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

D4.3

Assessment of Memory Architecture Tolerance to Correlated Errors and On-Chip Reliable Data Transport

Editor:	Goran Đorđević (ELFAK)
Deliverable nature:	Public
Due date:	October 31, 2015
Delivery date:	October 31, 2015
Version:	2.0
Date of current version:	April 15, 2016
Total number of pages:	104
Reviewed by:	i-RISC members
Keywords:	LDPC Codes, Taylor-Kuznetsov Memory, Date-Dependent Failures, Bit-Flipping, Timing Errors, Expander Codes, Gallager-B Decoder, Haar Transform, Reliable Interconnect, Polyhedral Memory.

Abstract: This deliverable presents an overview of the activities carried out during Month 25 to Month 33 (M25-M33) project period within the Work Package 4 (WP4) framework. We mainly investigate low complexity hard-decision decoders and corresponding memory architectures under uncorrelated and data-dependent gate failures and codec augmented interconnects that can enable energy effective reliable data transport. Related to faulty Gallager B decoder we demonstrate that randomness added through hardware unreliability can improve the overall decoder performance, and we develop theoretical framework that allows for iterative decoding process dynamics taking. Related to the Taylor-Kuznetsov memory architecture we: (i) provide a novel reliability analysis analytical method able to predict the reliability of the memory architecture for a fixed refresh time, codec parameters and (ii) improve the Deliverable D.4.2 results regarding the conditions in which we can guarantee that a number of worst-case component failures can be tolerated by the fit-flipping decoder and corresponding memory architecture, under data-dependent gate failures. We also propose a fault resilient 3D polyhedral memory general architecture and framework, which makes use of an LDPC based scrubbing approach to perform user transparent memory maintenance. Our experiments indicate that for the same redundancy requirements our approach outperforms the state of the art Hamming code based counterpart in terms of error correction capability and has very limited implications on memory performance and availability. Finally, we introduce codec based techniques to improve data transport, i.e., diminish the delay and energy consumption and increase fault resilience, within computation platforms.

List of Authors

Participant	**Author**
CEA	Valentin Savin (valetin.savin@cea.fr)
ENSEA	Elsa Dupraz (elsa.dupraz@ensea.fr)
	David Declercq (declercq@ensea.fr)
TU-Delft	Nicoleta Cucu Laurenciu (N.CucuLaurenciu@tudelft.nl)
	Mihai Lefter (M.Lefter@tudelft.nl)
	Thomas Marconi (T.Marconi@tudelft.nl)
	Sorin Cotofana (S.D.Cotofana@tudelft.nl)
ELFAK	Goran Đorđević (goran.t.djordjevic@elfak.ni.ac.rs)
	Bane Vasić (vasic@email.arizona.edu)
	Predrag Ivaniš (predrag.ivanis@etf.rs)
	Srđan Brkić (brka05@gmail.com)
	Omran Al Rasheed (omrano84@hotmail.com)
	Aleksandra Cvetković (aleksandra.cvetkovic@elfak.ni.ac.rs)

Contents

List of Dissemination Activities

Published papers

[P1] S. Brkic, O. Al Rasheed, P. Ivanis, B. Vasic, "On Fault Tolerance of the Gallager B decoder under data-dependent gate failures," *IEEE Communication Letters*, vol. 19, no. 8, pp. 1299–1302, Aug., 2015.

[P2] B. Vasic, P. Ivanis, S. Brkic, "Low Complexity Memory Architectures Based on LDPC Codes: Benefits and Disadvantages," *In Proc. of 12th International Conference on Advanced Technologies Systems and Services in Telecommunications (TELSIKS 2015)*, Nis, Serbia, Oct. 2015. (invited paper)

[P3] S. Brkic, P. Ivanis, B. Vasic, "Reliability of Memories Built from Unreliable Components under Data-Dependent Gate Failures," *IEEE Communication Letters* (accepted for publication)

Submitted papers

[P4] S. Brkic, P. Ivanis, B. Vasic, "Majority logic decoding under data-dependent gate failures," *IEEE Transactions on Information Theory* (submitted)

Workshop presentations

[P5] B. Vasic, P. Ivanis, S. Brkic, V. Ravanmehr "Fault-Resilient Decoders and Memories made of Unreliable Components," *In Proc. of Information Theory and Applications Workshop*, San Diego, USA, Feb. 2015

List of Figures

List of Tables

List of Abbreviations

APP	*A Posteriori* Probability
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BF	Bit-Flipping
BP	Belief-Propagation
BSC	Binary Symmetric Channel
CN	Check Node
CNU	Check Node Update
CPE	Codeword Prediction Encoder
DE	Density Evolution
DED	Double Error Detection
DRAM	Dynamic Random Access Memory
DSM	Deep SubMicron
ECC	Error Correcting Code
ELFAK	Elektronski Fakultet Nis
FAID	Finite Alphabet Iterative Decoder
FER	Frame Error Rate
GDBF	Gradient-Descent Bit-Flipping
IC	Integrated Circuit
IVRG	Intrinsic-Value Random Generator
LDPC	Low Density Parity Check
LDGM	Low Density Generator Matrix
LFSR	Linear Feedback Shift Register
LFSR-RG	Linear Feedback Shift Register Random Generator
LLR	Log-Likelihood Ratio
LS	Latin Squares
LUT	Look Up Table
MAJ	MAJority-logic
MS	Min-Sum
O-S-MAJ	One-Step MAJority logic
NoC	Network on Chip
PEG	Progressive Edge Growth
PGDBF	Probabilistic Gradient-Descent Bit-Flipping

PN	Parity check Node
QC	Quasi-Cyclic
RG	Random Generator
SEC	Single Error Correction
SECDED	Single Error Correction Double Error Detection
SoC	System-on-Chip
SP	Sign-Preserving
TSV	Through-Silicon-Via
TUD	Technische Universiteit Delft
UPT	Universitatea Politehnica Timisoara
UCC	University College Cork
VN	Variable Node
VNU	Variable Node Update
WER	Word Error Rate
WBF	Weighted Bit-Flipping
XOR	eXclusive-OR

Introduction

This deliverable addresses the problem of the reliable storage of digital information on a chip built from unreliable components. It represents summation of the research, done during the period Month 25 to Month 33 (M25-M33) of i-RISC project, conducted in order to complete tasks described in Work Package 4 (WP4) of the project proposal. The main objectives of WP4 include analysis of the state-of-the-art memory architectures under more realistic hardware failure modeling, proposing novel memory architectures designed for tolerating data-dependent gate failures and designing the codes that ensure reliable intra/inter-chip bus connections. The WP4 is divided into five complementary tasks, as represented in Fig. 1, which are in the scope of our research.

In this WP we mostly investigate known hard decision LDPC decoders and propose modifications that increase robustness of such decoders when decoding operations are not perfectly reliable. Successful completion of all tasks in this WP will give us theoretical guidelines for building LDPC codes based low-complexity fault-tolerant memories and ensuring reliable date transport on unreliable hardware. Based on techniques developed in this WP, the practical implementation of a simple processor core is anticipated, which will present a proof of i-RISC concept viability.

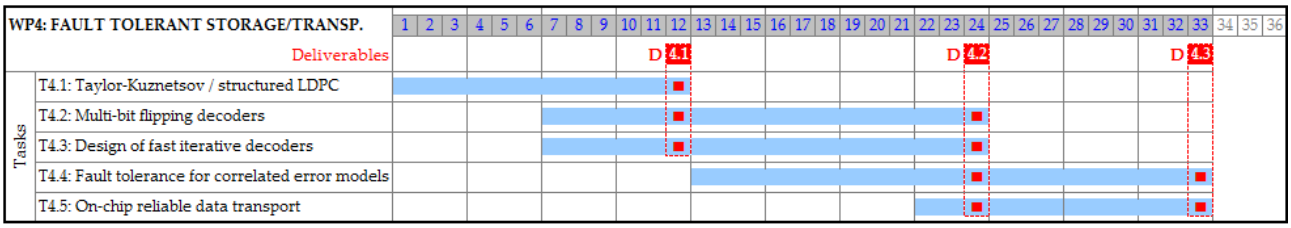


Figure 1: Gantt chart of WP4.

Executive Summary

In this chapter we present a summary of the activities carried out within the by WP4 framework, during the period Month 25 to Month 33 (M25-M33) of i-RISC project and the obtained results. We briefly discuss the most important technical contributions of our work with highlights on their relevance to the overall i-RISC project strategy.

During the M25-M33 time period we sought to achieve the following objectives:

- Objective 4.3: Analysing the robustness of memories designed for tolerating both spatial and temporally correlated errors.

- Objective 4.4: Designing the constrained codes for intra/inter-chip bus connections.

List of objectives indicates that Task 4.1 - Taylor-Kuznetsov memory architectures based on structured LDPC codes, Task 4.4 - Fault tolerance for correlated error models and Task 4.5 - On-chip reliable data transport were mostly investigated during M25-M33. Although it had been scheduled that the work regarding Task 4.1 is to be finished at the end of the first year of the project, our preliminary findings have encouraged us to continue this line of research. This lead to considerable contributions beyond the state-of-the-art research and enabled us to reveal fundamentally different behavior of faulty decoders of structural LDPC codes than those considered in the literature.

Our main contributions related to these tasks can be summarized as follows:

- **Fault-Resilient Decoders made of Unreliable Components (Task 4.1)**. Chapter 1 contains finite-length analysis of the faulty Gallager B decoder. We show evidence that probabilistic behavior of a decoder due to unreliable components can be exploited to our advantage and lead to an improved performance and reduced hardware redundancy. We notice that this unorthodox behaviour of the Gallgaer B decoder is related to structural properties of Tanner graph of a code, and performance of codes whose graphs contain small trapping sets can be improved by gate failures. Such codes are for example quasi-cycle LDPC codes. The performance of different codes are evaluated by using Monte Carlo simulations.

- **Iterative Decoders with Deliberate Message Flips and Rewinding (Task 4.1)**. In Chapter 2 we present new analytical tool which can be used for finite-length analysis of LDPC decoded by hard-decision decoders built form unreliable gates. By tracking the dynamics of decoding process we form Markov chain, which for a specific set of parameters mimics the behaviour of the iterative decoder. This enables quick and accurate derivation of frame and bit error rate for wide range of LDPC codes. However, due to complexity of the analysis the Markovian-based approach is applicable only for codes with moderate code lengths.

- **Design of Taylor-Kuznetsov Memories built from Gallager-B LDPC Decoders (Task 4.1/Task 4.4)**. In Chapter 3 we introduce a new time-dependent memory degradation model, which gives the amount of errors that are introduced by the hardware in the memory during a given time duration. We then propose an analytical method to predict the reliability of the memory architecture for fixed refresh time, code, and decoders parameters. Furthermore, we derive the minimum refresh time that can be tolerated by the memory to be reliable. Based on our method the optimal code and decoder parameters that minimize the redundancy can be selected.

- **Bit-flipping decoding under data-dependent gate failures (Task 4.4)** In Chapter 4 we investigate the guaranteed error-correction capability of the bit-flipping decoder partially built from faulty gates, which are subjected to data-dependent gate failures. We improve the results presented in Deliverable 4.2 by providing less strict conditions which guarantee correction of a fixed fraction of errors. We prove that for every regular code with column weight at least 8, correction capability increases linearly with code length.

- **Reliability of Memories Built from Unreliable Components under Data-Dependent Gate Failures (Task 4.4)**. Based on the results presented in Chapter 5, in Chapter 4 we propose a low complexity memory architecture that is capable to preserve all stored information for arbitrary long time period under data-dependent gate failures. We prove that a much large fraction of component failures can be tolerated than presented in the state-of-the-art literature. Similarly, our results are improved version of our previous work, which is given in Deliverable 4.2.

- **Error Resilient LDPC Enhanced 3D-Memory Architecture (Task 4.4)**. In Chapter 6 we create the practical premisses for the LDPC utilization for memory fault detection and correction. With this respect we introduce a fault resilient 3D polyhedral memory general architecture and framework, which makes use of an LDPC based scrubbing approach to perform user transparent memory maintenance. We rely on the memory stack augmentation with an LDPC codec dedicated die and take advantage of the polyhedral memory reach addressing mode set to minimize potential conflicts between data accesses coming from computational units and from the scrubbing controller. Our experiments indicate that for the same redundancy requirements our approach outperforms the state of the art Hamming code based counterpart in terms of error correction capability while having very limited implications on memory performance and availability.

- **Reliable data transport (Task 4.5)**. In Chapter 7 we investigate coding based possibilities to increase interconnect, e.g., on-chip busses, NoC links, performance, in terms of transmission delay, energy consumption, and error resilience. To this end we introduce and evaluate codec assisted data transport structures able to deal with technology scaling related phenomena, e.g., crosstalk and transmission delay variability, at the expense of a reasonably small area overhead. We present 3 types of data encoding techniques, namely, Constrained, Repetition, and Haar, apply them on an 8-bit wide interconnect segment, and evaluate their practical implications when using a 45nm commercial CMOS technology. Our simulations indicate that (i) substantial energy savings of up to 58% can be achieved by properly tuning the codec specifics to the interconnect length and workload data profile and (ii) the Haar codec based approach is the most general purpose one as it outperforms the other ones in most of the considered cases. In view of (ii) we augment the Haar based interconnect with error correction capabilities and introduce a Single Error Correction and Double Error Detection scheme adapted to the peculiarities of the Haar system, which makes the Haar augmented interconnect not only energy effective but also robust against deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) induced transmission errors.

Chapter 1

Fault-Resilient Decoders made of Unreliable Components

Abstract: *In this chapter we present our recent results on iterative Gallager B decoder made of unreliable logic gates. We show evidence that probabilistic behavior of a decoder due to unreliable components can be exploited to our advantage and lead to an improved performance and reduced hardware redundancy. We provide examples of such decoder behavior and give an explanation of this phenomena following from our insights in iterative decoding dynamics. Iterative decoding can be viewed as a recursive procedure for Bethe free energy function minimization, and the randomness in a message update may help the decoder to escape from local minima. The decoder operates in a stochastic fashion, but the random perturbations do not require any additional hardware as they are built-in the faulty hardware itself. Thus the decoder harvests good deeds of logic gate faults.*

Work presented in this chapter has been published in: B. Vasic, P. Ivanis, S. Brkic, V. Ravanmehr "Fault-Resilient Decoders and Memories made of Unreliable Components", *Proc. of Information Theory and Applications Workshop*, San Diego, USA, February 2015 [P5]. The project acknowledges the valuable contribution of Dr. Vida Ravanmehr from the University of Arizona, not funded from the i-RISC project. The aforementioned paper is reproduced here with her kind permission, to help maintain the consistency of the work presented.

1.1 Introduction

Due to the increase in density integration, lower supply voltages, and variations in technological process, complementary metal-oxide-semiconductor (CMOS) and emerging nanoelectronic devices are inherently unreliable. Moreover, the demands for energy efficiency require that in current CMOS design the energy consumption must be reduced by several orders of magnitude, which can be done only by an aggressive scaling of supply voltage. Consequently, the signal levels are much lower and closer to the noise level, which drastically reduces the component noise immunity and leads to unreliable behavior. It is widely accepted that future generations of circuits and systems must be designed to deal with such unreliable components.

The main feature of the existing work on fault-tolerant decoders is a focus on the analysis of the existing decoder types and demonstrating their robustness to unreliability of logic gates [2–7]. This was also underlying idea of our prior work [8, 9]. On the contrary, the idea of the this chapter is to allow or deliberately introduce randomness in a decoder in order to improve convergence in the spirit of stochastic approximation method [10, 11].

The first trace of this idea can be found in Gallager‘s work where the random flips are used to resolve ties in the majority voting operation in the variable node, while the first iterative decoding algorithm that explicitly relies on randomness to correct errors is Miladinovic and Fossorier's Probabilistic Bit Flipping (PBF) [12]. A closely related technique of adding noise to messages in a BP decoder on

the AWGN channel is by Leduc-Primeau *et al.* [13] for reducing error floor in the context of perfect decoders. Other schemes such as stochastic decoding also uses randomness but in a very different way - by representing messages as random sequences [14–17]. Randomness in message update schedule [18], is also observed to yield improved convergence.

Recently it was shown by Sundararajan *et al.* [19] that random perturbations can be used to increase the performance of a gradient descent bit flipping decoder (GDBF), introduced by Wadayama *et al.* [20]. At the same time we observed that randomness coming from computational noise even more improves the GDBF decoding. Based on that conclusion we developed a probabilistic gradient-descent bit flipping algorithm (PGDBF) [21] for the Binary Symmetric Channel (BSC). Our decoder incorporates the idea of GDBF with Miladinovic and Fossorier's probabilistic approach [12], but it contains some critical novelties which consists in modifying the inverse function [20, Eqn. (6)]. We showed that random perturbations applied to variable nodes lead to escaping from a local maximum of the GDBF objective function. In addition, the perturbations make the decoder inherently tolerant to hardware unreliability. Our most recent work contains the further improvement of the PGDBF algorithm based on multiple decoding attempts and random re-initializations (MUDRI) of decoders [22].

In this chapter we consider the Gallager B decoder in which operations are subjected to processing noise. We show how the hardware unreliability can be used to increase error-correction capability of certain quasi-cyclic low-density parity-check (LDPC) codes, in the error-floor region. Our conclusions relies on the fact that logic gates failures can break small trapping-sets, which are the main cause of the decoder failures in that region. Vasic and Chilappagari [23] observed that the faulty Gallager-B decoder is equivalent to the Taylor memory architecture [24], which means that our results can be directly applied to the reliability analysis of memories built from unreliable components.

The rest of the chapter is organized as follows. In Section 1.2 we give a brief description of the faulty Gallager B decoder. In Section 1.3 the idea of breaking trapping sets is discussed. Section 1.4 is dedicated to the numerical results. Finally, some concluding remarks and future research directions are given in Section 1.5.

1.2 The Faulty Gallager B Decoder

Consider a (γ, ρ)-regular binary LDPC code, denoted by (N, K), with code rate $R = K/N \geq 1 - \gamma/\rho$ and parity check matrix $\mathbf{H}$. The parity check matrix is the bi-adjacency matrix of a bipartite (Tanner) graph $G = (V \cup C, E)$, where V represents the set of N variable nodes, C is the set of $N\gamma/\rho$ check nodes, and E is the set of $N\gamma$ edges. Each matrix element $\mathbf{H}_{c,v} = 1$ indicates that there is an edge between nodes $c \in C$ and $v \in V$, which are referred as *neighbors*. Let $\mathcal{N}_v$ ($\mathcal{N}_c$) be the set of neighbors of the variable node v (check node c). Then, $|\mathcal{N}_v| = \gamma$, $\forall v \in V$ and $|\mathcal{N}_c| = \rho$, $\forall c \in C$, where $|\cdot|$ denotes cardinality.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ denote a codeword of an LDPC code, where x_v represents the binary value associated with the variable node v. During the transmission over a Binary Symmetric Channel (BSC), the code bits are flipped with probability α and received as $\mathbf{y} = (y_1, y_2, \ldots, y_N)$.

The Gallager B decoder works by sending binary messages over the edges of the graph. The messages are calculated based on the *nodes update functions*, following the rule that a message sent over an edge is obtained based on all received messages except the one arriving over that edge [25]. The check node update function corresponds to the $(\rho - 1)$-input XOR logic gate, and $(\gamma - 1)$-input majority logic (MAJ) gate is used for the variable node update function implementation.

Due to hardware unreliability the results of the update functions are not always correctly computed. We adopted the "wire" model described in [2], where an edge is modeled as a BSC with a known crossover probability. Let $\nu_{v,c}^{(\ell)}$ be the message sent by the variable node v to its neighbor $c \in \mathcal{N}_v$, at the iteration ℓ, and let $\hat{\nu}_{v,c}^{(\ell)}$ be the message that is actually received by the node c. Then, we define

the following relation

$$\hat{\nu}_{v,c}^{(\ell)} = \begin{cases} \nu_{v,c}^{(\ell)} & \text{with probability } 1-\varepsilon_{MAJ}, \\ \nu_{v,c}^{(\ell)} \oplus 1 & \text{with probability } \varepsilon_{MAJ}, \end{cases} \tag{1.1}$$

where ε_{MAJ} represents the probability of failure of MAJ gates. Similarly, let $\nu_{c,v}^{(\ell)}$ be the message sent by the variable node c to its neighbor $v \in \mathcal{N}_c$, at the iteration ℓ, and let $\hat{\nu}_{c,v}^{(\ell)}$ be the message that is actually received by the node v. Then, we have

$$\hat{\nu}_{c,v}^{(\ell)} = \begin{cases} \nu_{c,v}^{(\ell)} & \text{with probability } 1-\varepsilon_{\oplus}, \\ \nu_{c,v}^{(\ell)} \oplus 1 & \text{with probability } \varepsilon_{\oplus}, \end{cases} \tag{1.2}$$

where $\varepsilon_{\oplus}$ represents the probability of failure of XOR gates. We next summarize the faulty Gallager B decoder.

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

$$\nu_{v,c}^{(\ell)} = \begin{cases} s & \text{if } |\{c' \in \mathcal{N}_v \setminus c : \hat{\nu}_{c',v}^{(\ell-1)} = s\}| > \lfloor \gamma/2 \rfloor, \\ y_v & \text{otherwise.} \end{cases} \tag{1.3}$$

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

$$\nu_{c,v}^{(\ell)} = \bigoplus_{v' \in \mathcal{N}_c \setminus \{v\}} \hat{\nu}_{v',c}^{(\ell)}. \tag{1.4}$$

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

Note that in addition to logic gates needed to calculate messages that are passed on the edges of the bipartite graph the decoder also requires logic gates for the final bit estimations and parity-checks calculation. If we allow these gates to be unreliable, the performance of the decoder would be determined by the failure probabilities of these gates, not by the error control scheme. Thus, it is reasonable to assume that these gates are perfect. Similar assumption was also used in other relevant literature [23, 26, 27].

1.3 Eliminating The Trapping Sets by Gate Failures

It is well known that the failures of Gallager B decoding caused by the low-weight error patterns are mainly due to the existence of the harmful structures in the Tanner graph of LDPC codes called "trapping sets" [28]. A set of variable nodes T is called an (a, b) trapping set if it contains a variable nodes and the subgraph induced by these variable nodes has b odd degree check nodes. In [29], the most harmful structures of column weight 3 LDPC codes using the Gallager B decoder over the BSC are given.

A faulty (5,3) trapping set in a column weight 3 LDPC code of girth $g = 8$ is shown in Fig. 1.1. In this figure, the circles denote the variable nodes and the squares denote the check nodes. The faulty XOR gates are at the check nodes c_2 and c_3 and the faulty MAJ gates are at the variable nodes v_2, v_4 and v_5 which are shown with black arrows.

Logic gate failures as inherent sources of randomness in the existing literature, are usually seen as undesirable. This belief comes from the fact that the capacity of LDPC codes cannot be achieved under unreliable decoding operations [2, 6, 30]. However, the presence of randomness can be beneficial for the finite length codes, as can eliminate small trapping sets. To illustrate this, we study the

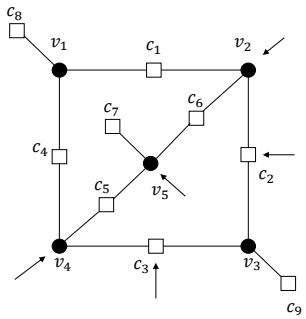
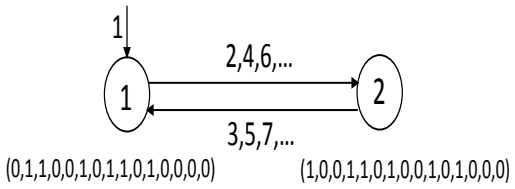
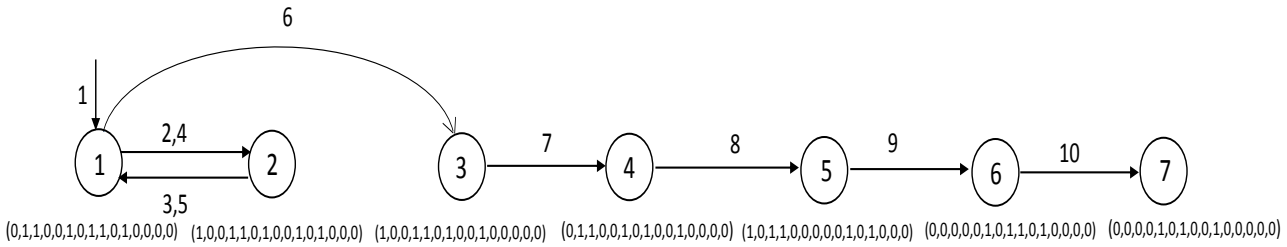


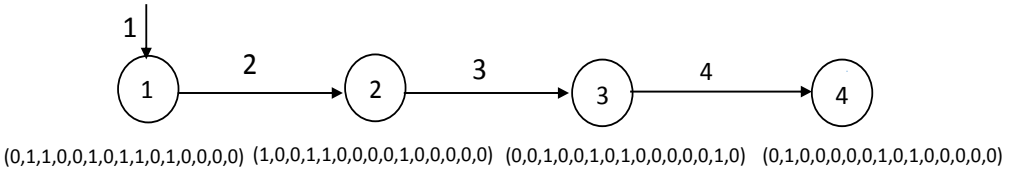
Figure 1.1: A faulty (5,3) trapping set.



(a)



(b)



(c)

Figure 1.2: The state diagram of the Markov chain model on a faulty (5,3) trapping set. (a) $\varepsilon_{\oplus} = 0.001$ and the decoder never converges to a codeword, (b) $\varepsilon_{\oplus} = 0.01$ and the decoder converges to the zero codeword after 10 iterations, (c) $\varepsilon_{\oplus} = 0.1$ and the decoder converges to the zero codeword after 4 iterations.

dynamic of messages passed from check nodes to variable nodes on a trapping set in each iteration of the faulty Gallager B decoder. For this purpose, we introduce a Markov chain model, which captures the dynamic of the messages.

Let C_T be the set of check nodes in the subgraph induced by the trapping set T. Let define the Markov source S with $2^{|T|\gamma}$ states, where the state i, $1 \leq i \leq 2^{|T|\gamma}$, corresponds to one possible binary sequence of length $|T|\gamma$. Clearly, a sequence $\{{\nu_{c,v}}^{(\ell)}\}_{c \in C_T, v \in T}$, where $\nu_{c,v}^{(\ell)}$ is the message passed from the check node c to the variable node v at time ℓ, maps to one state of the source S. A transition from a state i at time ℓ, to state j at time $\ell+1$ is obtained by the variable and check node operations, as well as the level of unreliability of logic gates parameters $\varepsilon_{\oplus}$ and ε_{MAJ}.

For a perfect decoder in which $\varepsilon_{MAJ} = 0$ and $\varepsilon_{\oplus} = 0$, the attractor basin has a relatively small number of states. In the state diagram of this model, the states oscillate and the decoder never

converges to a valid codeword. While in a faulty decoder where ε_{MAJ} and/or $\varepsilon_{\oplus}$ are sufficiently greater than 0, the number of states is much larger than that of a perfect decoder. That is because that in the faulty decoder, there is a positive probability for each of the possible $2^{|T|\gamma}$ messages to appear as a state in the Markov chain model. Moreover, there is a positive probability of leaving the states of the attractor basin of a perfect decoder. These features of the faulty decoder may cause the decoder to eventually correct the errors and converge to the zero codeword.

To show how the faulty gates can break a trapping set and correct the errors, we consider the (5,3) trapping set and keep track of the messages from the check nodes to variable nodes in each iteration of the faulty Gallager B decoder. For simplicity, we assume that $\varepsilon_{MAJ} = 0$. Thus, the faulty gates are only the XOR gates with the probability of failure $\varepsilon_{\oplus} > 0$. Therefore, the messages from check nodes $\{c_1, c_2, ..., c_9\}$ to variable nodes are flipped with probability $\varepsilon_{\oplus}$. We know that the critical number of the (5,3) trapping set is 3 and the 3 variable nodes that need to be in error are the those that are connected to degree-1 check nodes. Recall that the critical number of a trapping set is the minimum number of the erroneous variable nodes that cause the decoder fails [31]. Thus, assume that v_1, v_3 and v_5 are in error; i.e. $v_1 = v_3 = v_5 = 1$. Suppose the decoder stops if the decoder finds a codeword or reaches the maximum number of iterations. The state transitions corresponding to the messages from check nodes to variable nodes for $\varepsilon_{\oplus} = 0.001$, $\varepsilon_{\oplus} = 0.01$ and $\varepsilon_{\oplus} = 0.1$ are shown in Fig. 1.2. As can be seen in Fig. 1.2(a), for $\varepsilon_{\oplus} = 0.001$, the messages oscillate between states 1 and 2. In this case, the decoder never converges to a codeword. We note that when $\varepsilon_{\oplus} = 0$, the dynamic of messages is the same as the dynamic given in Fig. 1.2(a). Fig. 1.2(b) shows that the decoder corrects all errors in 10 iterations when $\varepsilon_{\oplus} = 0.01$. As shown in Fig. 1.2(b), in the first 5 iterations, the messages oscillate between states 1 and 2. However, after the 6th iteration, the states change from the state 3 to 7 in which the decoder stops and corrects all errors. Finally, in Fig. 1.2(c), the dynamic of messages is shown for $\varepsilon_{\oplus} = 0.1$ that depicts the convergence of the decoder to the zero codeword in 4 iterations.

In the above discussion, we investigated the dynamic of messages from the check nodes to variable nodes in an isolated (5,3) trapping set. To see how the faulty Gallager B decoder performs on a (5,3) trapping set in the (155,64) Tanner code, we put 3 errors on the variable nodes connecting to degree-1 check nodes in a subgraph corresponding to the (5,3) trapping set. Then, for different values of the probability in XOR gates, we ran the faulty Gallager B decoder for at most 100 iterations and stored the number of variable nodes that are eventually in error. The result is shown in Fig. 1.3. As we expected, for sufficiently small $\varepsilon_{\oplus}$, the decoder cannot correct errors located on the trapping set and for large enough values of $\varepsilon_{\oplus}$, the decoder incorrectly decodes some other variable nodes that were originally correct, to 1. However, there are some values of $\varepsilon_{\oplus}$ for which the decoder corrects all errors.

1.4 Numerical Results

In this section we further elaborate the idea that hardware unreliability can lead to the performance improvement. We evaluate the frame error rate (FER) and the bit error rate (BER) of various LDPC codes, under i.i.d. failures in logic gates, explained in Section 1.2.

We mostly analyze quasi-cyclic (QC) codes, based on circulant matrices [32] and Margulis codes [33], for which are known to perform poorly in the error floors, as their bipartite graphs contains small trapping sets [34]. We also consider Latin square (LS) based codes that are designed to be free of small trapping sets [35], as a reference for the improvement obtained by breaking the trapping sets in faulty gates. We only investigate (3,5)-regular LDPC codes, with girth $g = 8$, as they allow low decoding complexity, but we expect that similar conclusions can be derived for other regular LDPC codes.

First, we consider how the probability of random i.i.d. failures in XOR/MAJ gates of Gallager B decoder affects the FER of the two codes with the same construction parameters, but different error floor behavior. In Fig. 1.4 the two different scenarios are illustrated: (i) when check node processing is reliable and only MAJ gates are faulty, and (ii) when variable node processing is reliable and only XOR gates are faulty. The performance of LS(155,64) code (also denoted as C_1 code in [35]), marked with red lines, are approximately constant, up to the certain probability of logic gate failures. It

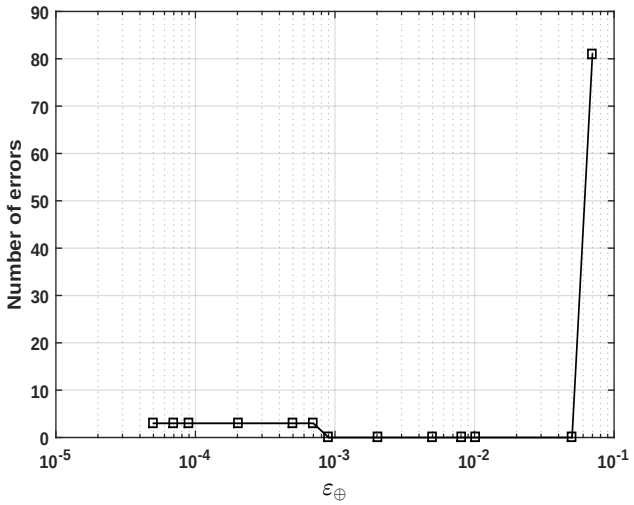


Figure 1.3: Number of errors versus $\varepsilon_{\oplus}$ after running the faulty Gallager B decoder for 100 iterations on the (155,64) Tanner code. The input of the decoder has 3 errors located on the variable nodes that are connected to degree-1 check nodes in a (5,3) trapping set.

can be noticed that MAJ gates failures are more damaging, as the performance degradation rapidly increase when $\varepsilon_{MAJ} > 10^{-3}$, while, in the case of faulty XOR gates, the similar effect is visible only for $\varepsilon_{\oplus} > 5 \times 10^{-3}$. On the other hand, the performance of Tanner $(155, 64)$ code, presented with blue lines, is not necessarily degraded with the increase of logic gates unreliability. In fact, there exist $\varepsilon_{MAJ} > 0$, and $\varepsilon_{\oplus} > 0$, which results in the highest error-correction capability. For that optimal case the performance of the Tanner $(155, 64)$ code approaches closely to the performance of LS $(155, 64)$, which are a magnitude higher when decoding is done by the perfect decoder.

This surprising effect is related to the positive impact of hardware failures to the breaking of the trapping sets, and it is similar to the effect reported for the gradient descent bit flipping decoder [21]. However, we noticed that the benefit of using unreliable logic gates is related also to the mutual relations of communication channel crossover probability α and $\varepsilon_{\oplus}$ and ε_{MAJ}, respectively. When channel errors are dominant, $\alpha \gg \varepsilon_{\oplus}, \varepsilon_{MAJ}$, the influence of hardware unreliability on the decoder performance is limited, and FER stays mostly determined by the smallest trapping sets. On the other hand, if the logic gate failures probabilities are much higher than the channel crossover probability ($\alpha \ll \varepsilon_{\oplus}, \varepsilon_{MAJ}$), gate failures are no more useful. In such a case, errors added during the decoding prevent the correction of channel errors.

As it can be observed from Fig. 1.5, the improvement caused by XOR logic gates unreliability is notable for a wide range of failure probabilities, and it is mostly pronounced in the error-floor region, when channel induced errors are rare. For that case the Tanner $(155, 64)$ code performs the same as LS $(155, 64)$, when decoded by the perfectly reliable decoder. However, a small degradation is observed for high α.

This results indicates that the performance of the Tanner code $(155, 64)$ can be improved even when perfectly reliable decoder is used, if the random binary sequence with $Pr(1) \approx 0.01$ is added in the check nodes, or the corresponding sequence with $Pr(1) \approx 0.002$ is added in variable nodes. For this purpose, stochastic XOR/MAJ gates can be used [36]. The performance of Tanner code can be further improved if no failures are added at the beginning of the decoding process, for instance during the first L_1 decoding iterations. In this case the Gallager B decoder is given time to correct the error pattern, and only if decoding fails after L_1 iterations the probabilism is used. The corresponding numerical results, presented in Fig. 1.5, reveals that, when L_1 adequately chosen ($L_1 = 10$), the perfectly reliable Gallager B decoder is outperformed for all considered crossover probabilities α. The use of probabilism in order to improve the Gallager B decoding is included in our future research.

We next compare the performance of the Tanner code $(155, 64)$ with the code from the same

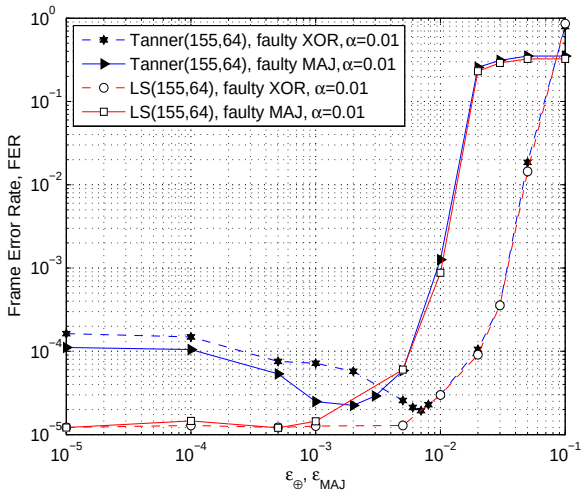


Figure 1.4: FER as a function of probability of the gate failure, $\alpha = 0.01$, $L = 100$.

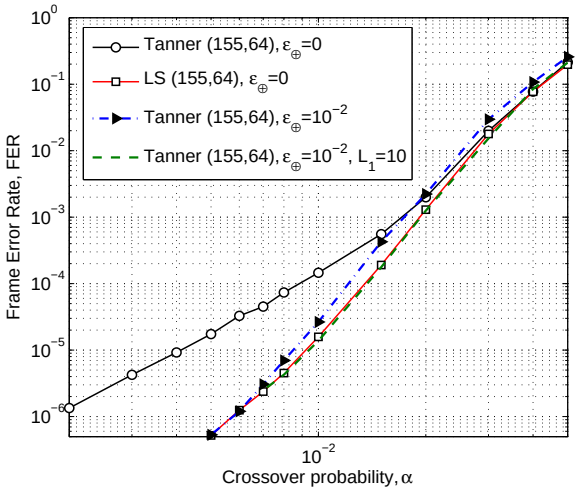


Figure 1.5: FER as a function of crossover probability, for the two $(155, 64)$ codes

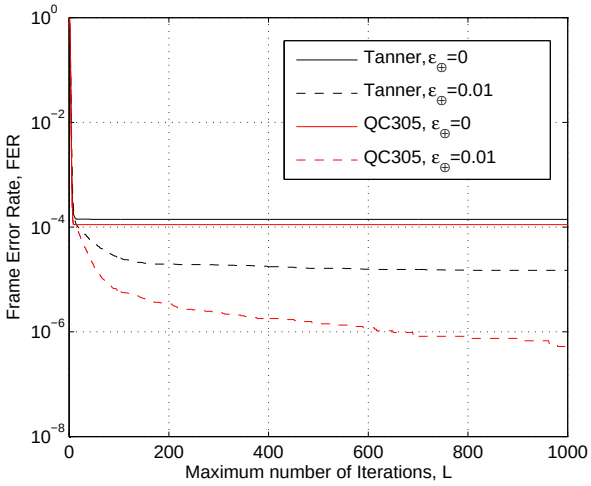


Figure 1.6: FER as a function of maximum number of iterations, impact of gate failures, $\alpha = 0.01$.

(3,5)-regular ensemble and approximately same code rate, but with the codeword length $N = 305$ (constructed over field GF(61)) [32]. The FER values of these two codes are illustrated in Fig. 1.6 as a function of maximal number of iterations L. The results are presented for the crossover probability $\alpha = 0.01$, with and without failures in XOR gates. Although the longer code have better correction capability when Belief-Propagation decoding algorithm is applied [32], decoding by the Gallager B algorithm results in the minor performance improvement, when compared to the Tanner $(155, 64)$ code. For both codes, the decoding saturates after 15 iterations and the performance is not further improved with the increase of parameter L. If hardware failures are present, the performance continues to improve with increase of L, up to the number of iterations that is comparable with the codeword length. If we allow L to be sufficiently large, FER of the code with $N = 155$ can be reduced for one order of magnitude and FER of the code with $N = 305$ can be reduced for the two orders of magnitude.

In Fig. 1.7, we illustrate XOR gate failures influence to the performance of QC codes with $N = 305$ and $N = 755$ [32]. Similarly, as observed for the Tanner $(155, 64)$ code, the performance of both codes are improved, if the number of decoding iterations is large. The code with $N = 755$ is of a special interest due to its atypical behavior. It is constructed over GF(151) and has the bad distance profile, and therefore has an inferior performance in the error floor region, when compared to the randomly constructed code with the same codeword length and code rate [32]. By using the faulty Gallager B decoder we implicitly randomize decoding process and the performance is improved as the impact of the small trapping sets is minimized. It should be noted that, for the code with $N = 755$, the probability of converging to a codeword different from the transmitted codeword is not negligible. We observed that, for example, for $\alpha = 0.01$ and $L = 1000$, in average half of the decoder failures are the result of that phenomenon.

The general conclusion that for the large codeword length we need larger maximal number of iterations to obtain the full gain of the gates unreliability, especially in the waterfall region, is additionally strengthened by the case of Margulis code $(2640, 1320)$. It can be easily observed from Fig. 1.8 that decoding convergence continue for more than $L = 1000$ iterations for both analyzed values of the crossover probability ($\alpha = 0.01$ and $\alpha = 0.015$). As the code has good distance properties probability of miss-correction is negligible even for very large values of L. It is clear that the increase of parameter L increase the maximum latency of the decoding process and this can be critical for certain applications.

The past work related to the faulty Gallager B decoder was mostly dedicated to the infinite code length analysis by the density evolution (DE) technique [4, 5]. The results obtained using the DE tool are rather pessimistic, as they show that the BER performance converges to a rather high value, especially when decoder is built from faulty MAJ gates. However, in Fig. 1.9 we show different

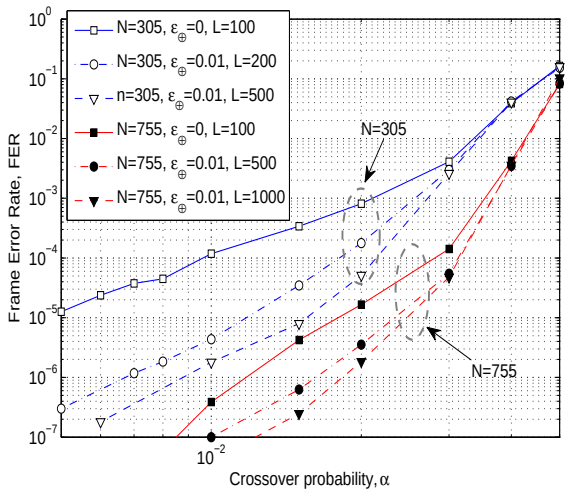


Figure 1.7: FER as a function of crossover probability, two codes with $\gamma = 3$, $\rho = 5$.

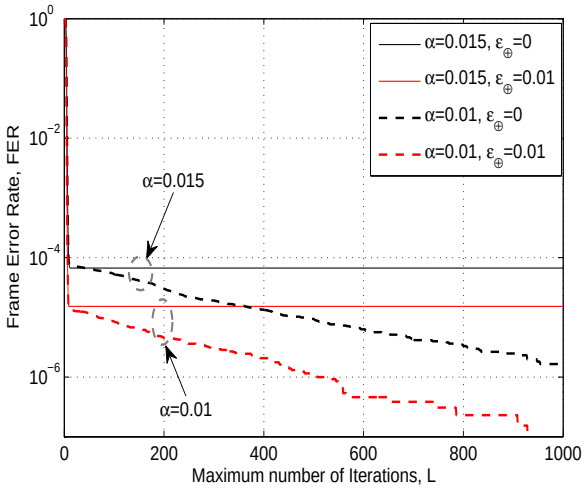


Figure 1.8: FER as a function of maximum number of iterations, Margulis $(2640, 1320)$ code.

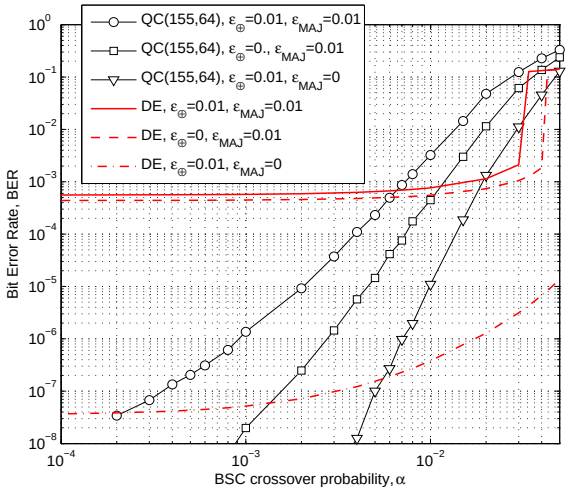


Figure 1.9: BER as a function of crossover probability, Monte Carlo simulation for a finite length code and numerical results obtained by using density evolution technique.

behavior of finite length codes. For example, it can be observed that the Tanner $(155, 64)$ code, when $L = 100$, outperforms the infinite length code from the same ensemble, in the error floor region. This additionally illustrates the significance of trapping set analysis, for the decoders built from unreliable components.

1.5 Conclusion

In this chapter we showed that uncertainty of the logic gate operations is not always undesirable in the message-passing decoding of LDPC codes. On the contrary, we observed that it can significantly improve the Gallager B decoder performance. Random failures of logic gates result in correction of some error patters that are uncorrectable by the decoder made of reliable components. By analyzing the topology of Tanner graphs, we come to the conclusion the improvement is mostly notable with codes that contain small trapping-sets. Accordingly, they performance in the error-flor region is highly improved.

Chapter 2

Iterative Decoders with Deliberate Message Flips and Rewinding

Abstract: *Many natural and man-made systems involving computing, control and communications are made of unreliable parts or parts exhibiting stochastic behavior. No system using less reliable components is known to operate better than a system using perfect components. Here we present such a system. We give an architecture of a storage system consisting of a storage medium made of unreliable memory elements and an error correction circuit made of a combination of noisy and perfect logic gates that is capable of retaining the stored information longer and with lower probability of error than a storage system with a correction circuit made completely of perfect logic gates. Our correction circuit is based on iterative decoding of low-density parity check codes, and uses the positive effect of errors in logic gates to correct errors in memory elements. In the spirit of Marcus Tullius Cicero's "Clavus clavo eicitur," ("one nail drives out another") the proposed storage system operates on the principle: "Error errore eicitur" - "one error drives out another." The randomness that is present in the logic gates makes these classes of decoders superior to their perfect counterparts. Moreover, random perturbations do not require any additional computational resources as they are inherent to unreliable hardware itself.*

2.1 Introduction

Origins of a system unreliability lie in the underlaying physics mechanisms governing the operation of its parts. For example, in micro and nano-electronics devices it is due to low supply voltages and imperfections in manufacturing process [37, 38], in space missions electronics due to high energy particles striking the semiconductor devices [39]. In order to ensure the robustness of a system to noise and/or faults in its parts, one relies on redundancy and computation, which compensate the negative effects of unreliable parts. Typical examples are systems for storage of information. Information written on a storage medium (magnetic, optical, flash memory, etc.) is physically represented as one of the several stable states of the memory elements comprising the medium (magnetization direction of magnetic grains, surface reflectivity, charge in capacitors, etc.). Since the reliability of the memory elements cannot be improved due to the underlying physics and manufacturing cost, one relies on a periodic refreshing of the stored data content to prevent the data decay. In this process, the errors that may have occurred in the medium are corrected in a computational device, called a correction circuit. Without loss of generality, a correction circuit can be assumed to perform a sequence of binary operations, and all traditional systems rely on the assumption that the correction circuit is perfect, i.e., made of perfect Boolean logic gates. In other words, computations performed in the correction circuit are deterministic, while randomness (in the form of noise and/or errors) exists only in the storage medium.

The above assumption is certainly appropriate for correction circuits in which the reliability of logic gates is many orders of magnitude higher than reliability of memory elements. However, an interesting situation arises when a correction circuit itself is made of noisy components. For example, in low-

powered submicron complementary metal-oxide-semiconductor (CMOS) chips mentioned above, the supply voltage is kept low in order to reduce power consumption, thus making logic gates susceptible to noise and increasing the probability of incorrect logic gate output. Similar effect exists in nano and quantum computing technologies. Due to unreliability of its logic gates, the correction circuit - whose purpose is to correct errors - introduces errors in the process of correcting errors from the storage medium. Making logic gates reliable (for example by using larger supply voltages) appears as a logical solution. We show that the correction circuit can operate in the unreliable (low-power) gate - regime, and still be able to correct more errors than the perfect correction circuit. Moreover, the random perturbations do not require any additional computational resources as they are built in the unreliable hardware itself. The key is to store the information in a coded form and use a special type of decoding algorithm in the correction circuit, as we explain next.

The first trace of the deliberate error idea can be found in Gallager's work where the random flips are used to resolve ties in the majority voting operation in the variable node, while the first iterative decoding algorithm that explicitly relies on randomness to correct errors is Miladinovic and Fossorier's Probabilistic Bit Flipping (PBF) [12]. A closely related technique of adding noise to messages in a BP decoder on the AWGN channel is by Leduc-Primeau *et al.* [13] for reducing error floor in the context of perfect decoders.

Recently it was shown by Sundararajan *et al.* [19] that random perturbations can be used to increase the performance of a gradient descent bit flipping decoder (GDBF), introduced by Wadayama *et al.* [20]. At the same time we observed that the randomness coming from computational noise even more improves the GDBF decoding performance. Based on that result we developed a probabilistic gradient-descent bit flipping (PGDBF) algorithm [40] for the Binary Symmetric Channel (BSC). Introducing a random perturbation is reminiscent of the operation of mutation in genetic algorithms [41] (the inverse or energy function in the PGDBF algorithm is reminiscent of the fitness function in genetic algorithms). However the similarity ends here because the gradient-descent bit flipping does not maintain a "population" of codeword candidates, i.e., only one candidate for the codeword estimate is kept during iterations.

The fact that noise can be used constructively has been observed in many of natural and artificial analog signal processing systems, and is known as stochastic resonance [42, 43]. However, due to huge complexity of our correction circuits, the available stochastic resonance analysis tools are not sufficient to characterize their improved robustness. We also note that the above positive effects of noise are also observed in analog decoders where randomness comes from transistor mismatch [44].

A dominant method for analysis of faulty decoders is the noisy-density evolution (DE) technique [27]. It provides a convenient measure of robustness of a decoder, but it cannot provide an explanation why and when a noisy decoder performs better than the perfect one. The underlaying reasons are based on the assumptions of message independence and infinite number of iterations, thus implying an infinitely long code. For length codes, the performance of a perfect iterative decoding depends upon the presence of trapping sets which are annihilated by the randomness in the decoder. Although the analysis is given for the storage system modeled as the BSC, the approach is general. It can be readily extended to different error models, including permanent failures. The proposed approach is also applicable to other decoding algorithms.

The rest of the chapter is organized as follows. In Section 2.2 the preliminaries on iterative message passing decoding of LDPC Codes is discussed. In Section 2.3 we give a description of our storage system architecture. Section 2.4 is dedicated to the theoretical analysis of the noisy Gallager-B decoder performance. The numerical results are presented in Section 2.5. Finally, some concluding remarks and future research directions are given in Section 2.6.

2.2 Preliminaries

2.2.1 Iterative Message Passing Decoding of LDPC Codes

Consider a (γ, ρ)-regular binary LDPC code, denoted by (n, k), with code rate $R = k/n \geq 1 - \gamma/\rho$ and parity check matrix H. The parity check matrix is the bi-adjacency matrix of a bipartite (Tanner)

graph $G = (V \cup C, E)$, where V represents the set of N variable nodes, C is the set of $n\gamma/\rho$ check nodes, and E is the set of $n\gamma$ edges. Each matrix element $H_{c,v} = 1$ indicates that there is an edge between nodes $c \in C$ and $v \in V$, which are referred as neighbors. Let $\mathcal{N}_v$ ($\mathcal{N}_c$) be the set of neighbors of the variable node v (check node c). Then, $|\mathcal{N}_v| = \gamma$, $\forall v \in V$ and $|\mathcal{N}_c| = \rho$, $\forall c \in C$, where $|\cdot|$ denotes cardinality. In irregular LDPC codes, nodes do not necessarily have the same number of neighbors.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ denote a codeword of an LDPC code, where x_v represents the binary value associated with the variable node v. During the transmission over a Binary Symmetric Channel (BSC), the code bits are flipped with probability α and received as $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The messages passed from a check c to a variable node v is denoted by $\mu^{(\ell)}_{c\to v}$. Message passed from a variable node v to the check node c is denoted by $\nu^{(\ell)}_{v\to c}$.

Additionally, let $\mathbf{m}^{(\ell)} = \mu^{(\ell)}_{\mathcal{N}_v\setminus c\to v}$ denote the incoming messages to a variable node v except a message from the check node c. Similarly, $\mathbf{n}^{(\ell)} = \nu^{(\ell)}_{\mathcal{N}_c\setminus v\to c}$ denote all incoming messages to the check node c except from variable node v. Finally, we denote by $\mathbf{l}^{(\ell)} = \mu^{(\ell)}_{\mathcal{N}_v\to v}$, messages incoming to v in the ℓ-th iteration, and by $\mathbf{i}^{(\ell)} = \mu^{(\ell)}_{\mathcal{N}_c\to c}$ messages incoming to c in the ℓ-th iteration.

An iterative decoder $\mathcal{F}$, is a 6-tuple $\mathcal{F} = (\mathcal{M}, \mathcal{Y}, \Phi, \Psi, \hat{\Phi}, \hat{\Psi})$, where $\mathcal{Y}$ and $\mathcal{M}$ are the channel (output) and message alphabets, respectively, Φ,Ψ are the update functions used in variable and check nodes, and $\hat{\Phi}$ is the decision function, and $\hat{\Psi}$ is the check estimate function. For the case of BSC, $\mathcal{Y} = \{\pm 1\}$. By convention, $+1$ corresponds to the input 0 and -1 to 1. For the decoders discussed here, messages are also binary and $\mathcal{M} = \{\pm 1\}$. The sign of a message $x \in \mathcal{M}$ can be interpreted as the estimate of the bit (positive for zero and negative for one) associated with the variable node to or from which x is being passed.

The message from variable node v are initialized to $\Phi(y_v, \mathbf{0})$, and in each iteration updated according to the rules Φ and Ψ. The function $\Psi : \{\mathcal{M}\}^{\rho-1} \to \mathcal{M}$ is used for update at a check node with degree ρ, so that $\mu^{(\ell)}_{c\to v} = \Psi(\mathbf{n}^{(\ell-1)})$. Note that Ψ is a symmetric function, i.e., it any permutation of its variables leaves the function unchanged. The function $\Phi : \mathcal{Y} \times \{\mathcal{M}\}^{\gamma-1} \to \mathcal{M}$ is used for updating outgoing message of a variable node with degree γ. $\nu^{(\ell)}_{v\to c} = \Phi(y_v, \mathbf{m}^{(\ell)})$. Note that Φ is partially symmetric in the variables $\mathbf{m}$. The "strength" of the variable node v in ℓ^{th} iteration, $\lambda^{(\ell)}_v$, is used to decide the value $\hat{x}_v$ of the v-th code bit. Let $\mathbf{l} \in \{\mathcal{M}\}^{\gamma}$, be an unordered γ-tuple representing all incoming messages to the variable node v, from its neighbors, the decided bit value in ℓ-th iteration is calculated on the signs of the elements of $\mathbf{l}$ as $\hat{x}^{(\ell)}_v = \hat{\Phi}(\mathbf{l}^{(\ell)}) = \mathbb{1}_{l^{(\ell)}_v<0}$. An estimate of the check node value c in the ℓ-th iteration is $\sigma^{(\ell)}_c = \hat{\Psi}\left(\mathbf{i}^{(\ell)}\right) = \operatorname{sgn}\left(\prod \mathbf{i}^{(\ell)}\right)$. A check node is said to be satisfied if $\sigma_c = 1$, unsatisfied if $\sigma_c = -1$, and undecided if $\sigma_c = 0$. A syndrome checking verifies that all the check node estimates $\hat{\sigma}^{(\ell)}_c = \hat{\Psi}\left(\hat{\mathbf{x}}^{(\ell)}_{\mathcal{N}_c}\right) = \operatorname{sgn}\left(\prod \hat{\mathbf{x}}^{(\ell)}_{\mathcal{N}_c}\right)$ based on the decoded word $\hat{\mathbf{x}}^{(\ell)}$ are satisfied. In this case, we say that a codeword is found. We say that a codeword is found if all M check nodes are satisfied.

If t is the error correction capability of a given iterative decoder, the decoder is said not to converge to the correct codeword if there exists some error pattern $\mathbf{e}$ of weight $\operatorname{supp}(\mathbf{e}) > t$, which leads to a decoding trajectory $\{\mu^{(\ell)}_{C\to V}\}_{l\geq 0}$ such for arbitrary ℓ there is at least one bit estimate $\hat{x}^{(\ell)}_v = \hat{\Phi}(\mathbf{l}^{(\ell)})$ different from x_v. The support of a vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ denoted by $\operatorname{supp}(\mathbf{x})$ is the set $\{x_i \mid x_i \neq 0\}$. The decision function $\hat{\Phi}$ applied on $\mathbf{l}^{(\ell)}$, the γ-tuple representing all incoming messages to the variable node v in ℓ-th iteration.

2.2.2 Perfect and Noisy Gallager-B Decoders

The Gallager-B decoder is a special case of the iterative decoder above, and it works by sending binary messages over the edges of the graph. The messages are calculated based on the node update functions, following the rule that a message sent over an edge is obtained based on all received messages except the one arriving over that edge. The check node update function Ψ corresponds to the $(\rho-1)$-input XOR logic gate, and $(\gamma-1)$-input majority logic (MAJ) gate is used for the variable node update function implementation. In other words the following operations are performed until the codeword

is found or a maximum number of iteration reached. In each iteration iteration ℓ, for each variable node $v \in V$, and for all $c \in \mathcal{N}_v$.

$$\nu_{v\to c}^{(\ell)} = \begin{cases} y_v & \ell = 0 \\ \Phi(y_v, \mathbf{m}^{(\ell-\mathbf{1})}) & l > 0 \end{cases}$$

where

$$\Phi(y_v, \mathbf{m}) = \begin{cases} \texttt{MAJ}(\mathbf{m}) & \texttt{MAJ}(\mathbf{m}) \neq 0 \\ y_v & \texttt{MAJ}(\mathbf{m}) = 0. \end{cases} \tag{2.1}$$

The function MAJ is defined as $\texttt{MAJ}(\mathbf{m}) = \text{sgn}(\sum \mathbf{m})$ wherein $\sum$ is taken componentwise, and by convention for the sign function, we take $\text{sgn}(0) = 0$. We note that an alternative rule is possible which does not require a register for storing the channel values. In this case the previous variable estimate is used when there is a tie in incoming messages the variable node v. More precisely, $\nu_{v\to c}^{(\ell)} = \hat{x}_v^{(\ell-1)}$, when $\texttt{MAJ}(\mathbf{m}^{(\ell)}) = 0$. Performance superiority of the rule given in Eq. (2.1), justifies the hardware overhead due to perfect registers for storing $\mathbf{y}$. Keeping the received word in a reliable temporary register is crucial for the decoder trajectory stabilization because it anchoress the decoder state to that determined by the initial channel values, and prevents trajectory divergence from the initial value, which in turns reduces the negative effects of gate errors.

In each iteration $l \geq 0$ for each check node $c \in C$ and for all $v \in \mathcal{N}_c$, $\Psi(\mathbf{n}) = \prod \mathbf{n}$, where $\prod$ is taken componentwise and is equivalent to computing the XOR of the incoming bits.

The decided bit value in ℓ-th iteration is calculated as $\hat{x}_v^{(\ell)} = \hat{\Phi}(\mathbf{l}^{(\ell)}) = \Phi(y_v, \mathbf{l}^{(\ell)})$, where $\mathbf{l}^{(\ell)}$ is the set of all incoming messages to a variable node. The estimate of the check node c in the ℓ-th iteration is $\sigma_c^{(\ell)} = \hat{\Psi}\left(\mathbf{i}^{(\ell)}\right) = \text{sgn}\left(\prod \mathbf{i}^{(\ell)}\right)$. where $\mathbf{i}^{(\ell)}$ is the set of all messages coming to c.

Again, by convention, $+1$ corresponds to the input 0 and -1 to 1. For decoders with binary messages, it is more convenient to define the message alphabet as $\mathcal{M} = \{0, 1\}$. Then a check node c performs XOR operation, and it is said to be satisfied if $\sigma_c = 0$, unsatisfied if $\sigma_c = 1$. The syndrome at ℓ-th iteration is the ordered set $\{\hat{\sigma}_c^{(\ell)}\}_{c\in C}$ obtained from the codeword estimates, i.e., $\hat{\sigma}_c^{(\ell)} = \bigoplus \hat{\mathbf{x}}_{\mathcal{N}_c}^{(\ell)}$. A codeword is found if all M check nodes are satisfied, i.e., if $\bigwedge_{c\in C} \hat{\sigma}_c^{(\ell)} = 0$. The last operation requires an m-input AND gate. The output of the MAJ function is defined straightforwardly.

Due to hardware unreliability the results of the update functions are not always correctly computed. We model this by XOR-ing the perfect output with the binary error e_{MAJ} or $e_\oplus$.

$$\nu_{v\to c}^{(\ell)} = \Phi(\mathbf{m}^{(\ell)}) + e_{MAJ}^{(\ell)}$$
$$\mu_{c\to v}^{(\ell)} = \Psi(\mathbf{n}^{(\ell-1)}) + e_\oplus^{(\ell)}.$$

We consider the von Neumann probabilistic failure model, according to which $e_{\texttt{MAJ}}$ and $e_\oplus$ as well as the channel errors are independent Bernoulli random variables with parameters α_{MAJ}, $\alpha_\oplus$ and α_M. Clearly, a gate with a smaller Bernoulli parameter are more reliable. Note that different realizations of Bernoulli random variables $e_{MAJ}^{(\ell)}$ and $e_\oplus^{(\ell)}$ correspond to different edges. To simplify the notation, the indices $v \to c$ and $c \to v$ are omitted.

As mentioned before, in addition to the logic gates needed for calculation of messages, the decoder also requires logic gates for the final bit estimations and parity-checks calculation performed by the functions $\hat{\Phi}$ and $\hat{\Psi}$. If we allow these gates used in $\hat{\Phi}$ to be unreliable, the performance of the decoder would be determined by the failure probabilities of these gates, not by the error control scheme. Thus these gates must be made perfect.

2.2.3 Trapping Sets for the Noisy Decoder

To illustrate how the logic gate randomness can be beneficial for decoding of the finite length codes and can annihilate trapping sets, we start with the assumption that $\alpha_{MAJ} = 0$, and that errors affect only the XOR gates. Consider the small (3,3) trapping set shown in Fig. 2.1. The variable nodes

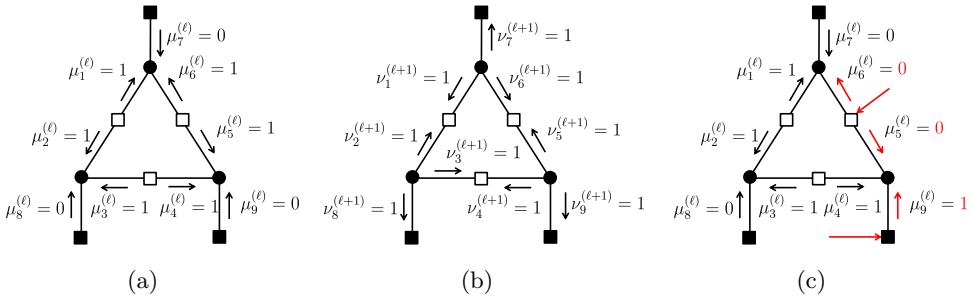


Figure 2.1: Message passing in a noisy (6,3) trapping set. (a) Messages that are sent from checks to variable nodes in the ℓ-th iteration. (b) Messages sent from variable nodes to check nodes in $(\ell+1)$-th iteration assuming perfect MAJ gates. (c) The faulty XOR gates are indicated by thick red arrows on the check nodes and the wrong messages from the noisy checks are shown with red arrows.

in the rest of the graph are correct, and suppose that the check nodes outside the this subgraph are perfect.

The state of a Markov chain at iteration ℓ is defined by the vector of messages from check nodes $\mu^{(\ell)} = (\mu_1^{(\ell)}, \mu_2^{(\ell)}, ..., \mu_{\gamma|T|}^{(\ell)})$, where $\mu_k^{(\ell)}$ for $k = 1, 2, ..., \gamma|T|$ is a message from a check node c to a variable node v where $c \in C_T, v \in T$. In Figure 2.1(a), the messages from check nodes to variable nodes are labeled with μ_k, where $k = 1, 2, ..., 9$. As an example, the state of a Markov chain in Figure 2.1(a) is $(\mu_1^{(\ell)}, \mu_2^{(\ell)}, ..., \mu_9^{(\ell)}) = (1, 1, 1, 1, 1, 1, 0, 0, 0)$.

As an example, suppose the state i of the Markov chain model at the iteration ℓ corresponds to $\mu^{(\ell)} = (\mu_1^{(\ell)}, \mu_2^{(\ell)}, ..., \mu_9^{(\ell)}) = (1, 1, 1, 1, 1, 1, 0, 0, 0)$ as shown in Figure 2.1(a). Assume that we want to find the probability of being at the state j at iteration $\ell + 1$ corresponding to $\mu^{(\ell+1)} = (1, 1, 1, 1, 0, 0, 1, 1, 0)$. It is easy to see that if $\alpha_\oplus = 0$, the messages from the variable nodes at the iteration $\ell + 1$ would be given by the vector $\nu^{(\ell+1)} = (1, 1, 1, 1, 1, 1, 1, 1, 1)$. To see this, note that the messages from degree-1 check nodes to variable node in T are always 0 when there is no faulty XOR gates. Finding the messages from degree-2 check nodes to variable nodes is done by updating from variable nodes to check nodes using $\mu^{(\ell)}$. For instance, $\nu_6^{(\ell+1)}$ is the majority of values $\mu_1^{(\ell)}$ and $\mu_7^{(\ell)}$ that is 1. According to our previous discussion, the probability of transition from the state i to j is $P_{\varepsilon,\delta} = \prod_{\beta=1}^{9} P(\mu_\beta^{(\ell+1)} | \nu_\beta^{(\ell+1)}) = (1 - \alpha_\oplus)^6 {\alpha_\oplus}^3$.

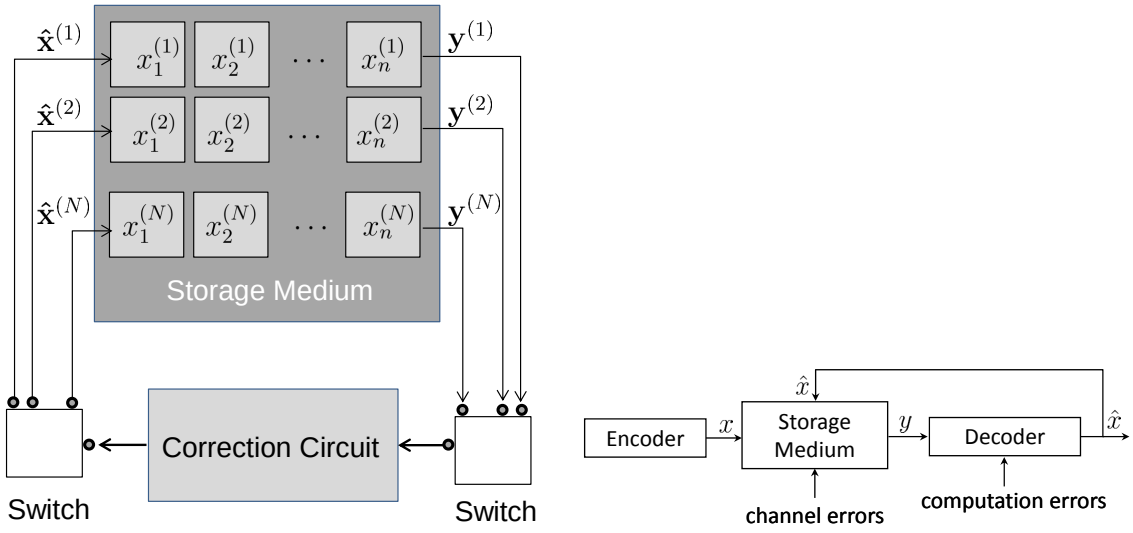
Clearly, the perfect Gallager-B decoder cannot correct the errors in all three variable nodes. However, the elementary trapping sets, for which the degree of each check node is no more than two, can be annihilated if $\alpha_\oplus > 0$. By this we mean that the decoder reaches a state in which the three bits in a cycle are decided as zeros

$$\texttt{MAJ}(\mu_1, \mu_6, \mu_7) = 0, \texttt{MAJ}(\mu_2, \mu_3, \mu_8) = 0, \texttt{MAJ}(\mu_3, \mu_5, \mu_9) = 0.$$

Simple counting reveals that there are 80 out of 2^9 such states. A few examples of such states are $(1, 1, 0, 1, 0, 0, 0, 0, 0)$, $(1, 1, 0, 1, 0, 0, 0, 0, 1)$ and $(1, 1, 0, 1, 1, 0, 0, 0, 0)$.

2.3 The Proposed Storage System Architecture

In our storage system, each k-bit user information is stored as a codeword of an (n, k) linear block code of length n and code rate $R = k/n$. The storage medium contains Nn memory elements and is capable of storing Nk user bits and $N(n - k)$ redundant bits. The memory elements are unreliable and fail transiently and independently of each other - they follow the von Neumann failure model [45]. To prevent data decay, the stored information is periodically refreshed. Each of N codewords is in a round-robin fashion processed by a common correction circuit, whose n-bit output is written back into the corresponding codeword location on the medium as shown in Fig. 2.2(a). The set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$ in Fig. 2.2(a) denotes the set of codewords stored on a medium. Due to errors



(a) (b)

Figure 2.2: A block diagram of an information storage system with error correction coding. (a) A period-$1/N$ round-robin update of codewords. (b) Communications channel model. $\mathbf{x}$ is a codeword stored on a medium, and $\mathbf{y}$ is an observed word. The decoder outputs $\hat{\mathbf{x}}$, an estimate of the stored codeword, which is then written back to the storage medium.

in memory locations, $\mathbf{x}^{(i)}$ is read back as $\mathbf{y}^{(i)}$. The role of the correction circuit is to correct errors in each word $\mathbf{y}^{(i)}$, and write the n-bit codeword estimate $\hat{\mathbf{x}}^{(i)}$ back into the corresponding location on the medium.

At the moment a codeword on the medium is scheduled to be processed, the probability of error in each memory element is α_M. Since the correction of each codeword is independent of others, the storage of the codeword $\mathbf{x}$ can be modeled as transmission through a communication channel (Fig. 2.2(b)). The medium is modelled as the Binary Symmetric Channel (BSC) in which the code bits are flipped independently with probability α_M.

A decoder, which plays a role of the correction circuit, performs computations on noisy logic gates. An obvious way to ensure robustness of a decoder is to employ the von Neumann multiplexing [45]. However, this comes with a price of very large redundancy because the von Neumann redundant design does not take into account the specifics of the decoding algorithm. The first attempt to use a more advanced coding scheme to ensure fault tolerance of storage systems made from unreliable components is due to Taylor [24] and Kuznetsov [46]. Fault-tolerant decoding and storage has attracted significant attention lately, and numerous approaches have been proposed which exploit the inherent redundancy of the existing decoders [6, 23, 26, 30]. It is not obvious that such noisy decoders can ensure reliable storage of information because more powerful decoders – capable of correcting more errors in the medium – may require more logic gates, thus introducing more computational errors. In light of that, it is even more surprising that the logic gate errors may help the decoder. We proceed to show exactly this. We present a class of noisy decoders that perform better than their perfect counterparts.

The First Ingredient: An LDPC Code. Our approach is based on a special type of linear block codes, known as low-density parity check (LDPC) codes and on their iterative decoding [47]. Our method is applicable to any LDPC code type, but for the sake of clarity, we mostly discuss a subclass of the (γ, ρ)-regular LDPC code ensemble [47]. The integer parameters γ and ρ determine code rate $R \geq 1 - \frac{\gamma}{\rho}$, and the structure of the decoder, namely the number of input arguments to Boolean functions used in the decoder. Applicability of our method to the irregular codes is illustrated in Section 4.4.

A codeword of an LDPC code, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, is stored on the medium, which is modeled as

the Binary Symmetric Channel (BSC), as shown in Fig. 2.2(b). In other words, the stored bits are subject to random and independent bit flips which occur with probability α_M. Thus, $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, the word that is read back from the storage medium, can be expressed as $\mathbf{y} = \mathbf{x} + \mathbf{e}$, where + denotes XOR operation, and the elements of the vector $\mathbf{e}$ are independent, Bernoulli random variables with parameter α_M. The vector $\mathbf{y}$ is input to the iterative decoding algorithm whose goal is to recover $\mathbf{x}$. The decoder must maintain the low probability of not recovering $\mathbf{x}$ - the so-called the frame error rate (FER).

The Second Ingredient: Noisy Gallager-B Message Updates. The iterative decoding algorithm [48] operates on a graphical representation of a code $G = (V \cup C, E)$. The code graph G is a bipartite graph whose edges in the set E connect the variable nodes in the set V with check nodes in C. The decoding algorithm consists of sending messages between variable nodes $v \in V$ corresponding to code bits and check nodes, $c \in C$ corresponding to parity check equations in which the variables (bits) are involved.

The messages $\nu^{(\ell)}$ from variable nodes and $\mu^{(\ell)}$ from check nodes in the ℓ-th iteration are computed as

$$\begin{aligned} \nu^{(\ell)}_{v\to c} &= \Phi(y_v, \mathbf{m}^{(\ell)}) + e^{(\ell)}_{MAJ} \\ \mu^{(\ell)}_{c\to v} &= \Psi(\mathbf{n}^{(\ell-1)}) + e^{(\ell)}_{\oplus}. \end{aligned} \tag{2.2}$$

where $\mathbf{m}^{(\ell)} = \mu^{(\ell)}_{\mathcal{N}_v \backslash c\to v}$ denote the incoming messages to a variable node v from all its neighbors $\mathcal{N}_v$ except a message from the check node c. Similarly, $\mathbf{n}^{(\ell)} = \nu^{(\ell)}_{\mathcal{N}_c \backslash v\to c}$ denote all incoming messages to the check node c from its neighbors except the variable node v.

Various choices of the update functions Φ and Ψ are considered in literature. Our decoding algorithm is based on the modification of the Gallager-B algorithm [47], where the update functions Φ and Ψ require only two types of logic gates: $(\gamma-1)$-input majority logic (MAJ) gates and $(\rho-1)$-input XOR gates [49]. These gates produce wrong output with probabilities $\alpha_\oplus$ and α_{MAJ}, respectively. Note the logic gate errors can occur because of gate unreliability, but can be also deliberately inserted with a goal to improve the performance of the decoder. With some abuse of notation, $e^{(\ell)}_{MAJ}$ and $e^{(\ell)}_{\oplus}$ in Eq. (2.2) denote the errors in majority logic and XOR gates affecting computation of the messages $\nu^{(\ell)}_{v\to c}$ and $\mu^{(\ell)}_{c\to v}$, respectively (we drop the indices $v \to c$ and $c \to v$ in e_{MAJ} and $e_\oplus$). If the decoder was deterministic, then $e^{(\ell)}_{MAJ}$ and $e^{(\ell)}_{\oplus}$ in Eq.(2.2) would be zero. For noisy decoders, they are independent Bernoulli random variables with parameters α_{MAJ} and $\alpha_\oplus$, respectively.

The decided bit value $\hat{x}^{(\ell)}_v$ in the ℓ-th iteration is calculated as $\hat{x}^{(\ell)}_v = \hat{\Phi}(\mathbf{l}^{(\ell)})$, where the γ-tuple $\mathbf{l} \in \{\mathcal{M}\}^\gamma$ represents all incoming messages to the variable node v, and $\hat{\Phi}$ is the decision function. An estimate of the check node value c in the ℓ-th iteration, $\hat{\sigma}^{(\ell)}_c$ is obtained by XOR-ing the decisions values of all neighboring variables $\mathcal{N}_c$ using the function $\hat{\Psi}$. The functions $\hat{\Phi}$ and $\hat{\Psi}$ are defined and explained in Section 4.2. The iterative procedure in Eq. (2.2) is halted when all parity checks are satisfied or the predefined maximum number of iterations, L, is reached. The decoding is called successful if a codeword (either correct or wrong) is found. Otherwise, the decoding is said to have failed. The event of producing a codeword estimate which is a wrong codeword is called miss-correction.

The Third Ingredient: Rewinding. To allow the decoder to benefit from errors, large number of iterations is needed under some conditions of gate unreliability. However, too many logic gate errors can overwhelm the decoder, and lead to miscorrection. In addition to the Gallager-B update rules given in Eq. 2.2, our decoder is equipped with the following key feature which prevents the accumulation of errors in the messages when the number of iterations is large. If a codeword is not found after L_R iterations, $L_R \ll L$, the decoding algorithm is re-initialized with the word received from the channel. Instead of running the whole L iterations, the decoder instead runs $r = L/L_R$ very short rounds. A decoder with such rewinding schedule is referred to as the rewind-decoder and denoted by $\mathcal{F}^{\circlearrowleft^r}(L_R)$. We write this as $\mathcal{F}^{\circlearrowleft^r}(L_R) = \mathcal{F}^{\circlearrowleft^1}(L_R)\Diamond\mathcal{F}^{\circlearrowleft^1}(L_R)\Diamond\cdots\Diamond\mathcal{F}^{\circlearrowleft^1}(L_R)$, where $\Diamond$ denotes the

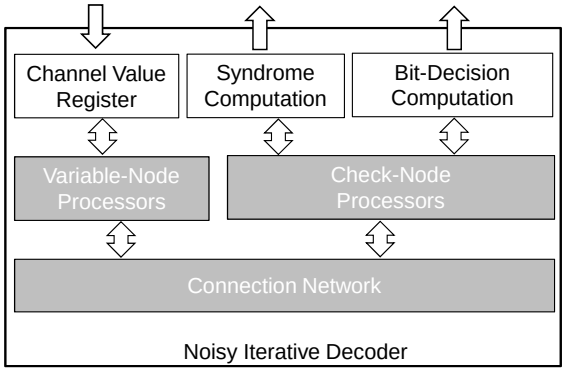


Figure 2.3: A block diagram of a noisy decoder. Shaded blocks are made of unreliable components, while white blocks are reliable.

rewinding schedule, and the last expression has r terms. Clearly, the plain noisy Gallager-B decoder with no rewinding, denoted by $\mathcal{F}(L)$, is a special case of the rewinding decoder, $\mathcal{F}(L) = \mathcal{F}^{\circlearrowleft^1}(L)$. Therefore,

$$\mathcal{F}^{\circlearrowleft^r}(L_R) = \underbrace{\mathcal{F}(L_R)\Diamond\mathcal{F}(L_R)\Diamond\cdots\Diamond\mathcal{F}(L_R)}_{r}. \tag{2.3}$$

Critical Gates Must be Perfect. Note that the logic gates gates used to extract the bits from any decoder must be perfect, otherwise the error rate would be bounded by the reliability of these external gates [48]. Thus, the majority-logic gates in the decoder's decision logic (the function $\hat{\Phi}$) are made of perfect gates. We also assume that the effect of errors in the encoder in Fig. 2.2(b) are incorporated in the value of the memory element error probability α_M.

The register inside the decoder which temporarily stores the word read from a memory medium (the channel values) is also reliable. This is necessary because otherwise the codeword estimate would drift away from the true codeword in the course of decoding, as iterations progress. Registers for storing the intermediate results of computations of Φ and Ψ are unreliable, and their unreliability is accounted for in α_{MAJ} and $\alpha_{\oplus}$. The blocks performed on perfect gates and those performed on noisy gates are shown in Fig. 2.3.

As syndrome checker (the function $\hat{\Psi}$ together with the m-input AND gate) is used as a decoding halting criterion, it must be also made perfect. To reduce power consumption in a decoder, we may want to perform syndrome checking not in every iteration but according to a predefined schedule. The optimal schedule is beyond scope of this chapter, and we will simply adopt a schedule in which the decoder runs on noisy hardware for maximum of L iterations, during which the (perfect) syndrome checker is used only in the a few first iterations and in the last Z iterations.

The decoding is halted in the first of these Z iterations in which a codeword is found. The Z/L ratio determines the decoding efficiency, thus is kept low. In the rewinding schedule, the syndrome checker is used in all L_R iterations in the first decoding round, and after the first rewinding it is used in the last Z iterations in each round.

2.4 Performance Evaluation of Noisy Gallager-B Decoder

To characterize the FER performance of a given faulty decoder $\mathcal{F}$, we need to compute the probability $\mathtt{Pr}\{\hat{\mathbf{x}}^{(\ell)} \neq \mathbf{x}\}$ and find how it varies with the parameters α_M, $\alpha_{\oplus}$, α_G, L and Z, and, for the rewind-decoder $\mathcal{F}^{\circlearrowleft^r}$, also with the parameters of L_R and r. We would also like to find the average number of iterations needed for a decoder to converge to a codeword. The goal is to find a region in this 7-dimensional design space in which $\mathcal{F}$ and $\mathcal{F}^{\circlearrowleft^r}$ outperform their perfect counterpart $\overline{\mathcal{F}}$.

We now introduce a Markov model which facilitates this analysis. We first derive the model for the entire code graph, and then show that it can be simplified and applied to specific code subgraphs while keeping its accuracy. Let $\boldsymbol{\nu}^{(\ell)} = (\nu_v^{(\ell)})_{v\in V}$ be the ordered set of all messages from variable nodes

in iteration ℓ, and $\boldsymbol{\mu}^{(\ell)} = (\mu_c^{(\ell)})_{c\in C}$ be the outgoing messages from check nodes. From Eq. (2.2), they can be expressed as

$$\begin{aligned}\boldsymbol{\nu}^{(\ell)} &= \Upsilon_V(\boldsymbol{\nu}^{(\ell-1)}) + \mathbf{e}_{\boldsymbol{\nu}}^{(\ell)} \\ \boldsymbol{\mu}^{(\ell)} &= \Upsilon_C(\boldsymbol{\mu}^{(\ell-1)}) + \mathbf{e}_{\boldsymbol{\mu}}^{(\ell)}.\end{aligned} \tag{2.4}$$

where the functions Υ_V and Υ_C are the compositions of Φ and Ψ, and define the dynamical system of the perfect decoder. The binary vectors $\mathbf{e}_{\boldsymbol{\nu}}^{(\ell)}$ and $\mathbf{e}_{\boldsymbol{\mu}}^{(\ell)}$ of length $n\gamma$ $(= m\rho)$ are the realization of errors at time ℓ that affect computation of messages $\boldsymbol{\nu}$ and $\boldsymbol{\mu}$, respectively. Their elements are deterministic functions of Bernoulli random variables representing the errors in the MAJ and XOR gates, and are time invariant but not independent. Probability distributions of $\mathbf{e}_{\boldsymbol{\nu}}$ and $\mathbf{e}_{\boldsymbol{\mu}}$ can be determined using elementary probability.

From the discussion above, the random processes $\{\boldsymbol{\nu}^{(\ell)}\}_{\ell\geq 0}$ and $\{\boldsymbol{\mu}^{(\ell)}\}_{\ell>0}$ are both homogenous Markov chains with finite state spaces. For the analysis purposes, it is sufficient to consider only the random process $\{\boldsymbol{\mu}^{(\ell)}\}_{\ell>0}$.

For a given a decoder $\mathcal{F}$ and error pattern $\mathbf{e}$ in the memory elements, consider the Markov chain $\mathcal{W}_{\mathbf{e}}$, with the state space $S = \{0,1\}^{|C|\rho}$, and the transition probability matrix $P = (p_{\varepsilon,\delta})_{\varepsilon,\delta\in S}$. The transition probabilities between states, $p_{\varepsilon,\delta} = \mathtt{Pr}\{\boldsymbol{\mu}^{(\ell)} = \delta | \boldsymbol{\mu}^{(\ell-1)} = \varepsilon\}$, depend on $\alpha_\oplus$ and α_{MAJ}. Since Φ is the function of the memory output $\mathbf{y}$, Υ_C also depends on $\mathbf{y}$, thus the transition probabilities depend on the the channel error vector $\mathbf{e}$, and for a given decoder we have an ensemble of Markov chains $\{\mathcal{W}_{\mathbf{e}}\}_{\mathbf{e}\in\{0,1\}^n}$.

Due to independence of logic gate errors, their effect can be combined into a single conditional probability α_G. Let $\bar{\delta} = \Upsilon_C(\varepsilon)$ be the state of the perfect decoder $\overline{\mathcal{F}}$ reached from the state ε. Then

$$p_{\varepsilon,\delta} = p_{\varepsilon,\bar{\delta}} \alpha_G^{d_{\bar{\delta},\delta}} (1-\alpha_G)^{|C|\rho - d_{\bar{\delta},\delta}}, \tag{2.5}$$

where $d_{\bar{\delta},\delta}$ is the Hamming distance between the binary vectors $\bar{\delta}$ and δ, and α_G is the probability that a single XOR gate output in a noisy decoder $\mathcal{F}$ is different from the corresponding XOR gate output in the perfect decoder $\overline{\mathcal{F}}$,

$$\alpha_G = (1-\alpha_\oplus)(1-(1-2\alpha_{MAJ})^{\rho-1}) + \alpha_\oplus(1+(1-2\alpha_{MAJ})^{\rho-1}).$$

The term multiplying $(1-\alpha_\oplus)$ is the occurrence probability of an odd number of MAJ gate errors, while the term multiplying $\alpha_\oplus$ is the probability of even number of MAJ gate errors.

Note that for small logic gate error rates, the above expression can be approximated by

$$\alpha_G \approx (\rho-1)\alpha_{MAJ} + \alpha_\oplus.$$

Let $\pi_\beta^{(\ell)} = \Pr\{\boldsymbol{\mu}^{(\ell+1)} = \beta\}$ for some $\beta \in S$, and let $\boldsymbol{\pi}^{(\ell)} = (\pi_\beta^{(\ell)})_{\beta\in S}$. The initial distribution $\boldsymbol{\pi}^{(0)}$ can be readily found from error statistics in the storage medium and is a function on the memory error vector $\mathbf{e}$, α_M and $\alpha_\oplus$. For the component at the position $c \to v$ we have

$$\mathtt{Pr}\{\mu_{c\to v}^{(1)} = 1\} = (1-\alpha_\oplus)(1-(1-2\alpha_M)^{\rho-1}) + \alpha_\oplus(1+(1-2\alpha_M)^{\rho-1}).$$

Let the vectors $\hat{\mathbf{x}}^{(\ell)}$ and $\hat{\sigma}^{(\ell)}$ be the variable node decisions and check node estimates in iteration ℓ. If they were computed by noisy gates, the probability of convergence to a codeword would be close to zero for any iteration ℓ. This follows from from from Eq. (2.4) and the fact that $\boldsymbol{\pi}^{(\ell)} = \boldsymbol{\pi}^{(0)} P^\ell$. This explains the requirement for using perfect gates to compute the variable node estimates by the perfect gates.

It is instructive to classify the states of $\mathcal{W}_{\mathbf{e}}$ with respect to their closeness to codewords. Due to channel, decoder, and the gate-error mechanism symmetries (each one treats the binary zeros and ones equally), the decoder behavior is independent on the readback codeword $\mathbf{y}$. However, it depends on the error pattern in memory elements, $\mathbf{e}$. Furthermore, since the code is linear, for the FER analysis it is sufficient to consider the all-zero codeword [50], $\mathbf{x} = \mathbf{0}$. Let $S_{\mathbf{0}} \subset S$ denotes the subset

of states for which all parity check are satisfied and the variable node decisions form the all-zero codeword $\mathbf{0}$. Similarly, $S_{\sim\mathbf{0}}$ denotes the set of states for which all parity check are satisfied, and the variable node decisions form a non-zero codeword. The set $S_{\sim\mathcal{C}}$ includes all states for which the variable node decisions are not codewords. Thus, the above three disjoint sets partition the set of states $S = S_{\mathbf{0}} \cup S_{\sim\mathbf{0}} \cup S_{\sim\mathcal{C}}$. We underline that the Markov chain and hence its any state partition depends on $\mathbf{e}$.

FER Performance and Time to Absorption for a Given Error Pattern. For a given error pattern $\mathbf{e}$, the frame error probability, and the conditional miscorrection probability of the decoder $\mathcal{F}$, (i.e. miscorrection error rate (MER)) in the iteration ℓ, $\Pr\{\hat{\mathbf{x}}^{(\ell)} \neq \mathbf{x}, \hat{\sigma}^{(\ell)} = \mathbf{0}\}$ can be now found from $\mathcal{W}$ and expressed as

$$FER_{\mathbf{e}}^{(\ell)}(\mathcal{F}) = \Pr\{\boldsymbol{\mu}^{(\ell)} \in S_{\sim\mathbf{0}} \cup S_{\sim\mathcal{C}}\}$$
$$MER_{\mathbf{e}}^{(\ell)}(\mathcal{F}) = \Pr\{\boldsymbol{\mu}^{(\ell)} \in S_{\sim\mathbf{0}}\}.$$

Theorem 1. *For a noisy Gallager-B decoding algorithm $\mathcal{D} = \mathcal{F}(L, L)$ on any LDPC code $\mathcal{C}$, $\exists L^*$ and Δ, which depends on L^*, such that $\forall L > L^*$*

$$|FER_{\mathbf{e}}^{(L)}(\mathcal{D}) - MER_{\mathbf{e}}^{(L)}(\mathcal{D})| < \Delta. \tag{2.6}$$

The average number of iterations to absorption to the states in $S_{\mathbf{0}}$ and $S_{\sim\mathbf{0}}$ can be calculated from the Markov chain $\mathcal{W}_{\mathbf{e}}$. If we combine all states in $S_{\mathbf{0}}$ into a single state, and do the same for $S_{\sim\mathbf{0}}$, we end up a with a reduced Markov chain denoted by $\mathcal{M}_{\mathbf{e}}$. Its transition probability matrix

$$P = \begin{pmatrix} I_2 & \mathbf{0} \\ R & Q \end{pmatrix} \tag{2.7}$$

can be obtained from that of $\mathcal{W}_{\mathbf{e}}$ by summing up the corresponding rows as explained in Appendix A. $\mathcal{M}_{\mathbf{e}}$ is also homogenous and absorbing but has only two absorbing states, one corresponding to the correct decision, and and one corresponding to miscorrection. The matrix Q in Eq. (2.7) is a transition probability matrix between the transient states in $S_{\sim\mathcal{S}}$ with no zero entries, and I_2 is the 2×2 identity matrix. The transition probabilities from transient to absorbing states are given by the matrix $R = (R_{\mathbf{0}}, R_{\sim\mathbf{0}})$. The fundamental matrix of the absorbing chain, $N = (I - Q)^{-1}$, determines the average times to absorption from different transient states. More specifically, $\sum_{\delta \in S_{\sim\mathcal{C}}} N_{\beta,\delta}$ is the average time to absorbtion from the transient state β.

Average FER Performance. We have shown that the Markov chain model allows us to determine these probabilities. By averaging over all error patterns, we obtain the average FER as

$$FER(\mathcal{D}) = \sum_{\mathbf{e} \in \{0,1\}^n} \mathtt{Pr}\{\mathbf{e}\} \times FER_{\mathbf{e}}(\mathcal{D}). \tag{2.8}$$

Note that the Eq. (2.6) is valid for a given error pattern $\mathbf{e}$, and it translates to the averages (i.e., $FER^{(L)} = MER^{(L)}$) for infinite L. However for finite L, the average $FER^{(L)}(\mathcal{D})$ and $MER^{(L)}(\mathcal{D})$ in Eq. 2.8 can differ significantly.

The probability of the error pattern $\mathbf{e}$, $\mathtt{Pr}\{\mathbf{e}\}$, depends on its Hamming weight $w(\mathbf{e})$. In the case of transient i.i.d. memory errors occurring with probability α_M, the probability $\mathbf{e}$, $\mathtt{Pr}\{\mathbf{e}\}$ can be expressed as

$$\mathtt{Pr}\{\mathbf{e}\} = \alpha_M^{w(\mathbf{e})}(1 - \alpha_M)^{n - w(\mathbf{e})}. \tag{2.9}$$

For a perfect decoder for which $\alpha_{MAJ} = 0$ and $\alpha_{\oplus} = 0$, transitions between states are deterministic, and the attractor basin of a dynamical system ($\boldsymbol{\mu}^{(\ell)} = \Upsilon_V(\boldsymbol{\mu}^{(\ell-1)})$ in Eq. (2.4) includes the codewords - which are the fixed points - and trapping sets, which can be either fixed points or cycle attractors. Perfect decoder may oscillate between these states, thus failing to converges to a codeword. On the other hand, in a noisy decoder - where α_{MAJ} and/or $\alpha_{\oplus}$ are nonzero - every state can be reached

with a nonzero probability. Thus, the faulty decoder will eventually converge to a codeword - either correct or incorrect one.

In the case when the decoding algorithm have small probability of miscorrection in the first decoding iterations, it is better to use the rewinding decoder with $r = L/L_R$ rounds, where the restarts and re-initializations are performed after L_R iterations, $L_R << L$. We denote this decoder by $\mathcal{D}^{\circlearrowleft} = \mathcal{F}^{\circlearrowleft^r}(L_R)$. From Eq. (2.3), it follows that the rewinding decoder is a composition of r rounds of the non-rewinding decoder $\mathcal{D} = \mathcal{F}(L_R)$ which runs for L_R iterations, with Z iterations of syndrome checking, i.e., $\mathcal{D}^{\circlearrowleft} = \mathcal{D}\Diamond\mathcal{D}\Diamond\cdots\Diamond\mathcal{D}$. This fact allows us to to obtain the miscorrection probabilities for every particular error pattern. The restart is performed only in the case of decoding failure, i.e. when the syndrome is not equal to zero. Therefore, the miscorrection probability of this decoder is

$$MER_{\mathbf{e}}(\mathcal{D}^{\circlearrowleft}) = \sum_{\ell=1}^{r} MER_{\mathbf{e}}(\mathcal{D}) \left(FER_{\mathbf{e}}(\mathcal{D}) - MER_{\mathbf{e}}(\mathcal{D})\right)^{\ell-1}, \tag{2.10}$$

and the frame error rate after r rewinds includes the cases of miss-corrections in all r rounds rounds and the case when the syndrome is not zero in all iterations. Therefore,

$$FER_{\mathbf{e}}(\mathcal{D}^{\circlearrowleft}) = MER_{\mathbf{e}}(\mathcal{D}^{\circlearrowleft}) + \left(FER_{\mathbf{e}}(\mathcal{D}) - MER_{\mathbf{e}}(\mathcal{D})\right)^{r}. \tag{2.11}$$

Finally, the average FER is obtained as

$$FER(\mathcal{D}^{\circlearrowleft}) = \sum_{\mathbf{e}} \mathtt{Pr}\{\mathbf{e}\} FER_{\mathbf{e}}(\mathcal{D}^{\circlearrowleft}). \tag{2.12}$$

Note that in the above discussion it is assumed that the perfect syndrome checker was used in every iteration. However, due to power consumption reasons, the perfect checker might be used in only last Z out of L iterations. The Markov chain corresponding to this case is not absorbing in the first $L-Z$ iterations, and for large enough L, there is nonzero probability to reach any state. The state probabilities after turning on the syndrome checker in the $L-Z+1$ iteration are $\boldsymbol{\pi}^{(L-Z)} = \boldsymbol{\pi}^{(0)} P^{L-Z}$, and act as the initial distribution for the absorbing Markov chain $\mathcal{W}$ in the last Z iterations.

While the ensemble of Markov chains $\{\mathcal{W}_{\mathbf{e}}\}$ completely determines the decoder's FER and MER performance on an entire code graph, its state space S is large, making the above approach numerically inefficient. However, the theory of perfect iterative decoders, gives the code graph topologies - known as trapping sets - responsible for decoding failures. The exact characterization of the relationship between the FER of an LDPC code and its trapping sets is known to be a formidable task. Thus, instead of considering the entire code graph, the decoder performance can be estimated by its ability to escape from Markov chain states corresponding to dominant trapping sets, as we explain next.

2.5 Numerical Results

Code (5,1), analytical approach

First, we present performance analysis of a short irregular code defined with the parity check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

with the codeword length n=5, and constant row weight (all check nodes have degree ρ_i=2). It is variable-node irregular and there are two variable nodes with degree three ($\gamma_1 = \gamma_5 = 3$) and three nodes with degree two ($\gamma_2 = \gamma_3 = \gamma_4 = 2$). It is easy to check that $\bigwedge_{c\in C} \hat{\sigma}_c^{(\ell)} = 0$, i.e., syndrome is

verified only if $\hat{\mathbf{x}}^{(\ell)} = \mathbf{0}$ or $\hat{\mathbf{x}}^{(\ell)} = \mathbf{1}$, i.e., there are only two codewords: "all-zero" codeword and "all-one" codeword. In fact, this is a repetition code with minimum Hamming weight $d_{min} = 5$, allowing the maximum likelihood (ML) decoder to correct all error patterns of weight $t = 2$ or less (this is a perfect code which attains the sphere packing bound). At the input of the decoder 32 different received words $\mathbf{y}$ can appear. As mentioned before, for the case of BSC and symmetric decoders, we can assume that the all-zero codeword is transmitted to analyze the effect of errors.

As we showed previously, if the noisy Gallager B decoder $\mathcal{F}(L)$ is applied, the decoding process can be described with Markov model where the states are defined with the massages send from check to the variable nodes. In this particular case, every state can be represented as a binary vector of length $\sum_{i=1}^{n} \gamma_i = 12$, and total number of states is $2^{12} = 4096$.

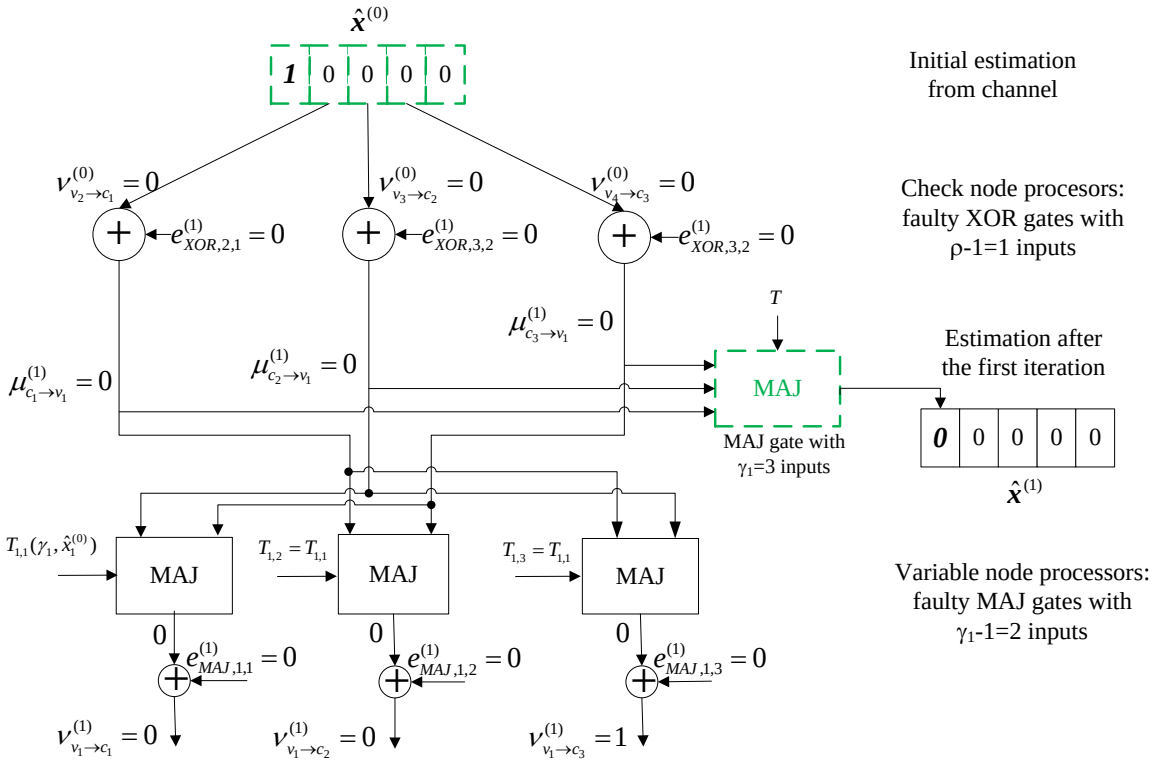


Figure 2.4: Faulty Gallager-B decoder with failures in MAJ and XOR gates, illustration for updating the first variable node in the first decoding iteration, corresponds to the graph representation from Figure 2.5.

The corresponding hardware realization is illustrated in Fig. 2.4, and the corresponding massages passed between the nodes in the bipartite graph are presented in Fig. 2.5. The all-zero codeword is transmitted, and it is assumed that an error appears only in the variable node v_1. Thus the received word is $\hat{\mathbf{x}}^{(0)} = (1, 0, 0, 0, 0)$. According to this initial codeword estimate, all variable nodes send its initial values to the check nodes.

In the case when the decoder is perfect, the XOR gates with $\rho_j - 1 = 1$ inputs generate massages $\mu^{(1)}_{c_j\to v_i}$ that are sent from the j-th check node to the neighboring variable nodes, as illustrated in Figure 2.5(a), and the corresponding state vector is $\boldsymbol{\mu}^{(0)} = (000, 10, 10, 10, 000)$. According to the messages $\mu^{(1)}_{c_j\to v_i}$, the codeword estimate after the first iteration is obtained as $\hat{\mathbf{x}}^{(1)} = (0, 0, 0, 0, 0)$, by using the perfect γ_i-input MAJ gate. The decision rule of the γ_i-input MAJ gate is slightly changed. If all inputs of the MAJ gate are the same, the output is equal to the values of the inputs. If this condition is not satisfied, the threshold is decremented and it is allowed that one input can be different than the other to make the decision. If more than one input is different than the others, the threshold is further decremented until the decision is made. As the threshold does not depend on the column weight, in the case of irregular codes this decision rule favorize the inverting of the variable nodes with

the larger column weight, that can improve the decoder performance.

In the variable nodes, the MAJ gates with $\gamma_i - 1$ inputs generate massages $\nu^{(1)}_{v_i \to c_j}$ which are sent to the check node neighbors. As explained in Section 4.2, the initial estimate from the channel appear at the output of a MAJ gate if number of zeros and ones at its inputs is equal. In the next iteration, massages $\mu^{(2)}_{c_j \to v_i}$ are send from the check nodes to the variable nodes, the corresponding state is $\boldsymbol{\mu}^{(1)} = (000, 00, 00, 00, 111)$, and the codeword estimate after the second iteration is obtained as $\hat{\mathbf{x}}^{(2)} = (0, 0, 0, 0, 1)$. In the subsequent iterations, the outputs of the $(\gamma_i - 1)$ XOR gates define inputs of the $(\gamma_i - 1)$ MAJ gates and vice versa. In further iterations, the codeword estimates will be $\hat{\mathbf{x}}^{(3)} = (0, 0, 0, 0, 0)$, $\hat{\mathbf{x}}^{(4)} = (1, 0, 0, 0, 0)$, etc.

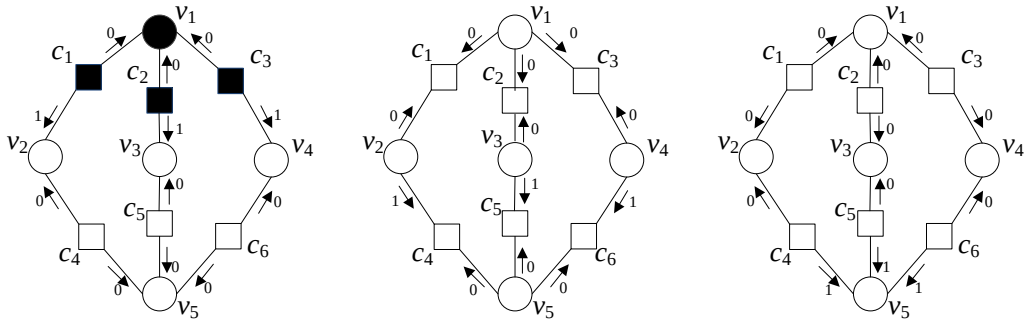


Figure 2.5: Noisy Gallager-B decoding for the example shown in Fig. 2.4. (a) Check to variable update, $\ell = 1$; (b) Variable to check update, $\ell = 1$; (c) Check to variable update, $\ell = 2$

In other words, in even iterations the codeword estimate will be incorrect and correspond to the sent codeword, while in odd iterations the codeword estimate will result in a zaro-syndrome and will be correct. This illustrates that increasing the maximum number of iterations L does not necessarily result in a successful decoding if the syndrome is checked in only in one (last) iteration. In the above case, successful decoding is possible only if the syndrome is checked (by the perfect checker) in no less than two successive iterations. In general, the minimal required number of consecutive syndrome checks may vary for different error patterns.

In the above example, we have assumed that all decoder components are perfect. Thus, the next state is completely determined by the check node and variable node processor functions. For example, if the current state of this Markov model is $\boldsymbol{\mu}^{(\ell)} = \varepsilon = (000, 10, 10, 10, 000)$ and the next state $\boldsymbol{\mu}^{(\ell+1)} = \bar{\delta} = (000, 00, 00, 00, 111)$ is reached with probability one, i.e. $\mathtt{Pr}\{\boldsymbol{\mu}(\ell + 1) = \delta | \boldsymbol{\mu}(\ell + 1) = \varepsilon\} = 1$, while the other transitions are forbidden.

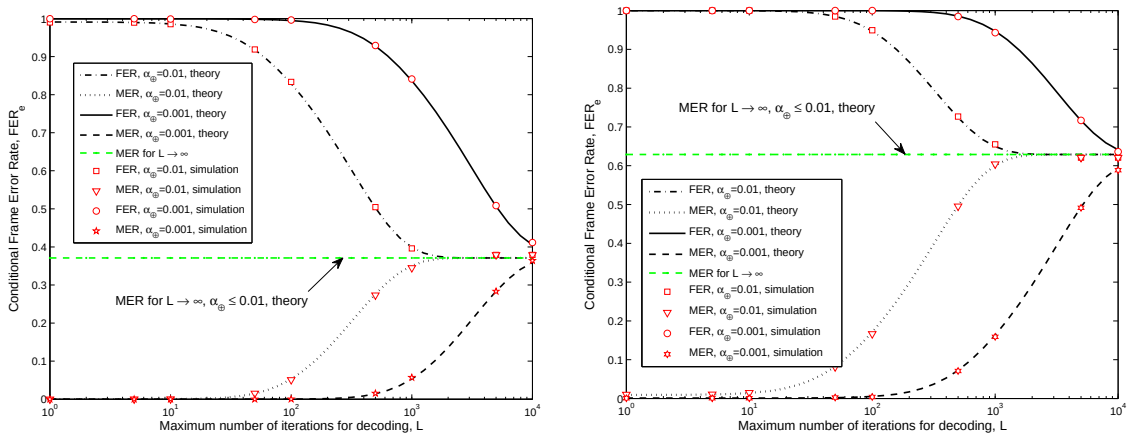
In the case of noisy Gallager-B decoder, the $(\rho_j - 1)$-input XOR gates which generate the massages $\mu^{(\ell)}_{c_j \to v_i}$ and $(\gamma_i - 1)$-input MAJ gates which generate massages $\nu^{(\ell)}_{v_i \to c_j}$ may be faulty, as well as the corresponding register where these messages are stored in the decoder. It is only assumed that the γ_i-inputs MAJ gate which produces the codeword estimation is perfect, as well as the syndrome checker logic. For such a noisy decoder, all other states $\boldsymbol{\mu}(\ell + 1) = \delta$ can be reached in the next iteration in addition to the states in $\boldsymbol{\mu}(\ell + 1) = \bar{\delta}$. The transition probabilities $\mathtt{Pr}\{\boldsymbol{\mu}(\ell + 1) = \delta | \boldsymbol{\mu}(\ell + 1) = \varepsilon\}$ are given in Eq. (2.5).

Probability that a specific error pattern is decoded by using the noisy Gallager-B algorithm in the ℓ-th iteration can be obtained from the transition probability matrix of the Markov chain and input error pattern received from memory which defines the initial state of the Markov chain. As we have shown in Section 2.4, by using a theory of absorbing Markov chains it is possible to analytically determine the conditional FER and MER of a noisy decoder $\mathcal{F}(L, Z)$ for any memory error vector $\mathbf{e}$. By averaging over all error patterns, we obtain FER from Eq. (2.8), and the average MER can be calculated similarly.

It is clear that some patterns correctable by the perfect decoder may become uncorrectable by

the noisy decoder, but that some uncorrectable patterns may become correctable by a noisy decoder with some probability. Now, we will illustrate the accuracy of the proposed analytical model based on absorbing Markov chains, by comparing with the results obtained by using Monte Carlo simulations. Also, we will show for a few illustrative error patterns and failure rates that the conditional FER is lower bounded with the corresponding conditional MER, and if ℓ tends to infinity we obtain $FER_{\mathbf{e}}^{(\infty)} = MER_{\mathbf{e}}^{(\infty)}$ (as stated in Theorem 1). As it is previously shown that errors and XOR and MAJ gates have the similar effect to the decoder performance, for the reason of simplicity, we will assume that only XOR gates are faulty with probability $\alpha_{\oplus}$.

First, we will consider weight-two error pattern $\mathbf{e}$=(11000), uncorrectable by using the perfect Gallager-B algorithm. As it is shown in Fig. 2.6(a), increase of the number of iterations results in better performance for the certain values of the failure rates. Probability that the pattern is not decoded converge to the corresponding MER, for the infinite number of the decoding iterations. Also, it can be noticed that the convergence speed is increased for larger failure rates α_G. We also consider weight-three error pattern $\mathbf{e} = (00111)$, uncorrectable by using the perfect Gallager-B algorithm. This pattern is uncorrectable also (with probability one) by using perfect ML decoder, but the numerical results presented in Fig. 2.6(b) indicate that there is non-zero probability to correct it by using noisy Gallager-B decoder. Although the probability of miscorrection is larger than probability of correct decoding, this effect is not negligible. If the failure rate in the logic gates is lower, more decoding iterations are required for successive decoding but probability that the pattern is not corrected always converge to the MER. Numerical results obtained by using the proposed analytical approach perfectly corresponds to the simulation results.



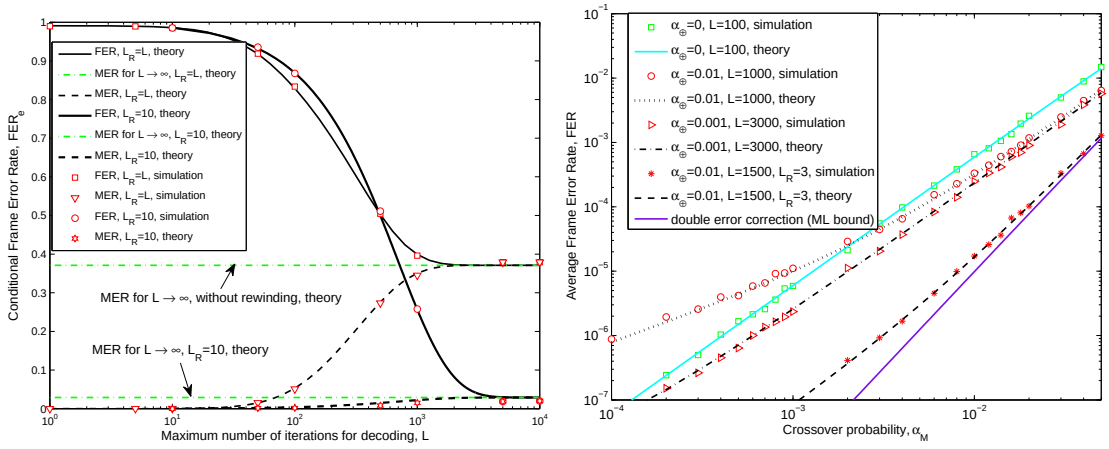
(a) (b)

Figure 2.6: Conditional FER and MER for error patterns, uncorrectable by using perfect decoder, as a function of maximum number of iterations: (a) Error pattern $\mathbf{e} = (11000)$, (b) Error pattern $\mathbf{e} = (00111)$.

In Fig. 2.7(a) we show that the rewinding reduces miscorrection probability. As MER lower bounds FER, the decoder performance can be improved for any error pattern if rewinding is applied with the appropriately chosen rewinding period L_R. If the rewinding is applied in the moment where the probability of the miscorrection is negligible and the probability of correct decoding is not negligible, this will affect the overall performance after rewinding. As the numerical values for conditional FER and MER are known for the particular error pattern are known for all iterations, the optimal value of parameter L_R can be estimated.

Average FER as a function of crossover probability is presented in Fig. 2.7(b). If the failure rate is high, it can help us to decode some high-weight patterns uncorrectable by using perfect decoder but the MER for low-error patterns is increased, and some error patterns correctable by using perfect decoder are now uncorrectable with high probability. Therefore, in the case of high failure rates

performance in error-floor region is typically poor. For the decreased failure rate, the average FER will be reduced for a wide range of α_M, with the price of increased number of iterations (effect from Fig. 2.6(a)). If the failure rate in logic gates is high, the decoder performance can be significantly improved by using the rewinding procedure. For the noisy decoding with $\alpha_{\oplus} = 0.01$, rewinding after $L_R = 3$ iterations result in performance close to the ML bound.



(a) (b)

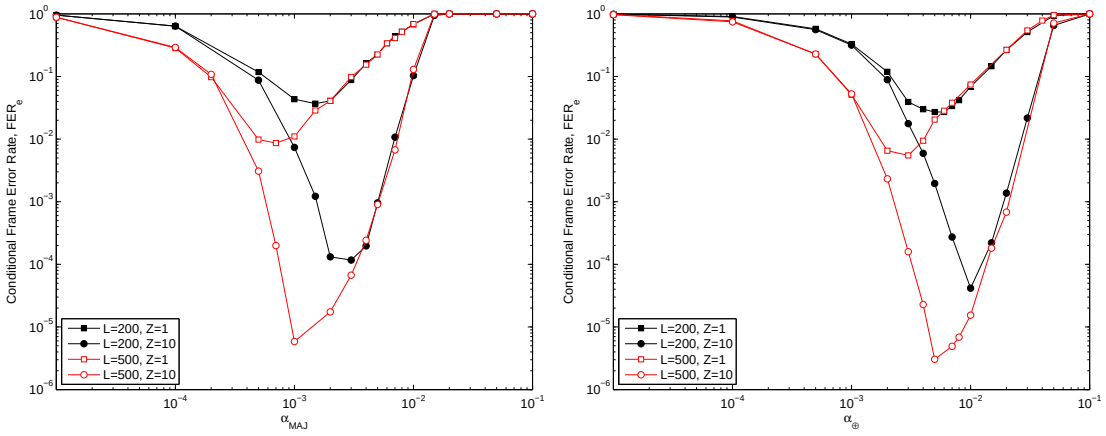
Figure 2.7: (a) Conditional FER for error pattern $\mathbf{e}$=(11000) as a function of Z for $\alpha_{\oplus MAJ} = 0.01$ when the retransmissions are applied, $L_R = 25$. (b) Average FER as a function of crossover probability.

Code (155,64), experimental results

Now show that a decoder with noisy gates is capable of correcting $(5, 3)$ trapping sets that are uncorrectable by the perfect Gallager-B decoder. We consider the $(3, 5)$-regular $(n, k) = (155, 64)$ LDPC code constructed by Tanner [51], which is widely used in literature. The minimum Hamming distance between codewords is $d_{min} = 20$, thus the perfect maximum likelihood (ML) decoder would be able to correct any nine-error pattern. However, the perfect Gallager-B decoder fails on some three-error patterns [52]. In the $(155, 64)$ Tanner code, every uncorrectable three-error pattern induces a (5,3) trapping set of a unique topology [48]. Now, we show that these three-error patterns can be corrected by our noisy decoder with high probability. We consider two different scenarios: (i) when check node processing is perfect and only MAJ gates are noisy, and (ii) when variable node processing is perfect and only XOR gates are noisy.

The conditional FER of the Tanner (155,64) code for the most critical three-bit error patterns is presented in (Fig. 2.8(a) and 2.8(b)). The particular low-weight error pattern cannot be decoded by perfect hardware, but it can be decoded with non-zero probability for a wide range of gate error probabilities α_{MAJ} and $\alpha_{\oplus}$. After sufficient number of iterations, the minimum FER is not achieved for perfect gates but for some nonzero value of the gate error rates α_{MAJ} and $\alpha_{\oplus}$. For a broad range of gate error rates, our decoder actually benefits from logic gate errors.

Increasing the maximum number of iterations, L, reduces the probability that the error pattern remains uncorrected. The impact of L is especially noticeable for high reliability gates (i.e., low α_{MAJ} and $\alpha_{\oplus}$), as it can be seen From Fig. 2.8(a) and Fig. 2.8(b). In this case, hardware errors cannot help much in annihilating trapping sets because the state transition probabilities in $\mathcal{W}$ are small for most transitions other than those that already exist in the perfect decoder. Consequently, the convergence to the subset $S_{\mathbf{0}}$ takes longer (it also grows with n). On the other hand, increasing Z has stronger effect when gates are very noisy. Hence, prolonging the second stage of the decoding algorithm greatly improves the performance of a decoder made of the less reliable hardware. For high gate error rates, L has smaller impact. If some FER degradation can be tolerated, the low ratio Z/L can be used in



(a) (b)

Figure 2.8: The frame error rate performance of faulty Gallager-B decoder for Tanner (155,64) code as a function of error rates in the logic gates, α_{MAJ} and $\alpha_{\oplus}$. The FER curves are estimated by using Monte Carlo simulations. The storage medium introduces the worst case three errors-pattern which induces the $(5, 3)$ trapping set. The maximum number of iterations is $L = 200$ and $L = 500$, and the perfect syndrome checker was used in the last $Z \leq L$ iterations only. Gate errors affect (a) MAJ gates only, (b) XOR gates only.

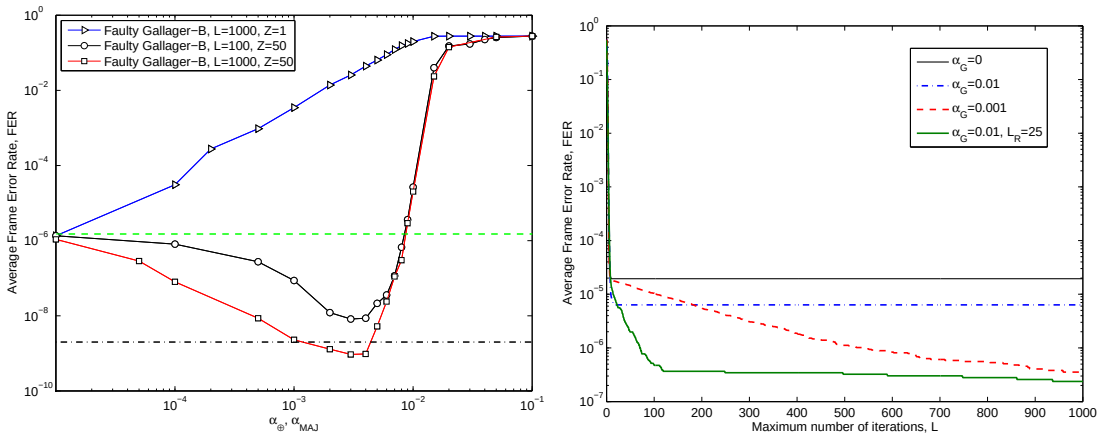
order to improve the decoder energy efficiency in this case. Also, it can be noticed that a faulty XOR gate has approximately $(\rho - 1)$ stronger effect on the FER performance than a faulty MAJ gate. In other words $\alpha_{\oplus} \approx (\rho - 1)\alpha_{MAJ}$ would result in approximately the same performance, as shown earlier. Therefore, the impact of the both types of failures can be represented by using a sinngle parameter denoted as α_G.

It is important to notice that the results in (Fig. 2.8) are given for an error pattern that is uncorrectable by the perfect decoder. Therefore, for this particular error pattern, the noisy decoder works better than the perfect one for any gate failure rates in the range $\alpha_{MAJ} \leq 0.02$ and $\alpha_{\oplus} \leq 0.1$, and for any L and Z. Optimization of these parameters can lead to further improvements in decoder performance and energy efficiency.

If the storage medium is modeled as BSC, any error pattern can occur at the decoder input with certain probability. Therefore, we estimate the performance of noisy Gallager-B decoder is this case as well. For the case when both XOR and MAJ gates are noisy with $\alpha_{\oplus}$=α_{MAJ}, and the numerical results are presented in Fig. 2.9(a) for the case when $\alpha_M = 2 \times 10^{-3}$. Since the previously considered low-weight pattern is most critical, the main effects are same as in Fig. 2.8. For the perfect decoder, the average FER is approximately equal to the probability of appearance of the dominant trapping set at the decoder input. In a noisy decoder, the lowest FER is achieved for the gate error rates that maximize successful correction of most critical trapping sets.

The faulty Gallager-B decoder is more efficient than its perfect counterpart for any values of the failure rates in the logic gates less than 10^{-2}. Even more importantly, when $L = 1000$ and the gate error rates have near-optimal values, the noisy hard-decision algorithm has better performance than the much stronger and more complex soft-decision min-sum algorithm realized in perfect hardware. For low gate failure rates, L has the dominant effect on the FER, especially for the lower values of the failure rates as shown in Fig. 2.9(a). Increasing number of iterations when syndrome checker is applied is crucial, as for small Z even very large L does not improve the performance (see e.g., the $L = 1000$, $Z = 1$ curve). This is the direct consequence of the fact that in this case the absorbing states are reached in only a small number of iterations when the syndrome checker is turned on, while in most of the iterations only transient states are visited.

In Fig. 2.9(b), the average FER as a function of number of iterations is presented for $\alpha_M = 5 \times 10^{-3}$. If the rewinding is not applied, larger values of the failure rates result in performance improvement when compared to the perfect decoder, but the increase of maximum number of iterations

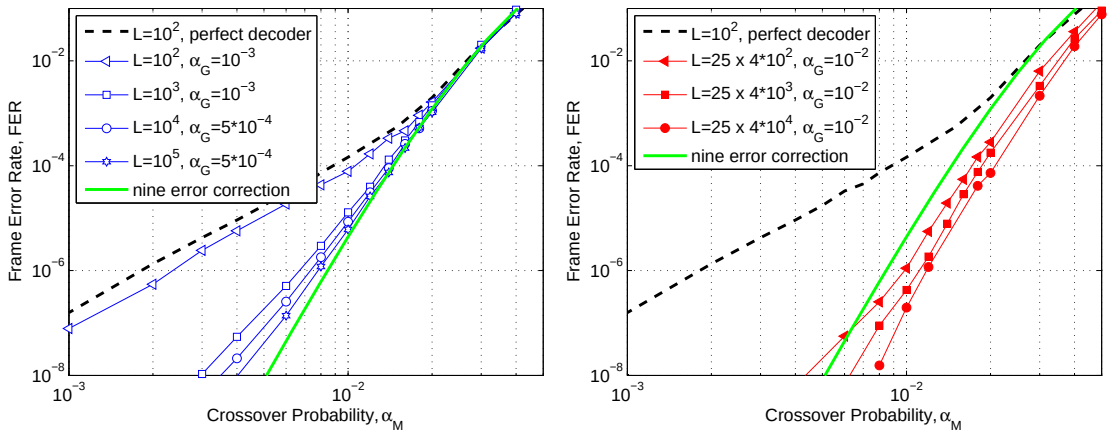


(a) (b)

Figure 2.9: (a) Performance of the faulty Gallager-B decoder for Tanner (155,64) code as a function of failure rates (both XOR and MAJ gates are faulty with the same failure rate); comparison with the perfect Gallager-B decoder and perfect min-sum decoder, errors in memory are i.i.d. with $\alpha_M = 2 \times 10^{-3}$. (b) The FER performance of the faulty Gallager-B decoder on the Tanner (155,64) code as a function of number of iterations. The decoders with and without rewinding are considered and the effect of the per-round number of iterations L_R is illustrated for $\alpha_M = 5 \times 10^{-3}$.

does not result in further decrease of FER. On the other hand, for smaller failure rates the significant performance improvement is noticeable for large values of parameter L. However, if the maximum number of decoding iterations is limited, the performance can be inferior when compared to the case of higher failure rates. If the rewinding is applied, the positive effect of the logic gate failures is exploited several times for the various reinitializations. It is reasonable to chose the rewinding period L_R to be equal to the number of iterations where the performance improvement saturates in the case without rewinding.

A comparison of different decoding strategies suitable for logic gates with high or low reliability is shown in Fig. 2.10. The FER curves for decoding with no rewinding are shown in Fig. 2.10(a). For all L, the decoder $\mathcal{F}$ outperforms the ideal decoder $\overline{\mathcal{F}}$, and for large L its performance approaches the ideal decoder capable of correcting any combination of nine errors. The positive effect of the rewinding is shown in Fig. 2.10(b) for various choices of $L = r \times L_R$ combinations and various maximal number of iterations L. For $\alpha_G = 10^{-2}$, the rewind decoder $\mathcal{F}^{\circlearrowleft}$ performs beyond the $\lfloor \frac{d_{min}-1}{2} \rfloor$ bound. The total number of iterations $L = r \times L_R$ is kept the same as for the corresponding decoder $\mathcal{F}$.



(a) (b)

Figure 2.10: Performance of the faulty Gallager-B decoder for Tanner (155,64) code as a function of the BSC crossover probability α_M for various decoding strategies (no rewind decoding (a) and rewind decoding (b)) effective for low and high failure rate ranges.

To conclude, the above two decoding strategies cover two gate error rate regimes. For more reliable logic gates, large number of iterations is needed before the decoder start benefiting from the positive effects of logic gate errors. In this case, we apply syndrome checker in the first twenty iterations, as the average FER rapidly decreases only at the beginning of the decoding. Then, we turn-off both the final bit-estimation circuit as well as the syndrome checker, and allow for sufficient number of iterations before we again activate the perfect syndrome checking. Clearly, this strategy results in energy saving as the perfect gates are used in a reduced number of iterations. When gate failures rate are high, errors correctable by the perfect decoder may turn uncorrectable, or lead to miscorrections as they may lead to large deviations from the trajectory of the perfect decoder. A solution for this case is to rewind the decoder. The higher the gate error rate, the lower optimal rewind period L_R.

2.6 Conclusion

The noisy decoder proposed in this chapter is a rare example of a system built from a mixture of noisy and perfect components that works better than a perfect system of the same or even higher complexity. The exact energy consumption analysis would require hardware implementation and is beyond the scope of this chapter, but the fact that our decoders use the perfect syndrome computations Z/L fraction of time, implies overall energy savings compared to completely perfect decoders.

Our methodology can be applied to two different scenarios. In the first scenario, errors in logic gate occur due their inherent unreliability, while in the second, the errors are inserted deliberately to improve the decoding convergence. For example, in low-powered CMOS chips, the reliability of logic gates can be tuned by changing their supply voltage. Underpowered noisy gates and perfect gates are used so that only low fraction of overall computations - critical computations at critical times - is performed on the perfect gates while the bulk of processing is done by the energy more efficient logic gates. In the second scenario performance of iterative decoders made of perfect logic gates is improved by deliberately inserting errors. The errors can be in a form of random bit flips added to logic gate outputs. Generating random bits using noisy hardware is a known concept in the very large scale integration (VLSI) community. The so called “true” random number generators (TRNG) of negligible complexity compared to pseudo random counterparts based on linear feedback shift registers, may be realized using variety of fundamental noise mechanisms in electronics circuits [53].

Chapter 3

Design of Taylor-Kuznetsov Memories built from Gallager-B LDPC Decoders

Abstract: *In this part of the deliverable, we provide an analysis of the reliability of the memory architecture proposed by Taylor [24] and Kuznetsov [46], under the framework of the i-RISC project. We introduce a refined time-dependent memory degradation model to capture the effects of the circuit noise. The successive error probabilities in the memory are expressed, from which we introduce a threshold definition to characterize the set of memory degradation parameters and decoder noise parameters that ensure the existence of a reliable memory. From the reliability analysis, we design practical code and decoder parameters in order to minimize the redundancy per time unit of the memory architecture.*

3.1 Introduction

Over the past years, the size of electronic chips has considerably reduced, while the integration factors on the chips have dramatically increased. Because of this huge scale change, energy consumption has become a major issue in the design of the next generations of electronic devices. A usual solution to lower the energy consumption is to decay the power supply of electronic chips by several orders of magnitude [54]. However, both chip size reduction and lower power supply make electronic components much more sensitive to noise [55, 56]. As a consequence, it is now widely recognized that the computation and storage units built on the next generations of electronic chips will become unreliable. This chapter addresses the issue of constructing reliable memories built from unreliable components.

Taylor [24] and Kuznetsov [46] were the first to propose a reliable memory architecture built from unreliable components. In the Taylor-Kuznetsov memory architecture, the information is stored as a codeword obtained from a Low Density Parity Check (LDPC) code. As the hardware introduces some errors in the stored information, the codeword is regularly extracted from the memory and passed through an LDPC decoder in order to correct the hardware errors. After a long time, the stored codeword can be either very close to the initial codeword or far away if the decoder is not able to compensate the memory degradation. The memory architecture is said to be reliable only if the initial codeword can always be recovered from the codeword that is actually stored in the memory. As the reliability of the memory architecture depends on the choice of the LDPC code and decoder, there is a need for a theoretical analysis to predict which codes and decoders lead to a reliable memory.

Recently, Chilappagari et al. [57, 58], and Brkic et al. [26] considered the use of a faulty One-Step Majority Logic (OS-MAJ) decoder in the memory architecture proposed by Taylor and Kuznetsov. The authors of [26, 57, 58] evaluated the Bit Error Rate (BER) performance under hardware error of the OS-MAJ decoder alone. The reliability of the whole memory architecture built from an OS-MAJ decoder was analyzed in [23, 59]. The analysis of [59] is based on graph expander conditions, which is difficult to characterize for large graphs. In [23], the reliability of the memory architecture is evaluated through Monte-Carlo simulations. Here, we would like to consider stronger decoders than the OS-MAJ

decoder, with possibly more than one iteration. Stronger decoders may lead to increased reliability at the price of a more complex memory architecture.

Information theory provides the fundamental limits of communication systems as the minimum rate at which the information can be transmitted reliably through a noisy channel. As a parallel, the fundamental limits of computation systems may be addressed through redundancy. The redundancy was defined in [24] as the number of noisy elements required to construct the memory architecture divided by the memory capability k, that is the number of information bits stored by the memory. However, as pointed out in [23], determining the minimum redundancy such that the memory built on faulty hardware is reliable is a hard problem. Indeed, the redundancy depends on the architecture itself, and in particular on the complexity of the considered decoder. An alternative would be to choose a particular decoder, and to design the architecture and decoder parameters such as to minimize the redundancy. To the best of our knowledge, this problem has not been addressed so far in the literature.

As a design parameter of the memory architecture, the refresh time can be defined as the time duration between two decoding instances. In the previous works on the Taylor-Kuznetsov memory architecture [2, 23, 24, 26, 46, 57–59], the refresh time is assumed to be fixed: the memory degradation between two decoding instances is represented by a Binary Symmetric Channel (BSC) of parameter α_0 that does not depend on the refresh time. However, there is a tradeoff between the decoder complexity and the refresh time in the sense that a stronger decoder (*e.g.*, with more iterations) will tolerate an increased refresh time. Thus, in addition to being an important design parameter, the refresh time could be used to find the code and decoder parameters that minimize the redundancy per time unit of the memory architecture.

In this chapter, we consider the Taylor-Kuznetsov memory architecture built from a Gallager B LDPC decoder with possibly more than one iteration. As the LDPC decoder runs on the same faulty hardware as the memory, it is assumed faulty as well. The BER performance of the faulty Gallager B decoder alone was analyzed in [30, 60, 61] using noisy Density Evolution (DE) initially introduced in [2]. It was shown that the Gallager B decoder under hardware errors only experiences a small loss in performance compared to the noise-free decoder, and thus it is reasonable to use it in the memory architecture. In this chapter, we want to design the code and decoder parameters in order to minimize the redundancy of the memory architecture, while guaranteeing its reliability. Not only this approach gives an upper bound on the minimum possible redundancy, but it also provides a rigorous guideline for the construction of a low redundancy reliable memory architecture.

First, in order to be able to minimize the redundancy of the architecture by expressing the tradeoff between the decoder complexity and the refresh time, we introduce a new time-dependent memory degradation model. The proposed model gives the amount of errors that are introduced by the hardware in the memory during a given time duration.

We then propose an analytical method to predict the reliability of the memory architecture for fixed refresh time, code, and decoders parameters. In order to analyze the reliability of the architecture, we express the evolution over time of the probability of errors in the codeword stored in the memory. This defines a sequence of error probabilities in the memory at successive time instants. We analyze the growing and convergence properties of this sequence of error probabilities, and based on these properties, we introduce a threshold definition that indicates the maximum degradation level that the memory can tolerate between two decoding instances. The threshold definition we propose is different from the threshold definitions introduced for faulty LDPC decoders [2, 62, 63], as it takes into account not only the LDPC decoder, but also the dynamic of the whole memory architecture. The threshold definition enables us to represent reliability regions as a set of degradation levels and decoder noise parameters for which a reliable storage of information is possible.

To finish, we proceed to the design of the memory architecture. From the reliability analysis, we derive the minimum refresh time that can be tolerated by the memory to be reliable. We then express the redundancy of the memory architecture, which depends both on the decoder complexity and on the minimum refresh time. From this expression, we provide the optimal code and decoder parameters that minimize the redundancy.

The outline of the chapter is as follows. Section 3.2 presents the memory architecture and in-

troduces the time-dependent memory degradation model. Section 3.3 derives the sequence of error probabilities in the memory and analyzes the growing and convergence properties of the sequence. Section 3.4 introduces the reliability definition and the memory threshold definition. Section 3.5 addresses the design of the reliable low redundancy memory architecture.

3.2 Memory Architecture

In this section, we present the memory architecture based on LDPC codes initially proposed by Taylor and Kuznetsov in [24, 46], and latter considered in [23, 26, 57–59]. We introduce the time-dependent model we consider to represent the memory degradation induced by the faulty hardware. We explain how LDPC codes enable to overcome the memory degradation induced by the faulty hardware and we describe the faulty LDPC Gallager B decoder we use as a correction circuit in the memory architecture.

3.2.1 Memory Architecture

Assume that the memory has a storage capability of k bits. Le $\mathbf{x}^{(0)}$ be a codeword obtained from an LDPC code of dimension k defined by a parity check matrix H of size $m \times n$, with $k \leq m - n$. The Tanner graph of the code is composed of n Variable Nodes (VN) $v \in \{1, \ldots, n\}$ and m Check Nodes (CN) $c \in \{1, \ldots, m\}$. The degree of the VN v is denoted as d_v and the degree of the CN c is denoted as d_c. Here, the code is assumed to be regular, *i.e*, d_v does not depend on v, and d_c does not depend on c. Denote by $\mathcal{N}(v)$ the set of CNs connected to the VN v, and denote by $\mathcal{N}(c)$ the set of VNs connected to the CN c.

At time instant $t = 0$, the codeword $\mathbf{x}^{(0)}$ is written in the memory. Denote by $\mathbf{x}^{(t)}$ the binary information vector of length k that is contained in the memory at time instant t. Let $x_v^{(t)}$ be the v-th component of the vector $\mathbf{x}^{(t)}$. The content stored in the memory is regularly degraded due to the hardware noise. Consider discrete time instants $t = 0, \Delta_t, 2\Delta_t, \cdots$. In the previous works on TK memories [23,24,26,46,57–59], the memory degradation between two time instants $j\Delta_t$ and $(j+1)\Delta_t$ is represented by a Binary Symmetric Channel (BSC) of parameter α, we denote BSC(α). Unfortunately, because of the memory degradation, the number of errors in $\mathbf{x}^{(t)}$ with respect to $\mathbf{x}^{(0)}$ increases with t. For large enough t, $\mathbf{x}^{(t)}$ will contain too many errors, and it will not be possible to recover the initial $\mathbf{x}^{(0)}$ from $\mathbf{x}^{(t)}$ anymore. In order to overcome this effect, an LDPC decoder is regularly applied to the $\mathbf{x}^{(t)}$ as follows.

As in [23, 24, 26, 46, 57–59], we assume that the LDPC decoder is applied at every time instant $t = j\Delta_t$. Denote $\mathbf{y}_{\text{in}}^{(j)} = \mathbf{x}^{(t)}$ the vector extracted from the memory at time instant $t = j\Delta_t$ and set at the input of the LDPC decoder. Denote $\mathbf{y}_{\text{out}}^{(j)}$ the output of the LDPC decoder which is written back in the memory. Here, we assume that the LDPC decoder duration δ is such that $\delta << T$. For simplicity, and as an abuse of notations, we thus set $\delta = 0$. Formally speaking, it corresponds to a discontinuity in the $\mathbf{x}^{(t)}$ at all the time instants $t = j\Delta_t$.

In the memory degradation model considered in [23, 24, 26, 46, 57–59], the BSC parameter α does not depend on the time interval Δ_t. This degradation model is far from being realistic since the longer the information is stored, the more it should be degraded. In addition, the time interval Δ_t is a parameter that should be properly designed, depending on the decoder, in order to minimize the total energy consumption of the memory. In the following, before introducing the LDPC decoder that will be used in the memory architecture, we first introduce the memory degradation model that will be considered in the chapter. In the degradation model we introduce, the parameter of the BSC depends on Δ_t.

3.2.2 Degradation Model

Here, we assume that the information stored in the memory continuously degrades over time, so that the parameter of the BSC applied during Δ_t depends on the value of Δ_t. Consider a time instant t such that $j\Delta_t < t < (j+1)\Delta_t$ and a time difference δ_t such that $j\Delta_t < t + \delta_t < (j+1)\Delta_t$. In order to

represent the degradation level in the vector $\mathbf{x}^{(t)}$ during δ_t, we define a degradation function $\alpha(\delta_t)$ as

$$\begin{aligned}\alpha(\delta_t) &= P(x_v^{(t+\delta_t)} = 1 | x_v^{(t)} = 0) &= P(x_v^{(t+\delta_t)} = 0 | x_v^{(t)} = 1),\\ &= P(x_v^{(\delta_t)} = 1 | x_v^{(0)} = 0) &= P(x_v^{(\delta_t)} = 0 | x_v^{(0)} = 1).\end{aligned} \tag{3.1}$$

The defined model is time-invariant. This model is equivalent to assuming that $\mathbf{x}^{(t+\delta_t)}$ is obtained by passing $\mathbf{x}^{(t)}$ through a BSC of parameter $\alpha(\delta_t)$ that depends on δ_t. The degradation model defined by (3.1) is symmetric since the probability that a bit is flipped does not depend on the bit value. It is also spatially memoryless in the sense that the probability of a given bit $x_v^{(t)}$ does not depend on the probability of the other bits $x_{v'}^{(t)}$, for $v' \neq v$. The symmetry and memoryless assumptions greatly simplify the analysis of the reliability of the memory architecture. More accurate models will be considered in future works.

In order to completely define the degradation model, it remains to give an analytical expression of the function $\alpha(\delta_t)$. We rely on the time-invariant property of the error model and define the function α within the first time interval $[0, \Delta_t]$. The following proposition states the expression of $\alpha(\delta_t)$ we consider in the chapter.

Proposition 1. *Denote by α_0 the elementary degradation per time unit, defined as*

$$\alpha_0 dt = P(x_v^{(\delta_t+dt)} = 1 | x_v^{(\delta_t)} = 0) = P(x_v^{(\delta_t+dt)} = 0 | x_v^{(\delta_t)} = 1) \tag{3.2}$$

The memory degradation function $\alpha(\delta_t)$ defined in (3.1) *then has expression*

$$\forall 0 \leq \delta_t \leq \Delta_t,\ \alpha(\delta_t) = \frac{1}{2}(1 - \exp(-2\alpha_0 \delta_t)). \tag{3.3}$$

Proof. From (3.1), we get that $\alpha(0) = 0$ and that the function $\alpha(\delta_t)$ verifies $\alpha(\delta_t + dt) = \alpha_0 dt(1 - \alpha(\delta_t)) + \alpha(\delta_t)(1 - \alpha_0 dt)$. Consequently $\alpha(\delta_t)$ is the solution of the differential equation

$$\alpha(0) = 0, \quad \alpha'(\delta_t) = \alpha_0(1 - 2\alpha(\delta_t)).$$

It it then straightforward to show that (3.3) is the solution of this differential equation. □

The definition of α_0 in (3.2) assumes that at time instant δ_t and during infinitesimal time interval dt, the vector $\mathbf{x}^{(\delta_t)}$ is passed through a BSC of parameter $\alpha_0 dt$. It can be shown that the resulting function $\alpha(\delta_t)$ in (3.3) is increasing with δ_t, and that $\lim_{\delta_t \to \infty} \alpha(\delta_t) = 1/2$.

3.2.3 Gallager B decoder

We first describe the noiseless version of the LDPC Gallager B decoder we consider in the memory architecture. The decoder runs in L iterations. Denote by $\eta_v^{(0)}$ the initial message at VN v. At iteration $\ell = 1, \cdots, L$, denote by $\gamma_{c\to v}^{(\ell)}$ the message from CN c to VN v, and by $\eta_{v\to c}^{(\ell)}$ the message from VN v to CN c.

At time instant $kT' + T$, the decoder receives the vector $\mathbf{x}^{(kT'+T)}$ and sets $\eta_v^{(0)} = x_v^{(kT'+T)}$, and $\eta_{v\to c}^{(1)} = x_v^{(kT'+T)}$. At iteration ℓ, each CN c computes all the messages $\gamma_{c\to v}^{(\ell)}$, for $v \in \mathcal{N}(c)$, as

$$\gamma_{c\to v}^{(\ell)} = \sum_{v' \in \mathcal{N}(c)\backslash v} \eta_{v'\to c}^{(\ell)} \tag{3.4}$$

where the sum corresponds to a XOR sum. Then, each VN v computes all the messages $\eta_{v\to c}^{(\ell+1)}$, for $c \in \mathcal{N}(v)$, as

$$\eta_{v\to c}^{(\ell+1)} = \begin{cases} \eta_v^{(0)} \oplus 1 \text{ if } |\{c' \in \mathcal{N}(v)\backslash c : \gamma_{c'\to v}^{(\ell)} = \eta_v^{(0)} \oplus 1\}| > b, \\ \eta_v^{(0)} \text{otherwise,} \end{cases} \tag{3.5}$$

where b is a parameter of the Gallager B decoder. At the end of iteration L, *A Posteriori* Probability (APP) values $\eta_v^{(L)}$ are calculated as

$$\eta_v^{(L)} = \begin{cases} \eta_v^{(0)} \oplus 1 \text{ if } |\{c' \in \mathcal{N}(v) \backslash c : \gamma_{c' \to v}^{(\ell)} = \eta_v^{(0)} \oplus 1\}| > 0, \\ \eta_v^{(0)} \text{otherwise.} \end{cases} \tag{3.6}$$

At the end of the decoding, each VN is set as $x_v^{((k+1)T')} = \eta_v^{(L)}$. The resulting vector $\mathbf{x}^{((k+1)T')}$ corresponds to the vector stored in memory at time instant $(k+1)T'$.

In the noisy version of the Gallager B decoder, we assume that the hardware introduces some noise at the end of the computation of $\gamma_{c \to v}^{(\ell)}$, $\eta_{v \to c}^{(\ell)}$, and $\eta_v^{(L)}$ in (3.4), (3.5), (3.6). Denote by ν the binary error probability on the message computation. Denote by $\tilde{\gamma}_{c \to v}^{(\ell)}$, $\tilde{\eta}_{v \to c}^{(\ell)}$, and $\tilde{\eta}_v^{(\ell)}$ the noisy versions of the messages $\gamma_{c \to v}^{(\ell)}$, $\eta_{v \to c}^{(\ell)}$, and $\eta_v^{(\ell)}$. The expressions of $\tilde{\gamma}_{c \to v}^{(\ell)}$, $\tilde{\eta}_{v \to c}^{(\ell)}$, and $\tilde{\eta}_v^{(\ell)}$ are given by

$$\tilde{\gamma}_{c \to v}^{(\ell)} = \gamma_{c \to v}^{(\ell)} \oplus e_{c \to v}^{(\ell)}, \tag{3.7}$$

$$\tilde{\eta}_{v \to c}^{(\ell)} = \eta_{v \to c}^{(\ell)} \oplus e_{v \to c}^{(\ell)}, \tag{3.8}$$

$$\tilde{\eta}_v^{(L)} = \eta_v^{(L)} \oplus e_v^{(L)}. \tag{3.9}$$

where $e_{c \to v}^{(\ell)}$, $e_{v \to c}^{(\ell)}$, and $e_v^{(L)}$ are random variables such that $P(e_{c \to v}^{(\ell)} = 1) = \nu$, $P(e_{v \to c}^{(\ell)} = 1) = \nu$, and $P(e_v^{(L)} = 1) = \nu$. Although the defined error model may not capture all the noise effects that could appear inside the XOR sum (3.4) and in the majority voting function (3.5), it does not require knowledge of a particular hardware implementation of the functions. As for the memory degradation model, the decoder error model is symmetric and memoryless. However, it appears sufficient for the first step of the analysis and more accurate models will be considered in future works.

Now, we would like to determine whether and for which noise parameters α_0 and ν the considered memory architecture is reliable in the sense that $\mathbf{x}^{(0)}$ can always be recovered from $\mathbf{x}^{(t)}$, even for large t. In order to define the recovery condition more precisely, we now explain how the stored information is read from the memory.

3.2.4 Final Decoder

When at time instant t, the vector $\mathbf{x}^{(t)}$ has to be read by a unit outside of the memory, a reconstruction operation has to be applied to $\mathbf{x}^{(t)}$ in order to recover perfectly the original $\mathbf{x}^{(0)}$. Here, we will assume that this reconstruction operation consists in applying a noiseless Belief Propagation (BP) decoder to $\mathbf{x}^{(t)}$.

At the end, the memory is said to be reliable if the original information vector $\mathbf{x}^{(0)}$ can be perfectly recovered by applying the BP decoder to any $\mathbf{x}^{(t)}$, even for large values of t. To verify this condition, we now express the evolution over time of the probability of errors in the stored codeword $\mathbf{x}^{(t)}$.

3.3 Error Probability Evaluation

In this section, we analyze the reliability of the memory architecture by expressing the evolution over time of the error probability in $\mathbf{x}^{(t)}$. Between time instants $t = kT'$ and $t = kT' + T$, the error probability in $\mathbf{x}^{(t)}$ is a its highest level just before decoding, at $t = kT' + T$, because $\alpha(t)$ is an increasing function of t. Thus, we express the successive error probabilities in the vectors $\mathbf{x}^{(kT'+T)}$ for $k = 0, 1, 2, \cdots$. For the reliability analysis, we assume that the refresh time T is fixed, and we set $\alpha = \alpha(T)$, where α corresponds to the amount of degradation in the memory between two decoding instances.

3.3.1 Error Probability Function of the Faulty Gallager B Decoder

In order to compute the successive error probabilities in the memory, we assume that we are given the expression of the function $\beta \to \mathrm{P}_{e,\nu}(\beta)$, which is the error probability function of the faulty

Gallager B decoder considered in the memory architecture. The value β for which $\mathrm{P}_{e,\nu}(\beta)$ is evaluated corresponds to the error probability at the input of the decoder. Note that $\mathrm{P}_{e,\nu}(\beta)$ also depends on ν, the decoder noise parameter. The expression of $\mathrm{P}_{e,\nu}(\beta)$ can be obtained from noisy Density Evolution for the faulty Gallager B decoder, see [30, 61].

The results of [30, 61] show that the faulty Gallager B decoder is symmetric in the sense of [2]. As a consequence, the error probability $\mathrm{P}_{e,\nu}(\beta)$ does not depend on the input codeword, but only on the input error probability β. In the following, we use the function $\mathrm{P}_{e,\nu}(\beta)$ to express the error probabilities in the memory at successive time instants.

3.3.2 Sequence of Error Probabilities in the Memory

From the error probability function $\mathrm{P}_{e,\nu}(\beta)$, we now want to express the successive error probabilities in the $\mathbf{x}^{(kT'+T)}$, for $k = 0, 1, 2, \cdots$. As the faulty Gallager B decoder and the memory degradation model represented by a BSC are symmetric in the sense of [2], the successive error probabilities do not depend on the codeword $\mathbf{x}^{(0)}$ initially stored in the memory, which greatly simplifies the analysis. According to the memory architecture defined in Section 3.2, the successive error probabilities in the $\mathbf{x}^{(kT'+T)}$ are given in the following proposition.

Proposition 2. *Denote $\beta_\nu^{(k)}(\alpha)$ the error probability in $\mathbf{x}^{(kT'+T)}$ with respect to $\mathbf{x}^{(0)}$,* i.e., *$\beta_\nu^{(k)}(\alpha) = P(x_v^{(kT'+T)} = 1 | x_v^{(0)} = 0)$. The successive error probabilities $\beta_\nu^{(k)}(\alpha)$ can be expressed recursively as*

$$\beta_\nu^{(0)}(\alpha) = \alpha, \tag{3.10}$$

and for all the integer values $k \geq 1$,

$$\beta_\nu^{(k)}(\alpha) = (1-\alpha) P_{e,\nu}\left(\beta_\nu^{(k-1)}(\alpha)\right) + \alpha\left(1 - P_{e,\nu}\left(\beta_\nu^{(k-1)}(\alpha)\right)\right). \tag{3.11}$$

At time instant $t = (k-1)T'$ (just after the $(k-1)$-th decoding), the stored vector $\mathbf{x}^{((k-1)T')}$ has error probability $\mathrm{P}_{e,\nu}\left(\beta_\nu^{(k'-1)}(\alpha)\right)$. It is then passed through the BSC of parameter α. The error probability $\beta_\nu^{(k)}(\alpha)$ (3.11) in the resulting $\mathbf{x}^{(kT'+T)}$ comes from the concatenation of two BSCs, of parameters $\mathrm{P}_{e,\nu}\left(\beta_\nu^{(k-1)}(\alpha)\right)$ and $\alpha(T)$, respectively.

Proposition 2 gives the recursive expression of the sequences $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ of error probabilities in the memory. The memory will be reliable if the successive error probabilities are small enough so that at any time instant, we can guarantee that $\mathbf{x}^{(kT'+T)}$ is in a close proximity of $\mathbf{x}^{(0)}$. In order to be able to check this condition for various values of refresh time T, we first analyze the increasing and convergence properties of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$.

3.3.3 Sequence Properties

The following proposition gives the increasing properties of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$.

Proposition 3. *Consider the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ given in Proposition 2. If for fixed value of ν, the function $\beta \to P_{e,\nu}(\beta)$ is increasing with β, the following properties hold*

1. *the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ is increasing with k*
2. *the function $\alpha \to \beta_\nu^{(k)}(\alpha)$ is increasing with α.*

If the function $(\beta, \nu) \to P_{e,\nu}(\beta)$ is increasing with both β and ν, the following property holds

4) the function $\nu \to \beta_\nu^{(k)}(\alpha)$ is increasing with ν.

Proposition 3 assumes that the function $\mathrm{P}_{e,\nu}(\beta)$ is increasing with β and with ν. Although these increasing assumptions are reasonable, the results of [21, 64] show that the second one is not always true. For example, for the discrete Min-Sum decoder with 7 quantization levels for the messages, the authors of [64] observe that the noise in the decoder can sometimes improve the decoder performance compared to the noiseless case. The same effect is observed for the Probabilistic Gradient Descent Bit-Flipping decoders introduced in [21], and Proposition 3 does not hold for such decoders. On the other hand, for the considered faulty Gallager B decoder, we observe that $\mathrm{P}_{e,\nu}(\beta)$ is increasing with β and with ν. As a consequence, for the memory architecture we consider in the chapter, higher values of α and ν, will lead to increased error probabilities $\beta_\nu^{(k)}(\alpha)$.

Proposition 3 also shows that the successive error probabilities $\beta_\nu^{(k)}(\alpha)$ are increasing with k. As a result, even with the Gallager B decoder, the hardware noise keeps degrading the stored information. However, we hope that the sequence of error probabilities $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ converges to a fixed point which is no too high, so that the initial $\mathbf{x}^{(0)}$ can always be recovered from $\mathbf{x}^{(kT'+T)}$, even for large values of k. In order to verify this condition, we now analyze the convergence behavior of $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$.

3.3.4 Fixed-Point Analysis

Here, we analyze the asymptotic behavior of $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ by determining the fixed points of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$. The fixed points of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ are the values β satisfying $\beta = (1-\alpha)\mathrm{P}_{e,\nu}(\beta) + \alpha(1-\mathrm{P}_{e,\nu}(\beta))$, or equivalently if $\alpha \neq 1/2$,

$$\mathrm{P}_{e,\nu}(\beta) = \frac{\beta - \alpha}{1-2\alpha}. \tag{3.12}$$

From the condition (3.12), the fixed points of the sequence of $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ correspond to the intersection of the curve representing $\mathrm{P}_{e,\nu}(\beta)$ and the straight line $y = \frac{\beta-\alpha}{1-2\alpha}$. This gives a simple condition to determine the fixed points of $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$. Note that if $\mathrm{P}_{e,\nu}(1/2) = 1/2$ (which is always satisfied), then $\beta = 1/2$ is always a fixed point of $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$, whatever the value of α is.

The fixed points correspond to the possible limits of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$. We know that $\beta = 1/2$ is always a fixed point, but it is a bad one for which we cannot recover the original $\mathbf{x}^{(0)}$ from $\mathbf{x}^{(kT'+T)}$. Thus, we hope that the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ has other fixed points which correspond to degradation levels that can be handled when the information vector is read from the memory. In the following, we propose a definition of the memory reliability that accounts for this condition.

3.4 Reliability Conditions

In this section, we give a definition of the reliability of the memory. The reliability definition relies on the above asymptotic analysis on the sequence of error probabilities $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$. From the reliability definition, we give a new threshold definition, that determines the maximum parameter α for which the memory is reliable. We then provide reliability regions that represent the set of noise parameters α and ν for which the memory is reliable.

3.4.1 Reliability Conditions

The reliability conditions we give here are based on the reliability conditions originally introduced in [24, Section 2.2]. The following definition adapts the reliability conditions of [24] to our analysis of the convergence properties of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$.

Definition 1. *Consider the memory architecture of Section 3.2. Consider the sequence* $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ *given in Proposition 2. Denote* $\mathcal{B}$ *the set of fixed-points of* $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ *excluding* $1/2$*, and denote by* $\beta^\star$ *the threshold of the noiseless BP decoder.*

A memory is said to be reliable*, if the following two conditions are verified*

1. Stability*: The set $\mathcal{B}$ is nonempty,*

2. Admissibility*: The set $\mathcal{B}$ is such that* $\max \mathcal{B} \leq \beta^\star$.

The condition 1 requires that the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$ has fixed points other than $\beta = 1/2$. It guarantees that the memory is stable in the sense that the error probabilities converge to a fixed point. The condition 2 ensures that the fixed point corresponds to a degradation level that can be handled by the final BP decoder.

The validity of Conditions 1 and 2 depend on the value of α, and on the decoder noise parameter ν. In order to identify the set of parameters α and ν that lead to a reliable memory, we introduce a threshold definition as follows.

3.4.2 Threshold Definition

For LDPC codes in channel coding, the noiseless threshold was defined in [65] as the maximum channel parameter α such that $\mathrm{P}_{e,\nu}(\alpha) = 0$. This condition cannot be applied here because of the noise introduced by the faulty hardware, which prevents the decoder from reaching an error probability 0. This is why several other threshold definitions were introduced for noisy decoders: the Useful Threshold [2], the target-BER Threshold [2, 6], and the Functional Threshold [62]. However, these threshold definitions cannot be used in our context, because they characterize the behavior of the faulty decoder alone. Here, we introduce a new threshold definition that characterizes the reliability of the whole memory architecture.

Definition 2. *Consider the memory architecture of Section 3.2 and fix the decoder noise parameters ν. The* Degradation Threshold *is defined as*

$$\overline{\alpha} = \arg\max_{\alpha} \{\textit{The memory is reliable}\}. \tag{3.13}$$

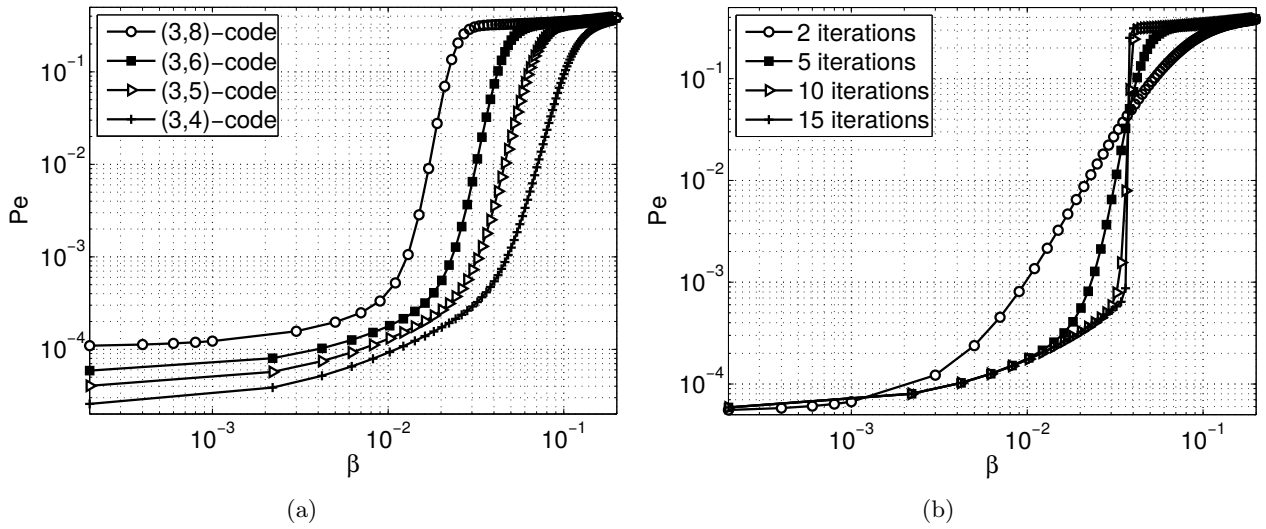
The Degradation Threshold $\overline{\alpha}$ is defined as the maximum parameter α for which the memory is reliable. The value of $\overline{\alpha}$ can be calculated directly from the fixed-point analysis defined by (3.12). Note that $\bar{\alpha}$ does not correspond to the threshold definitions introduced for the noisy LDPC decoders alone, as we now illustrate.

3.4.3 Reliability Regions

In this section, we provide reliability regions as the set of parameters α and ν that lead to a reliable memory. We consider regular LDPC codes of VN degree $d_v = 3$ and CN degrees $d_c = 4, 5, 6, 8$, respectively.

In order to verify the reliability of the memory for given parameters α and ν, we first need the expression $\mathrm{P}_{e,\nu}(\beta)$ of the error probability of the faulty Gallaber B decoder. The error probabilities $\mathrm{P}_{e,\nu}(\beta)$ are calculated from noisy-DE, [30, 66]. Figure 3.1 (a) shows $\mathrm{P}_{e,\nu}(\beta)$ as a function of β for the four considered codes, for $L = 5$ iterations and $\nu = 10^{-3}$. As expected, the error probability increases with β and with the code rate. However, there is no distinguishable threshold value on β that would separate the low and the high error probability regions. Figure 3.1 (b) shows $\mathrm{P}_{e,\nu}(\beta)$ as a function of β for the $(3, 6)$-code, for various values of number of iterations L. We see that the error probability decreases with L. As before, there is no distinguishable threshold value on β, because the values of L are too low for a decoder threshold effect to appear. However, from the analysis carried in the chapter, we can identify a Degradation threshold by analyzing the properties of the sequence $\{\beta_\nu^{(k)}(\alpha)\}_{k=1}^{+\infty}$.

Figure 3.2 (a) represents reliability regions as the Degradation Threshold values $\bar{\alpha}$ with respect to ν obtained from Definition 2, for the codes with $d_v = 3$ and with $L = 5$ iterations. The reliability regions are convex, even for large values of ν, and, as expected, the reliability regions shrink with the code rate increase. When the decoder noise parameter ν becomes too large, the threshold value $\bar{\alpha}$ becomes 0, which means that the decoder noise is too high for the memory to be reliable. Figure 3.2 (b) represents the reliability regions for the $(3, 6)$-code, for various values of L. The reliability regions



(a) (b)

Figure 3.1: Error probability curves of the Gallager B decoder (a) $L = 5$ iterations, $d_v = 3$, various values of d_c , (b) $(3, 6)$-codes, various values of L

extend when L increases. However, as L increases, the extension of the reliability regions seems to be less and less significant.

An increased value of L also increases the complexity of the decoder. This illustrates the tradeoff between reliability of the memory and complexity of the decoder. In the following, we address this tradeoff and we show how to optimize the decoder parameters.

3.5 Memory Architecture Design

In the previous section, we obtained the noise conditions, and in particular the maximum parameter $\bar{\alpha}$, for which the memory is reliable. From the Degradation threshold $\bar{\alpha}$, we now compute, as a first design parameter, the maximum refresh time T that the memory can tolerate to be reliable.

Both the reliability of the memory architecture and the complexity of the decoder depend on the code and decoder parameters. In this section, we show that the tradeoff between memory reliability and decoder complexity can be addressed from the expression of the redundancy per time unit of the memory architecture. We then optimize the code and decoder parameters in order to minimize the redundancy per time unit of the architecture.

3.5.1 Minimum Refresh Time

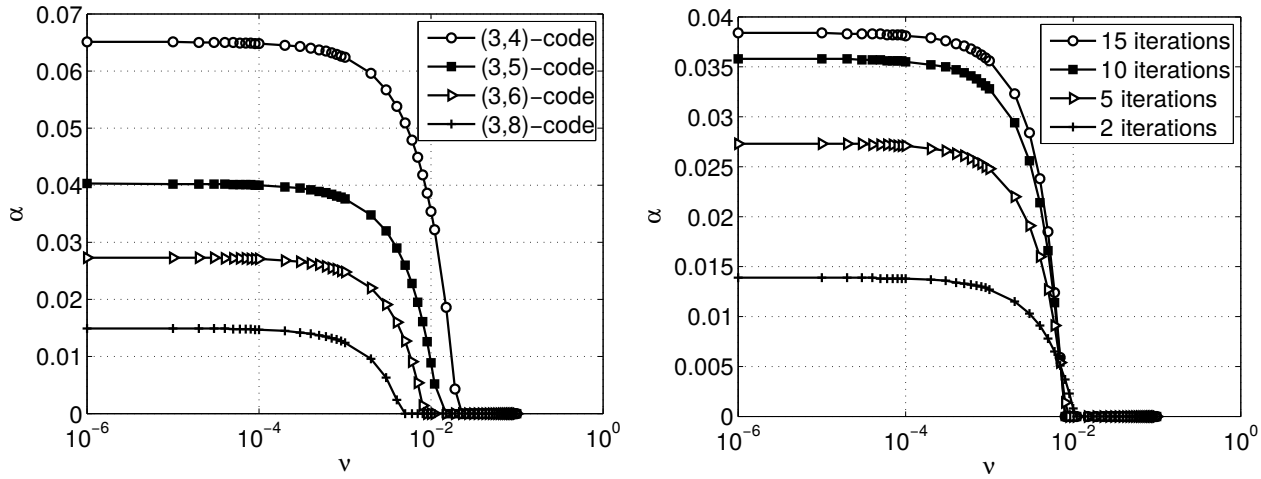
Denote by $\bar{T}$ the minimum refresh time that can be tolerated for the memory architecture to be reliable. The value of $\bar{T}$ can be calculated directly from the memory degradation threshold $\bar{\alpha}$, as expressed in the following proposition.

Proposition 4. *Assume that the Degradation threshold $\bar{\alpha}$ satisfies $\bar{\alpha} > 0$. The minimum refresh time $\bar{T}$ that can be tolerated by the memory is given by*

$$\bar{T} = \frac{1}{2\alpha_0} \log\left(\frac{1}{1 - 2\bar{\alpha}}\right). \tag{3.14}$$

In addition, $\bar{T} > 0$.

Proof. The expression of $\bar{T}$ in (3.14) is obtained from the definition of $\alpha(t)$ in (3.3), and from $\bar{\alpha} = \alpha(\bar{T})$. The condition $\bar{T} > 0$ directly follows from $\bar{\alpha} > 0$. □



(a) (b)

Figure 3.2: Reliability regions (a) $L = 5$ iterations, $d_v = 3$, various values of d_c, (b) $(3, 6)$-codes, various values of L

The parameter $\bar{T}$ is of practical importance in the memory architecture design as it sets how often the decoder have to be applied in order to get a reliable memory.

The value of $\bar{T}$ depends on the noise parameters α_0 and ν, but also on the code and decoder parameters such as the number of iterations L and the variable and check node degrees d_v, d_c. A stronger decoder with good correction capabilities, obtained *e.g.* for an increased value of L, will lead to an increased value of $\bar{T}$. However, a stronger decoder also comes at the cost of increased memory architecture complexity. In the following, we address the tradeoff between refresh time and complexity by expressing the redundancy per time unit of the architecture.

3.5.2 Redundancy of the Memory Architecture

Denote by C_+ and C_{maj} the complexity of the sum unit and of the majority voting unit, respectively. The complexity of the Gallager B decoder can be evaluated as

$$C_{\text{dec}} = nL\left((1-r)d_c C_+ + dvC_{\text{maj}}\right) + nC_{\text{maj}}. \tag{3.15}$$

The complexity of the decoder alone does not take into account how often the decoder have to be used. In order to take this into account, we assume that the refresh time is given by $\bar{T}$. We also assume that the decoder duration δ is such that $\delta \ll \bar{T}$, and, in order to simplify the expressions, we thus set $\delta = 0$. The validity of this assumption will be discussed latter in the chapter.

At the end, we express the redundancy per time unit of the whole memory architecture as

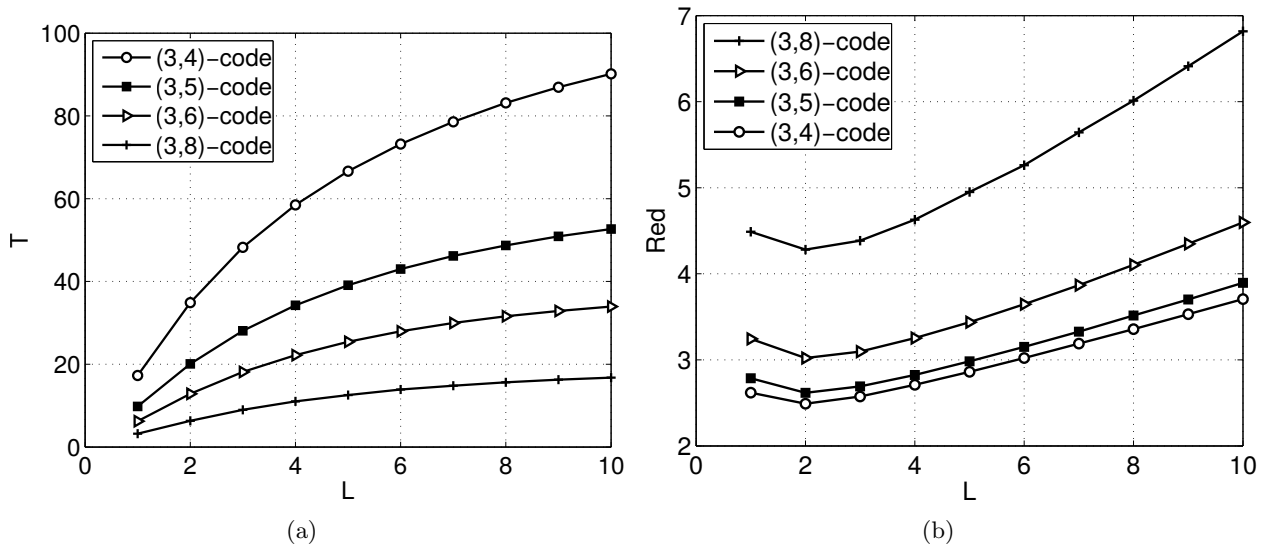
$$\text{Red} = \frac{L}{r\bar{T}}\left((1-r)d_c C_+ + dvC_{\text{maj}}\right) + \frac{C_{\text{maj}}}{r\bar{T}} + 1. \tag{3.16}$$

The redundancy Red depends on the code and decoder parameters L, d_v, d_c. It also depends on the minimum refresh time $\bar{T}$, which itself depends on the code and decoder parameters.

At the end, the redundancy (3.16) of the memory architecture expresses the tradeoff between the complexity of the decoder and the minimum refresh time. The code and decoder parameters L, d_v, d_c, thus have to be chosen in order to minimize the redundancy, as we now illustrate.

3.5.3 Decoder Design

In this section, we optimize the code and decoder parameters in order to minimize the redundancy per time unit. We consider several codes with $d_v = 3$ and with $d_c = 4, 5, 6, 8$, respectively.



(a) (b)

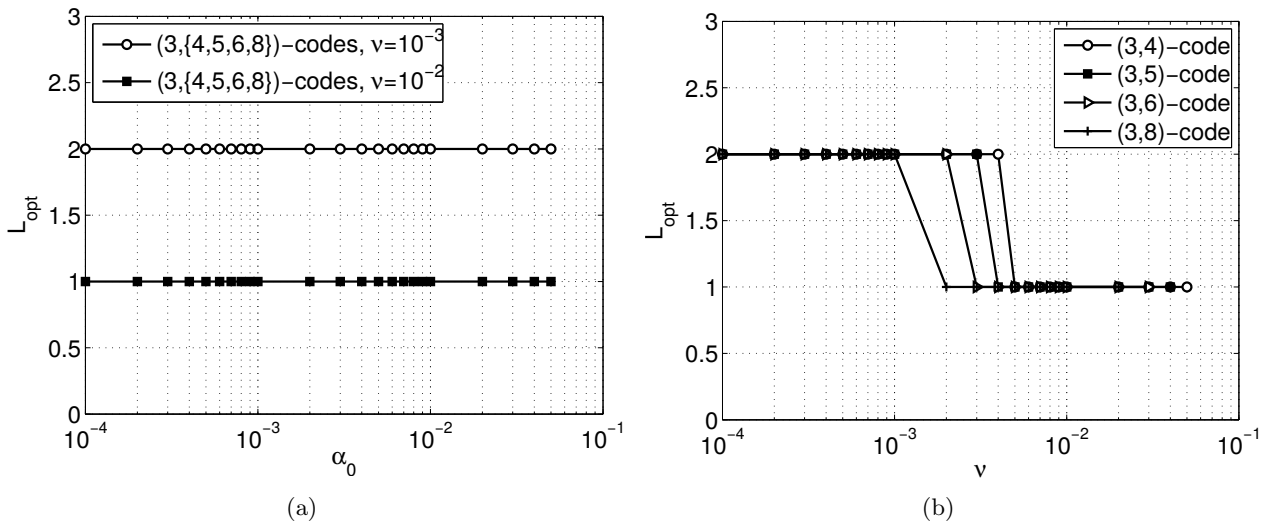
Figure 3.3: For Gallager B decoder, $\nu = 10^{-3}$, $\alpha_0 = 10^{-3}$ (a) Refresh time T with respect to number of iterations L, (b) Redundancy Red with respect to number of iterations L

We first fix the memory degradation parameter $\alpha_0 = 10^{-3}$ and the decoder noise parameter $\nu = 10^{-3}$ and analyze the variations of the minimum refresh time $\bar{T}$ and of the redundancy Red with respect to the code and decoder parameters. Figure 3.3 (a) represents the minimum refresh time $\bar{T}$ with respect to the number of iterations L. As expected, the refresh time increases with L and also with the code rate. Figure 3.3 (b) represents the corresponding redundancy Red with respect to L. In this case, the redundancy first decreases, and then increases with L. For the chosen parameters α_0 and ν, we see that the value $L = 2$ always minimizes the redundancy, whatever the code. We also see that the $(3, 4)$-code minimizes the redundancy.

We now analyze the influence of the noise parameters α_0 and ν on the optimal number of iterations L_{opt}. Figure 3.4 (a) represents the optimal number of iterations L_{opt} with respect to α_0, for the four considered codes, for $\nu = 10^{-3}$ and for $\nu = 10^{-2}$. We observe that the value of L_{opt} does not seems to depend on α_0, but depends on ν. Indeed, for $\nu = 10^{-3}$, the optimal value of L is always equal to $L_{\text{opt}} = 2$, while for $\nu = 10^{-2}$, we always get $L_{\text{opt}} = 1$. In order to verify the dependency with respect to ν, Figure 3.4 (b) represents the optimal number of iterations L_{opt} with respect to ν, for the four considered codes. The optimal value of L is indeed $L_{\text{opt}} = 2$ for low values of ν, and $L_{\text{opt}} = 1$ for higher values of ν. The transition on the value of L depends on the considered code. This shows that the optimal value of the number of iterations depends on the decoder noise parameter ν and on the considered code, but not on the memory degradation parameter α_0.

To finish, we want to choose the code that will be used in the architecture. As we have observed that the optimal choices of parameters do not depend on α_0, we fix $\alpha_0 = 10^{-3}$. Figure 3.5 (a) represents the optimal refresh time T_{opt} with respect to ν for the four considered codes. The sharp transitions between low values and high values of T_{opt} correspond to passing from $L_{\text{opt}} = 2$ to $L_{\text{opt}} = 1$. We also see that low rate codes, although giving more complex decoders, have the highest refresh time T_{opt}. Figure 3.5 (b) represents the minimized Redundancy values R_{opt} with respect to ν for the four considered codes. We see that although the redundancy values are all close, the code that minimizes the redundancy is always the $(3, 4)$-code, whatever the value of ν. Paradoxically enough, the $(3, 4)$-code is the code with the lowest rate, for which the decoder complexity is the highest.

As a summary, we have seen that the optimal choice of code and decoder parameters depend on ν, but not on α_0. Surprisingly, the optimization leads to a low value of L (lowers the decoder complexity), but also to a low rate code (increases the decoder complexity). The optimization method could easily be extended to irregular codes. Note also that the optimization process, and in particular the curves of Figure 3.5 (b), provide upper bounds on the minimum redundancy that can be achieved by the



(a) (b)

Figure 3.4: Optimization of the number of iterations L, for various codes (a) Optimal value of L with respect to α_0 (b) Optimal value of L with respect to ν for $\alpha_0 = 10^{-3}$

memory architecture, independently on the code and on the decoder.

3.6 Conclusion

In this chapter, we provided an analysis of the reliability of the memory architecture proposed by Taylor [24] and Kuznetsov [46]. We introduced a new time-dependent memory degradation model. We expressed the successive error probabilities in the memory and we introduced a threshold definition to characterize the set of memory degradation parameters and decoder noise parameters that lead to a reliable memory. From the reliability analysis, we designed the code and decoder parameters in order to minimize the redundancy per time unit of the memory architecture.

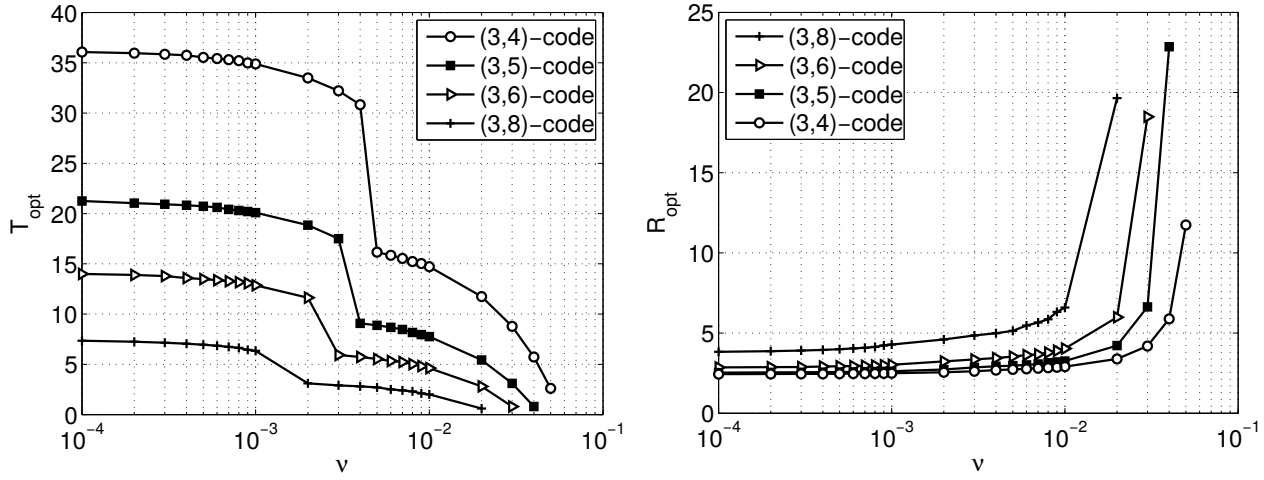


Figure 3.5: Parameter optimization with respect to ν for various codes, for $\alpha_0 = 10^{-3}$, (a) Optimal refresh time (b) Minimum redundancy

Chapter 4

Bit-flipping decoding under data-dependent gate failures

Abstract: *In this chapter we present new result related to the bit-flipping decoding under data-dependent gate failures. We improve the results presented in Deliverable 4.2 providing a less stricter condition for the guaranteed error correction of the bit-flipping (BF) decoder. We show that* $(\gamma, \rho, \alpha, (7/8 + \epsilon)\gamma)$*,* $\epsilon > 0$ *expander codes guarantee that a fixed fraction of errors can be corrected using the BF decoder whose check-node operations are prone to data-dependent gate failures. In our previous pork we were able to prove that only* $(\gamma, \rho, \alpha, (15/16 + \epsilon)\gamma)$ *expander codes have the above characteristic.*

4.1 Introduction

Guaranteed error correction of LDPC codes has been only studied for the iterative decoders built from reliable components. Sipser and Spielman [67] showed that expander LDPC codes can be conveniently used to guarantee the correction of a fraction of errors, i.e. there exist some α, $0 < \alpha < 1$, for which the decoder can correct αn worst case errors, where n is the code length. They proved that both serial and parallel bit-flipping algorithms can correct a fixed fraction of errors if the underlying Tanner graph is a good expander. In the later work Burshtein [68] generalized their results and proved that a linear number of errors can be corrected by the parallel bit-flipping algorithm with almost all codes in $(\gamma \geq 4, \rho > \gamma)$-regular ensemble. The expander graph arguments can be also used to provide guarantees of the message passing algorithms, at it was shown by Burshtein and Miller [69] and linear programming shown by Feldman *et al.* [70]. Recently, Chilappagari *et al.* [71] provided another look on the guaranteed error correction of the bit-flipping algorithms. They found the relation between the girth of the Tanner graph and the guaranteed error correction capability of an LDPC code.

In this chapter we examine the effects of data-dependent gate failures to performance of the bit-flipping decoding. We investigate the error correction capabilities of the noisy bit-flipping decoders and show that expander graph arguments can be used to establish lower bounds on the guaranteed error correction capability in the presence of data-dependent gate failures.

The rest of the chapter is organized as follows. In Section 4.2 the preliminaries on codes on graphs as well as the memory architecture are discussed. In addition, in the same section we give a description of data-dependent approach to gate failure modeling. The error correction capability of the noisy bit-flipping decoder is investigated in Section 4.3. The numerical results are presented in Section 4.4. Finally, some concluding remarks and future research directions are given in Section 4.5.

4.2 Preliminaries

4.2.1 Codes on Graphs and the Bit-Flipping Decoding

Let $G = (U, E)$ be a graph with a set of nodes U and a set of edges E. An edge e is an unordered pair (v, c), which connects two *neighborly* nodes v and c. The cardinality of U, denoted as $|U|$, represents the order of the graph, while $|E|$ defines the size of the graph. A set of neighbors of a particular node u is denoted as $\mathcal{N}(u)$. The number of neighbors of a node u, denoted as $d(u)$, is called the degree of u. The average degree of a graph G is $\bar{d} = 2|E|/|U|$.

The girth g of a graph G is the length of smallest cycle in G. A bipartite graph $G = (V \cup C, E)$ is a graph constructed from two disjoint sets of nodes V and C, such that all neighbors of nodes in V belong to C and vice versa. The nodes in V are called variable nodes and nodes from C are check nodes. A bipartite graph is said to be γ-left-regular if all variable nodes have degree γ, and similarly, a graph is ρ-right-regular if all check nodes have degree ρ.

Consider a (γ, ρ)-regular binary LDPC code of length n and its graphical representation given by γ-left-regular and ρ-right-regular Tanner bipartite graph G, with $n\gamma/\rho$ check nodes and n variable nodes. We consider expander codes, i.e. LDPC codes whose Tanner graphs satisfy expansion property defined as follows.

Definition 3. *[67] A Tanner graph G of a (γ, ρ)-regular LDPC code is a $(\gamma, \rho, \alpha, \delta)$ expander if for every subset S of at most αn variable nodes, at least $\delta|S|$ check nodes are incident to S.*

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a codeword of a binary LDPC code, which appears at the input of a binary symmetric channel (BSC). The output of the channel $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, where $\Pr\{r_k \neq x_k\} = p$, is being decoded by our *majority logic decoder*. The number of flipped bits represents the Hamming distance between the transmitted codeword $\mathbf{x}$ and the received word $\mathbf{r}$, and is denoted as $d_H(\mathbf{x},\mathbf{r})$. The decoder is divided into *processing units* that correspond to nodes in Tanner graph representation of the decoder. Let $\overrightarrow{m}_i(e)$ and $\overleftarrow{m}_i(e)$ be messages passed on an edge e from variable node to check node and check node to variable node, during the i-th decoding iteration, respectively. Similarly $\overrightarrow{m}_i(F)$ and $\overleftarrow{m}_i(F)$ denote sets of all messages from/to a variable node over a set of edges $F \subseteq E$. We next summarize our majority logic decoder.

- At iteration $i = 0$ the variable-to-check messages are initialized by using values received from the channel, i.e. $\overrightarrow{m}_i(e) = r_v$, $\forall e \in \mathcal{N}(v)$. At iteration i, $i > 0$, a variable node processing unit v performs the majority voting on binary messages received from its neighboring check nodes as follows

$$\Phi(\overleftarrow{m}_{i-1}(\mathcal{N}(v))) = \begin{cases} s, & \text{if } |\{e' \in \mathcal{N}(v) : \overleftarrow{m}_{i-1}(e') = s\}| > \lceil \gamma/2 \rceil, \\ r_v, & \text{otherwise,} \end{cases} \tag{4.1}$$

  where $s \in \{0, 1\}$ and $\lceil \gamma/2 \rceil$ denotes the smallest integer greater than or equal to $\gamma/2$. The output of the majority logic (MAJ) gate, described by the function $\Psi(\cdot)$ is then passed to all neighboring check nodes, i.e $\overrightarrow{m}_i(e) = \Phi(\overleftarrow{m}_{i-1}(\mathcal{N}(v)))$, $\forall e \in \mathcal{N}(v)$.

- During each iteration i, $i \geq 0$, a check node processing unit c performs ρ eXclusive-OR (XOR) operations defined as follows

$$\Psi(\overrightarrow{m}_i(\mathcal{N}(c) \setminus \{e\})) = \bigoplus_{e' \in \mathcal{N}(c) \setminus \{e'\}} \overrightarrow{m}_i(e'), \quad \forall e \in \mathcal{N}(c). \tag{4.2}$$

  The results of the XOR operations represent estimates of bits associated to neighboring variable nodes and they are passed by mapping $\overleftarrow{m}_i(e) = \Psi(\overrightarrow{m}_i(\mathcal{N}(c) \setminus \{e\}))$, $\forall e \in \mathcal{N}(c)$.

If the decoding is terminated after the i-th iteration, the result of $\Phi(\overleftarrow{m}_i(\mathcal{N}(v)))$ represents the decoded bit x_v. Note that, when built from perfectly reliable logic gates, our decoder is functionally equivalent to the parallel bit-flipping decoder [67]. Hardware unreliability in the decoder comes from unreliable

computation of the operations $\Psi(\cdot)$ as XOR logic gates performing these functions are prone to data-dependent failures, which are described in the following section. We assume that gate failures are described by the GOS model, defined in the following subsection.

After each decoding iteration, the code bits are estimated based on the function $\Phi(\cdot)$, which results in probability of error of an estimated bit that is greater than or equal to the probability of failure of the MAJ gate performing this function. Since the error probability of the MAJ gate lower bounds the BER performance, MAJ gates must be made highly reliable. Otherwise, the probability of error would be determined by this gate, not by the error control scheme. Thus, it is reasonable to make an assumption that MAJ gates are perfect and that only the XOR gates are faulty. Reliable MAJ gates can be realized, for example, by using larger transistors. Similar assumptions regarding perfect gates were also used in other relevant literature [2, 24, 26].

When the decoding is terminated after only one iteration, and a bit x_v is decoded by $\Phi(\overleftarrow{m}_0(\mathcal{N}(v)))$, our decoder is reduced to the known One-Step Majority Logic (OS-MAJ) decoder, recently analyzed in our previous works [26, 58].

4.2.2 Gate-Output Switching Probabilistic Failure Model

Failures of subpowered CMOS logic gates are data-dependent and correlated in time and cannot be represented in the same manner as memory failures [56, 72]. In this letter we consider the *probabilistic gate-output switching model* (GOS), recently proposed by Amaricai *et al.* [72], which assumes that a gate failure depends on switching activity of the gate.

Let $z(\ell\tau)$ be the correct output of a logic gate at time $\ell\tau$. Due to unreliability of the gate, the actual output is $z(\ell\tau) \oplus \xi(\ell\tau)$, where $\xi(\ell\tau) \in \{0, 1\}$ is the error at time $\ell\tau$. In the GOS error model, when the correct gate output is unchanged during two consecutive time instants, the actual gate output is always correctly computed, i.e., $\Pr\{\xi(\ell\tau) = 1 | z(\ell\tau) = z((\ell-1)\tau)\} = 0$. On the other hand, the gate fails to switch with probability $\Pr\{\xi(\ell\tau) = 1 | z(\ell\tau) \neq z((\ell-1)\tau)\} = p_g$, $p_g > 0$, $g \in \{\oplus, \gamma\}$, where with slight abuse of notation γ signifies a γ-input MAJ gate. In general, the failure probability of XOR gates $p_\oplus$ can be different from the failure probability of MAJ gates p_γ. Although the GOS model does not capture all the effects that lead to failures of subpowered CMOS circuits, it was shown that loss of accuracy by using this modeling approach is relatively small [72]. The GOS model has been studied recently in a number of papers [26, 73, 74].

4.3 Guaranteed Error Correction under the GOS Error Model

In this section we prove that the correcting capability of the iterative majority logic decoder, built partially from unreliable gates, increases linearly with code length, when Tanner graph of a code satisfies the expansion property, defined in Section 4.2.1. We assume that following two conditions are satisfied: (i) the MAJ gates used in the decoder are reliable, and XOR failures follow the error mechanism introduced in Section 4.2.3, and (ii) no more than $|C_{XOR}|$ gates are erroneous in the first iteration. The need for previously described assumptions will be discussed later. Now we formulate the theorem that gives the error correction capability of the noisy majority logic decoder.

Theorem 2. *Consider a* $(\gamma, \rho, \alpha, (7/8 + \epsilon)\gamma)$ *expander,* $1/8 \geq \epsilon > 0$. *The majority logic decoder built from unreliable check nodes can correct any pattern of* $|V_1| < \left(3(3 + 8\epsilon)\alpha n/32 - \sqrt{2}|C_{XOR}|\right)$ *errors.*

Note that the decoder's correcting capability depends not only on the expansion property of its Tanner graph, but also on the number of XOR failures in the first iteration ($|C_{XOR}|$). For too many XOR gate failures during the first iteration, the decoding process will not converge to a correct codeword. Recall from the GOS error model that $|C_{XOR}|$ depends on the XOR gates failures at time instant prior to the first decoding iteration. We do not have any control over the number of XOR gate failures before decoding has started, but there is a practical way to overcome this, and force $|C_{XOR}|$ to be zero. Before we start decoding a new codeword we can force all transistor-level transient processes in the decoding circuitry to reach a stationary state, so that there are no transitions at gate outputs

nor accumulated errors, prior to the start of decoding. Practically, this can be done by slightly slowing down the clock in the first iteration and letting the signal level stabilize. Since the clock is slower, there are no-timing errors and the XOR computations are reliable, which yields $|C_{XOR}|$=0.

We next compare our results with the results from [67] where a reliable decoder was considered. It can be observed that the presence of the XOR gate failures reduces the number of errors that can be tolerated by the bit-flipping decoder. For example, when the Tanner graph has the expansion of $(7/8+\epsilon)$, the perfect decoder can correct $9/16\alpha n$ errors, which is two times higher than the error correction capability of the faulty decoder. In the limiting case $\epsilon = 1/8$ the number of correctable errors is upper bounded by $3\alpha n/8$, which is only the 3/8 of the number of errors correctable by the decoder built from reliable components.

The problem of explicit construction of expander graph, with the expansion arbitrary close to γ (called *lossless* expanders), was investigated by Capalbo *et al.* in [75], where it was shown that the required expansion $7/8+\epsilon$ can be achieved with graph left-degree $\gamma = \text{poly}(\log(\gamma/\rho), 8/(1-8\epsilon))$. This proves the existence of a expander code that can tolerate a fixed fraction of errors under data-dependent gate failures.

Another proof of the guaranteed error correction of LDPC codes was provided by Chilappagari *et al.* in [71], where the correction capability of an LDPC code was expressed in terms of girth of Tanner graph. In the following theorem we extend the results presented in [71] to the case of the noisy decoder.

Theorem 3. *Consider an LDPC code with γ-left-regular Tanner graph with $\gamma \geq 8$ and girth $g = 2g_0$. Then, the majority logic decoder built from unreliable check nodes can correct any error pattern $|V_1|$ such that $|V_1| < 9n_0(\gamma/4, g_0)/32 - \sqrt{2}|C_{XOR}|$, where*

$$n_0(\gamma/4, g_0) = n_0(\gamma/4, 2j+1) = 1 + \frac{\gamma}{4}\sum_{i=0}^{j-1}\left(\frac{\gamma}{4}\right)^i, \ g_0 \quad odd,$$

$$n_0(\gamma/4, g_0) = n_0(\gamma/4, 2j) = 2\sum_{i=0}^{j-1}\left(\frac{\gamma}{4}\right)^i, \ g_0 \quad even. \tag{4.3}$$

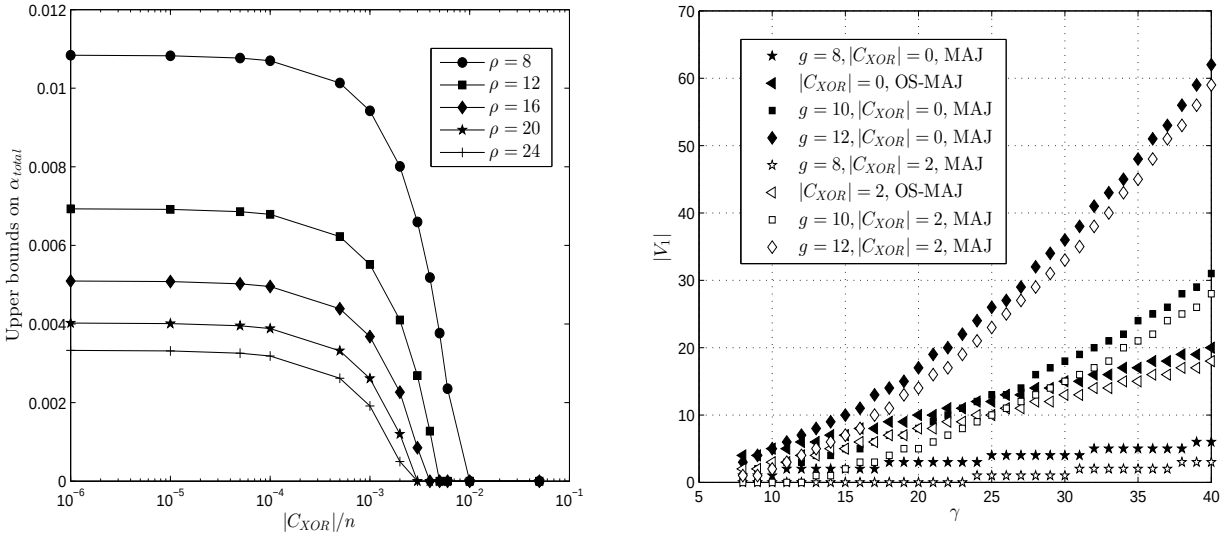
Note that it was shown in [71] that $\gamma \geq 4$ represents a sufficient condition for the guaranteed error correction on a Tanner graph with girth g. However, due to logic gate failures higher expansions are required compared to the perfect decoder, but the other conclusions remain the same as for the perfect decoder.

4.4 Numerical Results

From Theorem 3 follows that the number of errors that can be corrected depends on the expansion property, represented by α and ϵ, and the hardware failures inherited from the time instant prior to the decoding, $|C_{XOR}|$. Here we provide an upper bound on a fraction of channel errors, $\alpha_{total} = 3(3+8\epsilon)\alpha/32 - \sqrt{2}|C_{XOR}|/n$, that can be corrected by the decoder. We use the following lemma to numerically obtain the upper bound.

Lemma 1. *Let assume the existence of a $(\gamma, \rho, \alpha, (7/8+\epsilon)\gamma)$, $\epsilon > 0$ expander. Then, when code length goes to infinity, α and ϵ must satisfy $\epsilon \leq (1-(1-\alpha)^\rho)/(\alpha\rho) - 7/8$.*

In Fig. 4.1(a) we express $\alpha_{total}(\alpha^*, \epsilon^*)$ in terms of $|C_{XOR}|/n$, for different ρ-right-regular Tanner graphs. We consider only cases where $\rho \geq 8$. We can observe that, for example for $\rho = 8$, when the influence of inherited failures can be neglected, we can potentially correct more than 1% of erroneous bits. In addition, a code correction capability reduces with the increase of ρ. When XOR gate failures prior to the decoding become comparable with the correction capability of a code, a *threshold* is reached and the bound rapidly decreases. The threshold is independent of ρ. For sufficiently large $|C_{XOR}|/n$ the decoder performance is degraded up to the point where no error correction can be guaranteed. This happens, for example for $\rho = 8$, when $|C_{XOR}|/n \geq 1\%$.



(a) (b)

Figure 4.1: Guaranteed error correction under GOS error model: (a)Maximal tolerable fraction of errors; (b) Number of tolerable errors.

Another perspective on the error correction of the noisy decoders is provided in Fig. 4.1(b). Here we examine how the girth of γ-left-regular Tanner graphs affects the decoder performance. In addition, we compare the results given by Theorem 3 with the correction capability of the noisy OS-MAJ decoder, expressed by $\lfloor \gamma/2 \rfloor - |C_{XOR}|$. It can be observed that the error correction bound, guaranteed by Theorem 3, for small girth ($g \leq 8$), is not tight. It is actually lower compared to the known OS-MAJ decoder correction capability. However, for higher girths of Tanner graphs, the results given in Theorem 3 are significant. For example, when $g = 12$, $|C_{XOR}| = 0$ and $\gamma = 12$, we can guarantee correction of error patterns with weight 7, which is not possible using the OS-MAJ decoder.

4.5 Conclusion

Based on the expander properties of Tanner graphs, we established conditions required that correction capability of the majority logic decoder increases linearly with the code length. Although we were able to show that this property is achievable for codes with high left- and right-degrees, our results present the first known results regarding the guaranteed error correction of LDPC decoders made of unreliable components.

Chapter 5

Reliability of Memories Built from Unreliable Components under Data-Dependent Gate Failures

Abstract: *In this chapter we investigate fault-tolerance of memories built from unreliable cells. In order to increase the memory reliability, information is encoded by a low-density parity-check (LDPC) code, and then stored. The memory content is updated periodically by the bit-flipping decoder, built also from unreliable logic gates, whose failures are transient and data-dependent. Based on the expander property of Tanner graph of LDPC codes, we prove that the proposed memory architecture can tolerate a fixed fraction of component failures and consequently preserve all the stored information, if code length tends to infinity.*

Work presented in this chapter has been published in: S. Brkic, P. Ivanis, and B. Vasic, "Reliability of memories built from unreliable components under data-dependent gate failures", *IEEE Communication Letters*, vol 19, no. 12, pp. 1089-7798, December 2015 [P3]

5.1 Introduction

Recent research in the area of fault-tolerant memories based on decoders of low-density parity-check (LDPC) codes is mainly inspired by the studies presented in the late sixties and early seventies by Taylor [24] and Kuznetsov [46]. In their pioneering works, they proposed a memory architecture built entirely from unreliable components, which is capable of preserving stored information over arbitrary long time. The memory is composed of unreliable memory cells that are storing a codeword of an LDPC code, and are periodically updated using a faulty iterative decoder. Attractiveness of using LDPC codes lays in the theoretical guarantee that the decoding hardware overhead required to ensure reliable operation grows only linearly with the code length even when logic gates are faulty [24]. It was observed by Vasić *et al.* in [23] that an update cycle corresponds to one iteration of the Gallager-B decoder, built from unreliable logic gates. In the later work, Varshney [2] used density evolution analysis to prove that a memory based on the Gallager-A decoder is also capable of preserving information in asymptotic code length. Chilappagari and Vasić [59] used the expander arguments to show the existence of a reliable memory based on the bit-flipping algorithm. The reliability of the same architecture was recently studied by Dupraz *et al.* in [76].

Both Taylor and Kuznetsov as well as most of the related work modeled logic gate unreliability as transient independent failures, originally introduced by von Neumann [45]. Although the simplicity of this model makes it attractive for theoretical analysis, it is unrealistic. In practice, unreliability of logic gates is strongly data-dependent and correlated in time. One of the most dominant effects impacting reliability of logic gates built in energy-efficient subpowered CMOS technologies comes from

the so-called timing violations, which depend on a gate's switching activity [56, 72]. Their effects to different hard decision decoders have been recently studied in [26, 73, 74].

In this letter we establish a bound on a number of correctable errors for a memory system which employ the bit-flipping decoder. Unlike in the prior research [59,76], we evaluate the memory reliability in the presence of data-dependent gate failures. Following the recent work by Brkic *et al.* [74] on guaranteed error correction capability of faulty bit-flipping decoders, we prove that our memory can tolerate a fixed fraction of component failures. Consequently, we show that in the asymptotic case the memory can preserve all stored information, which presents the first proof of memory reliability under a failure model other then the von Neumann model. We refine the results presented in [59] by improving conditions required for the memory reliability. In addition to our analytical results, we present numerical results illustrating upper bounds on tolerable fractions of component failures.

5.2 System Model

5.2.1 The Memory Architecture

The information is stored in a memory as a codeword of a (γ, ρ)-regular LDPC code in n memory cells. Each memory cell stores one code bit. In order to preserve the stored codeword, the memory cells are periodically updated, at regular time instants $\tau, 2\tau, \ldots, L\tau, L \in \mathbb{N}$, based on the error correction scheme, described as follows.

Consider a graphical representation of a (γ, ρ)-regular binary LDPC code given by Tanner bipartite graph $G = (V \cup C, E)$, where V is a set of variable nodes (variables), C is a set of check nodes, and E is a set of edges. An edge $e \in E$ is an unordered pair (v, c) which connects two nodes $v \in V$ and $c \in C$. Nodes v and c are called neighbors iff there is an edge between them. Let E_v (E_c) be a set of edges connected to a variable node v (check node c). Then, $|E_v| = \gamma$, $\forall v \in V$, and $|E_c| = \rho$, $\forall c \in C$, where $|\cdot|$ denotes the cardinality.

Let $x_v(t)$ be a value of a memory cell associated to a variable node v at time t. Let $\overrightarrow{m}_\ell(e)$ ($\overleftarrow{m}_\ell(e)$) be messages passed on an edge e from/to variable node to/from check node during an update cycle $\ell\tau$, $\ell > 0$, respectively. The ℓ-th update cycle can be summarized as follows.

- The content of a memory cell v, $v \in V$, at time $\ell\tau - \delta_0$, where δ_0 denotes an infinitesimal duration of time, is passed to the neighboring check nodes, i.e., $\overrightarrow{m}_\ell(e) = x_v(\ell\tau - \delta_0)$, $\forall e \in E_v$.
- Each check node $c \in C$ calculates ρ XOR operations

$$\overleftarrow{m}_\ell(e) = \bigoplus_{e' \in E(c) \setminus \{e'\}} \overrightarrow{m}_\ell(e'), \ \forall e \in E_c.$$

- The content of a memory cell $v \in V$ is updated by using a γ-input majority logic (MAJ) gate as follows

$$x_v(\ell\tau + \delta_0) = \begin{cases} s, \ \text{if } |\{e' \in E_v : \overleftarrow{m}_\ell(e') = s\}| \geq \lceil \frac{\gamma}{2} \rceil, \\ x_v(\ell\tau - \delta_0), \quad \text{otherwise}, \end{cases}$$

  where $s \in \{0, 1\}$ and $\lceil \gamma/2 \rceil$ denotes the smallest integer greater than $\gamma/2$.

Note that during an update cycle a check node calculates estimates of the neighboring variable nodes, rather than the parity check equation, since the value of the bit that is estimated is not used in the calculation. However, the update cycle is functionally equivalent to one iteration of the parallel bit-flipping decoder [67].

Hardware unreliability of the correction scheme comes from unreliable computation of messages $\overrightarrow{m}_\ell(e)$ and $\overleftarrow{m}_\ell(e)$, as logic gates performing these functions are prone to data-dependent failures, which are described in the following subsection.

5.2.2 Failure Models

Two types of hardware components failures are considered in this letter: memory cell failures and logic gate failures. We assume that failures in the memory cells are a consequence of supply voltage variations. These errors are transient and manifest as random flips that corrupt values stored in memory cells without damaging the cells [77]. Hence, we can assume that between two update cycles memory content is transmitted through the binary symmetric channel for which $\Pr\{x_v(\ell\tau + \delta_0) \neq x_v((\ell+1)\tau - \delta_0)\} = p_m$, $\forall v \in V$ and $\ell > 0$. On the other hand, gate failures are data-dependent and modeled by the GOS failure model, presented in Section 4.2.2.

We will first prove that the memory architecture can tolerate a fixed fraction of errors in all components. Then we will use Chernoff bounds to extend our results to the presented probabilistic error models. Namely, in the first part of our proof we assume that a component failure follows the statistics described above, but we allow only a fraction of failures during the interval $((\ell-1)\tau,\ell\tau)$, $\ell > 0$. This means that the number of memory cell failures between two update cycles is bounded by $\alpha_m n$. Similarly, we allow $\alpha_\gamma n$ MAJ logic gates to be faulty, while the rest of $(1-\alpha_\gamma)n$ MAJ gates operate reliably. Note that we do not require reliable XOR gates, i.e., according to the GOS model failure of every XOR gate used in the correction scheme can occur when the gate output changes.

5.2.3 Error correction of bit-flipping decoders

Our proof that the memory can tolerate a fixed fraction of errors in all components relies on expanders, and here we give the necessary lemmas established by Sipser and Spielman [67] and Brkic *et al.* [74] regarding the error correction capability of bit-flipping decoders. Lemma 1 pertains to decoders made of reliable components, while Lemma 2 gives the correction capability of a faulty decoder whose gates fail as described in the previous subsection.

Let V_ℓ be a set of corrupt (erroneous) variables at the beginning of the ℓ-th decoding iteration. The following lemmas depict the error correction capabilities of bit-flipping decoders when the underlying Tanner graph is $(\gamma, \rho, \alpha, (7/8+\epsilon)\gamma)$, $\epsilon > 0$ expander.

Lemma 2. *The parallel bit-flipping decoder built from reliable components can correct any fraction of* $\alpha_r < (3+8\epsilon)\alpha/4$ *errors after at most* $\log_{2/(1-8\epsilon)} \alpha_r n$ *iterations. Also, for every* $|V_1| \leq \alpha_r n$ *and* $\ell > 1$ *holds* $|V_{\ell+1}| \leq (1-8\epsilon)|V_\ell|/2$.

Proof: See [67]. ■

Lemma 3. *The parallel bit-flipping decoder built from unreliable check nodes can correct any fraction of* $\alpha_u < 3(3+8\epsilon)\alpha/32$ *errors. Also, for every* $|V_1| \leq \alpha_u n$ *and* $\ell > 1$ *holds*

$$(1-8\epsilon)|V_\ell| \geq 2|V_{\ell+1}| - (1-8\epsilon)|V_{\ell-1}|. \tag{5.1}$$

Proof: See [74]. ■

5.3 Reliability of the Memory Architecture

5.3.1 Guaranteed Error Tolerance

When the memory is built entirely from unreliable components, the bits read from the memory at some time instant, in the most of the cases will not be the same as in the originally stored codeword. Thus, if we want to recover the information, the final step of codeword extraction must be performed by reliable logic gates. In this letter we follow the system setup proposed by Taylor [24], which states that *memory failure* is declared only if the sequence read from the memory cannot be successfully decoded by the noiseless version of the same decoder in a finite number of iterations. We show that our memory architecture under certain conditions achieves arbitrary low memory failure probability.

Note that logic gate failures at the time of the first update cycle depend on the instant before the codeword is stored in the memory cells. There is a practical approach to resolve this issue by slowing

down the clock in the first update cycle and letting the signal level stabilize [73]. This leads to fully reliable logic gate operations in the first update cycle, which is assumed in our analysis.

We first investigate what fraction of memory failures α_m can be tolerated by our memory if we allow all gates, used in the correction scheme, to be faulty. This is given in the following lemma.

Lemma 4. *The proposed memory architecture built on a (γ, ρ)-regular LDPC code free of four cycles can tolerate a fraction of memory failures if* $\alpha_m \leq \left\lfloor \gamma/6 \right\rfloor / n$.

Proof: Proof was provided in Deliverable 4.2. ■

Note that increasing n forces α_m to reduce, and the number of tolerable memory failures $\alpha_m n$ cannot exceed $\lfloor \gamma/6 \rfloor$. This means that under this conditions the arbitrary small memory failure probability can be achieved only if γ tends to infinity. The main reason for a such behavior lies in the fact that, under the GOS model, failures of MAJ gates can cancel out error correction gain achieved during the update cycle. The only way to prevent this is to allow a number of MAJ gates to operate fully reliable. In other words, we bound a fraction of faulty MAJ gates to α_γ, but we do not put any restrictions on the number of faults in check nodes. They all remain prone to data-dependent failures.

Theorem 4. *The proposed memory architecture based on a $(\gamma, \rho, \alpha, (7/8+\epsilon)\gamma)$ expander code can preserve all stored bits for an arbitrary long time period if*

$$\alpha_m + \alpha_\gamma < 3\epsilon(3+8\epsilon)\alpha/4. \tag{5.2}$$

Proof: At t=0 a codeword of our expander code is written into the memory. The memory cells are updated at time instants $\ell\tau$, $\ell > 0$, by performing one iteration of the bit-flipping algorithm. Let $V(t)$ be a set of corrupt variables (memory cells) at time t. The number of corrupt variables before the first update $|V(\tau - \delta_0)|$ is bounded by

$$|V(\tau - \delta_0)| \leq n\alpha_m.$$

After the update cycle we have

$$|V(\tau + \delta_0)| \leq \beta\alpha_m n + \alpha_\gamma n,$$

where, according to Lemma 2, $\beta = (1-8\epsilon)/2$. In the time interval $(\tau, 2\tau)$ there can be at most $\alpha_m n$ memory cells failures, which in the worst case will lead to $\alpha_m n$ additional corrupt variables. Then,

$$|V(2\tau - \delta_0)| \leq \beta\alpha_m n + \alpha_m n + \alpha_\gamma n.$$

Based on Eq. (5.1) and the previous discussion for all $\ell > 1$ we obtain

$$|V((\ell+1)\tau - \delta_0)| \leq \beta\Big(|V(\ell\tau - \delta_0)| + |V((\ell-1)\tau - \delta_0)|\Big) + \alpha_\gamma n + \alpha_m n.$$

From the previous inequality follows that the number of corrupt memory cells can be upper bounded, which is formally presented in the following lemma.

Lemma 5. *The number of corrupt memory cells before the ℓ-th update cycle $|V(\ell\tau - \delta_0)|$, for all $\ell > 0$, satisfies*

$$|V(\ell\tau - \delta_0)| \leq (\alpha_m n + \alpha_\gamma n)/(8\epsilon).$$

Since by Eq. (5.2)

$$(\alpha_m n + \alpha_\gamma n)/(8\epsilon) < (3+8\epsilon)\alpha n/4,$$

from Lemma 4 follows that the number of corrupt memory cells at any time instant does not exceed the error correction capability of the bit-flipping decoder, given by Lemma 1, and the memory content is preserved. This proves the theorem. ■

It is important to note that, in order to prove the memory reliability bound presented in [59], the number of faulty XOR gates had to be bounded. Here we do not need this condition for the data-dependent failure model and the larger number of faulty components can be used in the memory.

Now we utilize Theorem 1 to bound the memory performance under the probabilistic failure model. Let $\Delta_m > 0$ and $\Delta_\gamma > 0$ be such that $p_m + \Delta_m = \alpha_m$ and $p_\gamma + \Delta_\gamma = \alpha_\gamma$. When condition given by Eq. (5.2) is satisfied, the following lemma can be formulated.

Lemma 6. *The probability that memory failure occurs after L update cycles, $P(L)$, is bounded by*

$$P(L) \leq L(e^{-2\Delta_m^2 n} + e^{-2\Delta_\gamma^2 n}).$$

The previous lemma describes a weak bound on the memory performance and its main goal is to show that probability of failure $P(L)$ decreases exponentially when the code length increases. It proves the existence of a memory that can preserve all stored bits in asymptotic code length under the data-dependent gate failure model.

5.3.2 The Complexity Analysis

In this subsection we compare redundances of memories that are based on Bit-Flipping (BF), Gallager A (GA) and Gallager B (GB) decoders, under the same information capacity. The check node operations are common for all three architectures. It is known that a $(\rho-1)$-input XOR gate can be implemented as serial concatenation of $\rho - 2$ 2-input XOR gates. As there are $n\gamma/\rho$ check nodes, the total number of 2-input XOR gates needed for the decoder implementation is equal to $n\gamma(\rho-2)$.

The complexity of variable node operations is equal to nD_γ, where D_γ denotes the complexity of the γ-input MAJ gate. The following lemma bounds D_γ.

Lemma 7. *The complexity of γ-input MAJ gates, $\gamma \geq 4$, satisfies*

$$D_\gamma \leq \binom{\gamma}{\lceil \gamma/2 \rceil} - 1 + \sum_{i=0}^{\lceil \gamma/2 \rceil - 2} \binom{\gamma - i}{\lceil \gamma/2 \rceil - i}. \tag{5.3}$$

Since the number of memory cells is equal to n, the redundancy of the BF-based architecture satisfies

$$\begin{aligned} \mathcal{R}_{BF} &\leq n(1 + D_\gamma + \gamma(\rho-2))/(Rn) \\ &\leq (1 + D_\gamma + \gamma(\rho-2))/((1-\gamma/\rho)). \end{aligned} \tag{5.4}$$

On the other hand, γ $(\gamma-1)$-input MAJ gates need to be implemented in each variable node if the GB-based architecture is used. In this case we have

$$\mathcal{R}_{GB} \leq \gamma(1 + D_{\gamma-1} + \gamma(\rho-2))/((1-\gamma/\rho)). \tag{5.5}$$

In the GA-based architecture γ $(\gamma-1)$-input comparator gates are implemented in each variable node. It is known that the $(\gamma-1)$-input comparator gate can be implemented as $(\gamma-2)$ 2-input comparator gates, which gives the following redundancy [2]

$$\mathcal{R}_{GA} \leq (\gamma\rho - 1)/((1-\gamma/\rho)). \tag{5.6}$$

We illustrate the redundancies of different architectures in Fig. 5.1, for $\gamma = 4$. It can be observed that the GA-based architecture is slightly less complex then the BF-based architecture, while the GB-based architecture requires much higher redundancy. However, the GB-based memory enables the strongest protection against component failures. It can be noted that for all architectures exist optimal code parameters that guarantee minimal redundancy. The lowest redundancy is achieved when the GA-based memory architecture with (3,6)-regular LDPC code is used and it is upper bound to 34. From Taylor's definition follows that the storage capacity satisfies $\mathcal{C} \geq 1/34$ [2].

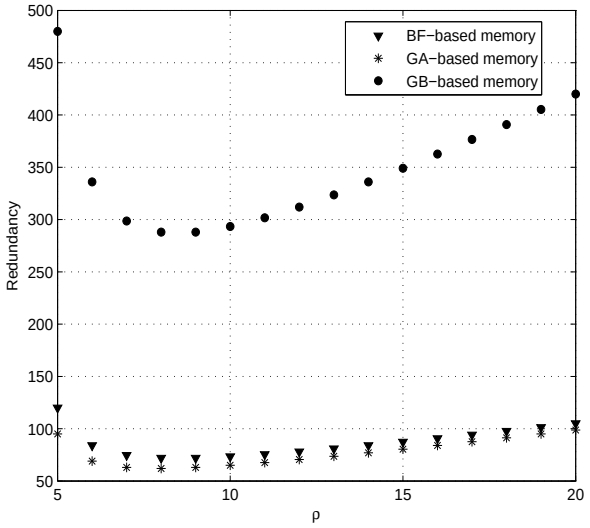


Figure 5.1: Complexities of different memory architectures ($\gamma = 4$).

5.4 Numerical Results

We next show how the right side of the Eq. (5.2), denoted by $\alpha_{total}(\alpha, \epsilon) = 3\epsilon(3 + 8\epsilon)\alpha/4$, can be upper bounded. For that purpose the following Lemma 1 is used.

We can numerically express upper bounds on α_{total}, which satisfy the condition given by Lemma 1, for fixed values of ρ. The bounding values are divided between α_m and α_γ, which creates the tolerable error regions presented in Fig. 5.2. It can be observed that by increasing ρ, under fixed γ, the number of neighbors of a set with αn variable nodes reduces. On the other hand, we require that the set with αn variable nodes have the expansion of more than $7\gamma/8$, which can be only satisfied by reducing α. Consequently, α_{total} is inversely proportional to ρ. For example, when $\rho = 8$ $\alpha_{total} = 0.003$, while for $\rho = 24$ the memory cannot tolerate the fraction of more then $\alpha_{total} = 0.0009$ errors.

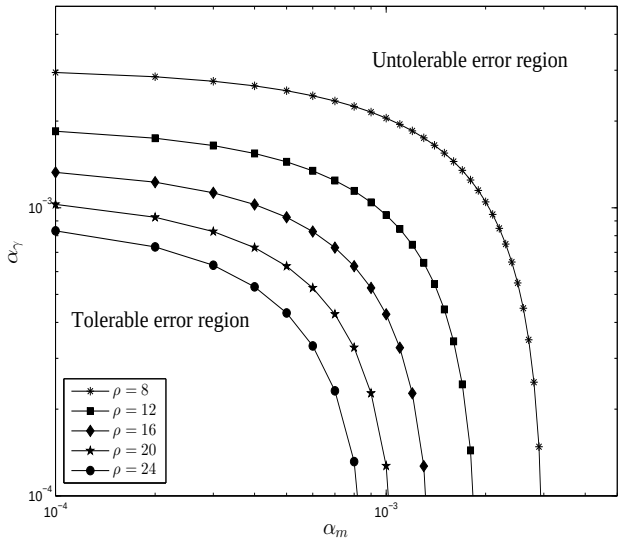


Figure 5.2: Upper bounds on tolerable error fractions.

5.5 Conclusion

In this letter we proved the existence of a memory architecture that achieves arbitrary small failure probability under the data-dependent gate failure model, which presents the first such result under failure models other than the von Neumann model. In addition, we provided upper bounds on fractions of component failures that can be tolerated by our memory.

Chapter 6

Error Resilient LDPC Enhanced 3D-Memory Architecture

Abstract: *In this chapter we introduce a novel error resilient memory architecture potentially applicable to a large range of memory technologies from, e.g., dynamic RAM to FLASH. In contrast with state of the art memory error correction schemes, which rely on (extended Hamming) Error Correcting Codes (ECC), we make use of Low Density Parity Check (LDPC) codes due to their close to the Shannon performance limit error correction capabilities. To allow for a cost-effective implementation we build our approach on the top of a 3D polyhedral memory organization which inherently fast and customizable wide-I/O vertical access allows for a smooth transfer of the required LDPC long codewords to/from an error correction dedicated die. To make the error correction process transparent to the memory users, e.g., processing cores, we propose an online memory scrubbing policy that performs the LDPC-based error detection and correction decoupled from the normal memory operation. After describing the proposed architecture and modus operandi we perform a preliminary performance analysis. Our results indicate that for the same redundancy (amount of parity bits) LDPC enhanced memories exhibit substantially higher fault resilience than state of the art extended Hamming ECC based counterparts.*

6.1 Introduction

Increased integration factor and technology shrinking make Integrated Circuits (ICs) more prone to different defect types during the manufacturing process [78] and to in field degradations [79]. Given that the typical System on a Chip (SoC) area is memory dominated (the ITRS roadmap predicts that the memory growing trend continues [80]), errors occurring in embedded memory systems are a significant threat to the overall SoC correct behaviour. In view of this, powerful but cost effective techniques to detect and correct memory errors are becoming crucial for future SoC related developments [1].

Three dimensional stacked ICs (3D-SICs) based on Through-Silicon-Via (TSV) interconnects ([81]) is an emerging technology which further boost the trends of increasing transistor density and performance, since it enables smaller footprints, high bandwidth low latency interconnection, and heterogeneous integration. Various 3D memory designs and architectures which exploit these benefits have been proposed ever since the technology was introduced [82]. While most of the 3D memory designs just follow a certain folding strategy the 3D polyhedral memory architecture proposed in [83] brings a different fresh view into the field. It consist of multiple identical memory banks stacked on top of each other, while TSVs bundles distributed across the entire memory footprint traverse all the stacked dies to enable an enriched memory access set not achievable in planar counterparts.

In this chapter we propose a novel memory error correction mechanism which takes advantage of the polyhedral memories flexible and powerful data access capabilities. Our approach relies on performing Low Density Parity Check (LDPC) encoding/decoding [84] on large codewords, which can be quickly transferred over the 3D polyhedral memory TSVs to a dedicated die, on which the

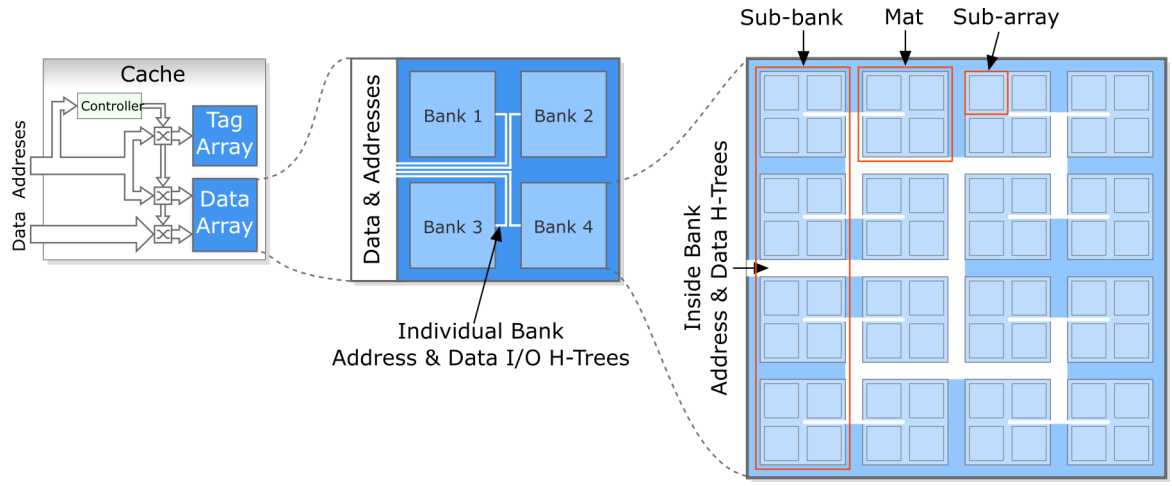


Figure 6.1: Traditional Cache and Memory Array Layout

actual error correction and detection is performed. Given that LDPC decoding is an iterative process An important component of the mechanism consists in an online memory scrubbing policy which enables transparent error detection and correction, i.e., the memory behavior as seen from outside is unaffected. We evaluated our LDPC-based proposal for various codeword lengths and compared its error correction capability against state of the art extended Hamming Error Correcting Codes. Our preliminary results indicate that memories with LDPC-based error correction tolerate considerable more faults when compared to extended Hamming ECC counterparts, while maintaining the same number of redundant parity bits.

The outline of the chapter is the following. In Section 6.2 we provide relevant background information on traditional memory organizations, Hamming based error correction, and Low Density Parity Check codes. In Section 6.3 we detail the implementation and basic operation of polyhedral memories. In Section 6.4 we describe our proposed LDPC-based error correction mechanism for polyhedral memories. Section 6.5 is dedicated to a preliminary analysis of the LDPC-based error correction potential performance and Section 6.6 concludes the chapter.

6.2 Background

In this section we first provide a brief overview of the considered traditional memory organization layout and standard Error Correcting Codes implementations. The section ends with a Low Density Parity Codes review.

6.2.1 Traditional 2D Memories

In order to balance area, delay, and power tradeoffs, large memories are usually constructed in a hierarchical manner. In the following we consider the representative layout employed by the Cacti [85] cache simulator, with a zoom-in into the design abstractions presented in Figure 6.1. At the highest level the address space is split across several identical banks, four in this example, with each bank having its own address and data bus, thus allowing for concurrent bank accesses. Each bank is composed of identical sub-banks, again four in this example, with only one being active per access. Further, each sub-bank is partitioned into multiple mats that simultaneously provide parts of the required data (cache block in a cache data array). Finally, each mat is composed of four identical sub-arrays that share predecoding/decoding logic and peripheral circuitry, and which again deliver together the requested data. An H-tree routing distribution network is used to drive addresses and data to/from the banks, and also to/from every mat inside a bank. This hierarchical memory organization approach is employed, with slightly variations regarding the terminology, at various levels in processors memory hierarchies, implemented in different technologies, i.e., caches ([86], [87]) and main memory ([88]).

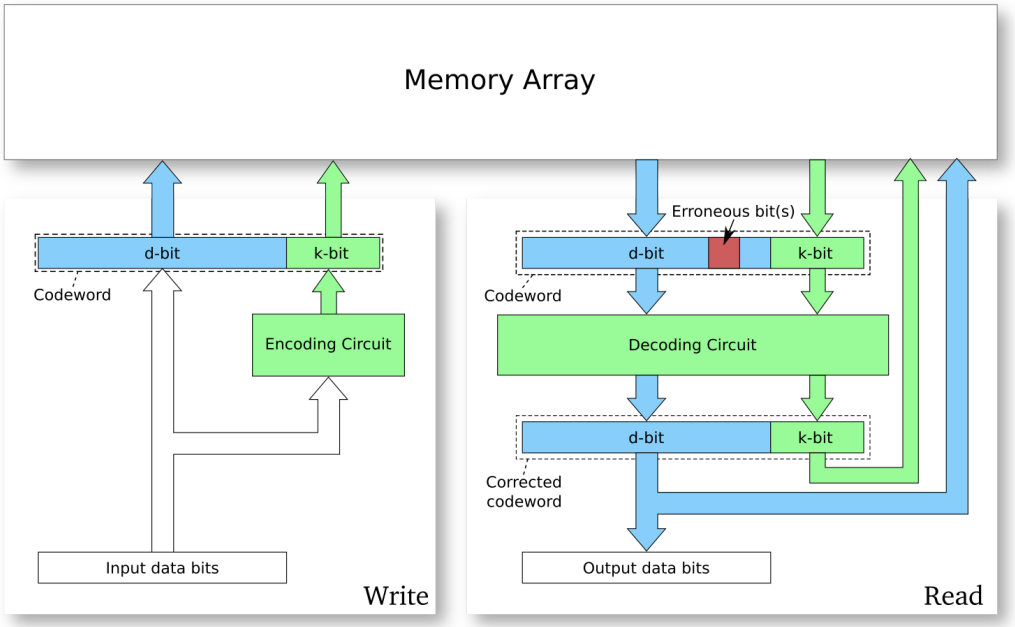


Figure 6.2: Error Detection and Correction Enhanced Memory

Table 6.1: Required parity bits for Hamming SEC-DED [1]

Data bits	4	8	16	32	64	128	256	512	1024	2048	4096
Parity bits	4	5	6	7	8	9	10	11	12	13	14

6.2.2 Error Correcting Codes

With continuous technology scaling employed towards meeting the requirements of increased density, capacity, and performance, memory arrays are more sensitive to environmental aggression and ageing. An increase in the number of bits per chip results in a significant increase in permanent (hard) and transient (soft) errors rate [89–91], which makes mandatory the utilization of Error Correcting Codes (ECC) to address both hard and soft errors [1].

The organization of an ECC enhanced memory is depicted in Figure 6.2. In the top of the usual components such memory requires additional encoding/decoding circuits and each d-bit data word is padded k-bit additional parity bits. The concatenation of the data bits with the parity bits is referred to as a $d + k$ *codeword*. During a write operation, the encoding circuit generates the codeword from the input data bits. The generation rule of the codeword is determined by the employed ECC and it is preferable that the ECC does not change the data bits (as it is the case in Figure 6.2) in order to hide de computation latency associated to the parity bits calculation. During a read operation the read codeword is decoded, the erroneous bits (if any) are identified, and their values corrected before outputting the final data. Optionally, the corrected data and parity bits are written back into the memory array to prevent soft error accumulation.

The detection and correction capabilities depends on the employed ECC and the amount of information redundancy (parity bits) one is willing to store into the protected memory. In Table 6.1 the required number of parity bits for Single-Error Correction (SEC) and Double-Error Detection (DED) are presented for various data widths, when Hamming ECC is employed. One can observe that the amount of storage overhead varies from 100% for 4-bit data down to about 0.34% for 4096-bit data, which suggests that from the error correction point of view wide data organizations are advantageous. However most of the up to date computation platforms, e.g., SoCs, rely on 32 or 64-bit architectures which means that the typical storage overhead for SEC & DED is about 12.5% .

6.2.3 Low Density Parity Check Codes

Low Density Parity Check (LDPC) codes [84], introduced in 1962, and later re-discovered in 1996, are linear block codes which have error correction capabilities very close to the Shannon performance limit, when using iterative decoding schemes [92]. For this reason, LDPC codes have been recently adopted by many communication standards (e.g., WiFi [93], WiMAX [94], DVB-S2 [95], 10GBase-T [96]) and have been employed in various applications (e.g., magnetic recording channels [97], space communications [98]).

Generally speaking, adding parity bits to source information/data bits by means of LDPC encoding, i.e., creating a codeword, is a way to achieve reliable communication through an unreliable channel between two parties located at the two channel ends. Each codeword should satisfy a set of pre-determined parity check equations and parties may only send codewords over the channel. By doing so, the receiving party gets the means to check if the received message is right and if this is not the case to identify and apply the appropriate error corrective changes.

An LPDC code can be represented by a set of Boolean equations, that capture the parity constraints to be satisfied by the bits of any valid codeword while LDPC decoding process attempts to find the closest codeword to the word received via an error prone channel. The quest for the identification of the closest codeword is guided by the parity check equations which can be perceived as an agreement between the two parties in order to conduct reliable communications. If the agreement is violated, it means that something went wrong during the transmission on the unreliable noisy channel and corrective actions are required.

One commonly employed approach for iterative LDPC decoding is the sum-product algorithm, which consists in passing messages (probability values of receiving an "1" bit) over the edges of the bipartite graph [99] associated with the LDPC code in use. A factor graph afferent to any LDPC code, can be generated based on the set of constraints among the code's information and parity bits [99], and defines two main operations: (i) parity checks, that implement the parity-check equations, and (ii) equality checks, that estimate and hold the being "1" probabilities of each and every input bit. Accordingly, the graph nodes are divided into two main groups: Parity-check Nodes (PN), and Variable Nodes (VN). Figure 6.3 illustrates the factor graph of an LDPC code, on which edges messages are exchanged iteratively between the parity nodes and the variable nodes, until either a codeword is found, or a maximum number of iterations is reached.

Given that a memory can be perceived as a communication channel, on which we "send" data during a memory write operation and later on at a different time instance we "receive" data during a memory read operation, LDPC protection may be fit into the landscape. There are however a number of hurdles which precludes its direct utilization for memory error detection and correction as follows:

- LDPC decoding exhibits a variable delay as is it may take a variable number of iterations before a codeword is identified. Thus, if LDPC decoding instead of Hamming ECC is embedded into the error correction framework in Figure 6.2 this will negatively affect the memory latency and predictability and by implication the performance of the entire SoC.
- It is well known that LDPC codes are effective on messages which are at least $1Kb$ long while current SoCs rely on 32 or 64-bit architectures.

However, as we demonstrate in Section 6.4 if the memory system follows the polyhedral architectural template described in Section 6.3, LDPC based error protection can be enabled.

6.3 Polyhedral Memories

Polyhedral memories, introduced in [83], rely on a 3D-stacked memory design as presented in Figure 6.4, where every silicon die is an individually addressable memory bank. Besides the traditional horizontal 2D interface, polyhedral memories have a vertical interface realized through TSV bundles that traverse the entire 3D stack. The TSV bundles are distributed over the entire memory footprint and traverse each mat through its center, being connected to the I/O data lines of each sub-array. In

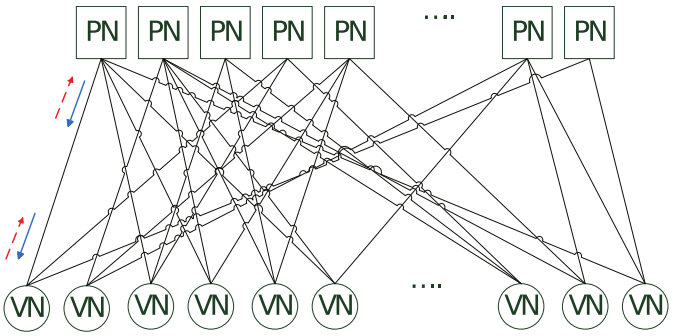


Figure 6.3: LDPC Code Factor Graph

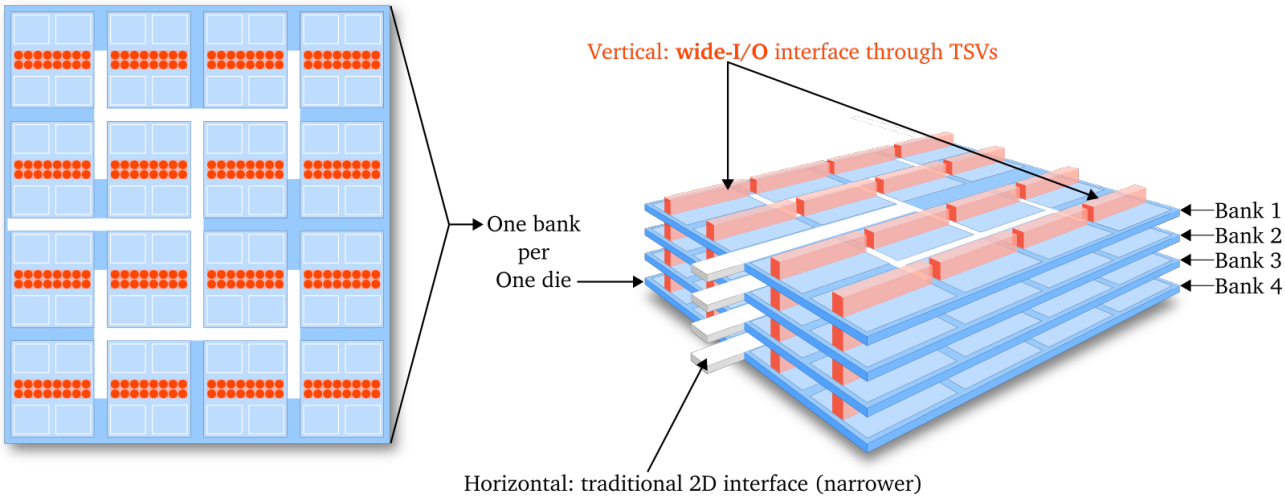


Figure 6.4: Polyhedral Memory Design

this manner a wide-I/O vertical interface is created, having a width equal with the sub-array I/O size multiplied by the number of sub-arrays in a mat. We note that different horizontal and vertical data flows can coexist within the polyhedral memory array as long as they do not conflict on TSV bundles and/or sub-banks.

The sub-array input/output data bit routing logic specific to the polyhedral approach allows for the following operations to be performed:

1. **Local** data access, when the issue and the storage dies are identical. This represents the traditional memory access mode.

2. **Remote** data access, when the issue die is different than the storage die. This requires TSVs utilization to perform operations on data located on a different die.

3. **Vertical (wide-I/O)** access, when the I/O data are transferred to/from the output/input directly on the TSVs. This operation allows data access at different granularities: multiple sub-banks from different banks could be simultaneously vertically accessed.

4. **Inter-die (wide) transfers**, when (large) data blocks need to be copied from one die (read die) to another die (write die). Again, this operation requires TSVs utilization and could be performed at different granularities.

In Figure 6.5 a possible access scenario is depicted, with four concurrent accesses: (i) a local access on sub-bank 3 of bank 1, (ii) a remote access from die 4 on sub-bank 1 of bank 1, (iii) a vertical access on sub-bank 4 of bank 1, and, (iv) another vertical access on sub-bank 2 of bank 4. It can be noticed the multitude of accesses which can be serviced in parallel by a polyhedral memory due to the

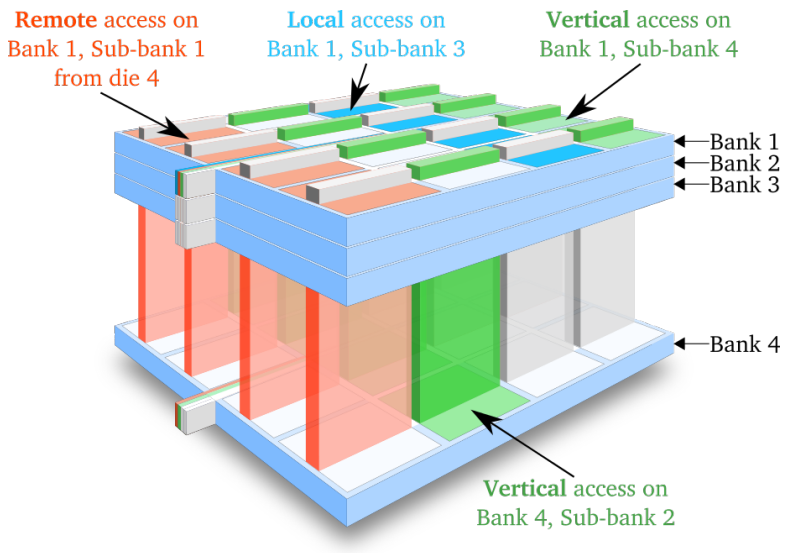


Figure 6.5: Concurrent Access Example in Polyhedral Memories

presence of TSVs bundles. In contrast, on a traditional 2D memory with the same number of banks, only one of the above accesses would have been possible at a time.

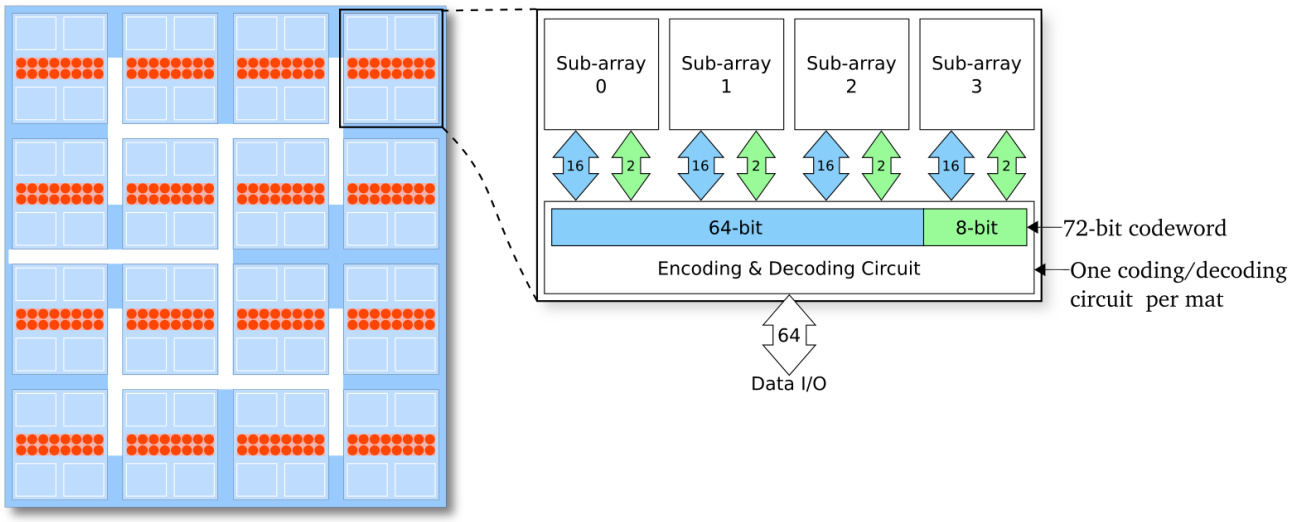


Figure 6.6: Traditional ECC for Polyhedral Memories

6.3.1 Traditional ECC for Polyhedral Memories

An example of a straight forward ECC implementation for polyhedral memories is the design depicted in Figure 6.6. The encoding/decoding circuits are physically located on each mat, and the 72-bit codeword is distributed across all the mat's comprising sub-arrays such that each of them stores 16 data and 2 parity bits. According to Table 6.1, by employing extended Hamming ECC, the correction capability for 64 data bits and 8 parity bits is SEC-DED, i.e., the correction of one error and the detection of two errors are enabled.

In the next section we propose a novel error correction mechanism for polyhedral memories, which better exploits their enriched mechanism access set, and provides a higher error correction capability.

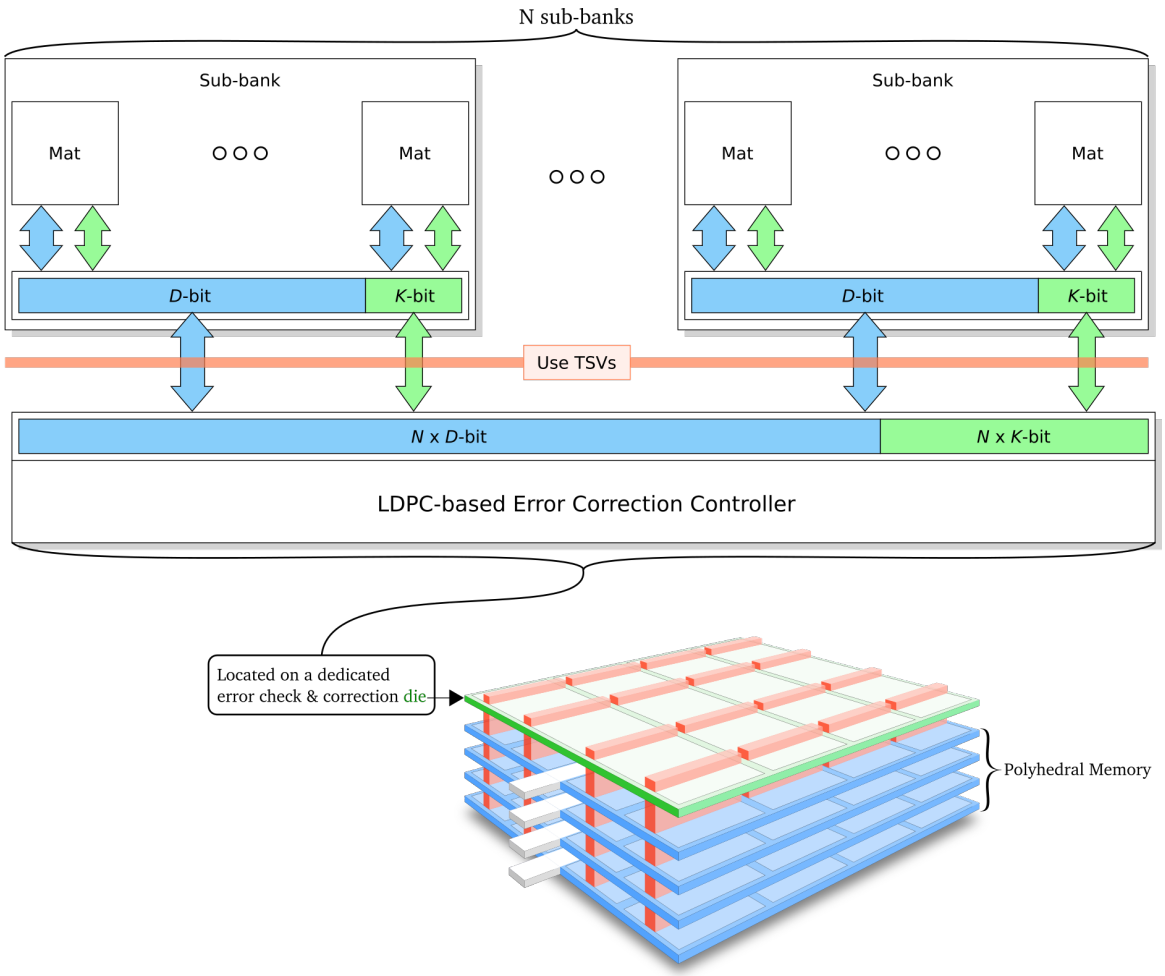


Figure 6.7: LDPC-based ECC for Polyhedral Memories

6.4 LDPC-based ECC for Polyhedral Memories

As detailed in Section 6.3, polyhedral memories allow vertical data access at different granularities: various sub-banks could be accessed in parallel on different dies, with the largest amount of vertically accessed data being obtained when all the TSVs are utilized. In the following we make use of this property to mitigate the issues (discussed at the and of Section 6.2.3) that preclude the direct LDPC utilization in conjunction with state of the art architectures and introduce the LDPC protected memory organization depicted in Figure 6.7. Specifically, we propose to: (i) form extended codewords by combining data and parity (further referred to as check) bits from multiple (N) sub-banks, (ii) augment the 3D memory stack with an extra die dedicated to the execution of the LDPC encoding and decoding actions, (iii) employ TSVs to transfer the extended codewords on/from the dedicated error correction die, and (iv) rely on an online memory scrubbing policy which performs memory maintenance LDPC-based error detection and correction activities without interfering with data requests coming from the SoC computation cores.

The memory maintenance operates in a similar way the DRAM refresh does and in case of conflicts its read/write accesses is has lower priority than normal read/write memory accesses issued by the computation cores. We note that, different from the DRAM refresh, memory scrubbing requires additional steps related to the codeword formation and transfer to/from the LDPC codec die and its coding and/or decoding, thus it is much more time consuming, and by implication maintenance related memory accesses are less frequent than the computation related ones. Consequently, the scrubbing controller can take advantage of: (i) the polyhedral memory reach access mode set and (ii) the low maintenance related accesses occurrence, to dynamically adapt its access schedule such that memory

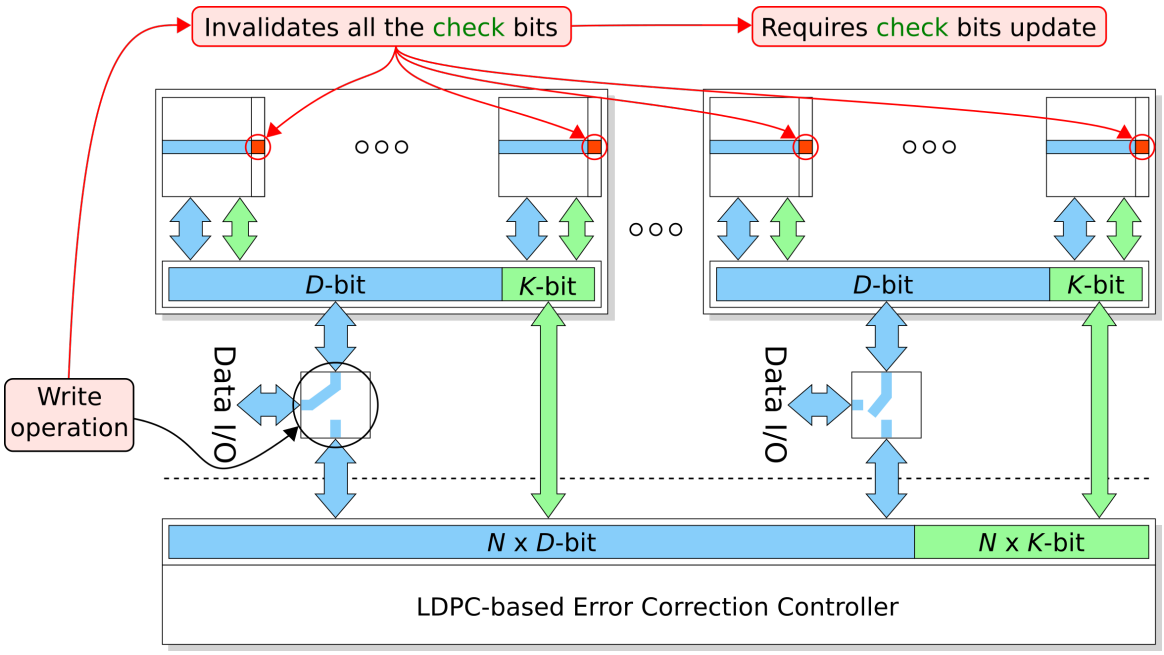


Figure 6.8: Check Bits Write Invalidation

conflicts are minimised if not completely avoided.

Following this policy the entire memory can be scanned within a certain period of time, which depends on: (i) the LDPC codec performance determined by its iteration delay and the number of required iterations, (ii) the bit error occurrence rate which depends on ageing and environmental aggressions, and (iii) the overall memory capacity.

We note that from the SoC point of view, which is the one that really matters, the maintenance is 100% successful when it can keep pace with the error formation rate such that data read generated by the running application(s) never return corrupted data. This doesn't mean that all memory locations should be error free, which is quite normal as memory locations not currently read by the application do not influence its behaviour and may contain erroneous bits.

While the utilization of long codewords spanning over more than one SoC memory word is certainly advantageous as it creates the premises for the LDPC approach to work at its best capabilities thus has some implications on the way normal word granularity data accesses have to be handled.

When a word write is requested at address A_i the changes at that location have potential consequences on all the check bits of the message M_j to which the location belongs. Thus in order to preserve memory integrity/consistency the following actions have to take place when serving a write request at address A_i:

1. Load via the afferent TSV bundle the entire message M_j on the codec die.
2. Interrupt the scrubbing in case only one LDPC codec can be accommodated on the codec die.
3. Decode M_j and correct possible errors if any.
4. Replace the value at A_i with the new to be written value.
5. Re-encode the message M_j.
6. Store M_j back to the memory via the same TSV bundle.

On the other hand when a word read request at address A_i (which is part of message M_j) is issued the following actions have to take place:

1. Load via the afferent TSV bundle the entire message M_j on the codec die.

2. Interrupt the scrubbing in case only one LDPC codec can be accommodated on the codec die.
3. Decode M_j and correct possible errors if any.
4. Send the (corrected) data value stored at A_i to the computation core that issued the request.
5. In case errors were found at Step 3
   - Re-encode the message M_j.
   - Store M_j back to the memory via the same TSV bundle.

While the previously mentioned write and read operations seems very expensive at the first glance their complexity can be substantially diminished in view of the following observations:

- If the scrubbing is successful in preventing error accumulation and correcting errors before their consequences are perceived by the application we can move the LDPC decode/encode out of the memory read process. In this way the read complexity is reduced to the one of a normal memory read and the computation units requests are served without any delay overhead. The read operation mode, i.e., with and without LDPC decoding before read, can be dynamically switched by the application by monitoring the scrubbing effectiveness. In this way the protection level is adjusted to the environmental aggression level and SoC ageing status.
- The 3D memory is constructed by stacking identical memory dies which dimensions determine the IC footprint. The codec die can be as large as the memory dies, thus for state of the art memory capacities it may accommodate more than one LDPC codec. If this is the case Step 2 is not required. Moreover, if more LDPC codecs are available the memory maintenance process can be parallelised which results an a higher error resilience.
- Write requests can be also simplified if M_j recoding is delayed until the scrubbing process arrives to it. This can be done by means of a write invalidation, i.e., when a write operation is performed on a message word, all the check bits of the extended codeword are invalidated, as depicted in Figure 6.8. Depending on the bit error rate we may decide to place the invalidated message in the front of the scrubbing queue or to take no further action as there is a large chance that another write may occur within the same message in the close future. Nevertheless, a read from an invalidated message immediately triggers the reconstruction of the check bits. Note that the write invalidation policy may have an impact on the error correction capability. Eventual bits errors which may appear between the invalidation and the actual check bits update become undiscoverable as when they are read in relation with the message re-coding they will be considered valid.

Finally it is worth noticing that if memory accesses have a larger granularity, e.g., an entire cache line instead of a word, the LDPC (re-)coding overhead is diminishing, and that the read/write complexity can be reduced by, e.g., write buffers, caching.

To get an inside in the potential of our proposal we evaluate in the next section the error correction capability of our approach and compare it with the one of extended Hamming for the same redundancy level.

6.5 Performance Analysis

To evaluate the performance of our proposal, we simulated the following memories:

1. Without error correction capability (**uncoded**).
2. With state of the art 72-bit (64 data bits and 8 parity bits) extended Hamming ECC (**Hamming**).

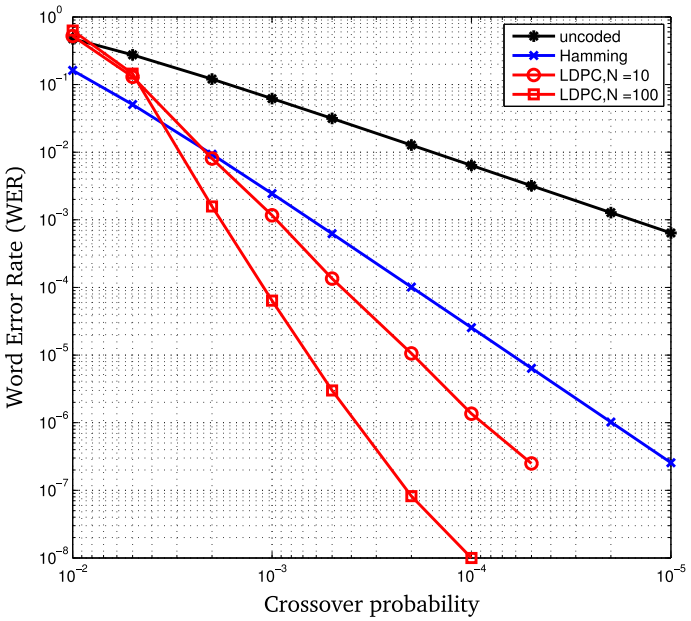


Figure 6.9: LDPC vs Extended Hamming

3. With our proposed LDPC-based based mechanism with $N \times 72$-bit codewords with N values of 10 and 100 (**LDPC**).

In the experiment, we use a regular LDPC code with variable-nodes degree $d_v = 3$. For LDPC decoding we used a modified version of the bit-flipping decoder and allowed for maximum 10 iterations. To simulate memory fault occurrence, we use a Binary Symmetric Channel (BSC) with crossover probabilities (α), i.e., the probability that a memory bit is being flipped, from 10^{-2} to 10^{-5}.

The performance is assessed in terms of Word Error Rate (WER) as plotted in Figure 6.9, where for each simulation point, 10 millions frames have been simulated. From these experimental results, one can observe that:

1. The uncoded memory performs the worst since it has no error correction capability.

2. When a $WER = 10^{-6}$ is targeted, the Hamming protected memory tolerates memory faults up to $\alpha = 2 \times 10^{-5}$ while the LDPC protected ones bear up to $\alpha = 9 \times 10^{-5}$ and $\alpha = 4 \times 10^{-4}$ for $N = 10$ and $N = 100$, respectively.

3. With respect to the same targeted WER, the LDPC based memories with $N = 10$ and $N = 100$ have coding gains of 7×10^{-5} and 3.8×10^{-4} in terms of crossover probability when compared to the memory with Hamming protection, respectively.

6.6 Conclusion

In this chapter we proposed a novel LDPC-based error correction mechanism for polyhedral memories. We proposed to utilize the polyhedral memories inherently fast and customizable wide-I/O vertical access to transfer large LDPC codewords to/from a dedicated error correction die. The proposed mechanism utilizes an online memory scrubbing policy that performs the LDPC-based error detection and correction as transparent as possible to the normal memory operation. We analyzed the error correction capability of our proposal for various LDPC codewords sizes. Our preliminary results indicate that for the same amount of redundancy LDPC enhanced memories can tolerate considerable more faults than the ones equipped with state of the art extended Hamming ECC.

Chapter 7

Reliable Data Transport

Abstract:
This chapter presents our findings related to interconnect performance improvement, in terms of transmission delay, energy consumption and robustness, by means of data encoding techniques. We introduce and evaluate codec assisted data transport structures, e.g., bus segments, Network on Chip (NoC) interconnects, able to deal with technology scaling related phenomena, e.g., crosstalk and transmission delay variability, at the expense of a reasonably small area overhead. In particular we present 3 types of data encoding techniques, namely, Constrained, Repetition, and Haar, apply them on an 8-bit wide interconnect segment, and evaluate their practical implications when using a 45nm commercial CMOS technology. Our simulations indicate that: (i) substantial energy savings can be achieved by properly tuning the codec to the interconnect length and workload data profile and (ii) the Haar codec based approach is the most general purpose one as it outperforms the other ones in most of the considered cases. In view of (ii) we augment the Haar based interconnect with error correction capabilities. We introduce a Single Error Correction and Double Error Detection scheme adapted to the peculiarities of the Haar system, which makes the Haar augmented interconnect not only energy effective but also robust against deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) induced transmission errors.

7.1 Introduction

In this chapter we introduce and evaluate codec assisted energy effective reliable data transport structures, e.g., bus segments, Network on Chip (NoC) interconnects, able to deal with technology scaling related phenomena, e.g., crosstalk and transmission delay variability, at the expense of a reasonably small area overhead.

Given that the amount of energy consumed for the transmission of a certain n-bit wide data workload W on an n-bit interconnect depends on the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions experienced by each interconnect wire, data pre-processing able to diminish the transition count is envisaged. Additionally, due to the fact that crosstalk phenomena among adjacent wires have a negative impact on the data arrival profile (a $0 \rightarrow 1$ switching wire surrounded by $1 \rightarrow 0$ switching wires has a larger than nominal propagation delay) the transmission of crosstalk occurrence favorable data patterns should be avoided. Thus assuming a data workload W, with $T(W)$ transitions and $C(W)$ crosstalk generating data patterns we seek a data transformation $ENC, ENC(W) = W^e$ such that $T(W^e) < T(W)$ and $C(W^e) < C(W)$. Note that ENC has to be reversible, i.e., a DEC function should exist such that $DEC(W^e) = W$.

The general idea behind our method is to improve the performance of a data transport structure by augmenting it with pre-processing logic implementing ENC and post-processing logic implementing DEC. This approach could be successful only if the overall energy consumed by the codec augmented interconnect to transport a certain data workload W is smaller than the one consumed by the raw interconnect, which essentially means that the encoding induced energy savings should be larger than the energy consumed by the codec additional hardware. Additionally, ENC and DEC transformations

implementable with low cost hardware and small delay should be sought.

Note that apart of energy savings we are also interested to alleviate time related aspects of the data transport process. By eliminating cross talks we can potentially diminish the data transmission latency (if ECN and DEC delays are small) and obtain a low variability data arrival profile, which is a key issue for interconnect robustness (reliability) in the context of high process parameter variability specific to deep sub-micron fabrication technologies. Moreover, given that deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) increases the interconnects susceptibility to errors it could be of interest to augment ECN and DEC with error correcting features, such that we can also combat errors arising during the data transport process.

7.2 Coding Schemes

This section presents the algorithms and the afferent hardware implementation details for the proposed coding techniques. Aiming to reduce the interconnect energy consumption, as well as to change the switching profile on adjacent wires such that the crosstalk induced delay is minimized, the following 3 modulation codes have been proposed and investigated:

- Constrained LUT-based coding. This coding technique makes use of two look up tables (one for encoding the input data, before transmitting it over the wires, and one for decoding the data at the bus receiving end), such that a consecutive sequence of input data bits are replaced by a codeword with redundant bits when compared to the original bits sequence. Two $4:5$ LUT-based coding systems are proposed in Section 7.2.1, which minimize the bus switching activity, both in time along each wire, and in space between adjacent wires.

- Repetition based coding. In this case, certain data bits are repeated on redundant additional wires, such that the time-wise transitions along each wire are minimized. This scheme is described in Section 7.2.2.

- Haar-based coding scheme. A $2:3$ stage 1 Forward Haar Transform [100] based coding system is introduced in Section 7.2.3, which reduces both the wire self transition count and the coupling transition count between adjacent wires.

7.2.1 Constrained LUT Based Coding

The first codes that we propose belong to the class of constrained codes [101], which in lieu of traditional unconstrained codes, forbid certain patterns in the arrays of coderwords, either in a symmetric or in an asymmetric manner, varying between different directions (e.g., isolated bit location constraint, where every bit should agree with at least one of its four neighbours), such that signal integrity impairments when certain input data patterns occur are alleviated, and a fine tuned interplay between information rate and energy consumption is achieved.

We start by constructing in a heuristic fashion a $4/5$ rate code, denoted by C_v1, which attempts to minimize the number of transitions between 0 and 1 in time. Its 4-input and 5-output corresponding Look-Up Table (LUT) mapping is presented in Figure 7.1.

Subsequently, we construct another $4/5$ rate code which aims to minimize the occurrence of the transitions presented in Figure 7.3. The intuition behind this is that in order to diminish crosstalk effects and energy consumption 0 should not be surrounded by 1s, and the other way around. Two types of transitions are considered, as indicated in Figure 7.3. The LUT-based mapping between input and output corresponding to this code, denoted by C_v2, is presented in Figure 7.2.

We note that we only defined $4/5$ rate codes in an attempt minimize the LUTs size and delay. However this is not limiting their application domain as one can apply those codes to any n-bit interconnect by splitting it $n/4$ 4-bit sections each of them being handled by a $4/5$ LUT. Thus encoding a byte of data requires two $4:5$ LUTs, while at the decoding end, two $5:4$ LUTs are necessitated. We note that one could also construct wider codes as, e.g., $8/10$, but whether this approach is beneficial

Input word	Output codeword
0 0 0 0	0 0 0 0 0
1 0 0 0	1 0 0 0 0
0 1 0 0	1 1 0 0 0
1 1 0 0	0 1 1 0 0
0 0 1 0	1 1 1 0 0
1 0 1 0	0 0 1 1 0
0 1 1 0	0 1 1 1 0
1 1 1 0	1 1 1 1 0
0 0 0 1	0 0 0 0 1
1 0 0 1	1 0 0 0 1
0 1 0 1	1 1 0 0 1
1 1 0 1	0 0 0 1 1
0 0 1 1	1 0 0 1 1
1 0 1 1	0 0 1 1 1
0 1 1 1	0 1 1 1 1

Figure 7.1: C_v1 System LUT-based Mapping.

Input word	Output codeword
0 0 0 0	0 0 0 0 0
0 0 0 1	0 0 0 0 1
0 0 1 0	0 0 0 1 1
0 0 1 1	0 0 1 1 1
0 1 0 0	0 1 1 0 0
0 1 0 1	0 1 1 1 1
0 1 1 0	1 0 0 0 0
0 1 1 1	1 0 0 0 1
1 0 0 0	1 0 0 1 1
1 0 0 1	1 0 1 1 1
1 0 1 0	1 1 0 0 0
1 0 1 1	1 1 0 0 1
1 1 0 0	1 1 0 1 1
1 1 0 1	1 1 1 0 0
1 1 1 0	1 1 1 0 1

Figure 7.2: C_v2 System LUT-based Mapping.

when applied to an n-bit interconnect is questionable as the delay of an 8/10 LUT is most likely larger than the one of a 4/5 LUT.

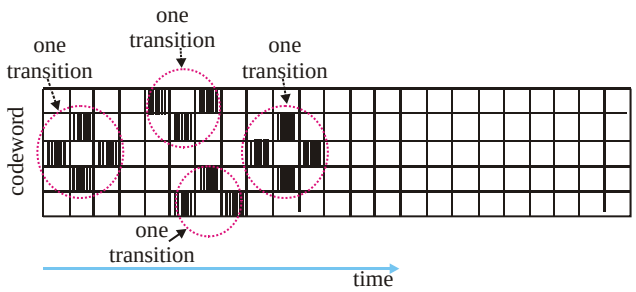


Figure 7.3: Transition Types Targeted for Minimization.

7.2.2 Repetition Based Coding

A second category of investigated coding techniques relies on signal repetition. Following this extremely simple principle redundant wires are appended to the to initial interconnect structure, such that certain transitions unfavorable from the energy point of view are alleviated, and thus energy savings enabled. Assuming a byte-wise transmission, we propose to add 4 additional wires, the encoder

code, denoted subsequently as R_v1, being governed by the following equations:

$$
\begin{aligned}
y_1 &= x_0 \\
y_2 &= x_1 \\
y_3 &= x_1 \\
y_4 &= x_2 \\
y_5 &= x_3 \\
y_6 &= x_3 \\
y_7 &= x_4 \\
y_8 &= x_4 \\
y_9 &= x_5 \\
y_{10} &= x_6 \\
y_{11} &= x_6 \\
y_{12} &= x_7,
\end{aligned}
$$

where $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ is the input data byte desired to be transmitted over the wires, and $\{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}\}$ represents the encoded sequence.
From the hardware implementation standpoint, as the encoding scheme is hardwired, as illustrated in Figure 7.4 the hardware complexity is minimal, which positively impacts the transmission delay and energy consumption figures.

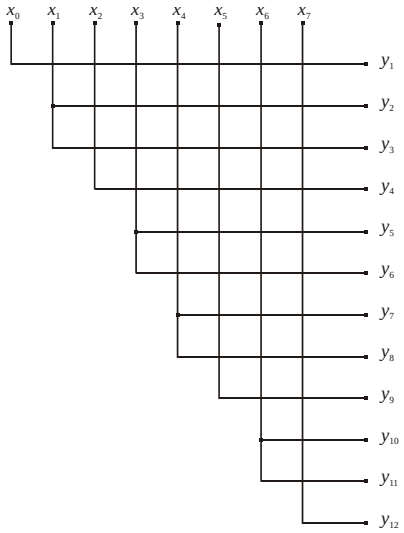


Figure 7.4: R_v1 System Hardwired Encoding.

7.2.3 Haar Based Coding

As in the case of the previous coding techniques described in Sections 7.2.1 and 7.2.2, the foremost goal is on energy reduction, more specifically on reducing the energy consumption per interconnect (bus) transaction. In particular, this is achieved via the minimization of the transition activity between adjacent wires, (which in turn reduces the signal distortion in the form of extra current glitches and thus the extra energy consumed), as well as the individual time-wise switching activity on every bus wire. However, not only the energy merits enabled by the coding technique are of interest. Delay and area criteria are also taken into account when constructing the code, as for an energy effective data transport, the encoder and decoder circuitry must not surpass the energy savings offered by the coding technique.

The Haar coding scheme algorithmic aspects are discussed subsequently, together with a succinct presentation of the afferent encoder and decoder architecture.
For the purpose of exposition, we assume that a byte-wise synchronous data transmission is desired but the discussion is general and can be easily extended for other interconnect widths.

7.2.3.1 Haar Encoder

The Haar encoder receives as input per each clock cycle, a data byte subsequently denoted by

$$\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\},$$

and generates as output a 12-bit wide encoded sequence to be sent over the bus wires.
To this end, the input data sequence is divided into 4 groups of two bits as follows:

$$\{x_0, x_1\}, \{x_2, x_3\}, \{x_4, x_5\}, \{x_6, x_7\}.$$

For each such pair of input bits, the encoder performs 1-bit pair-wise addition and subtraction. A block scheme depicting the operating principle of the Haar encoder is given in Figure 7.5.

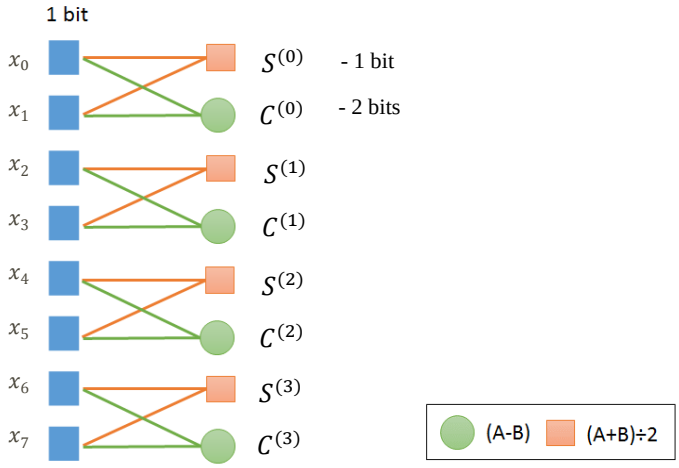


Figure 7.5: Haar Encoder.

Note: Normally, the sum of two bits in $2's$ complement notation requires two bits for representation (as the exact sum value with overflow is required). However, as one may observe, only the MSB bit of the sum (which corresponds to the carry out signal of a 1-bit full adder) is mandatory to be represented. This is because the LSB bit of the sum is the parity bit, which is identical to the LSB bit of the difference (since the sum of two bits has the same parity as their difference). Thus, it suffices to compute only the MSB bit of the sum. As concerns the difference of two input bits, it is likewise performed in $2's$ complement and requires a 2-bit representation.

Specifically, for the input bits $\{x_0, x_1\}$, for instance, the encoder computes the following three bits:

$$S^{(0)} = x_0 \wedge x_1 \tag{7.1}$$
$$C1^{(0)} = x_1 \tag{7.2}$$
$$C0^{(0)} = x_0 \vee x_1, \tag{7.3}$$

where $\wedge$ denotes a logical AND operation, and $\vee$ denotes a logical OR operation.

7.2.3.2 Haar Decoder

The Haar decoder receives as input 12 bits of encoded data:

$$\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}\right\},$$

and outputs the data byte:

$$\{\hat{x}_0, \hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6, \hat{x}_7\},$$

which coincides with the original transmitted data byte

$$\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\},$$

under the assumption of an error-free transmission.
Conceptually speaking, if we know the sum and difference of two numbers, we can easily compute the two numbers in cause. Exemplifying for the pair of bits $\{x_0, x_1\}$, the decoder is governed by the following equations:

$$\hat{x}_0 = S^{(0)} \oplus C1^{(0)} \oplus C0^{(0)} \tag{7.4}$$

$$\hat{x}_1 = S^{(0)} \oplus C1^{(0)}, \tag{7.5}$$

where $\oplus$ denotes a logical XOR operation. For an error-free transmission $\hat{x}_0 = x_0$ and $\hat{x}_1 = x_1$.

The following architectural related observations are in order:

- Implementation Complexity, Delay and Energy.
  As concerns the hardware implementation, both the Haar encoder and decoder consist of one logic level (one gate), and thus exhibit very low complexity. As a result, the Haar encoder/decoder delay is very small, i.e., the delay of a single logic gate for the encoder and of two logic gates for the decoder. Furthermore the codec simplicity has positive implications on the energy consumed by the encoder/decoder (i.e., the energy of OR/AND gates for the encoder, and XOR gates for the decoder), which is an important desideratum when aiming for energy-effective data transport.

- Scalability to larger interconnects.
  The low hardware complexity enables its usage for larger interconnects, as it scales linearly with respect to the number of wires (e.g., for 4 wires, the encoder requires 2 parallel OR gates and 2 parallel AND gates; for 8 wires, it requires 4 OR gates and 4 AND gates).

7.3 Simulation Results

To get inside on the practical implications of the codec enhanced data transport approach we evaluate by means of SPICE simulations the following systems (classified using as criterium the number of employed wires):

- One 8-wire reference system, denoted subsequently as "Ref".
  For this scheme, uncoded, raw data are transmitted over the wires. The 8-wire system serves as comparison reference from the timing and energy standpoints, as no coding scheme for optimizing the energy and reliability characteristics of the data transmission is applied in this case.

- Two 10-wire systems, denoted by "C_v1" and "C_v2".
  In this setup, the raw data, prior to/after its transmission over the wires, is encoded/decoded as described in Section 7.2.1, making use of $4:5$ LUTs.

- Two 12-wire systems: "H_v1" and "R_v1".
  The two systems are as follows: the first 12-wire system embeds the Haar encoder/decoder whose functional and architectural details are given in Section 7.2.3, while the other 12-wire system makes use of the coding scheme presented in Section 7.2.2.

For each system, the SPICE simulation setup consists of encoders & input buffers, interconnect, and output buffers & decoders. Figure 7.6 depicts the simulation setup for the proposed interconnect codec augmented systems. The setup for the 8-wire uncoded, reference system is similar, with the exception of the encoder and decoder blocks which are excluded.

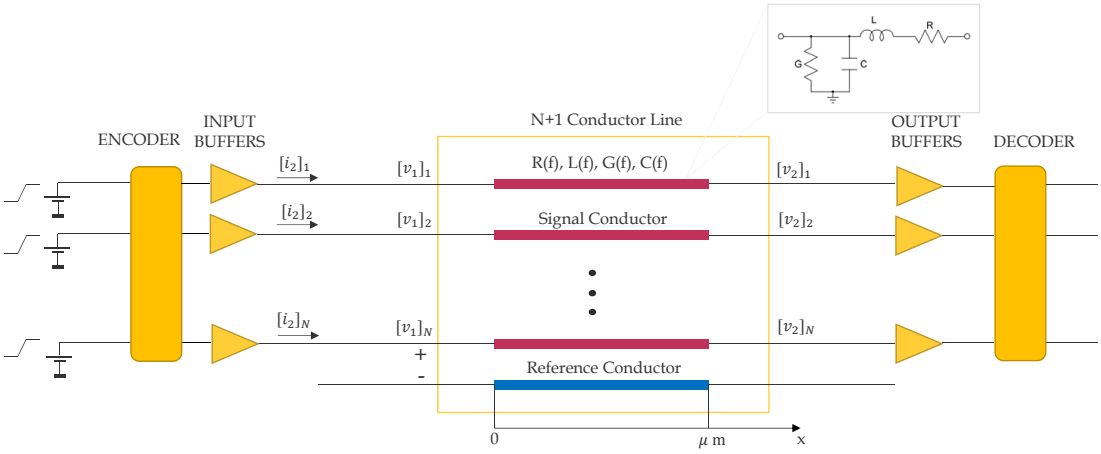


Figure 7.6: SPICE Simulation Setup for the Interconnect Coding-Based Systems.

The SPICE simulations were performed by using a 45nm commercial technology, under nominal operating conditions. As concerns the interconnect, for given specifications (e.g., wire length, number of parallel conductors), and technology parameters (e.g., related to the dielectric and metal layer stack conductivity, dielectric permittivity, wire pitch, etc.), a SPICE RLGC compatible model was obtained using the electromagnetic field solver from Synopsys, Raphael [102].
As data to be transmitted over the wires, in order to assess the systems energy savings potential sensitivity to the transmission data activity profile, the following two situations are considered:

- 10000 randomly generated bytes, are provided as system input, one byte per clock cycle, and
- 10000 bytes, of correlated data (sampled from multi-variate copulas with different linear correlation coefficients) are provided as system input, one byte per clock cycle as in the previous case.

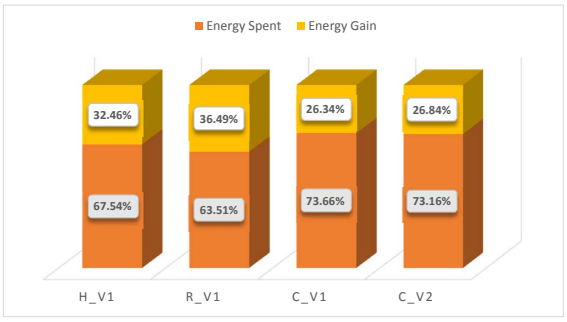
As qualitative and quantitative indicators of the interconnect coding schemes performance, we employ: (i) the energy consumed by the entire system for data transmission, and the associated hardware complexity, (ii) the system maximum achievable frequency, and data arrival profile for each wire, with respect to each other, and (iii) the resiliency of each system to crosstalk, or other bit flip inducing situations that can manifest during the transmission of data over wires. Subsequently, all considered systems are analyzed with respect to each of the three interconnect performance monitors.

7.3.1 Energy and Area.

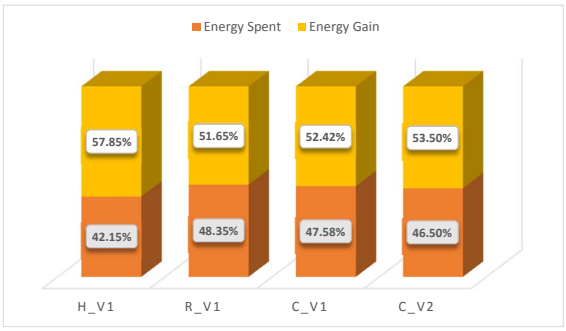
The energy consumed by each system is measured for the entire system (i.e., encoder/input buffers + wires + output buffers/decoder) and over the entire duration of transmission (i.e., $10000 \times T_{clk}$). To enable a fair comparison between the various analyzed systems, T_{clk} is tailored for each analyzed system (as a function of the wire length, and of the encoder/decoder maximum operation frequency), such that the data at each system output can be correctly sampled. The energy is measured in SPICE using the supply current integrated over the entire duration of transmission, as follows:

$$\text{Energy} = \int_0^{10000 \times T_{clk}} I_{VDD}(t) \cdot V_{VDD}\, dt, \tag{7.6}$$

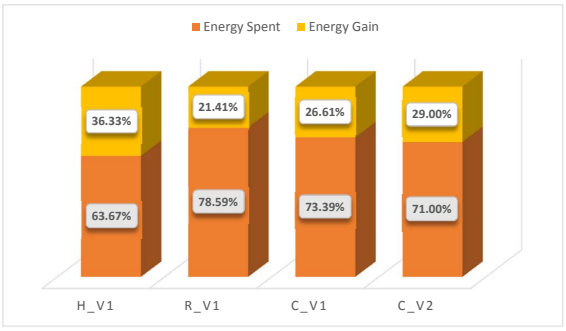
thus reflecting both the static and the dynamic energy components.
Various interconnect lengths are investigated: L = 1 mm, L = 5 mm, and L = 10 mm, in order to assess the energy reduction/bit-flip errors resiliency potential of proposed coding schemes for both short and long range interconnects and randomly generated data workloads.
Figure 7.7 graphically illustrates the energy breakdown corresponding to each system, for the considered interconnect lengths. The energy percentages are reported relative to the energy figures obtained



(a) L = 1 mm.



(b) L = 5 mm.



(c) L = 10 mm.

Figure 7.7: Energy Profile vs. Interconnect Length L.

for the reference 8-wire system, for which the raw, uncoded data are transmitted. In Figure 7.7(a), it can be noted that for L = 1 mm, R_v1 exhibits energy savings 4% bigger when compared to the 12-wire Haar system, and 10% bigger with report to the two 10-wire C_v1 and C_v2 systems. However, for the L = 5 mm and L = 10 mm cases from Figure 7.7(b) and Figure 7.7(c), the R_v1 system energy gain is surpassed by all other systems by at most 15%. The two 10-wire C_v1 and C_v2 systems and the 12-wire Haar system have similar energy gains for the wire lengths L = 1 mm and L = 5 mm (less than 6%), while for L = 10 mm, the energy gain for the 12-wire Haar surpasses the energy gain of the two 10-wire systems C_v1 and C_v2 by 10% and 7%, respectively.

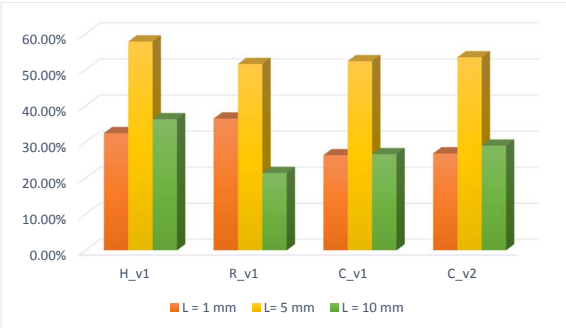


Figure 7.8: Energy Gain vs. Interconnect Length L.

Figure 7.8 depicts the energy gain percentage (reported as previously with respect to the 8-wire consumed energy values) variation with the interconnect length. Specifically, it can be observed that as the interconnect length increases, the energy gain also increases, which is as expected, since longer interconnects are more energy demanding than shorter ones, and thus benefit more from a switching activity reduction on its wires. Such is the case of the energy gain increase for all systems from

L = 1 mm to L = 5 mm: 25% for the Haar system, 15% for R_v1 system, 26% for the C_v1, and 27% for C_v2. The biggest percentage increase is manifested by the C_v1, C_v2, and Haar systems, which makes them more energy efficient for medium range interconnects. Nevertheless, they present energy reduction potential also for shorter interconnects (L = 1 mm) with energy gains ≈ 30%. Switching from a length L = 5 mm to a length L = 10 mm interconnect, less energy increase is reported for all systems: (30% less gain for R_v1 system, 26% and 25% less for C_v1 and C_v2 systems, and 22% for Haar system). This can be attributed to the fact that the bus and driving buffers consumed energy is increasing for longer interconnects and begins to counterbalance the energy benefits enabled by the coding scheme.

To assess whether the measured energy gains are consistent across input data with varying degree of patterns dependency (as often is the case for data/address buses), a new set of linearly correlated dataset is generated for 0.8 correlation coefficient and fed byte-wise to each system input. Measuring the energy for each system, reveals as illustrated in Figure 7.9 that the R_v1 scheme consumes 5 times more energy when compared to the energy of the 8-wire reference system. This can be attributed to the fact that the R_v1 system data repetition is hardwired, and thus its performance energy-wise is restricted to certain data profiles in the absence of a dynamic reshuffling scheme. Compared to the random input data case in Figure 7.7(b), an energy gain that is less with 12% and 25% is exhibited by the 12-wire Haar system and the 10-wire C_v1 system, respectively. For the C_v2 system, the energy consumed is bigger that the reference system energy by 2%, and thus in this case the energy savings enabled to a lesser extent by the coding scheme are surpassed by the codec energy consumption, resulting in no energy gain for the entire system.
Based on our simulations (for linearly correlated data with varying correlation coefficients), it was observed in all cases an increased resiliency of the Haar scheme and of the constraint coding scheme C_v1 to the input data profile. The situation may change in other use cases and to conjuncture a system energy gain robustness to the input data profile, extensive simulations have to be performed, for linearly and non linearly correlated input data. However, this is a matter of the in-field applicability (data bus or address bus; the encompassing circuit usage scenarios, etc.), and thus it is not in the scope of this deliverable.

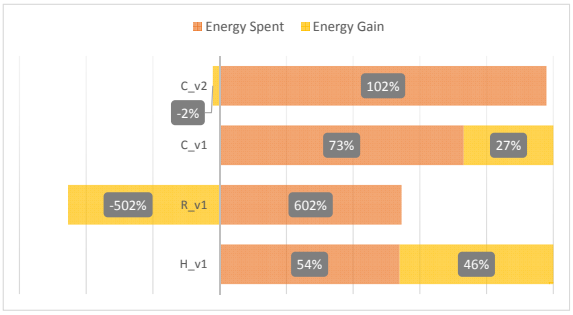


Figure 7.9: Energy Breakdown for Correlated Input Data for L = 5 mm.

Figure 7.10 depicts the compound (encoder + decoder + bus) area figures for each system under consideration. One can observe that the 12-wire system R_v1 exhibits the lowest area footprint. However, an effective energy-area trade-off assessment is hindered for this system, as without an energy optimized data reordering algorithm (and the afferent hardware), its energy saving depend to a big extent on a certain optimal data activity profile, contrary to the the 12-wire Haar, and 10-wire C_v1 and C_v2 systems which are less data sensitive. As concerns the 12-wire Haar system, a decreased hardware complexity is obtained (1.89× the reference system area), when compared to the two 10-wire C_v1 and C_v2 systems (2.17×, and 2.23× the reference area system, respectively). This is not unexpected, as the Haar encoder and decoder consist of a single level and two levels of logic, respectively, as opposed to the C_v1 and C_v2 LUT-based systems which contain multiple levels of logic (e.g., 7

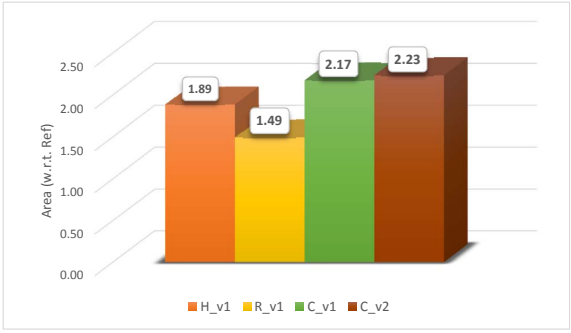


Figure 7.10: Area w.r.t. the 8-Wire Reference System.

levels of logic for the C_v1 decoder).

7.3.2 Data Arrival Profile

Another performance metric of interest is the data arrival profile at the bus receiving end, as this has direct implications on the system maximum frequency.
In order to inspect and compare the systems propagation delay distributions, a box-and-whiskers plot is employed, following the usual conventions as illustrated in Figure 7.11, repeated here for convenience:

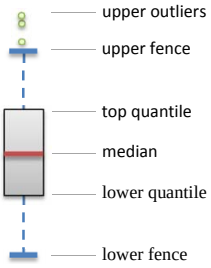


Figure 7.11: Box and Whiskers Plot Semantics.

- The upper quantile marks the value above which 25% of the data points lay.
- In a similar manner, the lower quantile marks the data value below which lie 25% of the data points.
- The median value is the middle of the dataset (not necessarily the mean), which means that 50% of the data are greater than this value.
- The upper fence corresponds to the greatest value, excluding the outlier data points.
- The lower fence corresponds to the lowest value, excluding the outlier data points.
- The upper outlier points are those data points with a value greater than $\frac{3}{2}\times$ the upper quantile value.
- Similarly, the lower outlier points are those data points with a value lower than $\frac{3}{2}\times$ the lower quantile value.

Figure 7.12 to Figure 7.16 depict the data arrival profile for all systems for the three interconnect lengths, i.e., 1 mm, 5 mm, and 10 mm. It can be noted that as the wire length increases, so does the arrival time spread and the maximum arrival time. It can be observed that for the reference

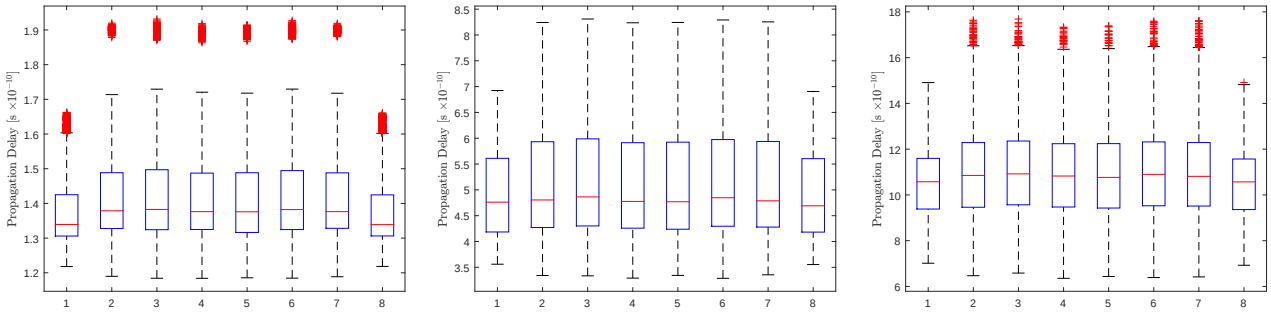
8-wire system, the bit arrival time for each wire exhibit a smaller spread when compared to all other coding based systems. However, the maximum arrival time is lower when compared to the maximum arrival time for all other systems, except for the L = 5 mm and L = 10 mm cases. This has positive implications on the transmission clock frequency for all coding systems, which can be increased as follows with:

- 40% (L = 5 mm) and 47% (L = 10 mm) for the 12-wire H_v1 system
- 37% (L = 5 mm) and 40% (L = 10 mm) for the 12-wire R_v1 system,
- 24% (L = 5 mm) and 33% (L = 10 mm) for the 10-wire C_v1 system, and
- 11% (L = 5 mm) and 29% (L = 10 mm) for the 10-wire C_v2 system.

For L = 1 mm, only the R_v1 enables a frequency increase with 20%, while for the Haar and the two constraint-coding systems, the frequency is negatively impacted, as it is decreased by:

- 1.4× for the H_v1 system,
- 2× for the C_v1 system,
- 2.6× for the C_v2 system.

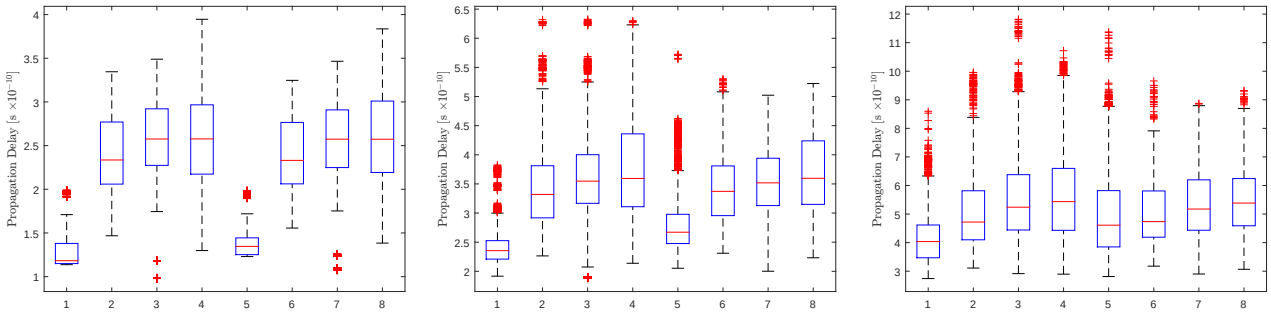
The frequency decrease for L = 1 mm can be attributed to the effects of bus switching activity diminution which are more prominent for medium and longer wires than for shorter wires (thus with greater impact on the bus delay for longer wires rather than for shorter wires). The total delay (the encoder+decoder+bus) of a coding-based system has to counterbalance the 8-wire bus delay, in order to obtain frequency benefits. However for shorter wires this may not be the case even if the coding system bus delay is reduced as result of a smaller transition count.
We note that the R_v1 system doesn't experience any frequency reduction penalty as it doesn't includes any codec; the crosstalk effects are diminished just by wire replication (area overhead in the interconnect only). However R_v1 effectiveness is limited to some input data profiles and to make it more general purpose (as the constrained and Haar counterparts) a programmable (partial) crossbar has to be implemented at each interconnect side, which will result in similar or even larger operation frequency reductions, as a crossbar implementation takes more that one gate delay.



(a) L = 1 mm. (b) L = 5 mm. (c) L = 10 mm.

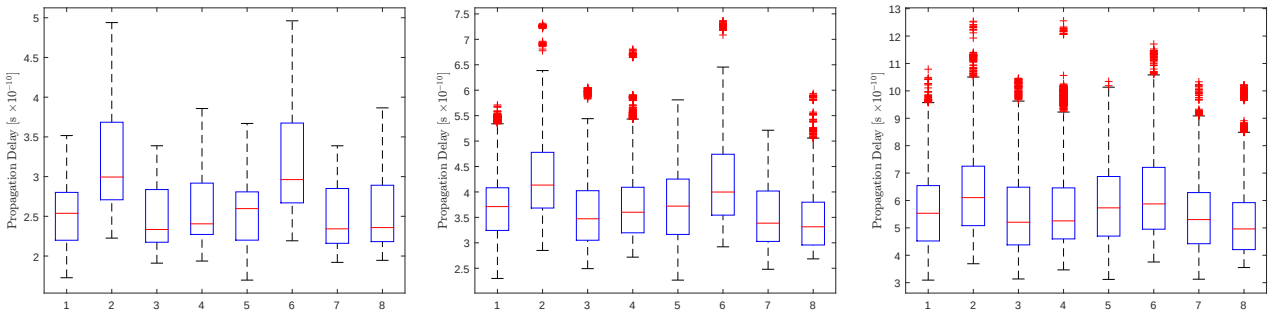
Figure 7.12: Data Arrival Profile for the Reference 8-Wire System.

Figure 7.17 depicts the minimum clock period for each system with regard to the interconnect length. It can be noted that as the interconnect length increases, so does the minimum clock period. A trend observed for each length is that the two constraint coding systems are less effective frequency-wise when compared to the 12-wire Haar and R_v1 systems.



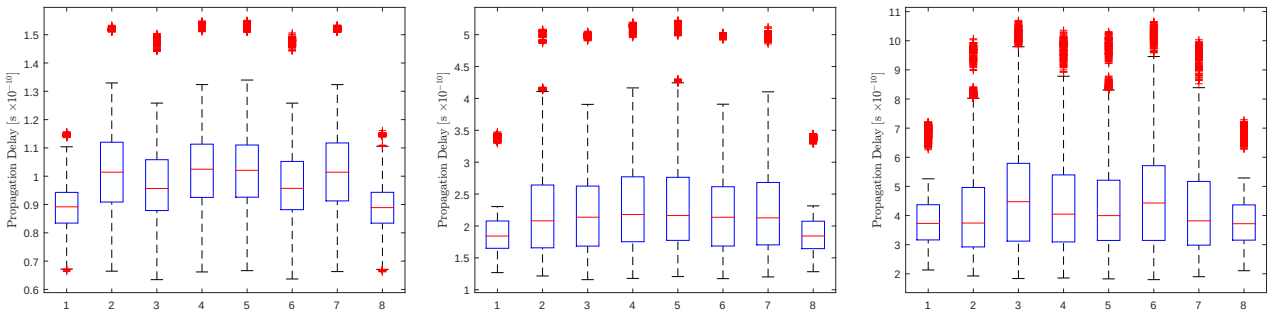
(a) L = 1 mm. (b) L = 5 mm. (c) L = 10 mm.

Figure 7.13: Data Arrival Profile for the 10-Wire C_v1 System.



(a) L = 1 mm. (b) L = 5 mm. (c) L = 10 mm.

Figure 7.14: Data Arrival Profile for the 10-Wire C_v2 System.



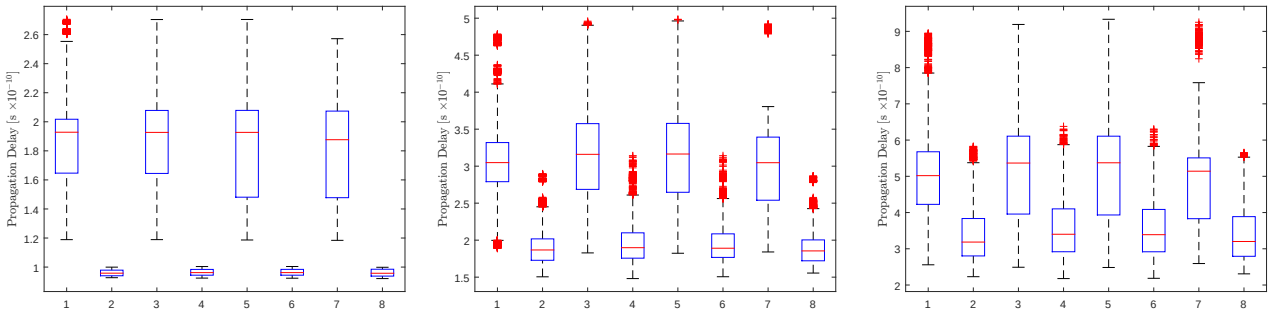
(a) L = 1 mm. (b) L = 5 mm. (c) L = 10 mm.

Figure 7.15: Data Arrival Profile for the 12-Wire R_v1 System.

7.3.3 Deep-Submicron Noise Induced Bit-Flip Resiliency

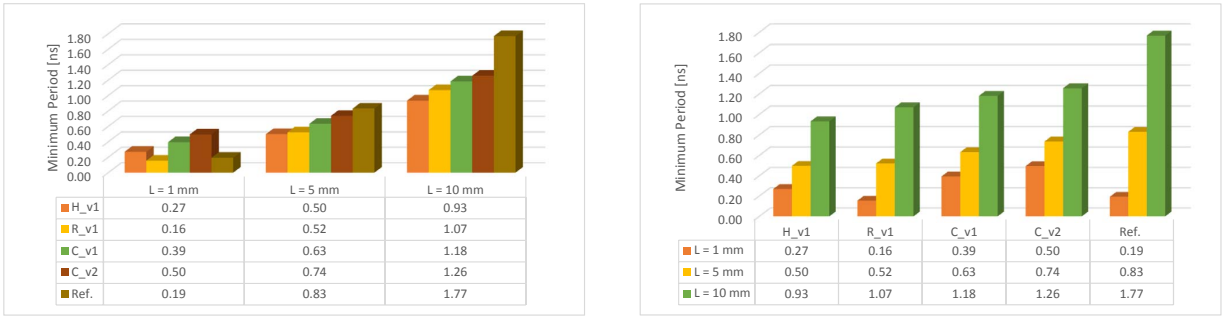
Even though the crosstalk between adjacent wires is reduced by some of the presented coding schemes, the deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) increases the interconnects susceptibility to errors. To combat the errors that can arise during the data transport, Error Correcting Codes (ECC) can be employed. Subsequently, a Single Error Correction, Double Error Detection (SECDED) [103] scheme adapted to the peculiarities of the Haar system is presented. We note that while potentially speaking the other systems can be also augmented with ECC features we decided to only focus on Haar as it is more general purpose and provides larger energy savings for a wider input data profiles.

In line with the previously employed notation, let (x_0, x_1) denote the first two bits of an input byte of raw, uncoded data. After the Haar encoding is applied to the input pair of bits (x_0, x_1), a 3-bit



(a) L = 1 mm. (b) L = 5 mm. (c) L = 10 mm.

Figure 7.16: Data Arrival Profile for the 12-Wire Haar System.



(a) (b)

Figure 7.17: Minimum Clock Period vs Interconnect Length.

encoded sequence $\{S^{(0)}, C1^{(0)}, C0^{(0)}\}$ to be transmitted over the wires is obtained, with the below expressions repeated here for convenience:

$$\begin{aligned} S^{(0)} &= x_0 \wedge x_1 \\ C1^{(0)} &= x_1 \\ C0^{(0)} &= x_0 \vee x_1. \end{aligned}$$

At the interconnect receiving end, the decoding is governed by the equations:

$$\begin{aligned} \hat{x}_0 &= S^{(0)} \oplus C1^{(0)} \oplus C0^{(0)} \\ \hat{x}_1 &= S^{(0)} \oplus C1^{(0)}, \end{aligned}$$

If the transmission is error-free, then the estimated data bits are equal to the original data bits (i.e., $\hat{x}_0 = x_0$ and $\hat{x}_1 = x_1$). However, as previously argued, even if the encoded sequence $\{S^{(0)}, C1^{(0)}, C0^{(0)}\}$ exhibits good crosstalk resilient characteristics (which lowers the probability of crosstalk induced bit-flips during transmission), the transmission over the interconnects may be subjected to certain error inducing conditions, resulting in one or multiple bit-flips in the transmitted encoded sequence. Thus, an error detection and correction mechanism is highly desirable.

Let m_0 denote the encoded sequence of transmitted bits $m_0 = (S^{(0)}, C1^{(0)}, C0^{(0)})$, and ϵ the error affecting the bits transmission. Table 7.1 summarizes all possible 1-bit error scenarios affecting m_0. The first two columns in the table represent the original data bits x_0 and x_1, the third and the fourth column denote the Haar encoded message at the interconnect transmitting end, and the message at the receiving end (Haar encoded message + noise), respectively; while the last two columns in the table correspond to the estimated data bits after Haar decoding. As for single bit errors, all possible

values that ϵ can take are: (0 1 0), (1 0 0), and (0 0 1), 3 situations should be analyzed at the interconnect receiving end for each possible x_0 and x_1 combination of bits. One may note in Table

x_0	x_1	TX end (m_0)	RX end ($m_0 \oplus \epsilon$)	$\hat{x}_0$	$\hat{x}_1$
0	0	(0 0 0)	(1 0 0)	1 (F)	1 (F)
			(0 1 0)		1 (F)
			(0 0 1)		0
0	1	(0 1 1)	(1 1 1)	1 (F)	0 (F)
			(0 0 1)		0 (F)
			(0 1 0)		1
1	0	(0 0 1)	(1 0 1)	0 (F)	1 (F)
			(0 1 1)		1 (F)
			(0 0 0)		0
1	1	(1 0 0)	(0 0 0)	0 (F)	0 (F)
			(1 1 0)		0 (F)
			(1 0 1)		1

Table 7.1: One-Bit Error Scenarios for Haar System.

7.1, that in all one-error scenarios, the decoded value $\hat{x}_0$ is always erroneous. This is expected, as all three encoded bits $(S^{(0)}, C1^{(0)}, C0^{(0)})$ are involved in the computation of the decoded bit $\hat{x}_0$. It follows that any single error affecting the sequence of encoded bits $(S^{(0)}, C1^{(0)}, C0^{(0)})$, will always result in an erroneous value of $\hat{x}_0$. Thus it is mandatory to protect the x_0 value in order to be able to correct an erroneously decoded value $\hat{x}_0$. On the other hand, when $\hat{x}_0$ is erroneous, we also need to be able to discriminate the correct value of $\hat{x}_1$, in which case $C0^{(0)}$ has to be protected.
Generalizing from the two input bits $\{x_0, x_1\}$ to the entire input byte $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, we propose to append the following 5 error control coding bits to the 12-bit Haar encoded sequence (i.e., to the 12-bit sequence $\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}\right\}$):

$$E_1 = x_0 \oplus x_2 \oplus x_6 \tag{7.7}$$
$$E_2 = x_0 \oplus x_2 \oplus x_4 \tag{7.8}$$
$$E_3 = x_2 \oplus x_4 \oplus x_6 \tag{7.9}$$
$$E_4 = x_0 \oplus x_4 \oplus x_6 \tag{7.10}$$
$$E_5 = C0^{(0)} \oplus C0^{(1)} \oplus C0^{(2)} \oplus C0^{(3)} \tag{7.11}$$

The first 4 bits $\{E_1, E_2, E_3, E_4\}$ correspond to a $(7, 4)$ Hamming code [104], while bit E_5 is simply a parity bit. Summarizing, bits E_1 to E_4 are used for error detection and correction of the input bits $\{x_0, x_2, x_4, x_6\}$, while bit E_5 is used for error detection and correction of the input bits $\{x_1, x_3, x_5, x_7\}$.

7.3.3.1 Single Error Detection and Correction

If there is one bit-flip error affecting any of the 17-bit sequence

$$\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}, E_1, E_2, E_3, E_4, E_5\right\},$$

Case 1	Case 2	Error Discrimination
$\hat{x}_0 = S^{(0)} \oplus C1^{(0)} \oplus C0^{(0)}$	$\hat{x}_0 = E_1 \oplus E_2 \oplus E_4$	$w_1 = \hat{x}_{0(case1)} \oplus \hat{x}_{0(case2)}$
$\hat{x}_0 = S^{(1)} \oplus C1^{(1)} \oplus C0^{(1)}$	$\hat{x}_0 = E_1 \oplus E_2 \oplus E_3$	$w_2 = \hat{x}_{2(case1)} \oplus \hat{x}_{2(case2)}$
$\hat{x}_0 = S^{(2)} \oplus C1^{(2)} \oplus C0^{(2)}$	$\hat{x}_0 = E_2 \oplus E_3 \oplus E_4$	$w_3 = \hat{x}_{4(case1)} \oplus \hat{x}_{4(case2)}$
$\hat{x}_0 = S^{(3)} \oplus C1^{(3)} \oplus C0^{(3)}$	$\hat{x}_0 = E_1 \oplus E_3 \oplus E_4$	$w_4 = \hat{x}_{6(case1)} \oplus \hat{x}_{6(case2)}$

Table 7.2: Single Error Correction for the Haar System.

the correct value of the input byte $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ can always be obtained.
Any error affecting one of the bits

$$\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}\right\}$$

can be corrected as follows:
We compute in parallel each of the bits $\{\hat{x}_0, \hat{x}_2, \hat{x}_4, \hat{x}_6\}$ in two manners, as summarized in the first two columns of Table 7.2. The bits $\{w_1, w_2, w_3, w_4\}$ in the third column in Table 7.2 are used for discriminating the correct set of values between the case 1 and case 2 estimates.

- In the error free scenario, $\hat{x}_{0\ (\text{case } 1)}$ coincides with the value of $\hat{x}_{0\ (\text{case } 2)}$ and thus $\{w_1, w_2, w_3, w_4\} = \{0, 0, 0, 0\}$.
- In one error occurs in the sequence

  $$\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}\right\},$$

  then:
  - one value of $\hat{x}_{(\text{case } 1)}$ is computed wrong;
  - one of the bits $\{w_1, w_2, w_3, w_4\}$ is equal to "1".

  In this situation, the case 1 decoded bits $\{\hat{x}_0, \hat{x}_2, \hat{x}_4, \hat{x}_6\}$ are the correct ones.
- If one error occurs in the sequence $\{E_1, E_2, E_3, E_4\}$, then:
  - three values of $\hat{x}_{(\text{case } 2)}$) are computed wrong;
  - three of the bits $\{w_1, w_2, w_3, w_4\}$ are equal to "1".

  In this situation, the case 2 decode bits $\hat{x}_0, \hat{x}_2, \hat{x}_4, \hat{x}_6$ are the correct ones.
- Thus, to summarize the discrimination bits are used as follows:
  - If $w_1 + w_2 + w_3 + w_4 = 3$ then choose case 2 $\{\hat{x}_0, \hat{x}_2, \hat{x}_4, \hat{x}_6\}$ decoded bits;
  - Otherwise, choose case 1 $\{\hat{x}_0, \hat{x}_2, \hat{x}_4, \hat{x}_6\}$ decoded bits.
- Suppose one error occurred and the correct value of $\hat{x}_0$ was obtained using the above methodology. However, there are two Haar encoded 3-bit sequences for the correct value of $\hat{x}_0 = 0$, per se. Specifically, $(S^{(0)}, C1^{(0)}, C0^{(0)})$ is then either (0, 0, 0) or (0, 1, 1), which means we do not know exactly whether the value of $\hat{x}_1$ is "0" or "1".
  However, since bit E_5 is correct and equal to $C0^0 \oplus C0^{(1)} \oplus C0^{(2)} \oplus C0^{(3)}$, it will allow the determination of the correct value of $\hat{x}_1$. In this case $\hat{x}_1 = E_5 \oplus C0^{(1)} \oplus C0^{(2)} \oplus C0^{(3)}$.
- If one error occurs in the bit E_5, it is of no relevance, as all the 12 S and C bits required to restore the correct input bits $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ are correct.

7.3.3.2 Double Error Detection

A first possible case is when all bit-flip errors affect any two bits from the sequence

$$\left\{S^{(0)}, C1^{(0)}, C0^{(0)}, S^{(1)}, C1^{(1)}, C0^{(1)}, S^{(2)}, C1^{(2}, C0^{(2)}, S^{(3)}, C1^{(3)}, C0^{(3)}, E_1, E_2, E_3, E_4\right\},$$

result in a value of $w_1 + w_2 + w_3 + w_4$ which is either equal to 2, or equal to 4.
A second case is when there is one bit-flip error in the previous 16-bits sequence, while the other error affects bit E_5. In such a case, the error in the sequence can be detected with the previous single detection flow (otherwise stated, $w_1 + w_2 + w_3 + w_4$ has to be either equal to 1 to equal to 3), while bit E_5 can be duplicated and transmitted twice over the wires, at the extra cost of an additional redundant wire. If area is a foremost design optimization goal, a plausible alternative is per se when the 18 bits are sent over a 9-wire bus as 9 bits on the rising edge of the clock signal, and the other 9 bits on the falling edge of the clock signal.

7.3.3.3 Energy-Area-Delay Evaluation

Assuming one is interested in single error detection and correction, one avenue as previously described is to use Hamming codes for protecting the even data bits $\{x_0, x_2, x_4, x_6\}$, and a parity bit for protecting the odd data bits $\{x_1, x_3, x_5, x_7\}$. Thus, in this situation for every data byte, a 17-bit codeword has to be transmitted over the wires. There are multiple configurations to transmit the 17-bit codeword with trade-offs between the number of wires used for transmission and the transmission time. One such configuration, depicted in Figure 7.18, is to use a 9-wire bus and transmit the 17 bits both on positive and on negative edge of the clock. In this way no additional wires are employed, while keeping the

Wire number								
w1	**w2**	**w3**	**w4**	**w5**	**w6**	**w7**	**w8**	**w9**
$S^{(0)}$	$C1^{(0)}$	$C0^{(0)}$	$S^{(1)}$	$C1^{(1)}$	$C0^{(1)}$	E_1	E_2	E_5
$S^{(2)}$	$C1^{(2)}$	$C0^{(2)}$	$S^{(3)}$	$C1^{(3)}$	$C0^{(3)}$	E_3	E_4	E_5

Figure 7.18: 9-Wire Bus Configuration for Haar-Based System with ECC.

total transmission duration the same (i.e., $10000 \times T_{clk}$). This configuration has also the advantage that bit E_5 can be duplicated at no additional cost, enabling thus the double error detection.
An alternative is to use a 17-wire bus, as illustrated in Figure 7.19, at the expense of some area overhead. Furthermore, in this situation, if double error detection capability is desired, an extra wire has to be employed to accommodate the duplication of bit E_5.
While to allow for an appropriate comparison of energy merits between the different coding-based

Wire number															
w1	**w2**	**w3**	**w4**	**w5**	**w6**	**w7**	**w8**	**w9**	**w10**	**w11**	**w12**	**w13**	**w14**	**w15**	**w16**	**w17**
$S^{(0)}$	$C1^{(0)}$	$C0^{(0)}$	$S^{(1)}$	$C1^{(1)}$	$C0^{(1)}$	$S^{(2)}$	$C1^{(2)}$	$C0^{(2)}$	$S^{(3)}$	$C1^{(3)}$	$C0^{(3)}$	E_1	E_2	E_3	E_4	E_5

Figure 7.19: 17-Wire Bus Configuration for Haar-Based System with ECC.

systems the 12-wire configuration was used, for evaluating the Haar-based coding system endowed with ECC, the 9-wire configuration serves better the purpose. In view of this the following systems are evaluated:

- 8-wire reference uncoded system, denoted as "Ref".
- 12-wire reference uncoded system with ECC protection, denoted as "Ref + ECC".
  The first 8 wires are identical to the reference uncoded system, while the last 4 wires correspond to 4 Hamming bits (as to protect 8 data bits, a $(15, 11)$ Hamming code can be employed).

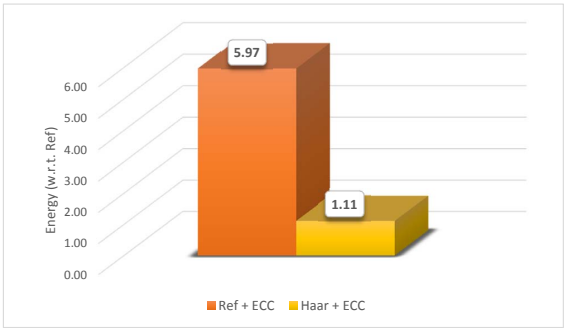
Wire number							
w1	w2	w3	w4	w5	w6	w7	w8
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7

Wire number											
w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	E_1	E_2	E_3	E_4

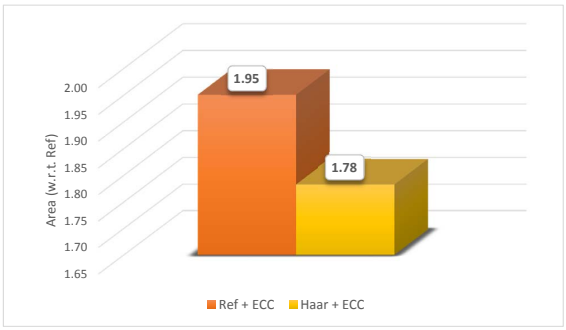
- 9-wire Haar system with ECC capacity, denoted as denoted as "Haar + ECC".
  This system takes advantage of both edges of the clock signal in order to transmit the data, as previously discussed.

Wire number								
w1	w2	w3	w4	w5	w6	w7	w8	w9
$S^{(0)}$	$C1^{(0)}$	$C0^{(0)}$	$S^{(1)}$	$C1^{(1)}$	$C0^{(1)}$	E_1	E_2	E_5
$S^{(2)}$	$C1^{(2)}$	$C0^{(2)}$	$S^{(3)}$	$C1^{(3)}$	$C0^{(3)}$	E_3	E_4	E_5

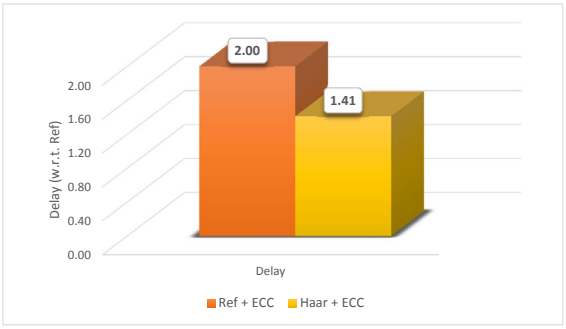
Figure 7.20 depicts the energy and area figures for the ECC enabled systems "Haar + ECC" and "Ref + ECC". Both energy and area measurements are reported to the figures obtained for the 8-wire uncoded and unprotected system "Ref". One can observe that energy wise the Haar system with ECC



(a) Energy.



(b) Area.



(c) Delay for L = 5 mm.

Figure 7.20: Energy-Area-Delay Figures for the ECC Enabled Systems.

protection consumes 1.11× more energy than the reference 8-wire system (thus almost equal to its energy), as opposed to the reference system protected with ECC which spends 5.97× more energy than the "Ref" system. Area wise, the "Haar + ECC" system has an overhead of 1.78× the "Ref" system area, which is similar to the "Ref + ECC" system area overhead of 1.95× the "Ref" system

area. As concerns the minimum clock period, simulation results for L = 5 mm reveal an increase of 2× the Ref period for the "Ref + ECC", and a 1.41× the Ref period increase for the "Haar + ECC" system.

7.4 Conclusion

In this chapter we introduced energy effective reliable data transport structures able to deal with technology scaling related phenomena, e.g., crosstalk, supply voltage variations, electromagnetic interference. Our approach relies on the augmentation of traditional interconnects, i.e., bus segments, NoC links, with a codec infrastructure and to this end we considered 3 types of data encoding techniques, namely, Constrained, Repetition, and Haar. We applied each of them on an 8-bit wide interconnect segment, and evaluate their practical implications when using a 45nm commercial CMOS technology.

Our simulations indicate that substantial energy savings can be achieved by properly tuning the codec to the interconnect length and workload data profile. In particular, when compared with the reference uncoded interconnect, Constrained, Repetition, and Haar provides energy savings of about 27%, 37%, and 33%, respectively, for an interconnect length L = 1 mm, 53%, 52%, and 58%, respectively, for an interconnect length L = 5 mm, and 27%, 22%, and 37%, respectively, for an interconnect length L = 10 mm. For correlated data Constrained and Haar provide energy savings of about 27%, and 46%, respectively, for an interconnect length L = 5 mm, while the Repetition results in an 500% energy increase, which is not surprising given its hardwired nature but which may be very much different for other data correlation profiles.

Given that the considered data encoding schemes diminish the crosstalk occurrence the codec augmented interconnects longer than L = 1 mm can operate at a higher frequency than the uncoded ones. In particular, Constrained, Repetition, and Haar enable a clock frequency increase of about 24%, 37%, and 40%, respectively, for an interconnect length L = 5 mm, and 33%, 40%, and 47%, respectively, for an interconnect length L = 10 mm.

The energy and data transmission delay reductions are obtained at an area increase of 2.2×, 1.5×, and 1.9×, with respect to the reference design, for the Constrained, Repetition, and Haar based interconnects, respectively.

Given that these results suggest that the Haar codec based approach is the most general purpose one as it outperforms the other ones in most of the considered cases we augmented it with error correction capabilities. We introduced a Single Error Correction and Double Error Detection scheme adapted to the peculiarities of the Haar system, which makes the Haar augmented interconnect not only energy effective but also robust against deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) induced transmission errors. The Error Correcting Code (ECC) enhanced Haar system consumes about the same energy as the reference 8-wire system while the direct ECC augmentation of the reference system would result in a 5.97× energy increase. Moreover, for the same error correction/detection capability, the ECC enhanced Haar requires less area and operates at a higher frequency than the ECC augmented reference counterpart.

Chapter 8

General Conclusion of WP4

Here we briefly discus main contributions of WP 4. During the M1-M33 period we have made considerable progress beyond the state of the art on the analysis of hard decision decoders, under uncorrelated and data-dependent gate failures. Our research was mostly aimed to proposing new analytical tools that can be used for performance evaluation of faulty hard-decision decoders. We have also design the probabilistic bit-flipping decoders, resistent to hardware unreliability. The main contributions for WP 4 can be summarized as follows:

- We propose a new class of bit flipping algorithms operating at perfect hardware was proposed. It was shown that introducing the collective error correction principle two-bit bit flipping algorithms can outperform min-sum and Gallager A/B message passing decoders.

- We proposed a novel class of fast convergence iterative decoders called decimation-enhanced finite alphabet iterative decoders. Furthermore, we showed that provable guaranteed error-correction can be achieved in a finite and small number of iterations, which represent a first result on guaranteed error-correction of a message-passing decoder other than Gallager-B decoder.

- We designed the fast convergent bit-flipping decoder resistent to hardware unreliability, which is based on multiple decoding attempts and random re-initializations. Our decoder outperforms all known bit-flipping decoders, and majority of of complex soft-decision message-passing decoder.

- We investigate the influence of uncorrelated gate failures to performance of Gallager B decoder. We noticed that performance of the decoder does not monotonically decreases as unreliability of logic gates increases. This actually means that under specific conditions the presence of hardware unreliability improves the overall decoder performance. We showed that the main reason for a such behaviour is the existence of small trapping sets in Tanner graph of a code. Random perturbations of gate outputs cause that the decoder converge to a codeword rather then to a fixed trapping set, which reduce the error rate.

- We developed the analytic tool for the finite code length analysis of the faulty decoders, subjected to independent identically distributed logic gate failures. This tool is based on Markov chains and enable us to follow the dynamics of iterative decoding and evaluate the performance of the decoder when moderate code length codes are employed. The above tool tracks the improvement cause by random perturbations in the decoder.

- We provided a new method for asymptotic analysis of memories based on one-step majority logic and Gallager-B decoders under uncorrelated gate failures. Our fixed-point analytical approach describes the memory reliability in more details than stat-of-the-art density evolution analysis. We use it to predict the reliability of memory architectures for fixed refresh time, code, and decoders parameters.

- We introduced a novel approach in gate failures modeling which enable us to capture data-dependence in more details, compared with state-of-the-art modeling. We developed the analytic

tool for the finite code length analysis of the one-step majority logic decoders, subjected to data-dependent logic gate failures. This tool enable us to evaluate the performance of memory architectures that are based on one-step majority logic decoding.

- We showed that guaranteed error correction concept can be extended to the faulty iterative decoders. We prove that error correction capability of the faulty bit-flipping decoder, increases linearly with code length, even when logic gates used in the decoder are prone to data-dependent failures. We were able to show that the above fact holds for codes with column weight $\gamma \geq 8$. In addition we proved the existence of low complexity reliable memories under correlated gate failures, which represents the first such result on the failure models other than simple independent failure model.

- We created practical premisses for the LDPC utilization for memory fault detection and correction. We proposed a fault resilient 3D polyhedral memory architecture which for the same redundancy requirements outperforms the state of the art Hamming code based counterpart in terms of error correction capability. Meanwhile, the memory performance and availability remain intact.

- We proposed codec enhanced data interconnects, e.g., bus segments, Network on Chip (NoC) links, in an attempt to combat technology scaling related phenomena, e.g., crosstalk and transmission delay variability, and provide error resilient energy effective data transport means in advanced CMOS Systems on Chip. To this end we introduced and evaluated data encoding techniques based on Constrained/Repetition coding and Haar transform. We demonstrated by means of SPICE simulations that significant energy savings and throughput increase can be achieved by properly tuning the codec to the interconnect length and workload data profile. We also introduced a Single Error Correction and Double Error Detection scheme, which made the codec assisted interconnect not only energy effective but also robust against deep sub-micron noise (e.g., supply voltage variations, electromagnetic interference) induced transmission errors.

Bibliography

[1] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*, 1st ed. Springer Publishing Company, Incorporated, 2011.

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

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

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

[5] 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.

[6] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of LDPC codes under unreliable message storage," *IEEE Communications Letters*, vol. 18, no. 5, pp. 849–852, May 2014.

[7] C. K. Ngassa, V. Savin, and D. Declercq, "Min-Sum-based decoders running on noisy hardware," in *IEEE Global Communications Conference*, Dec. 2013, pp. 1–10.

[8] S. K. Chilappagari, M. Ivkovic, and B. Vasić, "Analysis of one-step majority logic decoders constructed from faulty gates," in *Proceedings of IEEE International Symposium on Information Theory (ISIT '06)*, Seattle, WA, July 2006, pp. 469–473.

[9] S. K. Chilappagari and B. Vasić, "Fault tolerant memories based on expander graphs," in *Proc. IEEE Information Theory Workshop (ITW '07)*, Lake Tahoe, CA, Sept. 2007, pp. 126–131.

[10] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 09 1951.

[11] W. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," *Signal Processing*, vol. 6, no. 2, pp. 113 – 133, 1984.

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

[13] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. Gross, "Dithered belief propagation decoding," *IEEE Trans. on Commun.*, vol. 60, no. 8, pp. 2042–2047, August 2012.

[14] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *IEE Electronic Letters*, vol. 39, no. 3, pp. 299–301, February 6 2003.

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

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

[17] 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, 2009.

[18] Y. Mao and A. Banihashemi, "Decoding low-density parity-check codes with probabilistic scheduling," *IEEE. Commun. Letters*, vol. 5, no. 10, pp. 414–416, Oct 2001.

[19] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3385–3400, Oct. 2014.

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

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

[22] P. Ivanis, O. Al Rasheed, and B. Vasic, "MUDRI: A fault-tolerant decoding algorithm," in *in IEEE Int. Conf. on Commun. (ICC 2015)*, London, June. 2015, p. (paper acccepted).

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

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

[25] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA, USA: MIT Press, 1963.

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

[27] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Finite alphabet iterative decoders robust to faulty hardware: Analysis and selection," in *8th International Symposioum on Turbo Codes and Iterative Information Processing (ISTC)*, Bremen, Germany, Aug. 2014, pp. 1–10.

[28] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conference on Communications, Control and Computing*, Monticello, IL, USA, Sept. 2003, pp. 1426–1435.

[29] S. Chilappagari, D. Nguyen, and B. Vasić, "Trapping set ontology." [Online]. Available: http://www2.engr.arizona.edu/~vasiclab/Projects/CodingTheory/TrappingSetOntology.html

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

[31] B. Vasić, S. Chilappagari, D. Nguyen, and S. Planjery, "Trapping set ontology," in *Proc. 47th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, USA, Sep. 30–Oct. 2 2009, pp. 1–7.

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

[33] G. A. Margulis, "Explicit constructions of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.

[34] D. MacKay and M. Postol, "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," in *Proc. MFCSIT*, Galway, 2002.

[35] 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.

[36] K. Mansinghka, E. M. Jonas, and J. B. Tenenbaum, "Stochastic digital circuits for probabilistic inference," *MIT Computer Science and Artificial Intelligence Laboratory, Technical Report MIT-CSAIL-TR-2008-069*, 2008.

[37] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, Nov 2005.

[38] H. Jie and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," *IEEE Transactions on Nanotechnology*, vol. 1, no. 4, pp. 201–208, December 2002.

[39] A. Campbell, P. McDonald, and K. Ray, "Single event upset rates in space," *Nuclear Science, IEEE Transactions on*, vol. 39, no. 6, pp. 1828–1835, Dec 1992.

[40] O.-A. Rasheed, , P. Ivanis, and B. Vasic, "Fault-tolerant probabilistic gradient-descent bit flipping decoders," *IEEE Commun. Letters*, vol. 18, no. 9, pp. 1487 – 1490, September 2014.

[41] L. D. Davis and M. Mitchell, Eds. New York: Van Nostrand Reinhold, 1991.

[42] M. D. McDonnell, D. Abbott, and C. E. Pearce, "An analysis of noise enhanced information transmission in an array of comparators," *Microelectronics Journal*, vol. 33, no. 12, pp. 1079 – 1089, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269202001131

[43] M. Kawaguchi, H. Mino, and D. Durand, "Stochastic resonance can enhance information transmission in neural networks," *Biomedical Engineering, IEEE Transactions on*, vol. 58, no. 7, pp. 1950–1958, July 2011.

[44] F. Lustenberger and H.-A. Loeliger, "On mismatch errors in analog-vlsi error correcting decoders," in *The 2001 IEEE Inter. Symp. on Circuits and Systems, 2001. ISCAS 2001*, vol. 4, May 2001, pp. 198–201 vol. 4.

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

[46] A. Kuznetsov, "Information storage in a memory assembled from unreliable components," *Problems of Information Transmission*, vol. 9, pp. 254–264, 1973.

[47] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[48] B. Vasić, S. K. Chilappagari, S. Sankaranarayanan, and R. Radhakrishnan, "Failures of the Gallager B decoder: analysis and applications," in *Proc. 2nd Information Theory and Applications Workshop*. University of California at San Diego, Feb. 2006. [Online]. Available: http://ita.ucsd.edu/workshop/06/papers/160.pdf

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

[50] B. Vasic, D. V. Nguyen, and S. K. Chilappagari, *Failures and Error-Floors of Iterative Decoders (in press)*. New York: Academic Press Library in Mobile and Wireless, Communications, Elsevier, 2014.

[51] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. ISTA*, 2001.

[52] S. Chilappagari and B. Vasić, "Error-correction capability of column-weight-three LDPC codes," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2055–2061, May 2009.

[53] C. Petrie and J. Connelly, "A noise-based ic random number generator for applications in cryptography," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 5, pp. 615–621, May 2000.

[54] V. De, "Near-threshold voltage design in nanoscale cmos," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE, 2013, pp. 612–612.

[55] L. Anghel and M. Nicolaidis, "Defects tolerant logic gates for unreliable future nanotechnologies," in *Computational and Ambient Intelli- gence, ser. Lecture Notes in Computer Science, F. Sandoval, A. Prieto, J. Cabestany, and M. Graa, Eds. Berlin, Germany.* IEEE, 2007, p. 422429.

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

[57] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Construction of memory circuits using unreliable components based on low-density parity-check codes," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM 06)*, San Francisco, CA, USA, Nov. 2006, pp. 1–5.

[58] S. Chilappagari, M. Ivkovic, and B. Vasic, "Analysis of one step majority logic decoders constructed from faulty gates," in *Proceedings of IEEE International Symposium on Information Theory (ISIT 2006)*, Seattle, USA, July 2006, pp. 469–473.

[59] S. Chilappagari and B. Vasic, "Fault tolerant memories based on expander graphs," in *Proceedings of IEEE Information Theory Workshop*, Tahoe City, CA, USA, 2–7 Sep. 2007, pp. 126–131.

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

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

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

[63] E. Dupraz, D. Declercq, B. Vasic, and V. Savin, "Finite alphabet iterative decoders robust to faulty hardware: analysis and selection," in *8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC) (accepted)*, 2014, pp. 1–10.

[64] C. K. Ngassa, V. Savin, and D. Declercq, "Unconventional behavior of the noisy min-sum decoder over the binary symmetric channel," in *Information Theory and Applications Workshop*, Feb. 2014, pp. 1–10.

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

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

[67] M. Sipser and D. Spielman, "Expander codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[68] D. Burshtein, "On the error correction of regular LDPC codes using the flipping algorithm," *IEEE Transactions on Information Theory*, vol. 54, no. 2, pp. 517–530, Feb. 2008.

[69] D. Burshtein and G. Miller, "Expander graph arguments for messagepassing algorithms," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 782–790, Feb. 2001.

[70] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright, "LP decoding corrects a constant fraction of errors," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 82–89, Jan. 2007.

[71] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "On trapping sets and guaranteed error correction capability of LDPC codes and GLDPC codes," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1600–1611, Apr. 2010.

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

[73] S. Brkic, O. Al Rasheed, P. Ivanis, and B. Vasic, "On fault tolerance of the Gallager B decoder under data-dependent gate failures," *IEEE Communications Letters*, 2015, (early access).

[74] S. Brkic, P. Ivanis, and B. Vasic, "Majority logic decoding under data-dependent logic gate failures," submitted for publication, http://arxiv.org/abs/1507.07155.

[75] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, "Randomness conductors and constant-degree lossless expanders," in *STOC 02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, New York, NY, USA: ACM Press, 2002, pp. 659–668.

[76] E. Dupraz, Declercq, and B. Vasic, "Analysis of Taylor-Kuznetsov memory using one-step majority logic decoder," in *Proceedings of 10th Information Theory and Applications Workshop (ITA 2015)*, San Diego, CA, Feb. 2015, paper 273, [Online Available:] http://ita.ucsd.edu/workshop/15/files/paper/paper 3446.pdf.

[77] A. Khajeh, K. Amiri, M. Khairy, A. M. Eltawil, and F. Kurdahi, "A unified hardware and channel noise model for communication systems," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM 10)*, Miami, Florida, USA, 6–10 Dec. 2010, pp. 1–5.

[78] S. Nassif, "The light at the end of the cmos tunnel," in *Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on*, July 2010, pp. 4–9.

[79] S. Rusu, M. Sachdev, C. Svensson, and B. Nauta, "T3: Trends and challenges in vlsi technology scaling towards 100nm," in *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 16–. [Online]. Available: http://dl.acm.org/citation.cfm?id=832284.835426

[80] "The International Technology Roadmap for Semiconductors (ITRS), System Drivers," Tech. Rep., 2013. [Online]. Available: http://www.itrs.net

[81] P. Garrou, C. Bower, and P. Ramm, *3D Integration: Technology and Applications*. Wiley-VCH, 2008.

[82] B. Prince, *Vertical 3D Memory Technologies* . John Wiley and Sons, 2014.

[83] M. Lefter, G. Voicu, and S. Cotofana, "A shared polyhedral cache for 3d wide-i/o multi-core computing platforms," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, May 2015, pp. 425–428.

[84] R. Gallager, "Low-density parity-check codes," *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21–28, January 1962.

[85] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep., 2009.

[86] J. Shin, B. Petrick, M. Singh, and A. Leon, "Design and implementation of an embedded 512-kb level-2 cache subsystem," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 9, pp. 1815–1820, Sept 2005.

[87] F. Hamzaoglu, K. Zhang, Y. Wang, H. J. Ahn, U. Bhattacharya, Z. Chen, Y.-G. Ng, A. Pavlov, K. Smits, and M. Bohr, "A 3.8 ghz 153 mb sram design with dynamic stability enhancement and leakage reduction in 45 nm high-k metal gate cmos technology," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 1, pp. 148–154, Jan 2009.

[88] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*, 2nd ed. Wiley-IEEE Press, 2007.

[89] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

[90] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced srams," in *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, Dec 2003, pp. 21.4.1–21.4.4.

[91] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, "Sram immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect," *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 5, pp. 827–833, May 2004.

[92] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[93] The ieee 802.11n working group std. [Online]. Available: http://www.ieee802.org/11/

[94] The ieee 802.16 working group std. [Online]. Available: http://www.ieee802.org/16/

[95] The digital video broadcasting standard. [Online]. Available: https://www.dvb.org/standards/dvb-s2

[96] Ieee p802.3an (10gbase-t) task force. [Online]. Available: http://www.ieee802.org/3/an/

[97] H. Song, R. Todd, and J. Cruz, "Low density parity check codes for magnetic recording channels," *Magnetics, IEEE Transactions on*, vol. 36, no. 5, pp. 2183–2186, Sep 2000.

[98] W. Ping, L. Guangxia, Z. Hongpeng, and S. Xinying, "Generalized ldpc codes for deep space communication systems," in *Signal Processing (ICSP), 2012 IEEE 11th International Conference on*, vol. 2, Oct 2012, pp. 1279–1282.

[99] F. Kschischang, B. Frey, and H.Loeliger, "Factor graphs and the sum-product algorithm." in *IEEE Transactions on Information Theory.*, vol. 47, 2001, pp. 498–519.

[100] J. J. Benedetto, M. W. Frezier, *Wavelets: Mathematics and Applications.* CRC Press, 1994.

[101] K. A. S. Immink, *Coding Techniques for Digital Recorders.* Prentice Hall, 1991.

[102] "Deliverable 2.3 - Energy models of subpowered CMOS circuits," in *FP7ICT/FETOPEN/ iRISC project*, 2015.

[103] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications.* John Wiley and Sons, 2006.

[104] V. Pless, *Introduction to the Theory of Error-Correcting Codes.* John Wiley and Sons, 1998.