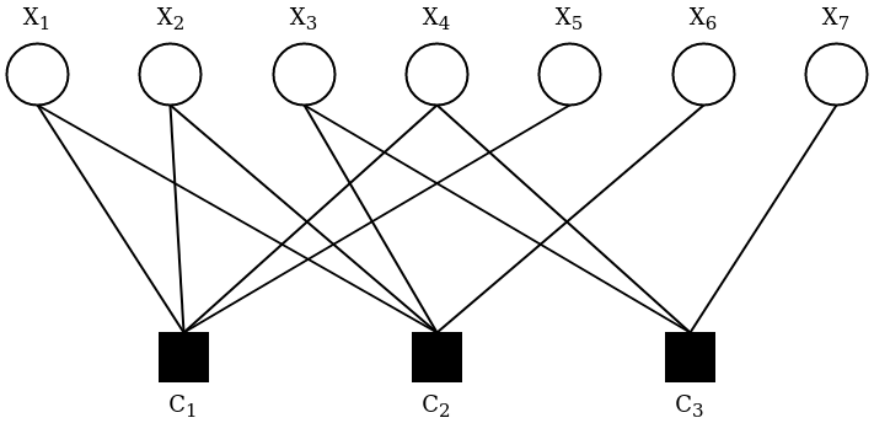


$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Error Correction in NAND Flash Memories Using Low-Density Parity-Check Codes

An investigation of how low-density parity-check codes can be constructed to efficiently correct errors due to noise mechanims in NAND flash memories.

Master's thesis in Complex Adaptive Systems

Fredrik Blomgren, Adrian Lundell

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Error Correction in NAND Flash Memories Using Low-Density Parity-Check Codes

An investigation of how low-density parity-check codes can be constructed to efficiently correct errors due to noise mechanims in NAND flash memories.

Fredrik Blomgren, Adrian Lundell

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communication Systems
Chalmers University of Technology
Gothenburg, Sweden 2022

Error Correction in NAND Flash Memories Using Low-Density Parity-Check Codes
An investigation of how low-density parity-check codes can be constructed to efficiently correct errors due to noise mechanims in NAND flash memories.
Fredrik Blomgren, Adrian Lundell

Supervisor: Jonas Källén, Cobham Gaisler AB
Advisor: Yingqi Zhang, Division of Antenna Systems, Department of Electrical Engineering
Examiner: Alexandre Graell I Amat, Division of Communication Systems, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Communication Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Tanner graph visualizing a low-density parity-check code along with parity-check matrix H.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Error Correction in NAND Flash Memories Using Low-Density Parity-Check Codes
An investigation of how low-density parity-check codes can be constructed to efficiently correct errors due to noise mechanims in NAND flash memories.
Fredrik Blomgren, Adrian Lundell
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In this work, several low-density parity-check codes (LDPC codes) of quasi-cyclic structure are designed for error correction of information stored in a NAND flash memory, achieving a maximum decoded bit error rate around $2 \cdot 10^{-8}$ for a 1% raw bit error rate for a (9, 73, 256) code of code rate 0.88 over the BI-AWGN channel. This is accomplished by optimizing the code ensemble using density evolution to achieve a good waterfall threshold and minimizing trapping sets with a progressive edge-growth algorithm to lower the error floor of the decoding performance curve. The software tools developed is available in three Python scripts for code design and and a modified `C++` library for simulation.

Emphasis is put on providing comprehensive understanding and reasoning regarding the full design chain from hardware, to channel model, to the details of the used algorithms. Additionally, it is shown how issues more specific to the NAND-flash memory can be analyzed using the tools to provide a basis for performance-complexity design decisions. The performance gain of higher readout precision is analyzed using the n-bit discretized BI-AWGN channel, the impact of asymmetric errors is demonstrated, and full Sum-Product decoding is compared to an approximate Min-Sum algorithm. As a conclusion to the thesis multiple improvements and suggestions on future work are given.

Keywords: Quasi-cyclic low-density parity-check codes, QC-LDPC, NAND flash memory, density evolution, progressive edge growth, PEG

Acknowledgements

First and foremost, we would like to thank our supervisor at Cobham Gaisler AB, Jonas Källén, for his his guidance throughout this study. Without him, this project would not have been possible and his ambitious involvement in our work contributed to the progress significantly. Furhermore, we would like to thank Yinqi Zhang for her valuable input during the project. We would also like to express our appreciation towards family and friends for their support, councel and encouragement.

Fredrik Blomgren, Adrian Lundell, Gothenburg, August, 2022

List of Acronyms

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BI-AWGN	Binary Input Additive White Gaussian Noise
BL	Bit Line
BP	Belief Propagation
BSC	Binary Symmetric Channel
BSL	Bitline Select Line
CB	Central Bit
CDF	Cumulative Distribution Function
DMC	Discrete Memoryless Channel
ECC	Error Correction Codes
EOL	End of Lifetime
GSL	Grounded Select Line
GND	Ground
ISPP	Incremental Step Programming Pulse
LDPC	Low-Density Parity-Check
LLR	Logarithmic Likelihood Ratio
LSB	Least Significant Bit
MLC	Multi-Level Cell
MM-QC-PEGA	Mult-Edge Metric-Constrained Quasi-Cyclic Progressive Edge-Growth Algorithm
MSB	Most Significant Bit
NVM	Non-volatile Memory
PDF	Probability Density Function
P/E	Program and Erase
PCM	Phase-change Memory
QC	Quasi-Cyclic
RBER	Raw Bit Error Rate
ReRAM	Resistive Random-access Memory
TLC	Triple-Level Cell
SLC	Single-Level Cell
SNR	Signal to Noise Ratio
SRAM	Static Random-access memory
SPA	Sum-Product Algorithm
WL	Word Line

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Sets

$\mathbb{Z}_2$	The set of bits
$\mathbb{Z}_2^n$	The set of n-length vectors with elements in $\mathbb{Z}_2$
$\mathcal{N}\{v\}$	Set of check nodes directly connected to variable node, v, in a Tanner graph
$\mathcal{X}, \mathcal{Y}$	Set of stochastic outcomes of X and Y.

Parameters

μ	General noise parameters
$P(X)$	Probability density function
ϵ	Error rate, bit flip probability
σ	Variance
S	Signal level
T	Transition matrix
φ	Gaussian PDF
Φ	Gausian CDF
$H(Y)$	Entropy
$MI(X,Y)$	Mutual information
C	Capacity
G	Generator matrix
H	Parity-check matrix
(m,n,N)	QC-LDPC code with m rows, n columns and scaling factor N
$\mathcal{G}$	Tanner graph

R	Code rate
r	MM-QC-PEGA search depth
g	local girth
P, Λ	Node degree distributions
ρ, λ	Edge perspective node degree distributions
$\mathcal{C}$	Channel
η	Skewness
d_v	Number of ones in columns of parity-check matrix, H.
d_c	Number of ones in rows of parity-check matrix, H.
$P^{(l)}, p^{(l)}$	CDF and PDF of the all-one codeword message from variable node to check node at iteration l of density evolution.
$Q^{(l)}, q^{(l)}$	CDF and PDF of the all-one codeword message from check node to variable node at iteration l of density evolution.
$BER^{(l)}$	Bit error rate at iteration l of density evolution.
$CBP^{(l)}$	Chernoff bound at iteration l of density evolution.

Variables

x, y	Single input/output bit
$\boldsymbol{x}, \boldsymbol{y}$	Input/output vector
X, Y	Single stochastic input/output bit
$\boldsymbol{X}, \boldsymbol{Y}$	Stochastic input/output vector
$\hat{\boldsymbol{x}}$	Estimated input
i, j	Indices
c	Check node index
v	Variable node index
$m_{c\to v}^{(l)}$	Message from check node c to variable node, v, in iteration l
$m_{ch\to v}$	Initial message from channel to variable node, v.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

NAND flash memories have since their invention in the '80s found a solid place in modern applications, ranging from solid state drives to cell phones, due to their relatively high reliability at a low cost, low power usage, and fast operational speeds. As a result of ever-increasing demands on memory storage devices, the requirements for keeping the reliability high have, however, been steadily increasing. Smaller and cheaper components result in larger quality variances, while new ways of storing more information per area using *multi-level cells* cut error margins drastically[1]. Without improved techniques for correcting information errors, these trends would drastically reduce the lifetime and reliability of NAND flash memories, and particularly for devices in hostile environments, e.g. space-related applications, this need is even more evident. The Swedish embedded computer systems company Cobham Gaisler AB, which specialize components for such environments, is currently developing a new NAND flash memory controller for which this research is therefore highly relevant.

Error correction codes (ECC) approach the reliability problem by encoding data before storing it and then utilizing this encoding to decode the noisy data back to the original data. The subject of this thesis, the special class of low-density parity-check codes (LDPC codes), has specifically been found to perform close to the theoretical limit on how much noise an optimal encoding can tolerate, making them an interesting topic of study. At the same time, the performance comes at the cost of high-complexity decoding algorithms and a non-trivial design process. Algorithmic complexity translates to slower, more expensive, hardware components and higher power consumption and thus, important considerations when applying LDPC codes are how to design them to leverage the most performance gain and how much complexity can be afforded for the improved performance.

1.2 Aim

The aim of this project is to initiate the potential future usage of LDPC codes in a NAND flash memory controller developed by Cobham Gaisler AB. This aim is approached by compiling a full theoretic framework from literature and implementing

it in a design chain which is thoroughly discussed. Targeted questions are both how to achieve good decoding, and how much reduced complexity-compromises affects the result. Codes tailored to the specific NAND flash memory are developed and simulated in different ways using the developed tools to answer these questions and to show the potential of LDPC codes.

1.3 Scope

The scope of this study has been limited to LDPC codes with a certain *quasi-cyclic* structure (QC-LDPC codes), designed with density evolution and a progressive edge-growth algorithm. The reasoning behind this choice is further discussed within the thesis.

The modeling of the memory is based on previous studies on the same topic and the choice of specific LDPC codes to study is based on the constraints set by the target memory which Cobham Gaisler is developing a memory controller for. Since modeling is not based on any real-world measurements, the study is to be seen mainly as a qualitative.

1.4 Thesis structure

This thesis is structured as follows:

Chapter 2 and 3 introduce general theory applied in the thesis. In chapter 2, the basic inner workings of NAND flash memories are presented along with the noise mechanisms they exhibit. In chapter 3, essential information theory and error correction coding concepts, needed to understand how errors over a transmission channel can be identified and corrected, are introduced. Emphasis is put on LDPC and QC-LDPC codes specifically.

In chapter 4, the information from the previous chapters is combined to introduce two communication channel models of the NAND flash memory, one for code design and one for simulation.

In chapter 5, the design strategy applied for designing good LDPC codes is presented. First, performance goals and constraints are presented, followed by descriptions of the methods used to reach those goals.

In chapter 6, results from the design algorithms and simulations of LDPC codes over the communication model are presented.

Lastly, in chapter 7, the results are analyzed to find some areas of improvement as well as promising topics for future research. The thesis is then concluded with a summary over what has been achieved.

2

NAND Flash Memory

The history of the NAND flash memory dates back to the ‘80s when it was first invented by a team at Toshiba, led by Fujio Masuoka. Around the same time, Masuoka also came up with another type of memory, called NOR flash. Both memories are non-volatile - meaning they do not need electricity to keep stored information intact - however, the popularity of NAND flash is significantly higher today than NOR flash. How come? While NOR flash has certain advantages, such as allowing random access to stored data, NAND flash is specialized in performing operations on large blocks of data in parallel [2]. This makes NAND flash well suited for secondary storage, which is long-term storage used to keep programs and data indefinitely for later retrieval [3].

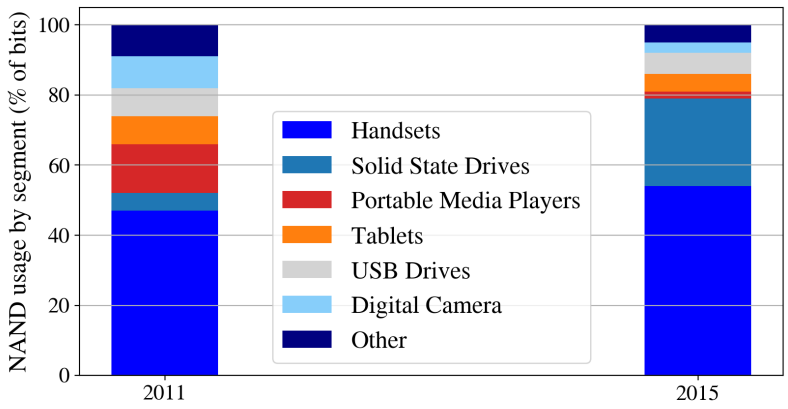


Figure 2.1: Change in application areas of NAND flash memories between 2011 and 2015 [4].

Attractive traits of memories used in secondary storage include low bit cost, high program and erase (P/E) speeds, low power consumption, and high reliability [2]. Modern NAND flash memories are cheap to produce and can reach a feature size of 4-6 nm, making it possible to pack a large number of bits in a small physical space [5], [6]. NAND flash memories also have a low power consumption and thus, NAND flash is cheap to operate as well [2]. These appealing traits of NAND flash have made them popular in many industrial applications such as solid-state drives, USB drives, and cell phones. The industry use cases of NAND flash memories are shown

in figure 2.1.

There are multiple competitors to NAND flash: phase-change memory (PCM), resistive random-access memory (ReRAM), and static random-access memory (SRAM) to name a few. These memories have attractive features which can be seen by studying table 2.1, and there has been an ongoing discussion on the replacement of NAND flash by ReRAM. The continuous scaling by introducing schemes to pack multiple bits of data into a single storage unit has, however, continued NAND flash placement as a strong mature memory technology [7].

The following sections aim to describe the basic functionality of NAND flash and the different noise mechanisms that come as a byproduct when performing different operations on NAND flash memories.

Table 2.1: Performance metrics of NAND flash memory and competitors [5], [6]

	NAND flash	SRAM	DRAM	PCM	ReRAM
Cell size (F^2)	4-6	120-200	6-10	4-30	≤ 2
Endurance (P/E)	$10^4 - 10^5$	$>10^{16}$	$>10^{16}$	$10^8 - 10^{15}$	$10^8 - 10^{12}$
Read time	14-35 μs	$\sim$ 1 ns	$\sim$ 110 ns	<10 ns	<10 ns
Write time	200-500 μs	$\sim$ 1 ns	$\sim$ 110 ns	50 ns	<10 ns
Leakage power	Low	Low	Medium	Low	Low

2.1 NAND flash architecture from the ground up

The lowest storage unit level of the NAND flash memory, commonly referred to as a **cell**, is a MOS transistor in which the stored data is represented by trapped electrons inside the transistor. This is made possible mainly through a control gate and a semiconductor crystalline layer that can store charge. Figure 2.2 illustrates the fundamental structure of a floating-gate NAND flash cell, where floating-gate refers to the crystalline layer used inside the cell. There exist different variants of the NAND flash cell, however, the essential and common characteristic of the different NAND flash cells is that the amount of electrons stored in the crystalline layer affects the threshold voltage needed at the control gate to open the transistor. The more electrons stored in the cell, the higher threshold voltage is needed to open the transistor [8].

The organization of memory cells plays an important part in the traits of NAND flash. Here, different architectural components are presented; the functionality of each component will become evident later in the chapter.

In a NAND flash memory, the cells are arranged horizontally and vertically along rows and columns. The drain and the source of cells are vertically coupled in series along **bit lines** (BL). The control gate of each cell is horizontally coupled in parallel along **word lines** (WL). Cells connected to the same WL are split into **pages** and cells that are connected to the same BL belong to the same **block** [10]. The

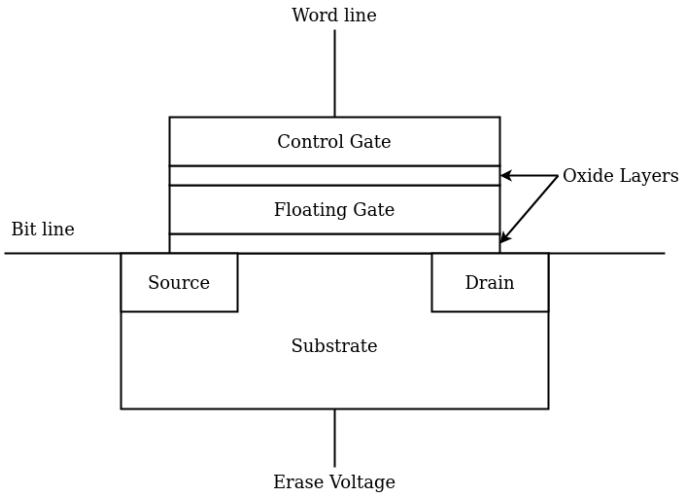


Figure 2.2: Basic structure of a floating-gate MOS transistor [9]

arrangement of cells in a block is illustrated in figure 2.3.

As can be seen in figure 2.3, a NAND flash memory also has a grounded select line (GSL) and a bitline select line (BSL). These are important when it comes to applying different operation modes. The blocks are further organized into **planes**, which in turn, make up a **die**. The die is the logical unit of the NAND flash memory [11]. In later developments of NAND flash memories, the cells have been arranged in 3D arrays such as the one in figure 2.4.

2.2 Basic operations

A NAND flash memory typically comes equipped with some input/output (I/O) control and control logic. Figure 2.5 illustrates an abstraction scheme of how these components communicate with the physical memory.

There are three basic operations that can be performed on a NAND Flash memory: **program**, **read** and **erase** [10].

- Program: The state of targeted cells is altered by adding charge to targeted cells. A page is also the smallest addressable unit for programming.
- Read: The read operation measures the charge levels inside a group of cells. At a minimum, this operation can be targeted to a page of cells.
- Erase: The state of targeted cells is changed to the default state by removing the charge from the group of cells. The smallest unit to erase is a block of cells.

In a NAND flash memory, simultaneous operations can be performed between planes and the die is the smallest unit that can independently execute commands and report status.

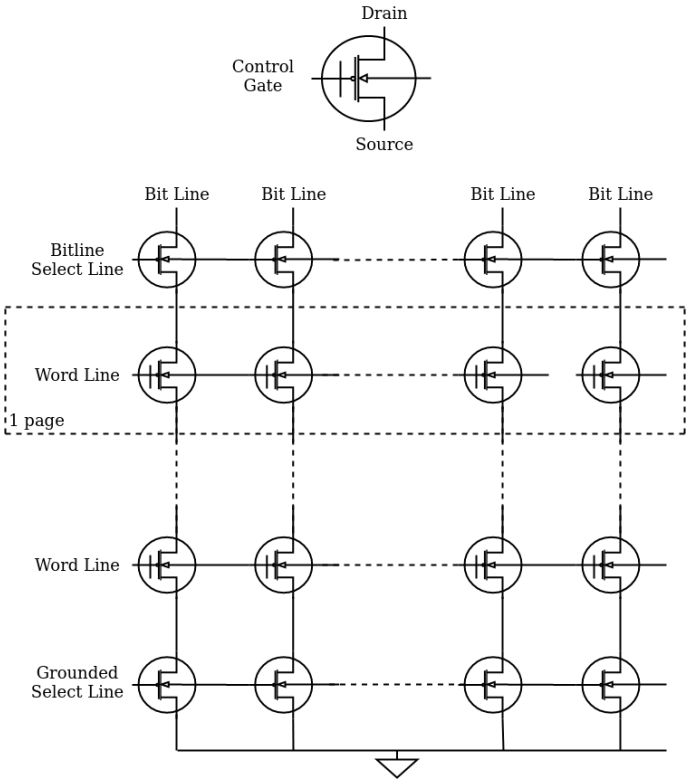


Figure 2.3: Arrangement of cells in a NAND flash memory. [10]

There exist many different techniques for operating a NAND flash memory and it is heavily dependent on how data is stored in the memory cell. In this study, the focus is on a common way of operating a NAND flash memory in which information is stored as **discrete voltage levels** inside the cells. An arbitrary amount of bits per cell can theoretically be stored using discrete voltage levels; however, most modern NAND flash memories store up to three bits per cell. Figure 2.6 illustrates how discrete voltage levels with corresponding bit values are stored inside a cell that stores one bit of information. The figure shows the probability density of a given voltage level with and without noise stemming from the noise mechanisms of the memory.

The number of bits stored inside a single cell is vital for both the functionality and the lifetime of a NAND flash memory. To easily separate memories depending on the number of bits stored inside a single cell, they have been given names based on the number of bits stored. A cell that stores 1 bit of information is called an **single-level cell** (SLC), a cell that stores 2 bits of information is called an **multi-level cell** (MLC) and a cell that stores 3 bits of information is called a **triple-level cell** (TLC). The three different types of cells is illustrated in figure 2.7.

When storing data as discrete voltage levels, the program operation translates to injecting charge into the cell so that the voltage required to open the cell is as close to the target level as possible. In a perfect world, inside an SLC, the voltage levels are set as either of the voltage levels in the left sub-figure of figure 2.6. However,

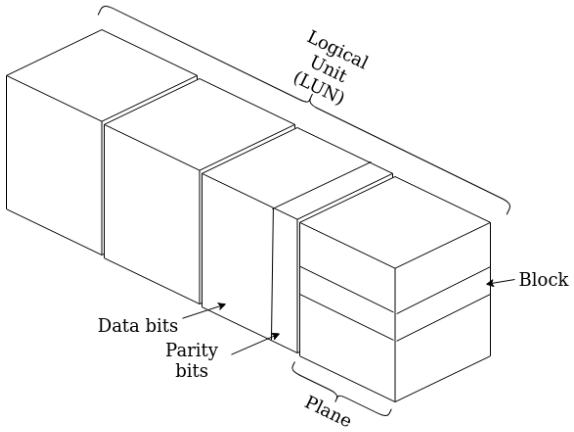


Figure 2.4: Illustration of a 3D memory array with some architectural components labeled [11]

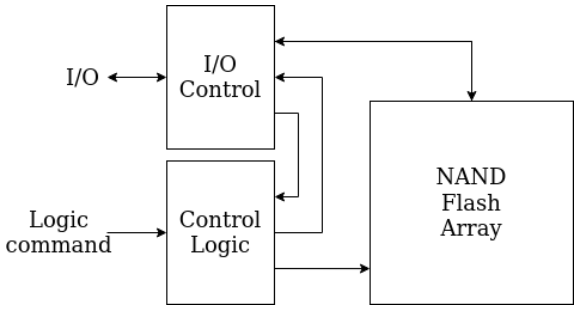


Figure 2.5: Simplified scheme of components that perform operations within a NAND flash memory. The control logic mainly performs operations such as read, program, and erase on the memory array, while the I/O control is responsible for the transfer of bits coming in and out from the NAND flash array [12]

in the real world, the programmed voltage levels are not accurate and reading a cell could result in a wrong read. Since it is most likely that error occur between adjacent voltage levels, it is desired to associate binary numbers to each level so that if adjacent levels are interchanged, only a single bit value is changed. For an SLC, this is simple because the only associated binary numbers are 0 and 1. For an MLC, on the other hand, it already becomes a bit complicated since there are four binary numbers involved. The task is then to associate a number to each voltage level so that there is only a difference of one bit value between adjacent levels. For example, the values 00 and 11 should not be placed next to one another. This concept is called **gray coding** [1] and it is shown how this is done for an MLC and a TLC in figure 2.7.

The read operation is performed based on certain read **thresholds** when storing data as discrete voltage levels. These thresholds are represented by the dashed vertical lines in figures 2.6 and 2.7. For an SLC, the read operation is simple since it only contains one read threshold. For an MLC, on the other hand, the sequence in which thresholds are read becomes important. A common way to read an MLC

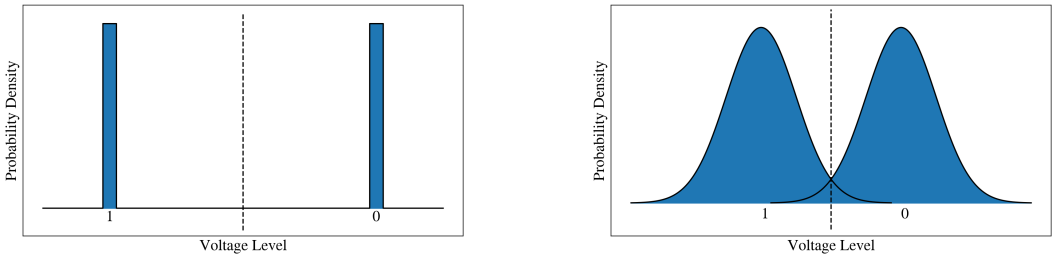


Figure 2.6: Illustration of discrete voltage levels in a SLC. The left figure shows the discrete voltages without any noise added, i.e. the voltages are modeled as Dirac delta functions. The right figure shows the discrete voltages when noise is added. In this case, the noise is modeled as a Gaussian distribution, which in turn causes the distribution of the discrete voltages to become Gaussian.

is to split up the two associated bits of each voltage level into **most significant bit** (MSB) and **least significant bit** (LSB). In figure 2.7, MSBs are written in green color while LSBs is written in red. To read the MSB, only the middle threshold is read since this is where the MSB bit value is flipped. Analogously, to read the LSB, the first and the last threshold are read. The same concept applies to a TLC, with the difference that an additional bit is added. Each additional bit increases the complexity of the read operation further [13].

To gain more information regarding the state of the cells one can add more thresholds to the given read reference level **threshold**. How additional thresholds are added to an SLC is illustrated in figure 2.8. The additional bit resulting from adding thresholds to an SLC should not be associated with the bits of an MLC. How this additional information regarding the state of the cell is utilized is explained in the next chapter.

The erase operation is the simplest case since it only translates to setting all the cells in a block to the lowest voltage level.

2.2.1 Program, read and erase on a technical level

When performing the program operation, charge is injected to the cells by applying a high voltage, V_{prog}, at the WL of the programmed page. All other WLs are set to V_{pass}, which is large enough to keep the cells at the other WLs open. Since it is difficult to control the amount of charge injected into the cells, the program operation is typically performed iteratively in steps until the desired discrete level is reached. This algorithm is called Incremental Step Programming Pulse (ISPP) and it guarantees that the written voltage level to the cells will be lower bounded by the desired voltage level [14]. The program operation is illustrated in figure 2.9.

The read operation is done by applying a voltage, V_{read}, at the WL of the page that is being read and V_{pass} at the other WLs. V_{read} can be any of the thresholds seen

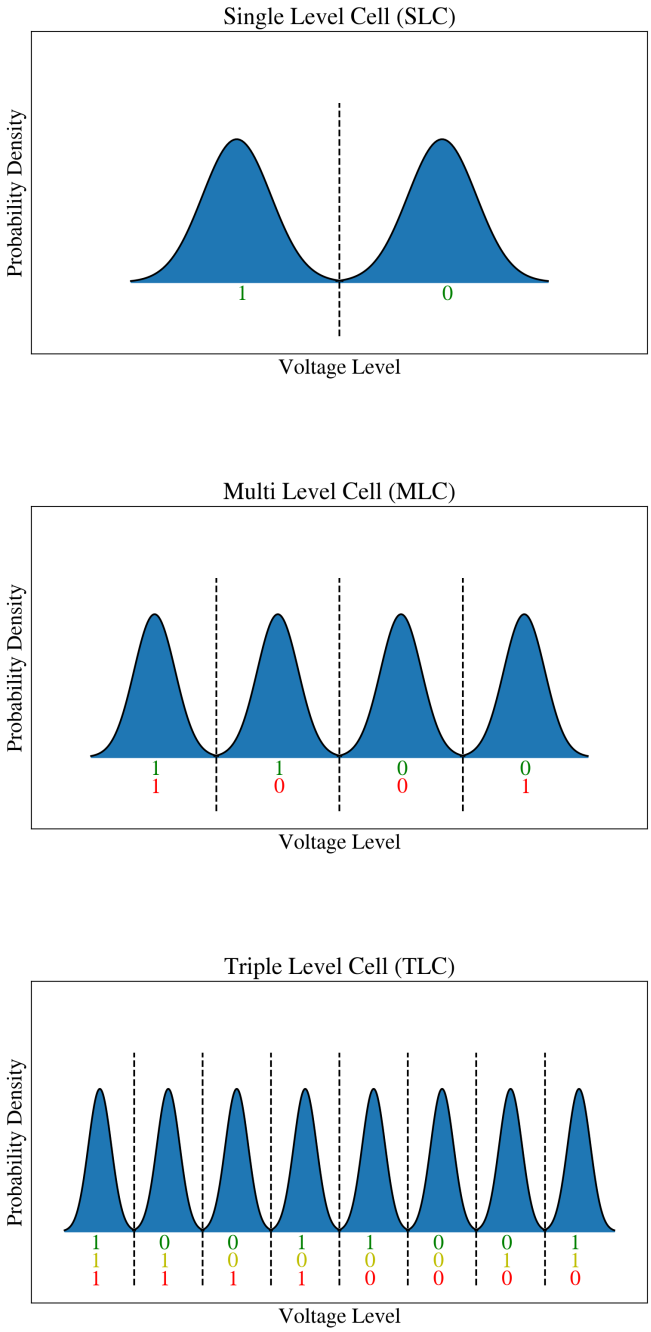


Figure 2.7: Illustration of how up to three data bits are stored as discrete voltage levels inside a memory cell. Each voltage level is assumed to be affected by disturb mechanisms inside the memory and thus, each level is modeled as a Gaussian distribution. Furthermore, the vertical lines represent the ideal read threshold voltages to be applied when reading the cell. Below each discrete voltage level, the corresponding bit values at readout are shown. The green color represents the most significant bit (MSB), the red color represents the least significant bit (LSB) and the yellow color represents the central bit (CB) [13].

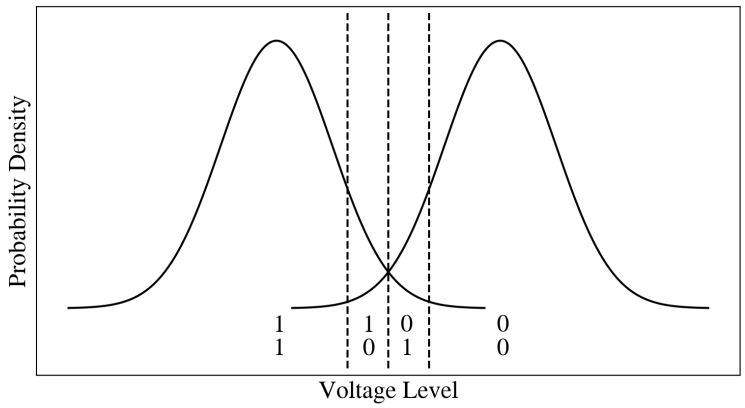


Figure 2.8: Illustration of how multiple thresholds change the readout of the memory in an SLC [9].

in figure 2.7. If the stored voltage level is below the applied V_{read}-level, the cells at the WL will be open, and current will flow through. This is then registered by a sense amplifier at the top of the BL. If the stored voltage level is above the applied V_{read}-level, the cells at the WL will be closed and the sense amplifier will not register a current. This way it is possible to determine whether the read threshold is above or below the discrete voltage level stored inside the cells [15]. Refer to figure 2.9 for an illustration of the read operation.

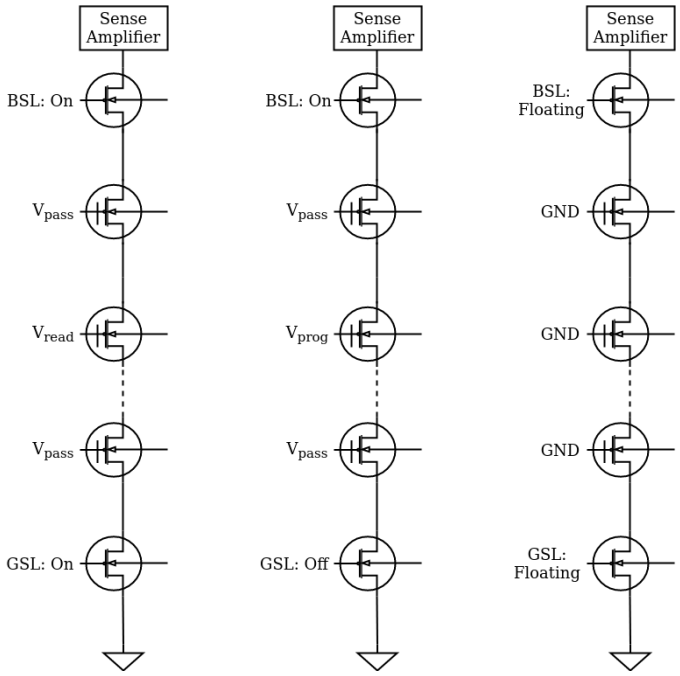
The erase operation is achieved by setting all WLs to ground (GND) as in figure 2.9. This will cause the charge in the floating-gate to be ejected into the substrate of the cells. Since all WLs are set to ground, the erase operation can only be applied to an entire block of cells as previously mentioned [15].

2.3 NAND flash controller

The flash controller is a critical component that makes imperfect NAND flash robust and reliable. The controller and its interaction with the physical memory is depicted in figure 2.10. It is designed to protect and control the physical storage media. The NAND flash controller is responsible for converting bit data into discrete voltages that can be programmed into the NAND flash memory cells. Furthermore, it is responsible for reading the NAND flash memory and correcting any errors that may have occurred [14].

2.4 Noise mechanisms

As the physical sizes of memories shrink and more bits are packed into each unit of storage, NAND flash memories become susceptible to noise mechanisms that could seriously harm the reliability of the memory. This section aims at describing the different noise mechanisms inside NAND flash.



(a) Read **(b)** Program **(c)** Erase

Figure 2.9: Illustration of how basic operations are performed in a NAND flash memory [15].

2.4.1 Raw Bit Error Rate

The basic parameter for measuring reliability in NAND flash is called **raw bit error rate** (RBER), which is a metric defined by the fraction of erroneous bits during the read operation [14]. The RBER of the lowest discrete voltage level is illustrated in figure 2.11. As the noise becomes more severe, consequently, RBER increases.

2.4.2 P/E cycles

The endurance of a NAND flash memory is measured by the number of program/erase (P/E) cycles it can perform. The voltages required to inject (V_{prog}) and eject charge from the floating-gate cause damage to the oxide layers of the cells. Consequently, charge becomes trapped inside the oxide layers and it becomes difficult to retain stored information and erase the cells. A NAND flash memory typically has a lifetime of 10^4-10^5 P/E cycles [15].

2.4.3 Data retention

Leaving memory cells unaltered for a long time will cause stored information to become degraded. This phenomenon is called *data retention* and it is due to charge leaking out at the floating-gate. When charge leaks out of the floating-gate the voltage level of the memory cell is shifted to the left and this could result in a read error of the memory [4]. The effect of data retention becomes more prominent when a large amount of P/E cycles have been performed and it is illustrated in figure 2.13.

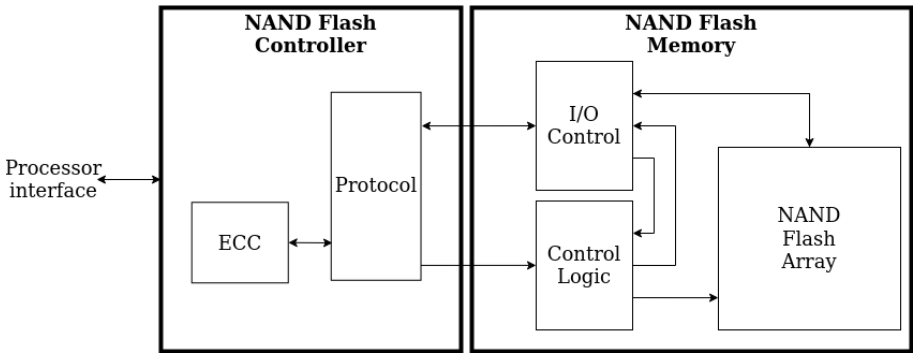


Figure 2.10: Simplified block diagram of how the NAND flash controller communicates with the NAND flash memory. It is the responsibility of the controller to correct any errors that have occurred in the data. This is revealed by the ECC label inside the controller [12]

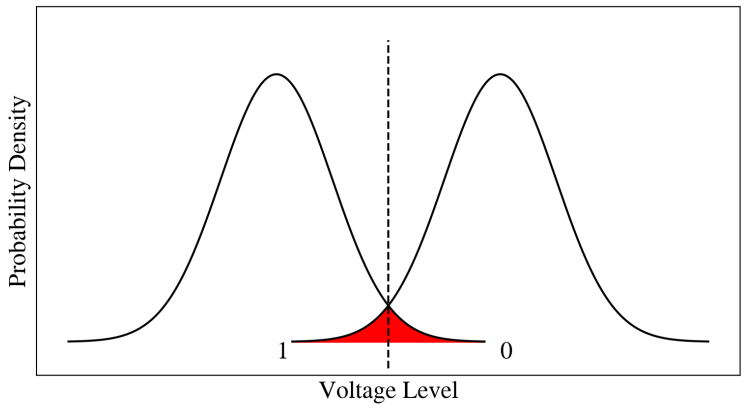


Figure 2.11: Illustration of the RBER of an SLC when the distributions are modeled as symmetrical Gaussian distributions. The region that gives rise to RBER is marked in red color.

2.4.4 Read/program disturb

The read operation is the most common type of operation applied to the memory and consequently, *read disturb* is the most frequent source of noise inside a NAND flash memory. When applying the read operation, the high voltage, V_{pass}, applied at the other WLs of cells that are not read may cause a small amount of charge to be injected into non-read cells. In absence of P/E cycles, the voltage levels of the cells will be shifted to the right with an increasing amount of sequential read operations. Read disturb affects cells with a low programmed voltage level the most. Higher voltage levels are more vulnerable to data retention noise, and therefore, when measuring read disturb, one may find something similar to figure 2.14. Read disturbs will also become more evident at a higher number of P/E cycles since the damage on the oxide layers will make it easier to inject charge into the floating-gate [4].

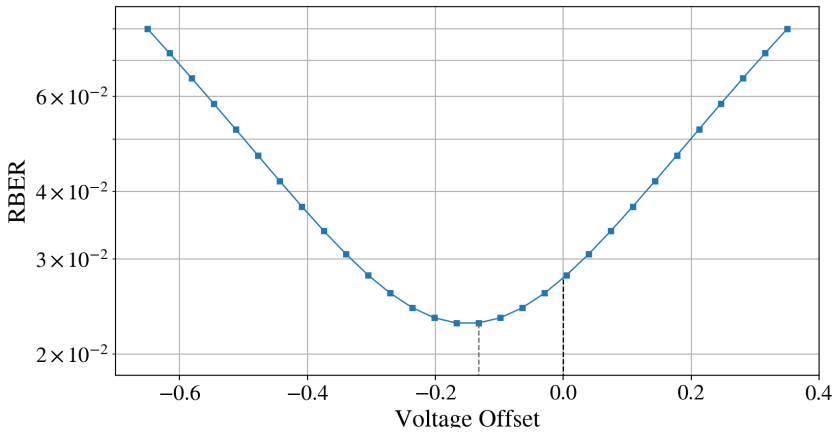


Figure 2.12: Illustration of valley search algorithm where the RBER is a function of the threshold offset. The black dashed line represents the default threshold while the gray dashed line represents the optimal read threshold [15]

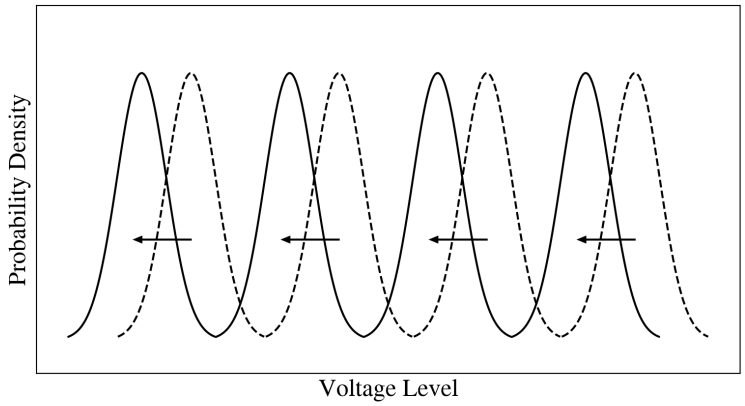


Figure 2.13: Simplified illustration of how the voltage distributions are changed by data retention [4]

Similar to the read operation, the voltage V_{pass} applied at the WLs of cells that are not programmed will cause the voltage level to shift slightly with each iteration of the program operation, causing *program disturb* [4].

2.4.5 Cross temperature

The voltage needed to open a cell with a fixed amount of charge in its floating-gate is dependent on the temperature of its surroundings. A cell programmed at a high temperature, say at 120°C, and then read at a low temperature, say at -160°C, will require a higher threshold voltage level to open the cells relative to the one initially programmed. This type of noise is called *cross temperature* [15]. It may not be very significant in conditions representing the surface of the earth, but in space where fluctuations in temperatures such as the ones mentioned happen as a result of being

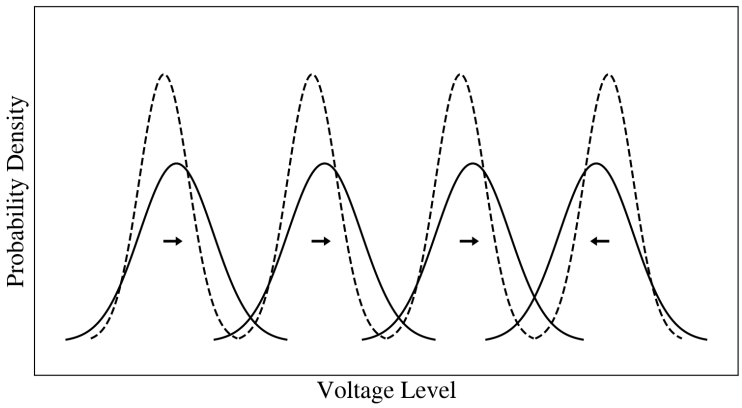


Figure 2.14: Simplified illustration of how the voltage distributions are affected by read disturb [4].

in either the dark or the bright side of the earth, cross temperature noise could be very important [16], [17].

2.4.6 Cell-to-cell interference

Cell-to-cell interference is caused by parasitic capacitance-coupling between adjacent cells. When programming a cell (victim cell), the charge injected into the floating-gate will be coupled with the programmed voltage of neighboring cells. Consequently, when the voltage level of neighboring cells is changed, the voltage level of the victim cell is changed as well. Figure 2.15 depicts the victim cell and the adjacent cells that affect it through cell-to-cell interference [10].

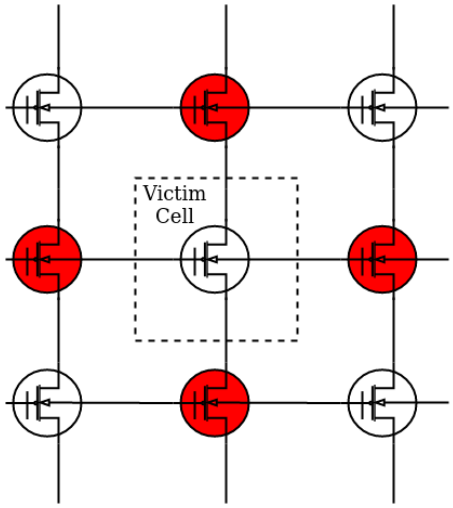


Figure 2.15: The victim cell in cell-to-cell interference and the adjacent cells that cause parasitic capacitance-coupling, marked in red color [15]

2.4.7 Dynamic thresholds

RBER is heavily affected by the read reference voltage levels, V_{read}, used inside the memory since the distribution of the programmed voltage levels are affected by shifts and changes in variances caused by its noise mechanisms. Thus, being able to dynamically adjust the V_{read}-levels could help to reduce the RBER significantly. Algorithms that aim to improve the reliability of NAND flash by dynamically adjusting the read reference levels are generally defined as *Read Retry*. There are many ways one can do this but a common and intuitive way to find the optimal read reference level is to apply an iterative valley search algorithm [18]. Figure 2.12 depicts how such an algorithm tries different read thresholds to find an optimal level. Being able to find the optimal read reference level is an important concept in this work.

3

Error correction with LDPC codes

3.1 Introduction to information theory

Error correction codes is a concept connected to the field of information theory first introduced by Shannon in 1948 [19]. In his groundbreaking paper, Shannon formulates an essential model of communication as the transfer of information from **transmitter** to **receiver** over a **channel**, see figure 3.1, and shows some general principles of this system. Importantly, the channel imposes limitations on the information transfer, and information theory is interested in quantifying these limitations and minimizing their effects. The information to be transmitted is a vector of n bits $\boldsymbol{x} = (x_1, x_2, ..., x_n)^\top \in \mathbb{Z}_2^n$, and to make no assumptions on the content, it is seen as the outcome of a stochastic variable, $\boldsymbol{X}$, from some information generating process in the transmitter. Similarly the received message, $\boldsymbol{y}$, is the outcome of $\boldsymbol{Y}$ from a stochastic process, $\mathcal{C}$, modeling the channel, conditioned on input $\boldsymbol{X}$ and with noise parameters, $\boldsymbol{\mu}$.

$$\boldsymbol{Y} \sim \mathcal{C}^{(\boldsymbol{\mu})}(\boldsymbol{X}) \tag{3.1}$$

Selection of $\mathcal{C}$ is flexible depending on the application, but here all considered channels are defined by one discrete or continuous probability density function (PDF) $P^{(\boldsymbol{\mu})}(Y_i|X_i)$ taking each bit X_i in $\boldsymbol{X}$ as input independently. The independence assumption is referred to as memorylessness of the channel and it follows from the product rule that the probability of receiving exactly output $\boldsymbol{y}$ from input $\boldsymbol{x}$ is

$$P^{(\boldsymbol{\mu})}(\,\boldsymbol{Y}{=}\boldsymbol{y}|\boldsymbol{X}{=}\boldsymbol{x}) = \prod_{i=1}^{n} P^{(\boldsymbol{\mu})}(Y_i = y_i|X_i = x_i). \tag{3.2}$$

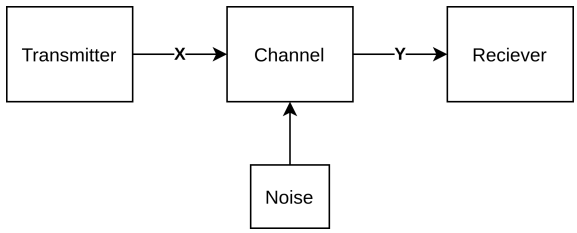


Figure 3.1: A simple diagram of the Shannon-Weaver communication system model.

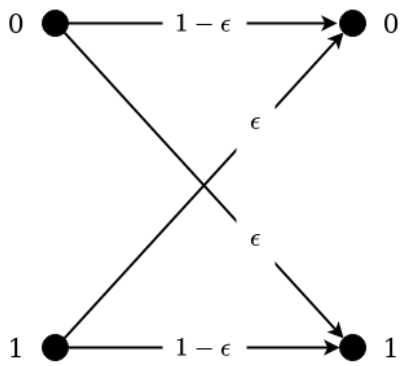


Figure 3.2: Graphical representation of the BSC channel with error rate ϵ.

As an example, consider a bit stream that is guaranteed to receive all transmitted bits, but where each bit has a small probability ϵ of being misread and flipped. This is an interpretation of the **binary symmetric channel** (BSC) with the noise parameter ϵ directly giving the RBER of the system. More generally it is a type of **discrete memoryless channel** (DMC) which is a term referring to all channels which can be described by a transition matrix, T, with the probability densities of going from input i to output j, or the transition probabilities, given in matrix entry T_{ij}. The corresponding transition matrix for the BSC is given below in equation 3.3, with a graphical representation in figure 3.2.

$$T_{BSC}^{(\epsilon)} = \begin{bmatrix} 1-\epsilon & \epsilon \\ \epsilon & 1-\epsilon \end{bmatrix}. \tag{3.3}$$

If rather than bits the signal levels of the stream are read, the output will instead have some continuous deviation from the expected value, leading to a mixed discrete-continuous system. With signal levels, $\pm S$, for zero and one input bits respectively, the **binary input additive white gaussian noise** (BI-AWGN) channel is a common choice for such systems, defined as a mix of two Gaussian distributions

$$P_{BI-AWGN}^{(S,\sigma)}(Y|X=x) = \begin{cases} \varphi(\frac{Y-S}{\sigma}), & \text{if } x = 0 \\ \varphi(\frac{Y+S}{\sigma}), & \text{if } x = 1 \end{cases}, \tag{3.4}$$

with φ representing the standard normal distribution. The noise level for discrete-continuous systems is measured in some variant of a **signal-to-noise ratio** (SNR). Throughout the project and without loss of generality, the SNR has been measured in decibels with S set to be one, leading to the final definition used:

$$SNR = 10 \cdot \log_{10}\Big(\frac{1}{2\sigma^2}\Big). \tag{3.5}$$

By observing that the output data does not completely determine the input data, the fundamental insight of Shannon is reached of a physical limit on possible information transfer. The question to ask becomes *how* determined the input is. This

question is intuitively tied to how ordered the system is, with more order allowing easier transmission of information, which led to the definition of **Shannon entropy**, defined analogously to the corresponding concept in thermodynamics for quantifying the disorder of microstates. For a stochastic variable Y the entropy $H(Y)$ thereby quantifies how surprising the average outcome is, and the similar concept conditioned entropy $H(X|Y)$ how surprising the outcome is if another another variable X is known. Or rephrased, how uncertain it is that Y corresponds with the knowledge of X. Summing over the set of all possible outcomes, $\mathcal{X}$ and $\mathcal{Y}$, of stochastic variables X and Y, the following definitions are used:

$$H(Y) = -\sum_{y\in\mathcal{Y}} P(Y=y)\log_2(P(Y=y)), \tag{3.6}$$

$$H(Y|X=x) = -\sum_{y\in\mathcal{Y}} P(Y=y|X=x)\log_2(P(Y=y|X=x)), \tag{3.7}$$

$$H(Y|X) = \sum_{x\in\mathcal{X}} P(X=x)H(Y|X=x). \tag{3.8}$$

Now to quantify the disorder of Y which the channel is responsible for, the difference between the disorder with and without knowing the input is used, defining the **mutual information**, MI, between X and Y over $\mathcal{C}$.

$$MI(X,Y) = H(Y) - H(Y|X) \tag{3.9}$$

While $H(Y)$ is given by the channel constraints, the distribution of X can be designed by the transmitter to maximize this measure up to a limit. That upper value of the mutual information defines the **capacity**, C, of the channel. C is a value between zero and one which describes how large a fraction of the message can be transmitted without error in theory. Inversely, it may be viewed as a lower bound on how much redundancy in information is needed for all information to remain secure for a certain level of noise. It is also of interest to know the maximum tolerable level of noise for a fixed amount of redundancy, and this quantity is known as the Shannon Limit [9].

As seen, error correction is a core problem in all types of information transfer, and the natural solution to gain trust in the received data is to include some redundancy in the information to at least match the capacity of the channel. Another interpretation is that the redundant information imposes a certain structure on the message, since certain bits have to match, delimiting a set of accepted messages from a larger set of all possible messages. Messages accepted by the code are referred to as **codewords**.This leads to the terminology of encoding-decoding and error correction now becomes a question of keeping to the code: If a message not fitting the code is received, it is immediately possible to recognize that there has been an error. If the likeness of messages is measurable in some way, it is even plausible to assume that the codeword most similar to the received message was the one intended, thereby correcting the error. How to construct efficient and reliable encoding and decoding schemes is the essence of error correction coding theory.

3.2 LDPC Encoding

For low-density parity-check codes (LDPC codes), the defining structure is nothing more than that certain bits of $\boldsymbol{x}$ has to sum to an even number of ones, constraints easily specified through a set of linear XOR-sum equations in $\mathbb{Z}_2$. As the name suggests, each equation is referred to as a parity-check and the only other limiting factor is that the number of bits involved in each equation cannot be too large to qualify the code as low-density. For example, to encode a four-bit vector $\boldsymbol{x}_{info} = (x_1, x_2, x_3, x_4)^\top$ with two parity check-equations, two extra **parity bits** $(x_5, x_6)^\top$ could be calculated as

$$x_1 \oplus x_2 = x_5,$$
$$x_3 \oplus x_4 = x_6,$$

and appended to the message to produce a code word $\boldsymbol{x}_{cw} = (x_1, x_2, x_3, x_4, x_5, x_6)^T$. The following equations are easily checked by the receiver to check the validity of the information:

$$x_1 \oplus x_2 \oplus x_5 = 0,$$
$$x_3 \oplus x_4 \oplus x_6 = 0.$$

A general description of a code is given by a generator matrix, G, and a parity-check matrix, H, to write the previous equations in compact form

$$\boldsymbol{x}_{cw} = \boldsymbol{x}_{info} G, \tag{3.10}$$
$$H\boldsymbol{x}_{cw} = \mathbf{0}. \tag{3.11}$$

Continuing with the above example would result in the following matrices which can be checked by carrying out the matrix multiplications.

$$G = \left[\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right] \tag{3.12}$$

$$H = \left[\begin{array}{cccc|cc} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \tag{3.13}$$

Note that the equation 3.11 results in a vector of zeroes, $\mathbf{0}$, only on the condition that $\boldsymbol{x}_{cw}$ is a codeword. If instead a noised message, the output $\boldsymbol{y}$, is tested, chances are that the equation would instead evaluate to a vector with multiple non zeros in it, referred to as the **syndrome** of the output. From this linear algebraic point of view, the set of codewords is formally defined by the kernel of H, and the decoding corresponds to a projection of $\boldsymbol{y}$ onto this kernel. One measure of how successful this decoding is, is by comparing the decoded message to the original message and counting the fraction of wrong bits to find the decoded **bit error rate** (BER).

This view further generalizes the construction of G and H. So far, $\boldsymbol{x}_{cw}$ is made up of separate information bits and parity bits, and it follows that G and H are split

into one identity matrix and one section defining the equations. Such codes are called **systematic**, but the kernel is of course not dependent on this structure. Any other matrix with the same kernel can be used to define a non-systematic version of the code. Furthermore, any variant of non-systematic H can be transformed into the systematic version and vice versa through row elimination [20]. In this thesis, a non-systematic H is designed and G is computed from H.

3.3 Quasi-Cyclic Codes

This thesis focuses on the special class of LDPC codes which are **quasi-cyclic** (QC-LDPC), for which H has a special block structure

$$H = \begin{bmatrix} H_{11} & H_{12} & \cdots H_{1_n} \\ H_{21} & H_{22} & \cdots H_{1_n} \\ \vdots & \vdots & \\ H_{m1} & H_{m2} & \end{bmatrix} \tag{3.14}$$

having m rows and n columns, and where each block H_{ij} is a square **circulant matrix** of size N

$$H_{ij} = \begin{bmatrix} h_1 & h_2 & h_3 & \cdots & h_N \\ h_N & h_1 & h_2 & \cdots & h_{N-1} \\ \vdots & \vdots & \vdots & & \\ h_2 & h_3 & h_4 & \cdots & h_1 \end{bmatrix} \tag{3.15}$$

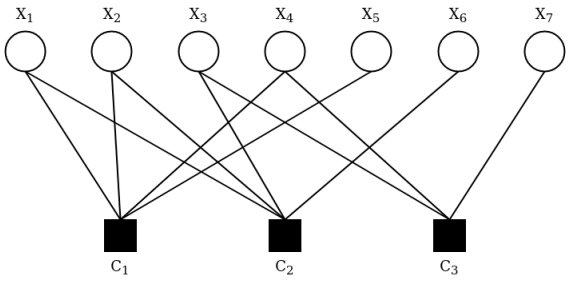
defined by the top row cyclically shifting one position to the right with each row going down. This scales the final code dimensions to $(N \cdot m, N \cdot n)$ and N is therefore referred to as the scaling factor of the QC-LDPC codes defined by (m, n, N). Often, the block structure of H is called the **protograph**, which is **lifted** to the full matrix size when the circulant matrices are expanded. If the circulant matrix additionally contains only one per row, the matrix is **single weighted**, and this is the case for all created matrices in the thesis. This class of codes is important because it has the potential for good performance due to its irregularity, while still having a structure suitable for hardware using shift registers and a rich algebraic structure for mathematical analysis [21].

3.4 LDPC Decoding

From the output $\boldsymbol{y}$ of the channel it is in theory possible to directly compute the estimated codeword, $\hat{\boldsymbol{x}}$, which is most likely to produce the resulting syndrome using Bayes formula

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}} \frac{P(\boldsymbol{X} = \boldsymbol{x} | \boldsymbol{Y} = \boldsymbol{y}) P(\boldsymbol{Y} = \boldsymbol{y})}{P(\boldsymbol{X} = \boldsymbol{x})}. \tag{3.16}$$

This optimal decoding is however far too computationally expensive and this gives rise to the need for iterative algorithms which instead decode parts of the message at a time until the syndrome is zero or the decoding is stuck. Such algorithms do



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

Figure 3.3: Tanner graph with corresponding parity check matrix H. Circular nodes represent variable nodes, black squares represent check nodes, and edges are defined by the ones of the matrix.

not necessarily always converge to the optimal result but perform well enough to be standard practice.

To understand how this type of decoding functions, it is useful to represent the parity check matrix H as an undirected bipartite **Tanner graph**, $\mathcal{G}$. The Tanner graph consists of two sets of nodes taking certain values, **variable nodes** representing the codeword and **check nodes** representing the parity checks. Edges show the connection between the two sets, so an element h_{cv} in H indicates a connection from check node c to variable node v in $\mathcal{G}$. One example of how this is visualized can be seen in figure 3.3. The variable nodes are initialized with the values of $\boldsymbol{y}$, and after that, the decoding works by passing messages back and forth between variable nodes and check nodes in such a way that the variable nodes approach the code word. This general class of algorithms is called message passing algorithms [20].

Messages can be single bits, such as the Gallager algorithms, or information describing how certain nodes are of a certain bit value, as in the **belief propagation** (BP) algorithms [22]. Belief propagation algorithms have been selected for simulations for this project due to their improved performance over single bit algorithms, even though they are more complex. As an example using the BSC, an output $y = 1$ translates to a probability $P(Y = 1|X = 1) = 1 - \epsilon$ and a probability $P(Y = 1|X = 0) = \epsilon$ in the context of BP. Instead of keeping track of this information separately, a nice way of handling these probabilities as one unit is by computing their **logarithmic likelihood ratio** (LLR)

$$LLR(y) = \ln\left(\frac{P(Y = y|X = 0)}{P(Y = y|X = 1)}\right) \tag{3.17}$$

With this definition, negative values indicate a one-bit being the most likely result, and vice versa, positive values indicate the zero-bit being the most likely result. This makes the final codeword easy to read by applying a hard threshold at zero. It should also be pointed out that the definition assumes full precision and knowledge of the

channel probability distribution, while an actual implementation makes an estimate and quantize messages to a maximum of typically 8 bits. As described in the previous chapter, one can add thresholds, as in figure 2.8, to a given read reference level to gain more information regarding the state of the cell. This additional information is called **soft information** and can be utilized by BP to improve the LDPC decoding. By adding more thresholds, it is possible to obtain a higher resolution of LLR messages sent over the LDPC decoding step [9].

It can then be shown that the decoding is an application of the more general Sum-Product algorithm defined by the message update rules summarized below. Here, $m_{c\to v}^{(l)}$ is referring to message passed in iteration l between check nodes c and variable nodes v, and $\mathcal{N}$ is the neighborhood of a node connected via its edges.

$$m_{ch\to v} = LLR(y) \tag{3.18}$$

$$m_{v\to c}^{(l)} = m_{ch\to v} + \sum_{c'\in\mathcal{N}\setminus\{c\}} m_{c'\to v}^{(l)} \tag{3.19}$$

$$m_{c\to v}^{(l)} = \ln\Big(\frac{1+\prod_{v'\in\mathcal{N}\setminus\{v\}}\tanh\frac{m_{v'\to c}^{(l-1)}}{2}}{1-\prod_{v'\in\mathcal{N}\setminus\{v\}}\tanh\frac{m_{v\to c}^{(l-1)}}{2}}\Big). \tag{3.20}$$

This yields optimal decoding as long as there are no cycles in the Tanner graph, with the intuitive reason being that with cycles there is no obvious termination point in the algorithm [20].

Many variants of BP exist modifying details such as the scheduling of the messages. Here, only the case of updating all nodes at once, or **flooding**, has been evaluated. Some simplifications are also suggested to make away with complexity in hardware, most notably the removal of the log-tanh-expression from equation 3.20 resulting in the Min-Sum algorithm [22]. Shortly explained, the shape of the tanh-function makes the expression dominated by the sign product and the smallest message which gives rise to the approximation

$$m_{c\to v}^{(l)} = \prod_{v'\in\mathcal{N}\setminus\{v\}} \operatorname{sign}\Big(m_{v'\to c}^{(l-1)}\Big)\cdot \min_{v'\in\mathcal{N}\setminus\{v\}}\Big|m_{v'\to c}^{(l-1)}\Big|. \tag{3.21}$$

3.5 Categorization of codes using the code ensemble

The capacity discussion from the first section of this chapter ended with an upper bound on the ratio of correctable information to total information. This is for an ECC equivalent to the **code rate**, R, being the ratio between the number of information bits to total bits. Regardless of the size of H of a LDPC code, if it contains the same number d_v of ones in all columns and number d_c of ones in all rows, R is given by

$$R = 1 - \frac{d_v}{d_c}. \tag{3.22}$$

Codes defined by these two numbers (d_v, d_c) are said to belong to the same code **ensemble** and they are called **regular** due to the even distribution of ones. As it turns out, however, codes where the amount of ones varies between columns, **irregular codes**, are often superior in performance. To be able to categorize irregular codes similarly to regular ones, polynomials describing the distribution of connections are instead used, and so **node degree distributions** are given for the variable nodes and check nodes as

$$P(x) = \sum_{i=1}^{k} P_i x^i, \tag{3.23}$$

$$\Lambda(x) = \sum_{i=1}^{k} \Lambda_i x^i, \tag{3.24}$$

$$\tag{3.25}$$

where the i:th coefficient is the fraction of nodes having i connections, and k is the maximum node degree. The calculation of the code rate can be performed using the average number of node connections. If one rather is interested in the number of edges connecting to a node with a specific number of connections the **edge-perspective degree distributions** is equivalently written as

$$\rho(x) = \sum_{i=1}^{k} \rho_i x^{i-1}, \tag{3.26}$$

$$\lambda(x) = \sum_{i=1}^{k} \lambda_i x^{i-1}, \tag{3.27}$$

$$\tag{3.28}$$

and it is possible to calculate from the node degree distribution as

$$\rho_i = \frac{i\rho_i}{\sum_{j=0}^{k} j\rho_j} \tag{3.29}$$

and equivalently for λ_i. It is clear that codes within the same ensemble have the same code rate, but in addition codes similar in this regard also perform roughly equivalent [23]. This case is further elaborated on in section 5.3.

4

Model framework

Previous research shows that picking a suitable channel model in the code design has an impact on the performance of the resulting codes. Besides that, it helps develop more accurate theoretical results such as the capacity or optimal threshold placement [1]. From the theory in chapter 2, two properties stand out that typical models do not capture: 1) In NAND flash memories, the output is a continuous signal discretized by an arbitrary number of voltage thresholds. 2) NAND-flash memories contain non-negligible asymmetric error sources. Because of these issues, a DMC with probabilities based on an underlying BI-AWGN-channel was used, inspired by the work in **threshold**.

4.1 Modeling noise with the skewed n-bit BI-AWGN channel

Definition 1. *The **skewed n-bit BI-AWGN channel**, with skewness parameter $\eta \in [0, 1]$, is a DMC with binary input and 2^n outputs. The transition probabilities are computed by evaluating the cumulative distribution function (CDF) of two Gaussian distributions*

$$T_{ij} = \begin{cases} \Phi^{((\sigma(1-\eta))^2,-1)}(t_i) - \Phi^{((\sigma(1-\eta))^2,-1)}(t_{i-1}), & \text{if } j = 1 \\ \Phi^{((\sigma\eta)^2,1)}(t_i) - \Phi^{((\sigma\eta)^2,1)}(t_{i-1}), & \text{if } j = 2, \end{cases}, \tag{4.1}$$

at thresholds $t_0 = -\infty$, $t_{2^n} = \infty$ and $t1, .., t_{2^n-1}$ selected to maximize the mutual information of the resulting channel.

The skewed n-bit BI-AWGN channel aims to mimic what is happening inside the hardware in a generalized manner. Similar to what is happening on the hardware there is an underlying noise distribution that is discretized, using the concept of mutual information to do this unambiguously and optimally. Furthermore, the noise distribution may be skewed between cells using the skewness parameter $\eta \in [0, 1]$ rather than the typical symmetric distribution[1]. The channel can also be seen as a generalization of other channels to simplify comparison. For example, with a

[1]The definition is indifferent to skewness parameterization, as a suggestion $\sigma_1^2 = \sigma^2(1 + \eta)$, $\sigma_2^2 = \sigma^2(1 - \eta)$ for $\eta \in [-0.5, 0.5]$ could be a more intuitive extension of the standard case.

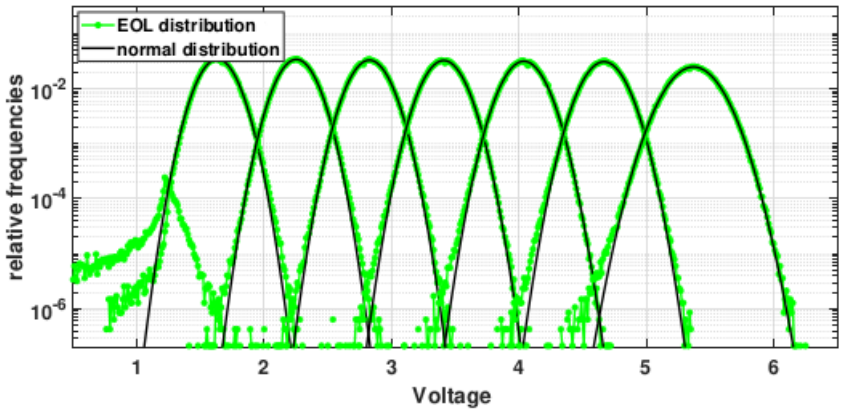


Figure 4.1: The distribution of voltage levels in a TLC at end of lifetime (EOL) (green). A normal distribution (black) has been fitted to the data [15].

skewness $\eta = 0.5$ and $n = 1$, it is easy to see that the resulting channel is equivalent to a BSC. Letting $n \to \infty$ gives back the BI-AWGN channel.

The simplification of considering only the case of two adjacent distributions, even for MLCs and TLCs, is argued for by the symmetry of the voltage levels. The read operation is designed such that in the read of a certain page the thresholds will always be placed between two voltage levels encoding opposite bit-values. On the other hand, the gray encoding ensures that at least two voltage levels are encoding equal bit-values between each threshold. In this way, errors are only likely to happen in one direction for each voltage distribution, over the threshold. In reality, the voltage distributions are not exactly equal and which bit-value is assigned to the left and the right would differ, but these effects are not deemed important at this stage of analysis. Another approximation is when reading voltage levels that are not next to a threshold, which is common for example in a read of the MSB. If those levels are considered error safe, because they are far from the threshold, it is easy to see that the computation of the actual RBER is only a matter of multiplication with the probability of selecting a voltage level next to a threshold.

One could attempt to model each noise mechanism mentioned in chapter 2 independently as functions of voltage levels and time, and many works have indeed attempted to do this [10]. However, the temperature-, data- and time-dependency of the noise make this a daunting task [24]. Thus, it is of interest to model the accumulated noise inside the memory when it performs the worst and logically, this occurs towards the end of lifetime (EOL) of the memory. Its worst performance sets bounds on the RBER that the code must be able to correct. Figure 4.1 depicts how a Gaussian distribution has been fitted on real EOL data according to

$$V_i(x) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{\frac{-(x-\mu_i)^2}{2\sigma_i^2}}, \tag{4.2}$$

where sub-index i refers to the discrete voltage level. This model is common in the literature [24] and some algorithms working on the memory such as Read Retry

algorithms [18] assume this model, motivating the use of the Gaussian distribution as the underlying noise distribution. If one has access to real EOL data of a specific memory used in its real environment, one could easily extend the model using a more accurate CDF. As can be seen in figure 4.1, the fitted curve explains the EOL data rather well, except at the leftmost tail of each distribution.

Using this model, it also becomes simple to draw a parallel between the RBER and SNR measures of noise. For a noise configuration (σ, η) of the underlying skewed BI-AWGN channel, the RBER, ϵ, is the average overlap over the threshold of the 1-bit discretized version:

$$\epsilon = \frac{1}{2}\Big[(1 - \Phi^{((\sigma(1-\eta))^2,-1)}(t_1)) + \Phi^{((\sigma\eta)^2,1)}(t_1))\Big]. \tag{4.3}$$

Inversely, the corresponding σ given (ϵ, η) can be found as a root to the equation using Newton's method. The big importance of this connection is to find equivalent noise levels for different choices of η. Contrary to the real world, where the thresholds are set imperfectly, this theoretic noise level will most likely be higher to result in the same RBER as measured on hardware. On the real device part of the error is rather a result of using sub-optimal thresholds.

4.1.1 The skewed 2-bit BI-AWGN channel

The skewed 2-bit BI-AWGN channel is important and studied in this thesis. The channel is a result of performing reads as in figure 2.8, which in turn, generates the channel depicted in figure 4.2. As previously mentioned, the thresholds are set to maximize the mutual information of the channel. Figure 4.3 shows how mutual information changes as a function of the threshold offset for the 2-bit model with $\eta = 0.5$ [9].

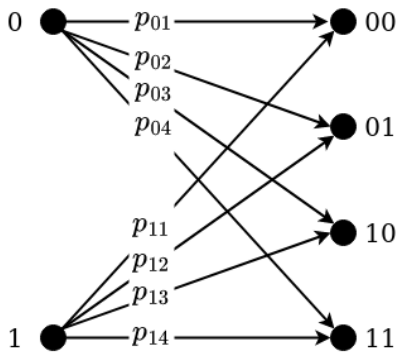


Figure 4.2: Channel model that is generated when using three thresholds. The channel model is referred to as the 2-bit channel model throughout this thesis.

It is of course possible to add more than two thresholds to gain more soft information. Each additional bit of soft information will likely improve the performance of the decoder. However, one has to bear in mind the loss in speed that this would cause.

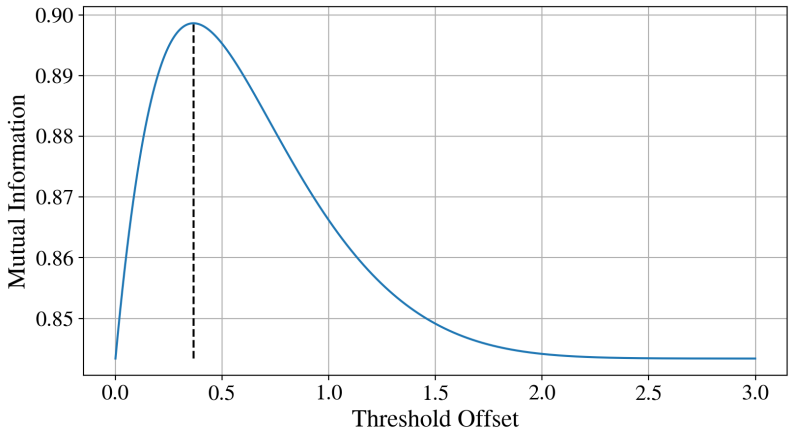


Figure 4.3: Mutual information as a function of the threshold offset when the skewed 2-bit BI-AWGN channel is analyzed with skewness parameter $\eta = 0.5$. In the symmetrical case, the thresholds are set symmetrically and can be represented by a single graph.

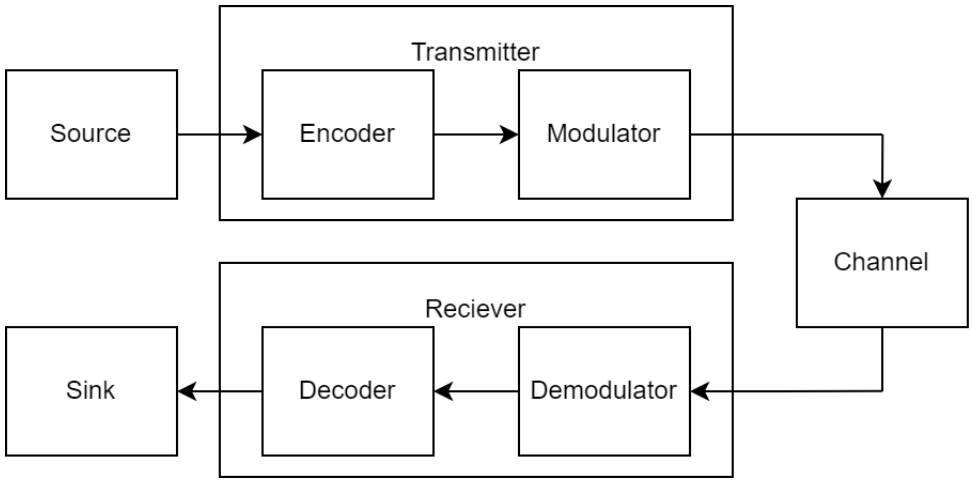


Figure 4.4: Extended communication system model.

Moreover, adding more thresholds means that more read operations would need to be performed. As described in chapter 2, there are noise mechanisms related to the read operation and thus, there is likely a reliability trade-off of adding additional thresholds. The skewed 2-bit AWGN channel is therefore a reasonable choice of channel to study.

4.2 Full simulation model

For the final simulation, a further detailed model was implemented to allow for a larger degree of freedom in testing. As visualized in figure 4.4, the transmitter of the standard communication model in this model receives information from a **source**, encodes it, and then **modulates** it. Modulation refers to the transformation of

information into a physical signal, suitable for the channel, before message transmission. The receiver performs the inverse operations, demodulating and decoding before the information finally ends up in a **sink**[22]. While the terminology of modulation comes from wave-based communication and the modulation of a carrier signal to store message information, the concept translates well to NAND flash memories. Here, the modulation is of digital information into physical voltage thresholds, and demodulating back to digital form is done by applying voltage thresholds.

The benefit of the new components is the separation between the modem and channel to make it easier to differentiate between changes in how information is stored in cells and the modeling of the noise in the memory cells. This flexibility allows for easier implementation of different ways of memory cell programming, more true-to-hardware implementations of thresholding and noise estimators, and new noise models in the future.

5

LDPC Code design

5.1 Design strategy

As indicated several times in the previous chapters, the design of an error correcting scheme from encoding to decoding is more often than not about good approximations rather than theoretical perfect solutions and a wide range of aspects has to be taken into account for in, many times in interacting and conflicting ways. Three main design considerations are summarized below:

1. **Error correcting performance:** Amount of errors before and after error-correcting.

2. **Hardware footprint:** Hardware resources needed to implement the algorithm.

3. **Data capacity:** Latency and throughput of corrected data.

Here (1) is the main measurable in the initial testing and design, while (2) and (3) are a function of the algorithm complexity which is more difficult to analyze directly without doing the full implementation. Rather by comparing the simulated performance of schemes of different complexity it is possible to find a good trade-off. Complexity increases for example with higher floating point precision, more exact decoding, and less structured codes. Notice also that already the choice of using QC-LDPC codes is motivated by targeting these aspects as detailed in chapter 3.

Figure 5.1 demonstrates the typical graph on the error-correcting performance of an LDPC code, with decoded BER plotted against RBER. For discrete-continuous channels, the graph is similar but the SNR is instead plotted on the x-axis. Two regions of interest are indicated in the figure: The **waterfall region** which is characterized by a steep decrease in performance below a certain noise level, and the **error floor** before the waterfall where correction improvement slows down dramatically. This shows a fundamental flaw for LDPC codes in how they may fail the decoding even for high SNRs, which is further discussed in section 5.4 [1]. Aside from being two important performance indicators, the waterfall region and error floor depend on different properties of the code. This leads to a design pattern of first optimizing the waterfall and then the error floor within the constraints of the first design pass.

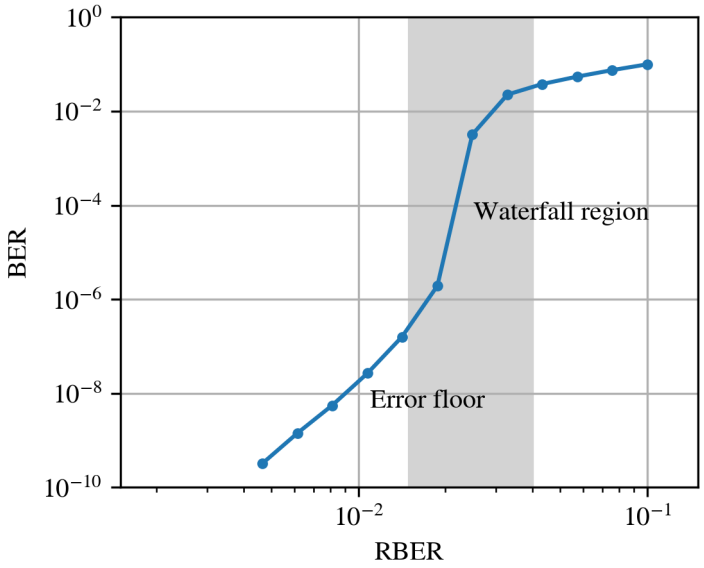


Figure 5.1: Example of typical performance of noise before and after decoding. Two regions of interest are marked, the waterfall region and the error floor

A couple of requirements are specific to the NAND flash memory for which the controller is designed. Most specific for this project is a specification stating that the error correcting code must be able to correct for RBERs up to 1%, giving a hard limit on performance. Also, the code rate is fixed by the dedicated data and parity bits of the page. Because of the error floor *being able to correct for* 1% *RBER* must be interpreted as having a waterfall threshold before this limit and not perfect decoding. Other strategies for removing the remaining errors exist, for example by applying a simpler outer code after the LDPC decoding [25]. It is also important to note that in opposite to use cases where information can be resent on decoding failure, in NAND flash devices information lost is lost forever. This strengthens the need for stronger codes than perhaps otherwise needed.

5.2 Code dimensions

The matrix dimensions of H and the corresponding code rate is as mentioned not very flexible since each page is read separately with a dedicated number of data bits and parity bits. On top of that, the usage of QC codes further limits the dimension choice because a large common denominator between the number of check nodes and variable nodes is needed as the scaling factor of the protograph. A (9, 73, 256) QC-LDPC code was determined to be good final dimensions to keep the protograph small, decreasing complexity, while still having a large enough amount of check equations.

To reach the exact dimensions of the device, several techniques have been developed to make rate-compatible codes while not degrading the decoding performance.

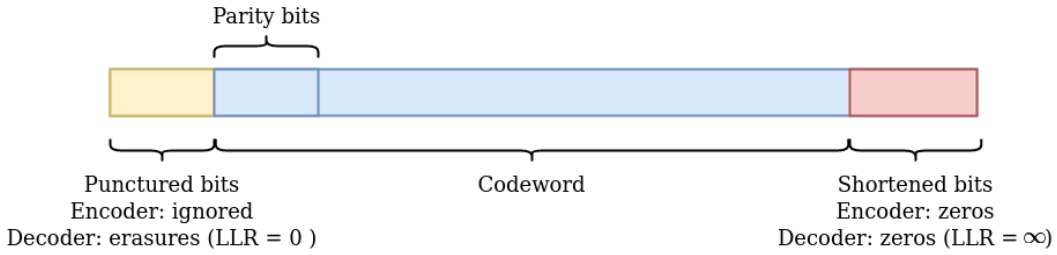


Figure 5.2: Illustration of how punctured and shortened bits are appended to a given codeword. Punctured bits are appended at the beginning of a given codeword and these bits are ignored at the encoder and treated as erasures at the decoder. Shortened bits, on the other hand, are appended at the end of a given codeword. These bits are set as zeros at the encoder and treated as reliable zeros at the decoder. Note that this means that the transmitter and the receiver know which bits are punctured and shortened [26].

The most common ones are shortening, puncturing, extending, and combining [27]. Puncturing and shortening are illustrated in figure 5.2.

The different techniques can be naturally divided into two groups, one group that cause the code rate to increase and another that cause the code rate to decrease. In the first group, puncturing and combining are included. Puncturing is a popular technique that essentially omits some of the parity bits to achieve higher code rates. Puncturing typically degrades the performance of the decoder, although it has been shown that, theoretically, puncturing can achieve the same gap to capacity as the base graph code [28]. The degradation in performance is typically due to to the LLR messages of punctured bits being set to a low value, i.e. they are treated as erasures at the decoder. Instead of modifying the given codeword to be compatible with the given code, one can adjust the code to be compatible with the given codeword. This can be achieved with the method of combining. By combining rows of a given parity check matrix, H, the code rate can be reduced by replacing a group of parity check nodes with a single parity-check node [29].

The other group that causes the code rate to decrease includes shortening and extending. Shortening is a method where some data bits are set to 0 with high reliability. This typically improves the performance of the decoder since the LLR messages of shortened bits are assigned a high LLR value, i.e. they are certain at the decoder. With the same analogy as the method of combining, one can also adjust the parity check matrix, H, instead of the codeword to achieve lower code rates. This is called extending and it ensures that the information block size remains the same. The originally high-rate code is then simply nested inside to produce a lower rate code [28].

5.3 Waterfall optimization trough density evolution

An optimal LDPC code is typically defined by a waterfall region close to the Shannon limit of the channel, i.e. the more the waterfall region of a specific code is shifted to the right in figure 5.1, the better it will be at handling large rates of errors. In principle, when finding an optimal code that performs well in the waterfall region, it would be possible to simulate a large number of codes and pick the code that performs the best. However, due to the large search space and the time it takes to simulate, this would take an immensely long time. Therefore, it is necessary to find a method in which the decoding performance at the waterfall region can be characterized through probabilistic measures of groups of codes. This is where the code ensembles mentioned in 3 come in handy. By looking at a group of codes, the search space becomes smaller and simulation times become much faster, which makes the optimization computationally feasible. To further motivate the use of code ensembles, it can be shown that decoding failures that characterize the type of failures that occur at the waterfall region are sharply concentrated around the ensemble average. This makes code ensembles representative of the decoding performance at the waterfall region of individual codes within [23].

As previously mentioned in chapter 3, the code ensemble can be defined in terms of the codeword length, n, and the variable node and check node edge-perspective degree distributions, λ and ρ. The task at hand is then to find how the waterfall region changes with iterations of LDPC-decoding in terms of n, ρ and λ. Unfortunately, it is difficult to find analytical expressions in terms of n; however, there is a key observation of how the characteristic shape of the waterfall region changes with n, namely, as $n \to \infty$, the waterfall region approaches a vertical line. Interestingly, in this asymptotic boundary, the cycles that cause the error floor disappear since the probability of connecting to a node generating a cycle is infinitesimal. Consequently, by letting $n \to \infty$ the problem at hand is simplified to finding whether a code ensemble LDPC(λ, ρ) will converge to zero BER as the number of iterations, l, approaches infinity. This is called **density evolution**, and as the name implies, it is a method that tracks how the average density of LLR messages sent between variable nodes and check nodes and vice-versa changes with iterations of BP for a code ensemble.

There are some essential preliminaries to be mentioned before density evolution is presented in more detail. Firstly, this section aims to go give an understanding of how density evolution works. If one wants complete mathematical proofs and more detail on density evolution, refer to the sources mentioned in this section. Secondly, the symmetry of the BP algorithm causes the BER to be independent of the transmitted codeword if the channel is symmetrical [30]. In this study, an asymmetrical channel is considered; however, in terms of density evolution, it is possible to symmetrize the channel. Thus, density evolution will firstly be presented for a symmetrical channel. Then, it will be shown how an asymmetrical channel can be symmetrized.

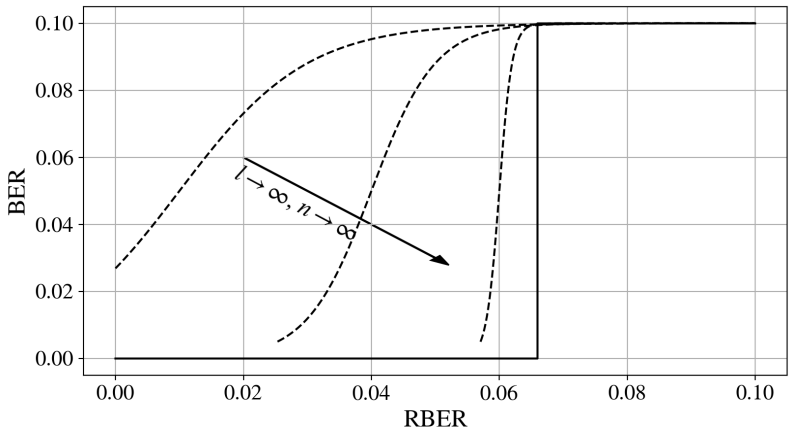


Figure 5.3: Illustration of how the characteristics of the waterfall region change with iterations, l, and codeword length, n. As both l and n go to infinity, the waterfall region approaches a vertical line and the error floor disappears.

Consider the messages sent over one iteration of belief propagation at the LDPC decoding step mentioned in chapter 3. For simplicity of later computations, for input variable x, map $0 \mapsto 1$ and $1 \mapsto -1$. Furthermore, define

$$\gamma(m) := (\gamma_1(m), \gamma_2(m)) := \Big(\operatorname{sgn}(m), -\ln\tanh\left|\frac{m}{2}\right|\Big), \tag{5.1}$$

where m is the LLR message. The message passing equations in 3.18-3.20 could then be rewritten as

$$m_{ch\to v} = \ln\Big(\frac{P(Y|X=1)}{P(Y|X=-1)}\Big), \tag{5.2}$$

$$m_{v\to c}^{(l)} = m_{ch\to v} + \sum_{c'\in\mathcal{N}\setminus\{c\}} m_{c'\to v}^{(l)}, \tag{5.3}$$

$$m_{c\to v}^{(l)} = \gamma^{-1}\Big(\sum_{v'\in\mathcal{N}\setminus\{v\}} \gamma\Big(m_{v'\to c)}^{(l-1)}\Big)\Big). \tag{5.4}$$

The above equations may seem a little bit confusing at first glance, especially equation 5.4. However, it will become evident later that when mapping how probabilities change over iterations of BP, being able to express messages as sums of messages from neighboring nodes will help computations significantly.

Let $P^{(l)}$ and $Q^{(l)}$ denote the CDFs of the random variables $m_{v\to c}^{(l)}$ and $m_{c\to v}^{(l)}$ respectively given that $X = 1$ is transmitted over a channel. Furthermore, let $p^{(l)}$ and $q^{(l)}$ denote the corresponding PDFs. For a symmetrical channel, the all-one codeword can be assumed [23]. Given a channel, the CDF of $P^{(0)}$ is a known function. For

example, for a BSC with crossover probability ϵ

$$P^{(0)}(m) = \int_{-\infty}^{m} \epsilon\delta(m' + \ln\frac{1-\epsilon}{\epsilon}) + (1-\epsilon)\delta(m' - \ln\frac{1-\epsilon}{\epsilon})dm' \tag{5.5}$$

where δ is the Dirac delta function.

Given that $P^{(0)}$ is known, the task is to compute the subsequent $Q^{(1)}$ and $P^{(1)}$. To compute $Q^{(1)}$ from the mapping in equation 5.4, two different traits of transformations of probabilities need to be stated.

The first one is a linearity trait which states the following. Assume a random variable A with CDF, P_A. If $g : A \mapsto g(A)$, then $B = g(A)$ is a random variable with CDF $P_B = T(P_A) = P_A \circ g^{-1}$. This transformation is linear, meaning if $P_D = T(P_C)$, then $T(\alpha P_A + (1-\alpha)P_C) = \alpha P_B + (1-\alpha)P_D$ for $\alpha \in$[0,1] [31].

The next trait states that the density when summing independent variables is represented by a convolution. Assume random independent variables A_i with CDF P_{A_i}, where $i = 1, 2, 3, ..., n$. Then, $B = \sum_{i=1}^{n} A_i$ is a random variable with CDF $P_B = \bigotimes_{i=1}^{n} P_{A_i}$ [23]. Here, $\bigotimes$ represents the convolution over all CDFs. For P_{A_1} and P_{A_2} this convolution is computed as

$$(P_{A_1} \otimes P_{A_2})(x) := \int P_{A_1}(x-y)p_{A_2}(y)dy. \tag{5.6}$$

Notice that there is one CDF and one PDF in the integral. This evaluation can be simplified by Fourier transforming the CDF and the PDF. This would turn the above expression into a multiplication [23].

It is now possible to derive an expression for $Q^{(1)}$. Firstly, define Γ as the function that maps CDFs under γ. Given a specific code ensemble, LDPC(λ, ρ), $Q^{(1)}$ can be computed as

$$Q^{(1)} = \Gamma^{-1}(\rho(\Gamma(P^{(0)}))) := \Gamma^{-1}\Big(\sum_{i\geq 2} \rho_i \overset{(i-1)}{\bigotimes} (\Gamma(P^{(0)}))\Big). \tag{5.7}$$

The subsequent distribution for $P^{(1)}$ is easier to compute since it only contains a sum. With reference to equation 5.3,

$$P^{(1)} = P^{(0)} \otimes \sum_{i\geq 2} \lambda_i \overset{(i-1)}{\bigotimes} Q^{(1)}. \tag{5.8}$$

Equations 5.7 and 5.8 can be directly generalized to iteration number l. The following evolution of densities is derived.

$$Q^{(l)} = \Gamma^{-1}\Big(\sum_{i\geq 2}\rho_i \bigotimes^{(i-1)} (\Gamma(P^{(l-1)}))\Big) \tag{5.9}$$

$$P^{(l)} = P^{(0)} \otimes \sum_{i\geq 2}\lambda_i \bigotimes^{(i-1)} Q^{(l)}. \tag{5.10}$$

The derived expression is true for symmetric binary memoryless channels of LDPC decoding. If one has an asymmetric channel, it is possible to symmetrize the channel [32]. Given an asymmetric channel, denote $P^{(0)}(x)$ the density given that variable $x \in \{-1, 1\}$ is transmitted. The channel can then be symmetrized by the following equation:

$$P^{(l)} = \frac{P^{(l)}(1) + P^{(l)}(-1) \circ I^{-1}}{2}. \tag{5.11}$$

Here, I^{-1} is simply a function that flips another function around the origin [31].

As previously mentioned, the idea of density evolution is to determine the boundary of RBER that a code ensemble can handle so that BER goes to zero as the number of iterations goes to infinity. For this reason, it is of interest to define the BER at iteration, l, of decoding. This variable is denoted as $BER^{(l)}$ and is computed as

$$BER^{(l)} = \int_{-\infty}^{0} P^{(l)} dm \tag{5.12}$$

Since it is not computationally possible to let $l \to \infty$, a stability condition is needed. Firstly, it has been shown that the BP algorithm generates a non-increasing function of BER in terms of iterations. Secondly, it is possible to use the Chernoff bound to define a condition for when the $BER^{(l)} \to 0$ as $l \to \infty$. The Chernoff bound is defined as

$$CBP^{(l)} := \int_{-\infty}^{\infty} e^{\frac{-m}{2}} P^{(l)} dm. \tag{5.13}$$

There are some interesting properties of this expression. Firstly, it can be shown that the Chernoff bound is related to $BER^{(l)}$ by

$$2BER^{(l)} \leq CBP^{(l)} \leq 2\sqrt{BER^{(l)}(1 - BER^{(l)})}. \tag{5.14}$$

Secondly, let $r := CBP^{(0)}$. If $\lambda_2\rho'(1)r < 1$ and ϵ^* is the smallest possible root for

$$\lambda(\rho'(1)\epsilon)r = \epsilon. \tag{5.15}$$

Then, if for some l^*, $CBP^{(l^*)} < \epsilon^*$, then $BER^{(l)} \to 0$ as $l \to \infty$. Furthermore, if $\lambda_2\rho'(1)r > 1$, then $BER^{(l)} \to 0$ as $l \to \infty$. These properties serve as a criterion for when density evolution should be stopped [31].

To summarize, density evolution is a tool for determining whether an infinite codeword-length code ensemble (ρ, λ) manages to decode a noised message given a specific noise parameter. Using this algorithm the noise parameter for which the ensemble starts to converge may be found, for example using a bisection search. This value is used as an objective function in an optimization algorithm to find the best codes. In the developed algorithm a genetic algorithm is applied which is a standard choice because of the combination of a large search space and complex objective function. Without going too much into detail, the genomes of the algorithm are made up of H from which (ρ, λ) may be easily calculated. This ensures that the optimal ensemble is compatible with the given matrix dimensions, and allows for a straightforward implementation of standard genetic operators [33].

5.4 Error floor optimization through progressive edge growth

5.4.1 Algorithm choice motivation

As stated in 3.1 it is the introduction of cycles in the graph which causes suboptimal decoding and raises the error floor. Not all cycles necessarily lead to worse performance, and so the terminology is made precise with the definition of **trapping sets**, which are exactly the structures of nodes that fail the decoding if a certain number of errors enter. Finding and removing trapping sets in a graph is not easy, being an NP-hard problem that differs depending on the channel model, and removing all sets is not even needed for good performance [34]. This strategy is also not flexible when it comes to keeping to the code ensemble (λ, ρ) since new trapping sets may be created as old ones are removed by swapping some edges. Different strategies utilizing proxy measurements for trapping sets have therefore been developed [21].

Broadly speaking, these strategies may be categorized as random-like computer search or structured algorithmic constructions, and both have their pros and cons. Computer search can bypass theoretical approximations and optimize by testing directly over the channel, or potentially even simulating on hardware level through high-level synthesis, but they are computationally slow and the loss of structure increases complexity as well as decreases reuse-ability. Algorithmic strategies on the other hand do rely heavily on approximations and are not as sure to always create a good code in practice [35], [36], [37]. Since the dimensions of the code in this project are large the approximations should work well, while the computational cost

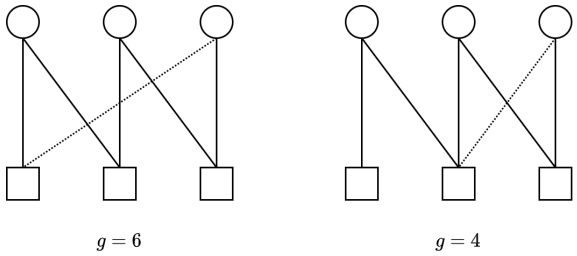


Figure 5.4: Demonstration of progressive edge growth. By trying both possible edge placements, the algorithm figures that the first alternative is the better choice since it yields a higher local girth.

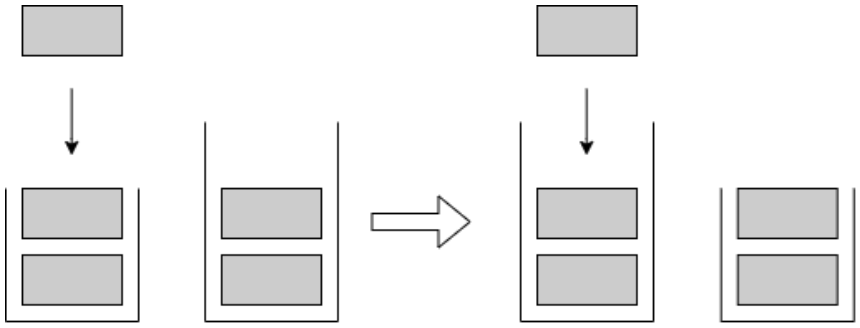


Figure 5.5: Demonstration of a check node distribution swap. The containers represent the desired number of connections assigned to two check nodes according to the distribution, while gray blocks represent the current number of connections. In the case where the selected check node is full, it swaps the container to fulfill the check node distribution constraint and the check node choice.

of random search would be very large. A strong case is finally made for utilizing an algorithmic approach because the complexity increases with less structure.

Algorithmic strategies come in many flavors, but focusing on flexibility concerning code dimensions and node distributions, and low computational cost, a modified version of the **multi-edge metric-constrained quasi-cyclic progressive edge-growth algorithm** (MM-QC-PEGA) proposed by He, Zhou, and Du was chosen for this project [21]. The authors also give a very generalized description of the class of progressive edge growth of algorithms which allows for easily incorporating other properties found to increase the performance of the code further on. A short informal description of the algorithm is given below, referring to [38] and [21] for a more in-depth explanation.

5.4.2 Description of MM-QC-PEGA

Progressive edge-growth algorithms construct the Tanner graph by, as the name suggests, progressively growing the graph by adding one edge at a time. Simplifying the explanation at first by ignoring the quasi-cyclic part, the algorithm starts with a Tanner graph $\mathcal{G}$ containing only nodes and no edges, and desired variable and check node distributions P, Λ, derived from the density evolution. The distributions

determine how many edges should be connected to each node, and the idea is now to draw these edges in such a way as to maximize some metric on the graph. In the version of the algorithm used, this metric was chosen to be the length of the minimal cycle that passes through the added edge, the **local girth**, g. In cases where no cycle exists, g is defined to be infinity.

One step of the algorithm in a simple case is shown in figure 5.4. Each variable node is here to be assigned two edges, and all but the last edge is already assigned. By trying both possible check nodes, it is easy to see that the first one is the best choice as it results in the largest local girth $g = 6$. If multiple check nodes were to result in the same local girth, one is chosen at random.

In more complex situations, it is not obvious that the best choice in this iteration necessarily maximizes the local girth further down the line. The multi-edge algorithm, therefore, extends the algorithm concept by not only adding one but up to r edges to find the edge maximizing the future potential metric with search depth r. Adding the quasi-cyclic structure is as simple as in each iteration adding all edges of one circulant matrix rather than single edges.

This process takes care of ensuring the variable node distribution P, but to enable the algorithm to fulfill also Λ without unnecessarily compromising with the local girth, inspiration was taken from Chen and Cao [38]. Check nodes now has a maximum amount of connections that may never be trespassed just as the variable nodes, but which node takes which amount may change. The picture is clear with figure 5.5 where the desired check node distribution for a node is illustrated as a container for edge connections. If the node selected by the algorithm is full in this sense, it may just swap the container with another node that has the same number of connections but a larger container. In the cases where this is not possible, the second best node has to be selected which compromises the result somewhat, but still less than without the swapping.

The final step after placing all edges is to find the columns of H which can be assigned to parity bits, remembering from section 3.2 that this set of columns must be possible to convert to an identity matrix by row elimination. This is equivalent to finding $N \cdot m$ linearly independent columns spanning the maximum rank in $\mathbb{Z}_2^n$ of the matrix and so for this purpose, a simple algorithm was constructed. Starting with the matrix H of full rank, N rows are removed at a time and the rank of the resulting matrix is calculated. If the rank decreases the columns are added back, otherwise the process continues, and this repeats until only the parity check columns remain. Note that nothing in the original algorithm guarantees a matrix of full rank, but from the experience of the experiments not having a full rank matrix is unlikely.

6

Simulations

In the simulations the developed design chain is applied to optimize channel specific codes. This entails first calculating variances, thresholds, and LLR:s for the n-bit discretized BI-AWGN channels for different RBER:s, optimizing the code ensemble over the calculated channel using density evolution, and in the end applying MM-QC-PEGA to get the code defined by H. Performance simulations are run in the `C++` ECC library Aff3ct [22], modified to use the NAND flash communication chain established in 4. Here, results from the design algorithms are presented, followed by the performance experiments.

6.1 Density evolution

The evolution of densities for the 2-bit BI-AWGN-optimized code is shown in figure 6.1 and 6.2. These graphs give insight into how iterations of density evolution affects an initial distribution. The former figure shows densities at the 0^{th} iteration, while the latter shows the evolved densities at the 50^{th} iteration. Furthermore, figure 6.3 shows how the Chernoff bound and the stopping criterion variable, ϵ, change with RBER. The figure also shows how $CBP^{(l)}$ changes with iterations of density evolution for the 2-bit BI-AWGN optimized code when RBER$= 0.02$.

The resulting polynomials for the 1-bit BI-AWGN-optimized code are

$$\begin{aligned}\rho(x) &= x0.0962963 + x^2 0.85925926 + x^3 0.04444444\\ \lambda(x) &= x^{20}0.19095477 + x^{21}0.10050251 + x^{23}0.22110553\\ &\quad + x^{25}0.36180905 + x^{26}0.12562814\end{aligned}$$

and for the 2-bit BI-AWGN-optimized code

$$\begin{aligned}\rho(x) &= x0.10606061 + x^2 0.89393939\\ \lambda(x) &= x^{18}0.09183673 + x^{19}0.09693878 + x^{20}0.30612245 + x^{23}0.11734694\\ &\quad + x^{25}0.12244898 + x^{27}0.26530612\end{aligned}$$

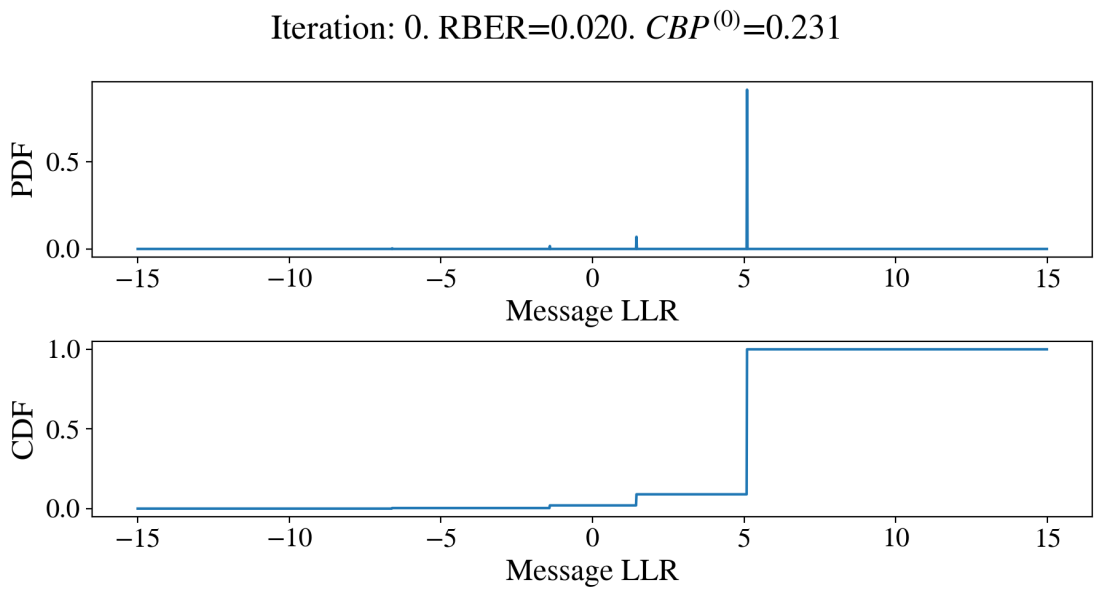


Figure 6.1: The PDF and CDF of message LLRs from variable node to check node at the 0^{th} iteration when RBER=0.02. At the top, the corresponding value of $CBP^{(0)}$ is given as well.

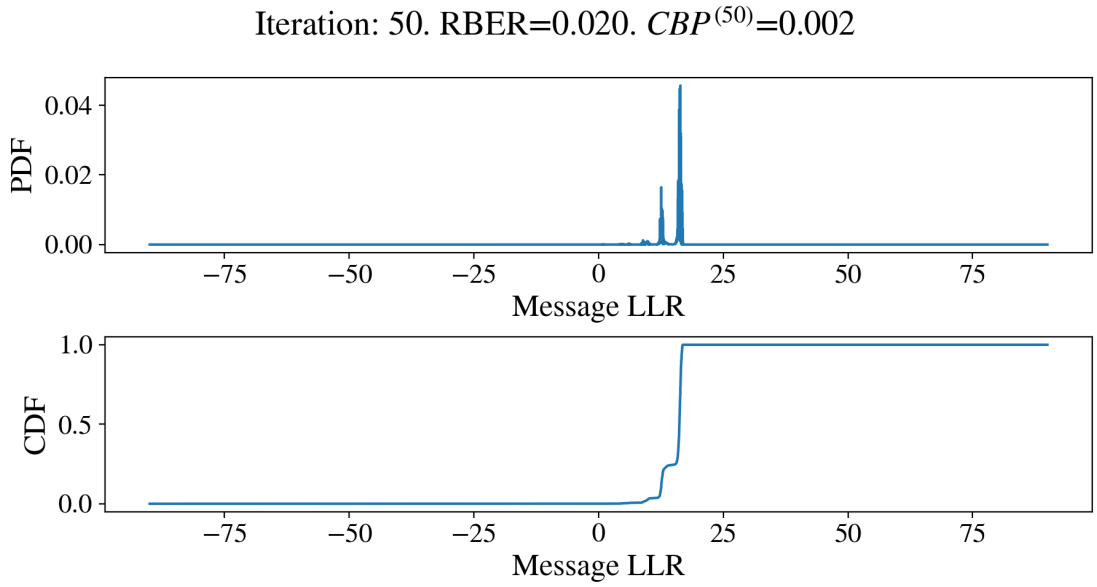


Figure 6.2: The pdf and CDF of message LLRs from variable node to check node at the 50^{th} iteration when RBER=0.02. At the top, the corresponding value of $CBP^{(50)}$ is given as well.

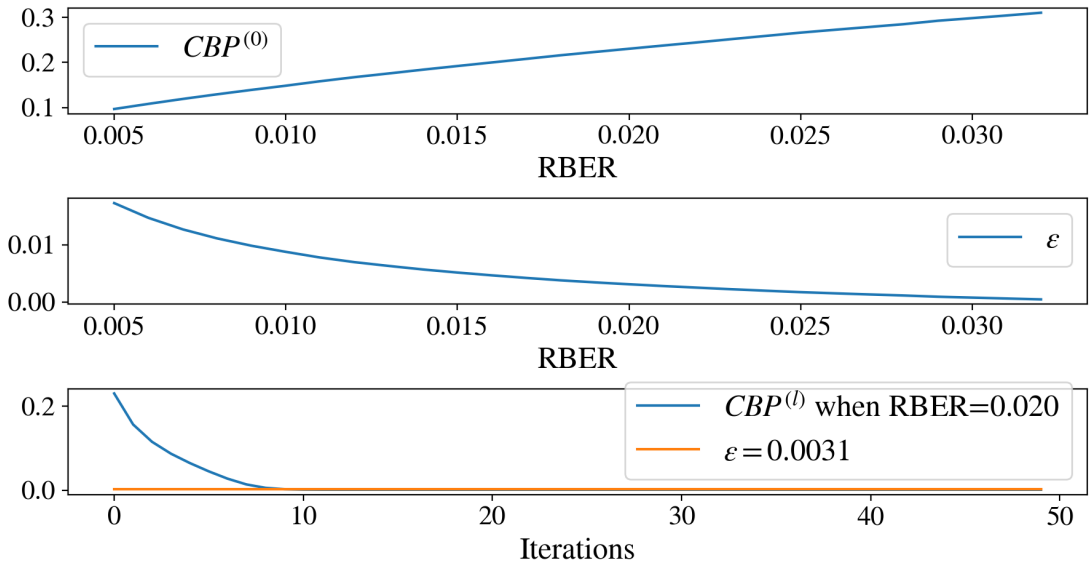


Figure 6.3: At the top: A graph showing how $CBP^{(0)}$ changes with RBER. In the middle: A graph showing how the computed ϵ changes with RBER. At the bottom: A graph showing how $CBP^{(l)}$ changes with iterations of density evolution when RBER=0.02. The graph shows that at approximately 10 iterations, the specific code ensemble goes below the threshold, ϵ.

Table 6.1: Local girth distributions for different parameters of the MM-QC-PEGA.

N	r	runtime $[s]$	4	6	8	10
1	1	2	30,5	69,5		
1	2	3	30,1	69,9		
4	1	90		31,5	15,7	52,8
4	2	1500		31,5	5,4	63,1
16	1	1200		1,8	18,4	79,9
16	2	86000		2,0	23,0	75,0
64	1	4800				100,0
128	1	13000				100,0

Table 6.2: Local girth distributions for optimized codes.

Description	10	12	14	16	18
1-bit optimized			35.6%	60.5%	3.9%
2-bit optimized	11.7%	75.6%	12.6%		

6.2 MM-QC-PEGA

Before optimizing the final codes through the MM-QC-PEG algorithm was evaluated on how well it maximized the local girth with different values on the scaling factor N and search depth r. Note that a smaller N in this phase does not necessarily restrict the size of the final code, it only limits how many shifts of the circulant matrices it considers. $N = 1$ for example considers only an identity matrix, although it can be of size 256. The results showing the percentage of variable nodes with local girths 4, ..., 10 and rough runtimes for different parameters are displayed in table 6.1 and it shows a quite clear result: N has a large effect, a higher r achieves a slightly better distribution for equal N but at a large computational cost. In fact, for $N = 16$ it performs worse after 24 hours of computations. In the end, a single search depth and scaling factor 64 was used for optimization since it had the best performance and in a reasonable time. The final optimal codes displayed in 6.2 achieved even better local girth distributions due to lesser density than the test code.

6.3 Decoding performance

Beyond benchmarking the optimized codes, three questions are targeted in this part. First, the performance gained from getting soft information by applying multiple thresholds is tried by running the same codes over a 1-bit, 2-bit, and full precision BI-AWGN-channel. Second, the impact of asymmetric errors is investigated by testing codes generated for a skewed BI-AWGN-channel. Finally, performance from the best code run with decoding algorithms of decreasing complexity is compared.

To give a neutral baseline to compare the result to, a random code was generated by selecting 2-3 non-zero elements of each column of the protograph and picking a random circulant matrix for them. The specified 1% RBER-limit and Shannon limit of the channel are also plotted, in dashed red and purple respectively, to help evaluation. Efforts were made by the authors to find ready-made codes or performance benchmarks fitting the specific use case but without success.

6.3.1 Impact of soft information

From the figures 6.4 - 6.6 it is clear that the performance gain by the optimized codes in the waterfall region is steeper, and improved by up to almost two decibels after the waterfall region. As expected more information enables better encoding, both moving the waterfall region by 1-2 decibels and reaching an error floor up to four decibels further down, however just one extra bit of soft information makes a large difference in catching up this difference, improving the performance by almost 4 decibels at 1% RBER. Theoretically, the limit for all channels lies past this threshold and so the perfect code should manage good performance even for one bit of hard information. On the other hand, this analysis disregards error sources such as limited decoding precision and simplified decoding schemes. The 1-bit optimized and 2-bit optimized codes perform similarly, with the only real difference being a lower error floor over the full precision BI-AWGN channel.

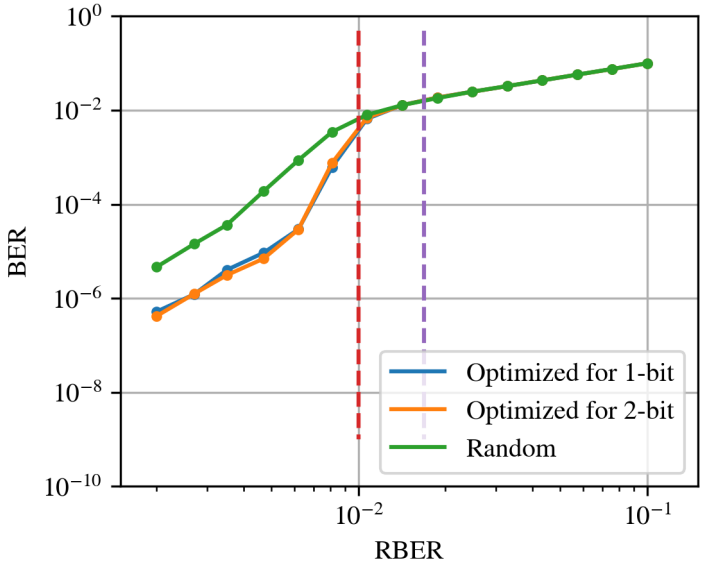


Figure 6.4: Comparison of codes optimized for three different channels, simulated over the 1-bit BI-AWGN-channel. The red dashed line to the left indicate 1% RBER, and the purple dashed line to the right shows the Shannon limit of the channel.

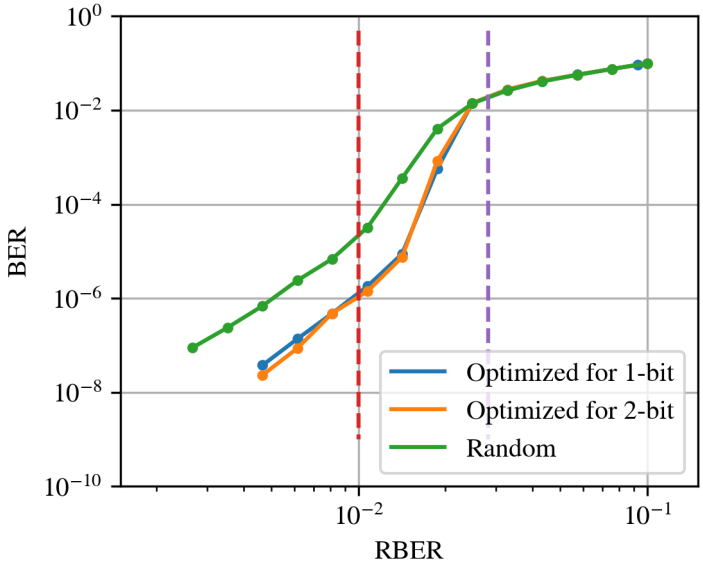


Figure 6.5: Comparison of codes optimized for three different channels, simulated over the 2-bit BI-AWGN-channel. The red dashed line to the left indicate 1% RBER, and the purple dashed line to the right shows the Shannon limit of the channel.

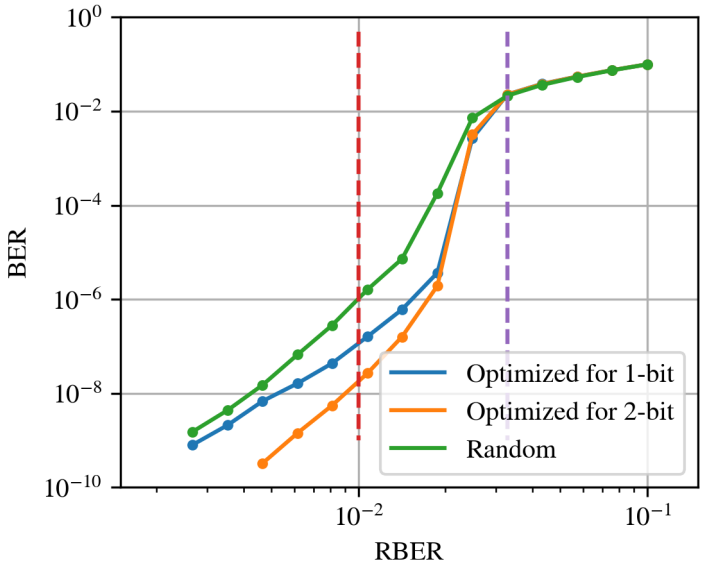


Figure 6.6: Comparison of codes optimized for three different channels, simulated over the full bit BI-AWGN-channel. The red dashed line to the left indicate 1% RBER, and the purple dashed line to the right shows the Shannon limit of the channel.

6.3.2 Impact of asymmetry

For asymmetry, the result of nine different experiments can be seen in figure 6.7. Different values of η were tried over the 1-bit and 2-bit channels, and two codes were tried and optimized for the asymmetric distribution using asymmetric density evolution as described. The results show that asymmetry deteriorates the decoding performance, $\eta = 0.3$ degrading the correction completely for the 1-bit channel, but on the other hand, the effect of $\eta = 0.45$ over the 2-bit channel has minimal effect. To say whether asymmetry will have a large effect on performance quantitative measurements of the voltage distributions would therefore be required. Moreover, it is not obvious that the channel specific optimization using the asymmetric density evolution improves the code in this case either because of the minimal change of the waterfall threshold.

6.3.3 Impact of decoding complexity

In this experiment the full Sum-Product Algorithm with a flooding scheduling scheme was compared to the Min-Sum algorithm and Normalized Min-Sum simplifications, results show that the simplifications in this configuration worsen the decoding performance by around half a decibel. The overall shape of the decoding curve stays the same and the waterfall threshold is not affected.

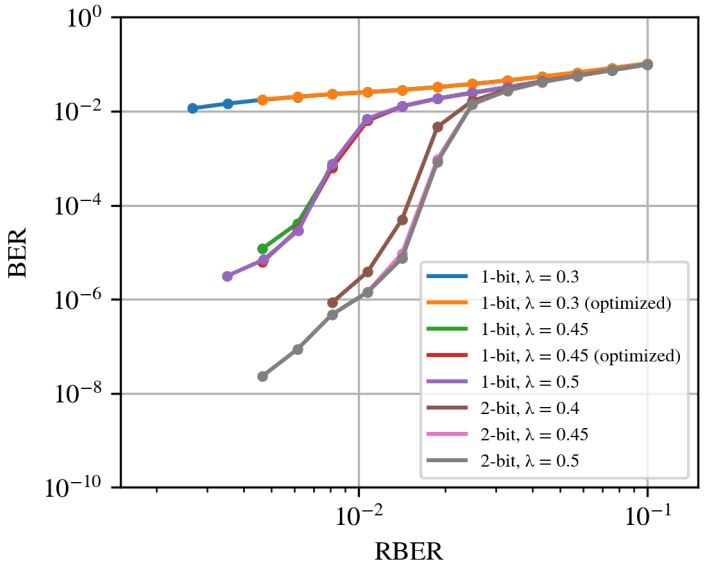


Figure 6.7: Decoding over many different skewed distributions.

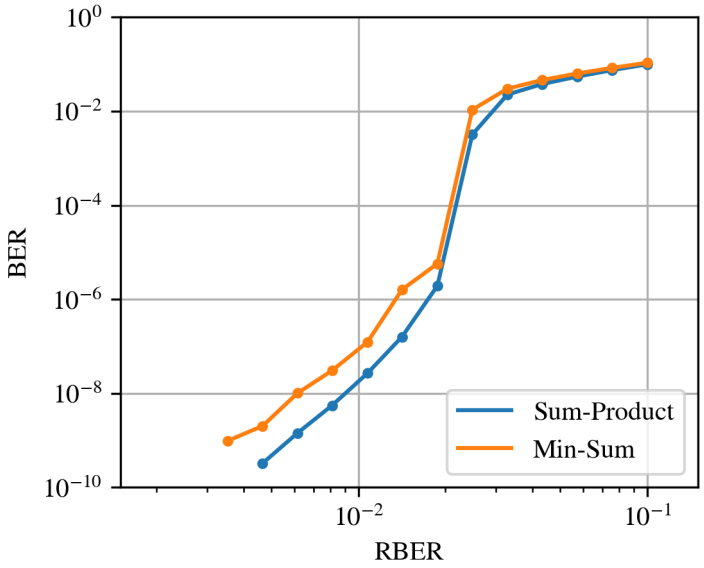


Figure 6.8: Comparison of different decoding schemes using the best-generated code over the full precision BI-AWGN channel.

7

Discussion and Conclusion

7.1 Discussion

7.1.1 Algorithmic improvements

Certain improvements on the algorithm implementations are left to be made from the insight of this work. For density evolution, the channel-specific optimized codes did for example not improve the waterfall region as expected, and the convergence criteria are dependent on hyper-parameters such as tolerance limits and discretizations in a quite non-transparent way. In the algorithm, the LLR-distribution is quantized onto a grid which introduce inaccuracy, the γ-transformation results in very large and very small numbers where floating point precision starts becoming an issue, and the discrete Fourier-transformation on a finite grid results in artifacts. These are some areas where more careful analysis is needed. During development, the algorithm has been compared to a sampling-based simulation of the density evolution and found to give similar results, but during many iterations, small errors may accumulate.

Another reason for lacking improvement may also be that the codes perform so close to capacity that the reminding gain is very unlikely for the stochastic genetic algorithm to find, or that the approximations of the density evolution do not provide exact enough results. If that is the case, some type of local exhaustive search could be needed to push the performance further. Finally, applying density evolution to the Min-Sum algorithm has previously been studied and implementing that could help bridge some of the performance loss while keeping the decreased complexity [39], [40].

More thorough testing of the waterfall optimization is perhaps of even larger use, as the error floor has more room for improvement than the waterfall threshold and here only one of many algorithm was tested. Research questions here could for example include testing metrics other than the local girth and speeding up the MM-QC-PEGA according to suggestions made in the original paper to allow quicker iteration, but also implementing other constructions [21]. Another question that was not investigated further was how the occurrence of trapping sets changes with asymmetry, which is mentioned in [1].

7.1.2 Future topics

Outside the scope of this thesis, an investigation into the hardware component would be beneficial for more precise quantitative studies, mainly fitting a noise model after real-world voltage measurements and finding a more realistic model for how noise and threshold-setting interact rather than assuming perfect knowledge of noise and corresponding thresholds. Moreover, that could help develop a more careful and quantitative analysis of skew, which in this project never reached any convincing results in simulations. On the same topic of quantitative study, going into more detail on the performance of outer codes to find a limit on how low the error floor needs to be and including effects of the puncturing and shortening would help create more clear constraints on how well the code must perform.

A couple of interesting concepts found in the literature but left out due to scope must also be mentioned. Of particular interest for NAND flash are the non-binary LDPC-codes, used for scenarios where all bits of one cell are read at once rather than one at a time. This yields an output alphabet of multiple symbols which is more noise sensitive but also gives more information for the decoder to exploit. Such a project would need to investigate both new design tools and decoding algorithms [1]. Another technique is the spatially-coupled code for which neighboring code words are interconnected in a special way which also has been shown to give good results [41].

7.2 Conclusion

One main intention of this thesis is to provide an overview of important aspects of error correction using LDPC codes in NAND-flash memories, starting from the hardware and connecting it to theory and existing design algorithms, as well as describing components such as puncturing, shortening, and threshold setting needed for the final implementation. Thinking of the NAND flash channel as a discretized discrete-continuous channel has been shown to be useful for comparing different amounts of soft information and clarifying the connection between σ of the BI-AWGN channel and the RBER. Finally, all ideas have been packaged together in a full design pipeline in three `python`-scripts and a modification of the `C++`-library Aff3ct for performance simulations. Since the Aff3ct has been in development for many years it provides different decoding schemes and floating point quantizers straight out of the box for helping with continued testing.

The simulation results further confirm the picture of QC-LDPC codes as a near capacity-reaching solution for error correction and all designed codes have a waterfall region within 0.5dB of the theoretical limit, a limit which surpasses the specification of managing to correct for an RBER of 1% even for the hard quantization. One bit of soft information adds a significant decoding gain of close to 4dB which makes the extra complexity worth the effort of this analysis, full precision leads to almost 2dB more albeit at higher complexity. In comparison, a skewness of $\eta = 0.4$ over the 2-bit channel or using the Min-Sum over Sum-Product decoding led to a loss of

around half of a decibel. With some more development, and by examining the exact requirements on the error correction, a good trade-off should be possible to design in this fashion.

References

[1] L. Dolecek and Y. Cassuto, "Channel coding for nonvolatile memory technologies: Theoretical advances and practical considerations," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1705–1724, 2017.

[2] S. Aritome, *NAND flash memory technologies*. John Wiley & Sons, 2015.

[3] C. Wang and W.-F. Wong, "Extending the lifetime of nand flash memory by salvaging bad blocks," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 260–263. DOI: 10.1109/DATE.2012.6176473.

[4] R. Micheloni *et al.*, *3D Flash memories*. Springer, 2016.

[5] S. Rubini and J. Boukhobza, "Non-volatile memories: A new deal for operating system design?" In *Workshop" Gestion des resources dans le Cloud"*, 2018.

[6] B. Li, B. Yan, and H. Li, "An overview of in-memory processing with emerging non-volatile memory for data-intensive applications," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 381–386.

[7] Y. Chen, "Reram: History, status, and future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[8] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND flash memories*. Springer Science & Business Media, 2010.

[9] J. Wang, K. Vakilinia, T.-Y. Chen, *et al.*, "Enhanced precision through multiple reads for ldpc decoding in flash memories," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 880–891, 2014.

[10] Q. Li, A. Jiang, and E. F. Haratsch, "Noise modeling and capacity analysis for nand flash memories," in *2014 IEEE International Symposium on Information Theory*, IEEE, 2014, pp. 2262–2266.

[11] J. Cooke, "The inconvenient truths of nand flash memory," *Flash Memory Summit*, vol. 3, no. 3, pp. 3–1, 2007.

[12] *Nandfctrl2 ip core user's manual*, NANDFCTRL2, Version 2022.1, Cobham Gaisler AB, Apr. 2022.

[13] N. Papandreou, H. Pozidis, T. Parnell, *et al.*, "Characterization and analysis of bit errors in 3d tlc nand flash memory," in *2019 IEEE International Reliability Physics Symposium (IRPS)*, IEEE, 2019, pp. 1–6.

[14] R. Micheloni, A. Marelli, and K. Eshghi, *Inside solid state drives (SSDs)*. Springer, 2013.

[15] M. Rajab, *Channel and source coding for non-volatile flash memories*. Springer, 2020.

[16] C. Zambelli, L. Crippa, R. Micheloni, and P. Olivo, "Cross-temperature effects of program and read operations in 2d and 3d nand flash memories," in *2018 International Integrated Reliability Workshop (IIRW)*, 2018, pp. 1–4. DOI: 10.1109/IIRW.2018.8727102.

[17] D. Weimer, E. Sutton, M. Mlynczak, and L. Hunt, "Intercalibration of neutral density measurements for mapping the thermosphere," *Journal of Geophysical Research: Space Physics*, vol. 121, no. 6, pp. 5975–5990, 2016.

[18] Q. Li, Q. Wang, Q. Xu, and Z. Huo, "A fast read retry method for 3d nand flash memories using novel valley search algorithm," *IEICE Electronics Express*, pp. 15–20 180 921, 2018.

[19] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[20] W. Ryan, *An introduction to ldpc codes*, 2004.

[21] X. He, L. Zhou, and J. Du, "Peg-like design of binary qc-ldpc codes based on detecting and avoiding generating small cycles," *IEEE Transactions on Communications*, vol. 66, no. 5, pp. 1845–1858, 2017.

[22] A. Cassagne, O. Hartmann, M. Leonardon, *et al.*, "Aff3ct: A fast forward error correction toolbox!" *SoftwareX*, vol. 10, p. 100 345, 2019.

[23] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE transactions on information theory*, vol. 47, no. 2, pp. 619–637, 2001.

[24] B. Peleato, R. Agarwal, J. M. Cioffi, M. Qin, and P. H. Siegel, "Adaptive read thresholds for nand flash," *IEEE Transactions on Communications*, vol. 63, no. 9, pp. 3069–3081, 2015.

[25] Y. Zhang and W. E. Ryan, "Toward low ldpc-code floors: A case study," *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1566–1573, 2009.

[26] Borkowski, Bonk, de Lind van Wijngaarden, and Schmalen, "Ldpc code length reduction," in *100G-EPON Task Force Meeting*, Orlando, Florida, 2017, pp. 4–5. DOI: 10.1109/ACSSC.2004.1399517.

[27] D. G. Mitchell, M. Lentmaier, A. E. Pusane, and D. J. Costello, "Randomly punctured ldpc codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 408–421, 2015.

[28] T. Van Nguyen, A. Nosratinia, and D. Divsalar, "The design of rate-compatible protograph ldpc codes," *IEEE Transactions on communications*, vol. 60, no. 10, pp. 2841–2850, 2012.

[29] A. Casado, W.-Y. Weng, and R. Wesel, "Multiple rate low-density parity-check codes with constant blocklength," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2, 2004, 2010–2014 Vol.2. DOI: 10.1109/ACSSC.2004.1399517.

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

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

[32] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[33] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[34] A. Price and J. Hall, "A survey on trapping sets and stopping sets," *arXiv preprint arXiv:1705.05996*, 2017.

[35] L. Lan, L. Zeng, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "Construction of quasi-cyclic ldpc codes for awgn and binary erasure channels: A finite field approach," *IEEE Transactions on Information Theory*, vol. 53, no. 7, pp. 2429–2458, 2007.

[36] A. Elkelesh, M. Ebada, S. Cammerer, L. Schmalen, and S. Ten Brink, "Decoder-in-the-loop: Genetic optimization-based ldpc code design," *IEEE access*, vol. 7, pp. 141 161–141 170, 2019.

[37] J. Andrade, N. George, K. Karras, *et al.*, "Design space exploration of ldpc decoders using high-level synthesis," *IEEE Access*, vol. 5, pp. 14 600–14 615, 2017.

[38] H. Chen and Z. Cao, "A modified peg algorithm for construction of ldpc codes with strictly concentrated check-node degree distributions," in *2007 IEEE Wireless Communications and Networking Conference*, IEEE, 2007, pp. 564–568.

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

[40] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No. 01CH37270)*, IEEE, vol. 2, 2001, pp. 1021–1025.

[41] K. Takeuchi, "Recent advances in spatial coupling–a review of spatially coupled ldpc codes," *IEICE Technical Report; IEICE Tech. Rep.*, vol. 113, no. 319, pp. 27–34, 2013.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY