise adder with respect to the exact adder, and C_c be the relative complexity of the imprecise comparator with respect to the exact comparator. They are defined as follows:

$$C_a = \frac{\text{number of logic gates of imprecise adder}}{\text{number of logic gates of exact adder}}$$

$$C_c = \frac{\text{number of logic gates of imprecise comparator}}{\text{number of logic gates of exact comparator}}$$

For the imprecise adder and comparator designed in Section 3.4, we have $C_a = 0.86$ and $C_c = 0.57$.

In order to evaluate the gain in complexity when imprecise arithmetic is used, we consider the ratio between the number of arithmetic operations (additions and comparisons) for imprecise and exact arithmetic. Moreover, in case of imprecise arithmetic, the number of additions is weighted by C_a and the number of comparisons in weighted by C_c. Hence, we obtain:

$$\text{gain} = \frac{2(C_a + C_c)d_v - 3C_c(1-r)}{4d_v - 3(1-r)},$$

where d_v is the average variable-node degree and r is the coding rate. Fig. 3.7 shows the evolution of the gain when $d_v = 3$ and the rate r varies. As it can be seen, the gain decreases slowly from 0.715 to 0.71, as the rate increases from 0 to 1.

The average decoding complexity per codeword of the exact MS, the exact SCMS, and the imprecise SCMS has also been evaluated. This complexity is defined as the average number of arithmetic operations (additions and comparisons) required to decode 1 codeword. Again, in case of imprecise arithmetic, the number of additions is weighted by C_a and the number of comparisons in weighted

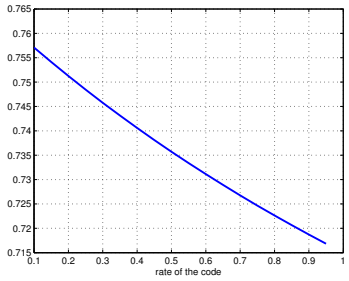


Figure 3.7: Gain in complexity

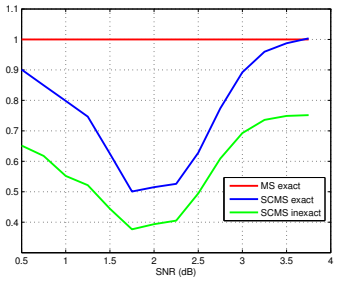


Figure 3.8: Relative complexity for the (504,252) regular code

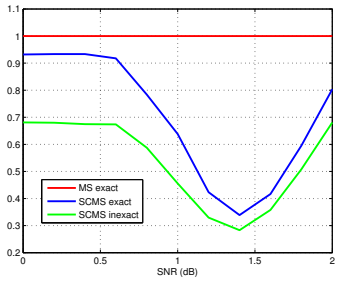


Figure 3.9: Relative complexity for the IEEE 802.16e code

by C_c. Fig. 3.8 and Fig. 3.9 plot the relative complexities of these decoders with respect to the exact MS (which is used as the reference). The complexity of the exact SCMS is much lower than those of the exact MS in the waterfall region, because SCMS needs in average a smaller number of decoding iterations. The imprecise SCMS exhibits the lowest complexity and the difference between its complexity and those of the exact SCMS complexity is due to the use of imprecise components.

3.6 Conclusion

In this work we investigated the performance of several Min-Sum-based decoders on devices with imprecise arithmetic circuits. Imprecise adders and comparators have been specially designed for this purpose, by pruning the exact circuits. The pruning operation has been constrained to meet specific requirements, such as to avoid a number of undesirable errors.

Simulation results have shown that MS, NMS, and OMS decoders manage to provide error protection, but the imprecise arithmetic circuits significantly degrade their performance. On the contrary, the imprecise SCMS proved to be robust to imprecise arithmetic circuits, due to its ability to detect unreliable messages during the decoding process. Hence, this work demonstrated that the SCMS decoder provides *efficient* error protection on devices with imprecise arithmetic circuits.

In addition, the complexity of the decoders has been evaluated in terms of number of logic gates of the circuits. For a code rate of 1/2 the use of imprecise arithmetic circuits yields a complexity decrease of about 27%. Moreover, the imprecise SCMS provides a complexity reduction between 30% and 60% with respected to the exact MS decoder.

Chapter 4

General Conclusion and Next Steps

As a general conclusion of the activities in WP3 for this first year of i-RISC project, we have made considerable progress with respect to the state-of-the-art related to the design and analysis of faulty iterative decoders, with a special focus on the quantized Min-Sum decoder, the FAID decoders, and the self-corrected Min-Sum decoder.

Compared to the expected milestone at this stage, which was *"Faults tolerant LDPC decoders, first release"*, we have made progress beyond what was expected and planned after the first year of the project. Not only we are able to provide to the other WPs and all the i-RISC partners efficient message passing decoders, which seem to be robust to faulty-hardware, but we also developped some new analysis and tools for the theoretical understanding of the faulty-decoders behaviors, with a predictive analysis which opens the avenue to the design of decoders which are specifically optimized for fault-tolerance.

On top of the partial conclusions of each chapter in this deliverable, the summary of our main contributions follows:

1. although we have considered very limited error models, which are memoryless and symmetric, and far from being realistic, our models are broader and more general than what is usually proposed in the litterature. The added complexity comes from the fact that we consider practical decoders as a base of our analysis. The decoders are supposed to be implemented with messages in finite precision, while the SoA usually propose works based on either hard decision decoders (bit-flipping and Gallager-B decoders), or non-realistic infinite precision Belief Propagation.

2. we have progressed toward a deep understanding of the dynamics of faulty decoders, with in particular the new concept of functional thresholds and functional region, to replace the existing useful region. We have also linked this new threshold to the observations made from finite length Monte-Carlo simulations,

3. we have identification that the noise can help iterative decoder to avoid unwanted fixed points. This was the case for example of the DE fixed points for the noisy Min-Sum decoder, and most probably the case of finite length fixed points located on trapping sets, from our first sets of simulations,

4. we provided evidence of the compliance between the asymptotical noisy DE analysis and the finite length performance of noisy decoders, so that the noisy DE analysis can be seen and used as a predictive tool,

5. we have shown first evidence that using memory in faulty decoders can be of great help for the robustness to transient errors, in particular, the SC-MS has been shown to tolerate much more noise coming from the hardware, both in the probabilistic model, and the inexact arithmetic model.

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

1. continue to develop the theoretical analysis of faulty decoders with analytical tools, and try to incorporate more and more complex noise models,
2. explore the systematic use of memory (from one iteration to another) in the faulty decoders to even improve their robustness to transient errors,
3. propose new direction of decoder design (specifically under the FAID framework), to maximize fault-tolerance,
4. apply fault-tolerant decoders to design fault-tolerant encoders.

Bibliography

[1] L. R. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4427–4444, 2011.

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

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

[4] M. G. Taylor, "Reliable computation in computing systems designed from unreliable components," *Bell System Technical Journal*, vol. 47, pp. 2339–2366, 1968.

[5] A. V. Kuznetsov, "Information storage in a memory assembled from unreliable components," *Problemy Peredachi Informatsii*, vol. 9, no. 3, pp. 100–114, 1973.

[6] 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.

[7] C. Winstead and S. Howard, "A probabilistic LDPC-coded fault compensation technique for reliable nanoscale computing," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 56, no. 6, pp. 484–488, 2009.

[8] Y. Tang, C. Winstead, E. Boutillon, C. Jego, and M. Jezequel, "An ldpc decoding method for fault-tolerant digital logic," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2012, pp. 3025–3028.

[9] A. M. Hussien, M. S. Khairy, A. Khajeh, A. M. Eltawil, and F. J. Kurdahi, "A class of low power error compensation iterative decoders," in *IEEE Global Telecom. Conf. (GLOBECOM)*, 2011, pp. 1–6.

[10] S. Yazdi, H. Cho, Y. Sun, S. Mitra, and L. Dolecek, "Probabilistic analysis of Gallager B faulty decoder," in *IEEE Int. Conf. on Communications (ICC)*, 2012, pp. 7019–7023.

[11] S. Yazdi, C. Huang, and L. Dolecek, "Optimal design of a Gallager B noisy decoder for irregular LDPC codes," *IEEE Comm. Letters*, vol. 16, no. 12, pp. 2052–2055, 2012.

[12] S. Yazdi, H. Cho, and L. Dolecek, "Gallager b decoder on noisy hardware," *IEEE Trans. on Comm.*, vol. 66, no. 5, pp. 1660–1673, 2013.

[13] R. G. Gallager, "Low density parity check codes," MIT Press, Cambridge, 1963, research Monograph series.

[14] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[15] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Proc. of the 2nd National Conference on Artificial Intelligence (AAAI-82)*, 1982, pp. 133–136.

[16] ——, *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers, 1988.

[17] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Communications*, vol. 47, no. 5, pp. 673–680, 1999.

[18] S. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[19] E. Eleftheriou, T. Mittelholzer, and A. Dholakia, "Reduced-complexity decoding algorithm for low-density parity-check codes," *IET Electronics Letters*, vol. 37, no. 2, pp. 102–104, 2001.

[20] R. L. Dobrushin and S. Ortyukov, "Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements," *Problemy Peredachi Informatsii*, vol. 13, no. 1, pp. 82–89, 1977.

[21] A. Amaricai *et al.*, "Circuit level fault models for sub-powered CMOS circuits for uncorrelated and correlated errors," FP7 / FET OPEN / 309129, i-RISC project, Deliverable D2.1, January 2014.

[22] T. J. Richardson and R. L. 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, 2001.

[23] D. J. MacKay. Encyclopedia of sparse graph codes. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[24] V. Savin, "Self-corrected min-sum decoding of LDPC codes," in *Proc. of IEEE Int. Symp. on Information Theory (ISIT)*, 2008, pp. 146–150.

[25] 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.

[26] 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.

[27] P. Grover and A. Sahai, "Green codes: Energy-efficient short-range communication," in *IEEE Int. Symp. on Inf. Theory (ISIT)*, 2008, pp. 1178–1182.

[28] A. Sahai and P. Grover, "The price of certainty: "waterslide curves" and the gap to capacity," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-1, Jan 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-1.html

[29] K. V. Palem, "Energy aware computing through probabilistic switching: A study of limits," *IEEE Trans. on Computers*, vol. 54, no. 9, pp. 1123–1137, 2005.

[30] L. N. B. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, and K. V. Palem, "Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation," in *Proc. of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems.* ACM, 2008, pp. 187–196.

[31] B. E. Akgul, L. N. Chakrapani, P. Korkmaz, and K. V. Palem, "Probabilistic CMOS technology: a survey and future directions," in *Proc. of IFIP/IEEE International Conference on VLSI*, 2006, pp. 1–6.

[32] A. Lingamneni, C. Enz, J. L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2011, pp. 1–6.

[33] I. Chong and A. Ortega, "Hardware testing for error tolerant multimedia compression based on linear transforms," in *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 523–531.

[34] H. Chung and A. Ortega, "Analysis and testing for error tolerant motion estimation," in *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 514–522.

[35] N. Zhu, W. Goh, W. Zhang, K. Yeo, and Z. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1225–1229, 2010.

[36] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low density parity check codes," *IEEE Trans. on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

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

[38] J. Chen, R. Tanner, C. Jones, and Y. Li, "Improved min-sum decoding algorithms for irregular LDPC codes," in *IEEE Int. Symp. on Inf. Theory (ISIT)*, 2005, pp. 449–453.

[39] J. Zhang, M. Fossorier, and D. Gu, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Communications Letters*, vol. 10, no. 3, pp. 180–182, 2006.

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

[41] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Tran. on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[42] IEEE-802.16e, "Physical and medium access control layers for combined fixe and mobile operation in licensed bands," 2005, amendment to Air Interface for Fixed Broadband Wireless Access Systems.

[43] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *Electronics Letters*, vol. 47, no. 16, pp. 919-921, Aug. 2011.

[44] J. Chen, *et. al.*, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. on Commun.*, vol. 53, pp. 1288-1299, Aug. 2005.

[45] S.K. Planjery, D. Declercq, L. Danjean and B. Vasic, "Finite Alphabet Iterative Decoders, Part I: Decoding Beyond Belief Propagation on the BSC", IEEE Trans. on Commun., vol. 61, no. 10, pp. 4033-4045, Oct. 2013.

[46] D. Declercq, B. Vasic, S.K. Planjery and E. Li, "Finite Alphabet Iterative Decoders, Part II: Towards Guaranteed Error Correction of LDPC Codes via Iterative Decoder Diversity", IEEE Trans. on Commun., vol. 61, no. 10, pp. 4046-4057, Oct. 2013.

[47] T. Richardson, "Error floors of LDPC codes," *Proc. 41st Annual Allerton Conf on Communications Control and Comuting*, 2003.

[48] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," *Proc. 47th Annual Allerton Conf. on Commun., Control, and Computing*, Sep. 2009.

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

[50] S. K. Planjery, D. Declercq, S. K. Chilappagari, and B. Vasic, "Multilevel decoders surpassing belief propagation on the binary symmetric channel," *Preprint. [Online]. Available: http://arxiv.org/abs/1001.3421*, 2010.

[51] G. Kuperberg, "Symmetries of plane partitions and the permanent - determinant method," *J. Comb. Theory, Ser. A*, pp. 115–151, 1994.

[52] F. Cai, X. Zhang, D. Declercq, S. Planjery and B. Vasic, "Finite Alphabet Iterative Decoders for LDPC codes: Optimization, Architecture and Analysis", submitted in IEEE Trans. Circuits and Systems (Series-I), 2013.