Error-Correction Techniques for Data Transmission and Storage

Verfahren zur Fehlerkorrektur bei der Datenübertragung und -speicherung

Vom Fachbereich Elektrotechnik und Informationstechnik
der Rheinland-Pfälzischen Technischen Universität Kaiserslautern-Landau
zur Verleihung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation

von

Kira Kraft

geboren in Frankfurt am Main

Tag der Einreichung:	11.02.2025
Tag der mündlichen Prüfung:	22.08.2025
Dekan des Fachbereichs:	Prof. Dr.-Ing. Daniel Görges
Vorsitzender der Prüfungskommission:	Prof. Dipl.-Ing. Dr. Gerhard Fohler
1. Berichterstatter:	Prof. Dr.-Ing. Norbert Wehn
2. Berichterstatter:	Prof. Dr. Stefan Ruzika

DE-386

Eidesstattliche Erklärung

Ich erkläre an Eides statt,

dass ich die vorliegende Dissertation selbst angefertigt habe und alle von mir benutzten Hilfsmittel angegeben habe,

dass ich die Dissertation oder Teile hiervon noch nicht als Prüfungsarbeit für eine staatliche oder wissenschaftliche Prüfung eingereicht habe, und

dass ich nicht die gleiche oder eine andere Abhandlung bei einem anderen Fachbereich oder einer anderen Universität als Dissertation eingereicht habe.

Kaiserslautern, 11. Februar 2025

Kira Kraft

Abstract

The ever increasing data traffic in our modern society poses severe challenges to any communication system. To ensure reliable data transmission and storage, error-correction coding is applied to make the data robust to disturbances. The challenge of error-correction coding lies in the decoder unit, where errors are detected and ideally corrected. In addition to providing a good error-correction performance, the increasing requirements posed by new standards are directly reflected to the decoder unit, stressing both a well-performing and highly efficient implementation. Requiring both a well-performing and highly efficient implementation equals a Pareto optimization problem, which cannot be solved optimally for all objectives at the same time. Therefore, application-specific trade-offs have to be made.

In this thesis, two applications to error-correction coding are investigated. The first application is maximum-likelihood decoding, a mathematical problem, which solves the decoding problem optimally. The goal of this use-case is therefore to improve the implementation efficiency. The second application is error correction for DRAMs. DRAMs are already highly optimized in their implementation efficiency, so the goal of this use-case is to improve the error-correction performance of the decoder unit.

Acknowledgments

This thesis is the result of a journey I did not take alone. I received continuous support from colleagues, friends and family on my way, whom I want to thank in the following.

First of all, I want to thank my doctoral advisor Prof. Dr.-Ing. Norbert Wehn for giving me this opportunity to do my PhD in this interesting field of channel coding and for pushing and supporting me throughout.

Second of all, I want to thank Prof. Dr. Stefan Ruzika for the cooperation in my favorite PhD project and for serving on my thesis committee.

I am very grateful for my colleagues I had the chance to work with in the Microelectronic Systems Design Research Group. First of all, I would like to thank Prof. Dr.-Ing. Matthias Jung. This journey wouldn't have started nor ended without you. Second of all, I would like to thank the Communications group: M.Eng. Oliver Griebel, Dipl.-Ing. Matthias Herrmann, M.Eng. Lucas Johannsen, and Dipl.-Inf. Claus Kestel. The numerous discussions with you always helped me solving my problems. Third, I would like to thank the DRAM group: Dr.-Ing. Deepak M. Mathew, M.Sc. Lukas Steiner, M.Sc. Chirag Sudarshan, and Dr.-Ing. Christian Weis. It was a pleasure working with you! A special thanks also goes to Martina Jahn for her continuous support and non-technical discussions and to Dipl.-Ing. (FH) Hans-Peter Goldhammer and Dipl.-Ing. (FH) Roland Volk for their technical support. Last but not least, I would like to thank my remaining colleagues. It's every single ones of your effort that turned this bunch of people into colleagues, into a team: M.Sc. Hussien Abdo, M.Sc. André Lucas Chinazzo, M.Sc. Johannes Feldmann, M.Sc. Muhammad Mohsin Ghaffar, Dr.-Ing. Bilal Hammoud, M.Sc. Jan Lappas, M.Sc. Frederik Lauer, M.Sc. Mohamed Moursi, M.Sc. Jonas Ney, M.Sc. Mohammadreza Esmaeilpour Quchani, M. Eng. Carl Rheinländer, B.Sc. Mohamed Amine Riahi, Dr.-Ing. Vladimir Rybalkin, M.Sc. Mohammad Hassani Sadi, M.Sc. Maximilian Schöffel, M.Sc. Menbere Tekleyohannes, Dr.-Ing. Javier Alejandro Varela, Dipl.-Math. Uwe Wasenmüller, Dr.-Ing. Stefan Weithoffer, Dipl.-Ing. Sebastian Wille, and Eng. Éder Ferreira Zulian. I would love to thank everyone of you specifically, but that would just not fit on this one page.

I also want to thank the colleagues from the Optimization Research Group of the RPTU, especially Dr. Florian Gensheimer and Dr. Tobias Dietz for the cooperation. Being a mathematician at heart (and on my master's certificate), working with you always felt like a home match.

Last but not least, a special thanks goes to my family and friends, who supported me throughout this journey. A huge thanks specifically goes to my mom Gaby, my dad Hansi, and my stepdad Roland, to my sister Katharina and her family, as well as to my boyfriend Lukas and his family. It wasn't always easy, but your continuous support kept me going.

Contents

Chapter 1

Introduction

Communication systems are an essential part of modern society. Today, being connected to the internet has become indispensable. Navigating when getting lost, staying in touch with friends and family over a far distance, holding a business meeting remotely – what seemed to be impossible 30 years ago is now possible due to the advances in mobile communication. According to the biannual Ericsson Mobility Report [1], the globally averaged mobile data traffic per smartphone was rising from around 1.9 GB per month in 2016 to around 15 GB per month in 2022, forecasted to further triple by the end of 2028.

This increasing mobile internet usage is enabled by constantly evolving communication standards. Recently, the mobile communication standard 5G, the successor of LTE, was released. LTE specified data rates of up to 300 Mb/s, whereas 5G features data rates of up to 10 Gb/s, thereby enabling the increase in mobile internet usage forecasted in the Ericsson Mobility Report. While 5G accounted for only 17% of mobile data traffic at the end of 2022, it is expected to rise to around 69% by the end of 2028. At the same time, the reliability of the data transmission must be ensured.

The increase in data rate and the need for reliable data transmission poses severe challenges to the communication system (see Figure 1.1). To ensure a reliable data transmission, techniques called Forward Error Correction (FEC), Error Correction Coding (ECC) or just Channel Coding are applied to make the transmitted data robust to channel noise and transmission errors. In his fundamental paper 'A Mathematical Theory of Communication' in 1948, Claude Shannon [2] laid the foundation to reliable data transmission over noisy channels. The idea is to add redundant data to the actual data to be transmitted in the encoder unit. This redundant data is then used on the receiver side in the decoder unit to identify errors that occurred during transmission and to recover the actual sent data. Although Shannon theoretically proved that an error-free transmission over a noisy channel can be achieved using ECC, this proof is nonconstructive and cannot be directly transferred to practice. Thus, reliable data transmission is still an active topic of research. The heart of reliable data transmission lies in the decoder unit, where transmission errors are detected and ideally corrected. In addition to providing a good error-correction performance, the

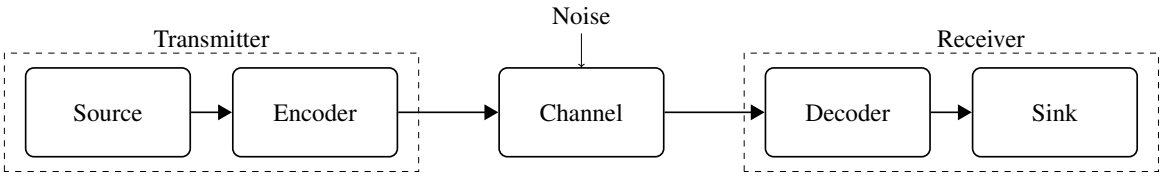


Figure 1.1: Simplified Model of a Communication System

challenges posed by new communication standards are also directly reflected to the decoder unit, stressing both a well-performing and highly efficient implementation.

Around the same time as Shannon's fundamental paper, the foundation to modern electronic systems was laid by the invention of the transistor in 1947. This invention widely replaced the much larger and power hungry vacuum tubes and mechanical relays and enabled the development of small and efficient integrated circuits (ICs) from the late 1950s and early 1960s on. In 1965, Gordon Moore predicted that the number of transistors on an IC will double every two years, while the manufacturing costs will remain about constant [3]. This famous prediction became known as 'Moore's Law' and was a driving force behind the technological advances in microelectronic systems. Although being declared dead already several times, technological and manufacturing advances like shrinking technology sizes, 2.5D packaging and 3D integration kept the law alive. Up to now, the number of transistors in a processor still follows an exponential growth, as shown in Figure 1.2.

Moore's Law still holding true could lead one to believe that the increasing demands on a communication system are completely met by the advances in microelectronic systems. However, even though the number of transistors on an IC still increases exponentially, implementations cannot become more complex in the same pace, as ICs are reaching the so-called 'Power Wall'. As shown in Figure 1.3, the power of processors is not increasing at the same rate as the number of transistors (see Figure 1.2), stagnating in the order of 100 W. This is due to a limited thermal design power, meaning that more power will result in heat that can no longer be dissipated. In turn, even though more and more transistors are on a chip, not all of them can be used at the same time to prevent overheating. In consequence, there has been a shift towards dedicated hardware accelerators known as application specific integrated circuits (ASICs), which are highly efficient ICs optimized for a specific application and are thus only activated when needed. This approach is known as dark silicon, which refers to the areas of the chip that are switched off due to power limitations. Concerning a flexibility-cost trade-off, ASICs perform a predefined application-specific task and are therefore very inflexible. In turn, they provide a high implementation efficiency.

The implementation efficiency of an ASIC implementation is qualified according to the following Key Performance Indicators (KPIs): throughput, area, latency, power, energy efficiency, area efficiency and power density. The decoding unit in a communication system is a crucial part and has to be highly efficient. It is therefore usually implemented as an ASIC. In addition to the aforementioned KPIs, the performance of the decoding unit is qualified by its communications performance, i.e., its capability of detecting and correcting

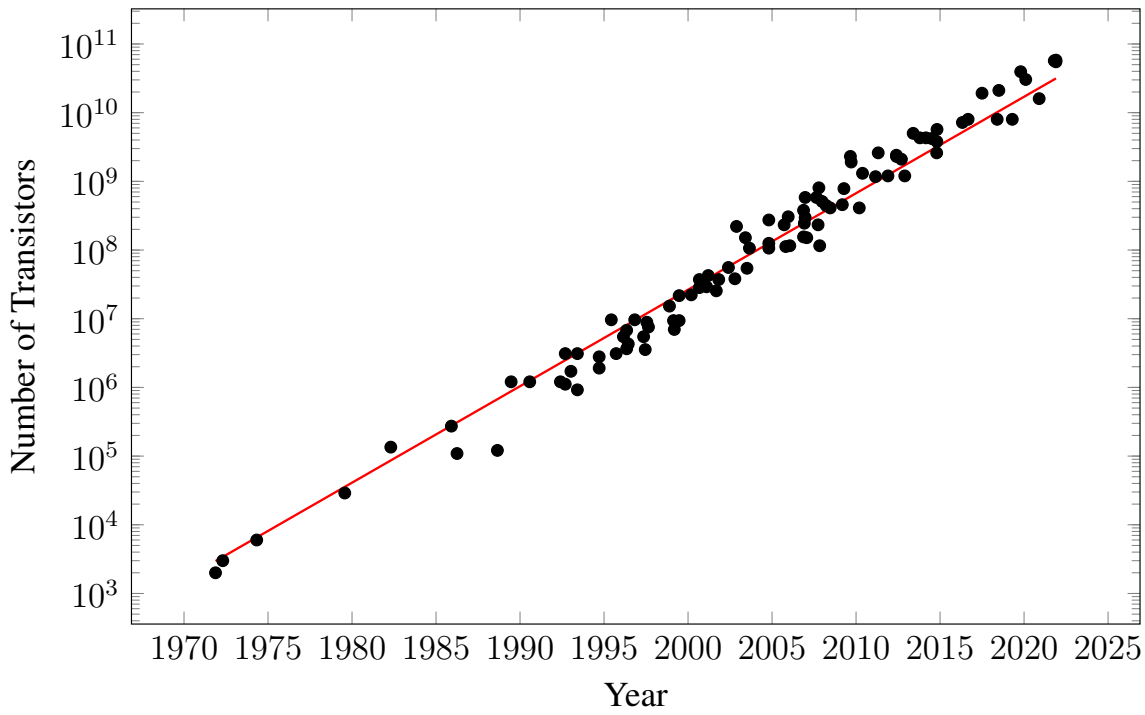


Figure 1.2: Number of Transistors in Processors [4]

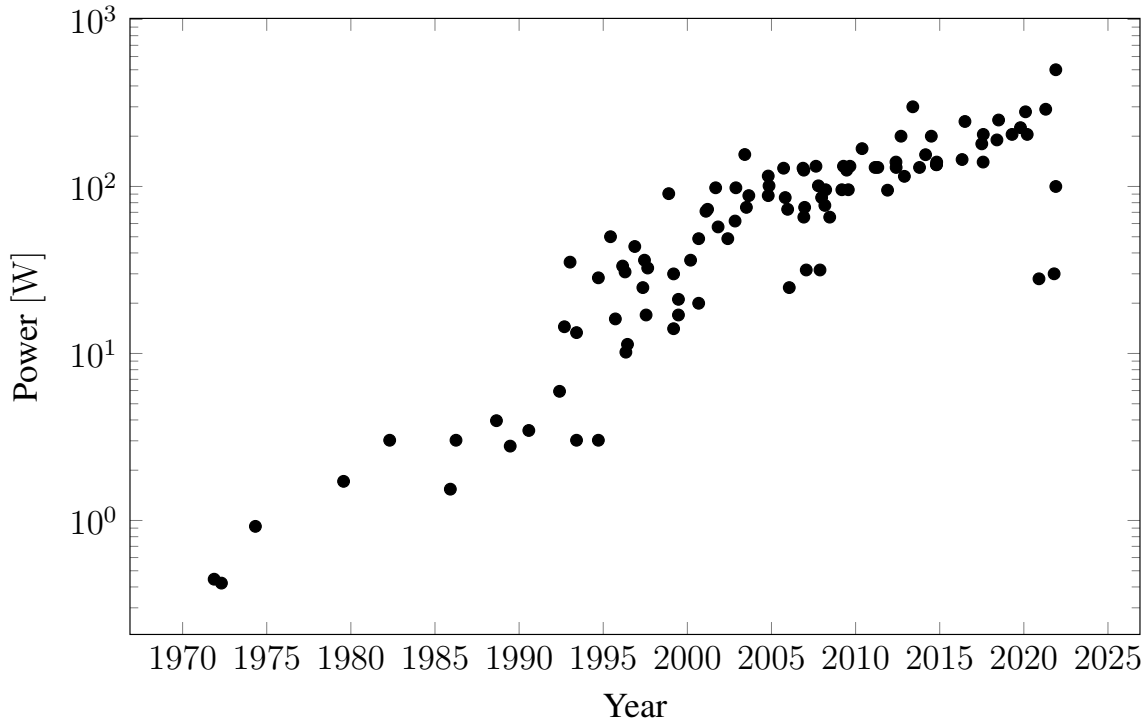


Figure 1.3: Typical Power of Processors [4]

errors from a transmission over a noisy channel. Table 1.1 gives an overview of the single KPIs. Optimizing both the KPIs of an efficient decoder implementation and its communications performance results in a Pareto optimization problem, where the single KPIs have to be minimized or maximized (see Table 1.1).

Table 1.1: KPIs of a decoder implementation

KPI	**Description**	**Unit**	**Min/Max**
Throughput	The amount of data that can be processed by the ASIC in a given time	$\mathrm{Gb/s}$	max
Area	The area of the ASIC	mm^2	min
Latency	The time it takes for one input to be output, can be derived as the number of clock cycles times the achieved frequency	ns	min
Power	The power consumption of the ASIC	mW	min
Energy Efficiency	Indicates how much energy is needed to process one bit, calculated as Power divided by Throughput	pJ / bit	min
Area Efficiency	Indicates how much data can be processed in a given time and area, calculated as Throughput divided by Area	$\mathrm{Gb/s/mm}^2$	max
Power Density	The amount of power per area, calculated as Power divided by Area	$\mathrm{mW/mm}^2$	min
Communications Performance	A measure for how well the decoding unit can detect and/or correct transmission errors	BER/FER @ SNR	min

The main challenge lies in the fact that the single KPIs cannot be derived from a closed formula, but have to be simulated elaborately, making it impossible to optimize in a structured way. In addition, the various KPIs are partially contradicting, thus, they have to be thoroughly trade off depending on the application's demands: For instance, the communications performance of a coding system improves with an increasing blocklength n of the used code and with a large amount of redundant information k, or, in other words, a low code rate $r = k/n$ (see Section 1.2). In addition, in case of an iterative decoding unit, a large number of iterations I positively impacts the communications performance. However,

the authors of [5] estimated the throughput T of a FEC decoder architecture by

$$T \sim n \cdot r \cdot \frac{1}{I} \cdot \pi \cdot f \cdot (1 - \omega), \tag{1.1}$$

where π reflects the amount of parallelism, f the clock frequency, and ω a normalized value to account for timing overheads. While the communications performance increases with a low code rate and a large number of iterations, the throughput benefits from a high code rate and a low number of iterations. This example highlights the impossiblity to find a solution which is optimal for every KPI at the same time.

When dealing with a Pareto optimization problem, it is of large interest to know so-called Pareto fronts, i.e., valid solutions to the optimization problem that are optimal for (at least) one objective. They act as a baseline and are of large value for classifying the quality and estimating the potential of possible improvements of a specific implementation. For the communications performance, obtaining the optimal performance of the decoding unit is possible through so-called maximum likelihood (ML) decoding. ML decoding mathematically optimizes the communications performance of the decoding unit by maximizing the probability of a correct recovery from transmission errors. Even though ML decoding is an NP-hard problem [6], i.e., it is unknown whether it can be solved in polynomial time, solving the decoding problem optimally is still of great interest for the following reasons:

- Having a Pareto front for the communications performance of any coding scheme serves as a baseline for any heuristic decoding algorithm applied to this coding scheme. It enables to estimate the potential for improvements and simplifies making trade-offs between the communications performance and other implementation-specific KPIs.
- ML decoding can be applied to any coding scheme. It is therefore a powerful tool to compare the quality of different coding schemes without the bias of a heuristic decoding algorithm.
- For applications that have very loose requirements on the implementation-specific KPIs, especially on latency, ML decoding can be applied as decoding algorithm. Note that solving an NP-hard problem is oftentimes possible for small problem sizes, but the complexity scales exponentially with the problem size.

The ML decoding problem can be formulated as a binary integer linear programming (ILP) (see Section 1.2). It has been shown that it can be solved efficiently using a dedicated branch-and-bound algorithm, which works as follows: By possibly setting every bit of the received information to ‘0’ or ‘1’, a binary search tree is spanned, which is traversed by the algorithm to find the most probable information that has been sent. While traversing the tree, linear programming (LP)-relaxations of the original binary ILP have to be solved in order to generate bounds for the objective value of the ILP. Parts of the search tree can be

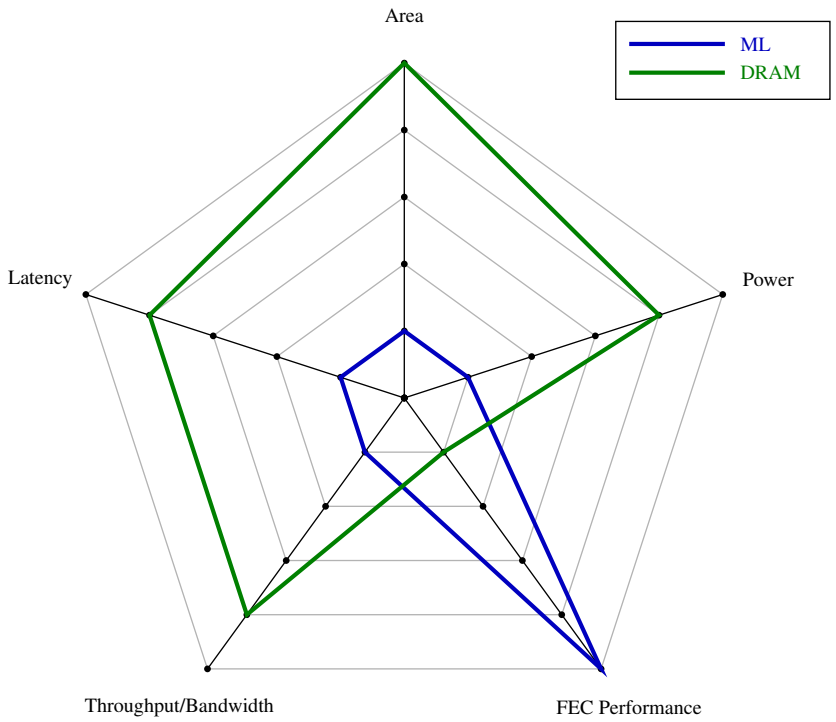


Figure 1.4: Optimality of the KPIs for ML Decoding and ECC in DRAM

pruned whenever these bounds can guarantee that the optimal solution cannot be found in the specific sub-tree. The main effort of ML decoding thus lies in repeatedly solving the LP problems, especially in the context of the branch-and-bound environment.

Error-correction techniques are not only useful to guarantee a reliable data transmission. Also data storage can be seen as a simplified communication system, with the data being transmitted temporally instead of spatially. Just like data transmission, data storage can also be modeled by a noisy channel. Since data storage is highly driven by cost per bit, shrinking technologies and higher densities are employed. In the case of modern dynamic random-access memorys (DRAMs), this leads to an increase of errors occurring, e.g., due to cell leakage or crosstalks between the different word- and bitlines. Thus, employing ECC is mandatory to maintain the reliability of modern DRAMs.

The challenge for ECC in DRAMs is to provide a solid error-correction capability with very hard restrictions on the implementation costs. First, the ECC unit cannot increase the overall read/write latency, which inhibits a tight latency constraint. Second, the ECC unit has to provide enough throughput to not lower the DRAM's bandwidth. Third, the area inside a DRAM or its controller is very limited. Because of this, only very basic, yet dedicated, decoding schemes can be employed for the use in DRAMs.

Viewed in context of Pareto optimization, efficient ML decoding and ECC in DRAMs are two opposing problems: For ML decoding, the optimal error-correcting performance is a hard constraint, while the implementation efficiency, especially regarding throughput,

latency and energy efficiency, has to be optimized. It can thus be expressed by the following optimization problem:

$$\begin{aligned} \max \quad & \text{Implementation Efficiency} \\ \text{s.t.} \quad & \text{Communications Performance = ML} \end{aligned}$$

For ECC in DRAMs, there is a hard constraint on the implementation efficiency, while the error-correcting performance has to be optimized:

$$\begin{aligned} \max \quad & \text{Communications Performance} \\ \text{s.t.} \quad & \text{Implementation Efficiency} \geq \text{Threshold} \end{aligned}$$

The multiple dimensions of the Pareto optimization problem are depicted as a spiderweb in Figure 1.4. The KPIs energy efficiency and area efficiency are left out on purpose as they can be derived from the other KPIs. The two applications of interest, ML decoding and ECC in DRAMs, are inserted as blue and green lines, depicting how far off they are from the optimum for the specific KPI. The line being close to the center of the spiderweb means that the application is not at all optimized, while the line being close to the outer edges of the spiderweb means that the application is already at the optimum. For example, DRAMs are highly optimized for area, power and bandwidth. But there is still much room for improvement on the ECC performance. In the case of ML decoding, the ECC performance is at its optimum, but all implementation-specific KPIs can still be largely optimized.

1.1 Contributions and Outline

This thesis will tackle both of the aforementioned problems: improving the implementation efficiency of ML decoding while maintaining the optimal error-correcting performance, and improving the error-correcting performance for ECC in DRAMs while respecting the hard constraints on the implementation efficiency. This thesis is split in two parts. Chapters 2, 3 and 4 focus on ML decoding and the improvement of parts of the algorithm. The remaining Chapters 5 and 6 focus on using ECC for data storage in DRAMs. In detail, the novel contributions of this thesis are:

- **Ensemble BP Decoding (Chapter 2)**

  To solve the ML decoding problem, a branch-and-bound framework is employed. This framework reduces the ML decoding problem to repeatedly solving LP decoding problems and running heuristic decoding algorithms to obtain good bounds on the optimal solution. Tight bounds reduce the search space of the branch-and-bound algorithms and result in a faster termination of the algorithm. Heuristic decoding algorithms are usually code-specific and therefore lack flexibility.

In this first chapter, the Ensemble Belief Propagation (BP) decoding algorithm, a new heuristic decoding algorithm, is presented as a generalization of Multiple-Bases Belief-Propagation (MBBP) decoding [7, 8] and Automorphism Ensemble Decoding (AED) [9–11]. It relies on the BP decoding algorithm, originally designed for low-density parity-check (LDPC) codes, but is modified in such a way that it can be used for any code class. The approach of Ensemble BP is two-step: First, a suitable sparse matrix representation of the code is constructed by means of mathematical optimization. Second, a suitable diverse set of matrix representations is selected by solving an instance of the Maximum Coverage Problem. Performance simulations show that Ensemble BP can be used as flexible, universal decoding algorithm for any code class with negligible performance loss over the code-specific state-of-the-art decoding algorithm.

- **LP Decoding (Chapter 3)**

  The second large hurdle to take for efficient ML decoding is a good LP decoding algorithm. In recent years, the alternating direction method of multipliers (ADMM) algorithm [12] has been shown to perform very efficiently in the context of LP decoding [13]. Its large bottleneck is a projection onto the so-called parity polytope, which is the most complex and time-consuming step. In this chapter, a new low-complexity projection algorithm optimized for an efficient hardware implementation is presented. While state-of-the-art projection algorithms require hardware-inefficient sorting algorithms, our new projection algorithm does not require sorting and relies on successively fixing components of the projection by an iterative approach. This improved projection algorithm largely reduces the complexity and runtime of the overall algorithm.

- **ML Decoding (Chapter 4)**

  To solve the ML decoding algorithm, a branch-and-bound framework is employed. Repeatedly, LP decoding and heuristic decoding algorithms have to be solved for specific sub-problems. In this chapter, an elegant way to incorporate the new projection algorithm from the previous chapter into the branch-and-bound framework is presented. This allows to fully benefit from the complexity reduction of the projection algorithm not only for LP decoding, but for the whole ML decoding problem.

  Additionally, the runtime of ML decoding is further reduced by modifying the representation of the code under consideration. With this reduction in runtime by up to 81%, a previously unknown ML Frame Error Rate (FER) performance curve was computed.

  Since ML decoding is a valuable benchmark for the performance of any code and decoding algorithm, all new findings made throughout this thesis were published open-source in our channel codes database [14].

- **DRAM Background and Modeling (Chapter 5)**

  As the first chapter in the second part of this thesis, the DRAM basics are recapped. As scientific contribution of this chapter, the DRAM model based on Petri nets from [15] is extended to include checking the DRAM's timing behavior instead of only checking for valid state transitions. The extended model provides an easily understandable DRAM system model and is used for checking the correctness of the DRAM controller according to the respective Joint Electron Devices Engineering Councils (JEDEC) standard and for easily calculating the DRAM's power consumption from its data sheet information. Maintaining the correct timing behavior is crucial to avoid errors, which can occur with incorrect timings at very high clock rates.

- **DRAM ECC (Chapter 6)**

  In this chapter, the error behavior of a single DRAM cell is analyzed and, for the first time, an accurate information-theoretic model is derived from the observed asymmetric error behavior. Based on this error model, a custom, lightweight error-mitigation method is derived. It improves the error behavior of DRAMs massively without affecting the DRAM's latency or bandwidth.

  In addition, a new method for error correction inside the DDR4 DRAM controller is presented, which uses data bus inversion (DBI), an inherent method in the DDR4 standard. While usual error-correcting schemes for DRAMs require a special ECC architecture (i.e., a 9th chip per Dual Inline Memory Module (DIMM) storing the redundancy for the other 8 memory chips), this error-correcting method is specifically customized to a DDR4 DRAM controller, makes use of the aforementioned asymmetric error behavior of a DRAM cell and thus allows to add a simple, yet effective, ECC mechanism for non-ECC DRAMs.

The contributions of this thesis were published in the following peer-reviewed conference and journal papers: [15–24]. The work based on [22] additionally resulted in a patent [25].

The work of Chapter 3 and parts of Chapter 2 and Chapter 4 were carried out in cooperation with the Optimization Research Group of the University of Kaiserslautern-Landau within the DFG grant no. WE 2442/9-2. The remaining work of Chapter 4 was supported by the InnoProm program (project no. 84003923) of the EU and the state Rhineland-Palatinate, Germany.

In the remainder of this chapter, the basics and terminologies used throughout this thesis are introduced.

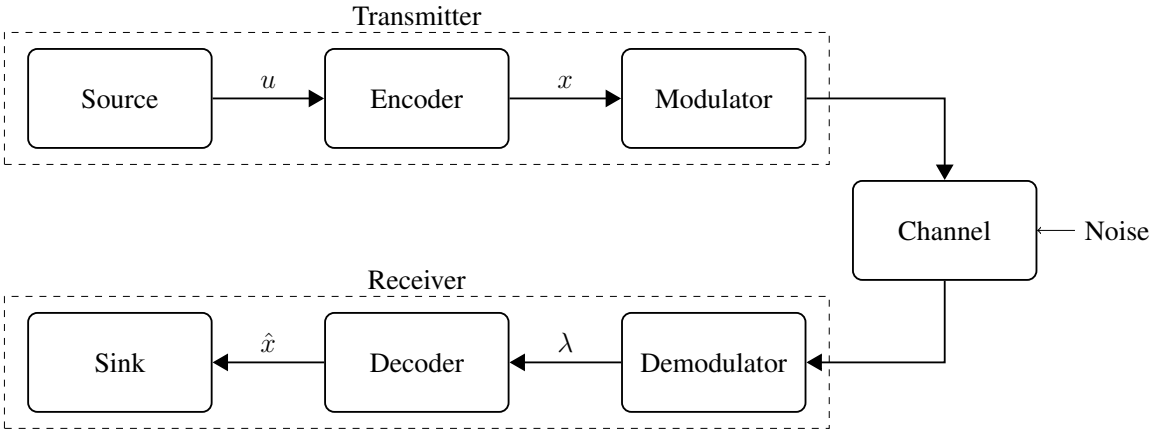


Figure 1.5: Model of a Communication System

1.2 Channel Coding Basics and Terminology

The following notation and naming convention will be used throughout this thesis.

Consider the transmission system in Figure 1.5. On transmitter side, the information to be transmitted u is an information word of length k. Throughout this thesis, u is a binary information word, i.e., $u \in \mathbb{F}_2^k$. In the encoder block, u is encoded to a codeword x of length n, $n \geq k$, by appending $m = n - k$ parity bits. In case of a binary linear code, the encoding function maps an information word u to a codeword x according to

$$\begin{aligned} \text{enc}\colon \mathbb{F}_2^k &\to \mathbb{F}_2^n \\ u &\mapsto u \cdot G =: x, \end{aligned} \tag{1.2}$$

where $G \in \mathbb{F}_2^{k \times n}$ is called generator matrix. The set of all codewords generated by a specific generator matrix G, $\mathcal{C} := \{\text{enc}(u) \mid u \in \mathbb{F}_2^k\} \subseteq \mathbb{F}_2^n$, is called code. The ratio $r = k/n$ is called coderate.

At the modulator, the codeword bits are mapped to complex transmission symbols, which encode the codeword bits into a sinusoid wave that is transmitted, e.g., over air in case of a wireless transmission. In this thesis, modulation is restricted to Binary Phase Shift Keying (BPSK), where one codeword bit is mapped to one transmission symbol.

The channel depicts the noisy transmission of the modulated signal from the transmitter to the receiver. Channel noise for a wireless data transmission is usually modeled stochastically using additive white Gaussian noise (AWGN). The amount of noise for an AWGN channel is measured as the signal-to-noise ratio (SNR) E_b/N_0, where E_b is the energy per bit and N_0 the power spectral density of the noise. E_b/N_0 is usually measured in dB, where a low value represents a very noisy channel and a high value a less noisy channel.

On receiver side, the demodulator's task is to map the incoming noisy sinusoid to binary data. Since the incoming signal contains more information than just '0' or '1', demodulation for BPSK modulation and an AWGN channel yields additional so-called soft-

information in the form of Log Likelihood Ratio (LLR) values λ_i for every position of the received information $0 \leq i < n$. They are defined as

$$\lambda_i = \ln\left(\frac{\mathbb{P}(y_i \mid x_i = 0)}{\mathbb{P}(y_i \mid x_i = 1)}\right). \tag{1.3}$$

The sign of λ_i determines the hard-decision y of the demodulation according to

$$y_i = \begin{cases} 1, & \lambda_i < 0, \\ 0, & \lambda_i \geq 0. \end{cases} \tag{1.4}$$

The absolute value $|\lambda_i|$ is called reliability of position i.

The heart of the transmission system lies in the decoding unit, which receives the soft-information λ and aims to find the most probable codeword that has been sent. In particular, the decoding function maps the incoming real-valued LLR values to binary values according to

$$\begin{aligned} \mathrm{dec}\colon \mathbb{R}^n &\to \mathbb{F}_2^n \\ \lambda \ &\mapsto \mathrm{dec}(\lambda) := \hat{x}, \end{aligned} \tag{1.5}$$

where $\hat{x} \in \mathbb{F}_2^n$ is the corrected codeword. Decoding was successful if $\hat{x} = x$. Otherwise, it is called decoding error. Multiplying $\hat{x}$ with the inverse generator matrix G^{-1} then yields $u' \in \mathbb{F}_2^k$, which equals the original information word u in case of a successful decoding. For decoding, a so-called parity-check matrix $H \in \mathbb{F}_2^{m \times n}$ is used. It is defined by

$$H \cdot G^\top = 0, \tag{1.6}$$

so it holds that

$$H \cdot x \equiv 0 \pmod 2 \quad \forall x \in \mathcal{C}, \tag{1.7}$$

i.e., the code is the *kernel* of the parity-check matrix. Solving the decoding task optimally, i.e., maximizing the probability of a successful decoding, yields the ML decoding problem

$$\begin{aligned} x_{\mathrm{ML}} &= \arg\max_{x \in \mathcal{C}} \mathbb{P}(y \mid x) \\ &= \arg\min_{x \in \mathcal{C}} \lambda^\top x, \end{aligned} \tag{1.8}$$

where the second equality was shown in [26]. ML decoding is an NP-hard problem and can be formulated as an ILP [27]:

$$\begin{aligned} \min_{x \in \mathbb{F}_2^n} \quad & \lambda^\top \cdot x \\ \text{s.t.} \quad & H \cdot x \equiv 0 \pmod 2 \end{aligned} \tag{1.9}$$

The quality or error-correction performance of a decoding algorithm is simulated in a virtual transmission model, which covers the whole transmission system from Figure 1.5. After every single transmission frame, an additional evaluation unit checks whether decoding was successful, i.e., whether $\hat{x} = x$, or a decoding error, i.e., $\hat{x} \neq x$. To obtain a good

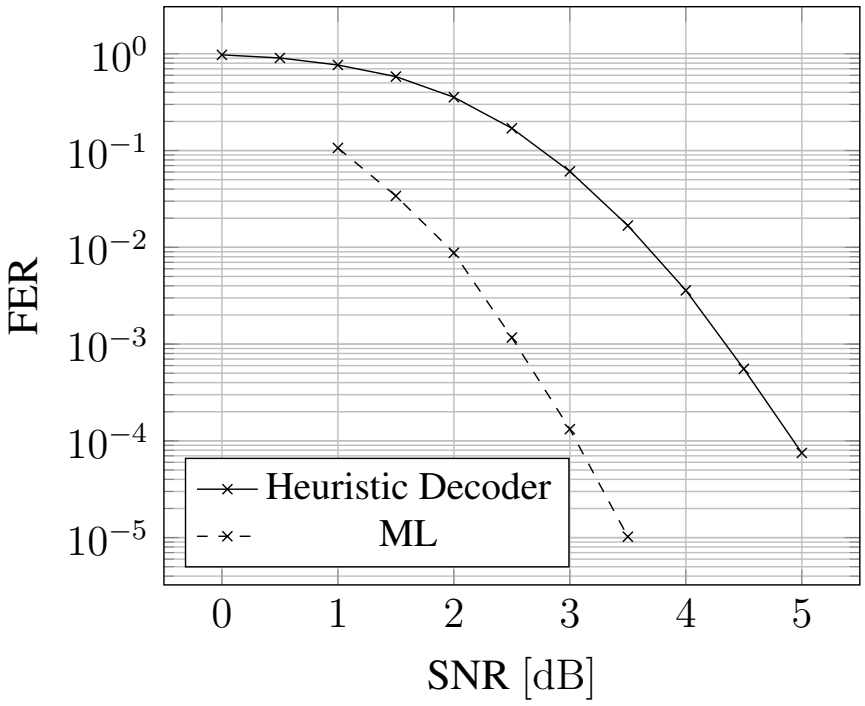


Figure 1.6: Result of a Communications Performance Simulation

performance indication of the decoder, the transmission system is simulated until around 100 decoding errors occurred for every channel SNR of interest. The result is measured as a FER, which is defined as

$$\text{FER} = \frac{\text{Number of incorrect frames}}{\text{Number of total frames}}. \tag{1.10}$$

Similarly, the Bit Error Rate (BER) is defined as the error rate on bit-level. An exemplary result of such a FER simulation is depicted in Figure 1.6. On the x-axis, the channel SNR is given in dB. The channel quality increases when moving from left to right, i.e., less noise is inflicted at a high SNR. On the y-axis, the FER is given logarithmically. Moving from bottom to top, the number of decoding errors increases. As a reference, for an FER of $10^0 = 1$, every simulated frame was a decoding error. The figure is now read as follows: The two plots depict the error-correcting performance of two different decoding algorithms. The dashed plot depicts the performance of the ML decoder and the solid plot the performance of a heuristic decoding algorithm. It can be seen that the ML decoder performs better than the heuristic decoder: For example, to achieve an error rate of 10^{-3}, which is oftentimes a point of reference for wireless transmission, the heuristic decoder needs an SNR of around 4.3 dB, whereas the ML decoder only needs an SNR of around 2.5 dB. This means that the ML decoder is able to perform well under poorer channel conditions, or, in turn, that the transmission power can be reduced when there are good channel conditions. The gap between the plots of the heuristic decoder and the ML decoder depicts the potential

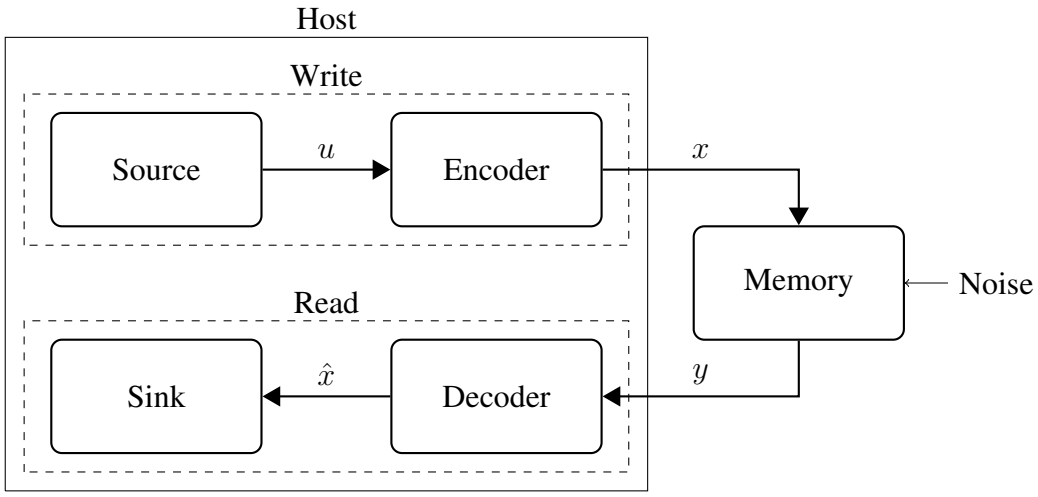


Figure 1.7: Data Storage as Communication System

for improvements on the heuristic decoder. ML decoding is optimal and can thus serve as a bound on the error-correcting performance of any heuristic decoding algorithm.

While transmission systems are the most obvious kind of communication system, also a data storage can be viewed as a noisy data transmission: Instead of a spatial data transmission in the case of a transmission system, data storage can be viewed as a temporal data transmission. In the same fashion, data storage is also prone to errors and can thus be modeled as a noisy channel. The model is shown in Figure 1.7. While for data transmission (see Figure 1.5), the binary information is first modulated onto transmission symbols, in data storage, the binary data can directly be stored in the memory. Thus, no (de-)modulation is necessary.

In this thesis, the focus will be on DRAMs, where a single bit is stored in each DRAM cell. The noisy channel for a single DRAM cell can be modeled using a discrete binary channel shown in Figure 1.8: A stored 0 is falsely read as a 1 with probability q and a stored 1 is falsely read as a 0 with probability p. The probabilities $0 \leq q, p \leq 1$ are called crossover probabilities.

The following variations of this discrete binary channel model will be used throughout this thesis:

- If $q = p$, the channel is called binary symmetric channel (BSC).
- If $q \neq p$, the channel is called binary asymmetric channel (BAC).
- If $q = 0, p \neq 0$, the channel is called Z-channel or fully asymmetric channel. Its name stems from the 'Z' shape in Figure 1.8 when removing the arc from 0 to 1.

A theoretical measure for the quality of a channel is given by its *channel capacity* C, as introduced in Shannon's "A Mathematical Theory of Communications" [2]. C can be

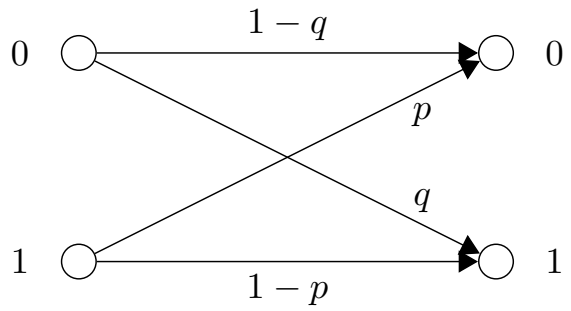


Figure 1.8: Discrete Binary Channel

defined as the maximum transmission rate, such that information can be transmitted reliably over the channel. With the help of the binary entropy function

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{1.11}$$

the channel capacity of the binary channel can be written as

$$\begin{aligned} C_{BC} = &\frac{q}{1-q-p}\cdot H(p) - \frac{1-p}{1-q-p}\cdot H(q) \\ &+ \log_2\left(1 + 2^{\frac{H(q)-H(p)}{1-q-p}}\right), \end{aligned} \tag{1.12}$$

with $q \leq p$. For the special case of a BSC, the channel capacity can be written as

$$C_{\text{BSC}} = 1 - H(p), \tag{1.13}$$

where $p = q$ is the crossover probability of the BSC. The channel capacity of the Z-channel calculates to

$$C_{\text{Z}} = \log_2\left(1 + (1-p)\cdot p^{\frac{p}{1-p}}\right). \tag{1.14}$$

For small crossover probabilities p, Equation 1.14 can be rewritten as

$$C_{\text{Z}} \approx 1 - \frac{1}{2}H(p), \tag{1.15}$$

so clearly $C_{\text{Z}} > C_{\text{BSC}}$. This means that, in theory, information can be transmitted more reliably over the Z-channel than over the BSC. However, suitable coding schemes have to be employed to benefit from this theoretical result.

Part I

Towards Hardware ML-Decoding

Chapter 2

Ensemble BP Decoding

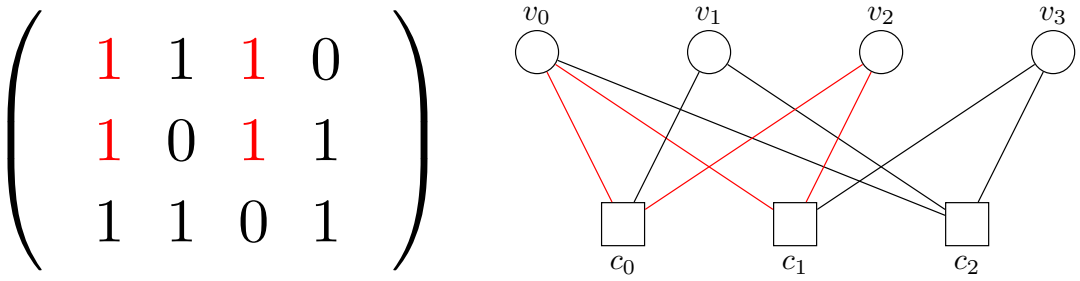
Everyday life's decoding applications use heuristic algorithms to complete the decoding task. Even though their performance is not optimal, they trade off a slight degradation in communications performance against a large degradation in decoding complexity, thereby making it a favorable choice for many applications. Heuristic algorithms are usually tailored to a specific code class and they use the code-specific structure to increase the communications performance.

Universal heuristic decoding algorithms are desirable, because they can be used for any code class. Thus, also improvements in the algorithm are beneficial for decoding any code class. On the downside, a universal decoding algorithm cannot increase the communications performance by making use of a code-specific structure. Therefore, a universal decoding algorithm is expected to have worse communications performance compared to code-specific heuristics.

In this chapter, a universal heuristic decoding algorithm is presented. It is applicable to any code class and can compete with the code's state-of-the-art decoding algorithm. In detail, this chapter makes the following contributions:

- The problem of binary matrix sparsification is formulated as an ILP and applied to the parity-check matrices of several codes to improve their performance under BP decoding.
- A new decoding technique, called Ensemble BP decoding, is presented as a universal decoding algorithm for any code class. This new algorithm is shown to compete with the communications performance of the code's state-of-the-art decoding algorithm.

The contributions of this chapter were published in [16, 17]. The publication [17] was carried out in cooperation with the Optimization Research Group of the University of Kaiserslautern-Landau.



(a) H-Matrix **(b)** Tanner Graph

Figure 2.1: H-Matrix and Corresponding Tanner Graph

2.1 Background and State of the Art

Heuristic decoding algorithms perform a good trade-off between achieving a good communications performance and keeping a low implementation complexity. The most popular heuristic decoding algorithms are iterative algorithms. By adjusting the number of performed iterations, the mentioned trade-off can easily be achieved and tailored to the demands of specific applications.

2.1.1 Belief Propagation Decoding

BP decoding [28] is a widely adopted concept in the heuristic decoding of FEC codes. BP is a so-called message-passing algorithm, i.e., it relies on the iterative exchange of messages ("beliefs") in a graphical model, which typically varies among different code classes. This graphical model is, e.g., the component code's Trellis for Turbo codes [29], the Factor Graph for Reed-Muller (RM) codes [30] or Polar codes [31], or the Tanner graph for LDPC codes [32, 33]. In the remainder of this chapter, BP decoding will denote BP decoding on the Tanner graph.

The Tanner graph representation [34] of a code is directly derived from its parity-check matrix $H \in \mathbb{F}_2^{m \times n}$. In detail, each column of H defines an instance of a so-called Variable Node (VN), and each row of H defines an instance of a so-called Check Node (CN). Each one-entry in H induces an edge between the corresponding variable and check node. Figure 2.1 shows the Tanner graph of a given H matrix with 3 CNs and 4 VNs. An equivalent representation of the connections in the Tanner graph is given via the following sets. First, for every VN $i \in \{0, \dots, n-1\}$ a set $\mathcal{M}(i)$ of connected CNs is defined:

$$\mathcal{M}(i) = \{j \in \{0, \dots, m-1\} \mid H_{i,j} \neq 0\}\} \tag{2.1}$$

Second, for every CN $j \in \{0, \dots, m-1\}$ a set $\mathcal{N}(j)$ of connected VNs is defined:

$$\mathcal{N}(j) = \{i \in \{0, \dots, n-1\} \mid H_{i,j} \neq 0\}\} \tag{2.2}$$

During the iterations of the BP algorithms, messages are exchanged on the edges of the Tanner graph and node updates are performed. The calculations in the CNs and VNs follow specific node update rules. In the following, the node update rules according to the Sum-Product Algorithm (SPA) are reviewed. Let $\varepsilon_{j\to i}$ denote the messages from CN j to VN i and $z_{i\to j}$ the messages from VN i to CN j. The first messages from a VN to all of its connected CN are initiated with the VN's corresponding LLR value, i.e.,

$$z_{i\to j} = \lambda_i \quad \forall j \in \mathcal{M}(i). \tag{2.3}$$

With these incoming messages $z_{i\to j}$, each CN calculates

$$\operatorname{sgn}\left(\varepsilon_{j\to i}\right) = \prod_{i'\in\mathcal{N}(j)\setminus i} \operatorname{sgn}\left(z_{i'\to j}\right) \tag{2.4}$$

and

$$|\varepsilon_{j\to i}| = \Phi^{-1}\left(\sum_{i'\in\mathcal{N}(j)\setminus i} \Phi\left(|z_{i'\to j}|\right)\right), \tag{2.5}$$

where

$$\Phi(x) = \Phi^{-1}(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right). \tag{2.6}$$

It is important to notice that the message $\varepsilon_{j\to i}$ from a CN j to a VN i is calculated from all incoming edge messages $z_{i'\to j}, i' \in \mathcal{N}(j)$, except for the edge message coming from VN i. This is important, since the BP algorithm requires the messages sent to a some node to be independent from the message that came from this specific node. This independence is a strong requirement for the algorithm to perform well.

At the VNs, new messages to the connected CNs are calculated as

$$z_{i\to j} = \lambda_i + \sum_{j'\in\mathcal{M}(i)\setminus j} \varepsilon_{j'\to i}. \tag{2.7}$$

Note, again, that the sum runs over all incoming messages, except for one coming from CN j. As soon as the maximum number of decoding iterations is reached, the so-called a posteriori probability (APP) Λ_i yield the decoding result. It is calculated as

$$\Lambda_i = \lambda_i + \sum_{j\in\mathcal{M}(i)} \varepsilon_{j\to i} \tag{2.8}$$

and the binary decoding result is finally determined via

$$x_i' = \begin{cases} 1, & \Lambda_i < 0, \\ 0, & \Lambda_i \geq 0. \end{cases} \tag{2.9}$$

The CN update performed in Equation 2.5 is quite complex due to the $\tanh$ operation in the calculation of Φ (see Equation 2.6). Thus, to achieve low implementation complexity,

the so-called Min-Sum (MS) is used for the CN update. Doing so, Equations 2.4 and 2.5 are replaced by

$$\varepsilon_{j\to i} = \gamma \cdot \left(\prod_{i' \in \mathcal{N}(j)\setminus i} \operatorname{sgn}\left(z_{i'\to j}\right) \right) \cdot \min_{i' \in \mathcal{N}(j)\setminus i} \left(|z_{i'\to j}|\right). \tag{2.10}$$

γ is called Extrinsic Scaling Factor (ESF) and aims to mitigate the loss in communications performance when using MS instead of SPA. Simulations have shown that good values for γ are 0.75 or 0.875, both in terms of communications performance and implementation complexity.

BP decoding, and especially MS decoding, can be implemented in a highly efficient high-throughput and low-latency decoding hardware. This is due to the low-complexity node update rules in terms of the MS approximation and an inherent decoding parallelism on the underlying Tanner graph [35].

Regarding its communications performance, BP was originally designed for graphs without cycles and it is even an optimal algorithm if the underlying graph is cycle-free [28], since the messages are independent in this case. This is also the reason for leaving the destination node out of the calculations in the node updates (see Equation 2.5 and Equation 2.7), i.e., the outgoing edge message from one node should not contribute to its returning incoming edge message. In [36], the authors proposed the use of "loopy" BP for the decoding task, since a code's Tanner graph usually contains cycles (see the red edges in Figure 2.1b or equivalently the red ones in Figure 2.1a). For graphs with cycles, the optimality of the algorithm can no longer be guaranteed, since information from one node is fed back to itself over a short number of iterations, thereby destroying the independence of the edge messages. Nevertheless, BP decoding works especially well for LDPC codes due to their sparse parity-check matrix. On the one hand, a sparse matrix naturally contains fewer short cycles, which disturb the communications performance. The more short cycles a code's parity-check matrix contains, the worse the decoding performance. On the other hand, a sparse matrix leads to few edges in the corresponding Tanner graph. This means in turn that the calculations from Equation 2.5 and Equation 2.7 become easier, as the sum has to consider less edge messages.

Due to the availability of high-throughput and low-latency BP decoding hardware and the universal applicability to any linear code, BP serves as a good candidate to explore its suitability as universal decoding algorithm.

Several works already tried to apply BP decoding to non-LDPC codes: For instance, the authors of [37] propose to adapt the parity-check matrix in every iteration of the BP algorithm, which results in a large computational overhead, while the authors of [38] introduce weights on the edges of the Tanner graph (obtained by a neural network) that can help to mitigate the effects of short cycles in a single BP decoding instance.

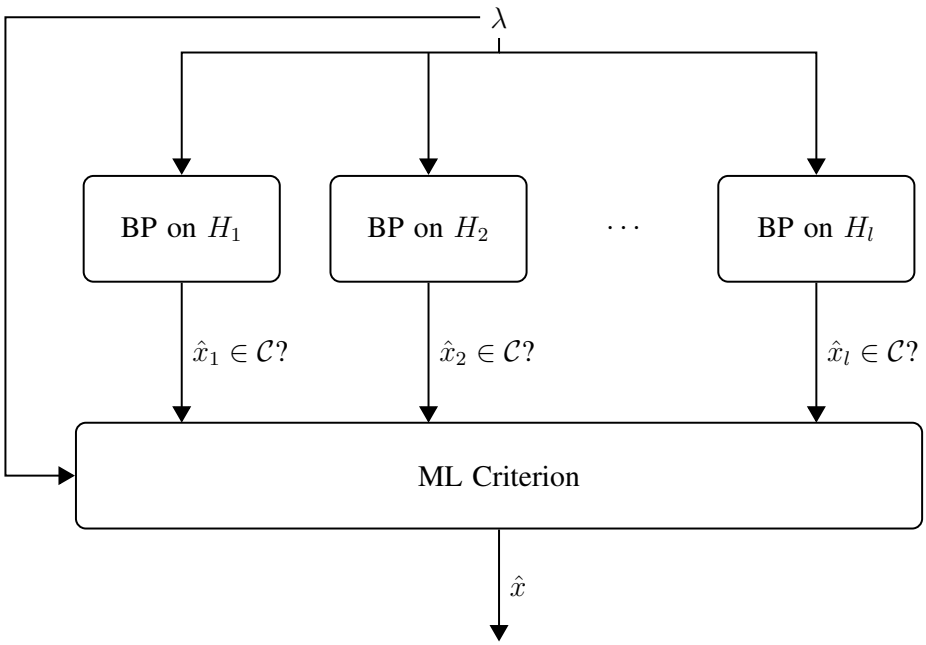


Figure 2.2: Architecture of MBBP Decoding [8]

2.1.2 Multiple-Bases Belief Propagation

Another approach to apply BP decoding on the Tanner graph for non-LDPC codes is MBBP [7, 8]. Since different parity-check matrices of the same code can lead to different Tanner graphs, the performance of the BP algorithm is tied to the parity-check matrix used to represent the code. The idea of MBBP is therefore to use an ensemble of different parity-check matrices for decoding with multiple instances of a BP algorithm in parallel. Thereby, a certain decoding diversity is introduces, which can mitigate the effects of short cycles.

In detail, the idea of MBBP decoding proposed in [7, 8] is depicted in Figure 2.2. It requires different representations of the code's parity-check matrix H, $H_1, \ldots, H_l$, $l \in \mathbb{N}$, from which l corresponding BP decoding instances are created that all decode the same input y. Every decoding instance that converged to a valid codeword ($\hat{x}_i \in \mathcal{C}$) processes its output to a list, from which the most likely codeword $\hat{x}$ according to a ML criterion is chosen. While the idea of MBBP is quite simple, its crucial point lies in the generation/selection of the different matrices H_i. To obtain a good set of different matrices, two points have to be considered: First, each single BP decoding instance has to perform well. Therefore, suitable matrix representations H_i have to be found. The requirements for matrix representations suitable for BP decoding are described in Section 2.2. Second, the ensemble of BP decoding instances has to be diverse, meaning that they complement each other well regarding their ability to correctly decode certain error patterns.

In [7, 8], MBBP decoding is therefore only applied to short cyclic codes, where the

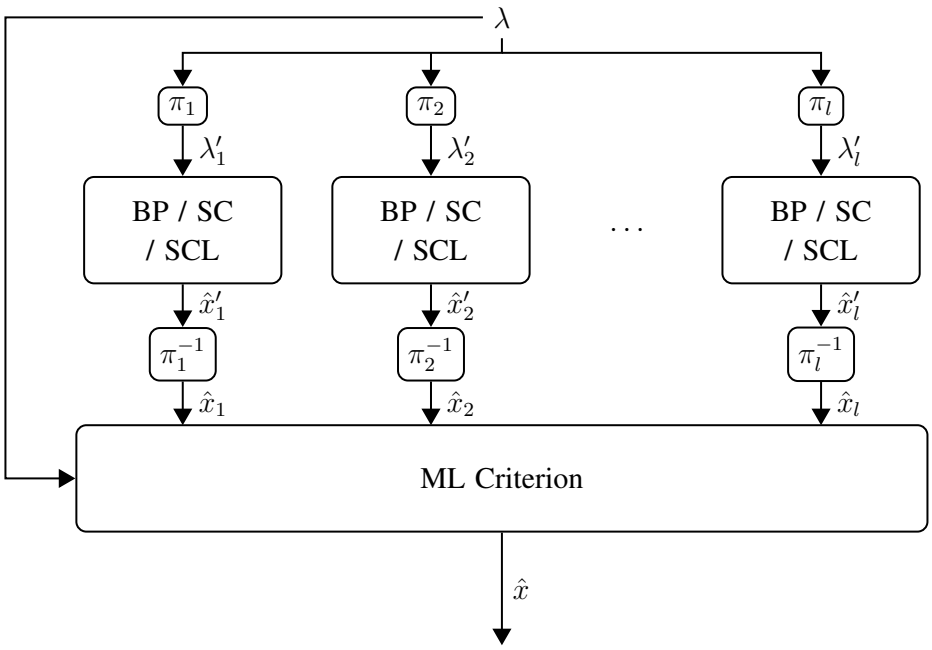


Figure 2.3: Architecture of AED [9]

different parity-check matrices are obtained by cyclic shifts of minimal dual codewords. Thus, MBBP is restricted to certain code classes and cannot be used as a universal decoding algorithm. A general approach to generate a diverse set of suitable parity-check matrices for all kinds of linear block code is so far not known.

2.1.3 Automorphism Ensemble Decoding

AED [9] has come up in recent years and can be seen as a generalization of MBBP decoding. The general idea is depicted in Figure 2.3. While in MBBP decoding, several different BP decoding instances are run in parallel, in AED, the same decoding instances are run with different inputs obtained by permutations π_i from the code's automorphism group. As in MBBP decoding, among all valid codewords found, the most likely one according to an ML criterion is chosen as output.

While the main effort in MBBP decoding lies in generating different suitable matrices, the main effort of AED lies in finding the automorphism group of the used code. So far, the automorphism groups of RM codes [9], Polar codes [10, 11, 39] and of some quasi-cyclic LDPC codes [40] have been identified and can be used for AED. In addition, researchers already implemented AED efficiently [41], highlighting its practical advantages. Some codes, however, do not have a non-trivial automorphism group [42], so AED cannot be applied. Among these codes are Long-Term Evolution (LTE) Turbo codes or non-structured LDPC codes.

In the remainder of this chapter, the last gap for applying BP decoding as universal decoding algorithm will be closed, namely applying BP to non-cyclic code classes (so they cannot be decoded using MBBP decoding) that do not possess a non-trivial automorphism group (so they cannot be decoded using AED). This yields a universal decoding algorithm that can be applied to *any* linear block code. The basis will be an MBBP-style decoding algorithm, where several different decoding instances working on different parity-check matrices of the same code are run in parallel. As the first step, in the following section, single well-performing matrices for the use in BP decoding will be constructed.

2.2 Methodology: Matrix Sparsification

As the first step, a well-performing parity-check matrix representation has to be constructed. In [43], the authors identified the following features for matrices to work well for BP decoding:

- The number of ones in each row should be small.
- The number of ones in each column should be large.
- The number of short cycles should be small.

One approach to break up short cycles was proposed by the authors of [43] as well as the author of [44] by inserting additional columns and rows into the H-matrix, which is the same as inserting additional VNs and CNs in the corresponding Tanner graph. There are two main issues with this approach: First, increasing the number of nodes increases the implementation complexity. Second, the additional VNs cannot be initialized with a channel LLR. Therefore, the results are very promising for Binary Erasure Channels (BECs), where the additional VNs can be initialized as erasure, but not for AWGN channels, where they have to be initialized with zero.

A second approach to reduce the number of cycles is via altering the parity-check matrix in a kernel-preserving way, i.e., without changing the code property (see Equation 1.7). Kernel-preserving operations can be performed in terms of elementary row operations that are commonly used in Gaussian elimination. In general, they consist of swapping rows, multiplying rows with non-zero scalars, and adding multiples of rows to other rows. Swapping rows does not affect the performance of the BP decoding algorithm for a flooding schedule, as the connections between the CNs and VNs remain the same. Since $H \in \mathbb{F}_2^{m \times n}$, one is the only non-zero scalar, so multiplications with scalars do not alter the matrix. Therefore, the only kernel-preserving operation that can be performed is the addition of a set of rows to another row. In the following, the problem of minimizing the number of cycles of a given parity-check matrix is reduced to the problem of minimizing the density of a given parity-check matrix.

2.2.1 Problem formulation

Formally, the goal is to solve the following optimization problem to minimize the number of one-entries in the matrix representation:

$$\text{(MinOnes)} \quad \min \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} H_{ij} \tag{2.11}$$

$$\text{s.t.} \quad H \cdot x \equiv 0 \mod 2 \quad \forall x \in \mathcal{C} \tag{2.12}$$

$$H \in \mathbb{F}_2^{m \times n} \tag{2.13}$$

In the objective function (2.11) the number of one-entries of H is minimized, while constraint (2.12) ensures that H is a parity-check matrix of $\mathcal{C}$. The following theorem states that (MinOnes) is an NP-hard problem.

Theorem 2.1. *The problem of finding a parity-check matrix H of a given binary linear block code $\mathcal{C}$ with a minimum number of one-entries is NP-hard.*

Proof. Recall that every code $\mathcal{C}$ has a dual code $\mathcal{C}^\perp$ with the parity-check matrix H of $\mathcal{C}$ being the generator matrix of $\mathcal{C}^\perp$, i.e., the rows of H form a basis for the dual code $\mathcal{C}^\perp$. Consider the problem (CodeDist) to compute the Hamming distance of a given code C. This problem is known to be NP-hard [45]. In the following, a polynomial reduction from (CodeDist) to (MinOnes) is derived.

Let H be a solution to (MinOnes), i.e., a basis for the dual code $\mathcal{C}^\perp$ with a minimum number of one-entries. Let p be the minimum number of one-entries inside a row of H. We'll prove by contradiction that p equals the Hamming distance of $\mathcal{C}^\perp$. Assume that the Hamming distance of $\mathcal{C}^\perp$ equals $q < p$. Then there exists a dual codeword $c' \in \mathcal{C}^\perp$ with exactly q one-entries. By the Steinitz exchange lemma, there then exists a row l of H, such that replacing row l with c' still yields a basis of $\mathcal{C}^\perp$. This is a contradiction to the assumption that H was a basis with a minimum number of one-entries.

Consequently, the problem (CodeDist) can be solved by solving (MinOnes), implying that (MinOnes) is NP-hard, too. □

2.2.2 Row-Wise IP and Heuristic Algorithm

As a first step to tackle (MinOnes), the problem is transformed into a row-wise ILP. It starts with the original parity-check matrix H and first minimizes the one-entries in the first row of H. Then, it continues with the resulting matrix and reduce the number of one-entries in the second row, repeating until all rows are considered. The procedure is shown in Algorithm 1. It is shown in [17] that this row-wise algorithm yields the same optimal result as (MinOnes). While most of the algorithm is very straight-forward, the crucial point

Algorithm 1 Row-Wise Matrix Sparsification

Require: Parity-check matrix $H \in \{0,1\}^{n-k \times n}$

1: $i = 0$
2: **for** $i = 0, \ldots, n-k-1$ **do**
3: find a subset S_i of rows that minimizes the number of one-entries in row i
4: $H_{i,}. = H_{i,}. + \sum_{i' \in S_i} H_{i',}.$
5: **return** one-minimized matrix H

lies in finding the subset S_i of rows, which need to be added to row i in order to minimize its number of one-entries.

In the following, an ILP formulation for (MinRow_i) is set up to answer the question of finding the subset S_i of row indices, which are needed to minimize the number of one-entries in row i.

The binary variables $x_{li},\ l \in \{0, \ldots, m-1\} \setminus \{i\}$ are the decision variables and the integer variables $z_{ij} \in \mathbb{Z}_{\geq 0},\ j \in \{0, \ldots, n-1\}$ are auxiliary variables. The variables x_{li} are interpreted as

$$x_{li} = \begin{cases} 1 & \text{if row index } l \text{ is added to the set } S_i \\ 0 & \text{otherwise.} \end{cases}$$

The integer program is then given by:

$$(\text{MinRow}_i) \quad \min \quad \sum_{j=0}^{n-1} (H_{ij} + (\sum_{\substack{l=0 \\ l \neq i}}^{m-1} x_{li} H_{lj}) - 2z_{ij}) \tag{2.14}$$

$$\text{s.t.} \quad H_{ij} + \sum_{\substack{l=0 \\ l \neq i}}^{m-1} x_{li} H_{lj} \geq 2z_{ij}\ \forall j \in \{0, \ldots, n-1\} \tag{2.15}$$

$$x_{li} \in \mathbb{F}_2\ \forall l \in \{0, \ldots, m-1\} \setminus \{i\} \tag{2.16}$$

$$z_{ij} \in \mathbb{Z}_{\geq 0},\ \forall j \in \{0, \ldots, n-1\} \tag{2.17}$$

The objective value is a summation over all entries of the modified row i, i.e., the original row i added with all rows corresponding to the indices in set S_i. Additionally, the factor $2z_{ij}$ is subtracted to make sure that only those feasible solutions are also optimal, which have a high value of $2z_{ij}$. This is needed due to the first constraint in Equation (2.15), which makes sure that the addition of rows is performed in the binary field $\mathbb{F}_2$: Due to the objective function, $2z_{ij}$ will be chosen as the largest possible number, thus forcing the new value in position $H_{i,j}$ to be a binary value.

Since solving (MinRow_i) is still an NP-hard problem, a time limit $t_{\max}$ for solving the ILP is added, i.e., the best-found set S_i is output when this time limit is reached. Due to this time limit, the algorithm turns into a heuristic algorithm and the result might not be

Table 2.1: Sparsified H-matrices for LTE Turbo codes

k	n	H-matrix	#1's	#4-cycles
40	132	original	1388	79848
		1-min	472	205
48	156	original	1880	149178
		1-min	568	277
56	180	original	2444	258652
		1-min	663	319
64	204	original	3084	407954
		1-min	760	383
80	252	original	4584	907630
		1-min	952	509
128	396	original	10836	5241252
		1-min	1538	884

optimal. The heuristic algorithm was applied to several LTE Turbo codes. These codes possess a very dense parity-check matrix (see [14]), which also contain large numbers of short 4-cycles. The improvements after optimization regarding the number of one-entries and the number of 4-cycles are shown in Table 2.1. The algorithm was run with Matlab and Gurobi on a single core of an Intel® i7-6700 CPU @ 3.5 GHz with time limit $t_{\max}$ set to 60 s for solving the problems (MinRow_i). It can be seen that the number of ones and number of short cycles in the matrices can be drastically reduced, promising an improved performance under BP decoding.

2.3 Methodology: Matrix Diversification

Being able to construct a single sparse matrix for the use in BP decoding, the first goal of this section is to extend the method to generate a set of sparse matrices. The second goal consists of generating a diverse subset of these matrices that complement each other when combined in MBBP decoding. This finally enables the use of MBBP decoding for all kinds of linear block codes.

Algorithm 2 Minimize Ones - Multiple Matrices

Require: Parity-check matrix $H \in \{0,1\}^{n-k \times n}$, time limit $t_{\max}$, number of matrices N

1: $i = 0$
2: **while** $i < N$ **do**
3: randomly permute the rows of H
4: call Algorithm 1 with time limit $t_{\max}$ to obtain H'_i
5: **if** H'_i is unique up to row permutations **then**
6: $H_i = H'_i$
7: $i = i + 1$
8: **return** one-minimized matrices $H_i, i = 0, \ldots, N-1$

2.3.1 Obtaining a Set of Matrices

Using Algorithm 1 we are able to find a sparse representation of a dense parity-check matrix. For applying MBBP decoding we however need a diverse set of suitable parity-check matrices. In a first step, we therefore apply Algorithm 1 with time limit $t_{\max}$ in different orders of row processing (Line 2 of Algorithm 1) to obtain different matrices. Since the time limit turns Algorithm 1 into a heuristic algorithm, its output is not exact and depends on the input. With a different order in which the rows are processed, we therefore obtain different outputs. The procedure is depicted in Algorithm 2.

After calling Algorithm 2, N matrices are obtained that are unique up to row permutations, as permuting rows results in the same performance in BP decoding (see Section 2.2).

2.3.2 Obtaining a Diverse Subset of Matrices

In a second step, out of these N matrices, a set of l matrices has to be selected, which are used for decoding in MBBP decoding. The goal is to not only select "good" matrices under BP decoding, but to select several matrices that complement each other best. This problem can be viewed as a *Maximum Coverage Problem* [46]:

$$\text{Given: A set of sets } S = \{S_0, \ldots, S_{N-1}\} \text{ and a natural number } l \leq N \tag{2.18}$$

$$\text{Goal: A subset } S' \subseteq S \text{ of size } l \text{ such that } \left| \bigcup_{S_i \in S'} S_i \right| \text{ is maximized} \tag{2.19}$$

In the case of finding a diverse set of matrices, this translates to the following: For obtaining the sets S_i, we performed test simulations (1000 frames at an SNR of 2 dB) for every decoding instance $i \in \{0, \ldots, N-1\}$ corresponding to the N different matrices. The sets S_i then represent the correctly decoded frames for this decoding instance, so $S_i \subseteq \{0, \ldots, 999\}$. Out of those N sets, the goal is to select l sets such that the number of correctly decoded

Algorithm 3 Maximum-Coverage Heuristic

Require: Sets $S_i, i = 0, \ldots, N-1$, integer l

1: Find $j \in \{0, \ldots, N-1\}$ such that $|S_j|$ is maximal
2: $L = [j]$
3: **for** $i = 0, \ldots, l-1$ **do**
4: Find $j \in \{0, \ldots, N-1\}$ such that $\left|\bigcup_{k=0}^{|L|-1} S_{L(k)} \cup S_j\right|$ is maximized
5: L.append $[j]$
6: **return** list of set indices L

frames in MBBP decoding is maximized. As the maximum coverage problem is an NP-hard problem, we use a straight-forward heuristic that successively adds the set to the output, which complements the previously chosen sets most [46]. The procedure is formalized in Algorithm 3.

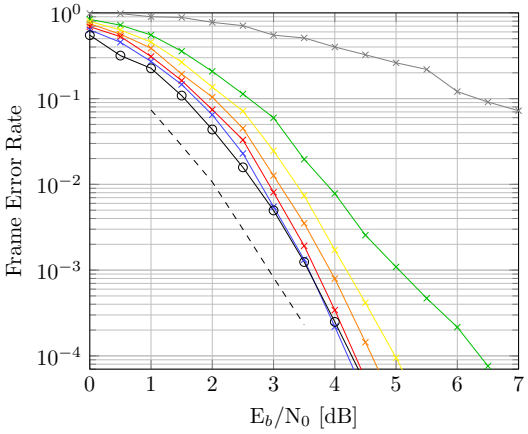
2.4 Results

In this section, the methodology is validated for LTE Turbo codes. As previously mentioned, LTE Turbo codes are non-cyclic codes that only possess a trivial automorphism group. Therefore, they are a code class that could not yet be decoded using AED or MBBP decoding.

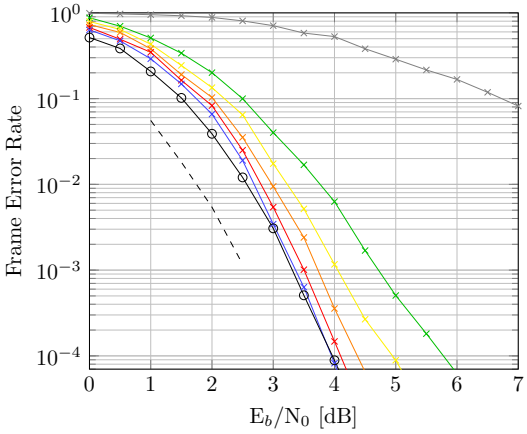
2.4.1 FER Performance of LTE Turbo Codes

In this section, the FER performance under Ensemble BP decoding is investigated and compared to the Turbo code's state-of-the-art decoding algorithm. To obtain the results, first, Algorithm 2 is run with $N = 25$ to create 25 different sparse parity-check matrices. Second, Algorithm 3 is applied to select $l = 2, 4, 8, 16$ matrices out of these $N = 25$ matrices, respectively, to be combined in Ensemble BP decoding.

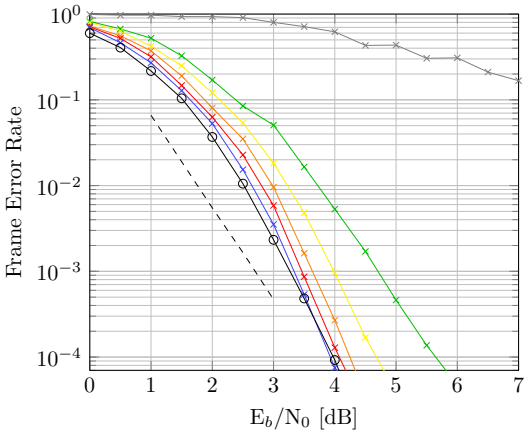
Figure 2.4 shows the error-correcting performance of Ensemble BP decoding with the matrices obtained in the previous section. All simulations were carried out as Monte-Carlo simulations over an AWGN channel using BPSK modulation with a minimum of 100 error frames. The BP decoding algorithm was run with min-sum approximation for a maximum of 40 iterations, an ESF of 0.875 and floating point precision. The gray curve shows the performance of the BP algorithm on the original parity-check matrix obtained from [14], the green curve the performance of the BP algorithm on the best one-minimized matrix obtained by setting $l = 1$ in Algorithm 3. The yellow, orange, red and blue curves show the performance under Ensemble BP decoding setting $l = 2, 4, 8, 16$, respectively. For reference, the black curve depicts the resulting performance of the Turbo code's dedicated



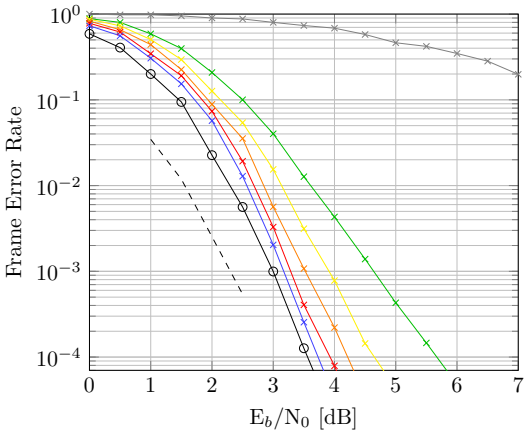
(a) $k = 40$ Turbo Code



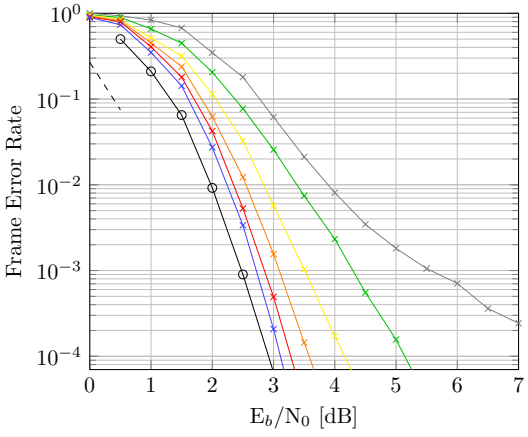
(b) $k = 48$ Turbo Code



(c) $k = 56$ Turbo Code



(d) $k = 64$ Turbo Code



(e) $k = 128$ Turbo Code

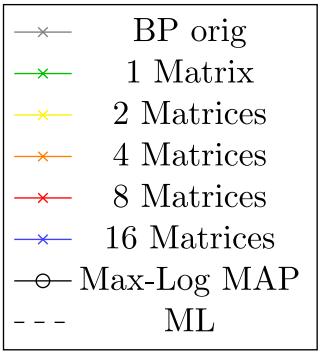


Figure 2.4: Error-Correcting Performance of Short LTE Turbo Codes Under the Proposed Ensemble BP

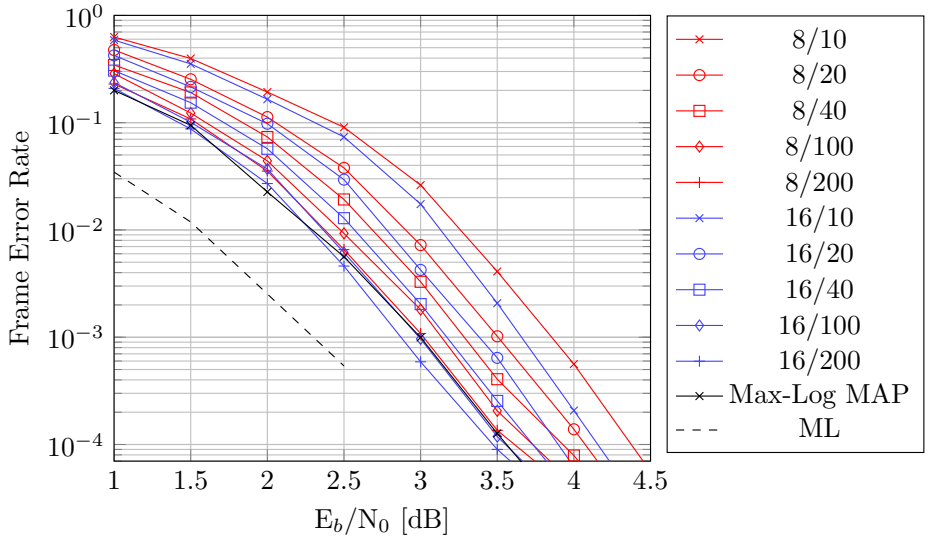


Figure 2.5: FER of $k = 64$ Turbo code. The notation x/y means that x different matrices are taken for Ensemble BP, each with a maximum number of iterations y.

decoding algorithm, the max-Log Maximum a posteriori (MAP) algorithm, for serial processing of 14 turbo half-iterations and an ESF of 0.75[1]. The dotted ML curve was taken from [14].

It can be observed that moving from the original Turbo code parity-check matrix to a sparsified one improves the FER performance of BP decoding immensely, which is due to the drastically decreased number of one-entries and short cycles in the optimized parity-check matrix (see Table 2.1). The performance improves the more matrices are taken for Ensemble BP decoding, reaching the max-Log MAP performance at $l = 16$ matrices for almost all considered blocklengths. Although the Ensemble BP does not outperform the max-Log MAP in terms of FER performance, it should be noted that a respective hardware implementation of the parallel Ensemble BP will have a much lower latency than the max-Log MAP due to the parallel nature of the decoding algorithm.

2.4.2 Convergence Analysis

The performance of Ensemble BP decoding for Turbo codes for different maximal iterations is depicted in Figure 2.5 for the length $k = 64$ Turbo code. The notation x/y means that x different matrices are taken for Ensemble BP, each with a maximum number of iterations y. It can be seen that the code ensembles only converge slowly, which is most likely due to the short cycles still existent in the code. To tackle this issue, a weighted BP

[1]The max-Log MAP results for $k = 128$ were obtained from [47] with a fully pipelined iteration unrolled XMAP architecture, a quantization of 6 bits for the channel LLRs, and 8 turbo half-iterations.

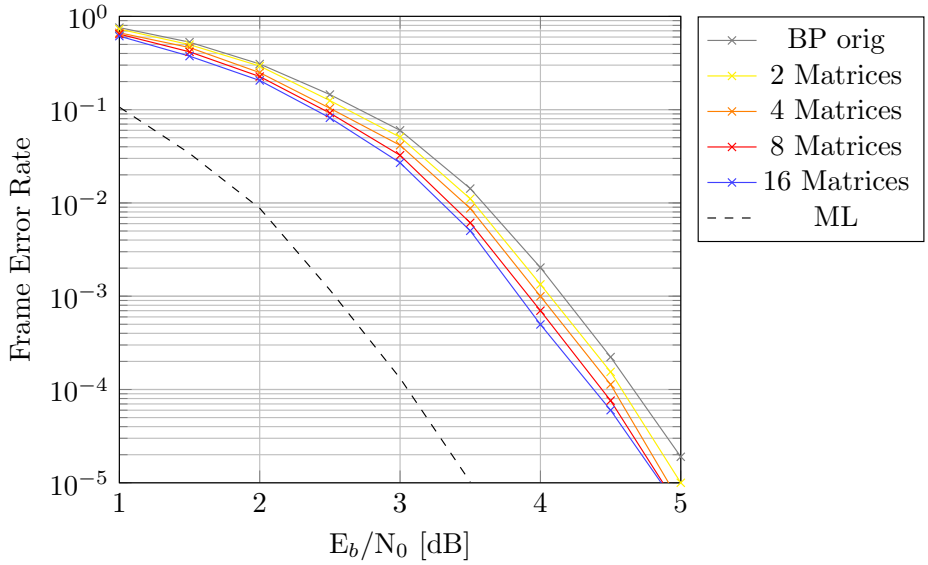


Figure 2.6: Ensemble BP for the $(128, 64)$ Consultative Committee for Space Data Systems (CCSDS) Code

approach as proposed in [38] could be beneficial to compensate for the short cycles still existent.

2.4.3 FER Performance of CCSDS LDPC Code

In Figure 2.6 the FER performance of the $(128, 64)$ CCSDS LDPC code [14, 48] under Ensemble BP decoding for different l is depicted. As the code's parity check matrix already has maximum sparsity, applying Algorithm 2 will always output the same matrix. Therefore, the original matrix were first randomly densified by performing random row operations and then applied Algorithm 2 to obtain different matrices for the $(128, 64)$ CCSDS code that contain up to 15% more one-entries than the original matrix. As most of the frames were already correctly decoded using the original matrix, the additional decoder instances were only considered if the original one failed. It can be seen that the FER improvement under Ensemble BP decoding is around 0.5 dB at an FER of 10^{-3}.

Chapter 3

Linear Programming Decoding

LP decoding has come up in recent years as a promising approach to obtain a close-to-ML decoding performance with affordable complexity. Instead of solving the NP-hard ML decoding problem, a linear relaxation is solved, which can be done in polynomial time. In addition, solving this LP relaxation is an important step when aiming to solve the ML decoding problem, as will be shown in Chapter 4. Thus, lowering the implementation complexity of LP decoding is of great interest.

This chapter makes the following main contribution:

- A new projection algorithm for ADMM-based LP decoding is presented. This projection algorithm relies on a recursive structure of the parity polytopes and significantly lowers the implementation complexity.

The contribution of this chapter was published in [18, 19]. The work was carried out in cooperation with and under the lead of the Optimization Research Group of the University of Kaiserslautern-Landau.

3.1 Background and State of the Art

The LP-relaxation of the ML decoding problem is given by

$$\begin{aligned} \min \quad & \lambda^\top x \\ \text{s.t.} \quad & H \cdot x = 0 \pmod 2. \end{aligned} \tag{3.1}$$

Note that in contrast to the ML decoding ILP (Equation 1.9) the integrality condition is omitted. Even though it is no longer ensured to find the ML solution, the problem is no longer NP-hard and can be solved in polynomial time by LP solvers. The idea behind this relaxation is that, if the LP solver finds an integral solution, this solution will be the ML solution. On the other hand, if the LP solver finds a fractional solution, the hope is that this fractional solution will be "close" to the ML solution. In the following, the ADMM algorithm for solving the LP decoding problem will be recapped.

3.1.1 ADMM-based LP Decoding

The ADMM is an iterative method from convex optimization, which has shown to perform well as an LP solver, especially in case of a sparse parity-check matrix [49]. It was first mentioned by Boyd et al. in [12] and first applied to the LP decoding problem by Barman et al. [50].

To fit the LP decoding problem for the ADMM, it first has to be rewritten to

$$\begin{aligned} \min \quad & \lambda^\top x \\ \text{s.t.} \quad & T_j \cdot x = z_j \\ & z_j \in \mathcal{P}_{d_j},\ \forall j \in \{0, \dots, m-1\}, \end{aligned} \tag{3.2}$$

as described in [13]. The transfer matrix $T_j \in \mathbf{F}_2^{d_j \times n}$ is derived from the parity-check matrix H by inserting the corresponding unit vector as a row for every neighboring variable of check j. For instance, if $H_{.,j} = (0,1,1,0,1,0)$, then

$$T_j = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.3}$$

Multiplying some n-dimensional vector x to T_j thus selects exactly those components from x, which correspond to positions relevant for check j.

To solve the LP problem from Equation 3.2 using the ADMM, the following updates have to be performed in iteration k:

$$\begin{aligned} x^{k+1} &= \arg\min_x \left(\lambda^\top x + \frac{\rho}{2} \sum_{j \in J} \left\| T_j x - z_j^k + u_j^k \right\|_2^2 \right), \\ z_j^{k+1} &= \Pi_{\mathcal{P}_{d_j}} \left(T_j x^{k+1} + u_j^k \right) \quad \forall j \in \{0, \dots, m-1\}, \\ u_j^{k+1} &= u_j^k + T_j x^{k+1} - z_j^{k+1} \quad \forall j \in \{0, \dots, m-1\}. \end{aligned} \tag{3.4}$$

ρ denotes a so-called penalty parameter, which has to be chosen properly to speed up the algorithm's convergence, and the notation $\Pi_{\mathcal{P}_{d_j}}(\,\cdot\,)$ defines the Euclidean projection onto the parity polytope $\mathcal{P}_{d_j}$ defined as

$$\mathcal{P}_{d_j} = \operatorname{conv} \left\{ x \in \{0,1\}^{d_j} \mid \sum_{i=0}^{d_j - 1} x_i \text{ is even} \right\}. \tag{3.5}$$

From the update rules in Equation 3.4, a message-passing algorithm can be derived. It performs on the code's Tanner graph, but with more complex node update rules than the BP algorithm. The algorithm is recalled from [13] in Algorithm 4.

The most complex part of this algorithm lies in the check node update, where the projection $\Pi_{P_{d_j}}(\,\cdot\,)$ onto the parity polytope is calculated. In [13], the authors present a projection

Algorithm 4 Message-Passing ADMM-based LP Decoding

1: **Initialize** $y_j = 0 \quad \forall j$,
$x_i = \begin{cases} 1, & \lambda_i < 0, \\ 0, & \lambda_i \geq 0. \end{cases} \forall i$
2: **Check Node Update** Update
$w = T_j x + y_j, \quad z_j = \Pi_{\mathcal{P}_{d_j}}(w), \quad y_j = w - z.$
The messages transmitted from check node j to variable node i are calculated as $L_{j\to i} = (z_j)_i - (y_j)_i$.
3: **Variable Node Update** Update
$x_i = \frac{1}{d_j}\left(\sum_{j' \in \mathcal{M}(i)} L_{j'\to i} - \frac{\lambda_i}{\rho}\right)$.
4: **Stopping Criterion** Stop the algorithm if $\sum_{j\in J}\|T_j x - z_j\|_2 < \varepsilon^{\text{pri}}$ and $\sum_{j\in J}\|z_j^k - z_j^{k-1}\|_2 < \varepsilon^{\text{dual}}$, where $\varepsilon^{\text{pri}} > 0$ and $\varepsilon^{\text{dual}} > 0$ are chosen tolerances.

algorithm based on the so-called cut-search algorithm (CSA), originally introduced in [51] to identify forbidden-set inequalities. Here, it is used to determine whether a given vector lies inside the parity polytope $\mathcal{P}_d$ or not. Using this cut-search algorithm, calculating the projection onto the parity polytope is reduced to multiple projections onto the unit hypercube and a sorting operations. The authors of [52–54] reduce the projection onto the parity polytope to the projection onto a simplex, which, again, requires a sorting [52,53] or partial sorting [54] operation. The authors of [55] presented an iterative and sorting-free method for calculating the projection, which is, however, only approximate.

In the following, a new exact projection algorithm for ADMM-based LP decoding is derived, which does not require any sorting operation.

3.2 New Projection Algorithm for ADMM-based LP Decoding

The idea of the new projection algorithm is to fix components of the vector to be projected in an iterative manner. This is possible due to the iterative structure of the parity polytope, which will be derived in the following.

3.2.1 Parity Polytope Structure

In order to establish the recursive structure of the parity polytope, we will first define the so-called odd parity polytope. In the following, the parity polytope defined in Equation 3.5, P_d, will be denoted as $P_{d,\text{even}}$. Accordingly, the odd parity polytope is defined as

$$\mathcal{P}_{d,\text{odd}} := \text{conv}\left\{ x \in \{0,1\}^d : \sum_{i=0}^{d-1} x_i \text{ is odd} \right\}. \tag{3.6}$$

It is shown in [18] that properties of $P_{d,\text{even}}$ hold similarly for $P_{d,\text{odd}}$ with exchanged roles of "even" and "odd".

The following theorem establishes the recursive structure of the parity polytope.

Theorem 3.1 ([19], Theorem 6)**.** *Let $d \geq 2$ and $i \in \{0, \ldots, d-1\}$. Then it holds that*

i) $\{x \in \mathcal{P}_{d,\text{even}} : x_i = 1\} = \left\{(\widetilde{x}_0, \ldots, \widetilde{x}_{i-1}, 1, \widetilde{x}_i, \ldots, \widetilde{x}_{d-2})^\top \in \mathbb{R}^d : \widetilde{x} \in \mathcal{P}_{d-1,\text{odd}}\right\}$

ii) $\{x \in \mathcal{P}_{d,\text{even}} : x_i = 0\} = \left\{(\widetilde{x}_0, \ldots, \widetilde{x}_{i-1}, 0, \widetilde{x}_i, \ldots, \widetilde{x}_{d-2})^\top \in \mathbb{R}^d : \widetilde{x} \in \mathcal{P}_{d-1,\text{even}}\right\}$

iii) $\{x \in \mathcal{P}_{d,\text{odd}} : x_i = 1\} = \left\{(\widetilde{x}_0, \ldots, \widetilde{x}_{i-1}, 1, \widetilde{x}_i, \ldots, \widetilde{x}_{d-2})^\top \in \mathbb{R}^d : \widetilde{x} \in \mathcal{P}_{d-1,\text{even}}\right\}$

iv) $\{x \in \mathcal{P}_{d,\text{odd}} : x_i = 0\} = \left\{(\widetilde{x}_0, \ldots, \widetilde{x}_{i-1}, 0, \widetilde{x}_i, \ldots, \widetilde{x}_{d-2})^\top \in \mathbb{R}^d : \widetilde{x} \in \mathcal{P}_{d-1,\text{odd}}\right\}$

With the structure revealed by the above theorem, the idea of the new projection algorithm becomes clear. By fixing single components of the vector x to be projected, the problem of projecting the remaining components can again be described as a projection onto a smaller-dimensional even or odd parity polytope.

Next, the question remains how to find components that can be fixed to 0 or 1. According to Theorem 3 of [13], the projection onto the parity polytope lies on the intersection of the unit hypercube with the hyperplane defined by the forbidden-set inequality. The following theorem shows that exactly those components can be fixed, which move the vector to be projected into the feasible halfspace of the forbidden-set inequality.

Theorem 3.2 ([19], Theorem 4)**.** *Let $x \in \mathbb{R}^d$ with $d \geq 2$. Let*

$$\theta^\top w = \sum_{j \in V} w_j - \sum_{j \in \{0,\ldots,d-1\} \setminus V} w_j \leq |V| - 1$$

with $V \subseteq \{0, \ldots, d-1\}$ ($|V|$ even or odd) be a forbidden-set inequality. Let $v = \Pi_H(x)$, be the projection of x onto the hyperplane $H := \left\{w \in \mathbb{R}^d : \theta^\top w = |V| - 1\right\}$ and let $z = \Pi_F(x)$ be the projection of x onto the face $F := \left\{w \in [0,1]^d : \theta^\top w = |V| - 1\right\}$. Let $i \in \{0, \ldots, d-1\}$. Then it holds:

i) If $v_i > 1$ and $\theta_i = 1$, then $z_i = 1$.

ii) If $v_i < 0$ and $\theta_i = -1$, then $z_i = 0$.

According to Theorem 5 from [19] there will always be at least one component, which fulfills the requirements of Theorem 3.2 and can thus be fixed to 0 or 1.

To further simplify the projection algorithm, last but not least, the following theorem shows that there is no need to repeatedly run the CSA from scratch. In detail, the following theorem shows how the result of the CSA changes whenever a component of the vector x to be projected is fixed.

Theorem 3.3 ([19], Theorem 8)**.** *Let* $x = (\widetilde{x}_0, \ldots, \widetilde{x}_{i-1}, \beta, \widetilde{x}_i, \ldots, \widetilde{x}_{d-2})^\top \in \mathbb{R}^d$ *with* $d \geq 2$*.*

i) Let $(\theta, p) \in \mathbb{R}^{d+1}$ *be the forbidden-set inequality returned by CSA applied to* x *and* $\mathcal{P}_{d,\text{even}}$*. Let* $\theta_i = 1$*. Then* $\left((\theta_0, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_{d-1})^\top, p-1\right)$ *is the output of CSA applied to* $\widetilde{x}$ *and* $\mathcal{P}_{d-1,\text{odd}}$*.*

ii) Let $(\theta, p) \in \mathbb{R}^{d+1}$ *be the forbidden-set inequality returned by CSA applied to* x *and* $\mathcal{P}_{d,\text{odd}}$*. Let* $\theta_i = 1$*. Then* $\left((\theta_0, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_{d-1})^\top, p-1\right)$ *is the output of CSA applied to* $\widetilde{x}$ *and* $\mathcal{P}_{d-1,\text{even}}$*.*

iii) Let $(\theta, p) \in \mathbb{R}^{d+1}$ *be the forbidden-set inequality returned by CSA applied to* x *and* $\mathcal{P}_{d,\text{even}}$*. Let* $\theta_i = -1$*. Then* $\left((\theta_0, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_{d-1})^\top, p\right)$ *is the output of CSA applied to* $\widetilde{x}$ *and* $\mathcal{P}_{d-1,\text{even}}$*.*

iv) Let $(\theta, p) \in \mathbb{R}^{d+1}$ *be the forbidden-set inequality returned by CSA applied to* x *and* $\mathcal{P}_{d,\text{odd}}$*. Let* $\theta_i = -1$*. Then* $\left((\theta_0, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_{d-1})^\top, p\right)$ *is the output of CSA applied to* $\widetilde{x}$ *and* $\mathcal{P}_{d-1,\text{odd}}$*.*

With this theoretical background, the new projection algorithm can be stated.

3.2.2 New Projection Algorithm

The whole algorithm is described in Algorithm 5.

First, in lines 1 – 5, the CSA is applied to the vector x to be projected. Next, in lines 6 – 8, it is checked whether $u = \Pi_{[0,1]^d}(x)$ already lies on the parity polytope. If yes, u is returned as the projection onto the parity polytope. Otherwise, the recursive part of the algorithm is entered, where single components are successively fixed to 0 or 1 according to Theorem 3.2. Before entering the recursive part, a set of variables is initialized: l tracks how many components of x are still un-fixed, f and f_{old} track the beginning of the yet un-fixed part of the vector x and Q keeps track of the swappings. Note that, for better readability, the algorithm is written in such a way that already fixed components are moved to the front of x. Therefore, in the later recursions, only the tail of x, denoted by $x_{f,\ldots,d-1}$, is considered. Upon entering the recursion, lines 11 – 16 catch the special case that only one component of x is yet un-fixed and finish the algorithm. Line 17 computes the projection of x (or the remaining components of x) onto the hyperplane defined by the forbidden-set inequality. In lines 18 – 25, the single components of x, which meet the requirements of Theorem 3.2. Lines 20 and 24, respectively, perform the swapping of the found component to the front of the remaining vector x and update f accordingly. Lines 21 and 25, respectively, fix the component in the output vector z. Line 22 updates the value p for the CSA according to Theorem 3.3. If no component was found to be fixed in the current recursion then the

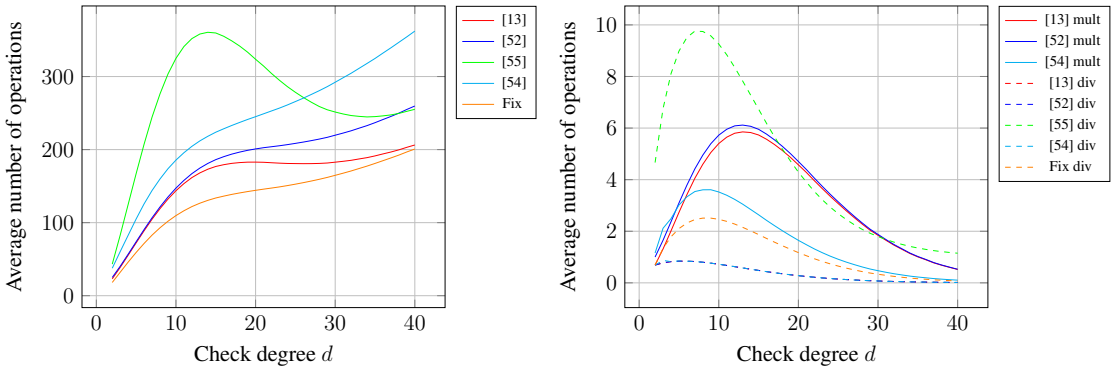
Algorithm 5 Projection Algorithm

Require: $x \in \mathbb{R}^d$, Output: $z = \Pi_{\mathcal{P}_d,\text{even}}(x)$

1: $\theta_i \leftarrow \text{sgn}(x_i - \frac{1}{2}) \quad \forall i = 0, \dots, d-1$
2: **if** $|\{i : \theta_i = 1\}|$ is even **then**
3: $\quad i^* \leftarrow \arg\min_i |x_i - \frac{1}{2}|$
4: $\quad \theta_{i^*} \leftarrow -\theta_{i^*}$
5: $p \leftarrow |\{i : \theta_i = 1\}| - 1$
6: $u \leftarrow \Pi_{[0,1]^d}(x)$
7: **if** $\theta^\top u \leq p$ **then**
8: $\quad$ **return** u
9: $l \leftarrow d$, $f \leftarrow 0$, $f_{\text{old}} \leftarrow 0$, $Q \leftarrow [0, \dots, d-1]$
10: **while** True **do**
11: $\quad$ **if** $l = 1$ **then**
12: $\quad\quad$ **if** $\theta_f = 1$ **then**
13: $\quad\quad\quad z_f \leftarrow 0$
14: $\quad\quad$ **else**
15: $\quad\quad\quad z_f \leftarrow 1$
16: $\quad\quad$ **break**
17: $\quad v_{f,\dots,d-1} \leftarrow x_{f,\dots,d-1} - \frac{\left(\theta^\top_{f,\dots,d-1} \cdot x_{f,\dots,d-1}\right) - p}{l} \theta_{f,\dots,d-1}$
18: $\quad$ **for** $i = f_{\text{old}}, \dots, d-1$ **do**
19: $\quad\quad$ **if** $v_i > 1$ and $\theta_i = 1$ **then**
20: $\quad\quad\quad$ swap Q_f and Q_i, $x_i \leftarrow x_f$, $\theta_i \leftarrow \theta_f$, $f \leftarrow f + 1$
21: $\quad\quad\quad z_f \leftarrow 1$
22: $\quad\quad\quad p \leftarrow p - 1$
23: $\quad\quad$ **else if** $v_i < 0$ and $\theta_i = -1$ **then**
24: $\quad\quad\quad$ swap Q_f and Q_i, $x_i \leftarrow x_f$, $\theta_i \leftarrow \theta_f$, $f \leftarrow f + 1$
25: $\quad\quad\quad z_f \leftarrow 0$
26: $\quad$ **if** $f_{\text{old}} = f$ **then**
27: $\quad\quad z_{f_{\text{old}},\dots,d-1} \leftarrow v_{f_{\text{old}},\dots,d-1}$
28: $\quad\quad$ **break**
29: $\quad l \leftarrow d - f$, $f_{\text{old}} \leftarrow f$
30: **return** $z[Q]$

algorithm is terminated (lines 26 – 28) finished. Line 29 updates the variables for the next recursion. Upon termination, the output z is swapped back to its original order according to the swapping vector Q.

As an observation based on Monte-Carlo simulations, the algorithm requires around $\log_2(d)$ iterations until the projection is fully computed. This is much faster than the ex-



(a) Low-complexity operations **(b)** Multiplications and divisions

Figure 3.1: Comparison of arithmetical operations for different projection methods with random input from $[-2, 2)^d$

pected worst case of $\mathcal{O}(d^2)$ iterations, where only one component would be fixed per iteration.

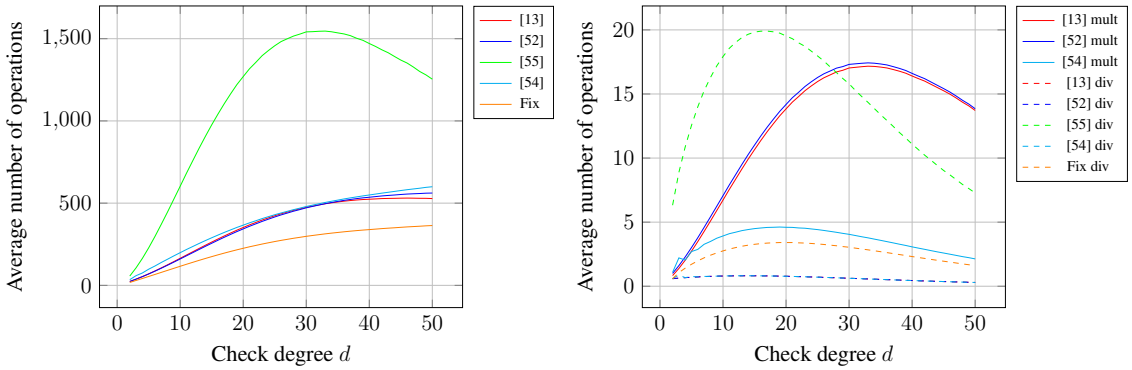
3.3 Results

To evaluate the performance benefit of the new projection algorithm, several performance indicators were measured and compared to the state-of-the-art projection algorithms by Zhang and Siegel [13], Wasson and Draper [52], Zhang et al. [54], and Wei and Banihashemi [55].

Instead of simulating the algorithm's runtime in software, the number of arithmetic operations was counted to estimate the algorithm's complexity in a hardware implementation. It is differentiated between low-complexity arithmetic operations, including comparisons, additions, subtractions and negations, multiplications and divisions. To make a fair comparison, all considered algorithms were optimized beforehand to reduce the number of high-complexity arithmetic operations. The number of arithmetic operations is averaged over 10^6 Monte-Carlo simulations for every value of d with inputs uniformly distributed from $[-2, 2)^d$, $[-5, 5)^d$, and $[-10, 10)^d$, respectively, where d varies from 2 to 40 for inputs from $[-2, 2)^d$ and from 2 to 50 for inputs from $[-5, 5)^d$ and $[-10, 10)^d$.

The results are depicted in Figures 3.1a to 3.3a for the averaged low-complexity arithmetic operations and in Figures 3.1b to 3.3b for the averaged multiplications and divisions.

Considering the low-complexity arithmetic operations, the average number of operations increases with an increasing value of d. However, there is a peak, which can be well observed in the green curves corresponding to the algorithm of [55]. This behavior was further examined and we found that for increasing d, the probability that $\Pi_{[0,1]^d}(x) \in \mathcal{P}_{d,\text{even}}$



(a) Low-complexity operations

(b) Multiplications and divisions

Figure 3.2: Comparison of arithmetical operations for different projection methods with random input from $[-5, 5)^d$

increases, thus, the algorithm is finished before the recursion starts (see Lines 6 – 8 in Algorithm 5). The new projection algorithm, denoted by "Fix" achieves the lowest number of low-complexity arithmetic operations over all considered input ranges and check degrees.

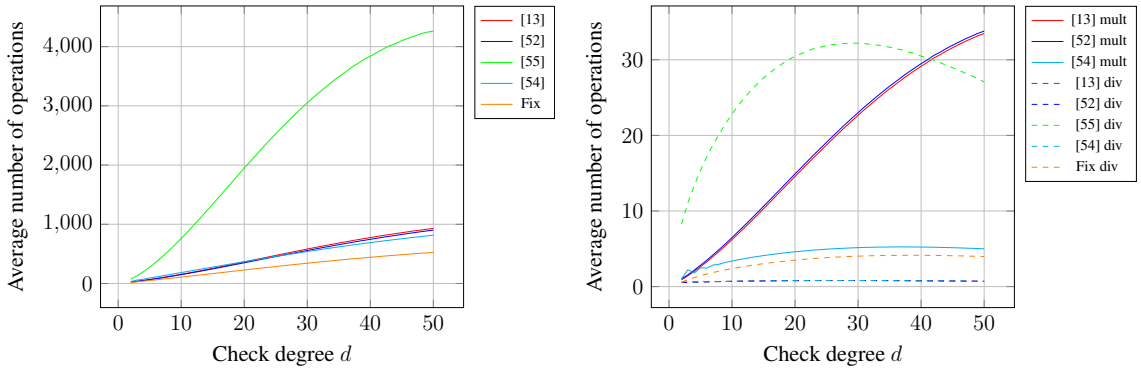
Table 3.1: Maximum Gain

input	low-complexity operations	all arith. op.
$[-2, 2)^d$	-24%	-26%
$[-5, 5)^d$	-36%	-37%
$[-10, 10)^d$	-37%	-37%

Considering the multiplications and divisions, first, it should be noted that "Fix" and the algorithm from [55] do not require any multiplications. Therefore, the corresponding curves are omitted. As in the case of low-complexity operations, a "peak" behavior can be observed for the multiplications and divisions at a certain value of d. Even though "Fix" requires slightly more divisions than the algorithms from [13], [52] and [54], it requires significantly less multiplications. Thus, when considering high-complexity arithmetic operations as the sum of multiplications and divisions, "Fix" achieves the lowest number of high-complexity arithmetic operations.

The gains for the low-complexity operations and for all arithmetic operations are summarized in Table 3.1. These gains are computed as the maximum gain over the different values of d for all considered ranges. As a baseline, the best algorithm from the other four considered algorithms is taken. Overall, "Fix" needs up to 37% less arithmetic operations, underlining the benefit of the new projection algorithm and making it attractive for future efficient hardware implementations.

To highlight the benefit of the new projection algorithm in a real-world example, the projection algorithm was applied in the scenario of ADMM-based LP decoding of two



(a) Low-complexity operations

(b) Multiplications and divisions

Figure 3.3: Comparison of arithmetical operations for different projection methods with random input from $[-10, 10)^d$

different WiMax LDPC codes. The two codes considered were the $(576, 384)$ WiMax code and the $(576, 480)$ WiMax code obtained from [14]. For the $(576, 384)$ WiMax code, only projections of the rows with check degrees $d = 10$ were considered. The $(576, 480)$ WiMax code has a regular check degree of $d = 20$ for all rows. The input values were achieved by simulating a transmission over a AWGN channel with an SNR of 2. The results shown in Table 3.2 are in line with the previous results and underline the practical benefit of the algorithm.

Table 3.2: Average Gain for Different WiMax Codes @SNR=2

input	low-complexity operations	all arith. op.
$(576, 384)$ WiMax	-23%	-25%
$(576, 480)$ WiMax	-34%	-35%

Chapter 4

ML Decoding

ML decoding is the champions league of all decoding algorithms. Instead of solving the decoding problem heuristically, it is solved optimally and therefore provides the best possible communications performance among all decoding algorithms. On the downside, ML decoding is an NP-hard problem and can therefore not be performed efficiently. Knowing the ML performance of different coding schemes is still very valuable for the following reasons:

- *Evaluation of Code Potential:* As ML decoding is applicable to any coding scheme, it is a powerful tool to compare different coding schemes without the bias of a heuristic decoding algorithm.
- *Evaluation of Decoder Potential:* The ML decoding performance of a specific coding scheme serves as a baseline for any other decoding scheme. It is therefore very beneficial in estimating the quality of a heuristic decoding algorithm.
- *Decoding for Real Applications:* ML decoding can be applied in applications with very loose requirements on the implementation-specific KPIs or in applications with very short blocklengths.

Thus, even though ML decoding is an NP-hard problem, it is still worth to put a lot of effort in. To obtain the ML decoding performance of a specific code, only one ML Monte-Carlo simulation has to be performed, which usually involves between 10^5-10^9 decoding runs for every SNR of interest. As of today, calculating the ML decoding performance of a code is possible for blocklengths of up to 500-1000 bits [14].

This chapter makes the following contributions:

- The new projection algorithm presented in Chapter 3 is incorporated into the Branch-and-Bound (B&B) framework for ML decoding. It allows for an efficient update of components fixed in the B&B directly into the ADMM's projection algorithm. Thereby, as the B&B search tree is traversed, the problem size of the projection algorithm is reduced.

- The matrix sparsification algorithm presented in Chapter 2 is used to largely speed up the ML decoding algorithm, thereby enabling the calculation the previously unknown ML performance of the $(155, 125)$ Reed-Solomon (RS) code. Calculated ML performance curves are provided in our Channel Codes Database (see [14]), serving as valuable benchmarks for researchers in the channel coding community.

The contributions of this chapter were published in [17, 20]. The publication [17] was carried out in cooperation with the Optimization Research Group of the University of Kaiserslautern-Landau.

4.1 Branch-and-Bound Method

Recall from Equation 1.9 that ML decoding is equivalent to solving the following ILP [27]:

$$\begin{aligned} z \quad = \quad & \min_{x \in \mathbb{F}_2^n} \quad \lambda^\top \cdot x \\ & \text{s.t.} \quad H \cdot x \equiv 0 \pmod 2 \end{aligned} \tag{4.1}$$

In optimization theory, integer programs are oftentimes solved by B&B algorithms. These algorithms first calculate upper and lower bounds $z_{\mathrm{LB}} \leq z \leq z_{\mathrm{UB}}$ for the optimal objective value z. Upper bounds $z_{\mathrm{UB}} = \lambda^\top \widetilde{x}$ are induced by valid codewords $\widetilde{x} \in \mathcal{C}$, which can be obtained by using heuristic decoders like ensemble BP presented in Chapter 2. Lower bounds are computed solving a relaxation of 4.1. In the case of ML decoding, this relaxation leads to the LP decoding problem

$$\begin{aligned} \min \quad & \lambda^\top \cdot x \\ \text{s.t.} \quad & H \cdot x \equiv 0 \pmod 2, \end{aligned} \tag{4.2}$$

where the integrality condition is omitted (see 3.1). This LP can efficiently be solved using ADMM-based LP decoding with the improved projection algorithm presented in Chapter 3.

Second, in the branching step, the original ML decoding problem is subdivided into several subproblems by partitioning the solution set. A natural branching strategy for binary ML decoding is to fix one component $x_{i'}$ to zero or one, thus inducing two subproblems and spanning a binary tree, the so-called B&B tree. Note that the problem dimension for these two subproblems reduces by 1. Therefore, the problem dimension is successively reduced the deeper we modve in the B&B tree. For every node in this tree, new lower and upper bounds are computed to repeatedly improve the bounds of the original problem, called bounding. These branching and bounding steps are repeated until $z_{\mathrm{LB}} = z = z_{\mathrm{UB}}$. The effectiveness of the B&B algorithm relies on the fact that good bounds allow to prune large parts of the B&B tree, i.e., the set of feasible solutions.

There are three main pruning rules that allow to remove large parts of the B&B tree, which is created in the branching steps:

- *Pruning by Infeasibility:* A subproblem can be pruned if its feasible set is empty, i.e., there exists no valid solution in the remaining subtree.

- *Pruning by Optimality:* A subproblem can be pruned if it is solved optimally, i.e., its bounds fulfill $z'_{\mathrm{LB}} = z'_{\mathrm{UB}}$. This is typically the case if the solution of the linear relaxation is integral.

- *Pruning by Bound:* A subproblem can be pruned if the lower bound of the subproblem is larger than the upper bound of the overall problem, i.e., the optimal solution cannot be found in the remaining subtree.

An example of such a tree traversal is given in Figure 4.1. Assuming that the tree is traversed depth-first, the following steps are taken:

1. Upper and lower bounds are computed for the root node, $z^{(\emptyset)}$, resulting in $z_{\mathrm{UB}}^{(\emptyset)} = 0$ and $z_{\mathrm{LB}}^{(\emptyset)} = -15$. As none of the pruning rules are applicable, a branching is performed by fixing the first component of x to 0 and 1, respectively, thereby creating two child nodes.

2. For the first child node $z^{(0)}$ ($x_0 = 0$), upper and lower bounds are calculated, resulting in $z_{\mathrm{UB}}^{(0)} = -5$ and $z_{\mathrm{LB}}^{(0)} = -10$. Still no pruning rule can be applied, so a branch is performed by fixing the second component of x to 0 and 1, respectively.

3. For the left child node $z^{(00)}$ ($x_0 = x_1 = 0$), the computation of the lower bound resulted in the identification of an unsolvable problem, indicating that the problem is unfeasible (i.e., there exists no valid solution with $x_0 = x_1 = 0$). Thus, the remaining subtree can be pruned by infeasibility.

4. For the right child node $z^{(01)}$ ($x_0 = 0$, $x_1 = 1$), the computation of the lower bound resulted in a valid codeword with objective value of $z_{\mathrm{LB}}^{(01)} = -10 = z_{\mathrm{UB}}^{(01)}$. The remaining subtree can thus be pruned by optimality.

5. The only node left unconsidered is the rightmost node $z^{(1)}$ ($x_0 = 1$). From the valid codeword found in the previous step, the upper bound of -10 carries over to this node. Calculating the lower bound results in $z_{\mathrm{LB}}^{(1)} = -5$. Since $z_{\mathrm{UB}}^{(1)} < z_{\mathrm{LB}}^{(1)}$, the remaining subtree can be pruned by bound. As there is no more node unvisited, the B&B terminates and the best found solution from node $z^{(01)}$ is output.

There are many choices to be made to adjust the B&B algorithm and they can have a heavy impact on the algorithm's performance, so they have to be investigated for every use-case and trade off well. The most impactful options are depicted in the B&B design space in Figure 4.2:

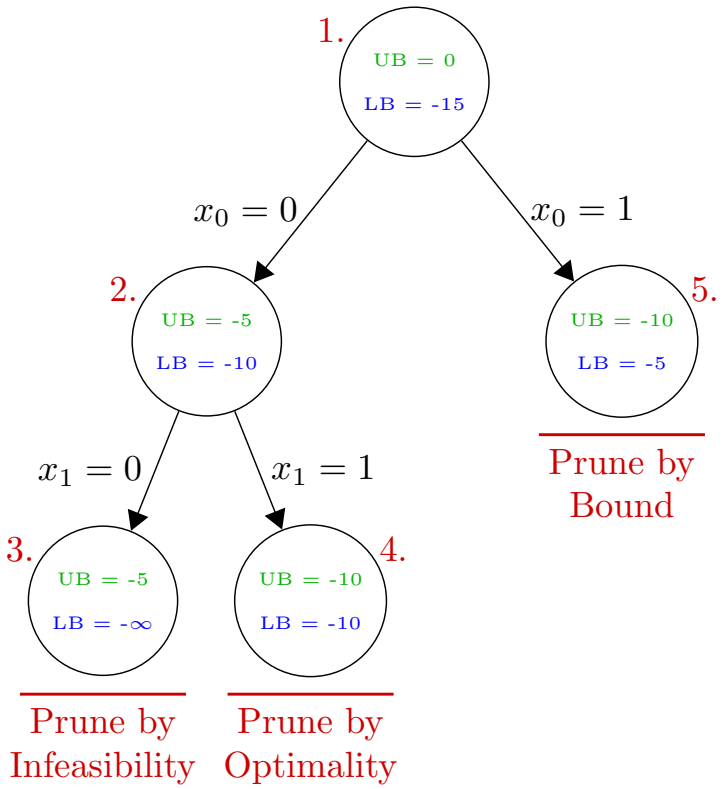


Figure 4.1: Example of Traversing the Branch-and-Bound Tree

- Lower bounds are computed by solving the LP relaxation of the ML decoding problem. The most prominent LP solvers are the ADMM, the Simplex method and the dual Simplex method. Which LP solver to choose depends heavily on the code under investigation and the target platform. A detailed comparison of these three methods is presented in [49].

- The order, in which the B&B tree is traversed, plays a large role and has to be trade-off depending on the requirements, especially regarding memory requirements and convergence speed. For reference, in a breadth-first-search, at most 2^n nodes must be stored, as they were branched but will be computed later. For depth-first-search, at most n nodes must be stored. In turn, breadth-first-search can help calculating good upper and lower bounds fast and thus help pruning the B&B tree early. A good trade-off between depth-first and breadth-first-search is the so-called best-first-search, which is a mixture of both search strategies presented in [56]. It traverses the B&B tree in a depth-first-manner until the lower bounds do no longer improve more than a certain threshold and then switches to breadth-first.

- Upper bounds are computed by finding valid codewords of the code under investigation. Applying a strong heuristic algorithm increases the overall complexity, especially for a hardware implementation, but can significantly improve the convergence

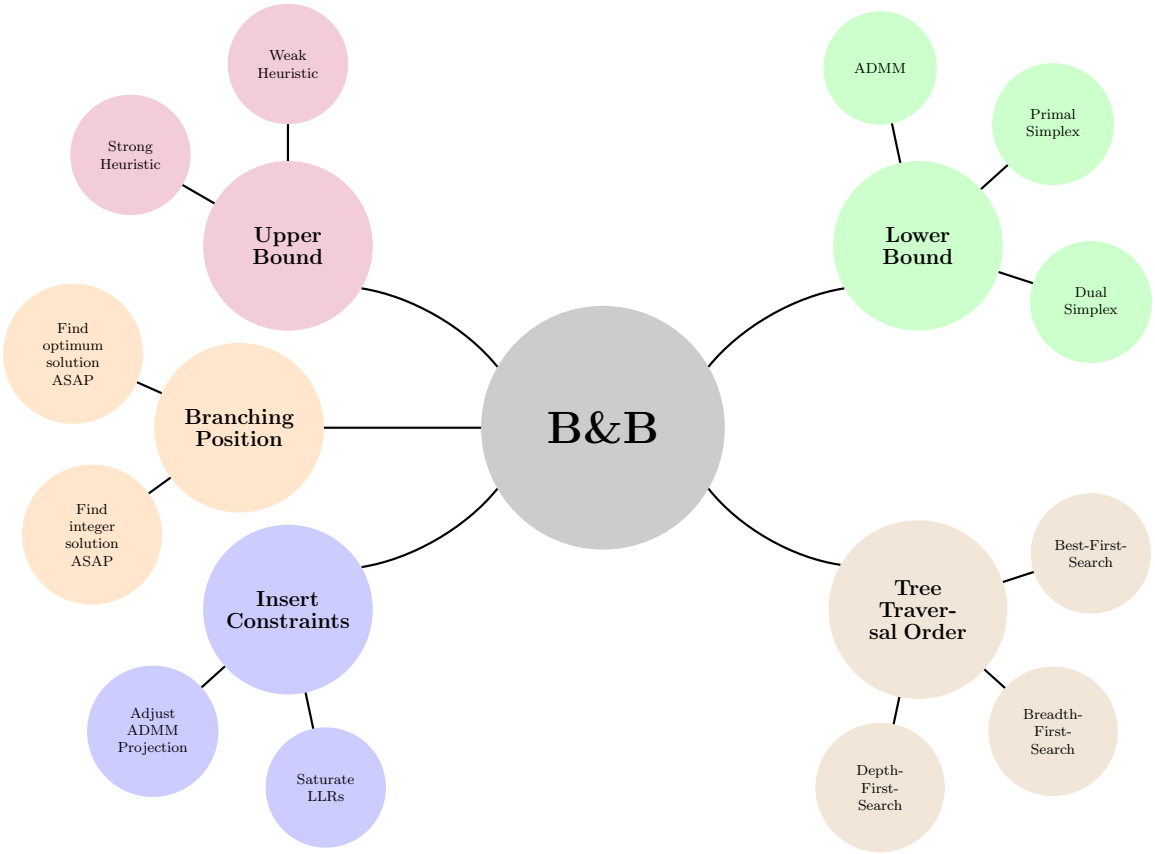


Figure 4.2: Branch-and-Bound Design Space

speed. Using a weak heuristic algorithm has a lower impact on the convergence speed, but is also of much lower implementation complexity.

- There are different possibilities do decide the branching position, i.e., which position i is branched to $x_i = 0$ and $x_i = 1$ for the next child nodes. One strategy can be to aim finding the optimal solution as soon as possible. To achieve this, always the position i with the highest absolute LLR,

$$i = \underset{j \in \{0,\dots,n-1\}}{\arg\max} \, |\lambda_j| \,,$$

is branched next. Another strategy can be to aim finding an integer solution as soon as possible. To achieve this, always the most fractional solution i of the current LP solution z, i.e.,

$$i = \underset{j \in \{0,\dots,n-1\}}{\arg\min} \, |0.5 - |z_j||$$

is branched next.

- Lastly, an open question is how to insert constraints from the B&B into the LP solver. One possibility is to saturate the corresponding LLR values to the highest or lowest possible value, i.e.,

$$\lambda_i = \begin{cases} \infty, & \text{if } x_i = 0, \\ -\infty, & \text{if } x_i = 1. \end{cases}$$

Another possibility will be presented in the next section, namely to insert the information about fixed components directly into the ADMM projection algorithm.

4.2 Adjusting the Projection Algorithm for ML Decoding

In this section, a method to insert the information about components fixed by the B&B into the ADMM-based LP solver is presented. It relies on the new projection algorithm presented in Chapter 3 and allows to remove components fixed by the B&B algorithm from all further calculations in the ADMM by adjusting the projection algorithm accordingly. Recall that for the ADMM the LP decoding problem is rewritten to

$$\begin{aligned} \min \quad & \lambda^\top x \\ \text{s.t.} \quad & T_j \cdot x = z_j \\ & z_j \in \mathcal{P}_{d_j,\text{even}}, \ \forall j \in \{0,\dots,m-1\}. \end{aligned} \tag{4.3}$$

Recall that the transfer matrix T_j serves as a "masking" of the vector x according to the jth row of H: If $H_{i,j} = 1$, then x_i will be a component of $T_j x$.

4.2.1 Fixing Components in the Branch-and-Bound

When fixing components in the branching steps, there are initially four possible cases. Assume that the i-th component of x is fixed.x' denotes the vector x without its i-th component and λ' the vector λ without its i-th component.

Even Parity Polytope

In the first four cases, consider the case that the current LP, before fixing the ith component of x, is of form

$$\begin{aligned} \min \quad & \lambda^\top x \\ \text{s.t.} \quad & T_j x = z_j \\ & z_j \in \mathcal{P}_{d_j,\text{even}}. \end{aligned} \tag{4.4}$$

First Case Assume that $x_i = 0$ is fixed, and $H_{j,i} = 0$, so there is no 1 in the i-th column of T_j. Denote by $T'_j \in \{0,1\}^{d_j \times (n-1)}$ the matrix T_j without its i-th column. Then, the above LP can be rewritten as

$$\begin{aligned} \min \quad & \lambda'^\top x' \\ \text{s.t.} \quad & T'_j x' = z_j \\ & z_j \in \mathcal{P}_{d_j,\text{even}}. \end{aligned} \tag{4.5}$$

Note that neither the objective function nor the property of z_j was changed in this case. Thus, the rest of the ADMM can be run as usual by replacing x by x' and T_j by T'_j.

Second Case Let $x_i = 1$ be fixed and $H_{j,i} = 0$, so there is still no 1 in the i-th column of T_j. Denote by $T'_j \in \{0,1\}^{d_j \times (n-1)}$, as above, the matrix T_j without its i-th column. So, the LP can be rewritten as

$$\begin{aligned} \min \quad & \lambda'^\top x' + \lambda_i \\ \text{s.t.} \quad & T'_j x' = z_j \\ & z_j \in \mathcal{P}_{d_j,\text{even}}. \end{aligned} \tag{4.6}$$

In this case, the objective function has changed. However, as the function is minimized, the constant λ_i does not have to be considered. It is only relevant in the computation of the lower bound in the current node of the B&B tree. As in the above case, z_j is not changed. Therefore, the rest of the ADMM can be run as usual by replacing x by x' and T_j by T'_j.

Third Case Assume that $x_i = 0$ is fixed and $H_{j,i} = 1$. Then some component in z_j is equal to 0, namely the one corresponding to x_i. Theorem 3.1 $ii)$ states that the remaining components of z_j, i.e., z'_j, must be contained in $\mathcal{P}_{d-1,\text{even}}$. Furthermore, this means that

$T'_j \in \{0,1\}^{(d_j-1)\times(n-1)}$. The LP can then be rewritten as

$$\begin{aligned} \min \quad & \lambda'^{\top} x' \\ \text{s.t.} \quad & T'_j x' = z'_j \\ & z'_j \in \mathcal{P}_{d_j-1,\text{even}}. \end{aligned} \tag{4.7}$$

Note that the dimension of the parity polytope to be projected on is reduced by one, but the rest of the ADMM can still be run as usual by replacing x by x' and T_j by T'_j. .

Fourth Case Assume that $x_i = 1$ is fixed and $H_{j,i} = 1$. Then some component in z_j is equal to 1, namely the one corresponding to x_i. Theorem 3.1 $i)$ states that the remaining components of z_j, i.e., z'_j, must be contained in $\mathcal{P}_{d-1,\text{odd}}$. As above, this means that $T'_j \in \{0,1\}^{(d_j-1)\times(n-1)}$. The LP can be rewritten as

$$\begin{aligned} \min \quad & \lambda'^{\top} x' + \lambda_i \\ \text{s.t.} \quad & T'_j x' = z'_j \\ & z'_j \in \mathcal{P}_{d_j-1,\text{odd}}. \end{aligned} \tag{4.8}$$

Note that, in this case, the projection changed from a projection onto the even parity polytope to a projection onto a smaller-dimensional odd parity polytope. As in the second case, the constant λ_i does not have to be considered.

As can be seen in the last case, fixing a component in the B&B algorithm can lead to projecting onto the odd parity polytope instead of the even one. Therefore, in the following, the same four cases starting with the odd parity polytope are considered.

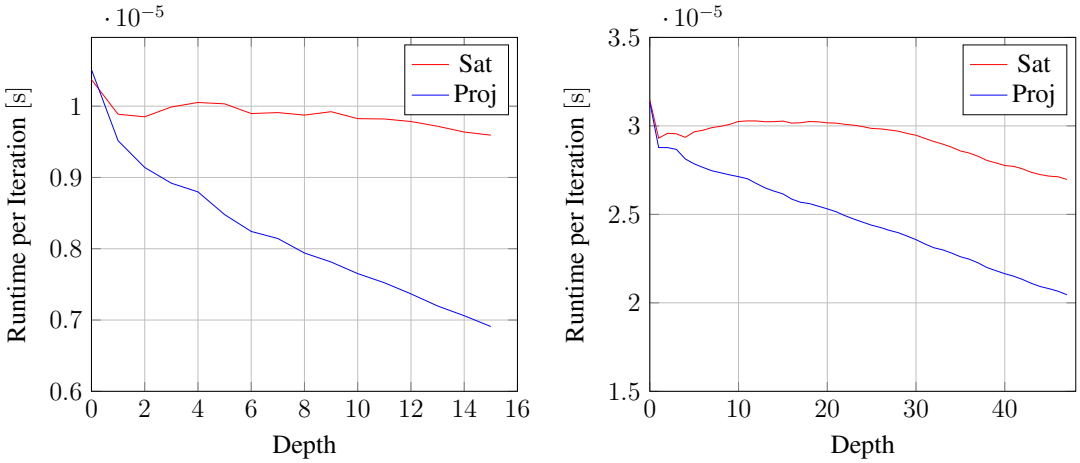
Odd Parity Polytope

In addition to the four cases above, consider the case that the current LP, before fixing the i-th component of x, is of the form

$$\begin{aligned} \min \quad & \lambda^{\top} x \\ \text{s.t.} \quad & T_j x = z_j \\ & z_j \in \mathcal{P}_{d_j,\text{odd}}. \end{aligned} \tag{4.9}$$

The four cases starting with the odd parity polytope can be treated analogously to the four cases starting with the even parity polytope by replacing every "even" by "odd" and vice versa. Here, parts $iii)$ and $iv)$ of Theorem 3.1 can be applied.

To summarize, the LP problem in the child nodes (subproblems) of the B&B tree can be stated in the form of Equation 4.3, where the size of the data T'_j, z'_j, and x' decreases the more components of x are fixed, thus potentially also decreasing the runtime and complexity.



(a) Runtime of CCSDS(32,16) from [14]

(b) Runtime of LDPC(96,48) from [14]

Figure 4.3: Runtime per ADMM Iteration of the two Different Approaches

4.2.2 Complexity Comparison

In this section the two approaches for incorporating the information about components fixed by the B&B into the LP solver are compared (see Figure 4.2). The first approach described above, in the following called "Proj", instantiates a new ADMM instance for every node in the B&B according to the above mentioned theory. For each row of the parity-check matrix, a boolean is passed indicating whether the projection has to be performed onto the even or odd parity polytope, respectively. In contrast, the second approach, in the following called "Sat", saturates the incoming LLR values of fixed components to the highest or lowest possible value, thereby enforcing the component to be set to zero or one, respectively. This approach only needs one instance of the ADMM and only modifies the input LLR values.

In Figure 4.3, the runtime of the ADMM is compared over different depths in the B&B tree for two different codes. The measurements were carried out on one core of an Intel® Core™ i7-6700 CPU 3.4 GHz with 16 GB memory. The parity-check matrices for these codes can be found in [14]. It can be observed that the runtime of the ADMM with the "Sat" approach remains about constant over the depth of the current node processed in the B&B, whereas under "Proj", the runtime decreases with increasing depth of the processed node. This is due to the fact, that in "Proj", smaller-dimensional optimization problems are solved by the ADMM with increasing depth in the B&B, whereas the size of the optimization problem in "Sat" remains constant. Therefore, "Proj" is superior when targeting an efficient software implementation.

However, for an efficient hardware implementation, "Sat" is superior due to its regular structure: While "Proj" either consumes a lot of hardware resources for the multiple instances of the ADMM or a complex control logic to en- and disable the parts of the ADMM that are not needed for the current subproblem, "Sat" can use the same instance of the ADMM for every subproblem. Even though "Proj" efficiently uses the fixed values from

Table 4.1: Results of Matrix Sparsification

(a) BCH Codes

n	k	#1s H	#1s H^{opt}	loss
31	11	120	no improvement	
31	16	120	no improvement	
31	21	120	no improvement	
31	26	80	no improvement	
63	30	594	396	33.3%
63	36	486	384	21.0%
63	39	672	336	50.0%
63	45	432	288	33.3%
63	51	336	288	14.3%
63	57	192	no improvement	

(b) RS Codes

n_b	k_b	#1s H	#1s H^{opt}	loss
60	12	408	208	49.0%
60	20	500	272	45.6%
60	28	448	288	35.7%
60	36	552	308	44.2%
60	44	392	272	30.6%
60	52	240	192	20.0%
155	85	2800	2196	21.6%
155	125	2040	1440	29.4%

the B&B to reduce the problem size of the ADMM, which can be used for efficient software implementations, the reduction cannot directly be translated to a complexity reduction in hardware implementations. Therefore, "Sat" is superior when targeting an efficient hardware implementation.

4.3 Improved Runtime of ML Decoding with Sparse Matrices

Due to the similar message-passing structure of the BP decoding algorithm and the ADMM, using sparse matrices might benefit the decoding performance of the LP decoder and consequently, also the ML decoder. To this end, the effect of minimizing the one-entries of the H-matrix of short dense codes before performing ML decoding is investigated. To obtain the sparsified matrices, the heuristic of Chapter 2, Algorithm 1, was applied to short Bose-Chaudhuri-Hocquenghem (BCH) and RS codes. The considered BCH codes have a blocklength of $n = 31$ and $n = 63$. The investigated RS codes have a blocklength of $n = 60$ and $n = 155$ in their binary image representation. The initial parity-check matrices were taken from [14]. Algorithm 1 was run with Python and Gurobi on a PC with Intel® Xeon® CPU E5-1650 v3 and 64 GB memory. The integer programs (P_i) from Equation 2.14 were solved with Gurobi $6.5.1$ with a timelimit of $t_{\max} = 12$ seconds per integer program and the Gurobi parameter values MIPFocus $= 1$ and Heuristics $= 0.5$.

The results of the matrix sparsification are depicted in Table 4.1. It can be observed that, on average, the BCH matrices were improved by around 30% and the RS matrices

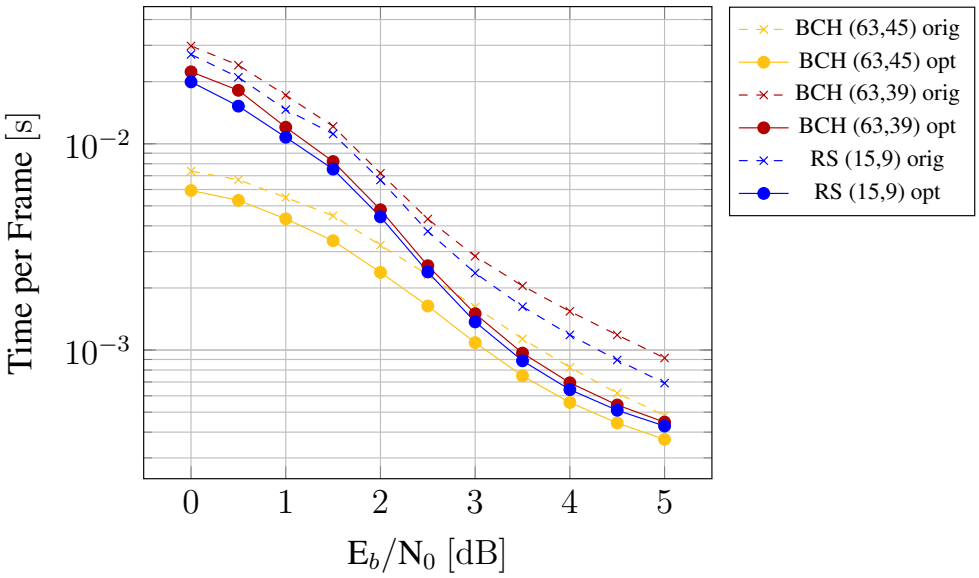


Figure 4.4: Runtime Improvements

were improved by around 35%. Note that the BCH matrices, which could not improved further by Algorithm 1, were already of minimum sparsity as shown in [17].

The one-minimized matrices of all considered codes are provided online for further usage [14].

To elaborate the benefit of the one-minimized matrices for ML decoding, the state-of-the-art B&B ML decoder from [56] was run with original and optimized parity-check matrices as input. The decoder was run on a PC with Intel® Xeon® CPU E5-1650 v3 and 64 GB memory. Figure 4.4 depicts the runtime improvements for the $(63, 45)$ BCH code, the $(63, 39)$ BCH code and the $(60, 36)$ RS code. The achieved speedups are at most 54% and 33% for the $(63, 39)$ and $(63, 45)$ BCH code, respectively, and up to 45% for the considered RS code. To put these speedups in perspective, consider calculating the ML decoding performance of the $(63, 39)$ BCH code. For this Monte-Carlo simulation, 10^3 ML decoding runs were performed for SNRs 0 to 1.5, 10^4 decoding runs for SNRs 2 and 2.5, 10^5 decoding runs for SNRs 3 and 3.5 and 10^6 decoding runs for SNRs 4 and 4.5. In total, the time needed for this Monte-Carlo simulation reduced from about *37 h 40 min* to *8 h 20 min*, while the matrix preprocessing only took around *5 min*.

Part II

Channel Coding for DRAM

Chapter 5

DRAM Background and Modeling

DRAMs were invented by Robert H. Dennard in 1966. Today, they are widely used as main memories as they meet a favorable trade-off between high capacity and low latency. DRAMs are so-called *volatile* memories, i.e., they lose their content when the single memory cells are not refreshed regularly. Scheduling these regular refresh requests is, among others, the task of the DRAM's controller. It is the brain of the DRAM and responsible for orchestrating all incoming requests, scheduling refreshes, while staying compliant with the respective standard defined by the JEDEC. This standard specifically defines valid command sequences and timings, which have to be fulfilled in order to ensure a running system. Not being fully standard compliant can lead to errors, e.g., if old data is still on the data bus because timings were not met. Therefore, a formal description of the DRAM standard is very valuable to check the controller for standard compliance. In addition, a model of state-of-the-art DRAM devices, which can easily be adopted to new standards, is crucial to evaluate new ECC schemes to ensure an error-free data storage.

In detail, the main contributions of this chapter are the following:

- The DRAM Petri net model from [15] is extended into a timed Petri net model. This model enables a verification of the DRAM controller for correct functioning regarding allowed state transitions and correct timing behavior. The model is described by a domain-specific language (DSL), namely DRAMml, from which an executable model can be generated. This model is correct by construction and enables a fast simulation-based controller validation.

The contributions of this chapter were published in [15, 21].

5.1 DRAM Basics and Terminology

A DRAM cell stores a binary value as capacity in a single transistor-capacitor-pair as shown in Figure 5.1. As soon as a specific cell should be accessed, charge on the connected

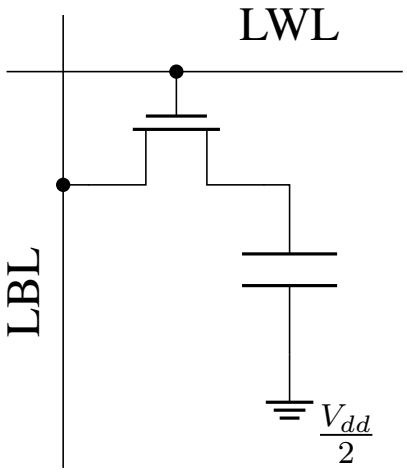


Figure 5.1: One-Transistor DRAM Cell

local wordline (LWL) enables the access transistor such that the capacity from the storage capacitor can be written from or read on the connected local bitline (LBL). The single DRAM cells are arranged in a grid-like structure, so-called sub-arrays, consisting of rows and columns that are connected via the word- and bitlines. These sub-arrays are connected to sensing circuits and sense amplifiers, where the cell values are finally read.

In addition, the DRAM is arranged in so-called banks consisting of several rows and column. Different banks can be accessed in parallel. Knowing the bank, row and column, the position of a specific DRAM cell is completely characterized.

The architecture of the sense amplifiers leads to different ways how the information is stored in the actual DRAM, namely:

- **True-Cell** bit storage, where a logical `1` is always stored at high voltage (e.g. 1.1 V) and a logical `0` is stored at low voltage (0 V).
- **Anti-Cell** bit storage, where a logical `1` is stored at low voltage (0 V) and a logical `0` is stored at high voltage (1.1 V) value, i.e., the bit value is stored inverted.
- **Mixed-Cells** a combination of both, true and anti-cells depending on the address of the accessed cell.

The heart of the DRAM lies in its controller. It is responsible for translating incoming access requests from the cache or the central processing unit (CPU) into commands to a specific DRAM address. The controller determines the actual mapping of the requested data onto the physical memory address. A good address mapping, that ideally results in as few row-misses as possible, can largely increase the DRAM's bandwidth (see [57]). In addition, the controller schedules refresh operations to keep the stored data error-free and performs the ECC (see Chapter 6). The controller's functioning has to be compliant with

Table 5.1: Description of DRAM States

State	Description
Active	At minimum one bank is active.
IDLE	No bank is active, no refresh, no power-down.
PDNP	*Precharge Power-Down*: Full power-down mode with no bank active.
PDNA	*Active Power-Down*: Partial power-down mode with at least one bank active.
SREF	*Self-Refresh*: No bank is active, all cells are refreshed.

the respective JEDEC standard, which specifies all the states the DRAM can be in, all the commands that can be sent, and all the timings that have to be met. The specified states and commands are shown in Tables 5.1 and 5.2, respectively. To access the DRAM cell at a given address, an activate (`ACT`) command has to be sent to the specific bank. The specific row is then loaded into the bank's row buffer[1]. Afterwards, a number of read (`RD`) or write (`WR`) commands can be executed to the columns of interest. Whenever data from a different row in the same bank should be accessed, a precharge (`PRE`) command has to be scheduled to close the current row. This includes refreshing the current row and clearing the current row buffer to be ready to load a different row. Accessing a different row than the one currently in the row buffer is called row-miss and results in a large time penalty. Recall that different banks can be accessed in parallel, so accessing a different row on a different bank does not necessarily result in a row-miss. DRAMs are volatile memories, as the capacitor leaks the stored charge over time. Thus, the charge in every single storage capacitor has to be refreshed regularly in order to keep the correct data. A refresh (`REF`) command thus has to be performed regularly to avoid data loss. Errors occurring in the DRAM due to leaked charge are called retention errors. When the DRAM is currently not used, different power-down modes can be entered to save energy (see Table 5.1). The respective JEDEC standard specifies, which command sequences and which state transitions are valid. E.g., there cannot be an `ACT` command to an already active bank and an `SREFEN` command can only be executed if the DRAM is in *IDLE* state.

In addition to specifying valid command sequences and state transitions, the JEDEC standard also specifies timings, which have to be respected by the DRAM controller. A detailed description of all DRAM timing parameters is given in [15, 21, 58]. In general, the DRAM timings can be distinguished into four categories:

- **Command-to-Command Timing Dependencies:**

[1] The row buffer is actually not a buffer but a combination of the DRAM's internal sense amplifiers. As this exact behavior exceeds the scope of this thesis, the row buffer model is used for the sake of comprehensibility. The interested reader is referred to [58].

Table 5.2: Description of DRAM Commands

Command	Description
ACT	*Activate*: A specific row in one bank is activated.
PRE	*Precharge*: The currently activated row is closed and the bank is precharged.
RD	*Read*: Read from an activated row.
RDA	*Read with Auto-Precharge*: Read from a row and precharge the row afterwards.
WR	*Write*: Write to an activated row.
WRA	*Write with Auto-Precharge*: Write to a row and precharge the row afterwards.
PDE	*Power-Down Entry*: Enters the *PDNA* mode if in *Active* or *PDNP* if in *IDLE*.
PDX	*Power-Down Exit*: Exits *PDNP* or *PDNA* mode.
PREA	*Precharge All*: All active banks are precharged.
REFA	*Auto-Refresh*: Refresh one or more rows in all banks.
SREFEN	*Self-Refresh Entry*: Enters the *SREF* mode.
SREFEX	*Self-Refresh Exit*: Exits the *SREF* mode.

The execution of one command implies that other commands have to wait for a specific time interval. For example, between two ACT commands targeting different banks there must be a minimum time interval of t_{RRD}. These command-to-command timing dependencies are the majority of timing dependencies in the JEDEC standards.

- **Command Bus Occupancy Dependency:**
  Only one command can be scheduled to the command bus during one cycle. For example, if there are several commands ready, the controller will have to resolve this issue by prioritizing the commands.

- **N-Activate Window:**
  In order to limit peak currents there exists a rolling time-frame in which a maximum of N banks can be activated, called t_{NAW}. For double data rate (DDR)3, $N = 4$, and it is called *Four Activate Window* (t_{FAW}), whereas in Wide I/O DRAM, $N = 2$, called *Two Activate Window* (t_{TAW}).

- **Refresh Mechanism:**
  Each DRAM cell must be refreshed every $t_{REF} = 64\,\text{ms}$. Therefore, a REF com-

(a) Petri Net **(b)** Petri Net After Firing τ_0

Figure 5.2: Firing Transitions in a Petri Net

mand must be scheduled every $t_{REFI} = 7.8\,\mu$s to avoid data corruption. However, the standards allow postponing the `REF` command up to 8 times. In case that 8 `REF` commands are postponed in a row, the resulting maximum interval between the surrounding refresh commands is limited to $t_{REFMAX} = 9 \cdot t_{REFI}$.

The goal of this chapter is to provide a Petri net model of the DRAM according to the JEDEC standard. In the following section, the concept of Petri nets is recapped.

5.2 Petri Nets Background

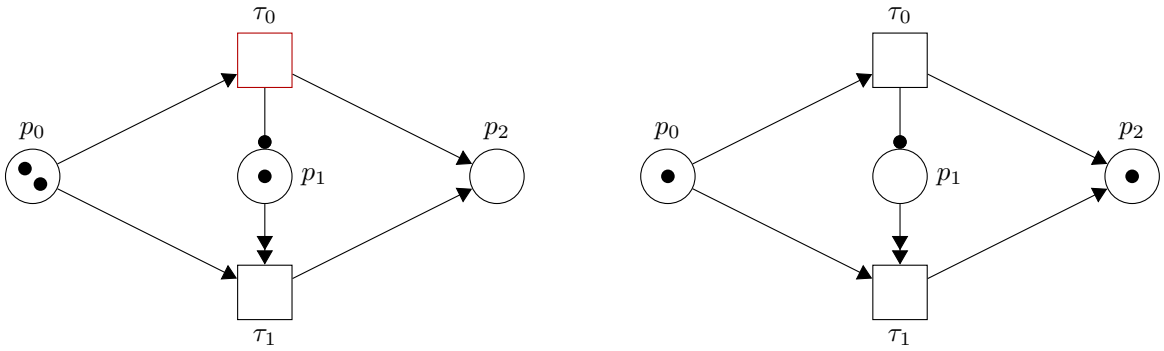
Petri Nets [59] are very general system models especially used to describe concurrent systems. According to [60], a Petri net is defined as follows:

Definition 5.1 (Petri Net)**.** *A Petri Net $N = (P, T, F, W, M_0)$ is a 5-tuple where $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places, $T = \{\tau_1, \tau_2, ..., \tau_n\}$ is a finite set of transitions, which satisfy $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$. $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs ($\rightarrow$) from places to transitions and vice versa, $w : A \rightarrow \mathbb{N}_{>0}$ is a weight function (if not indicated the weight is set to 1) and $M : P \rightarrow \mathbb{N} \cup \{0\}$ is the marking of the places with tokens.*

The main concept of Petri nets is the "firing" of transitions, which moves tokens from the input places to the output places. The firing rules [60] works as follows: A transition τ is called enabled, if each input place p of τ holds at least $w(p, \tau)$ tokens. An enabled transition may fire, which removes $w(p, \tau)$ tokens from every input place p of τ and adds $w(\tau, p')$ tokens to every output place p' of τ.

Figure 5.2 shows an example of a simple Petri Net. The transition τ_0 is enabled since there are enough input tokens in its two input places. As soon as the τ_0 fires, one token is removed from p_0, two tokens are removed from p_1 and two tokens are generated at p_2. After firing, τ_0 is no longer enabled.

In order to model the behavior of DRAMs with Petri nets, two extensions to Definition 5.1 are required, namely reset nets [61, 62] and inhibitor nets [62–64].



(a) Petri Net **(b)** Petri Net After Firing τ_1

Figure 5.3: Reset and Inhibitor Arcs in a Petri Net

Definition 5.2 (Reset Net)**.** *A Reset Net is a tuple, $N^R = (N, R)$, where N is a Petri Net and $R \subseteq (P \times T)$ denotes the set of reset-arcs (↠).*

The firing rules for reset-arcs are as follows: A reset-arc does not affect the enabling of transitions. Let (p, τ) be a reset-arc. As soon as τ is fired, all tokens from p will be removed.

Definition 5.3 (Inhibitor Net)**.** *An Inhibitor Net is a triple, $N^I = (N, I, W_I)$, where N is a Petri Net, $I \subseteq (P \times T)$ specifies the set of inhibitor-arcs (⟞), and $w_I : I \to \mathbb{N}_{>0}$ is a weight function.*

In the case of an inhibitor net, the firing rules for inhibitor-arcs are as follows: Let (τ, p) be an inhibitor arc of weight $w_I(\tau, p)$. Then, τ is disabled as long as p holds at least $w_I(\tau, p)$ tokens. Vice versa, it is enabled, if p holds less than $w_I(\tau, p)$ tokens.

The example in Figure 5.3 shows the behavior of reset and inhibitor arcs. Initially, transition τ_0 is inhibited by the token in place p_1. Firing τ_1 once resets the token from p_1, so that τ_0 is no longer inhibited.

To be able to model timing behavior, the authors of [65] defined timed-arc nets, in which tokens are modified to have an assigned age and transitions have additional firing rules depending on the token's age.

Definition 5.4 (Timed-Arc Net)**.** *A Timed-Arc Petri Net is a Petri Net, where each token of the marking M gets an age $x_m \in \mathbb{R}_{\geq 0}$ assigned. The set of arcs A is extended by time intervals, i.e., $A \subseteq (P \times \mathcal{J} \times T) \cup (T \times P)$, where $\mathcal{J}$ is the set of all valid time intervals. Timed-arcs are denoted by $\xrightarrow{[t_1, t_2]}$ for a time interval $[t_1, t_2] \in \mathcal{J}$.*

The firing rules for Timed-Arc Petri Nets are modified as follows: A transition τ is enabled, if each input place p of τ holds at least $w(p, \tau)$ tokens of age $x_m \in J$, where $w(p, \tau)$ is the weight of the arc from p to τ and $J \in \mathcal{J}$ is its time interval. Firing the

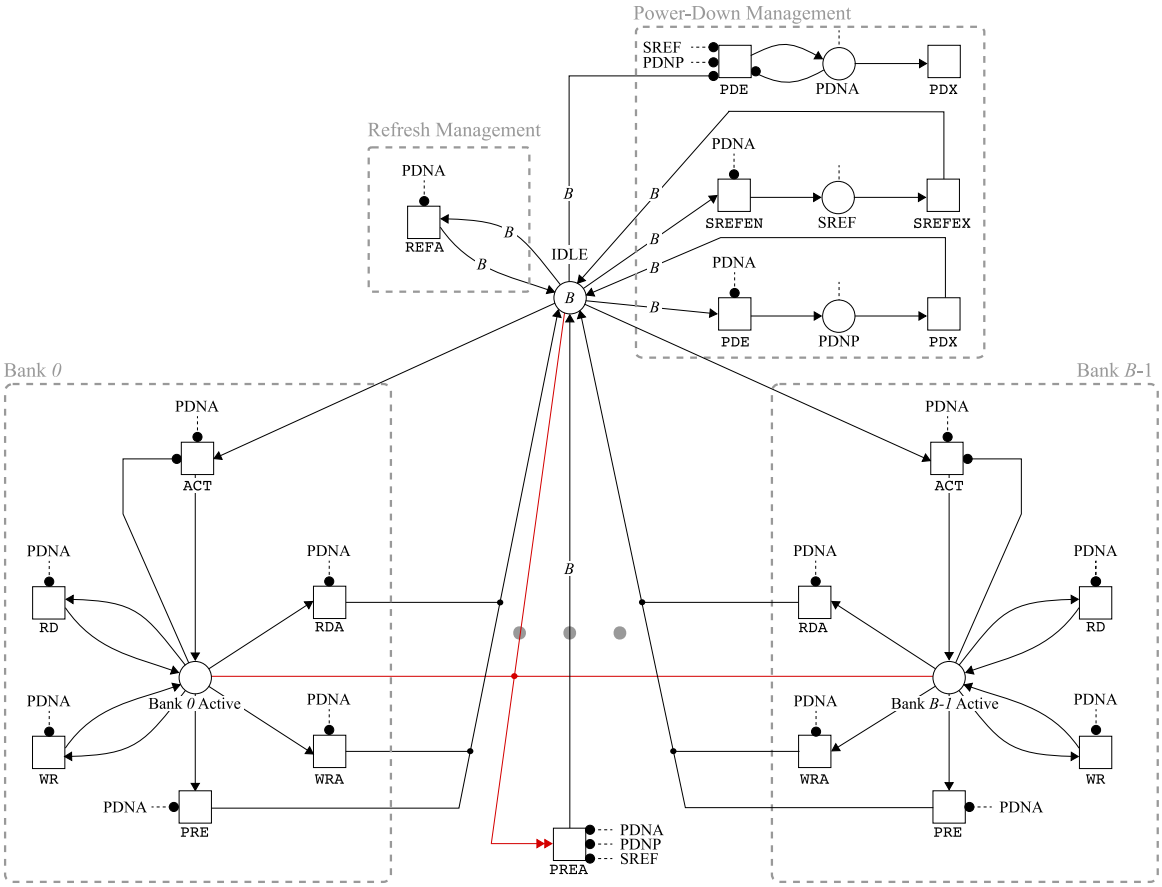


Figure 5.4: DRAM Petri Net Model [15]

transition τ removes $w(p, \tau)$ tokens of age $x_m \in \mathcal{J}$ from each input place and adds $w(\tau, p')$ tokens of age 0 to each output place p' of τ.

Following this logic, a timed inhibitor-arc from a place p to a transition τ inhibits the transition τ from firing as long as p holds at least $w_I(\tau, p)$ tokens of age $x_m \in J$, where J is the age guard of the timed inhibitor-arc. Timed inhibitor-arcs are depicted as $\overset{[t_1,t_2]}{\multimap}$ for $[t_1, t_2] \in \mathcal{J}$.

5.3 DRAM Modeling

In this section, the DRAM protocol defined in the JEDEC standard will be modeled as a Petri net. This section will assume a DDR3 DRAM, but can be easily adapted to different standards. First, a model of the basic DRAM states and commands will be modeled as Petri net with inhibitor and reset arcs. This model can be used to validate the DRAM controller regarding valid state transitions. Second, the model will be extended by timed-arcs to also model the DRAM timings, allowing a full validation of a DRAM controller according to the respective JEDEC standard.

5.3.1 Modeling States and Commands

Using Definitions 5.1, 5.2 and 5.3, the states and command dependencies for DRAMs can be modeled as an *Inhibitor-Reset Petri Net* as depicted in Figure 5.4. The transitions correspond to the executed DRAM commands from Table 5.2 (`ACT`, `PRE`, `RD`, `RDA`, `WR`, `WRA`, `PDE`, `PDX`, `PREA`, `REFA`, `SREFEN`, and `SREFEX`) and the places correspond to the different DRAM states from Table 5.1 (*IDLE*, *Active*, *PDNP*, *PDNA*, and *SREF*). It is assumed that the DRAM consists of B banks.

The DRAM Petri net can be divided into several subnets, namely

- B subnets corresponding to the B banks,
- a refresh management, and
- a power-down management.

The initial marking consists of B tokens in the *IDLE* place. A row in some bank $b \in [0, \ldots, B-1]$ is activated by executing an `ACT` command to the specific bank, which is the same as firing the corresponding `ACT` transition. Firing this transition moves one token to the *Bank* b *Active* state and enables the `RD`, `WR`, `RDA`, `WRA` and `PRE` command on bank b. The initial marking ensures that B banks can be active in parallel. If at least one bank is in *IDLE* state, the active power-down mode (*PDNA*) can be triggered. Conversely, if all banks are *IDLE*, self-refresh (*SREF*) or precharge power-down (*PDNP*) can be entered. In addition, a refresh command can be executed (`REFA`). When executing a `PREA` command, all active banks will be closed and precharged. This is modeled by the red reset-arcs, which remove all tokens from the active banks and returns B tokens to the *IDLE* state, resetting the Petri net to its initial marking. In addition, several inhibitor-arcs ensure that no invalid state transitions can take occur.

This model ensures the correct functioning of any DRAM controller regarding valid state transitions. In addition, it can be used to calculate the DRAM's power consumption: The power consumption of DRAMs can be divided into a background or static part, which depends on the time spent in each DRAM state (*IDLE*, *Active*, *PDNP*, etc.), and an active part, which is needed for each command (`ACT`, `PRE`, `RD`, etc.). Using, e.g., the simulation tool DRAMPower [66–68], the overall power consumption can be easily derived while executing the Petri net model.

In the next step, this model is extended to also check for invalid timings.

5.3.2 Modeling Timing Dependencies

With Definition 5.4, the timing dependencies for the DRAM protocol can be modeled by a *Timed-Inhibitor-Reset Petri Net*. As described in Section 5.1 the different timings distinguished into four categories. For each category, a solution based on timed-arc nets is

presented. The resulting timing dependency network can be treated separately or as an add-on to the state and command model in Figure 5.4.

Command-to-Command Timing Dependencies

In order to model command-to-command timing dependencies, for example t_{RRD} between two `ACT` commands targeting different banks, a new custom arrow $\xrightarrow{t_x}$ is introduced, which is composed of a place, a reset-arc, and a timed inhibitor-arc, as shown in Figure 5.5.

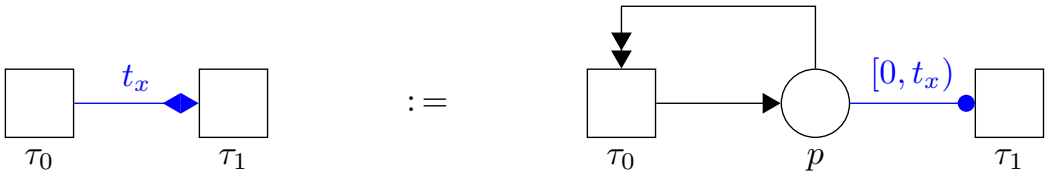


Figure 5.5: Custom Timing Arc

As soon as transition τ_0 is fired, all existing tokens on place p are cleared by the reset-arc and a new token of age 0 is generated. The timed inhibitor-arc attached to τ_1 inhibits its firing as long as the age of the token is smaller than the timing value t_x (i.e., as long as its age lies in the interval $[0, t_x)$). By the use of this arc all command-to-command timing dependencies can be modeled easily as shown in Figure 5.6 for an artificial DRAM with 2 banks. For instance, between the `ACT` command of Bank 0 and the `ACT` command of Bank 1, there exists a special arc with t_{RRD} as timing constraint and vice versa.

Command Bus Occupancy

The command bus occupancy could be modeled with the special command-to-command arc by connecting each command transition to each other command transition with a t_{ck} timing dependency. However, in this case, the number of special timing arcs would grow exponentially with the number of transitions and slow down the execution of the model. Therefore, a special place is used, which represents the command bus. This place is connected to all command transitions in the Petri Net as shown in Figure 5.7. With this approach, the number of arcs only grows linearly with the number of transitions.

If the τ_i transition is fired (e.g., an `ACT` command) it consumes the token on the command bus and generates a new token of age 0. The other transitions τ_n (e.g., a `RD` to a different bank) can only be fired if the token has at least age t_{ck}. With this constraint only one command can be sent to the DRAM at a time within one clock cycle. This small net also shows the flexibility of this approach. For example, in the case of LPDDR4, a variable command length was introduced (i.e., commands are longer than just one clock cycle). This new requirement can be modeled easily by changing the timing parameter from t_{ck} to the new timing parameter.

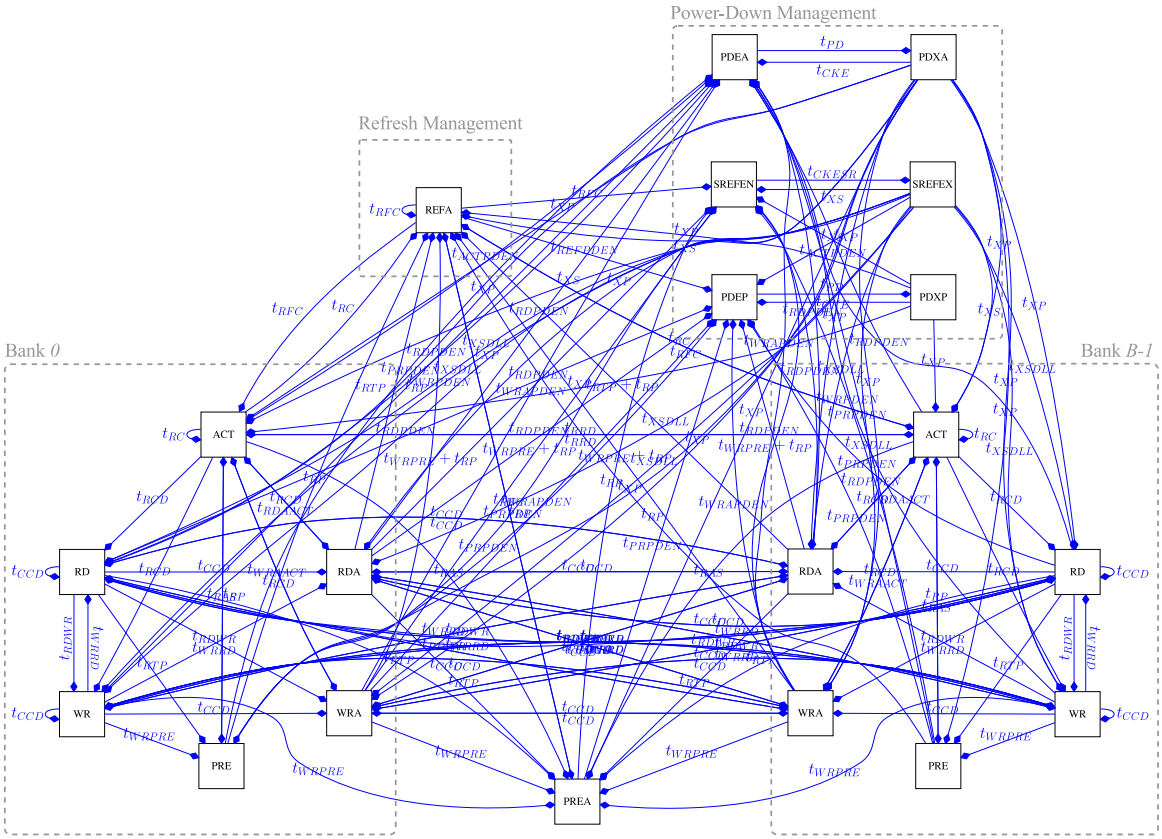


Figure 5.6: 2-Bank Petri Net with Command-to-Command JEDEC Timing Delays

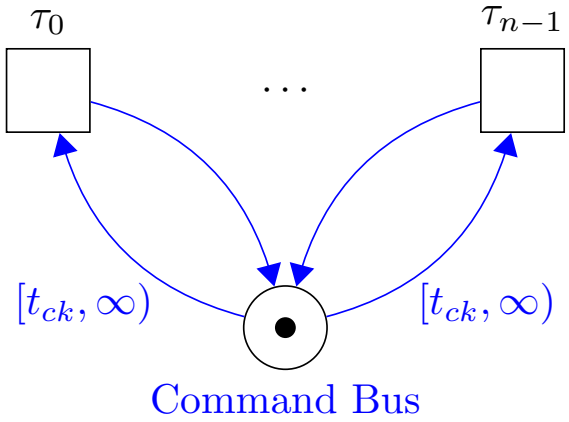


Figure 5.7: Modeling of the Command Bus

N-Activate Window

As shown in Section 5.1, there exists the timing dependency t_{NAW}, the so-called N-*Activate Window*. In the case of DDR3, $N = 4$, it is also called *Four Activate Window* or in short t_{FAW}. In this window only four `ACT` commands can be issued due to power constraints. Similar to the command bus dependency a special place called “FAW Pool” is

used, which holds exactly four tokens. Figure 5.8 shows an abstract DRAM model omitting non-relevant states in order to explain the functionality.

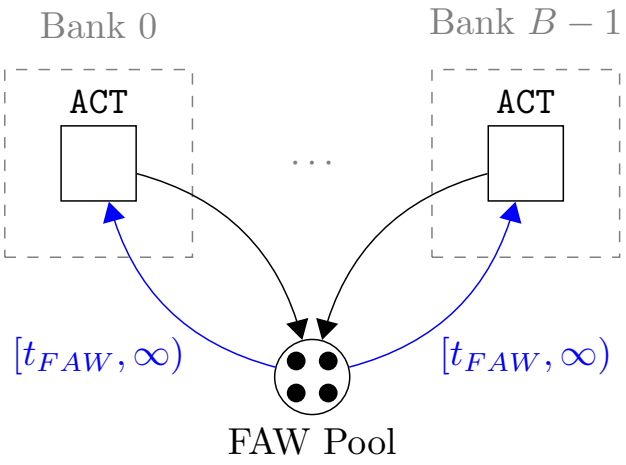


Figure 5.8: Modeling of t_{FAW} with B Banks

Whenever an ACT transition is fired, a token of age at least t_{NAW} is consumed and a new token of age 0 is generated. Therefore, the FAW Pool always holds four tokens of different ages. The tokens in the FAW Pool can be considered as "disabled" after their consumption for t_{FAW} time. Since there are four tokens in the pool, it is ensured that only four ACT transitions can happen in a t_{FAW} time window. Again, this model can be easily adapted to, e.g., the case of a *Two Activate Window* by replacing the timing t_{FAW} by t_{TAW} and initiating the pool with only two tokens.

Refresh Mechanism

To avoid data losses, refresh commands have to be scheduled regularly, i.e., refreshes have to be scheduled every $t_{REFI} = 7.8\mu$s. Since the JEDEC standard allows postponing refresh commands up to eight times, after at most $t_{REFMAX} = 9 \cdot t_{REFI}$ a refresh has to be performed. Figure 5.9 depicts the realization of this refresh mechanism in our Petri net.

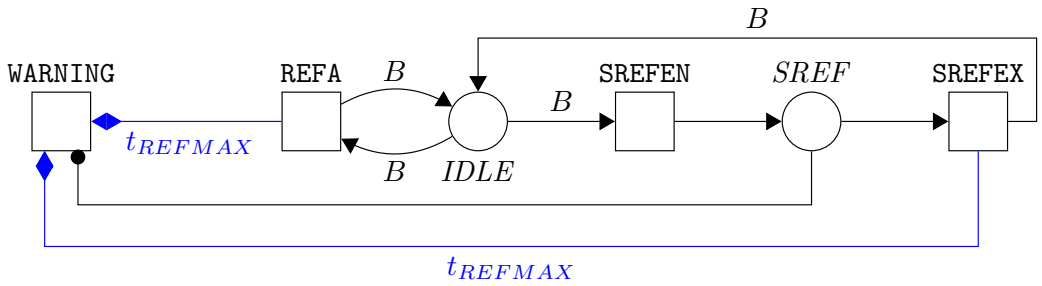


Figure 5.9: Modeling of Refresh Mechanism

If no REFA transition is fired within t_{REFMAX} time, the WARNING transition becomes enabled, which should trigger a warning output to the user. To avoid the warning during self-refresh, the *SREF* state additionally inhibits the WARNING and the SREFEX resets the t_{REFMAX} timer. Instead of using the WARNING transition, a specific *Halt* place could be used in a similar way, which would inhibit **all** transitions from firing whenever it holds a token of age larger than t_{REFMAX} to prevent the Petri net from proceeding any further. However, especially in the context of new approaches like *Approximate DRAM* [69, 70], the refresh period can be enlarged or refresh can be completely turned off on purpose. Therefore, instead of forcing a complete halt, only a warning is thrown.

5.3.3 DRAM Modeling Language and Practical Usage

In the following, the benefits for modeling the DRAM as a Petri net are highlighted.

First, a DSL called *DRAMml* was developed, which describes all the timings, states and commands of the JEDEC in a formal, comprehensive, readable, understandable and easily adaptable format. DRAMml is described in more detail in Appendix A.

From this DSL, executable SystemC code was generated in order to interact directly with controller simulation models for a fast simulation-based validation. A key feature is that the generated executable model is correct by construction. Details on how the SystemC model is generated from the DSL can be found in [21].

Finally, this generated executable model was connected to the most prominent DRAM simulators, namely DRAMSim2 [71], DRAMSys [72], Ramulator [73], as well as the DRAM controller in gem5 [74, 75]. Furthermore, we connected the executable model to an RTL memory controller [76]. As result, a bug was found in the DRAMSys simulator. In detail, the timing t_{WRPDEN}, which denotes the minimum time interval between a WR command and a PDE command, was violated. This highlights the benefit of the correct-by-construction executable model obtained from the DSL DRAMml.

Chapter 6

DRAM ECC

Since DRAMs are the premier choice for state-of-the-art main memories, keeping the stored data error-free is crucial to maintain a running system. To ensure this, regular refresh operations have to be performed to recharge the volatile DRAM cells and avoid retention errors. These refresh operations have a huge impact on the DRAM's overall power consumption, thus, being able to increase the refresh interval while maintaining error-free data is of great interest. Therefore, ECC techniques are employed. However, due to the strict requirements of DRAMs regarding area, power and bandwidth (see Figure 1.4), DRAM ECC techniques have to be powerful enough to provide a solid error-correction capability, but cannot exceed the very hard restrictions on the implementation KPIs. Because of this, only very basic, yet dedicated, decoding schemes can be employed for the use in DRAMs. In order to find dedicated ECC schemes, first, the error behavior of DRAMs has to be investigated and well understood.

This chapter makes the following contributions:

- Based on theoretical considerations and observations on the error types of a single DRAM cell, DRAM retention errors are modeled as an information-theoretic noisy channel, the so-called Z-channel. Due to the higher channel capacity of the Z-channel compared to other binary channels, the search for better and dedicated ECC techniques is motivated, which have to exist according to Shannon's fundamental theorem of information theory.
- A new, lightweight error-mitigation method, Flip ECC, is derived, which is based on the aforementioned DRAM error model. It massively improves the DRAM error behavior without affecting its latency or bandwidth.
- An ECC method tailored for DDR4 DRAMs is presented. It makes use of DDR4's internal DBI mechanism to exploit the DRAM's asymmetric error behavior with low overhead.

The contributions of this chapter were published in [22, 23] and resulted in one patent [25].

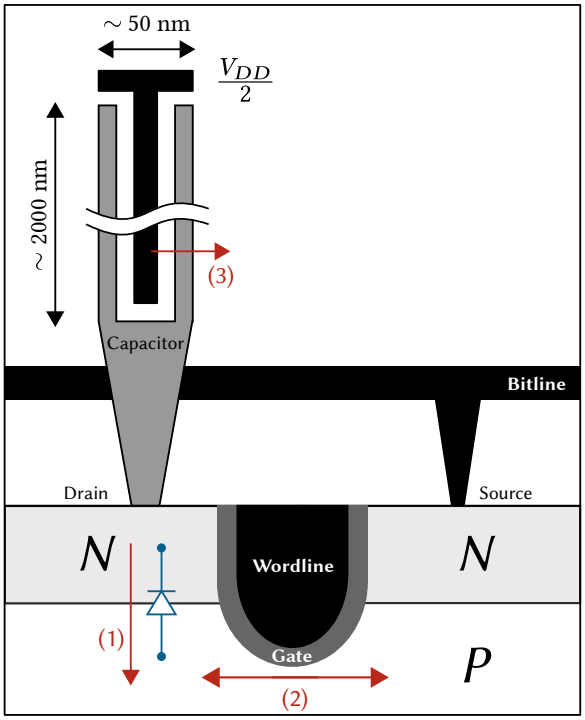


Figure 6.1: Overview of DRAM Leakage Paths [22]

6.1 DRAM Retention Errors

Recall from Chapter 5.1, that the data in a DRAM cell is stored as charge in a capacitor (see Figure 5.1). Therefore, it is natural to believe, that DRAM retention errors can only be of type `1`→`0` for true-cell DRAMs or of type `0`→`1` for anti-cell DRAMs. However, there are several DRAM retention error sources, which might also lead to reverse flips in rare cases.

6.1.1 DRAM Retention Error Sources

In the following, the main effects on DRAM retention errors will be presented. The focus on this overview is less on the technical details and more on how stored `1`s and `0`s are affected. To simplify the considerations, assume a true-cell DRAM in the following, so high voltage implies a logical `1`. For more information on this topic, the reader is referred to [22, 77–80].

Cell Leakage

First, the different cell leakage effects can lead to DRAM retention errors. There exist the following three cell leakage paths, as shown in Figure 6.1:

- *Drain Leakage* (1) is the main leakage path in the DRAM cell and is the main leakage source for a 1→0 error.
- *Sub-Threshold Leakage* (2) can slightly charge the cell and thus lead to a 0→1 error when the connected bitline is in precharged state. It can also slightly discharge a stored 1, but the leakage will be very small.
- *Cell Capacitor Leakage* (3) can degrade both stored 1s and stored 0s.

Crosstalks

Second, in addition to the before mentioned cell leakage effects, the cell content or its sensing can also be disturbed by crosstalk effects. The following two major sources of crosstalks exist:

- *Bitline-to-Bitline Coupling* is the main source of crosstalk. It can result in sensing errors, especially if the cell is already degraded due to one of the before mentioned leakage paths.
- *Wordline-to-Wordline Coupling* can increase the sub-threshold leakage of cells connected to wordlines neighboring the currently activated wordline.

6.1.2 DRAM Retention Error Measurement

Knowing the main retention error sources, it is still unknown how they will affect the DRAM error distribution. Therefore, measurements on off-the-shelf DDR4 DRAM were conducted. To this end, the measurement platform from [81] was used to investigate the retention error behavior of stored 1s and stored 0s. For the experimental setup, random data pattern were written in the DRAM over a varying amount of time and for two different temperatures. The result are depicted in Figure 6.2. It can be seen, that the error rate of 0→1 flips is several orders of magnitude lower than the error rate of 1→0 flips. For instance, at 60°C and after 1s, only 0.005% of all errors were 0→1 flips. Thus, the retention error behavior of DRAMs is highly asymmetric.

6.1.3 DRAM Retention Error Modelling

With this insight about the asymmetric error behaviour in DRAMs, the goal of this section is to model the DRAM as a noisy channel and evaluate its benefit with an information theoretic approach.

Recall from Chapter 1 that data storage can be modeled as a binary channel (see Figure 1.8) with crossover probabilities p for 1→0 flips and q for 0→1 flips. With the data

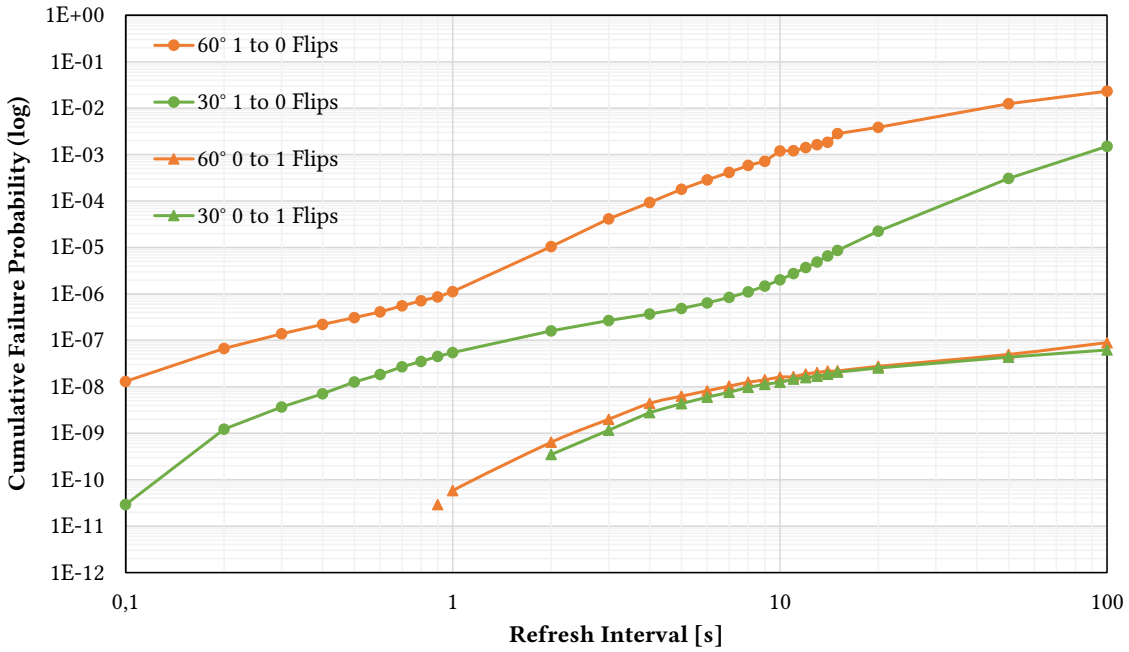


Figure 6.2: Failure Probabilities for 0's and 1's

plotted in Figure 6.2, crossover probabilities of $p = 9 \cdot 10^{-6}$ and $q = 4.6 \cdot 10^{-10}$ can be calculated for a single cell in a random data pattern for 1s refresh interval at 60°C. Under these probabilities, the channel capacity calculated from Equation 1.12 equals the capacity calculated from Equations 1.14

$$C_{DRAM-BC} = 0.999918 = C_{DRAM-Z}.$$

If the DRAM consists of both true- and anti-cells, the so-called U-channel can be used to model the DRAM as a noisy channel. The U-channel either behaves like a Z-channel or like the inverted $\bar{\text{Z}}$-channel, so it can be used to model a mixed-cell DRAM with a regular structure. The channel capacity C_U of the U-channel is so far not fully determined but can be approximated by

$$C_Z - \frac{1}{n} \leq C_Z - \frac{H(p)}{n} \leq C_U \leq C_Z. \tag{6.1}$$

Since the channel capacity of the U-channel is smaller than C_Z, it is beneficial to first use the reverse-engineering technique presented in [23] to identify the true- and anti-cell regions and to logically reduce to the case of a full true-cell DRAM.

Figure 6.3 depicts the theoretical error behavior of the BSC, the Z-channel and the U-channel depending on the number of 1s in the DRAM burst element consisting of 64 bit. In addition, a graph for DRAM errors obtained from real measurements (see [23]) is depicted. It can be seen that the Z-channel approximates the real DRAM retention error behavior best: Calculating the average error as a discrete integral of the shown curves, the Z-channel reduces the deviation from the measurements to 24.5%, whereas the U-channel

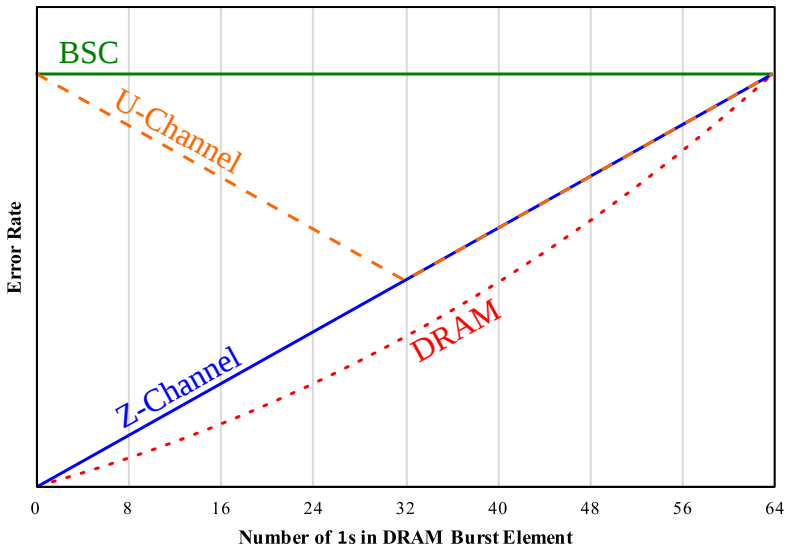


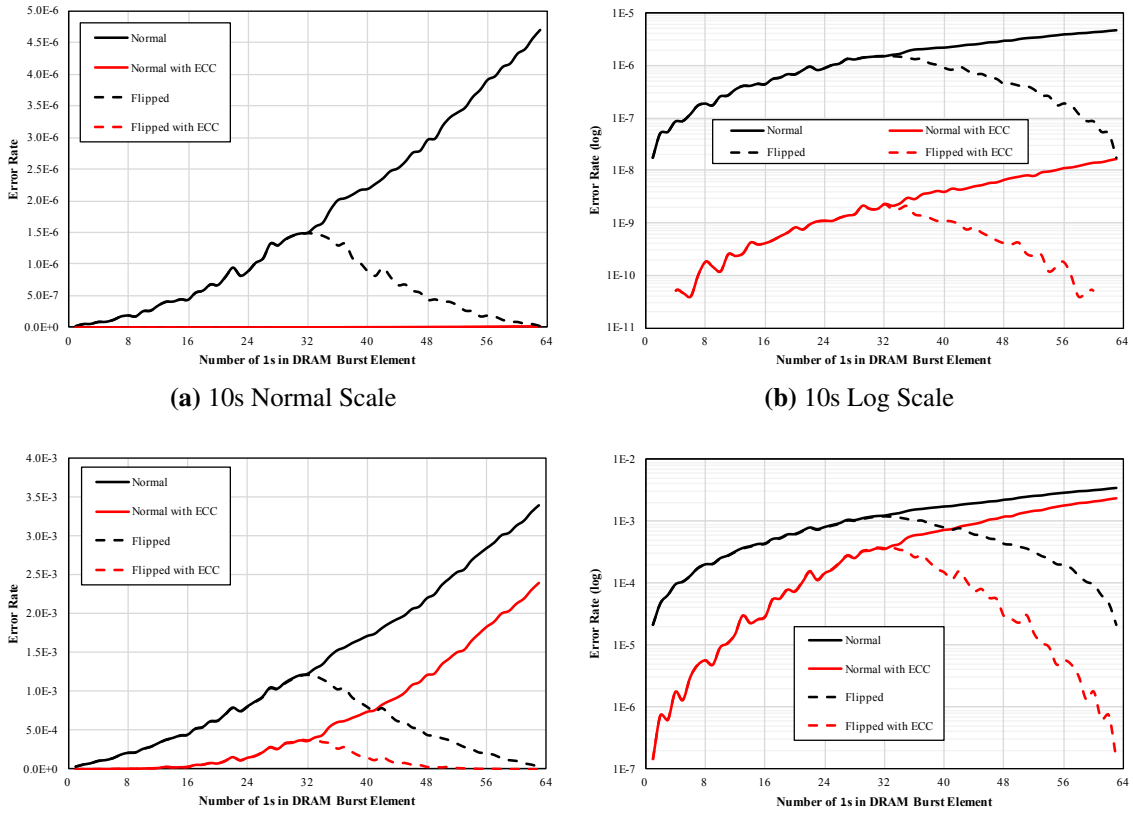
Figure 6.3: Error Rate Depending on the Number of 1s

assumes 87.7% and the BSC 149% more errors than actually occurred. When weighting the occurrences of the input with a binomial distribution, which corresponds to the natural distribution of ones and zeros in a random bitstring, the Z-channel assumes 37.6% more errors, whereas the U-channel and the BSC assume 49.2% and 171% more errors, respectively.

6.2 ECC for DRAM

The state-of-the-art DRAM ECC method extends an 8-device DRAM DIMM by an additional 9th device to store the ECC redundancy. The code that is usually employed to perform the error correction and detection is a Hamming code, that is a single error correction / double error detection (SED/DED) code [82, 83]. Another well-known approach is IBM's Chipkill [84], which is capable of restoring a full chip failure. In addition, several vendors already introduced on-die ECC [85, 86] to avoid retention errors. Using this ECC, they were able to lower the refresh rates by a factor of $4\times$ while maintaining a low error rate. Thereby, the DRAM power consumption was reduced. This shows the importance of ECC for retention errors in DRAMs.

State-of-the-art DRAM ECC methods, however, assume a symmetric error behavior. However, being able to model the data storage in a DRAM as a noisy Z-channel motivates the search for even better coding schemes taking the special asymmetric error structure of DRAM retention errors into account. According to Shannon's fundamental work [2], better coding schemes have to exist due to the higher channel capacity of the Z-channel. One popular coding scheme for the Z-channel is the so-called Berger code [87], which can detect (but not correct) all asymmetric errors. However, a pure error detection is not useful for the use of DRAMs as main memories, since the data cannot be recovered.



(a) 10s Normal Scale

(b) 10s Log Scale

(c) 100s Normal Scale

(d) 100s Log Scale

Figure 6.4: DRAM Retention Errors with Flip ECC

In the following, two DRAM ECC methods will be presented, which rely on the DRAM's asymmetric error behavior.

6.2.1 Flip ECC

In this section, a bit-flipping approach is presented, which is used to largely increase DRAM's reliability with little overhead. The approach relies on the reverse-engineering approach presented in [23] as it requires the DRAM controller to know the pattern of true- and anti-cells. In the following, a full true-cell DRAM will be assumed.

As shown in Figure 6.3, the Z-channel behavior of the DRAM retention errors yields more errors happening when the corresponding DRAM burst element contains a large number of ones. Therefore, the presented ECC method, Flip ECC, flips the whole DRAM burst element whenever it contains more ones than zeros. This approach requires a one bit flip information per burst element, which is set to `1` if the burst is flipped and to `0` if the burst is not flipped. This one bit has to be stored and secured along with the data. Therefore, the SED/DED Hamming code in ECC DRAM, which needs 8 bit redundancy to protect the 64 bit burst element, is reduced to 7 bit redundancy to protect 65 bit of data. Obviously, this

reduction of redundancy comes with a drawback: The extra parity bit, which enables the double-error-detection, is deleted. In consequence, upon occurrence of a double-bit error in the burst element, the reduced Hamming code will detect the error, but handle it as in the case of a correctable one-bit error, resulting in an additional error because of the decoder's attempt to correct a single-bit error. However, this behavior only affects the probability of undetected errors, but not the error rate.

The results of the bit-flipping approach for a DDR4 true-cell DRAM with refresh switched off for 10s and 100s, respectively, are depicted in Figure 6.4. To obtain these results, randomized patterns were applied to the entire DRAM device containing a specific number of 1s in the DRAM write burst. First, a random bit string of length equal to the device's burst size is generated with a fixed number of 1s. Second, the DRAM is written completely with this bit string. Third, the refresh is disabled, the DRAM is kept at 25 °C room temperature, and the data is read out after a specific time. For each number of 1s the above steps are repeated five times with shuffled data patterns. In the Flip ECC approach, the complete DRAM burst element is inverted if and only if it contains more than 32 ones, which results in a mirroring of the retention error curve and therefore in a smaller error probability. In addition to the results of the bit-flipping approach, also the error rates for ECC DRAM using a Hamming SED/DED, and the combination of both schemes are shown. With Flip ECC, the error probability of DRAMs can be lowered significantly, thus increasing its reliability. In some applications, where parts of the data contain a large number of ones, like applications using negative 2's complement numbers with a small dynamic range, the effect of only flipping the data can even outperform the reliability increase through ECC (see Figures 6.4c and 6.4d).

6.2.2 DDR4 ECC Using Data Bus Inversion

DDR4 DRAMs support DBI to save power and to reduce the simultaneous switching noise [88] on the bus. It is shown that a typical memory usage of 70% reads to 30% writes yields about 27% system power savings when DBI is enabled [89]. This is an optional feature which can be enabled in the DRAM's mode register settings. DDR4 supports DBI in write direction, read direction and both write and read direction, with one `dbi_n` pin for every byte lane, indicating whether the corresponding byte is inverted or not. Once DBI is enabled in the mode register settings, it works as follows:

- For reads, the DRAM checks if more than four bits of a byte lane are low. If yes, then it will invert the data on that byte lane and drive the `dbi_n` pin low. If four or less number of bits in a byte lane are low it sends the original stored data and drives the `dbi_n` pin high. A value of `dbi_n` = `0` indicates that the data was sent inverted, and needs to be inverted again at the controller PHY.
- For writes, the DRAM controller PHY checks the number of bits which are low. If more than four bits in a byte lane are low, PHY will invert the byte lane and drive the

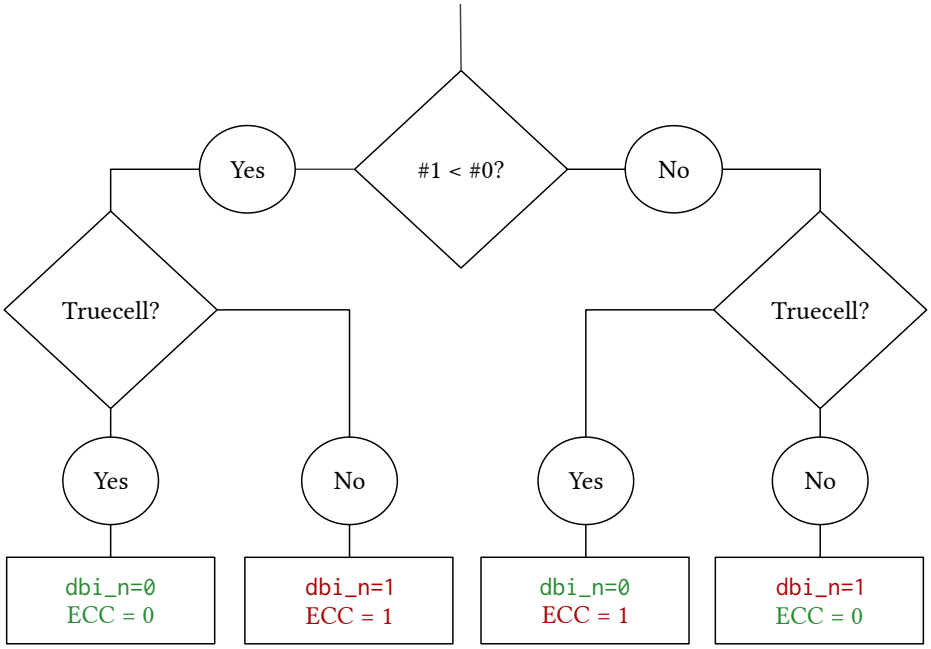


Figure 6.5: Flowchart for DBI ECC

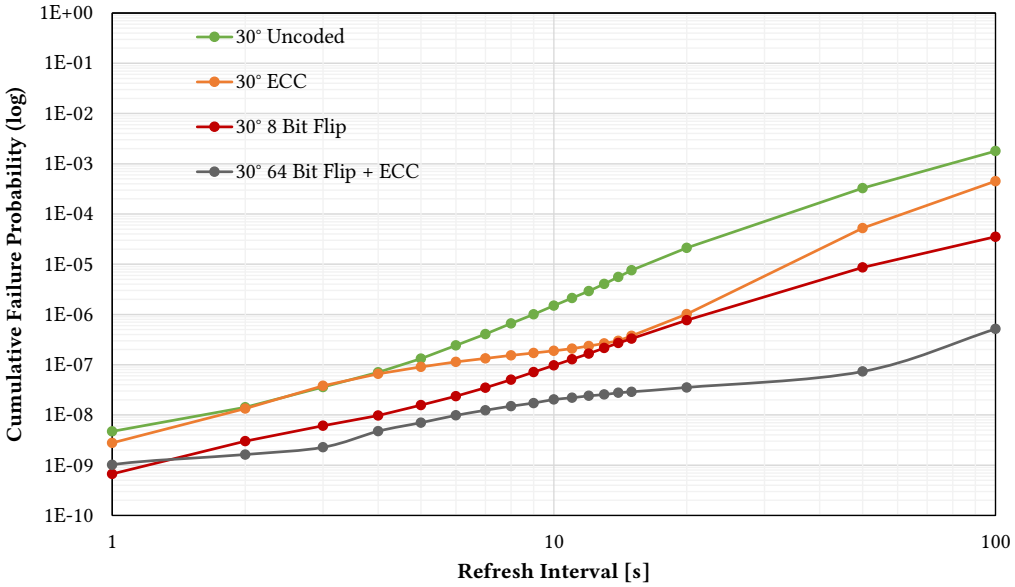
`dbi_n` pin low, so the DRAM has to invert the data back internally before storage. If four or less number of bits are low, no data is inverted and `dbi_n` remains high.

The advantage of the DBI mechanism in DDR4 DRAMs lies in the availability of a flip mechanism, which flips the data depending on the incoming data. In the following, a coding scheme will be presented, which works similar to Flip ECC (see Section 6.2.1) but incorporates the flipping information directly into the `dbi_n` pin. The goal is to store the data zero-dominant for true-cell DRAMs and one-dominant for anti-cell DRAMs. Note that the original `dbi_n` signal is set in such a way that each byte of data is transmitted one-dominant.

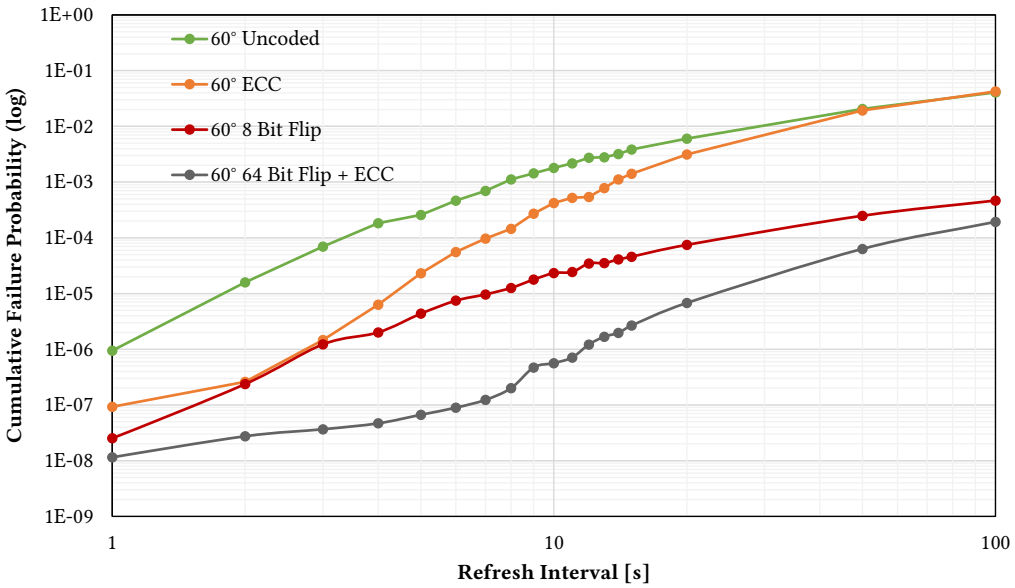
To combine this one-dominant flavor for the bus and the zero-dominant flavor for the storage in a true-cell DRAM, an additional ECC bit is required for every byte of data. This ECC bit indicates, whether the data stored in the DRAM is the original data or not. Whenever data is written to the DRAM it may or may not be flipped depending on the modified `dbi_n`. Whenever data is read from the DRAM, the according ECC bit is loaded and XORed with the corresponding `dbi_n` to get the information whether the data has to be inverted or not. For a DRAM burst element consisting of 64 bit, 8 ECC bits are needed, which is the same as for the traditional Hamming SED/DED code.

The flowchart in Figure 6.5 shows how the ECC bit and the `dbi_n` are set for each byte of data. If DBI is only enabled in write direction, the data can still be retrieved correctly on the read path by setting `dbi_n` = 0 for true-cell regions and `dbi_n` = 1 for anti-cell regions.

To evaluate the gain of this method, error-rate measurements were conducted using the measurement platform presented in [22, 81] for different test data. To model an application



(a) 30°C



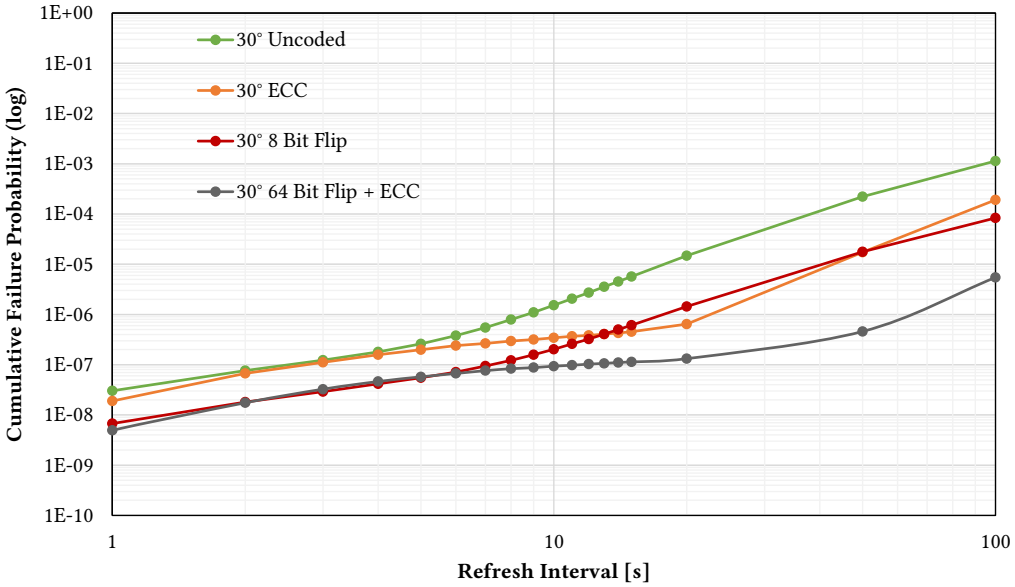
(b) 60°C

Figure 6.6: DBI ECC with 8 Bit Dynamic Range

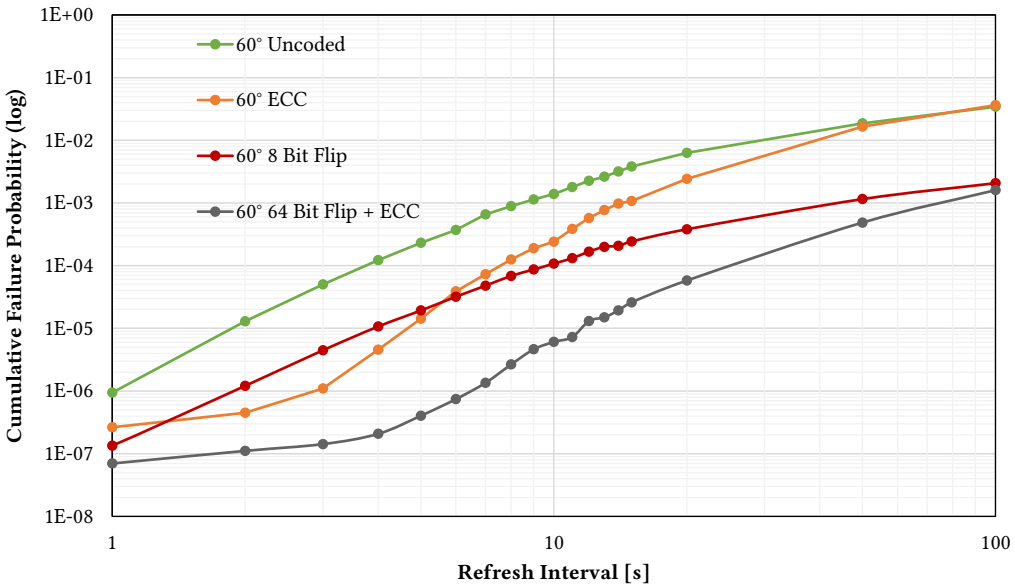
using 2's complement numbers with small dynamic range, each 64 bit of test data consist of a varying part of random data (8 or 16 bit), while the rest is padded with either ones or zeros. The results are depicted in Figures 6.6 and 6.7.

In all figures, the green curve corresponds to uncoded data, the orange curve shows the error probability when securing the data with a SED/DED Hamming code, and the red curve depicts the results of the proposed DBI ECC. The gray curve was obtained by combining a modified inversion approach with a modified ECC scheme as follows: As for Flip ECC, the SED/DED Hamming code is reduced by its extra parity bit, which enables the double error detection. Instead, the ECC bit (see Figure 6.5) is stored for a flip granularity of 64 bit. Note that this requires heavy modifications to the DBI mechanism in order to make a decision whether or not to flip the data on an 64 bit granularity.

As can be observed in the figures, the proposed DBI ECC (red curve) outperforms the conventional ECC (orange curve), especially for the higher temperatures and increased refresh intervals, while utilizing the same amount of redundancy. This indicates that the proposed scheme can indeed exploit the higher capacity of the Z-channel. In addition, the modified approach depicted in the gray curves shows significant improvement over both the proposed method and the traditional SED/DED.



(a) 30°C



(b) 60°C

Figure 6.7: DBI ECC with 16 Bit Dynamic Range

Chapter 7

Conclusion and Future Work

Communication systems are essential in our modern world. Data transmission, both spatially and temporally, must be reliable and efficient to keep up with upcoming standards. In this thesis, two applications to error-correction coding were investigated.

Regarding channel coding for data transmission, Ensemble BP was presented in Chapter 2 as a flexible universal decoding algorithm. In a two-step approach, an ensemble of different sparse matrix representations of the same code was created. This ensemble was then combined in BP decoding. The universal approach is applicable to any linear code class and can compete with the communications performance of the code's state-of-the-art decoding algorithm. In the context of ML decoding, a large improvement for ADMM-based LP decoding was achieved in Chapter 3. In particular, a new projection algorithm was presented, which does not require a sorting operation and thus significantly lowers the hardware implementation complexity of the algorithm. This improvement in ADMM-based LP decoding was further elevated in the context of ML decoding in Chapter 4. The new projection algorithm was incorporated directly into the ML's B&B framework, even further reducing the runtime of ML decoding.

Regarding channel coding for data storage, the theoretical basis for treating data storage in DRAMs as a noisy channel was laid in Chapter 6. Observations on the error types of a single DRAM cell showed that DRAM retention errors can be modeled as an information-theoretic Z-channel. This Z-channel has a higher channel capacity than usual symmetric binary channels, which motivates the search for better and dedicated ECC techniques for DRAMs. In the remainder of this thesis, two different ECC schemes were presented. The first one, Flip ECC, is a lightweight error-mitigation method, the second one an ECC method tailored to DDR4 DRAMs, using DDR4's internal DBI mechanism. Both schemes rely on the DRAM's asymmetric error behavior and improve the DRAM error behavior without affecting its latency or bandwidth.

While notable improvements were achieved in both applications, there is still room for future work.

Chapter 2 made large steps towards using BP as a universal decoding algorithm, but the decoding performance for short and dense codes still shows potential for improvement when compared to the ML decoding performance. I see a large potential when combining Ensemble BP with weighted BP as proposed in [38]. Weighted BP can be combined with Ensemble BP in two ways: First, it can be applied to improve the performance of the single BP instances, leading to a faster convergence and less required iterations. Second, it can be used to further diversify the different decoding units and thereby improve the performance of the BP ensemble. In addition, to save resources, a neural network could be trained to identify the BP instance with highest probability to correctly decode the incoming LLR values instead of executing the whole BP ensemble.

In this regard, an ML approximation for BP decoding, similar to the ML approximation for Chase decoding and ordered statistics decoding performed in [90], would be of interest. Potentially, the task could be broken down to finding an optimal matrix representation for a single BP instance, either through elementary row operations as proposed in Chapter 2 or by altering the matrix dimensions as proposed in [43], [44] or [91].

Regarding ECC for DRAMs, future work will involve custom ECC schemes for existing and upcoming standards, such as DDR5 or DDR6. While future standards will most likely already foresee the use of internal ECC mechanisms, an additional error-correction or error-mitigation method can always be used to qualify the memory for high-reliability applications.

Chapter 8

Zusammenfassung

Kommunikationssysteme sind ein wesentlicher Bestandteil unserer modernen Gesellschaft, was sich zum Beispiel durch eine verlässliche mobile Verbindung zum Internet verdeutlicht. Sich navigieren lassen, mit Freunden und Familie auch über große Distanzen in Kontakt zu bleiben oder ein Geschäftsmeeting remote abzuhalten - was vor 30 Jahren noch undenkbar war, ist heute dank der Fortschritte in der mobilen Kommunikation alltäglich.

Die stetige Erhöhung der Datenraten und die Notwendigkeit zuverlässiger Datenübertragung stellen erhebliche Herausforderungen an Kommunikationssysteme. Um eine zuverlässige Datenübertragung zu gewährleisten werden Fehlerkorrekturverfahren angewendet, welche die zu übertragenden Daten robust gegenüber Kanalrauschen und Übertragungsfehlern machen. In seiner fundamentalen Arbeit „A Mathematical Theory of Communication“ von 1948 legte Claude Shannon [2] die Grundlage für eine zuverlässige Datenübertragung über verrauschte Kanäle. Die Idee besteht darin, den zu übertragenden Daten redundante Informationen hinzuzufügen. Diese Redundanz wird dann auf Empfängerseite in der Decodiereinheit genutzt, um sowohl Fehler zu identifizieren, die während der Übertragung aufgetreten sind, als auch die ursprünglichen gesendeten Daten wiederherzustellen.

Das Herzstück der zuverlässigen Datenübertragung liegt in der Decodiereinheit, welche Übertragungsfehler erkennt und idealerweise korrigiert. Neben einer guten Fehlerkorrekturleistung spiegeln sich in der Decodiereinheit jedoch auch die Herausforderungen neuer Kommunikationsstandards wider, die sowohl eine leistungsstarke als auch eine hocheffiziente Implementierung erfordern. Die Optimierung der Decodiereinheit hinsichtlich ihrer Implementierungseffizienz, d.h. hinsichtlich Parametern wie Fläche, Durchsatz oder Leistung, und ihrer Kommunikationsperformanz stellt ein multikriterielles Optimierungsproblem dar.

Bei multikriteriellen Optimierungsproblemen ist es von großem Interesse, sogenannte Pareto-Fronten zu kennen, d.h. gültige Lösungen des Optimierungsproblems, die für mindestens eine Zielfunktion optimal sind. Sie stellen wichtige Referenzen dar und sind

von großem Wert, um die Qualität und das Potenzial einer spezifischen Lösung zu bewerten. Für die Kommunikationsperformanz kann die optimale Performanz der Decodiereinheit durch sogenanntes Maximum-Likelihood (ML)-Decoding erreicht werden. ML-Decoding optimiert die Kommunikationsleistung der Decodiereinheit mathematisch, indem die Wahrscheinlichkeit einer korrekten Wiederherstellung von Übertragungsfehlern maximiert wird. Obwohl ML-Decoding ein NP-schweres Problem ist, ist diese optimale Lösung dennoch von großem Interesse als Referenz für die Performanz des verwendeten Codes und jedes heuristischen Decodieralgorithmus.

Fehlerkorrekturtechniken werden nicht nur zur Gewährleistung einer zuverlässigen Datenübertragung verwendet. Auch die Datenspeicherung kann als Kommunikationssystem betrachtet werden, bei dem die Daten zeitlich statt räumlich übertragen werden. Genau wie bei der Datenübertragung kann auch die Datenspeicherung durch ein verrauschtes Kanalmodell beschrieben werden. Da Datenspeicher stark kostengetrieben sind, werden Technologien mit kleineren Strukturen und höherer Dichte eingesetzt. Im Fall moderner Arbeitsspeicher (DRAMs) führt dies zu einer Zunahme von Fehlern, z.B. durch Leckströme. Daher ist auch hier der Einsatz von Fehlerkorrekturverfahren notwendig, um die Zuverlässigkeit moderner Arbeitsspeicher aufrechtzuerhalten.

Die Herausforderung besteht dabei darin, eine solide Fehlerkorrekturleistung bei sehr strengen Beschränkungen der Implementierungskosten zu ermöglichen. So darf die Fehlerkorrektur z.B. nicht die Lese- oder Schreiblatenz des Speichers erhöhen oder die Bandbreite des Speichers verringern. Aus diesem Grund können nur einfache, speziell abgestimmte Decodierverfahren für den Einsatz in DRAMs verwendet werden.

Im Kontext der multikriteriellen Optimierung stellen ML-Decoding und Fehlerkorrektur in DRAMs zwei gegensätzliche Probleme dar: Für das ML-Decoding ist die optimale Fehlerkorrektur eine strikte Vorgabe, während die Implementierungseffizienz verbessert werden muss. Für die Fehlerkorrektur in DRAMs gibt es strikte Beschränkungen hinsichtlich unter anderem Durchsatz und Latenz, während die Fehlerkorrekturperformanz optimiert werden muss.

Diese Dissertation behandelt beide der zuvor genannten Probleme: die Verbesserung der Implementierungseffizienz des ML-Decodings bei gleichzeitiger Beibehaltung der optimalen Fehlerkorrektur sowie die Verbesserung der Fehlerkorrektur in DRAMs unter Berücksichtigung der strengen Vorgaben an die Implementierungseffizienz. Die Arbeit ist in zwei Teile gegliedert. Kapitel 2, 3 und 4 konzentrieren sich auf ML-Decoding und die Verbesserung einzelner Teilalgorithmen. Die verbleibenden Kapitel 5 und 6 befassen sich mit Fehlerkorrekturverfahren für die Datenspeicherung in DRAMs. Im Detail umfassen die neuartigen Beiträge dieser Arbeit:

- **Ensemble BP-Decoding (Kapitel 2)**

  Um das ML-Decodierproblem zu lösen, wird ein Branch-and-Bound-Framework verwendet. Dieses Framework reduziert das ML-Decodierproblem auf die wiederholte

Lösung von LP-Decodierproblemen und das Ausführen heuristischer Decodieralgorithmen, um gute Schranken für die optimale Lösung zu erhalten. Gute Schranken verkleinern den Suchraum des Branch-and-Bound-Algorithmus und führen zu einer schnelleren Terminierung des Algorithmus. Heuristische Decodieralgorithmen sind jedoch in der Regel codespezifisch und daher wenig flexibel.

Im ersten Kapitel wird der neue heuristische Decodingalgorithmus Ensemble Belief Propagation (BP) vorgestellt, eine Verallgemeinerung des Multiple-Bases Belief Propagation (MBBP)-Decoding [7, 8] und des Automorphism Ensemble Decoding (AED) [9–11]. Dieser Algorithmus basiert auf dem Belief Propagation Algorithmus, welcher ursprünglich zur Decodierung von Low-Density Parity-Check (LDPC)-Codes eingesetzt wurde, jedoch so modifiziert ist, dass er für beliebige Codeklassen einsetzbar ist. Der Ansatz von Ensemble BP besteht aus zwei Schritten: Zunächst wird eine geeignete dünne Matrixdarstellung des Codes mittels mathematischer Optimierung konstruiert. Anschließend wird eine geeignete Menge verschiedener Matrixdarstellungen durch die Lösung eines Maximum Coverage-Problems ausgewählt. Simulationen zeigen, dass Ensemble BP als flexibler, universeller Decodieralgorithmus für jede Codeklasse mit vernachlässigbarem Verlust in der Kommunikationsleistung gegenüber den codespezifischen Decodieralgorithmen verwendet werden kann.

- **LP-Decoding (Kapitel 3)**

  Das zweite große Hindernis für ein effizientes ML-Decoding ist ein leistungsfähiger LP-Decodieralgorithmus. In den letzten Jahren hat sich die Alternating Direction Method of Multipliers (ADMM) [12] als sehr effizient im Kontext der LP-Decodierung erwiesen [13]. Den Flaschenhals dieses Verfahrens stellt jedoch die Projektion auf ein sogenanntes Parity Polytope dar. In diesem Kapitel wird ein neuer, komplexitätsarmer Projektionsalgorithmus vorgestellt, der für eine effiziente Hardwareimplementierung optimiert ist. Während bestehende Projektionsalgorithmen auf hardware-ineffiziente Sortieralgorithmen angewiesen sind, kommt dieser neue Algorithmus ohne Sortieren aus und basiert auf einem iterativen Ansatz, bei dem die Komponenten der Projektion sukzessive festgelegt werden. Dieser verbesserte Projektionsalgorithmus reduziert die Komplexität und Laufzeit des gesamten Algorithmus erheblich.

- **ML-Decoding (Kapitel 4)**

  Zur Lösung des ML-Decodierproblems wird ein Branch-and-Bound-Framework verwendet, in dem LP-Decoding und heuristische Decodieralgorithmen für spezifische Teilprobleme wiederholt gelöst werden müssen. In diesem Kapitel wird eine elegante Methode vorgestellt, den neuen Projektionsalgorithmus aus dem vorherigen Kapitel in das Branch-and-Bound-Framework zu integrieren. Dadurch können die Vorteile der Komplexitätsreduktion nicht nur für das LP-Decoding, sondern für das gesamte ML-Decodierproblem genutzt werden.

Darüber hinaus wird die Laufzeit des ML-Decodings weiter reduziert, indem die Darstellung des betrachteten Codes modifiziert wird. Mit dieser Laufzeitreduktion von bis zu 81% konnte eine bislang unbekannte ML-Performanzkurve berechnet werden. Da ML-Decoding einen wertvollen Richtwert für die Kommunikationsperformanz jedes Codes und Decodieralgorithmus darstellt, wurden alle neuen Erkenntnisse dieser Arbeit in unserer Channel Codes Database veröffentlicht [14].

- **DRAM-Hintergrund und Modellierung (Kapitel 5)**

  Im ersten Kapitel des zweiten Teils dieser Arbeit werden Grundlagen von DRAMs rekapituliert. Als wissenschaftlicher Beitrag dieses Kapitels wird das DRAM-Modell basierend auf Petrinetzen aus [15] erweitert, um zusätzlich zur Validität der Zustandsübergänge auch das Timing-Verhalten des DRAMs zu prüfen. Das erweiterte Modell stellt ein leicht verständliches DRAM-Systemmodell dar und wird verwendet, um die Korrektheit des DRAM-Controllers gemäß dem entsprechenden JEDEC-Standard zu überprüfen und den Stromverbrauch des DRAMs einfach aus dessen Datenblattinformationen zu berechnen. Die Einhaltung des korrekten Timing-Verhaltens ist entscheidend, um Fehler zu vermeiden, die bei sehr hohen Taktfrequenzen auftreten können. Zudem ist ein DRAM-Systemmodell, welches sich schnell für aufkommende Standards anpassen lässt, essentiell, um neue Fehlerkorrekturverfahren zu evaluieren und eine fehlerfreie Datenspeicherung zu gewährleisten.

- **DRAM-ECC (Kapitel 6)**

  In diesem Kapitel wird das Fehlerverhalten einer einzelnen DRAM-Zelle analysiert und erstmals ein präzises informationstheoretisches Modell basierend auf dem beobachteten asymmetrischen Fehlerverhalten abgeleitet. Auf Grundlage dieses Fehlermodells wird ein maßgeschneidertes Fehlerkorrekturverfahren entwickelt. Dieses verbessert das Fehlerverhalten von DRAMs erheblich, ohne dabei dessen Latenz oder Bandbreite zu beeinträchtigen.

  Zusätzlich wird eine neue Methode zur Fehlerkorrektur im DDR4-DRAM-Controller vorgestellt, die Data Bus Inversion (DBI) verwendet, eine Besonderheit des DDR4-Standards. Während übliche Fehlerkorrekturverfahren für DRAMs eine spezielle DRAM-Architektur erfordern (z.B. einen 9. Chip pro DIMM, der die Redundanz für die anderen 8 Speicherchips speichert), ist diese Methode speziell auf einen DDR4-DRAM-Controller angepasst. Das Verfahren nutzt das asymmetrische Fehlerverhalten von DRAM-Zellen aus und ermöglicht so die Implementierung eines einfachen, aber effektiven Fehlerkorrekturverfahrens.

Die Beiträge dieser Dissertation wurden in den folgenden Konferenz- und Zeitschriftenartikeln veröffentlicht: [15–24]. Die Arbeiten der Publikation [22] führten zudem zu einem Patent [25].

Die Arbeiten aus Kapitel 3 sowie Teile aus Kapitel 2 und Kapitel 4 wurden in Zusammenarbeit mit der Forschungsgruppe Optimierung der RPTU Kaiserslautern-Landau im Rahmen des DFG-Projekts Nr. WE 2442/9-2 durchgeführt. Die übrigen Arbeiten aus Kapitel 4 wurden durch die EU und das Land Rheinland-Pfalz im Rahmen des InnoProm-Programms (Projektnr. 84003923) unterstützt.

Appendix

Appendix A

DRAMml

An example for the domain-specific language (DSL) *DRAMml* (see Section 5.3.3) describing a 1 Gb DDR3-800D dynamic random-access memory (DRAM) with 1 KB page size and fast power-down exit mode is shown in Listing A.1. The description starts with setting the number of banks B to 8 and the number N for the N-activate-window to 4. Next, the DRAM device is described in a top-down approach. First, all the places and state transitions on device-level are described. Second, from Line 71 on, B banks are inserted, which contain the places, transitions and tokens corresponding to the banks. Lastly, the timing constraints are inserted. Lines 104–140 correspond to the definition of the DRAM timings in ns for the given device. In addition, lines 142–145 define the additional places needed for the timing constraints, which do not correspond to the DRAM's states.

Finally, from line 147 on, the command-to-command timing arcs are inserted for modelling the device, the command bus occupancy, the NAW and the refresh mechanism are included.

The arcs used in DRAMml follow the symbols defined earlier:

- **`->`** denotes a normal arc.
- **`-> [t1, t2]`** denotes a timed-arc with age guard $[t_1, t_2]$.
- **`->>`** denotes a reset-arc.
- **`-o`** denotes an inhibitor-arc.
- **`-<> (t1, ... ,tn)`** denotes a command-to-command timing arc. Here, the first timing t_1 holds for the lowest hierarchical level (here: inside the banks), while the last timing t_n holds for the highest hierarchical level (here: on device-level). This notation allows a flexible inclusion of more hierarchical levels if a future standard requires it. A value of 0 means that no arc exists on the corresponding hierarchical level.

- ⋆ denotes a wildcard for all commands.

As an example for the used arcs, consider, the command-to-command timing arc `ACT -<> RD (tRCD,0)` in Line 152. The first part of this line, `ACT -<> RD`, means that there are timed-arcs going from **all** `ACT` commands (in all banks) to **all** `RD` commands (in all banks). The given addendum `(tRCD,0)` specifies that the timing inside the banks is t_{RCD}, while the timing on device-level is zero, i.e., there exists no timing constraint for inter-bank `ACT` and `RD` commands.

This code describes the whole DDR3 DRAM in a machine-readable and easy-to-modify way.

Listing A.1: DRAMml Describing DDR3

```
1  DRAM {
2     Name = "DDR3-800D";
3     B = 8;
4     N = 4;
5
6     Device {
7        Places {
8           IDLE (B);
9           PDNP;
10          SREF;
11          PDNA;
12       }
13
14       Transitions {
15          REFA;
16          PREA;
17          PDEP;
18          PDXP;
19          SREFEN;
20          SREFEX;
21          PDEA;
22          PDXA;
23       }
24
25       Arcs {
26          // Normal Arcs:
27          IDLE   -> REFA   (Weight = B);
28          REFA   -> IDLE   (Weight = B);
29          PREA   -> IDLE   (Weight = B);
30          IDLE   -> PDEP   (Weight = B);
31          PDEP   -> PDNP;
32          PDNP   -> PDXP;
33          PDXP   -> IDLE   (Weight = B);
34          IDLE   -> SREFEN (Weight = B);
35          SREFEN -> SREF;
36          SREF   -> SREFEX;
37          SREFEX -> IDLE   (Weight = B);
38          PDEA   -> PDNA;
39          PDNA   -> PDXA;
40
41          IDLE -> ACT;
42          RDA  -> IDLE;
43          WRA  -> IDLE;
44          PRE  -> IDLE;
45
```

```
46          // Inhibitor Arcs:
47          PDNA -o REFA;
48          PDNA -o PREA;
49          PDNP -o PREA;
50          SREF -o PREA;
51          PDNA -o PDEP;
52          PDNA -o SREFEN;
53          PDNA -o PDEA;
54          PDNP -o PDEA;
55          SREF -o PDEA;
56          IDLE -o PDEA (Weight = B);
57
58          PDNA -o ACT;
59          PDNA -o RD;
60          PDNA -o WR;
61          PDNA -o PRE;
62          PDNA -o RDA;
63          PDNA -o WRA;
64
65          // Reset Arcs:
66          ACTIVE ->> PREA;
67          IDLE   ->> PREA;
68       }
69
70       // Define 8 Banks:
71       B : Bank {
72          Places {
73             ACTIVE;
74          }
75
76          Transitions {
77             ACT;
78             RD;
79             RDA;
80             PRE;
81             WR;
82             WRA;
83          }
84
85          Arcs {
86             // Normal Arcs:
87             ACT    -> ACTIVE;
88             ACTIVE -> RD;
89             RD     -> ACTIVE;
90             ACTIVE -> RDA;
91             ACTIVE -> WR;
92             WR     -> ACTIVE;
93             ACTIVE -> WRA;
94             ACTIVE -> PRE;
95
96             // Inhibitor Arcs:
97             ACTIVE -o ACT;
98          }
99       }
100   }
101
102   // Define Timing Constraints:
103   TimingConstraints {
104      Timings {
105         tCK     = 2.5;
106         tCCD    = 10;
107         tRCD    = 12.5;
108         tRP     = 12.5;
```

```
109         tRAS    = 37.5;
110         tRL     = 5*tCK;
111         tWL     = 5*tCK;
112         tRTP    = 4*tCK;
113         tWTR    = 4*tCK;
114         tRRD    = 4*tCK;
115         tWR     = 15;
116         tFAW    = 40;
117         tRFC    = 110;
118         tREFI   = 7800;
119         tREFMAX = 9*tREFI;
120
121         tRC     = tRAS + tRP;
122         tRDWR   = tRL + tCCD + 2*tCK - tWL;
123         tWRRD   = tWL + tCCD + tWTR;
124         tRDAACT = tRTP + tRP;
125         tWRPRE  = tWL + tCCD + tWR;
126         tWRAACT = tWRPRE + tRP;
127
128         tXP     = 3*tCK;
129         tXS     = tRFC + 10;
130         tXSDLL  = 512*tCK;
131         tCKE    = 3*tCK;
132         tCKESR  = tCKE + tCK;
133         tPD     = tCKE;
134         tRDPDEN = tRL + 5*tCK;
135         tWRPDEN = tWL + 4*tCK + tWR;
136         tWRAPDEN= tWL + 5*tCK + tWR;
137         tREFPDEN= tCK;
138         tACTPDEN= tCK;
139         tPRPDEN = tCK;
140      }
141
142      Places {
143         CMD_BUS;
144         NAW_POOL (N);
145      }
146
147      Arcs {
148         // First timing for intra-bank
149         // Second timing for inter-bank
150         // Timing constraints from ACT
151         ACT -<> PRE  (tRAS,0);
152         ACT -<> RD   (tRCD,0);
153         ACT -<> WR   (tRCD,0);
154         ACT -<> RDA  (tRCD,0);
155         ACT -<> WRA  (tRCD,0);
156         ACT -<> ACT  (tRC,tRRD);
157         ACT -<> PDEA (0,tACTPDEN);
158         ACT -<> REFA (0,tRC);
159         ACT -<> PREA (0,tRAS);
160
161         // Timing constraints from RD/RDA
162         RD  -<> PRE    (tRTP,0);
163         RD  -<> PREA   (0,tRTP);
164         RD  -<> PDEA   (0,tRDPDEN);
165         RD  -<> PDEP   (0,tRDPDEN);
166         RDA -<> PDEA   (0,tRDPDEN);
167         RDA -<> PDEP   (0,tRDPDEN);
168         RD  -<> RD     (tCCD,tCCD);
169         RD  -<> RDA    (tCCD,tCCD);
170         RDA -<> RD     (0,tCCD);
171         RDA -<> RDA    (0,tCCD);
```

```
172         RD  -<> WR      (tRDWR,tRDWR);
173         RD  -<> WRA     (tRDWR,tRDWR);
174         RDA -<> WR      (0,tRDWR);
175         RDA -<> WRA     (0,tRDWR);
176         RDA -<> ACT     (tRDAACT,0);
177         RDA -<> REFA    (0,tRTP+tRP);
178         RDA -<> PREA    (0,tRTP);
179         RDA -<> SREFEN  (0,tRDPDEN);
180
181         // Timing constraints from WR/WRA
182         WR  -<> PRE     (tWRPRE,0);
183         WR  -<> PREA    (0,tWRPRE);
184         WR  -<> PDEA    (0,tWRPDEN);
185         WRA -<> PDEA    (0,tWRAPDEN);
186         WRA -<> PDEP    (0,tWRAPDEN);
187         WR  -<> WR      (tCCD,tCCD);
188         WR  -<> WRA     (tCCD,tCCD);
189         WRA -<> WR      (0,tCCD);
190         WRA -<> WRA     (0,tCCD);
191         WR  -<> RD      (tWRRD,tWRRD);
192         WR  -<> RDA     (tWRRD,tWRRD);
193         WRA -<> RD      (0,tWRRD);
194         WRA -<> RDA     (0,tWRRD);
195         WRA -<> ACT     (tWRAACT,0);
196         WRA -<> REFA    (0,tWRPRE+tRP);
197         WRA -<> PREA    (0,tWRPRE);
198         WRA -<> SREFEN  (0,tWRPRE+tRP);
199
200         // Timing constraints from PRE/PREA
201         PRE  -<> ACT    (tRP,0);
202         PRE  -<> REFA   (0,tRP);
203         PRE  -<> PDEA   (0,tPRPDEN);
204         PRE  -<> PDEP   (0,tPRPDEN);
205         PRE  -<> SREFEN (0,tRP);
206         PREA -<> ACT    (0,tRP);
207         PREA -<> REFA   (0,tRP);
208         PREA -<> PDEP   (0,tPRPDEN);
209         PREA -<> SREFEN (0,tRP);
210
211         // Timing constraints from PDN
212         PDEP -<> PDXP   (0,tPD);
213         PDEA -<> PDXA   (0,tPD);
214         PDXA -<> PDEA   (0,tCKE);
215         PDXP -<> PDEP   (0,tCKE);
216         PDXP -<> REFA   (0,tXP);
217         PDXP -<> SREFEN (0,tXP);
218         PDXA -<> ACT    (0,tXP);
219         PDXP -<> ACT    (0,tXP);
220         PDXA -<> PRE    (0,tXP);
221         PDXA -<> PREA   (0,tXP);
222         PDXA -<> RD     (0,tXP);
223         PDXA -<> RDA    (0,tXP);
224         PDXA -<> WR     (0,tXP);
225         PDXA -<> WRA    (0,tXP);
226
227         // Timing constraints from REFA/SREF
228         REFA   -<> ACT    (0,tRFC);
229         REFA   -<> REFA   (0,tRFC);
230         REFA   -<> PREA   (0,tRFC);
231         REFA   -<> SREFEN (0,tRFC);
232         REFA   -<> PDEP   (0,tREFPDEN);
233         SREFEX -<> ACT    (0,tXS);
234         SREFEX -<> REFA   (0,tXS);
```

```
235             SREFEX -<> PDEP   (0,tXS);
236             SREFEX -<> SREFEN (0,tXS);
237             SREFEX -<> RD     (0,tXSDLL);
238             SREFEX -<> RDA    (0,tXSDLL);
239             SREFEX -<> WR     (0,tXSDLL);
240             SREFEX -<> WRA    (0,tXSDLL);
241             SREFEN -<> SREFEX (0,tCKESR);
242
243             // Arcs for the CLK
244             CMD_BUS -> * [tCK, inf];
245             * -> CMD_BUS;
246
247             // Arcs for the NAW
248             NAW_POOL -> ACT [tFAW, inf];
249             ACT      -> NAW_POOL;
250
251             // Arcs for the Refresh
252             REFA   -<> WARNING (0,tREFMAX);
253             SREF   -o  WARNING;
254             SREFEX -<> WARNING (0,tREFMAX);
255         }
256     }
257 }
```

Acronyms

ADMM	alternating direction method of multipliers. 8, 33–35, 40, 43, 44, 46, 48–52, 81
AED	Automorphism Ensemble Decoding. 8, 22, 23, 28
APP	a posteriori probability. 19
ASIC	application specific integrated circuit. 2
AWGN	additive white Gaussian noise. 10, 23, 28, 41
B&B	Branch-and-Bound. 43–46, 48–53, 81
BAC	binary asymmetric channel. 13
BCH	Bose-Chaudhuri-Hocquenghem. 52, 53
BEC	Binary Erasure Channel. 23
BER	Bit Error Rate. 12
BP	Belief Propagation. 8, 17–23, 26–31, 34, 44, 52, 81, 82, 109
BPSK	Binary Phase Shift Keying. 10, 28
BSC	binary symmetric channel. 13, 14, 72, 73
CCSDS	Consultative Committee for Space Data Systems. 31, 109
CN	Check Node. 18–20, 23
CPU	central processing unit. 58
CSA	cut-search algorithm. 35–37
DBI	data bus inversion. 9, 69, 75–79, 81, 110
DDR	double data rate. 60, 63, 66, 69, 71, 75, 76, 81, 82
DIMM	Dual Inline Memory Module. 9, 73
DRAM	dynamic random-access memory. 6, 7, 9, 13, 57–61, 63–65, 67–76, 81, 82, 91, 92, 110, 111

DSL domain-specific language. 57, 68, 91

ECC Error Correction Coding. 1, 6, 7, 9, 57, 58, 69, 73–79, 81, 82, 110
ESF Extrinsic Scaling Factor. 20, 28, 30

FEC Forward Error Correction. 1, 5, 18
FER Frame Error Rate. 8, 12, 28, 30, 31, 109

IC integrated circuit. 2
ILP integer linear programming. 5, 11, 17, 24, 25, 33, 44

JEDEC Joint Electron Devices Engineering Councils. 9, 57, 59–61, 63, 67, 68

KPI Key Performance Indicator. 2, 4, 5, 7, 43, 69, 111

LDPC low-density parity-check. 8, 18, 20–22, 31, 41
LLR Log Likelihood Ratio. 11, 19, 23, 30, 48, 51, 82
LP linear programming. 5–8, 33–35, 40, 44, 46, 48–52, 81
LTE Long-Term Evolution. 22, 26, 28, 29, 109, 111

MAP Maximum a posteriori. 30
MBBP Multiple-Bases Belief-Propagation. 8, 21–23, 26–28
ML maximum likelihood. 5–8, 11–13, 21, 22, 30, 33, 43, 44, 46, 52, 53, 81, 82
MS Min-Sum. 20

RM Reed-Muller. 18, 22
RS Reed-Solomon. 44, 52, 53

SED/DED single error correction / double error detection. 73–76, 78
SNR signal-to-noise ratio. 10, 12, 27, 41, 43, 53
SPA Sum-Product Algorithm. 19, 20

VN Variable Node. 18, 19, 23

Bibliography

[1] Ericsson. Ericsson mobility report. https://www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf, November 2022.

[2] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[3] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[4] K. Rupp. Microprocessor Trend Data. https://github.com/karlrupp/microprocessor-trend-data. Accessed: 2023-03-23.

[5] M. Herrmann, C. Kestel, and N. Wehn. Energy efficient FEC decoders. In *2021 11th International Symposium on Topics in Coding (ISTC)*, pages 1–5, 2021.

[6] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[7] T. Hehn, J. B. Huber, S. Laendner, and O. Milenkovic. Multiple-bases belief-propagation for decoding of short block codes. In *2007 IEEE International Symposium on Information Theory*, pages 311–315. IEEE, 2007.

[8] T. Hehn, J. B. Huber, O. Milenkovic, and S. Laendner. Multiple-bases belief-propagation decoding of high-density cyclic codes. *IEEE transactions on communications*, 58(1):1–8, 2010.

[9] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink. Automorphism ensemble decoding of Reed–Muller codes. *IEEE Transactions on Communications*, 69(10):6424–6438, 2021.

[10] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink. On the automorphism group of polar codes. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1230–1235. IEEE, 2021.

[11] C. Pillet, V. Bioglio, and I. Land. Polar codes for automorphism ensemble decoding. In *2021 IEEE Information Theory Workshop (ITW)*, pages 1–6. IEEE, 2021.

[12] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, Jan. 2011.

[13] X. Zhang and P. H. Siegel. Efficient iterative LP decoding of LDPC codes with alternating direction method of multipliers. In *2013 IEEE International Symposium on Information Theory*, pages 1501–1505, July 2013.

[14] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn. Database of channel codes and ML simulation results. www.uni-kl.de/channel-codes, 2019.

[15] M. Jung, K. Kraft, and N. Wehn. A new state model for DRAMs using Petri nets. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 221–226. IEEE, 2017.

[16] K. Kraft, M. Herrmann, O. Griebel, and N. Wehn. Ensemble belief propagation decoding for short linear block codes. In *WSA & SCC 2023; 26th International ITG Workshop on Smart Antennas and 13th Conference on Systems, Communications, and Coding*, pages 1–6, 2023.

[17] F. Gensheimer, T. Dietz, S. Ruzika, K. Kraft, and N. Wehn. Improved maximum-likelihood decoding using sparse parity-check matrices. In *2018 25th International Conference on Telecommunications (ICT)*, pages 236–240. IEEE, 2018.

[18] F. Gensheimer, T. Dietz, S. Ruzika, K. Kraft, and N. Wehn. A low-complexity projection algorithm for ADMM-based LP decoding. In *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, pages 1–5. IEEE, 2018.

[19] F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn. A reduced-complexity projection algorithm for ADMM-based LP decoding. *IEEE Transactions on Information Theory*, 66(8):4819–4833, 2020.

[20] K. Kraft and N. Wehn. ADMM-based ML decoding: From theory to practice. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8278–8282. IEEE, 2021.

[21] M. Jung, K. Kraft, T. Soliman, C. Sudarshan, C. Weis, and N. Wehn. Fast validation of DRAM protocols with timed Petri nets. In *Proceedings of the International Symposium on Memory Systems*, pages 133–147, 2019.

[22] K. Kraft, D. M. Mathew, C. Sudarshan, M. Jung, C. Weis, N. Wehn, and F. Longnos. Efficient coding scheme for DDR4 memory subsystems. In *Proceedings of the International Symposium on Memory Systems*, pages 148–157, 2018.

[23] K. Kraft, C. Sudarshan, D. M. Mathew, C. Weis, N. Wehn, and M. Jung. Improving the error behavior of DRAM by exploiting its Z-channel property. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1492–1495. IEEE, 2018.

[24] L. Steiner, K. Kraft, D. Uecker, M. Jung, M. Huonker, and N. Wehn. An LPDDR4 safety model for automotive applications. In *The International Symposium on Memory Systems*, pages 1–9, 2021.

[25] K. Kraft, D. M. Mathew, C. Sudarshan, M. Jung, C. Weis, N. Wehn, F. Longnos, L. Gezi, and W. Yang. Memory access technology and computer system for reducing data error probability, Dec. 6 2022. US Patent 11,521,674.

[26] J. Feldman and D. R. Karger. *Decoding Error-Correcting Codes via Linear Programming*. PhD thesis, USA, 2003. AAI0805681.

[27] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51(3):954–972, March 2005.

[28] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[29] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on selected areas in communications*, 16(2):140–152, 1998.

[30] G. D. Forney. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47(2):520–548, 2001.

[31] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.

[32] R. Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[33] D. J. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457, 1997.

[34] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547, 1981.

[35] C. Kestel, M. Herrmann, and N. Wehn. When channel coding hits the implementation wall. In *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Dec. 2018.

[36] B. J. Frey and D. J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In *Proceedings of the 10th International Conference on Neural Information Processing Systems*, NIPS'97, page 479–485, Cambridge, MA, USA, 1997. MIT Press.

[37] J. Jiang and K. R. Narayanan. Iterative soft decision decoding of Reed Solomon codes based on adaptive parity check matrices. In *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, page 261. IEEE, 2004.

[38] E. Nachmani, Y. Be'ery, and D. Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 341–346. IEEE, 2016.

[39] C. Pillet, V. Bioglio, and I. Land. Classification of automorphisms for the decoding of polar codes. In *ICC 2022-IEEE International Conference on Communications*, pages 110–115. IEEE, 2022.

[40] M. Geiselhart, M. Ebada, A. Elkelesh, J. Clausius, and S. ten Brink. Automorphism ensemble decoding of quasi-cyclic LDPC codes by breaking graph symmetries. *IEEE Communications Letters*, 2022.

[41] A. Zunker, M. Geiselhart, L. Johannsen, C. Kestel, S. ten Brink, T. Vogt, and N. Wehn. Row-merged polar codes: Analysis, design, and decoder implementation. *IEEE Transactions on Communications*, 73(1):39–53, 2025.

[42] N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.

[43] J. S. Yedidia, J. Chen, and M. P. Fossorier. Generating code representations suitable for belief propagation decoding. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 40, pages 447–456. The University; 1998, 2002.

[44] S. Sankaranarayanan. Iterative decoding of codes on graphs. 2006.

[45] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Inf. Theor.*, 43(6):1757–1766, Nov. 1997.

[46] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998.

[47] S. Weithoffer, C. A. Nour, N. Wehn, C. Douillard, and C. Berrou. 25 years of turbo codes: From Mb/s to beyond 100 Gb/s. In *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, pages 1–6. IEEE, 2018.

[48] Consultative Committee for Space Data Systems. TC Synchronization and Channel Coding. *Green Book*, 2021.

[49] F. Gensheimer, S. Ruzika, S. Scholl, and N. Wehn. ADMM versus simplex algorithm for LP decoding. In *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pages 211–215. IEEE, 2016.

[50] S. Barman, X. Liu, S. C. Draper, and B. Recht. Decomposition methods for large scale LP decoding. *IEEE Transactions on Information Theory*, 59(12):7870–7886, 2013.

[51] M. H. Taghavi N. and P. H. Siegel. Adaptive methods for linear programming decoding. *IEEE Transactions on Information Theory*, 54(12):5396–5410, 2008.

[52] M. Wasson and S. C. Draper. Hardware based projection onto the parity polytope and probability simplex. In *Proceedings of the 49th Asilomar Conference on Signals, Systems and Computers*, pages 1015–1020, Pacific Grove, California, USA, November 2015. IEEE.

[53] M. Wasson, M. Milicevic, S. C. Draper, and G. Gulak. Hardware-based linear program decoding with the alternating direction method of multipliers. *IEEE Transactions on Signal Processing*, 67(19):4976–4991, 2019.

[54] G. Zhang, R. Heusdens, and W. B. Kleijn. Large scale LP decoding with low complexity. *IEEE Communications Letters*, 17(11):2152–2155, November 2013.

[55] H. Wei and A. H. Banihashemi. An iterative check polytope projection algorithm for ADMM-based LP decoding of LDPC codes. *IEEE Communications Letters*, 22(1):29–32, January 2018.

[56] M. Helmling, E. Rosnes, S. Ruzika, and S. Scholl. Efficient maximum-likelihood decoding of linear block codes on binary memoryless channels. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2589–2593. IEEE, 2014.

[57] M. V. Natale, M. Jung, K. Kraft, F. Lauer, J. Feldmann, C. Sudarshan, C. Weis, S. Krumke, and N. Wehn. Efficient generation of application specific memory controllers. In *The International Symposium on Memory Systems*, MEMSYS 2020, page 233–247, New York, NY, USA, 2021. Association for Computing Machinery.

[58] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.

[59] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[60] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[61] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85 – 104, 1976.

[62] H. M. W. Verbeek, M. T. Wynn, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Reduction rules for reset/inhibitor nets. *J. Comput. Syst. Sci.*, 76(2):125–143, Mar. 2010.

[63] T. Agerwala and M. Flynn. Comments on capabilities, limitations and correctness of Petri nets. In *Proceedings of the 1st Annual Symposium on Computer Architecture*, ISCA '73, pages 81–86, New York, NY, USA, 1973. ACM.

[64] M. Hack. Petri net language. Technical report, Cambridge, MA, USA, 1976.

[65] L. Jacobsen, M. Jacobsen, M. H. Møller, and J. Srba. Verification of timed-arc Petri nets. In I. Černá, T. Gyimóthy, J. Hromkovič, K. Jefferey, R. Králović, M. Vukolić, and S. Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science*, pages 46–72, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[66] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, O. Naji, M. Jung, N. Wehn, and K. Goossens. DRAMPower: Open-source DRAM power & energy estimation tool. `http://www.drampower.info`.

[67] M. Jung, D. M. Mathew, E. F. Zulian, C. Weis, and N. Wehn. A new bank sensitive DRAM-Power model for efficient design space exploration. In *International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS 2016)*, 2016.

[68] D. M. Mathew, E. F. Zulian, S. Kannoth, M. Jung, C. Weis, and N. Wehn. A bank-wise DRAM power model for system simulations. In *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '17, pages 5:1–5:7, New York, NY, USA, 2017. ACM.

[69] M. Jung, D. M. Mathew, C. Weis, and N. Wehn. Approximate computing with partially unreliable dynamic random access memory - approximate DRAM. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 100:1–100:4, New York, NY, USA, 2016. ACM.

[70] M. Jung, E. F. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn. Omitting refresh - a case study for commodity and wide I/O DRAMs. In *1st International Symposium on Memory Systems (MEMSYS 2015)*, Washington, DC, USA, October 2015.

[71] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1):16–19, Jan 2011.

[72] M. Jung, C. Weis, and N. Wehn. DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework. *IPSJ Transactions on System LSI Design Methodology (T-SLDM)*, August 2015.

[73] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters*, PP(99):1–1, 2015.

[74] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[75] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udipi. Simulating DRAM controllers for future system architecture exploration. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 201–210, March 2014.

[76] C. Sudarshan, J. Lappas, C. Weis, D. M. Mathew, M. Jung, and N. Wehn. A lean, low power, low latency DRAM memory controller for transprecision computing. In *Embedded Computer Systems: Architectures, Modeling, and Simulation: 19th International Conference, SAMOS 2019, Samos, Greece, July 7–11, 2019, Proceedings 19*, pages 429–441. Springer, 2019.

[77] S.-L. Lu. Tutorial: DRAM - Circuits, Organization, Interfaces. In *IEEE Microarchitecture Conference (MICRO)*, 2016.

[78] S. K. Kurinec and K. Iniewski. *Nanoscale semiconductor memories: technology and applications*. CRC press, 2013.

[79] S. M. Seyedzadeh, D. Kline Jr, A. K. Jones, and R. Melhem. Mitigating bitline crosstalk noise in dram memories. In *Proceedings of the International Symposium on Memory Systems*, pages 205–216, 2017.

[80] Z. Yang and S. Mourad. Crosstalk in deep submicron drams. In *Memory Technology, Design and Testin, IEEE International Workshop on*, pages 125–125. IEEE Computer Society, 2000.

[81] M. Jung, D. M. Mathew, C. C. Rheinländer, C. Weis, and N. Wehn. A platform to analyze ddr3 dram's power and retention time. *IEEE Design & Test*, 34(4):52–59, 2017.

[82] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[83] M.-Y. Hsiao. A class of optimal minimum odd-weight-column SEC-DED codes. *IBM Journal of Research and Development*, 14(4):395–401, 1970.

[84] T. J. Dell. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics Division*, 11, 1997.

[85] H. J. Kwon, E. Seo, C. Y. Lee, Y. H. Seo, G. H. Han, H. R. Kim, J. H. Lee, M. S. Jang, S. G. Do, S. H. Cho, J. K. Park, S. Y. Doo, J. B. Shin, S. H. Jung, H. J. Kim, I. H. Im, B. R. Cho, J. W. Lee, J. Y. Lee, K. H. Yu, H. K. Kim, C. H. Jeon, H. S. Park, S. S. Kim, S. H. Lee, J. W. Park, S. S. Lee, B. T. Lim, J. y. Park, Y. S. Park, H. J. Kwon, S. J. Bae, J. H. Choi, K. I. Park, S. J. Jang, and G. Y. Jin. 23.4 An extremely low-standby-power 3.733Gb/s/pin 2Gb LPDDR4 SDRAM for wearable devices. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 394–395, Feb 2017.

[86] C. K. Lee, Y. J. Eom, J. H. Park, J. Lee, H. R. Kim, K. Kim, Y. Choi, H. J. Chang, J. Kim, J. M. Bang, S. Shin, H. Park, S. Park, Y. R. Choi, H. Lee, K. H. Jeon, J. Y. Lee, H. J. Ahn, K. H. Kim, J. S. Kim, S. Chang, H. R. Hwang, D. Kim, Y. H. Yoon, S. H. Hyun, J. Y. Park, Y. G. Song, Y. S. Park, H. J. Kwon, S. J. Bae, T. Y. Oh, I. D. Song, Y. C. Bae, J. H. Choi, K. I. Park, S. J. Jang, and G. Y. Jin. 23.2 A 5Gb/s/pin 8Gb LPDDR4X SDRAM with power-isolated LVSTL and split-die architecture with 2-die ZQ calibration scheme. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 390–391, Feb 2017.

[87] J. M. Berger. A note on error detection codes for asymmetric channels. *Information and control*, 4(1):68–73, 1961.

[88] JEDEC. JESD79-4B, JEDEC Standard, DDR4 SDRAM. 2017.

[89] H. To, C. Su, J. Wang, D. Klokotov, L. Zhu, J. Schmitz, P. Niu, and Y. Wang. Optimal ddr4 system with data bus inversion. In *DesignCon 2016*, 2016.

[90] S. Scholl, F. Kienle, M. Helmling, and S. Ruzika. Integer programming as a tool for analysis of channel codes. In *SCC 2013; 9th International ITG Conference on Systems, Communication and Coding*, pages 1–6, 2013.

[91] S. Cammerer, M. Ebada, A. Elkelesh, and S. ten Brink. Sparse graphs for belief propagation decoding of polar codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1465–1469, 2018.

List of Figures

List of Tables

List of Algorithms

Lebenslauf

Persönliche Daten

Name: Kira Kraft

Ausbildung

Hochschulstudium:

Master Mathematik
Anwendungsfach: Informatik
Studienschwerpunkt: Algebra und Zahlentheorie
Technische Universität Kaiserslautern
2014–2016

Bachelor Mathematik
Anwendungsfach: Informatik
Vertiefung: Algebra, Geometrie und Computeralgebra
Technische Universität Kaiserslautern
2010–2014

Beruflicher Werdegang

01/2017 - 03/2023

wissenschaftlicher Mitarbeiter
Arbeitsgruppe Entwurf Mikroelektronischer Systeme
Fachbereich Elektrotechnik und Informationstechnik
Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau