Channel Coding for Flash Memories

Doctoral thesis for obtaining

the academic degree

Doctor of Engineering Sciences

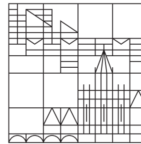
(Dr.-Ing.)

submitted by

Spinner, Jens

at the

Universität
Konstanz



Faculty of Science

Department of Computer and Information Science

Konstanz, 2019

Konstanzer Online-Publikations-System (KOPS)
URL: http://nbn-resolving.de/urn:nbn:de:bsz:352-2-1ro71he3fo8zt0

Date of the oral Examination: 18.10.2019

1. Reviewer: Prof. Dr. Dietmar Saupe

2. Reviewer: Prof. Dr.-Ing. Jürgen Freudenberger

Acknowledgments

I like to thank Prof. Jürgen Freudenberger and Prof. Dietmar Saupe.

Especially, I like to express my sincere gratuite to my family.

Contents

List of Acronyms **XI**

1 Introduction **1**
1.1 Goals . . . 2
1.2 Thesis Structure . . . 2

2 Media and Channel Models **4**
2.1 Storage systems . . . 4
2.2 Flash Memories . . . 5
2.2.1 Flash Cell . . . 5
2.2.2 Error Characteristics . . . 8
2.2.3 Flash Layer Design . . . 9
2.2.4 Flash Channel Model . . . 9
2.3 Other Modern Storage Systems . . . 12
2.4 Summary . . . 13

3 Preliminaries for Channel Coding **15**
3.1 Galois Fields . . . 15
3.2 Channel Coding Basics . . . 18
3.2.1 Linear Block Codes . . . 18
3.2.2 Reed-Solomon Codes . . . 18
3.2.3 Bose-Chaudhuri-Hocquenghem Codes . . . 19
3.2.4 Algebraic Decoding . . . 20
3.3 Basic Error Models . . . 20
3.4 Summary . . . 21

4 Construction of high-rate Generalized Concatenated Codes for Flash Memories **22**
4.1 GCC Preliminaries . . . 23
4.1.1 Encoding . . . 24
4.1.2 Decoding . . . 26
4.2 Error Bounds . . . 27
4.3 Code Parameters . . . 29
4.4 Summary . . . 31

5 Interleaver Construction Based on Gaussian Integer **33**
5.1 Preliminaries . . . 34
5.2 Set Partitioning . . . 35
5.3 Interleaver Design . . . 39

5.4 GCC for Correcting Two-Dimensional Cluster Errors and Independent Errors . 41
5.4.1 Generalized Concatenated Codes . . . 42
5.4.2 Code Constructions . . . 43
5.4.3 Simulations . . . 45
5.5 Conclusions . . . 47

6 Soft-Decoding for Generalized Concatenated Codes 48
6.1 Sequential Stack Decoding . . . 48
6.1.1 Sequential Stack Decoding using a Single Trellis . . . 48
6.1.2 Supercode Decoding for nested BCH Codes . . . 50
6.1.3 List-of-two Decoding . . . 53
6.2 GCC Decoding and Decoding Error Probability . . . 56
6.2.1 Probability of a Decoding Error . . . 56
6.2.2 Comparison Error Correction Performance . . . 56
6.3 Other Soft Decoders . . . 59
6.4 Summary . . . 61

7 Implementation of an Algebraic BCH Decoder 62
7.1 A Mixed Parallel and Serial BCH Implementation . . . 62
7.1.1 Syndrome Computation . . . 63
7.1.2 Mixed Serial and Parallel BMA . . . 64
7.1.3 Multi Speed Chien Search . . . 65
7.1.4 Overall Considerations . . . 67
7.2 GF Optimization . . . 68
7.3 Summary . . . 72

8 Implementation of GCC with Hardinformation 73
8.1 GCC Data Matrix and Pipelined Decoder . . . 73
8.1.1 BCH Decoding . . . 75
8.1.2 Re-imaging . . . 76
8.1.3 RS decoding . . . 77
8.1.4 Hardware Architecture . . . 78
8.1.5 Encoder . . . 81
8.2 Codeword Buffer . . . 81
8.3 Comparison . . . 83
8.4 Improvements . . . 86
8.4.1 Speed . . . 86
8.5 Summary . . . 89

9 Implementation of GCC with Softinformation 90
9.1 Introduction . . . 90
9.2 Stack Algorithm and Supercode Decoding . . . 90
9.2.1 Decoding Logic . . . 91
9.2.2 Area Comparison and Throughput Estimation . . . 93
9.2.3 Concluding Remarks and Future Directions . . . 94
9.3 Chase Decoder . . . 94
9.3.1 GCC Soft-decoder Architecture . . . 96
9.3.2 Synthesis Result and Throughput Estimation . . . 97
9.4 Summary . . . 98

10 Conclusion **100**

Bibliography **100**

Abstract

Flash memories are non-volatile memory devices. The rapid development of flash technologies leads to higher storage density, but also to higher error rates. This dissertation considers this reliability problem of flash memories and investigates suitable error correction codes, e.g. BCH-codes and concatenated codes.

First, the flash cells, their functionality and error characteristics are explained. Next, the mathematics of the employed algebraic code are discussed. Subsequently, generalized concatenated codes (GCC) are presented.

Compared to the commonly used BCH codes, concatenated codes promise higher code rates and lower implementation complexity. This complexity reduction is achieved by dividing a long code into smaller components, which require smaller Galois-Field sizes. The algebraic decoding algorithms enable analytical determination of the block error rate. Thus, it is possible to guarantee very low residual error rates for flash memories.

Besides the complexity reduction, general concatenated codes can exploit soft information. This so-called soft decoding is not practicable for long BCH-codes. In this dissertation, two soft decoding methods for GCC are presented and analyzed. These methods are based on the Chase decoding and the stack algorithm. The last method explicitly uses the generalized concatenated code structure, where the component codes are nested subcodes. This property supports the complexity reduction.

Moreover, the two-dimensional structure of GCC enables the correction of error patterns with statistical dependencies. One chapter of the thesis demonstrates how the concatenated codes can be used to correct two-dimensional cluster errors. Therefore, a two-dimensional interleaver is designed with the help of Gaussian integers. This design achieves the correction of cluster errors with the best possible radius.

Large parts of this works are dedicated to the question, how the decoding algorithms can be implemented in hardware. These hardware architectures, their throughput and logic size are presented for long BCH-codes and generalized concatenated codes. The results show that generalized concatenated codes are suitable for error correction in flash memories, especially for three-dimensional NAND memory systems used in industrial applications, where low residual errors must be guaranteed.

Zusammenfassung

Flashspeicher sind nichtflüchtige Speicherbausteine, deren rasante Entwicklung zu höheren Dichten und damit verbunden zu höheren Fehlerraten führt. Die Dissertation greift diese Problematik auf und untersucht für Flashspeicher geeignete Fehlerkorrekturverfahren, wie BCH-Codes und die verallgemeinerte Codeverkettung.

Zunächst wird auf die Flashzellen eingegangen, deren Funktionsweise und das resultierende Fehlerverhalten erläutert. Die Basis für die Fehlerkorrektur bilden algebraische Codes, deren mathematische Grundlagen beschrieben werden. Im Anschluss wird die verallgemeinerte Codeverkettung vorgestellt.

Verkettete Codes ermöglichen eine höhere Coderate und geringere Implementierungskomplexität im Vergleich zu den bisher noch überwiegend verwendeten BCH-Codes. Die Reduktion der Komplexität folgt aus der Aufteilung langer Codeworte in kleinere Komponenten, die aufgrund der kleinen Galois Feldgröße mit geringem Aufwand decodiert werden können. Ihre algebraische Dekodierung ermöglicht eine analytische Bestimmbarkeit der Blockfehlerrate. Daher können die für Flashspeicher erforderlichen geringen Restfehlerwahrscheinlichkeiten garantiert werden.

Neben der geringeren Komplexität ermöglicht die verallgemeinerte Codeverkettung auch eine Decodierung mit Zuverlässigkeitsinformation. Diese sogenannte Soft-Decodierung ist für lange BCH-Codes nicht mit vertretbarem Aufwand möglich. In dieser Dissertation werden zwei Soft-Decodierverfahren vorgestellt und untersucht. Diese Verfahren basieren auf der Chase-Decodierung und auf der Listendekodierung mittels Stack-Algorithmus. Letzteres Verfahren nutzt explizit die Struktur der verallgemeinerten Codeverkettung aus, bei der die Komponenten-Codes aus einer Reihe von Teilcodes besteht. Diese Struktur hilft, die Decodierkomplexität zu reduzieren.

Außerdem eignet sich die zweidimensionale Struktur der verallgemeinerten Codeverkettung gut für die Korrektur von Fehlermustern mit statischen Abhängigkeiten. Ein Kapitel der Dissertation ist der Frage gewidmet, wie die Codeverkettung für die Korrektur von zweidimensionalen Clusterfehlern verwendet werden kann. Dazu wird mithilfe von Gauß-Zahlen ein Interleaving bestimmt, das die Korrektur von Clusterfehlern mit maximalem Radius ermöglicht.

Desweitern widmet sich ein Großteil der Arbeit der Frage, wie diese Codierverfahren möglichst günstig in Hardware implementiert werden können. Sowohl für lange BCH-Codes als auch für die Decodierung der verketteten Codes werden Hardware-Architekturen entworfen, der erzielbare Durchsatz und der Flächenbedarf für die Logikelemente bestimmt. Diese Ergebnisse belegen, dass die verallgemeinerte Codeverkettung für die Fehlerkorrektur in Flashspeichern gut geeignet ist. Dies gilt insbesondere für 3D-NAND Speichersysteme im industriellen Einsatz, die sehr niedrige Restfehlerwahrscheinlichkeiten gewährleisten müssen.

List of Acronyms

ARQ	automatic repeat-request
ASK	amplitude shift keying
ASIC	application-specific integrated circuit
AWGN	additive white gaussian noise
BCH	Bose Chaudhuri Hocqenhem
BER	bit error rate
BL	bit line
BMA	Berlekamp-Massey algorithm
BMD	bounded minimum distance
BWE	bit write enable
BSC	binary symmetric channel
BSEEC	binary symmetric error and erasure channel
CIRC	Cross-interleaved Reed-Solomon Code
CD	Compact Disc
CF	Compact Flash
CPU	central processing unit
DFT	discrete Fourier transformation
DMA	direct memory access
DP-SRAM	dual-port static random-access memory
SP-SRAM	single-port static random-access memory
DVB-T	terestrical digital video broadcast
ECC	error correction code
EOL	end of lifetime
FEC	forward error correction
FER	frame error rate
FET	field effect transistor
FNT	Fowler-Nordheim tunneling
FPGA	field programmable gate arrays
FG	floating gate
GCC	generalized concatenated code
GE	gate equivalent
GF	Galois field
GSL	ground selector line
HDD	hard disc drive
iBMA	inversionless Berlekamp-Massey algorithm
IDFT	inverse discrete Fourier transformation
JEDEC	Joint Electron Device Engineering Council
LDPC	low-density parity check
LLR	log-likelihood ratio

LRP	least reliable bit position
LP	lower page
LFSR	linear feedback shift register
LUT	look-up table
ML	maximum-likelihood
MLC	multi-level cell
MMC	Multimedia Card
MP	middle page
OFDM	orthogonal frequency-division multiplexing
PCIe	Peripheral Component Interconnect Express
QR	quick response code
RAID	redundant array of independent disks
RBER	raw bit error rate
RS	Reed-Solomon
RSV	Reed-Solomon/Viterbi
SATA	Serial AT Attachement
SD	Secure Digital Memory Card
SLC	single-level cell
SNR	signal to noise ratio
SPC	single parity check
SRAM	static random-access memory
SSD	solid-state drive
SSL	sensing selector line
TLC	triple-level cell
UP	upper page
UPIBA	universal parallel inversionless Blahut algorithm
USB	Universal Serial Bus
VLSI	very large scale integration
WL	word line

Nomenclature

$\mathcal{A}$ outer code

$\mathbf{a}$, $a(x)$ outer codeword vector/polynomial

$\mathcal{B}$ inner code

$\mathbf{b}$, $b(x)$ inner codeword vector/polynomial

$\mathcal{C}$ code

$\mathbf{c}$, $c(x)$ codeword vector/polynomial

d, d_a, d_b codeword distance

$\mathbf{e}$, $e(x)$ error vector/polynomial

$g(x)$ generator polynomial

$\mathbf{i}$, $i(x)$ information vector/polynomial

k, k_a, k_b codeword dimension

L number of code levels

n, n_a, n_b codeword size

$p(x)$ primitive polynomial

$\mathbf{r}$, $r(x)$ received vector/polynomial

R code rate

r number of redundancy symbols

$\mathbf{S}$, $S(x)$ Syndrom values vector/polynomial

t, t_a, t_b error correction capability

? erasure symbol

α primitive element

ε channel error probability

λ channel erasure probability

$\Omega(x)$ error-value polynomial

$\sigma(x)$ error-location polynomial

Chapter 1

Introduction

Thousands of years ago, stones were used to store information. Today, storage systems continue their important role on the second most element on the earth crust, silicon (Si), the element from which Flash memories are produced.

Medias are physical information carriers. The most common modern medias can be divided into optical or magnetic discs or tapes and Flash memories. In optical systems, mainly the surface reflection or the transmittance of a photon source is measured. With the magnetic discs used in hard disk drives, the magnetic polarization of a surface is measured with coils. Flash memory uses floating gates, where electrons can be loaded and unloaded into an insulated gate using the tunnel effect. These non-volatile trapped electrons in the floating gate form an electrical field, which can be measured. The advantage of disc-based systems over Flash memory is the absence of moving mechanical elements. At present, it can be observed that the density has increased over time while the reliability of storage systems typically has decreased with each new technology. As the silicon-based structure shrink is reaching its limits given by the lattices, the so-called 3D technology tries to overcome this issue by increasing the layers of the die.

Each of these media share in common the fact that they are not absolutely accurate. Errors can occur during the writing and reading process as well as in idle state. An additional challenge in storage systems is that the sender cannot request retransmit erroneous data using automatic repeat-requests (ARQs), and thus forward error correction (FEC) codes are applied. The error correction ensures that the requirements regarding the reliability for a given application are met. In coding theory, many error correction codes (ECCs) exists with different properties that can be divided into complexity characteristics and their error correction performance. As already mentioned, the provable error correction boundaries are an important property. In order to check whether a code is applicable it can be simulated or analyzed analytically. Simulation is possible if the target error rate is high. If a code has to be tested for a very low error rate, the simulation can exceed all available resources such as time and costs. Algebraic codes are then taken into account. These codes can be decoded with bounded minimum distance (BMD) decoding, where the residual error rate can be calculated. Bose Chaudhuri Hocqenhem (BCH) and Reed-Solomon (RS) codes are the most famous algebraic codes.

NAND flash memories are important components in embedded systems as well as consumer electronics. The flash cells keep their electrical charge without a power supply. However, errors may occur while the information is read. Consequently, error correction coding ECC is required to ensure data integrity and reliability for the user data [51, 63, 16]. In the past, mostly BCH codes with hard-input algebraic decoding were used for error correction [51, 76] and [83].

Reliability information about the state of the cell can significantly improve the ECC performance [17]. Soft-input decoding algorithms are required to exploit the reliability information. For instance, low-density parity check (LDPC) can provide strong error-correcting performance in NAND flash memories [77, 68, 43, 32, 30, 79]. However, the implementation complexity of an LDPC decoder can be high when high data throughput should be achieved. Concatenated codes constructed from long BCH codes can achieve low residual error rates [13, 36, 37], although they require very long codes and hence a long decoding latency, which might not be acceptable for all applications of flash memories. LDPC codes have high residual error rates (the error floor) and are not suitable for applications that require very low decoder failure probabilities [81]. For instance, the Joint Electron Device Engineering Council (JEDEC) standard for solid-state drive (SSD) recommends an uncorrectable bit error rate of less than 10^{-15} for client applications and of less than 10^{-16} for enterprise solutions [1]. For some applications, block error rates less than 10^{-16} are required [41].

1.1 Goals

The initial impulse of this work was the demand of improvements of error correction codes for Flash memories in industrial applications. This impulse triggered a first investigation of several code types and their performances, which led to the design of an ECC candidate and finally its implementation in hardware [81]. The choice of an ECC candidate is determined by the industrial requirements, placing an emphasis on the verifiability of the error correction capability. Additional requirements for mass-storage systems include the ECCs demand for overhead, costs for implementation and throughput.

From this preliminary investigation, the generalized concatenated code (GCC) emerged as a favorite. In this work, the code construction and its underlying theory will be explained and it will be described why this code is suitable for industrial applications. Subsequently, a strong focus of this work lies on the implementation of the necessary algorithms in field programmable gate arrayss (FPGAs) and application-specific integrated circuit (ASIC), where the aspects of area and throughput are highlighted.

We demonstrate that GCC can achieve such low residual error rates. ECC based on GCC has a high potential for various applications in data communication and data storage systems, e.g. for digital magnetic storage systems [20] and non-volatile flash memories [81, 92]. These codes are typically constructed from short inner binary BCH codes and outer RS codes [19, 80]. Hence, the hard-input decoding is based on low-complexity algebraic decoding. New constructions have recently proposed that enable higher code rates [98] and [78], where the inner codes are extended BCH codes. In particular, single parity check (SPC) codes are used in the first level of the GCC. A soft-input decoding algorithm for GCC was proposed in [97], where the decoding is based on sequential stack decoding of the inner codes.

1.2 Thesis Structure

This thesis starts with the brief presentation of several modern storage medias, their functionality and error characteristics in Chapter 2. It then concentrates on Flash memories and describes error models that are used to design ECC codes.

Chapter 3 offers an introduction to algebraic codes and its basis are introduced, namely the Galois fields. This is the preliminary for the GCC encoding decoding scheme as its error bounds and code parameter design, which is described in Chapter 4.

In Chapter 5, an interleaving concept based on Gaussian integers combined with GCC codes is presented that can decode special shaped burst errors. This work was published in [86, 84, 85].

The algebraic decoder described in chapters 3 and 4 is limited to decoding information with binary quantization, also called hard information. In Chapter 6, techniques and their use for GCC are introduced, which enable higher quantization levels and thus increases the error correction performance. Research in constructing high-rate was published in [97].

Chapters 7, 8 and 9 describe the hardware implementation of the algorithms. In [82, 95, 83], an efficient algebraic decoder for large block size BCH codes are published. Its architecture, impact on complexity and throughput is discussed in Chapter 7. The implementation and its analysis for hard information GCC en- and decoder in very large scale integration (VLSI) is described in Chapter 8. Publications on hard information GCC implementations include [91, 99, 92]. This GCC implementation is then extended by a soft information decoder in Chapter 9, which were published in [96, 87, 93, 94, 98, 101].

Chapter 2

Media and Channel Models

Storage medias are carriers that store information in a non-volatile manner over a period of time. This chapter provides an overview of modern storage medias, its functionality and channel models. In Section 2.2, the focus lies on Flash memories. Using these devices will lead to the problem that the original information is disturbed by several effects that even grow over the lifetime of this memory. In order to encounter the problem of erroneous data that is read back from a device, ECCs are used. A huge variety of ECC types exist with different parameters. A proper choice is essential to address the demands for different areas such as industrial applications where a provable error bound is an important requirement. In Section 2.2.4, channel models are discussed to assess this either analytically or these curves are measured by Monte Carlo simulation. At the end of this chapter, the reader will know why errors in Flash memories occur and how they can be modeled. Because there are similarities with other carriers magnetic and optical carriers are also considered in Section 2.3.

2.1 Storage systems

In systems like mobile phones, cloud storage, different-scaled computers, embedded systems and others, non-volatile memories are integrated. An example combination for Flash storage systems is depicted in Figure 2.1, which shows three components: Flash device, Flash micro controller and a host. The Flash device is connected via the Flash channel with a micro controller's Flash interface that is connected using the host channel with a host. A standardized Flash interface is specified in [56]. Further examples for links between the host and micro controller are SATA, USB, PCIe or SD, where each standard has different application areas. For data transportation, storage systems with pluggable links like USB-sticks or SD-cards are available. In some cases, the carrier is a removable device that must be read by a reading device, like floppy discs, CDs, MMCs or quick response code (QR) codes on different materials. Therefore, the reading device comprises a micro controller that provides the interfacing between the carrier and other systems. A common Flash micro controller has a CPU running firmware on it that manages the data stored on the devices and controls the communication in both directions between the host and device. Further modules serve features like cryptography, DMA mechanisms, data compression and error correction.

Application areas of memories are manifold. Some examples of data that is stored include measurements, personal data, accounting information, operating systems and programs. A few application examples are the control logic for robot arms, medical instruments, routers, cellphone base stations, large machine control units, drill heads and automotive navigation systems.

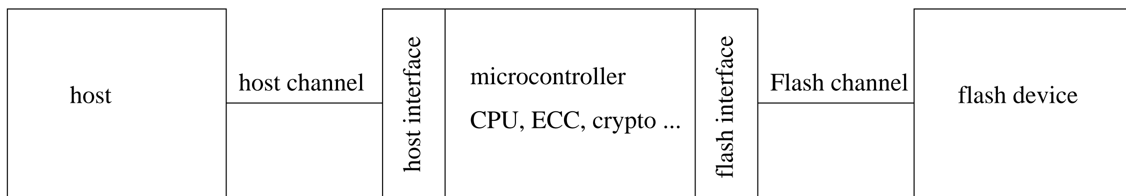


Fig. 2.1: Flash microcontroller

Similar to the applications, its requirements are also manifold. High storage capacity and data density or high reliability and data throughput are some examples. An additional factor is the data overhead needed for the data management and the redundancy for error correction. Moreover, data integrity, data encryption and long time storage are subjects that have to be taken into account. Similar to an ECC, the entire storage system has to be designed depending on its requirements given by the application.

In order to provide an introduction to storage devices with different carrier materials, the following sub-sections present four categories of storage devices.

2.2 Flash Memories

Flash memories are silicon-based semiconductors that can store data in a non-volatile manner. The first models were presented by Fuji Masuoka (Thoshiba) in 1984. They are divided into NAND and NOR technology, where NAND memories are used for mass storage and NOR traditionally for code storage due to the faster access time. Nowadays development is proceeding towards higher density by shrinking the silicon structure size and using more layers, called the 3D-NAND Flash memory.

2.2.1 Flash Cell

Figure 2.2 shows a floating gate (FG) cell. It is similar to a FET and comprises four connecting elements, source, drain, control gate and the bulk substrate. An additional FG is surrounded by isolating oxide between the control gate and the substrate. This isolation makes this FG an electron trap that holds information for a longer time period [51]. An FG has three basic functionalities, namely reading, writing and erasing the FG state. Figure 2.3 depicts FGs in a NAND matrix. In one dimension, these cells are connected in series. One of these series comprising several NANDs is called a bit line (BL). In the other dimension, all gates are connected in parallel to a word line (WL), which represents a page. All pages that are connected with the same BL are called a Flash block. The ground selector line (GSL) and the sensing selector line (SSL) are necessary to apply the different read, write and erase modes. A Flash memory die comprises several Flash blocks.

The cell has a designed ideal load level for a distinct state that represents '1' or '0' for a single-level cell (SLC). In multi-level cell (MLC) memories, each cell has four designed states and in triple-level cell (TLC) cells eight states. Flash memories with more than eight states per cell are already described in literature. However, these states are not accurate, caused by several effects. This inaccuracy shifts the threshold Voltage V_{TH}. Figure 2.4 briefly depicts the distribution of V_{TH} for all TLC states.

The data within a page is organized in several system blocks like 512, 1k, 2k or 4k bytes with additional system meta information and ECC redundancy. If the FG cells are MLC

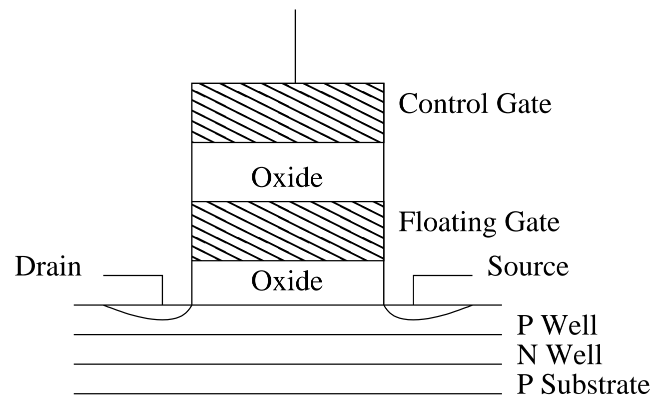


Fig. 2.2: Floating Gate

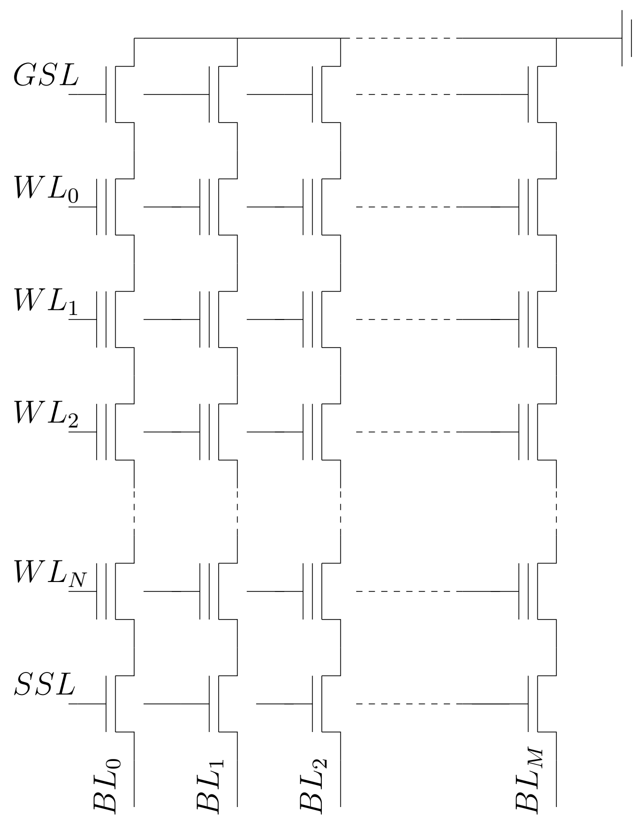


Fig. 2.3: NAND structure

	ERA	*A*	*B*	*C*	*D*	*E*	*F*	*G*
LP	1	1	1	1	0	0	0	0
MP	1	1	0	0	0	0	1	1
UP	1	0	0	1	1	0	0	1

(a)

	ERA	*A*	*B*	*C*	*D*	*E*	*F*	*G*
LP	1	0	0	0	0	1	1	1
MP	1	1	0	0	1	1	0	0
UP	1	1	1	0	0	0	0	1

(b)

Tab. 2.1: Flash state level encoding with (a) 1:2:4 and (b) 2:3:2 levels

or TLC, an WL represents a shared page that comprises two or three pages with the same electrical address but different logical addresses with a special arrangement such that a writing causes a low electrical bias [7]. These pages are named lower page (LP), middle page (MP) and upper page (UP). A system block does not cross into different page levels.

The FG states of a TLC are the erased state ERA and the programmed states A, B, C, D, E, F, G. Two exemplar different state-level encodings are shown in Table 2.1. Depending on the page level the reference voltage is between different state levels. In Table 2.1(a), the LP reference is between C and D, and MP between A and B, E and F. Under the assumption of random data, the encoding (a) has a smaller bit error rate (BER) in the LP, a moderate BER in the MP and its highest BER in the UP. An alternative encoding is shown in (b) where, in comparison to (a), the LP and MP has an increased BER but the UP level has a lower BER.

In order to understand the electrical circuit of a NAND Flash memory, we briefly present the basic access functions for the FGs:

Read In the read operation, the cell gates in an WL that is read is put to V_{READ} while all other WL are tied to $V_{PASS,R}$. Only the electrical load from the FG cell that is in read mode affects the current that flows through the BL. This current is measured using an integrator circuit and a configurable threshold voltage V_{TH}.

V_{TH} has to be determined by either a built-in mechanism inside the Flash memory or with single state reads from the Flash controller. With these single state reads, the controller scans the changes of ones to zeros with fine-grained V_{TH} steps. For symmetric error probabilities, the optimal V_{TH} for a distinct state level is where the change from one to zero between two V_{TH} steps is minimal.

If information is read with additional reliability information, several offsets per state level are read. This results in soft bits for each information (hard) bit. Because the distributions of the state levels vary, it also holds interest which state level was read. This information is converged by the level indicator bits. Figure 2.4 shows the V_{TH} distribution and Figure 2.6 exemplar soft information thresholds for one level.

Program The programming operation tunnels electrons onto the FG using the Fowler-Nordheim tunneling (FNT). An electron injection is triggered with the following configuration. All unselected gates in the BLs are in passing mode with $V_{PASS,P}$, the selected gates within a WL are connected to a much higher gate voltage V_{PRG}. Those BLs that are programmed are tied to GND and all others are on V_{DD}. An

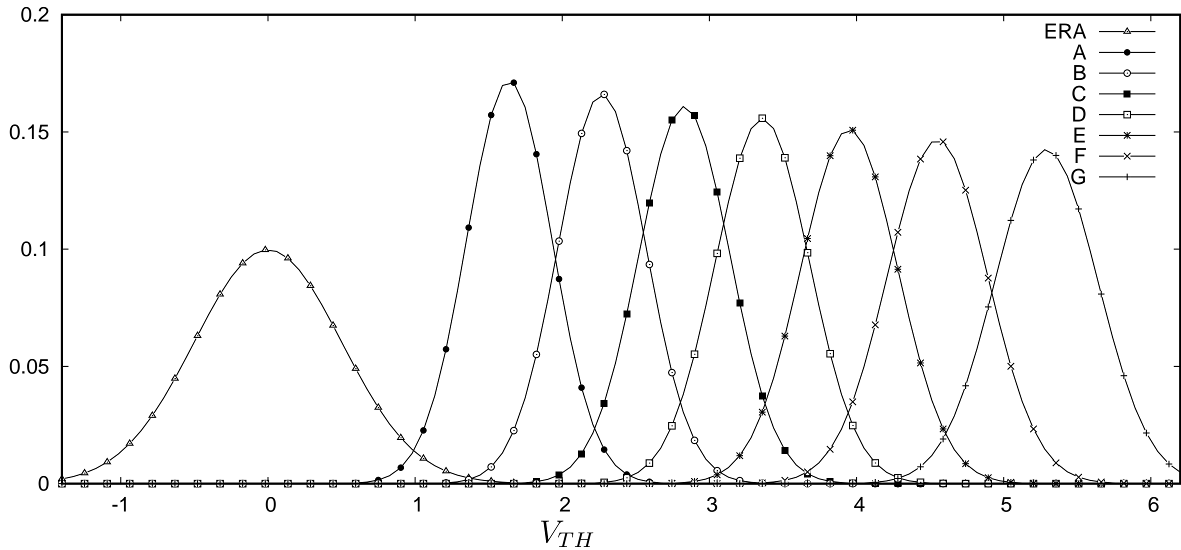


Fig. 2.4: Threshold distribuition

WL is programmed at once. Because this process does not load the FG to an accurate state, the programmed cells are verified in terms of whether the desired charge level is reached. If not, the program process of those cells is repeated. This is a so-called program and verify procedure.

The pages can only be written in sequential order without biased data to avoid inter cell interference. In MLCs and TLCs, the lower pages are programmed first, then the middle and optionally the upper pages are written.

Erase A Flash block comprises a Flash matrix with a sharing bulk substrate and thus all cells are connected with the p-well plain. In contrast to the page-wise read and program operation, the cells can only be erased within an entire Flash block. Similar to the programming, the erase operation is undertaken using the FNT. A negative voltage is not needed. Instead, the p-well is on a high erasing voltage V_{ERASE} and the gates are tied to GND. The erasing process is not a per cell-based individual erase and verify operation and thus the erased state has a higher deviation (see Figure 2.4).

2.2.2 Error Characteristics

In practice, the information stored on a FG cell is inferred by imperfections from various causes, which results in the state variance shown in Figure 2.4. Due to the imperfection, the distribution area of a written state crosses the threshold level and the distribution area of the opposite side represents the BER. Errors occurring raise from endurance, retention and read disturb are subject of the ECC.

Endurance The FNT during program and erase cycles cause a defect in the oxide, caused by electron trapping within it. This is a progressive degradation process caused by cycling the cells. This effect can not be reverted.

Retention The definition of non-volatile memory suggests that the load stored in the cells are stable over time. Nevertheless, the oxide is responsible for the FG losing

electrons. Practically information stored on other medias suffer the same problem of transience. The time period that can be supported strongly depends on the device temperature. At a normal room temperature of $300K$, the error correction capability limit is reached within years. A so-called baked device at $360K$ reaches this limit within a few days. For testing the degradation in Flash memory devices, special baking procedures are used. In order to reload the FG, a rewrite after a specified time period, which is set in the firmware, is used to refresh the information.

Read disturb The read voltage disturbs neighboring cells due to its electrical field. Unfortunately, the writing process has a higher impact because it has a higher operation voltage, although it occurs less often than cell reads. This read disturb is handled by rewriting cells after a specific number of reads.

Media defect Different causes lead to total defect of an entire die, e.g. the defect of the charge pipes that are needed for V_{PRG} and V_{ERASE}. This problem is encountered by redundant data storage in different dies or chips using a redundant array of independent disks (RAID).

Radiation Several radiations cause mutations of the cell load and thus a threshold shift [28]. This holds interest in several environments like space, where cosmic ray is much more intensive. Possible correlated errors are thinkable caused by radiation trails. Their influence in GCC is not investigated in this work and might be interesting for further research.

In various publications [38, 60, 8], the investigations of the Flash channel error and its characteristics shows in common a per-bit statistical independence and that an additive white gaussian noise (AWGN) channel model can be assumed.

2.2.3 Flash Layer Design

Figure 2.5 depicts the data flow for writing and reading from a device. First, the data is compressed and thus the entropy increases, which is also necessary for the encryption in the next step. Either before or after channel encoding, the data is scrambled to achieve a bias-free data pattern. If the decoder requires the original pattern read from the flash memory for a channel estimation, then it is essential that the scrambler is applied above the channel code. Because the erased state of the flash memory is a logical '1', an inverter is applied to decode an erased data block as a valid "zero codeword". Whereas the part above the dashed line in Figure 2.5 is usually implemented in a Flash controller, the part beyond takes place in common Flash memories. The Flash memory does the mapping between the logical page address and its electrical page and level. It contains the components to read, write and erase the FG cells.

2.2.4 Flash Channel Model

In flash memories, the FG transistors store data in the form of a charge level. In order to read out the data, the previously described V_{TH} is applied to turn on the transistor. The stored charge level changes the V_{TH} of the transistor. The probability density function of the variation of V_{TH} is usually modeled by a Gaussian distribution with a variance σ^2, mean μ, and probability density [71, 40]

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \quad (2.1)$$

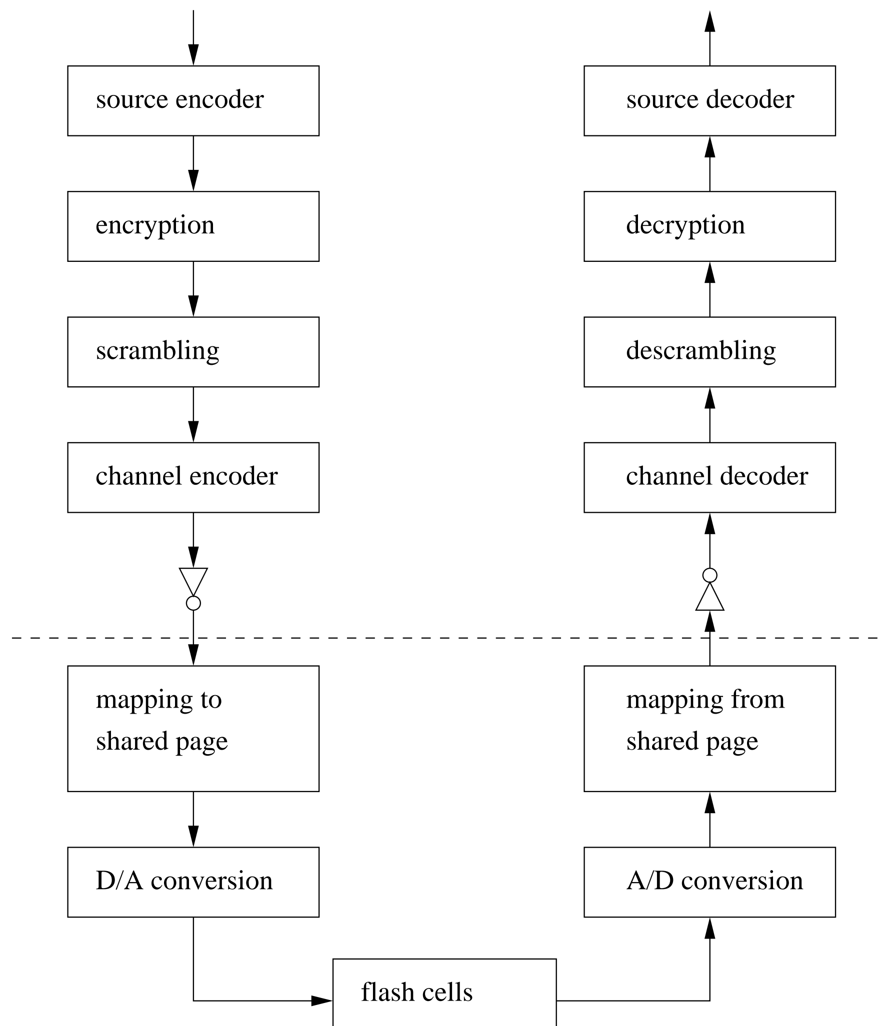


Fig. 2.5: Channel abstraction layers

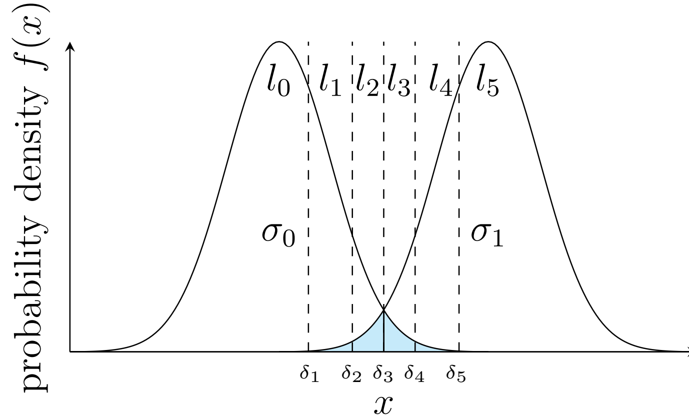


Fig. 2.6: Probability density function with reading thresholds. Reproduced by permission of the Institution of Engineering & Technology [101]

Hence, the channel model for flash cells is equivalent to an AWGN channel where the noise variance σ^2 depends on the stored bit, i.e. the mean μ of the charge level. Furthermore, a flash memory with reliability information can be considered as a channel with quantized channel values.

In order to obtain reliability information, the cell is read several times with small changes of the threshold voltages. Figure 2.6 shows the probability distribution with five threshold voltages, where the reading threshold voltages are denoted by δ_i with $\delta_1 < \delta_2 < ... < \delta_k$. We have $k + 1$ different voltage intervals l_i with $i \in \{0, ..., k\}$. Hence, in Figure 2.6 δ_3 denotes the reference voltage for the hard decision and the intervals l_2, l_3 correspond to the most unreliable input values. In this case, we can calculate the conditional probability of observing a value in the interval from level δ_i and δ_{i+1} given the charge level with mean μ as

$$w_i(\mu, \sigma) = \int_{\delta_i}^{\delta_{i+1}} f(x)dx \tag{2.2}$$

where σ is standard deviation of the distribution. Assuming that information bits '0' and '1' are stored with equal probability, we obtain the conditional error probability given that we observe a value in the interval l_i

$$p_i = \begin{cases} \frac{w_i(\mu_0,\sigma_1)}{w_i(\mu_0,\sigma_0)+w_i(\mu_1,\sigma_1)} & , i \leq \frac{k}{2} \\ \frac{w_i(\mu_1,\sigma_0)}{w_i(\mu_0,\sigma_0)+w_i(\mu_1,\sigma_1)} & , i > \frac{k}{2} \end{cases} \tag{2.3}$$

Here, μ_0 and μ_1 are the mean values for the information bit '0' and '1' that is inferred with the particular reading. Likewise, σ_0 and σ_1 denote the corresponding standard deviations. The variance depends on the charge level, i.e. some charge levels are less reliable. Moreover, the error probability is not equal for zeros and ones. Note that this model is a simplification of the Gaussian mixture model presented in [31], where we assume that the probabilities p_i only depend on the two charge levels adjacent to the considered reference threshold. The reliability information is typically represented by log-likelihood ratios, where we have the log-likelihood ratio (LLR) for the interval l_i

$$LLR_i = \log\left(\frac{1 - p_i}{p_i}\right). \tag{2.4}$$

Methods to estimate the LLR values are presented in [31, 67, 25].

2.3 Other Modern Storage Systems

In order to provide an introduction to storage devices with different carrier materials, the following subsections present an outtake of four storage devices.

Magnetic Device The hard disc drive (HDD) is a successor of the magnetic tape. Its media is constructed as a constantly-rotating disk that is coated with a thin magnetic layer on which the information is stored by the magnetic polarization. The data is arranged in concentric circles, which are called tracks. A read-/write head arm can be moved between different tracks to read or write blocks in a random order. Depending on the position of a requested block, the head has to wait until the sector is beyond. This read/write head is air buffered by the airflow induced by the disc rotation. Nowadays, with HDDs the air gap between the flying head and the rotating disc is around 10nm [53]. For further developments for higher densities like $1Tbit/inch$, a $2nm$ gap [3] is believed to be necessary. This means that an object equal to or larger than this gap might cause damages. Because the entire mechanical structure is covered in a dust-proof but not airtight housing, scratches are not expected. The mechanics are very sensible against accelerations. An additional disadvantage is the energy-consuming motor, which also rotates in idle states to keep the air gap up.

A magnetic devices suffers from several noise sources. [21] describes sources of disturbances as material imperfections that appears "through fluctuations concentrated close to the recorded transition centers" and "the random micro structural properties". Furthermore, [21] describes electronics noise that occurs in the surrounding electronic circuits, which is responsible for the analog signal acquisition and processing. Transition jitter noise also exists due to timing imperfections, inter track interference and thermal asperities. An ECC solution similar to the one described in this work is discussed in [21].

Optical Devices Optical discs comprises a similar mechanical construction compared with the previously-described HDDs, although they are usually designed as a removable single media. Similar to the HDD, the disc rotates and a moving head can access the information written on the disc. Instead of measuring magnetic polarization, the reflection of photons is determined. An optical disc comprises multiple photo layers, which are covered with a protective layer. Inside this protection comprises the first interference layer, the phase-change layer, the second interference layer and finally the reflective metal layer. The interference layers act as an optical contrast enhancement. In the phase-change layer, the information is stored in forms of pits and lands. Usually, the CD is designed to be written only once (e.g. CD-ROM) or a few times (CD-RW). The data is stored in one long snail track [50].

The optical channel of optical discs is considered as a "good-quality channel" because the system is "designed to operate slightly beyond the resolution limit". [46] further describes that optical discs can be manufactured with a BER around 10^{-6} but low-end CDs and CDROMs are in the range of a BER around 10^{-5} to 10^{-4}.

In contrast to HDDs, the CDs underlie different physical stress like scratches and dust. Most of these irritations draw an erroneous line over the stored data, which leads to an error burst characteristic. A radial-oriented scratch tends to generate single errors over several blocks, whereas concentric scratches tend to create burst errors. Depending on the orientation of the same scratch shape, there are drastic differences for the error correction demands. A classical approach to encounter this problem is interleaving.

CD audio was introduced in the early-1980' by Philips and Sony. It deploys the Cross-interleaved Reed-Solomon Code (CIRC). This specific RS code comprises 255 symbols with a size of eight bits and can correct up to four symbol errors or eight as defect declared symbols. In order to avoid larger burst errors, interleaving is used.

Solid State Device SSD are successors of the HDDs. The name solid stands for storing data on semiconductor materials instead of drives with moving parts, tubes, relays, etc. This makes it robust against mechanical stress. Because SSDs are developed as successors of HDDs their form factor and interface are similar but not necessary the same. Their random access time is less than the HDD because a readjustment of the reading head is not necessary. Overall they have a higher throughput. As SATA interfaces were common, faster PCIe are upcoming to the end of the 2010s, which is the result of the performance increase.

In contrast to other Flash memory storage systems like CF cards, SD cards and USB sticks the SSD is designed for large storage capacity. An SSD system comprises a Flash controller with a multi-core processor, serving several Flash dies in multiple Flash channels. Common systems comprises four or eight channels, each with eight, sixteen or more dies. In enterprise systems, the flash controller manages 128, 256 or 512 Flash dies [52].

Matrix Barcode Matrix barcodes are two-dimensional codes. A famous example of such a code is the QR code, which has different levels of error correction capability. They are applied to various objects made of different materials like paper, metal, synthetics, etc. Some considerations on how to apply GCC with an optimal interleaving mechanism on this code class are discussed in Chapter 5.

2.4 Summary

This chapter has provided a brief introduction to storage systems. In particular, the Flash memory was explained, which was the state-of-the-art storage media when this work was written in 2018. Its functionality was described, including the property that Flash memories are erroneous channels and their reliability suffer from production defects, endurance and retention. In general, their error characteristics are assumed as independent and equal distributed. It was also described how data is arranged within the memory and how it can be accessed.

Own measurements have shown that with some Flash memory models the errors are not equal distributed per page. Plotting the page error number in their sequence, which also represents the local relation, show bumps within a block. Thus, an interleaving covering

several pages would increase the decoding performance. Because the data is then distributed over different pages the performance will suffer.

From a coding theorist perspective, it would be highly efficient to apply multi-level coding to achieve higher error correction performance. Multi-level codes can be applied using the threshold voltage as an amplitude shift keying (ASK). At present, there are no Flash memories on the market that support such a multilevel coding. The structure of the existing shared pages is historically grown. This topic should be targeted in future research.

Finally other storage media, their components and challenges in data reliability were discussed. Similar to the Flash memory, they suffer from an erroneous channel and ECCs are applied. For this and further work they hold interest due to their two-dimensional information arrangement. In the case of correlated errors techniques from Chapter 5 could be used to increase the error correction capability.

Chapter 3

Preliminaries for Channel Coding

A channel model that represents the Flash memory can be modeled by a source emitting information, the transmission channel that adds noise and a receiving sink. In Chapter 2 the Flash memory and its error sources were discussed. In order to encounter the data corruption at the sink, channel codes that correct errors by adding redundancy are applied. Several channel code classes exist, which in turn also have a high range of decoding algorithms with different properties. The effect of channel codes is that the number of errors of a received message is reduced. This chapter provides a basic introduction to algebraic codes that can be decoded with BMD and thus an analytical determination of the residual error rate can be made. The aim of this chapter is to offer an introduction to algebraic codes. An important base is the Galois field (GF), which enables basic operations on finite symbols. Section 3.1 provides an overview of GF with some examples and implementation approaches. Section 3.2 explains the algebraic code. Based on this, a basic error model is presented in Section 3.3.

3.1 Galois Fields

GF provide the mathematical operations that are necessary for algebraic codes over a finite set of symbols.

Definition 1. *Group [6]*
A group $\mathcal{A}$ is a set of elements combined with an operand $*$, iff:

I *Closure* $\forall_{a,b \in \mathcal{A}} : a * b \in \mathcal{A}$

II *Associativity* $\forall_{a,b,c \in \mathcal{A}} : a * (b * c) = (a * b) * c$

III *Identity element* $\exists_{e \in \mathcal{A}} : \forall_{a \in \mathcal{A}} : a * e = a$

IV *inverse element* $\forall_{a \in \mathcal{A}} : \exists_{b \in \mathcal{A}} : a * b = e$

If commutativity ($\forall_{a,b \in \mathcal{A}} : a * b = b * a$) holds, it is called an Abeilan group.

Definition 2. *Ring [6]*
A set $\mathcal{A}$ combined with two operands $+$ and $\cdot$ is a ring, iff:

I *$\mathcal{A}$ is an Abelian group regarding $+$*

II *Closure with respect to $\cdot$ in $\mathcal{A}$:* $\forall_{a,b \in \mathcal{A}} : a \cdot b \in \mathcal{A}$

III *Associativity of $\cdot$:* $\forall_{a,b,c \in \mathcal{A}} : a \cdot (b \cdot c) = (a \cdot b) \cdot c$

IV *Distributivity :* $\forall_{a,b,c \in \mathcal{A}} : a \cdot (b + c) = a \cdot b + a \cdot c$

Definition 3. *Field [6]*
$\mathcal{A}$ is a field, iff:

I Abelian group with respect to addition

II $\mathcal{A}$ without the zero element is an Abelian group with respect to multiplication

III Distributivity : $\forall_{a,b,c\in\mathcal{A}} : a \cdot (b + c) = a \cdot b + a \cdot c$

Definition 4. *Prime field [6]*
The GF is a field with finite number of elements. Let $p \in N$ be prime. The set of elements $\{0, 1, \ldots, p-1\}$ with the operations $(+, \cdot)$ mod p satisfies the axioms of a field and is called a prime field and denoted with $GF(p)$.

Table 3.1 shows the truth table for the addition and multiplication operation. For $GF(2)$, an exception is that addition $\oplus$ and subtraction is equivalent to the boolean XOR function, whereas $GF(2)$ multiplication $\otimes$ is equivalent to the boolean AND operation. An example of non-binary symbols is shown in Figure 3.2.

$\oplus$	**0**	**1**
0	0	1
1	1	0

(a)

$\otimes$	**0**	**1**
0	0	0
1	0	1

(b)

Tab. 3.1: GF $p = 2$

+	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

(a)

$\cdot$	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

(b)

Tab. 3.2: GF $p = 5$

Extension fields $GF(p^m)$ over the base field $GF(p)$ can be constructed with a degree m. In digital applications like channel coding, the use of binary extension fields $GF(2^m)$ is applied because binary symbols can be mapped without gaps. The electrical logic elements are binary.

Definition 5. *Irreducible Polynomial [6]*
A polynomial $p(x)$ over $GF(p)$ is irreducible over $GF(p)$ if $p(x)$ is not divisible by any other polynomial over $GF(p)$ of degree less than the degree of $p(x)$ but greater than zero.

Definition 6. *Primitive Polynomial*
An irreducible polynomial p(x) is primitive if all powers of α generate all elements of an extension field.

Definition 7. *Extension field $GF(p^m)$ [6]*
Let $p(x)$ be irreducible over $GF(p)$ and $\alpha \notin GF(p)$ a root of $p(x)$ with the degree $p(x) = m$, then $GF(p^m)$ is the smallest field that contains $GF(p)$ and α.

In the context of coding theory, sometimes a polynomial is expressed as a vector. Therefore, the bijective relation is used as

$$\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}) \longleftrightarrow a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}. \tag{3.1}$$

Definition 8. *discrete Fourier transformation (DFT)*
The DFT for a polynomial $A(x)$ with coefficients α from GF is:

$$a_i = A(\alpha^i), i = 0, \ldots, n-1 \tag{3.2}$$

and its inverse discrete Fourier transformation (IDFT) is:

$$A_j = n^{-1} \cdot a(\alpha^{-j}), j = 0, \ldots, n-1 \tag{3.3}$$

The elements of a GF can be expressed in exponential form and in its components. In Table 3.3, an example of $GF(2^3)$ is given with the primitive polynomial $p(x) = x^3 + x + 1$. In hardware, the components are represented as bits that are either set or cleared. The

exponent	components			binary
$-\infty$			0	000
0			1	001
1		α		010
2	α^2			100
3		α	1	011
4	α^2	α		110
5	α^2	α	1	111
6	α^2		1	101

Tab. 3.3: Example of $GF(2^3)$ with primitive polynomial $x^3 + x + 1$

implementation of the $GF(2^m)$ addition $(a(x) + b(x))\ mod\ 2 = c(x)$ operator is achieved by bitwise XOR.

Example 1. *A bit sequence* $\mathbf{a} = 010$ *is interpreted as the polynomial* $a(x) = \alpha = x$ *and* $\mathbf{b} = 110$ *as* $b(x) = \alpha^4 = x^2 + x$*, then*

$$\begin{aligned} c(x) &= (a(x) + b(x)\ mod\ 2) && (3.4a)\\ &= (x + x^2 + x)\ mod\ 2 && (3.4b)\\ &= (x^2 + 2x)\ mod\ 2 && (3.4c)\\ &= x^2\,. && (3.4d) \end{aligned}$$

Thus $c(x) = x^2 = \alpha^2$ *is the binary sequence* $\mathbf{c} = (100)$.

In Chapter 7, the implementation of an algebraic code and its GF operations are discussed. A software multiplication for binary extension fields $GF(2^m)$ is implemented using look-up tables (LUTs) that first translate the bit component representation into the exponential form. Subsequently, the exponents are added and rebuilt to a component representation using the inverse LUT. A hardware multiplication can be realized by a slow but less complex linear feedback shift register (LFSR) or a fast full-parallel multiplier that needs more logic [42].

3.2 Channel Coding Basics

Channel codes expand information with additional redundancy and are used to protect the information against errors such that they can be decoded by FEC. There are two main different approaches. The first domain is formed by the convolutional codes where the message bits depend on μ previous data bits. They have their strength in correcting correlated errors. The other code domain is block codes, which convert information of k-bits into a block codeword of n bits. It has its strength in decoding statistically-independent errors. In this work, ECC is used to correct errors that occur in flash memories that are considered as independent errors. Thus, for this work algebraic block codes like BCH and RS codes are relevant.

3.2.1 Linear Block Codes

A linear block code is used to correct errors by adding redundancy to the original information. The block code has an overall length of n symbols comprising k information symbols. The number of redundancy symbols is the difference $r = n - k$. The code rate is $R = \frac{k}{n}$.

Definition 9. *Hamming weight [6] The Hamming weight of a symbol c_j is:*

$$wt(c_j) = \begin{cases} 0, c_j = 0 \\ 1, c_j \neq 0 \end{cases} \tag{3.5}$$

and the weight of a codeword $\mathbf{c}$ *is*

$$wt(\mathbf{c}) = \sum_{j=0}^{n-1} wt(c_j) \tag{3.6}$$

Definition 10. *Hamming distance [6] The Hamming distance of two codewords* $\mathbf{a}$ *and* $\mathbf{c}$ *is:*

$$dist(\mathbf{a}, \mathbf{c}) = wt(\mathbf{a} - \mathbf{c}) \tag{3.7}$$

Definition 11. *Minimal Hamming distance [6] The minimal distance of a code is the minimal distance between two different codewords:*

$$d = \min_{\substack{a,c \in \mathcal{C} \\ a \neq c}} \{dist(a, c)\} \tag{3.8}$$

Definition 12. *Linear block code [45]*
A code $\mathcal{C} \subset GF(p^m)^n$ is linear if for any two codewords $\mathbf{c_1}, \mathbf{c_2} \in \mathcal{C}$ the following is satisfied:

$$a\mathbf{c_1} + b\mathbf{c_2} \in \mathcal{C}, \qquad a, b \in GF(p^m) \tag{3.9}$$

3.2.2 Reed-Solomon Codes

RS codes are popular ECCs and they are applied to many telecommunication applications. These codes form the basis of the introduction in algebraic codes in this thesis. In the next step, the BCH codes are derived based on RS codes. In this work, we define the RS codes using the DFT.

Definition 13. *Reed-Solomon codes [6]*
Let $\alpha \in GF(p^m)$ be an element of order n, then the RS code $\mathcal{C}$ is defined by the set of polynomials $A(x)$ with a degree less than $k : A(x) = A_0 + A_1x + A_2x^2 + \ldots + A_{k-1}x^{k-1}, A_i \in GF(p^m), k \leq n$.

$$\mathcal{C} := \{\mathbf{a}|a_i = A(\alpha_i), i = 0, 1, \ldots, n-1, degree\ A(x) < k\} \tag{3.10}$$

In definition 13, the RS code is defined using the DFT as $\mathcal{F}(a(x)) = A(x)$. This shows a relation between the codeword $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ and the information in $\mathbf{A} = (A_0, A_1, A_2, \ldots, A_{k-1}, 0, \ldots, 0)$.

Alternatively, an information polynomial $i(x)$ is encoded using the generator polynomial $g(x)$ such that $a(x) = i(x)g(x)$. The RS generator polynomial follows from definition 13 as:

$$g(x) = \prod_{j=k}^{n-1}(x - \alpha^{-j}) \tag{3.11}$$

3.2.3 Bose-Chaudhuri-Hocquenghem Codes

We consider the binary BCH codes. It is convenient to represent the BCH codewords $\mathbf{b}$ as a codeword polynomial $b(x)$. A BCH code can be either defined by the cyclotomic coset or similar to the RS code using the DFT.

Definition 14. *BCH code using DFT [6]*
The primitive BCH code $\mathcal{C}$ of length $n = 2^m - 1$ and the designed minimum distance d, is:

$$\mathcal{C} := \{a(x)|a(x) = \mathcal{F}^{-1}(A(x)), A_{n-1} = \ldots = A_{n-d+1} = 0, \forall_i : A_i^2 = A_{2i}\} \tag{3.12}$$

The symmetry relation $A_i^2 = A_{2i}$ guarantees that the coefficients $a(x)$ are in the base field of $GF(2)$ as $a_i \in GF(2)$. Now we define the BCH code defined using the cyclotomic coset $\mathcal{K}_i$.

Definition 15. *Cyclotomic Coset*
Let $n = q^m - 1$. The cyclotomic coset $\mathcal{K}_i$ is:

$$\mathcal{K}_i := \{i \cdot q^j\ mod\ n, j = 0, 1, \ldots, m-1\} \tag{3.13}$$

Definition 16. *BCH code using cyclotomic coset [6] Let $\mathcal{K}_i$ be the cyclotomic coset of $n = 2^m - 1$, α the primitive element of $GF(2^m)$ and $\mathcal{M}$ the union of arbitrary cyclotomic cosets $\mathcal{M} = \mathcal{K}_{i_1} \cup \mathcal{K}_{i_2} \cup \ldots$. A primitive BCH code has the length $n = 2^m - 1$ and the generator polynomial:*

$$g(x) = \Pi_{i \in \mathcal{M}}(x - \alpha^{-i}), \mathcal{M} = \bigcup_{i=0}^{t-1} \mathcal{K}_i\,. \tag{3.14}$$

The parameter $t = \lfloor \frac{d-1}{2} \rfloor$ in 3.14 is the error-correcting capability and in this case the number of correctable bits of the codeword, α is a primitive element of the Galois field $GF(2^m)$, $\mathcal{K}_i$ is the cyclotomic coset, and the index i denotes the coset numbering [54]. Like the BCH definition in 14, definition 16 generates the same code with coefficients in $GF(2)$ using the conjugate complex elements that follows from the cyclotomic coset in 15. Similar to an RS code, a BCH codeword $b(x) = i(x)g(x)$ can be generated by multiplication of the information polynomial $i(x)$ with the coefficients in $GF(2)$ and the generator polynomial $g(x)$.

The generator polynomial $g(x)$ embeds zero positions into the DFT domain of a codeword. An RS and a BCH codeword $u(x)$ of length n can be either encoded systematically (see

Equation 3.15)), where the information $i(x)$ with the length k itself is part of the codeword, or non-systematically, where the original information has to be decoded to retrieve the original information.

$$u(x) = i(x) \cdot x^{(n-k)} + i(x) \cdot x^{(n-k)} \mod g(x) \tag{3.15}$$

3.2.4 Algebraic Decoding

Consider the received vector $r(x) = u(x) + e(x)$ with the error vector $r(x)$. A vector $r(x)$ can be algebraically decoded using its syndromes. These syndromes $S(x)$ are calculated by Equation 3.16 and they depend only on the errors $e(x)$ that occurred in a received vector.

$$S_i = r(\alpha^{i+1}), i = 0, \ldots, 2t-1 \tag{3.16}$$

From $A_i^2 = A_{2i}$ in definition 14 follows that in contrast to the syndrome values of an RS codeword, the even values of the BCH syndromes are linear depended on the odd values.

The received codeword $r(x)$ is assumed to be a valid codeword if $S(x) = 0$. If $S(x) \neq 0$ the so-called key equation has to be solved, e.g. using the Berlekamp-Massey algorithm (BMA) [48, 9, 44, 54]. This results in the error-location polynomial $\sigma(x)$. The roots of this polynomial indicate the error positions if the actual number of errors is less or equal t. The next decoding step, the Chien search, evaluates $\sigma(x)$ to determine the roots $\sigma(\alpha^{-l}) = 0$ where l corresponds to an error position [12, 4].

For the RS decoding, the syndrome $S(x)$ and the error locations can be determined as for BCH codes. However, given that this code is not binary it is not sufficient to calculate the error positions. Additionally, the error value δ_l has to be calculated with the so-called Forney algorithm, i.e. we first calculate the error-value polynomial $\Omega(x)$ that solves the so-called Berlekamp-Forney key equation

$$\sigma(x)S(x) = -\Omega(x) \bmod x^{2t}\,. \tag{3.17}$$

Next, the error value δ_l is determined

$$\delta_l = \left.\frac{\Omega(x)}{\sigma'(x)}\right|_{x=\alpha^{-l}}\,. \tag{3.18}$$

The derivative $\sigma'(x)$ can easily be calculated as a by-product from the Chien search and does not require additional logic. The error-value polynomial $\Omega(x)$ can be derived with an expanded BMA [35].

3.3 Basic Error Models

In order to design an ECC and its parameters, the channel should be known and modeled for either a simulative or an analytical analysis. In this section, the binary symmetric channel (BSC) followed from the underlying analogs and quantized channel, which is given from the flash model, is described. An extension from memoryless to correlated errors is made in Chapter 5.

Figure 3.1a shows the BSC. It has two main properties, namely the symmetry and the fact that errors are independent. It distinguishes between two boolean states false '0' and true '1' with each two transitions, a correct and an erroneous one. The erroneous transition has the probability ε and is also known as the raw bit error rate (RBER). This BSC model is extended

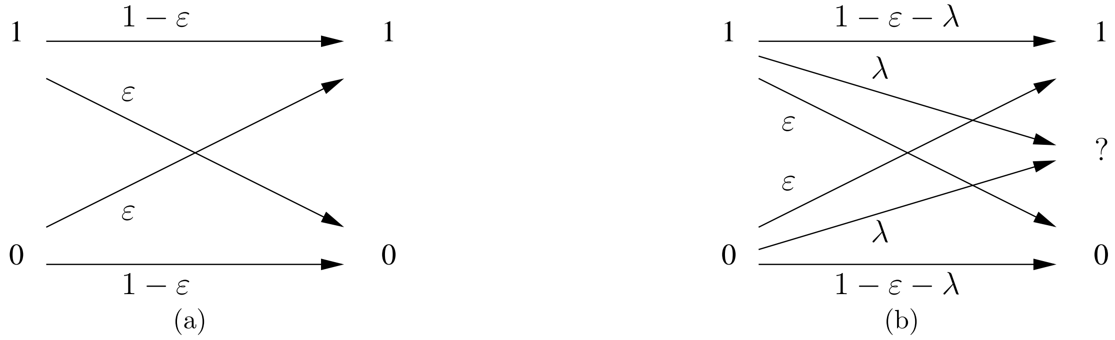


Fig. 3.1: Binary symmetric channel (a) and its erasure extension (b)

in Figure 3.1b by an additional erasure in the binary symmetric error and erasure channel (BSEEC). With the probability λ an erasure '?' occurs if a bit is detected as undetermined. This additional information can be used to improve the decoding capability.

Based on the BSC and a code with a designed distance d, the frame error rate (FER) is determined by the binomial distribution.

$$FER_{error\ only} = 1 - \sum_{j=0}^{t} \binom{n}{j} \varepsilon^j (1-\varepsilon)^{n-j} \tag{3.19}$$

An algebraic code that supports erasure decoding can decode a number e of erasures, where $2t + e = d - 1$. The binomial distribution 3.19 is extended to a multinomial distribution that covers all possible error and erasure combinations.

$$FER \leq 1 - \sum_{j=0}^{t} \sum_{k=0}^{d-2j-1} \binom{n}{j} \binom{n-j}{k} \varepsilon^j \lambda^k (1-\varepsilon-\lambda)^{n-j-k} \tag{3.20}$$

Considering flash memories the binary model underlies a channel that can be modeled by the AWGN [31, 8],

$$f(y|x=j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(y-m_j)^2}{2\sigma_j^2}} \tag{3.21}$$

where j is the original symbol, which is $+1$ for the false and -1 for the true state. In practice, this channel is quantized with one, three or six thresholds.

3.4 Summary

In this chapter, we have provided an introduction to algebraic coding. First the GF arithmetic for algebraic codes was described, before the channel codes were introduced in Section 3.2. The definition of RS and BCH codes were described, as well as the basic idea concerning how they can be decoded algebraically. Later in this work, in Chapter 7 and 8, this is used for the discussion of the hardware implementation. Finally, the basic error models were described, which are later used in Sections 4.2 and 4.3 to design GCC parameters to analyze the possibilities of this construction and whether they can meet the requirements to support particular Flash memories.

Chapter 4

Construction of high-rate Generalized Concatenated Codes for Flash Memories

Code concatenation enables possibilities to combine codes with different characteristics. Forney describes his view on concatenation first in [22]. This very intuitive view extends the channel model shown in 4.1 with an additional outer encoder and decoder block as can be seen in Figure 4.2. The outer code considers the inner coder as a super channel. The same setup but with a different view is described by Blokh-Zyablov. They consider both inner and outer codes as a single code. This type of code concatenation is called generalized concatenated code (GCC).

A famous example for code concatenation is the DVB-T Standard. In its first version an inner convolution code is used which is concatenated with an outer RS code. The successor version, DVB-T2 standard, uses an inner LDPC and outer BCH code. Each of these inner codes supports soft decision decoding like the Viterbi algorithm for the convolutional decoding and the belief propagation algorithm for the LDPC code. Soft decision decoding reaches a higher gain than hard decision if the channel supports more than binary quantized values. In contrast to the convolutional code which produces in most error cases burst errors the LDPC code returns independent bit errors. The Reed Solomon code has good burst error correction capabilities and thus it is used in combination with convolutional codes. The BCH code has its advantages in independent error correction and is used in addition to the LDPC code as an outer code. RS and BCH codes have good error correction performance in hard decision decoding algorithms. Another famous combination of inner convolution and outer RS code is also used in the voyager program which started in 1977 [15] called Reed-Solomon/Viterbi (RSV).

One aspect of using concatenated codes in flash controllers is the fact that the component codes have small decoding complexity. The component codeword size is smaller and thus a smaller GF arithmetic is necessary. Furthermore, the component codes need less error

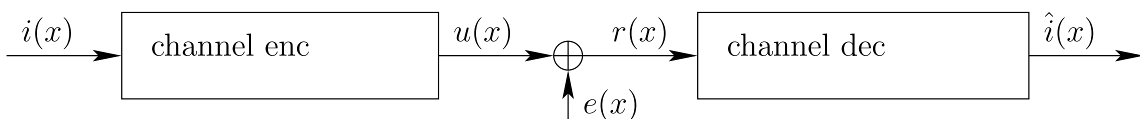


Fig. 4.1: Transmission Channel Model

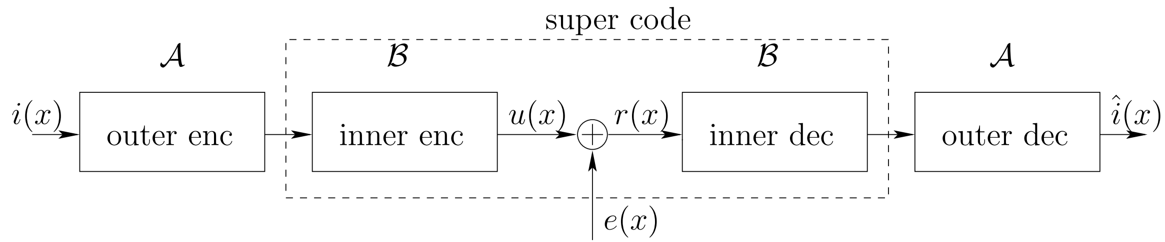


Fig. 4.2: Code concatenation by Forney

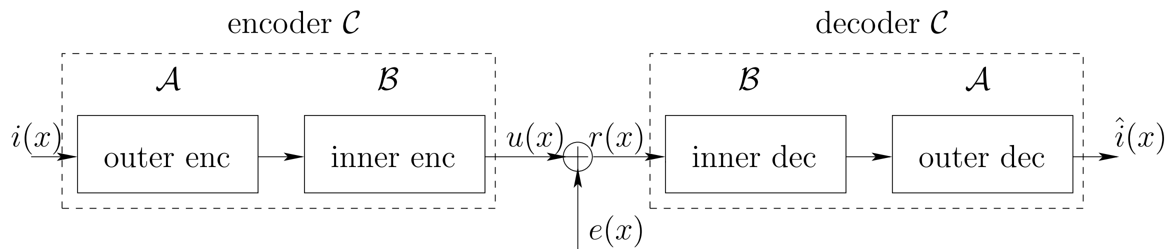


Fig. 4.3: Code concatenation by Blokh-Zyablov

correction capability to reach the same overall block error probability as a long code, thus the decoders have a smaller complexity.

For industrial applications a provable error correction capability is recommended, which BMD codes provide. Therefore GCC with inner BCH and outer RS codes are investigated in the following sections. First the codeword construction will be discussed. In order to prepare the basics for the hardware implementation, which is presented in Chapter 8 and 9, the encoding and decoding process is presented in Section 4.1.1 and 4.1.2. In this chapter we extend the multi-level code construction by erasure decoding, that is possible by using the SPC. From that point we discuss the error bounds of this construction. In Section 4.3 we present a technique how parameters can be determined properly depending on the channel and application requirements.

The novelties in this work are:

- The use of erasure decoding that is possible by applying SPC for the inner component code and thus reaching high-rate codes.
- The investigation and description of the error bounds of GCCs including erasures.
- A technique to determine proper GCC parameters to reach a designed FER.

Parts of this chapter are published in [98, 94].

4.1 GCC Preliminaries

The GCC is a multi-level code with two component codes for each dimension. Figure 4.4 shows the GCC codeword matrix. In vertical direction each column is protected using a nested BCH code. Depending on the outer GF size 2^{m_a}, where m_a is the degree of the primitive polynomial used, each set of rows is protected with the outer RS, i.e. m data bits in the same column

$\mathbf{b}_0 \in \mathcal{B}$ n_a $\mathbf{b}_{n_b-1} \in \mathcal{B}$

$\mathbf{a}_0 \in \mathcal{A}^{(0)}$

n_b

$\mathbf{a}_{L-1} \in \mathcal{A}^{(i)}$

Fig. 4.4: GCC codeword matrix. Reproduced by permission of the Institution of Engineering & Technology [92]

form a symbol for the RS code. An information vector has first to be reshaped into this matrix form while keeping the redundancy bit zero before it can be encoded.

In [6] an exemplar GCC is described that uses a parity-check code $\mathcal{A}$ and a repetition code $\mathcal{B}$. For this work we use RS and BCH codes. We start to describe the encoding scheme in Section 4.1.1 to gain an understanding of the GCC structure. In Section 4.1.2 the decoding steps are discussed. A detailed discussion can be found in [19].

4.1.1 Encoding

Figure 4.5 shows the encoding process. First an empty GCC codeword matrix has to be formed where the information is stored at distinct positions. The redundancy positions (gray area) have to be kept blank. The encoding process begins by encoding the rows with the outer RS code. Once the entire matrix is encoded with the outer code, each column is processed with inner nested BCH codes. Each column is the sum of L codewords of nested linear BCH codes. This GCC encoding is not systematic. In order to read the information the codeword has to be re-imaged.

$$\mathcal{B}^{(L-1)} \subset \mathcal{B}^{(L-2)} \subset \ldots \subset \mathcal{B}^{(0)} \tag{4.1}$$

A higher level code is a sub-code of its predecessor, where the higher levels have a higher error-correcting capability, i.e. $t_{b,L-1} \geq t_{b,L-2} \geq \ldots \geq, t_{b,0}$, where $t_{b,i}$ is the error-correcting capability of the BCH code used in level i. The codeword of the j-th column is the sum of L codewords. These codewords $\mathbf{b}_j^{(i)}$ are formed by encoding the symbols $a_{j,i}$ for $0 \leq i < L$ with the corresponding sub-code $\mathcal{B}^{(i)}$. For this encoding $(L-i-1)m$ zero bits are prefixed onto the symbol $a_{j,i}$. Figure 4.6 illustrates the resulting column codewords for three levels, where $\mathbf{p}_{j,i}$ denotes the parity of the codeword $\mathbf{b}_j^{(i)}$. The final column codeword is

$$\mathbf{b}_j = \sum_{i=0}^{L-1} \mathbf{b}_j^{(i)}. \tag{4.2}$$

Due to the linearity of the nested codes, $\mathbf{b}_j$ is a codeword of $\mathcal{B}^{(0)}$.

In the following we use the well known notation $\mathcal{C}(2^m, n, k, d)$ to denote a code $\mathcal{C}$ of length n and dimension k over the Galois field $GF(2^m)$. The last parameter d denotes the minimum Hamming distance of the code $\mathcal{C}$. The overall generalized concatenated code $\mathcal{C}$ has length $n = n_a n_b$. The dimension is $k = m \sum_{i=0}^{L-1} k_{a,i}$ and the overall minimum distance is $d = \min_i\{d_{a,i} d_{b,i}\}$, where $d_{a,i}$ and $d_{b,i}$ denote the minimum Hamming distance of the outer and inner code of the i-th level.

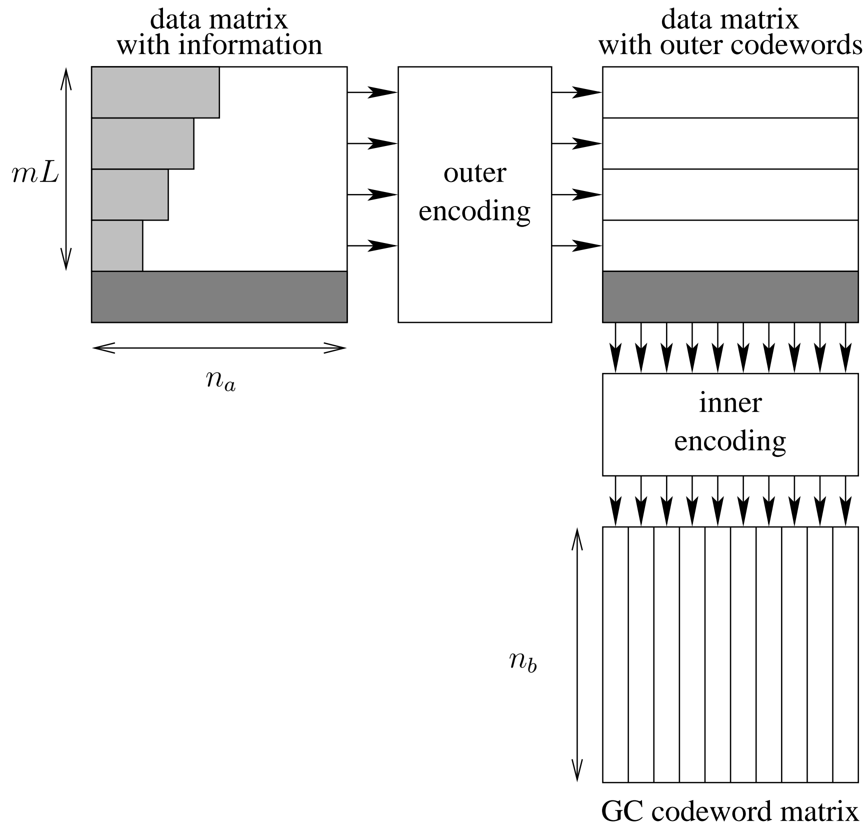
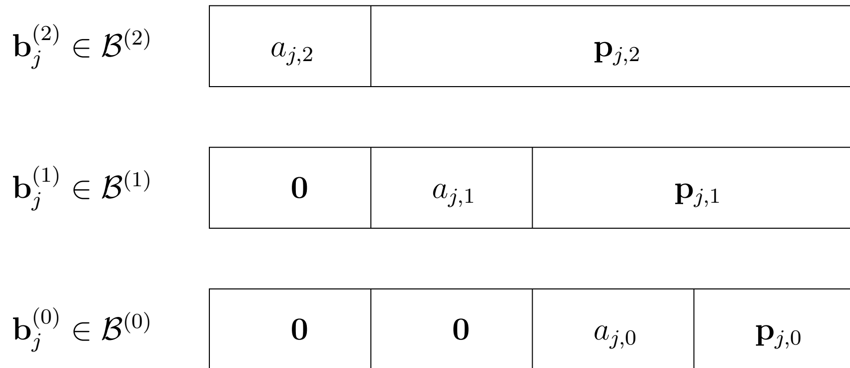


Fig. 4.5: GCC encoding



$$\mathbf{b}_j^{(0)} + \mathbf{b}_j^{(1)} + \mathbf{b}_j^{(2)} = \mathbf{b}_j \in \mathcal{B}^{(0)}$$

Fig. 4.6: Nested BCH codeword. Reproduced by permission of the Institution of Engineering & Technology [92]

level j	$k_{b,j}$	$d_{b,j}$	$k_{a,j}$	$d_{a,j}$
0	117	2	84	69
1	108	4	130	23
2	99	6	136	17
3	90	8	142	11
4-12	81	12	148	5

Tab. 4.1: Parameters of the code from example 3. $k_{b,j}$ and $d_{b,j}$ are the dimension and minimum Hamming distance of the binary inner code of level j. $k_{a,j}$ and $d_{a,j}$ are the dimension and minimum Hamming distance of the outer RS codes.

Example 2. *We use two outer RS codes over $GF(2^{10})$: $\mathcal{A}_0(2^{10}, 508, 353, 156)$ and $\mathcal{A}_1(2^{10}, 508, 477, 32)$ to construct a two level GCC. The inner codes are binary BCH codes: $\mathcal{B}_0(2, 20, 20, 1)$ and $\mathcal{B}_1(2, 20, 10, 5)$. The overall code has length $n = 20{\cdot}508 = 10160$, dimension $k = 10(477 + 353) = 8300$, and minimum distance $d = 156$.*

Example 3. *We consider a GCC suitable for error correction in flash memories. The GCC is designed for 2kbyte information blocks, i.e. a code which can be used to store 2kbyte of data plus 4 bytes of meta information. For this GCC we use thirteen levels with inner nested BCH codes over $GF(2^7)$ and outer RS codes over $GF(2^9)$. In the first level, we apply SPC codes. Hence, all inner codes are extended BCH codes of length $n_b = 13 \cdot 9 + 1 = 118$.*

The outer RS codes are constructed over the Galois field $GF(2^9)$. Hence, the dimension of the inner codes is reduced by $m = 9$ bits with each level. The GCC is constructed from $L = 13$ outer RS codes of length $n_a = 152$. The parameters of the codes are summarized in Table 4.1, where we use the same RS code in the level 4 to 12. The code has overall dimension $k = m \sum_{j=0}^{L-1} k_{a,j} = 16416$ and length $n = n_a \cdot n_b = 17936$. The code has a code rate $R = 0.915$. The design of this code will be discussed later on.

4.1.2 Decoding

Let $\mathbf{r}$ be the faulty received binary $n_b \times n_a$ data matrix. The decoding procedure operates level by level starting in the lowest level $i = 0$, where we first process the inner code $\mathcal{B}$ and then the outer code $\mathcal{A}$. Hence, in the first step we decode the column $\mathbf{r}_j$ with respect to the first inner code $\mathcal{B}^{(0)}$ which results in an estimate for the codeword $\mathbf{b}_j$. For the outer decoding, we have to infer the code symbols $a_{j,0}$ from the vectors $\mathbf{b}_j$

Figure 4.7 depicts the GCC decoding flow graph. First the GCC matrix $\mathbf{r}$ is decoded by the BCH or SPC decoder $\mathcal{B}^{(0)}$. Because the inner code is a nested BCH code it must be re-imaged to retrieve the original codeword of the outer code $\mathcal{A}^{(0)}$. This is achieved by re-encoding the received information $a_{j,i}$ and subtracting the entire codeword from the column with $L > i \geq 0$. After decoding $\mathcal{A}^{(0)}$ the partial codeword $\hat{a}_0$ is ready. In order to enter the next decoding level this partial result and its corresponding inner codeword $\hat{b}_{j,0}$ has to subtracted using re-encoding it with $\mathcal{B}^{(0)}$ from the received GCC matrix $\mathbf{r}$. Then, the next level is ready for decoding using the same work flow but increasing the index i of $\mathcal{A}^{(i)}$ and $\mathcal{B}^{(i)}$. This is repeated for each level of the codeword $\mathcal{C}$. Once all level codes are subtracted from $\mathbf{r}$ and the correction was successful, the remaining GCC data matrix only contains the errors that occurred in the channel. When all $\hat{a}_i$ are available they are joined together to obtain the complete codeword.

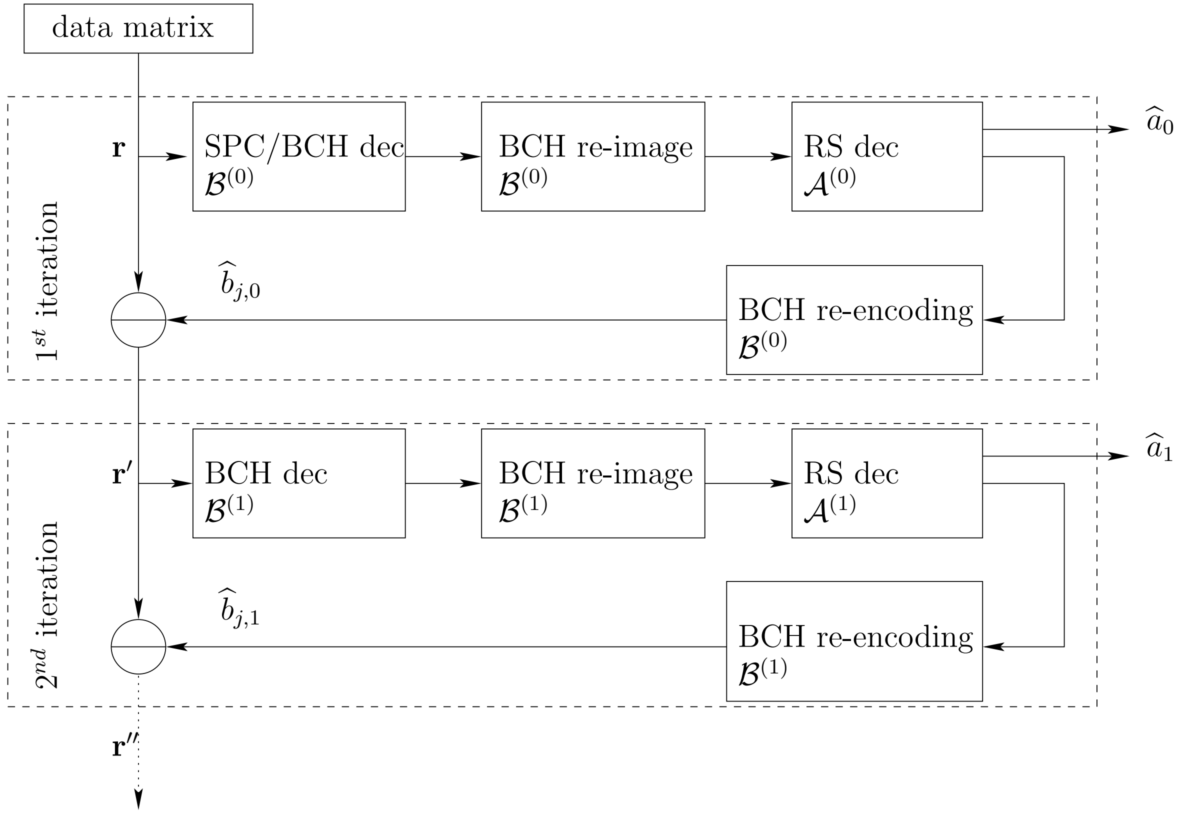


Fig. 4.7: GCC decoding scheme

4.2 Error Bounds

In industrial applications a deterministic system error probability is required. To design a code for file systems using flash memories a maximum uncorrectable block error rate FER must be guaranteed. Using algebraic codes and BMD the error bounds can be determined using the binomial distribution. Given a specific codeword size n, the target FER, the channel bit error probability ε, which represents the erroneous flash memory and the code parameters t, the code rate R can be determined. Equation 3.19 delivers the block error rate for a single codeword. In this section, we extend this equation to determine the FER of the GCC.

The generalized concatenated code comprises several inner and outer codes. Each level is indicated with the index i. First the column error rate $P_{b^{(i)}}$ has to be determined with Equation 4.3. The column code error correction parameters $t_{b,i}$ are given from the nested BCH code. The column size n_a corresponds to the GCC matrix height. An uncorrectable column means a symbol error for the outer row code.

$$P_{b^{(i)}} \leq 1 - \sum_{j=0}^{t_{b,i}} \binom{n_a}{j} \varepsilon^j (1-\varepsilon)^{n_a - j} \tag{4.3}$$

In a second step the row error rate $P_{row^{(i)}}$ is determined using the Equation 4.4. The row code size is defined by n_b and the error correction capability of each outer code is represented by $t_{a,i}$.

$$P_{a^{(i)}} \leq 1 - \sum_{j=0}^{t_{a,i}} \binom{n_b}{j} P_{b^{(i)}}^j (1 - P_{b^{(i)}})^{n_b - j} \tag{4.4}$$

In order to reach high-rate codes, erasure decoding in the RS decoder is used. Thus a parity-check code in the lowest level leads to an improvement. In this case, 4.4 has to be extended to calculate the erasure probability λ_b and the error probability P_b of the inner code $\mathcal{B}$ separately. In the case that an SPC is used in level 0 an error occurs if the number of errors is even, whereas an erasure is declared if the number of errors is odd. Hence, we can calculate the error probability of the inner codes of the first level as

$$P_{b^{(0)}} \leq \sum_{j=2,4,6,\dots}^{n_b} \binom{n_b}{j} \varepsilon^j (1-\varepsilon)^{(n_b-j)} \tag{4.5}$$

and

$$\lambda_{b^{(0)}} \leq \sum_{j=1,3,5,\dots}^{n_b} \binom{n_b}{j} \varepsilon^j (1-\varepsilon)^{(n_b-j)} . \tag{4.6}$$

In general, a decoding error with inner decoding occurs only if the number of errors in the j-th column is greater or equal to $t_{b,l}+2$, whereas a decoding failure occurs for $t_{b,l}+1$ errors. Hence, we can bound the error and erasure probabilities for the inner code of the other levels by

$$P_{b^{(l)}} \leq \sum_{j=t+2}^{n_b} \binom{n_b}{j} \varepsilon^j (1-\varepsilon)^{(n_b-j)} \tag{4.7}$$

and

$$\lambda_{b^{(l)}} \leq \sum_{j=t+1}^{n_b} \binom{n_b}{j} \varepsilon^j (1-\varepsilon)^{(n_b-j)} . \tag{4.8}$$

Thus we obtain an outer code error probability $P_{a,l}$ per level l of

$$P_{a^{(l)}} \leq 1 - \sum_{j=0}^{t_{a,l}} \sum_{k}^{d_{a,l}-2j-1} \binom{n_a}{j} \binom{n_a-j}{k_a} P_{b,l}^j \lambda_{b,l}^k (1-P_{b,l}-\lambda_{b,l})^{(n_a-v-k_a)} . \tag{4.9}$$

Finally the sum of all row error probabilities $P_{row^{(i)}}$ is the overall block error FER rate of the GCC codeword.

$$FER \leq \sum_{i=0}^{L-1} P_{a^{(i)}} \tag{4.10}$$

As already mentioned in practice ε, the codeword size and FER are given and the appropriate parameters n_a, n_b, $t_{a,i}$ and $t_{b,i}$ have to be found. The optimum parameters cannot be determined analytically. This means that a search algorithm is needed to solve this problem, which is presented in the next Section 4.3. First a target block size (e.g. 1k, 2k, 4k ...) is chosen. Based on this decision a combination consisting of number of levels and number of columns must be selected. Depending on the number of columns the proper $GF(2^{m_a})$ size for the outer code has to be selected such that $n_b \leq 2^{m_b}-1$ is true. Based on the number of levels and the symbol size m_b of the outer code, the column code size is $mi = n_a$. Starting with an $R=1$ inner code each $t_{b(i)}$ is defined by the maximum ability of the nested BCH code. The outer code error correction capability $t_{b,i}$ has to be found such that each level reaches a $P_{row^{(i)}} \leq FER$ and the condition in Equation 4.10 is satisfied.

Iteration above ε shows the code rate characteristics of the GCC with the chosen parameters Levels, outer GF size and the error correction capability of the first inner code $t_{a,0}$. Adjustments can be made by changing the amount of L_{GCC} and $t_{a,0}$.

The concatenation comprises several codes and thus it is possible to construct a code with a so-called unequal error protection like [34, 58] by adjusting the error correction capability of individual levels. Because the outer code has quantization steps in the error correction capability t it is likely to have not an optimal equal error protection.

4.3 Code Parameters

First of all, we want to define the requirements, which are given by the channel and the use case of non-volatile Flash memory for industrial environments. These memories have a specific page-size that determines together with the physical block size k the demands of a file system, the maximum size of the codeword n and its available space for redundancy r that defines a lower bound of the code rate R. On the other hand, the property of the floating gates defines the channel error probability ε that varies over the lifetime. Additionally, the application sets the requirements for a maximum FER which we assume in this work to be $FER \leq 10^{-16}$ at the specified end of lifetime (EOL) of a Flash memory. The task in this section is to find the code parameters for a given codeword size and channel signal to noise ratio (SNR) that satisfy the designed FER and the code rate R boundaries.

The GCC comprises the two-dimensional codeword matrix that have the size n_a and n_b. Its product forms the GCC codeword size n. A shorter column size n_b leads to more columns n_a. As the column sizes shrinks, the number of errors in this column also declines. In this case, the outer codeword size increases and thus the row code needs a higher codeword distance to meet the FER condition that leads to a lower R.

An additional aspect of how to determine the column size is the error correction capability of the inner levels. This depends on the constructibility of inner algebraic codes because the column code is limited to $n_b \leq 2^{m_b} - 1$. For the different levels of l, we have $l \cdot m_a + r_0$ redundancy bits available. The cyclotomic cosets determine the error correction capability that is achievable in the corresponding level l. Furthermore, the number of columns n_a determines the GF size m_a of $\mathcal{A}$. A larger n_a leads to a larger m_a and thus the distance gap between the inner levels of $\mathcal{B}$ will increase.

We propose a search algorithm in 1. Given the parameters k, FER, ε, n_b and d_b we can determine R, n_a and d_a.

In case of soft-decoding, the block error $P_{b,l}$ respectively erasure rate $\lambda_{b,l}$ of those levels l that are soft decoded have to be determined simulative and replaces the results of the binomial distribution. Example measurements of different soft-decoding performances of inner codes will be described in Chapter 6.

Because Algorithm 1 optimizes regarding R with a fixed codeword size, a plot over E_s/N_0 and R can be drawn as in Figure 4.8. This enables the comparison of the code rate efficiency for several codes. In this case, the comparison of soft and hard decoding. In the region $0.91 \leq R \leq 0.95$ the soft-decoding presented later leads to a gain of about $1dB$.

Figure 4.9 shows a plot of the FER over E_s/N_0 for each level code $\mathcal{A}_l$. In this example the GCC code has twelve levels. In order to increase the decoding capability by using soft decoding, the first three hard decoding levels are replaced by a bit-flipping algorithm and the Chase decoder, both will be described in Section 6.3. The overall GCC FER is the union bound of all level codes $\mathcal{A}_l$, which corresponds to the envelope curves in Figure 4.9. In Figure 4.9 the code parameters are optimized for $FER = 10^{-16}$ in the soft-decoding mode. The curve has a error floor like shape with a bend around the designed FER.

Example 4. *The target* FER *is* 10^{-16} *at* $\varepsilon = 3.8 \cdot 10^{-3}$. *We use the Galois field*

Algorithm 1: search code parameters for hard decoding and error correction only

```
begin
    n_a ← ⌈k/n_b⌉ − 1 ;
    repeat
        n_a ← n_a + 1;
        for l in 0:L-1 do
            P_{b,l} ← 1 − Σ_{j=0}^{t_{b,i}} (n_b choose j) ε(1 − ε)^{n_b − j};
            t_{a,l} ← −1;
            repeat
                t_{a,l} ← t_{a,l} + 1;
                P_{a,l} ← 1 − Σ_{j=0}^{t_{a,l}} (n_a choose j) P_{b,l}(1 − P_{b,l})^{n_a − j} ;
            until P_{a,l} ≤ FER;
        k ← n_a · n_b − 2 Σ_{i=0}^{L−1} t_{a,i} − n_a r_{b,0} ;
    until k ≥ blocksize;
```

Math form of the algorithm steps:

$n_a \leftarrow \lceil \frac{k}{n_b} \rceil - 1$;

$n_a \leftarrow n_a + 1$;

$P_{b,l} \leftarrow 1 - \sum_{j=0}^{t_{b,i}} \binom{n_b}{j} \varepsilon (1-\varepsilon)^{n_b - j}$;

$t_{a,l} \leftarrow -1$;

$t_{a,l} \leftarrow t_{a,l} + 1$;

$P_{a,l} \leftarrow 1 - \sum_{j=0}^{t_{a,l}} \binom{n_a}{j} P_{b,l} (1 - P_{b,l})^{n_a - j}$;

until $P_{a,l} \leq FER$;

$k \leftarrow n_a \cdot n_b - 2 \sum_{i=0}^{L-1} t_{a,i} - n_a r_{b,0}$;

until $k \geq$ *blocksize*;

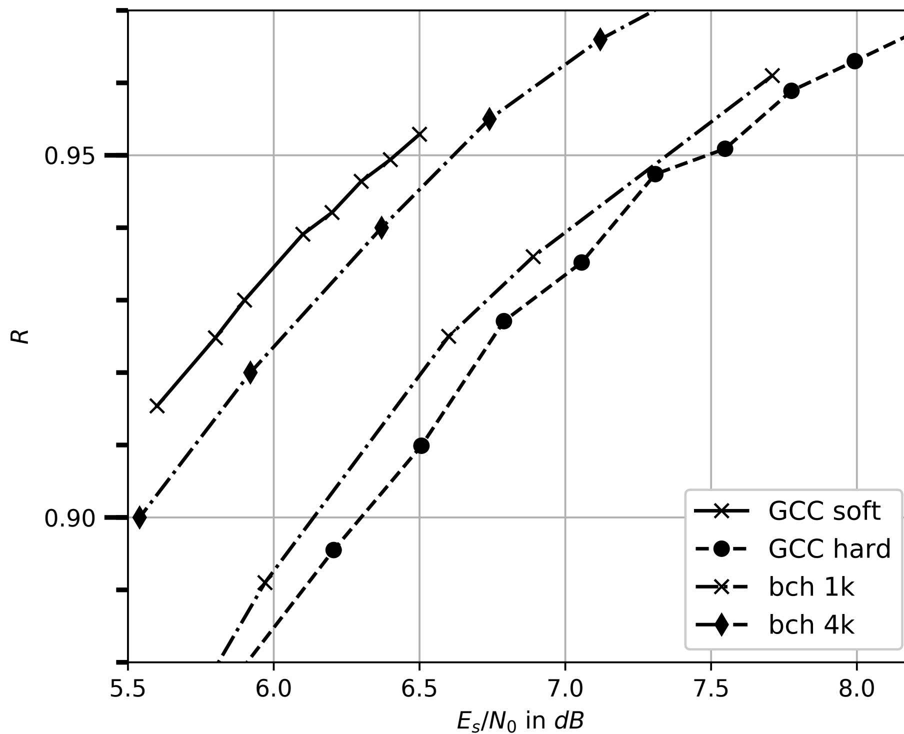


Fig. 4.8: Code rate estimation example

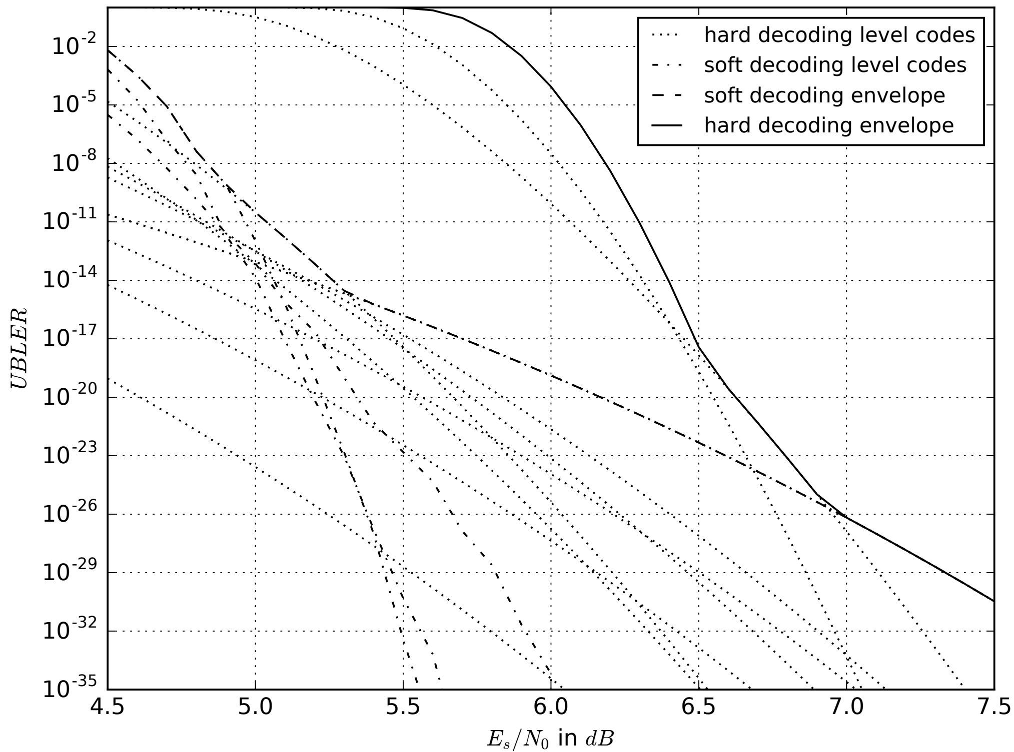


Fig. 4.9: Block error rate example. Reproduced by permission of the Institution of Engineering & Technology [101]

$GF(2^6)$ for the inner BCH codes and $GF(2^9)$ for the outer RS codes. With these parameters the GC code needs $L = 4$ levels and has $n_b = 482$ rows. The inner codes are $\mathcal{B}^{(0)}(2; 42, 36, 3)$, $\mathcal{B}^{(1)}(2; 42, 27, 5)$, $\mathcal{B}^{(2)}(2; 42, 18, 9)$ and $\mathcal{B}^{(3)}(2; 42, 9, 13)$. The outer codes are $\mathcal{A}^{(0)}(2^9; 482, 414, 69)$, $\mathcal{A}^{(1)}(2^9; 482, 466, 27)$, $\mathcal{A}^{(2)}(2^9; 482, 474, 9)$ and $\mathcal{A}^{(3)}(2^9; 482, 478, 5)$. The resulting code rate is $R = 0.81$. For a 2-kByte sector size this codeword has a size of 20268 bits.

4.4 Summary

In this chapter, the idea of GCC has been discussed. Because single block codes suffer from code rate and complexity for higher error correction capability, the GCC approach promises to overcome this by combining smaller component codes. For the later understanding on how the overall error correction capability is determined, the encoding and decoding scheme of this codes was described. Therefore, the idea of nested BCH codes was introduced that uses the linearity of this code type to form several subcodes. The steps to decode a GCC codeword comprises:

- decode inner code $\mathcal{B}^{(i)}$
- re-image to outer code $\mathcal{A}^{(i)}$
- decode outer code $\mathcal{A}^{(i)}$
- re-encode results of $\mathcal{A}^{(i)}$ with $\mathcal{B}^{(i)}$ and subtract from GCC matrix

In principle the component codes of GCC are not necessary algebraic codes but the GCC with algebraic codes enable an analytical error correction analysis based on the BMD property of its component codes, which is one of the important requirement of this work.

This GCC construction has the parameters:

- number of levels L
- GCC matrix dimension n_a and n_b
- the GF field for the inner $\mathcal{B}$ and outer $\mathcal{A}$ component code
- error correction capabilities of each level for both component codes $t_{a,l}$ and $t_{b,l}$

These parameters depend on each other and an optimal combination for a designed channel state has to be searched e.g. with the proposed Algorithm 1 that still needs some manual intervention.

Chapter 5

Interleaver Construction Based on Gaussian Integer

Codes over finite Gaussian integer fields were first studied by Huber in [33]. In this chapter set partitioning of Gaussian integer constellations is considered. Set partitioning and multi-level coding is a common technique to combine coding and modulation. With this method the signal constellation is split into subsets of equal size and the numbering of the subsets is then encoded with different codes. For ordinary Gaussian integer fields, there exist no signal set partitions, because the number of field elements is a prime. In [24, 23], it was demonstrated that the concept of set partitioning can be applied to Gaussian integer rings of size $m = c \cdot d$, where c and d are primes of the form $c \equiv d \equiv 1 \bmod 4$, respectively. This construction is based on additive sub-groups of the ring constellation. Such constellations do not exist for integers m that have any prime factor congruent to 3 modulo 4.

This work extends the results from [24, 23] to Gaussian integer constellations that are constructed from the extension field $GF(p^2)$. In fact, the new construction is applicable for arbitrary integers p, therefore we consider two-dimensional $\mathbb{Z}$-modules over integer rings $\mathbb{Z}_p$. The presented set partitioning is useful for the design of two-dimensional interleavers. Such interleavers can be utilized to correct clusters of errors, which occur for instance in magnetic or optical storage systems.

In contrast to most publications on two-dimensional interleavers, we consider error clusters that have a cyclic structure. Such interleavers may be useful for transmission with orthogonal frequency-division multiplexing (OFDM) over frequency selective channels. The corresponding channel transfer functions typically lead to circularly correlated channel coefficients and therefore cyclic error patterns [59]. To the best of our knowledge, similar error clusters and interleavers were only considered by De Almeida and Palazzo in [14], where the interleaving was designed based on the set partitioning of a $p \times p$ array. The partitioning was obtained by optimizing the minimum squared Euclidean distance in the subsets. In this work, we demonstrate that the Mannheim metric is a suitable distance measure for the set partitioning of circular square arrays. Therefore, we propose a set partitioning based on the Mannheim metric. The interleaver design for square arrays presented in [5] is also based on the partitioning of lattices. This approach was later generalized to rectangular arrays [29]. However, the proposed partitioning optimizes the Manhattan distance in the subsets, which does not regard cyclic structures. Therefore, the construction in [5] cannot be directly applied to circular arrays.

Parts of this chapter are published in [86].

5.1 Preliminaries

Gaussian integers are complex numbers such that the real and imaginary parts are integers. The modulo function of a complex number z is defined as

$$\mu(z) = z \bmod \lambda = z - \left\lfloor \frac{z\lambda^*}{\lambda \cdot \lambda^*} \right\rceil \cdot \lambda, \tag{5.1}$$

where λ^* is the conjugate of the complex number λ. $\lfloor \cdot \rceil$ denotes rounding to the closest Gaussian integer, i.e. for a complex number $z = a + ib$, we have $\lfloor z \rceil = \lfloor a \rceil + i \lfloor b \rceil$. Primes p of the form $p \equiv 1 \bmod 4$ can be decomposed into two complex numbers λ and λ^*, i.e. $p = \lambda \cdot \lambda^*$. In this case, the complex modulo function is an operation-preserving isomorphism for the finite integer field $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ and hence $\mathcal{G}_p = \{\mu(z) | z \in \mathbb{Z}_p\}$ is also a finite field with respect to complex addition and complex multiplication modulo λ.

In his seminal work [33], Huber presented Gaussian integer fields that are constructed from extension fields $GF(p^2)$ for primes p of the form $p \equiv 3 \bmod 4$. For these constellations we use the modulo function with $\lambda = p$. Hence, we have

$$\mu(z) = z \bmod p = z - \left\lfloor \frac{z}{p} \right\rceil \cdot p. \tag{5.2}$$

The set $\mathcal{G}_{p^2}$ is obtained as

$$\mathcal{G}_{p^2} = \{\mu(k + i \cdot l) | k, l \in \mathbb{Z}_p\}. \tag{5.3}$$

We will also consider the case where p is not a prime and therefore $\mathbb{Z}_p$ is a ring over integers. In this case, $\mathcal{G}_{p^2}$ is a module, which is a generalization of the notion of a vector space such that the scalars are the elements of an arbitrary ring.

The L1-norm (Manhattan norm) of a complex number $z = a + ib$ is $||z|| = |a| + |b|$. We define the Mannheim weight of a Gaussian integer z by

$$wt_{\mathrm{M}}(z) = ||\mu(z)||. \tag{5.4}$$

The Mannheim distance between two Gaussian integers y and z is

$$d_{\mathrm{M}}(y, z) = wt_{\mathrm{M}}(y - z). \tag{5.5}$$

Note that this definition coincides with the original definition introduced by Huber in [33]. However, this definition of the Mannheim distance is not a proper distance for all possible Gaussian integer constellations, because the triangle inequality might not hold [47]. This issue occurs if there are Gaussian integers z with $||z|| < ||\mu(z)||$, i.e. the L1-norm of z is smaller than the norm of its representative in the finite set. In the following, we will show that d_{M} fulfills the metric axioms for Gaussian integer sets $\mathcal{G}_{p^2}$ as defined in Equation (5.3).

Lemma 1. *For any module $\mathcal{G}_{p^2}$ with modulo function $\mu(\cdot)$ and for any Gaussian integer z we have*

$$||\mu(z)|| \leq ||z||. \tag{5.6}$$

Proof. Let a and b be the real part and the imaginary part of $\frac{z}{p}$. Then, with (5.1) we have

$$\frac{\mu(z)}{p} = \frac{z}{p} - \left\lfloor \frac{z}{p} \right\rceil = a - \lfloor a \rceil + i(b - \lfloor b \rceil).$$

Due to rounding we have $|a - \lfloor a \rceil| \leq |a|$ and $|b - \lfloor b \rceil| \leq |b|$. Consequently, we obtain

$$||\frac{\mu(z)}{p}|| \leq ||\frac{z}{p}|| = |a| + |b|$$

and therefore $||\mu(z)|| \leq ||z||$. □

Theorem 1. *The Mannheim distance d_{M} defines a distance over $\mathcal{G}_{p^2}$.*

Proof. Let a, b, c be three elements from $\mathcal{G}_{p^2}$. We have to prove that the triangular inequality is satisfied. Consider $x = \mu(a-c)$ and $y = \mu(c-b)$. We have $\mu(a-b) = \mu(x+y)$ and therefore $d_{\mathrm{M}}(a,b) = ||\mu(a-b)|| = ||\mu(x+y)||$. From Lemma 1 follows

$$||\mu(x+y)|| \leq ||x+y|| \leq ||x|| + ||y||.$$

With $d_{\mathrm{M}}(a,c) = ||x||$ and $d_{\mathrm{M}}(c,b) = ||y||$ we obtain

$$d_{\mathrm{M}}(a,b) \leq d_{\mathrm{M}}(a,c) + d_{\mathrm{M}}(c,b).$$

□

5.2 Set Partitioning

In the following, we will construct subsets of the module $\mathcal{G}_{p^2}$, i.e. we partition $\mathcal{G}_{p^2}$ into p disjoint subsets $\mathcal{G}_{p^2}^{(k)}, k = 0, \ldots, p-1$ with p elements. Each subset should have a large minimum distance, where we consider the Mannheim distance as distance measure. We define the minimum intra-partition distances $\Delta^{(k)}$ of $\mathcal{G}_{p^2}^{(k)}$ as

$$\Delta^{(k)} = \min_{x,y \in \mathcal{G}_{p^2}^{(k)}, x \neq y} d_{\mathrm{M}}(x,y). \tag{5.7}$$

The overall minimum Mannheim distance is the minimum of all intra-partition distances

$$\Delta = \min_{k=0,\ldots,p-1} \Delta^{(k)}. \tag{5.8}$$

For the minimum Mannheim distance we have the following sphere-packing bound.

Theorem 2. *Let Δ be the minimum Mannheim distance in any of p subsets $\mathcal{G}_{p^2}^{(k)}$ with $k = 0, \ldots, p-1$. The number of subsets p satisfies*

$$p \geq \begin{cases} \frac{\Delta^2+1}{2} & \text{for odd } \Delta \\ \frac{(\Delta-1)^2+1}{2} & \text{for even } \Delta \end{cases} \tag{5.9}$$

Proof. Let Δ be the minimum Mannheim distance in any subset $\mathcal{G}_{p^2}^{(k)}$. Now consider any set of points (sphere) where every point has at most Mannheim distance $t = \lfloor \frac{\Delta-1}{2} \rfloor$ to a center point. All of these points belong to different subsets, because the distance between any two points in this sphere is smaller than Δ. Consequently, the number of subsets has to be greater or equal to the number of points in the sphere. We bound p by counting the number of points

in a sphere of radius t. Note that there are at least $4j$ points with distance j to the center. Therefore, the total number of points in a sphere of radius t is

$$N \geq 1 + 4\sum_{j=1}^{t} j = 2t^2 + 2t + 1. \tag{5.10}$$

If Δ is odd, then $t = \lfloor \frac{\Delta-1}{2} \rfloor = \frac{\Delta-1}{2}$. With $p \geq N$ we have

$$p \geq 2t^2 + 2t + 1 = 2\left(\frac{\Delta-1}{2}\right)^2 + 2\left(\frac{\Delta-1}{2}\right) + 1 = \frac{\Delta^2+1}{2}$$

Similarly, for even values of Δ we have $t = \lfloor \frac{\Delta-1}{2} \rfloor = \frac{\Delta}{2} - 1$ and

$$p \geq 2t^2 + 2t + 1 = \frac{(\Delta-1)^2+1}{2}.$$

□

Note that the sphere-packing bound of Theorem 2 can be fulfilled with equality. Therefore, we define the notion of a perfect set partitioning in analogy to perfect error-correcting codes.

Definition 17. *Let Δ be the minimum Mannheim distance in any subset $\mathcal{G}_{p^2}^{(k)}$. This set partitioning is perfect, if*

$$\Delta = \sqrt{2p-1}. \tag{5.11}$$

Next, we construct set partitions with good minimum Mannheim distance. We first consider the case, where p is an arbitrary prime. Note that we can consider the extension field $GF(p^2)$ as a two-dimensional vector space (Euclidean geometry) over the base field $\mathbb{Z}_p$ [42]. The mapping $\mu(\cdot)$ is operation-preserving with respect to addition and multiplication with elements from the base field. Therefore, $\mathcal{G}_{p^2}$ is also a two-dimensional Euclidean geometry. We will denote the corresponding base field by

$$\mathcal{G}_p = \{\mu(k) | k \in \mathbb{Z}_p\}. \tag{5.12}$$

In the following we construct set partitions as lines (1-flats) in $\mathcal{G}_{p^2}$. We use bold symbols to denote elements $\mathbf{a} \in \mathcal{G}_{p^2}$, i.e. points in the Euclidean geometry. Let $\mathbf{a} \in \mathcal{G}_{p^2}$ be a non-zero point. Then, the p points $\{\mu(\beta\mathbf{a}) | \beta \in \mathcal{G}_p\}$ form a line that passes through the origin (the zero point). Let $\mathbf{a}_0$ and $\mathbf{a}$ be two linearly independent points in $\mathcal{G}_{p^2}$, i.e. $\beta_0\mathbf{a}_0 + \beta\mathbf{a} \neq 0$ unless $\beta_0 = \beta = 0$. Then, the p points $\{\mu(\beta\mathbf{a} + \mathbf{a}_0) | \beta \in \mathcal{G}_p\}$ form a line through the point $\mathbf{a}_0$. The two lines $\beta\mathbf{a}$ and $\beta\mathbf{a} + \mathbf{a}_0$ are parallel lines, i.e. they have no common points. The set $\{\mu(\beta\mathbf{a} + \mathbf{a}_0) | \beta \in \mathcal{G}_p\}$ is a coset of $\{\mu(\beta\mathbf{a}) | \beta \in \mathcal{G}_p\}$. For any line through the origin there are $p-1$ parallel lines. We use this fact to form the partitions of the set $\mathcal{G}_{p^2}$. Clearly choosing $\mathbf{a}_k = \mu(k), k \in \mathbb{Z}_p$ as an element of the base field and $\mathbf{a} \in \mathcal{G}_{p^2}$ with non-zero imaginary part, the two points $\mathbf{a}_k, \mathbf{a}$ are linearly independent. Hence, we obtain the p subsets for $k = 0, \ldots, p-1$

$$\mathcal{G}_{p^2}^{(k)} = \{\mu(\beta\mathbf{a} + \mathbf{a}_k) | \beta \in \mathcal{G}_p\} \tag{5.13}$$

Now, we consider the general case. If p is not a prime, then $\mathcal{G}_p$ is a ring and $\mathcal{G}_{p^2}$ is a module. In this case, Equation (5.13) defines a coset if the real part and imaginary part of $\mathbf{a}$ are no zero divisors.

We chose $\mathbf{a} = c + id$ to construct the set of parallel lines, where $c, d \in \mathbb{N}, c \geq d$ are the two largest integers satisfying $c^2 + d^2 \leq p$. This construction generates all perfect partitions with respect to the minimum Mannheim distance. In order to prove this result we require the following lemmas.

Lemma 2. *Consider the partition of the module $\mathcal{G}_{p^2}$ into the p subsets $\mathcal{G}_{p^2}^{(0)}, \ldots, \mathcal{G}_{p^2}^{(p-1)}$. The minimum intra-partition distances $\Delta^{(k)}$ for $k = 0, 1 \ldots, p-1$ satisfy*

$$\Delta^{(k)} = \min_{\mathbf{z} \in \mathcal{G}_{p^2}^{(0)} \setminus \{0\}} ||\mathbf{z}||. \tag{5.14}$$

Proof. Consider two elements $\mathbf{z} \neq \mathbf{z}'$ from the same subset. With $\mathbf{z}'' = \mu(\mathbf{z} - \mathbf{z}')$, the Mannheim distance is

$$d_{\mathrm{M}}(\mathbf{z}, \mathbf{z}') = ||\mu(\mathbf{z} - \mathbf{z}')|| = ||\mathbf{z}''||, \quad \mathbf{z}'' \neq 0, \mathbf{z}'' \in \mathcal{G}_{p^2}^{(0)}$$

Hence, we have

$$\delta^{(k)} = \min_{\mathbf{z}, \mathbf{z}' \in \mathcal{G}_{p^2}^{(k)}, \mathbf{z} \neq \mathbf{z}'} ||\mathbf{z} - \mathbf{z}'||^2 = \min_{\mathbf{z}'' \in \mathcal{G}_{p^2}^{(0)}, \mathbf{z}'' \neq 0} ||\mathbf{z}''||^2.$$

which holds for all parallel lines (cosets). □

Next, we bound the minimum distance in the subsets of $\mathcal{G}_{p^2}$. However, we consider only integers p that satisfy the following Diophantine equation $p = c^2 + d^2$ for integers c and d.

Construction 1 Let p be an integer satisfying $p = c^2 + d^2$ with $c, d \in \mathbb{N}$, where c and d are relatively prime. Furthermore, neither c nor d is a zero divisor. We obtain the p subsets for $k = 0, \ldots, p-1$ according to Equation (5.13) with $\mathbf{a} = c + id$ and $\mathbf{a}_k = \mu(k), k \in \mathbb{Z}_p$.

Theorem 3. *Consider the partition of the module $\mathcal{G}_{p^2}$ into the p subsets $\mathcal{G}_{p^2}^{(0)}, \ldots, \mathcal{G}_{p^2}^{(p-1)}$ according to Construction 1. The minimum Mannheim distance Δ satisfies*

$$\Delta \geq c + d. \tag{5.15}$$

Proof. For integers of the form $p = c^2 + d^2$, p can be decomposed as $p = \lambda\lambda^*$ with $\lambda = c + id$, where λ is a Gaussian prime, because c and d are relatively prime. Now note that all elements $\mathbf{z} \in \mathcal{G}_{p^2}^{(0)} \setminus \{0\}$ are multiples of $\lambda = c + id$, i.e. $\mathbf{z} = \beta \cdot \lambda$ with $\beta \in \mathcal{G}_p \setminus \{0\}$. Therefore, we have

$$\begin{aligned} \mu(\mathbf{z}) &= \beta\lambda - \left\lfloor \frac{\beta\lambda}{p} \right\rceil p \\ &= \lambda \left(\beta - \left\lfloor \frac{\beta\lambda}{p} \right\rceil \lambda^* \right) \\ &= \lambda \cdot A, \text{ with } A = \left(\beta - \left\lfloor \frac{\beta\lambda}{p} \right\rceil \lambda^* \right). \end{aligned}$$

Note that $\mu(\mathbf{z}) \neq 0$ implies that A is a non-zero Gaussian integer. Moreover, any non-zero Gaussian integer has a norm of at least one. Hence, we have

$$||\mu(\mathbf{z})|| = ||\lambda|| \cdot ||A|| \geq ||\lambda|| = c + d.$$

Using Lemma 2, we conclude that $\Delta \geq c + d$ holds for all parallel lines. □

Combining the results from Theorem 2 and Theorem 3, we note that the upper bound may coincide with the lower bound.

$p = \delta$	c	d	Δ	Δ upper bound
5	2	1	3	3
10	3	1	4	4
13	3	2	5	5
17	4	1	5	5
25	4	3	7	7
26	5	1	6	7
29	5	2	7	7
34	5	3	8	8
37	6	1	7	8
41	5	4	9	9
50	7	1	8	9
53	7	2	9	10
58	7	3	10	10
61	6	5	11	11
65	7	4	11	11
73	8	3	11	12
74	7	5	12	12
82	9	1	10	12
85	7	6	13	13
89	8	5	13	13
97	9	4	13	13
106	9	5	14	14
113	8	7	15	15
130	9	7	16	16
145	9	8	17	17

Tab. 5.1: Number of subsets, parameters for best partitioning, lower and upper bounds on the minimum Mannheim distance. Bold numbers indicate perfect set partitions.

Corollary 1. *For p of the form $p = c^2 + d^2$ with $c = d + 1$, the partition according to Construction 1 of the module $\mathcal{G}_{p^2}$ into the p subsets $\mathcal{G}_{p^2}^{(0)}, \dots, \mathcal{G}_{p^2}^{(p-1)}$ is a perfect set partitioning. Furthermore, the resulting partitions for $d = 1, 2, 3, \dots$ are the only perfect set partitions.*

Proof. With $c = d+1$ we have $p = c^2+d^2 = 2d^2+2d+1$. Furthermore, with $\Delta = c+d = 2d+1$ we have

$$\frac{\Delta^2 + 1}{2} = 2d^2 + 2d + 1 = p$$

and therefore $\Delta = \sqrt{2p-1}$. Finally, note that $d = 1, 2, 3, \dots$ generates all possible integer solutions to $p = 2d^2 + 2d + 1$. □

Table 5.1 presents all integers up to 145 that fulfill the condition of Construction 1. The bold numbers indicate perfect partitions.

Construction 1 is only applicable for integers p that satisfy the Diophantine equation $p = c^2 + d^2$ for integers c and d. For other integers, we suggest a simple search algorithm to find the line $\mathbf{a}$ that achieves the maximum minimal distance.

Construction 2

1. Initialize $\Delta = 0$ and $c_{max} = \lfloor\sqrt{2p-1}\rfloor + 1$
2. for c :=1 to c_{max} and for d := 1 to c_{max} repeat the following calculation:
   - set $\mathbf{a}' = c + id$
   - construct the set
   $$\mathcal{G}_{p^2}^{(0)} = \{\mu(\beta\mathbf{a}')|\beta \in \mathcal{G}_p\}.$$
   - determine
   $$\Delta' = \min_{\mathbf{z}\in\mathcal{G}_{p^2}^{(0)}\setminus\{0\}} ||\mathbf{z}||.$$
   - if $(\Delta' > \Delta)$ then set $\mathbf{a} = \mathbf{a}'$ and $\Delta = \Delta'$.

The algorithm requires $c_{max}^2 \approx 2p$ iterations. The construction of the set $\mathcal{G}_{p^2}^{(0)}$ as well as the calculation of the minimum norm require $O(p)$ operations in standard O notation. Hence, the algorithm has complexity $O(p^2)$. According to Lemma 2 the minimum distance is determined by the norm of the elements of the set $\mathcal{G}_{p^2}^{(0)} \setminus \{0\}$. In order to find the line $\mathbf{a}$ that achieves the maximum minimal distance in all subsets, it is sufficient to consider all values of c and d with $c, d \in [1, c_{max}]$, because according to Theorem 2, the minimum Mannheim distance satisfies $\Delta \leq \sqrt{2p-1} + 1$.

5.3 Interleaver Design

An application of the presented set partitioning is the design of two-dimensional interleavers that can be used to correct clusters of errors. Such errors occur in magnetic or optical storage systems. We consider error clusters that have a square and possibly cyclic shape. Similar error clusters and interleavers were presented by De Almeida and Palazzo in [14]. De Almeida and Palazzo designed the interleaving based on the set partitioning of a $p \times p$ array. However, the partitioning is obtained by optimizing the minimum squared Euclidean distance in the subsets. We will demonstrate that the cyclic error patterns considered by De Almeida and Palazzo can be characterized with the Mannheim distance. Therefore, we propose to use the Mannheim distance for the design of the interleavers.

Figure 5.1 depicts 5×5 square arrays with two examples for two-dimensional clusters of errors. In Figure 5.1 a) the center of the cluster lies in the center of the array. The cluster represents a sphere with radius one, i.e. all elements of the cluster have Mannheim distance one to the center point. Similarly, Figure 5.1 b) depicts a cluster with a cyclic structure where the center point is in the right bottom corner of the square array. Again, the cluster represents a sphere with radius one with respect to the Mannheim distance.

The aim of the interleaving is to distribute all points in an error cluster to different rows and columns of the array. Therefore, the errors can be corrected by applying error-correcting codes to each row or column of the array. In order to design the interleaver, we consider the elements of the module $\mathcal{G}_{p^2}$ as indices of a $p \times p$ square array, i.e. the real part of $z \in \mathcal{G}_{p^2}$ can be interpreted as column index and the imaginary part as row index. Hence, the interleaver is defined by a bijective mapping $\mathcal{G}_{p^2} \to \mathcal{G}_{p^2}$. To obtain an interleaving as depicted in Figure 5.1, all indices of one row of the array (all elements of $\mathcal{G}_{p^2}$ with the same imaginary part) should be mapped to the same coset of $\mathcal{G}_{p^2}$. Similarly all indices of one column of the array should be mapped to the same coset of $\mathcal{G}_{p^2}$. Such a mapping can be constructed as follows:

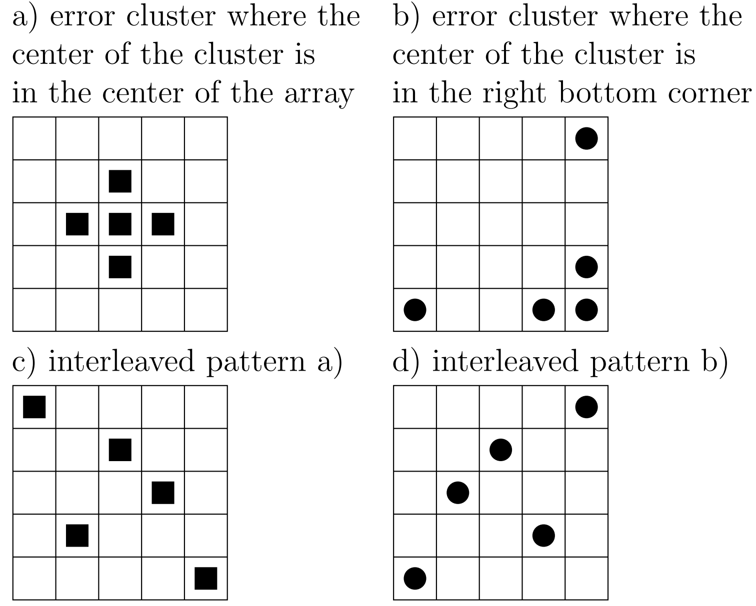


Fig. 5.1: Example of the interleaving of two-dimensional error clusters. Reproduced by permission of the Institution of Engineering & Technology [86]

Construction 3 We choose two Gaussian integers $\mathbf{a}_1 = c + id$ and $\mathbf{a}_0 = d + ic$ in $\mathcal{G}_{p^2}$ such that $\mathbf{a}_1$ maximizes the minimum Mannheim distance in all subsets and $c^2 - d^2$ is non-zero and has a multiplicative inverse in $\mathbb{Z}_p$. The interleaver design is based on the following mapping

$$z = a + ib \rightarrow \tilde{z} = \tilde{a} + i\tilde{b} = \mu\left(a\mathbf{a}_1 + b\mathbf{a}_0\right) \; \forall z \in \mathcal{G}_{p^2} \tag{5.16}$$

with the inverse mapping

$$\begin{aligned} \tilde{z} = \tilde{a} + i\tilde{b} \rightarrow z &= a + ib \, \forall \tilde{z} \in \mathcal{G}_{p^2} \\ a = \mu\left(\frac{\tilde{a}c - \tilde{b}d}{c^2 - d^2}\right) &, \quad b = \mu\left(\frac{\tilde{b}c - \tilde{a}d}{c^2 - d^2}\right) \end{aligned} \tag{5.17}$$

Theorem 4. *Consider the module $\mathcal{G}_{p^2}$ with a partitioning according to Construction 3 with minimum Mannheim distance Δ. The inverse mapping according to Equation (5.17) maps all elements of a cluster of radius*

$$t = \lfloor \frac{\Delta - 1}{2} \rfloor \tag{5.18}$$

to different rows and different columns (elements of $\mathcal{G}_{p^2}$ with different real and different imaginary part).

Proof. The mapping defined in Construction 3 is bijective, if the inverse mapping exists, i.e. if $c^2 - d^2$ is non-zero and has a multiplicative inverse in $\mathbb{Z}_p$, where $c^2 - d^2 \neq 0$ follows from $c > d$.

Note that for $c, d \in \mathcal{G}_p$ with $c > d > 0$ the two points $\mathbf{a}_1$ and $\mathbf{a}_0$ are linearly independent. Therefore, all elements of $\mathcal{G}_{p^2}$ with the same imaginary part are mapped to the same coset of $\mathcal{G}_{p^2}$. Similarly all elements of $\mathcal{G}_{p^2}$ with the same real part are mapped to the same coset of $\mathcal{G}_{p^2}$. This guarantees that all elements with the same real or imaginary part have a Mannheim distance of at least Δ after interleaving.

Now, assume an error cluster (sphere) of radius t. Any two elements of this cluster have a Mannheim distance smaller than Δ. Therefore, all elements of this cluster lie in different

cosets. Consequently, the inverse mapping according to Equation (5.17) maps all elements of the cluster to different rows and different columns. □

Example 5. *We consider the module* $\mathcal{G}_{5^2}$. *Construction 1 obtains a perfect partitioning with* $\mathbf{a} = 2 + i$ *and minimum Mannheim distance* $\Delta = 3$. *Using this partitioning, Construction 3 results in the interleaving depicted in Figure 5.1. The mapping according to Equation (5.16) maps any element of the same row to a different coset. For example, all elements with real part equal to zero are mapped to* $\mathcal{G}_{5^2}^{(0)}$. *With* $\Delta = 3$ *the de-interleaving maps all elements of a cluster of radius* $t = 1$ *to different rows and different columns.*

Example 6. *For* $p > 29$, *optimization according to the squared Euclidean distance or the minimum Mannheim distance typically results in different partitions. Consider for instance* $p = 113$. *For* $p = 113 = 8^2 + 7^2$, *Construction 1 results in a perfect partitioning (with respect to the minimum Mannheim distance) with* $\mathbf{a} = 8 + 7i$, $\Delta = 15$, *and* $\delta = 113$. *However, this partitioning is not optimal with respect to the minimum squared Euclidean distance. The algorithm of Construction 2 (using the minimum squared Euclidean distance) results in* $\mathbf{a}' = 11 + 2i$ *with minimum squared Euclidean distance* $\delta' = 11^2 + 2^2 = 125$, *but* $\Delta' = 13$. *A sphere with respect to the best possible squared Euclidean distance* $\delta = 125$ *contains 97 points, whereas we have 113 points in a sphere with respect to the Mannheim distance* $\Delta = 15$. *Moreover, the sphere with respect to the Mannheim distance covers the complete sphere with respect to the squared Euclidean distance, i.e. every point in the sphere with respect to the squared Euclidean distance is also a point in the sphere with respect to the Mannheim distance.*

In the last example, the sphere with respect to the Mannheim distance covers a larger number of points than the sphere with respect to the squared Euclidean distance. This motivate the optimization of the partitioning with respect to the Mannheim distance. Unfortunately, this observation is not fulfilled in general. Consider for instance the constellations presented in Tab. 5.1. For all values of $p > 29$ with the exception of 53 the optimization with respect to the Mannheim distance leads to a sphere that contains more points than the sphere with respect to the squared Euclidean distance. Moreover, all constellations except of 73 and 97 cover the complete sphere with respect to the Mannheim distance.

Finally, we provide an example that demonstrates that a partitioning which optimizes the minimum Manhattan distance, does not necessarily result in an optimal circular interleaving.

Example 7. *We refer to Example 2.3 from [5] which is an* 8×8 *interleaver obtained with* $\mathbf{a} = 5 + i$. *Each of the eight subsets has minimum Manhattan distance 4. However, it is easily verified that the minimum Mannheim distance is only three. Note that* $\mu(2\mathbf{a}) = 2 + 2i$. *This is the vector with minimum Mannheim weight among the multiples of* $\mathbf{a}$. *However, 2 is a zero divisor in* $\mathbb{Z}_{10}$. *Therefore, the multiples of* $2+2i$ *do not generate the complete coset. According to Construction 2, we find* $\mathbf{a} = 3 + i$. *This leads to a set partition with minimum Mannheim distance* $\Delta = 4$.

5.4 GCC for Correcting Two-Dimensional Cluster Errors and Independent Errors

Two-dimensional interleavers can be used to correct clusters of errors. The shape of the clusters can be taken into account by using an appropriate metric for the interleaver design. Such interleavers are based on the set partitioning of an $n \times n$ array, where the partitioning is obtained by optimizing the minimum distance in the subsets. For instance, interleavers

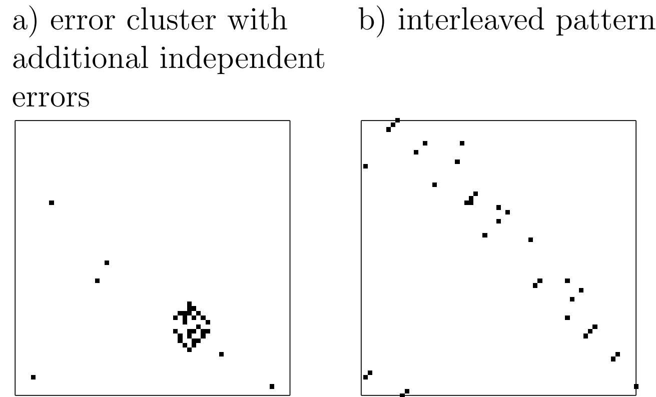


Fig. 5.2: Example of the interleaving of two-dimensional error clusters (61×61 square arrays). Reproduced by permission of the Institution of Engineering & Technology [86]

for circular clusters were presented by De Almeida and Palazzo in [14] using the squared Euclidean distance. In [86], we considered error clusters that have a square and possibly cyclic shape applying the Mannheim metric (The Mannheim distance is similar to the taxicab distance between two points located on a grid, where an allowed path considers only vertical or horizontal lines.).

Let $V = \{(i,j)|0 \leq i < n, 0 \leq j < n\}$ be the set of all indices of an $n \times n$ matrix, where i denotes the row index and j the column index, respectively. An interleaver is a bijective mapping $\mathbf{M} : V \to V$. Such a mapping can for instance be defined by a matrix multiplication $\tilde{\mathbf{v}} = \mathbf{v}\mathbf{M}$, $\tilde{\mathbf{v}}, \mathbf{v} \in V$ where addition and multiplication are performed modulo n [14]. For example, Figure 5.1 depict 5×5 square arrays with examples for a two-dimensional interleaver and a two-dimensional cluster of errors. In Figure 5.1 a) the center of the cluster lies in the center of the array. The cluster represents a sphere with radius one, i.e. the elements $(2,1), (3,2), (2,3), (1,2)$ of the cluster have Mannheim distance one to the center point $(2,2)$. The interleaving is obtained with

$$\mathbf{M} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} . \tag{5.19}$$

Hence, the errors are mapped to the positions $(0,4), (3,2), (1,1), (2,3), (4,0)$, i.e. all errors in the cluster are interleaved to different rows and different columns. Similarly, Figure 5.2 a) and d) depict 61×61 square arrays. Here an error cluster and some additional independent errors are depicted. The channel interleaving guarantees that all bit errors within a cluster of errors are interleaved to independent rows of the matrix [86]. Therefore, these errors can be corrected with interleaved codes, where binary error-correcting codes are applied independently to each row. For the examples presented in Figure 5.1 and 5.2 single-error correcting codes are sufficient to correct the cluster errors. In general, employing a T error-correcting code in each row, these n interleaved codes can decode T clusters of errors. However, additional independent errors that occur outside the clusters (cf. Figure 5.2 b)) may lead to decoding errors in some rows. Subsequently, we will demonstrate that such errors can be corrected with an additional outer code in a generalized concatenated coding scheme.

5.4.1 Generalized Concatenated Codes

In this section, we provide a brief introduction to GCC, before we discuss the multistage decoding algorithm for GCC and derive the error correction capability with respect to cluster errors. In the following we use the well known notation $\mathcal{C}(2^m, n, k, t)$ to denote a code $\mathcal{C}$ of

length n and dimension k over the Galois field $GF(2^m)$. The last parameter t denotes the error-correcting capability of the code $\mathcal{C}$.

In contrast to the GCC in Chapter 4, we do not necessarily use outer encoding in all levels. Therefore, we can also omit the restriction $k_{i,l} = lm$. Note that an interleaved code is a special case of the GCC coding scheme, where we use no outer encoding and the information is encoded with n parallel inner encoders for the same code $\mathcal{B}_1(2, n_i, k_i, t_i)$. Assuming proper interleaving, this interleaved code can correct t_i error clusters. However, if t_i error clusters occur, a single independent error may cause a decoding failure, because one row of the codeword matrix may contain more than t_i errors. Such an error event could be corrected with a single outer code with $t_o = 1$. We demonstrate the basic concept with the following example.

Example 8. *To construct a code for cluster error correction, we use binary BCH codes as inner codes:* $\mathcal{B}^{(1)}(2, 31, 26, 1)$ *and* $\mathcal{B}^{(2)}(2, 31, 21, 2)$. *However, we only use one outer RS code over* $GF(2^5)$: $\mathcal{A}^{(1)}(2^5, 31, 29, 1)$ *to form five columns of the matrix. These five bits per row are encoded with the code* $\mathcal{B}^{(1)}(2, 31, 26, 1)$ *where the remaining 21 = 26 - 5 bits are set to zero. The remaining 21 columns of the codeword matrix are filled with information bits without outer encoding. These information bits are encoded with the inner code* $\mathcal{B}^{(2)}(2, 31, 21, 2)$. *Finally, the rows of the codeword matrix are obtained by adding all corresponding codewords from* $\mathcal{B}^{(1)}$ *and* $\mathcal{B}^{(2)}$. *Due to* $\mathcal{B}^{(2)} \subset \mathcal{B}^{(1)}$, *every row is a codeword of* $\mathcal{B}^{(1)}$. *The overall code has length* $31^2 = 961$, *dimension* $k = 5 \cdot 29 + 31 \cdot 21 = 796$ *and requires only ten additional redundancy bits compared to an interleaved code with* $\mathcal{B}^{(1)}(2, 31, 26, 1)$.

$n = 31$ *is sufficiently large to interleave a cluster of radius* 3 *(cf. [86]). Such a cluster contains up to 25 bit errors. Hence, the code can correct any single-error cluster with at most 25 bits. However, an additional independent error may cause a row in the codeword matrix with two errors. Therefore, decoding of the corresponding row will lead to a decoding error after the inner decoding. Such a decoding error affects at most one symbol of the outer RS code. Hence, the outer code can be decoded correctly. Once, the code symbols of the outer code are known, we can start a second round of inner decoding. In this decoding step we can use the nested inner code* $\mathcal{B}^{(2)}(2, 31, 21, 2)$. *Hence, the row containing two errors can be corrected in the second decoding round.*

5.4.2 Code Constructions

Construction 1 We construct a GCC with L levels and with an $n \times n$ codeword matrix that corrects T clusters of errors and $t \leq L$ independent errors. For inner encoding we use the nested binary BCH codes $\mathcal{B}^{(L)} \subset \mathcal{B}^{(L-1)} \subset \ldots \subset \mathcal{B}^{(1)}$ of length n with $t_{i,j} = T + j - 1, j = 1, \ldots, L$. As outer codes we use RS codes with

$$t_{o,j} = \lfloor \frac{t}{j} \rfloor. \tag{5.20}$$

For interleaving we apply a two-dimensional channel interleaver as proposed in [86].

Theorem 5. *The GCC according to Construction 1 corrects T clusters of errors and additionally t independent bit errors.*

Proof. The channel interleaving guarantees that all bit errors within a cluster of errors are interleaved to independent rows of the codeword matrix [86]. Therefore T cluster errors cause at most T errors in each row of the codeword matrix. This T errors can be corrected in the first decoding round by the inner code $\mathcal{B}^{(1)}$ if no additional errors occur.

We prove the proposition for t independent errors by induction. For the base case, note that any independent error can cause a row with $T+1$ or more errors. However, there are at most t such rows and therefore at most t symbol errors in the first outer code. Hence, $\mathcal{A}^{(1)}$ can be decoded correctly.

For the inductive step, we assume that all outer codes up to level j were decoded correctly. Thus, the round $j+1$ of decoding can be performed with the inner code $\mathcal{B}^{(j+1)}$ that corrects $T+j$ errors. Now, note that t independent errors can result in at most $\lfloor \frac{t}{j+1} \rfloor$ rows with more than $T+j$ errors and thus at most $\lfloor \frac{t}{j+1} \rfloor$ errors in the outer code $\mathcal{A}^{(j+1)}$. However, from condition (5.20) we conclude that these errors can be corrected by the outer code $\mathcal{A}^{(j+1)}$. □

Most known code constructions (including Construction 1) consider only the worst case, where an error cluster contains the maximum number P of errors. However, if we assume that errors within a cluster occur not with probability one, but with probability p, the expected number of errors pP might be significantly smaller than P. Next, we construct a code that can correct two clusters of errors, where we assume that one cluster contains only a few bit errors.

Construction 2 We construct a GCC with two levels and with an $n \times n$ codeword matrix that corrects one cluster with up to $e_1 \leq P$ errors and one cluster with at most $e_2 < P$ errors. For inner encoding we use the nested binary BCH codes $\mathcal{B}^{(2)} \subset \mathcal{B}^{(1)}$ of length n with $t_{i,1} = 1$ and $t_{i,2} = 2$. We use one outer RS code with $t_o = e_2$. The information bits for the second level are not encoded with an outer code. For interleaving we apply a two-dimensional channel interleaver as proposed in [86].

Theorem 6. *The GCC according to Construction 2 corrects two clusters of errors, where one cluster may contain $e_1 \leq P$ and the second cluster at most $e_2 < P$ errors.*

Proof. Without loss of generality we assume $e_1 \geq e_2$. Again, the channel interleaving guarantees that all bit errors within a cluster of errors are interleaved to independent rows of the codeword matrix [86]. Therefore any cluster error causes at most one error in each row of the codeword matrix. Therefore, $e_1 - e_2$ rows contain at most one error and can be corrected in the first decoding round by the inner code $\mathcal{B}^{(1)}$. Furthermore, there are at most e_2 rows, which contain two errors. These rows may cause symbol errors in the outer code $\mathcal{A}^{(1)}$. However, the number of symbol errors is at most e_2 and can therefore be corrected by the outer code. Thus, the second round of decoding can be performed with the inner code $\mathcal{B}^{(2)}$ that corrects two errors. Consequently, all rows containing two errors can be corrected in the second decoding step. □

Actually, a code according to Construction 2 may correct many error patterns with $e_1 \geq e_2 > t_o$. To demonstrate this, assume that two independent clusters with $e_1 \geq e_2$ errors occurred. We consider the conditional probability $p(j, e_1, e_2)$ that j rows of the codeword matrix contain two errors. Note that there are $\binom{n}{e_2}$ possibilities to place e_2 errors in n rows. Similarly, there are $\binom{e_1}{j}$ possibilities, where j errors from cluster two are mapped to the same row as one of the e_1 errors of cluster one. For each of these patterns, there are $\binom{n-e_1}{e_2-j}$ possible patterns for the remaining $e_2 - j$ errors. Hence, we have

$$p(j, e_1, e_2) = \frac{\binom{e_1}{j}\binom{n-e_1}{e_2-j}}{\binom{n}{e_2}} \tag{5.21}$$

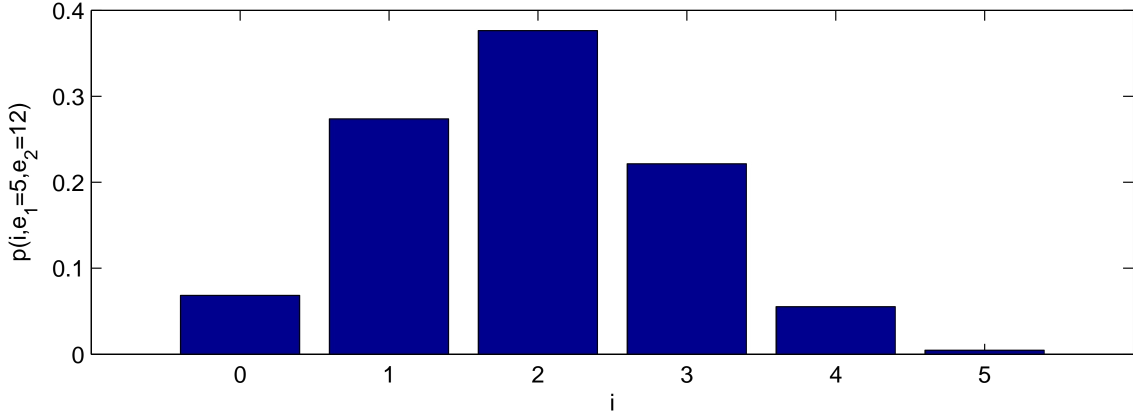


Fig. 5.3: Conditional probability $p(j, e_1, e_2)$ for $n = 31$, $e_1 = 12$, and $e_2 = 5$. Reproduced by permission of the Institution of Engineering & Technology [86]

Now, note that a code according to Construction 2 can correct any error pattern, where the number of rows containing two errors is less or equal t_o, i.e. the fraction

$$\sum_{j=0}^{t_o} p(j, e_1, e_2) \tag{5.22}$$

of all error patterns with two clusters with $e_1 \geq e_2$ errors.

Example 9. *We consider a code according to Construction 2 with $n = 31$. We use binary BCH codes as inner codes: $\mathcal{B}^{(1)}(2, 31, 26, 1)$ and $\mathcal{B}^{(2)}(2, 31, 21, 2)$. We only use one outer RS code over $GF(2^5)$: $\mathcal{A}^{(1)}(2^5, 31, 23, 4)$. According to Theorem 6 this code can correct any two cluster errors with e_1 up to 25 and e_2 at most 4. However, this code can correct 99.5% of all errors with $e_1 = 12$ and $e_2 = 5$. The corresponding conditional probabilities $p(j, e_1, e_2)$ are plotted in Figure 5.3.*

5.4.3 Simulations

In order to compare the performance of the constructed GCC to interleaved codes, we consider a Monte Carlo simulation for a channel with two-dimensional errors. For the simulation we use codes of size 31×31. Such a code could be used for instance for QR codes [65]. For the interleaved codes we use 31 BCH codes with error-correcting capability one or two. Hence, the two codes have code rate 0.84 or 0.68, respectively. For the GCC, we use the code from Example 9 with code rate 0.8. For the channel we assume that one cluster with at most 25 errors occurs. The center is placed randomly and the probability of a bit error within the cluster is 0.5. Additionally, independent bit errors occur with probability p. The corresponding simulation results are depicted in Figure 5.4, where the curves represent FER after decoding. The GCC outperforms the interleaved code with similar rate. Furthermore, we observe that the performance of the GCC is close to that of the two-cluster error-correcting code for $p < 0.005$. We conclude that the GCC obtains the same results as interleaved codes for $t = 2$ at medium-low error probability, whereas the code rate of the GCC is much higher.

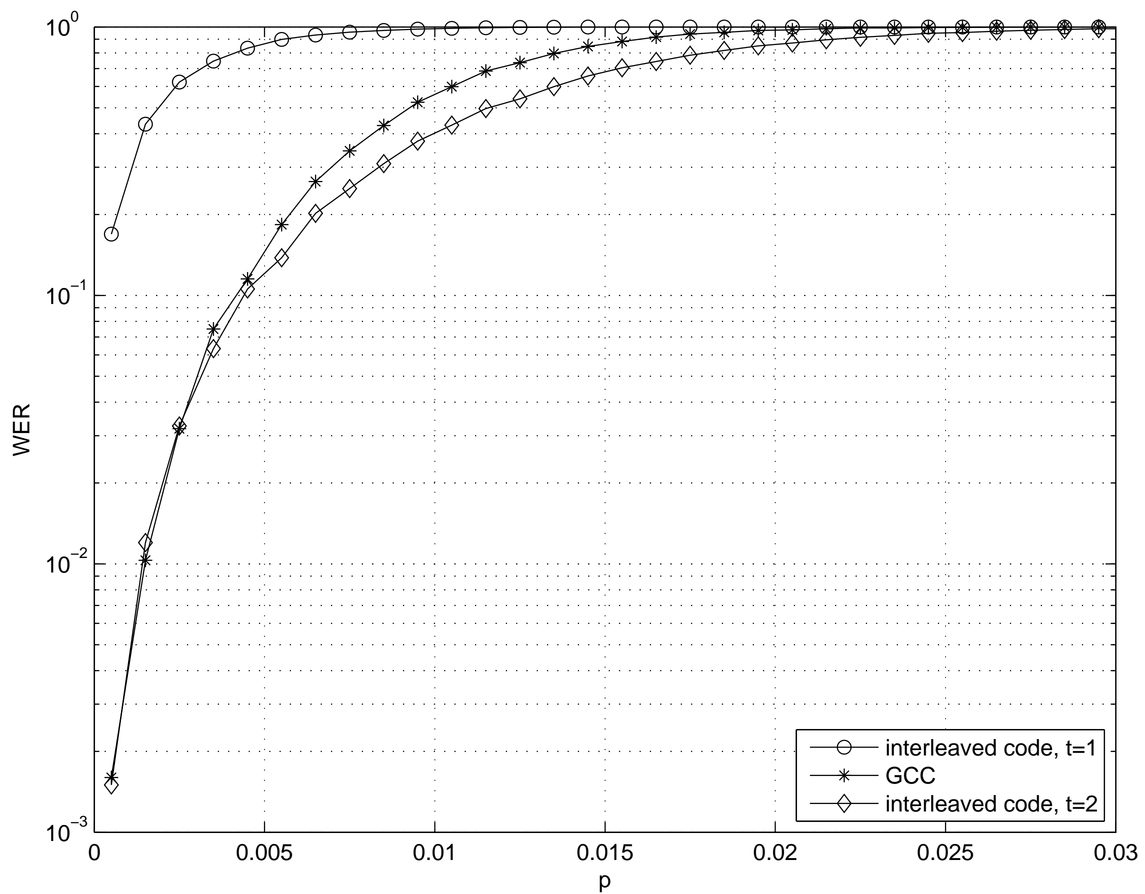


Fig. 5.4: Simulation results for interleaved codes and the GCC of Example 9 for a channel that produces one cluster of errors and independent errors with probability p. Reproduced by permission of the Institution of Engineering & Technology [86]

5.5 Conclusions

In this chapter, we have introduced the concept of partitioning of Gaussian integer modules $\mathcal{G}_{p^2}$. We have shown that Huber's original definition of the Mannheim metric satisfies the distance axioms over the considered class of Gaussian integers. Furthermore, we have derived upper bounds on the minimum distance for the set partitioning. Next we have presented two constructions for the set partitioning. Both constructions achieve optimum or close to optimum set partitions. In particular, we have demonstrated that perfect set partitions with respect to the sphere packing bound are attainable. This set partitioning enables multi-level code constructions over Gaussian integers. Furthermore, we have demonstrated that this set partitioning can be applied to an interleaving technique for correcting two-dimensional cyclic clusters of errors.

Chapter 6

Soft-Decoding for Generalized Concatenated Codes

In this chapter, we describe soft-decoding techniques for the GCC. This promises an increase of the error correction capability by considering more channel information using a quantized information of an FG as described in Section 2.2.1. In [21], the author stated that it is in principle possible to extend the inner algebraic decoder of the GCC decoder by a more complex decoding algorithm employing soft-decoding. The soft-decoding algorithms lead to increased complexity. For instance, the trellis representation requires for a block code $O(2^{min[Rn,(1-R)n]})$ states and thus the memory complexity growths exponentially with the error correction capability. Nevertheless, in the literature exists a large variety of soft-decoding algorithms. In this chapter the stack algorithm and Chase decoding algorithm are presented and investigated for their application in GCC and Flash memories. For the stack algorithm, or also called sequential decoder, we propose a less complex supercode decoding algorithm and show its relation to the subcodes of the nested BCH codes.

The novelties in this chapter are:

- Sequential stack decoding of nested BCH codes using supercodes [96].
- Applying different soft-decoding algorithms to decode the inner block codes in the GCC.

6.1 Sequential Stack Decoding

In this section, we describe sequential decoding procedures using the stack algorithm for block codes. These decoding methods are used to decode the binary inner codes. In Section 6.1.2 the supercode decoding is introduced that reduces the trellis and thus the decoding complexity.

6.1.1 Sequential Stack Decoding using a Single Trellis

First, we consider the sequential decoding procedure as presented in [2]. All algorithms considered in this section are based on this decoding method which uses a trellis to represent the code. Later on, we will present improvements to this decoding algorithm.

The stack decoding procedure uses the trellis representation. A trellis $\mathcal{T} = (\mathcal{S}, \mathcal{W})$ is a labeled, directed graph, where $\mathcal{W} = \{w\}$ denotes the set of all branches in the graph and $\mathcal{S} = \{\sigma\}$ is the set of all nodes. The set $\mathcal{S}$ is decomposed into $n + 1$ disjoint subsets $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_n$ that are called levels of the trellis. Similarly, there exists a partition

of the set $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2 \cup \ldots \cup \mathcal{W}_n$. A node $\sigma \in \mathcal{S}_t$ of the level t may be connected with a node $\widetilde{\sigma} \in \mathcal{S}_{t+1}$ of the level $t+1$ by one or several branches. Each branch w_t is directed from a node σ of level $t-1$ to a node $\widetilde{\sigma}$ of the next level t. We assume that the end levels have only one node, namely $\mathcal{S}_0 = \{\sigma_0\}$ and $\mathcal{S}_n = \{\sigma_n\}$. A trellis is a compact method of presenting all codewords of a code. Each branch of the trellis w_t is labeled by a code symbol $v_t(w_t)$. Each distinct codeword corresponds to a distinct path in the trellis, i.e. there is a one-to-one correspondence between each codeword $\mathbf{v}$ in the code and a path $\mathbf{w}$ in the trellis: $\mathbf{v}(\mathbf{w}) = v_1(w_1), \ldots, v_n(w_n)$. We denote code sequence segments and path segments by $\mathbf{v}_{[i,j]} = v_i, \ldots, v_j$ and $\mathbf{w}_{[i,j]} = w_i, \ldots, w_j$, respectively. The *syndrome trellis*, can be obtained using its parity-check matrix [70]. The syndrome trellis is minimal inasmuch as this trellis has the minimal possible number of nodes $|\mathcal{S}|$ among all possible trellis representations of the same code.

The sequential decoding procedure as presented in [2] is a stack algorithm, i.e. a stack is required to store interim results. The stack contains code sequences of different lengths. Let $\mathbf{v}_t$ denote a code sequence of length t, i.e. $\mathbf{v}_t = v_1, \ldots, v_t$. Each code sequence is associated with a metric and a node σ_t. The node σ_t is the node in the trellis that is reached if we follow the path corresponding to the code sequence through the trellis. The metric rates each code sequence and the stack is ordered according to these metric values where the code sequence at the top of the stack is the one with the largest metric value. There exist different metrics in the literature to compare code sequences of different length. In the following, we consider the Fano metric which is defined as follows. Let v_i be the i-th code bit and r_i the i-th received symbol for transmission over a discrete memoryless channel. The Fano metric for a code bit v_i is defined by

$$M(r_i|v_i) = \log_2 \frac{p(r_i|v_i)}{p(r_i)} - B \tag{6.1}$$

where $p(r_i|v_i)$ is the channel transition probability and $p(r_i)$ is the probability to observe r_i at the channel output. The term B is a bias term that is typically chosen to be the code rate R [49]. The Fano metric of a code sequence $\mathbf{v}_t$ is

$$M(\mathbf{r}_t|\mathbf{v}_t) = \sum_{i=1}^{t} M(r_i|v_i) \tag{6.2}$$

where $\mathbf{r}_t$ is the sequence of the first t received symbols. Note that the Fano metric according to Equation (6.1) is only defined for discrete memoryless channels (DMC). We consider the quantized AWGN channel which is a DMC. Binary block codes typically have no tree structure. Consequently, the Fano metric is not necessarily the best metric for all binary block codes. For instance, in [62] a metric with variable bias term was proposed for linear block codes. However, in our simulations for binary BCH codes we found that $B = R$ provides good results for all considered channel conditions.

We demonstrate Algorithm 2 in the following example, where for simplicity we assume transmission over a binary symmetrical channel.

Example 10. Consider for instance the code $\mathcal{B} = \{(0000), (1110), (1011), (0101)\}$ with parity-check matrix

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

The corresponding trellis is depicted in Figure 6.1a). We assume transmission over a binary symmetrical channel with error probability 0.1. Hence, we have

$$M(r_i|v_i) \approx \begin{cases} 0.3 & \text{for } r_i = v_i \\ -2.8 & \text{for } r_i \neq v_i \end{cases}$$

Algorithm 2: Sequential stack decoding using a single trellis.

Data: received word $\mathbf{r}$

Result: estimated codeword $\hat{\mathbf{v}}$

sequential decoding starts in the first node σ_0 of the trellis;
calculate the metric values for $v_1 = 0$ and $v_1 = 1$;
insert both paths into the stack according to their metric values;
while *the top path has not approached the end node* σ_n **do**
  remove the code sequence $\mathbf{v}_t$ at the top from the stack;
  if *the branch* $v_{t+1} = 0$ *exists in the trellis for the node* σ_t *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 0)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 0)$ into the stack;
  if *the branch* $v_{t+1} = 1$ *exists in the trellis for the node* σ_t *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 1)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 1)$ into the stack;

return the codeword $\hat{\mathbf{v}}$ corresponding to the top path in the final iteration;

The following tables represent the stack for the received sequence $\mathbf{r} = (0010)$.

1st iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0	0.3
1	-2.8

2nd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
00	0.6
01	-2.5
1	-2.8

3rd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
000	-2.2
01	-2.5
1	-2.8

4th iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0000	-1.9
01	-2.5
1	-2.8

◇

6.1.2 Supercode Decoding for nested BCH Codes

For codes with larger redundancy, the trellis that has to be stored in a memory exploding exponentially. To overcome this, we first describe the supercode decoding in this section. Subsequently, we discuss the proposed application of supercode decoding for the nested BCH codes that are used in the GCC code. A supercode is a superset $\mathcal{B}_1$ of the original code $\mathcal{B} \subset \mathcal{B}_1$. In order to decode the original code $\mathcal{B}$, two supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$ have to be constructed such that $\mathcal{B}_1 \cap \mathcal{B}_2 = \mathcal{B}$. The supercodes have fewer redundancy bits and thus fewer trellis states. The supercodes can be constructed such that each code has half of the original redundancy bits. This reduces the number of states from $O(2^r)$ to $O(2^{\frac{r}{2}})$ in standard order notation, where r is the number of parity bits. The concept of supercode decoding is well suited for decoding of GCCs, because the higher levels of the nested BCH codes are supercodes of the lower levels (cf. Equation (4.1)).

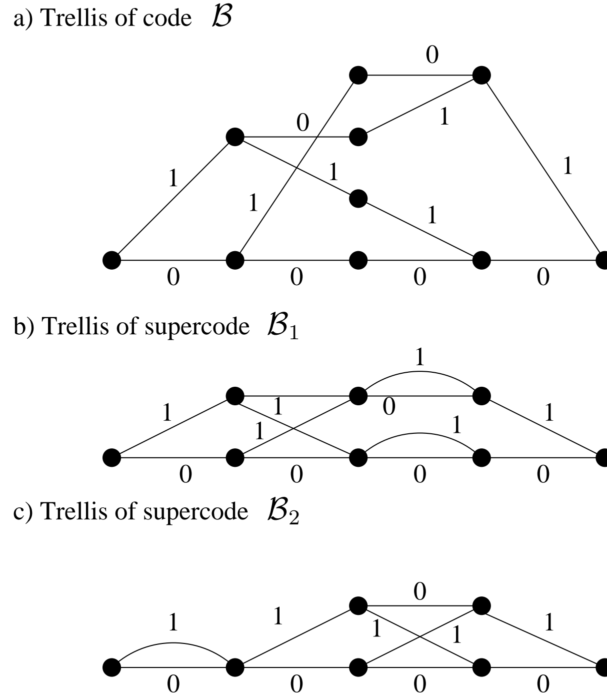


Fig. 6.1: Trellises of the example code and of its two supercodes. Trellis a) is a representation of the complete code, whereas the trellises b) and c) are the representations of the supercodes.

A *supercode* $\mathcal{B}_i$ of the block code $\mathcal{B}$ is a code containing all codewords of $\mathcal{B}$. For a linear code $\mathcal{B}$ with parity-check matrix $\mathbf{H}$, we can construct two supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B} = \mathcal{B}_1 \cap \mathcal{B}_2$. Let $\mathbf{H} = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{pmatrix}$ be the parity-check matrix of the code $\mathcal{B}$, this means that $\mathbf{H}_1$ and $\mathbf{H}_2$ are two sub-matrices of $\mathbf{H}$. Then, the sub-matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ define the supercodes $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.

Example 11. Consider the code $\mathcal{B}$ from Example 10. We obtain

$$\begin{aligned} \mathbf{H}_1 &= \begin{pmatrix} 1 & 1 & 0 & 1 \end{pmatrix} \\ &\Downarrow \\ \mathcal{B}_1 &= \begin{array}{l} \{\underline{(0000)}, (1100), \underline{(1110)}, (0010), \\ \underline{(1011)}, (1001), \underline{(1011)}, \underline{(0101)}\} \end{array} \end{aligned}$$

and

$$\begin{aligned} \mathbf{H}_2 &= \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix} \\ &\Downarrow \\ \mathcal{B}_2 &= \begin{array}{l} \{\underline{(0000)}, (1000), (0110), \underline{(1110)}, \\ \underline{(1011)}, (1101), (0011), \underline{(0101)}\} \end{array} \end{aligned},$$

where the underlined vectors are the codewords of the code $\mathcal{B}$. The corresponding supercode trellises are depicted in Figure 6.1b) and 6.1c).

◇

Next we state the proposed sequential decoding algorithm. Any path stored in the stack is associated with a metric value as well as two states $\sigma_{t,1}$ and $\sigma_{t,2}$ which are the states in the trellis for supercode $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.

Algorithm 3: Sequential stack decoding using supercode trellises.

Data: received word $\mathbf{r}$
Result: estimated codeword $\hat{\mathbf{v}}$
sequential decoding starts in the nodes $\sigma_{0,1}$ and $\sigma_{0,2}$ of the supercode trellises;
calculate the metric values for $v_1 = 0$ and $v_1 = 1$;
insert both paths into the stack according to their metric values;
while *the top path has not approached the end nodes* $\sigma_{n,1}$ *and* $\sigma_{n,2}$ **do**
  remove the code sequence $\mathbf{v}_t$ at the top from the stack;
  if *the branch* $v_{t+1} = 0$ *exists in the trellis for both nodes* $\sigma_{t,1}$ *and* $\sigma_{t,2}$ *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 0)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 0)$ into the stack;
  if *the branch* $v_{t+1} = 1$ *exists in the trellis for both nodes* $\sigma_{t,1}$ *and* $\sigma_{t,2}$ *corresponding to the top path* $\mathbf{v}_t$ **then**
    calculate the metric $M(\mathbf{r}_{t+1}|\mathbf{v}_{t+1}) = M(\mathbf{r}_t|\mathbf{v}_t) + M(r_{t+1}|v_{t+1} = 1)$;
    insert the code sequence $\mathbf{v}_{t+1} = (\mathbf{v}_t, 1)$ into the stack;
return the codeword $\hat{\mathbf{v}}$ corresponding to the top path in the final iteration;

We demonstrate decoding Algorithm 3 in the following example, where we consider the same setup as in Example 10.

Example 12. The following tables represent the stack for the received sequence $\mathbf{r} = (0010)$ for the proposed algorithm.

1st iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0	0.3
1	-2.8

2nd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
00	0.6
01	-2.5
1	-2.8

3rd iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
001	0.9
000	-2.2
01	-2.5
1	-2.8

4th iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
000	-2.2
01	-2.5
1	-2.8

5th iteration

$\mathbf{v}_t$	$M(\mathbf{r}_t\|\mathbf{v}_t)$
0000	-1.9
01	-2.5
1	-2.8

Note that the stack in the third iteration differs from Example 10, because the code sequence 001 exists in both supercode trellises but not in the actual code. This code sequence is deleted in the next iteration, because it cannot be extended in both supercode trellises. ◇

As the previous example demonstrates, the time complexity of the proposed algorithm may be larger than with Algorithm 2. This results from code sequences that exist in the super codes, but are not valid in the actual code. Nevertheless, both algorithms result in the same codeword.

Theorem 7. *Algorithm 2 and Algorithm 3 result in the same estimated codeword.*

Proof. Both algorithms differ only with respect to the representation of the code. To prove the proposition it is sufficient to verify that both representations are equivalent. We first prove by induction that the estimated codeword corresponds to a valid path in both supercode trellises, i.e. it is a codeword in both supercodes. The base case is the initial step where the code bits 0 and 1 are inserted in the stack. Note that a linear code has no code bit positions with constant values. Hence, the transitions $v_1 = 0$ and $v_1 = 1$ exist in both supercode trellises. For the inductive step, we assume that a path for the code sequence $\mathbf{v}_t$ exists in both supercode trellises. It follows from Algorithm 2 that this path is only extended if the extended path exists in both supercode trellises. This proves the claim that the estimated codeword corresponds to a valid path in both supercode trellises. Now note that $\mathcal{B} = \mathcal{B}_1 \cap \mathcal{B}_2$, i.e. a path is only valid in both supercode trellises if and only if it is a valid codeword of the code $\mathcal{B}$. □

Algorithm 3 reduces the space complexity required for representing the code. We demonstrate this in the following example.

Example 13. We consider three BCH codes from Table 4.1. All codes have length $n = 60$. In the first level, we use a single-error correcting code. This code has 3,262 nodes in the trellis and is a supercode of the BCH code of the second level. The trellis of the second level has 159,742 nodes. However, utilizing the trellis of the first level code, we require only a single additional supercode trellis with 2,884 nodes to represent the code at the second level. Finally, the code at the third level has a trellis with 7,079,886 nodes. Using supercode decoding, we utilize the trellises of the first and second level and require one additional supercode trellis with 2,410 nodes to represent the third code.

With sequential decoding the number of visited nodes in the trellis (the number of iterations) depends on the number of transmission errors. Note that with the presented codes the time complexity with Algorithm 3 is at most 1.75 times larger than with Algorithm 2. ◇

6.1.3 List-of-two Decoding

Next, we present two techniques to improve the performance and the complexity of Algorithm 2. First, we demonstrate that the soft-input decoding can be omitted in cases where the hard decision of the received vector corresponds to a valid codeword. Moreover, we propose a sequential list-of-two decoding algorithm. List-of-two decoding is motivated by the fact that Algorithm 2 is not a maximum-likelihood decoding procedure. Hence, we may search for further codewords to find better candidates than the result of Algorithm 2.

Consider an additive white Gaussian noise channel with binary phase shift keying. A binary code symbol $v_t \in \mathbb{F}_2$ is mapped to the transmission symbol $x_t \in \{+1, -1\}$ by $x_t = 1 - 2v_t$. The transmitted symbol vector $\mathbf{x}$ is distorted by a noise vector $\mathbf{n}$ such that the received sequence is $\mathbf{r} = \mathbf{x} + \mathbf{n}$. The noise vector $\mathbf{n}$ is a vector of independent identically distributed Gaussian random variables with mean zero. Hence,

$$p(r_t \mid x_t = \pm 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(r_t \mp 1)^2}{2\sigma^2}} \quad , \tag{6.3}$$

where σ^2 denotes the variance of the Gaussian distribution. For this channel, it is common practice to use the quadratic Euclidean distance $d_E^2(\mathbf{x}, \mathbf{r}) = \sum_{t=1}^{n} |x_t - r_t|^2$ as metric, because

$$\arg\left(\max_{\mathbf{v} \in \mathcal{C}} P(\mathbf{r}|\mathbf{v})\right) = \arg\left(\min_{\mathbf{v} \in \mathcal{C}} d_E^2(\mathbf{x}, \mathbf{r})\right) \quad . \tag{6.4}$$

However, we have

$$d_E^2(\mathbf{x},\mathbf{r}) = \sum_{t=1}^{n} x_t^2 - 2\sum_{t=1}^{n} x_t r_t + \sum_{t=1}^{n} r_t^2 \tag{6.5}$$

Let $\tilde{r}_t = \operatorname{sgn}(r_t)$ denote the sign, i.e. the hard decision, of r_t. Using

$$\sum_{t=1}^{n} x_t r_t = \sum_{t=1}^{n} |r_t| - 2 \sum_{t:\ x_t \neq \tilde{r}_t} |r_t| \tag{6.6}$$

we obtain

$$d_E^2(\mathbf{x},\mathbf{r}) = n + 4 \sum_{t:\ x_t \neq \tilde{r}_t} |r_t| - 2\sum_{t=1}^{n} |r_t| + \sum_{t=1}^{n} r_t^2 \tag{6.7}$$

Note that $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t|$ is the only term in (6.7) that depends on $\mathbf{x}$. Consequently, instead of minimizing the quadratic Euclidean distance we may also minimize $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t|$. Note that $\sum_{t:\ x_t \neq \tilde{r}_t} |r_t| = 0$ if the vector $\tilde{\mathbf{r}} = (\tilde{r}_1, \ldots, \tilde{r}_n)$ corresponds to a valid codeword. Hence, in this case, $\tilde{\mathbf{r}}$ is the maximum-likelihood estimate.

Now we consider list-of-two decoding. In order to enable a trade-off between performance and complexity, we introduce a threshold ρ for the metric of the estimated codeword.

Algorithm 4: Sequential list-of-two decoding.

Data: received word $\mathbf{r}$, threshold ρ
Result: estimated codeword $\hat{\mathbf{v}}$
if $\tilde{\mathbf{r}}$ *corresponds to a valid codeword* **then**
  return the codeword $\hat{\mathbf{v}}$ corresponding to $\tilde{\mathbf{r}}$;
else
  calculate a first estimate $\mathbf{v}_1$ using either Algorithm 2 or Algorithm 3;
  if $M(\mathbf{r},\mathbf{v}_1) \geq \rho$ **then**
    return the codeword $\hat{\mathbf{v}} = \mathbf{v}_1$;
  else
    remove $\mathbf{v}_1$ from the stack;
    calculate a second estimate $\mathbf{v}_2$ using either Algorithm 2 or Algorithm 3;
    if $M(\mathbf{r},\mathbf{v}_1) \geq M(\mathbf{r},\mathbf{v}_2)$ **then**
      return $\hat{\mathbf{v}} = \mathbf{v}_1$;
    else
      return $\hat{\mathbf{v}} = \mathbf{v}_2$;

In Algorithm 4 we apply Algorithm 2 to decode the inner codes at the first level, i.e. the codewords of the code $\mathcal{B}^{(0)}$, whereas we apply Algorithm 3 for the lower levels. Figure 6.2 presents the performance of Algorithm 2 and Algorithm 4 with respect to the residual word error rate (WER) for transmission over the AWGN channel. The code is a one error-correcting binary BCH code of length $n = 60$. This code is used later on as inner code in the first level of the GCC. The decoding performance and the number of decoding iterations depend on the threshold ρ. Figure 6.3 presents a comparison with respect to the number of iterations. The value of ρ were chosen to demonstrate that Algorithm 4 can reduce the word error rate compared with Algorithm 2 with a similar complexity.

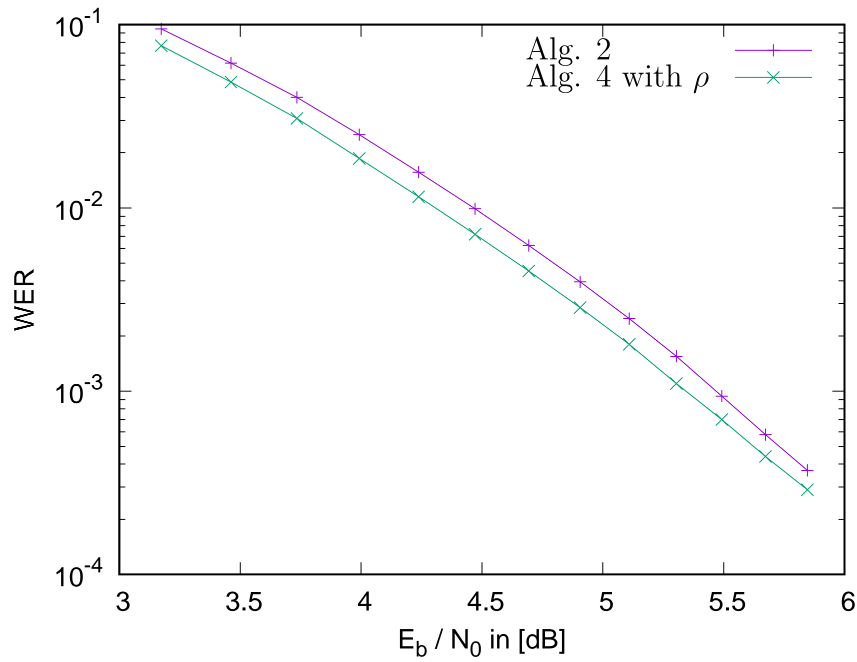


Fig. 6.2: Comparison of Algorithm 2 and Algorithm 4 with respect to the residual word error rate (WER).

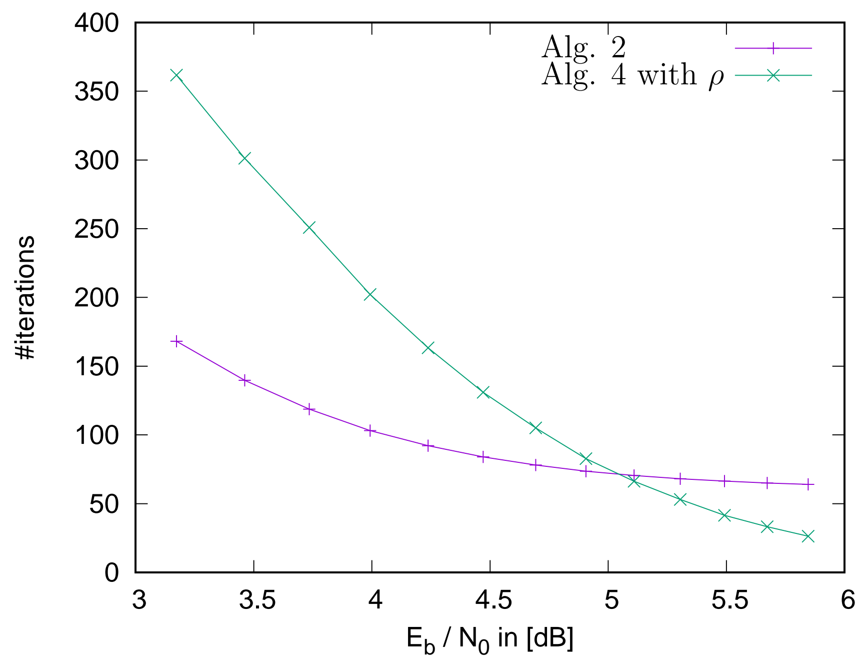


Fig. 6.3: Comparison of Algorithm 2 and Algorithm 4 with respect to the number of iterations.

6.2 GCC Decoding and Decoding Error Probability

In contrast to the decoding procedure presented in Section 4.1.2, the algebraic inner code decoder is substituted by a soft decoder and thus different column error probabilities $P_{b,i}$ must be considered.

We consider the code level distances $d_{a,i}$ of the outer code from Example 3. The lowest level needs the highest distance to reach the required WER. While the level increases, the higher obtain the inner code distance $d_{b,i}$. Thus, the outer code error correction capability decreases. Applying soft-decoding in the lower levels starting from $i = 0$ leads to a higher improvement in decoding complexity and code rate. However, with increasing codeword distance of the inner code in the level $i = optimal_complexity$ gets, the complexity increases for the stack algorithm by $O(r^2)$ and its decoding improvements does not lead to a significant complexity reduction in the outer code, which has a complexity of at least $O(2^n)$. In this section, we focus on the stack decoding algorithm as inner decoder. The next Section 6.3 discusses alternative soft-decoding algorithms.

In the first level $i = 0$ we use soft-input decoding according to Algorithm 2. Starting with the second level, we exploit the structure of the nested BCH codes and use Algorithm 3, where the code at level $i-1$ can be used as supercode of the code of level i. For the implementation, we limit the number of decoding iterations for each inner code. If the number of iterations exceeds a threshold a decoding failure is declared. For the outer RS codes we employ error and erasure decoding [69], where the decoding failures of the inner codes are regarded as erased symbols of the RS code.

In the following, we present an analysis of the probability of a decoding error for the GCC decoder. Subsequently, we present an example that illustrates the performance of the proposed decoding procedure.

6.2.1 Probability of a Decoding Error

The performance of the soft-input decoding of the inner codes can be determined using Monte Carlo simulation. Let $P_{b,i}$ be the error probability for the decoding of the inner code $\mathcal{B}^{(i)}$. Furthermore, let $\lambda_{b,i}$ be the corresponding probability of a decoder failure. We can calculate the WER using the formulas in Section 4.2.

Example 14. *Consider the code from Example 3. This code has a code rate* $R = 0.806$ *and was designed to guarantee* $WER \leq 10^{-16}$ *according to (4.10) for* $\frac{E_b}{N_0} \geq 4.7dB$*, where soft-input decoding is used in the first three levels and hard-input decoding in the remaining levels.*

6.2.2 Comparison Error Correction Performance

We compare the error correction performance of the GCC in different decoding modes with the performance of long BCH codes with hard-input decoding. As performance measure, we use the code rate that is required to guarantee for a given signal to noise ratio an overall word error rate less than 10^{-10} or 10^{-16}, respectively. All codes are constructed similar to the code presented in Example 3. In particular, the inner codes are chosen according to Table 4.1. Whereas the error-correcting capability of the outer codes are adapted to obtain the highest possible code rate for a given signal to noise ratio. Note that in this example, the overall code rate of the GCC is at most $R = 0.9$, due to the choice of the inner code.

Figure 6.4 depicts the code rate versus the signal to noise ratio for $FER = 10^{-10}$, whereas the results for $FER = 10^{-16}$ are presented in Figure 6.5. The GCC with soft-input decoding

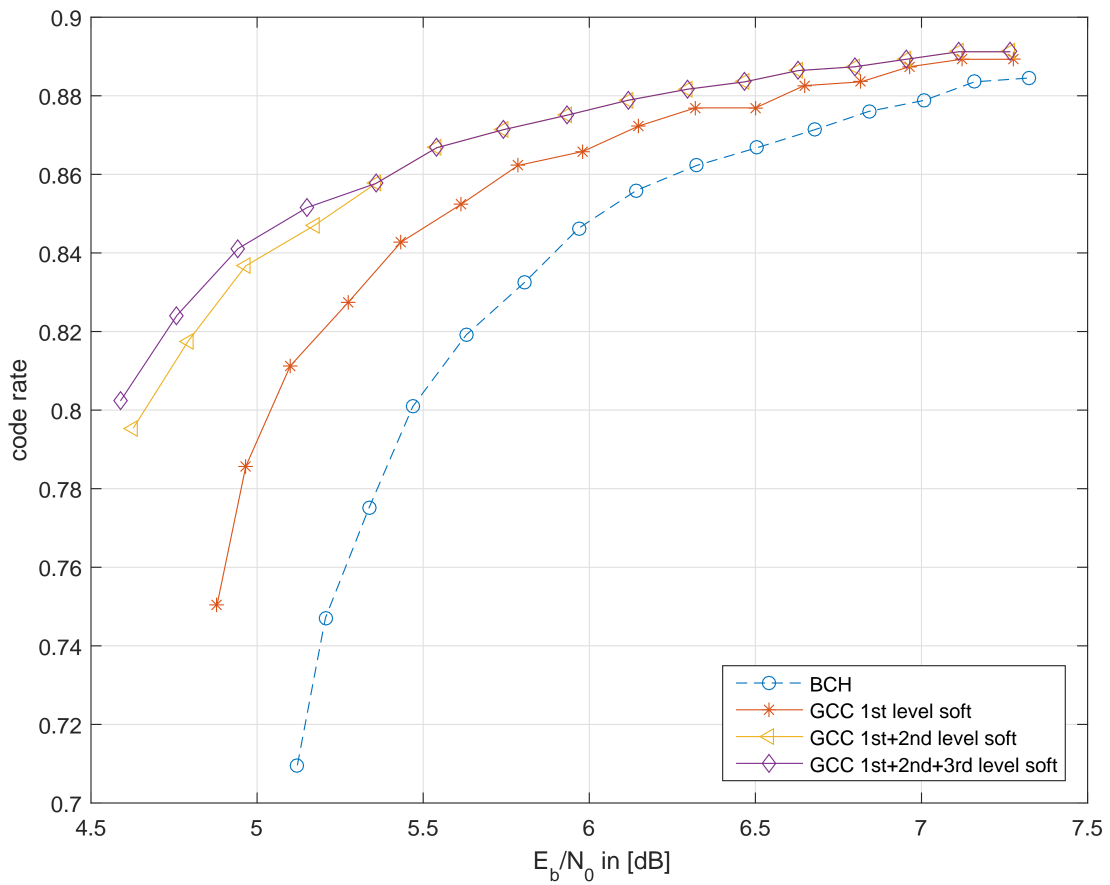


Fig. 6.4: Code rate versus signal to noise ratio for $P_e = 10^{-10}$.

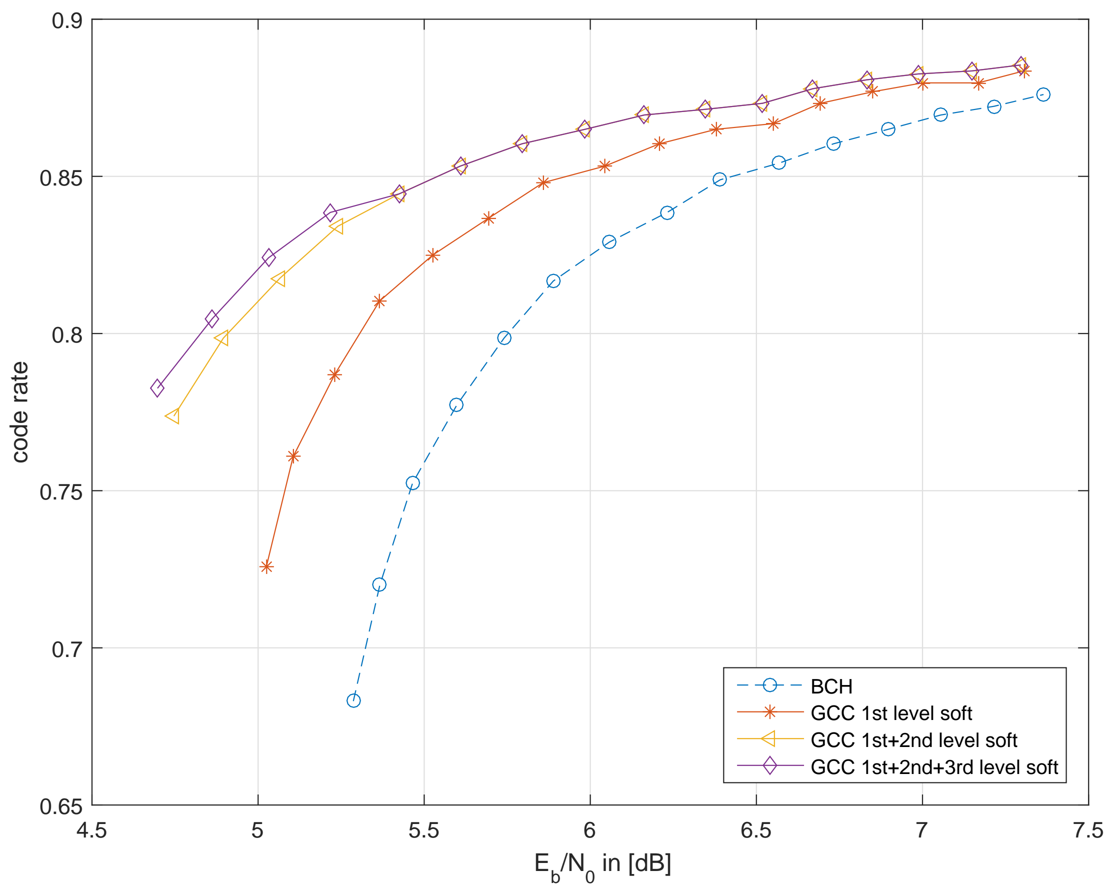


Fig. 6.5: Code rate versus signal to noise ratio for $P_e = 10^{-16}$.

outperforms the GCC with hard-input decoding for all error probabilities and the BCH code for code rates below 0.88. The soft-input decoding was simulated with a three-bit quantization. The three curves with soft-input decoding use different decoding strategies, where the soft-input decoding is applied only to the first level, first and the second level, or levels 0 to 2, respectively. The soft-input decoding improves the performance by up to 1.3 dB. For instance, the GCC with code rate $R = 0.8$ achieves a block error rate less than 10^{-16} at a signal to noise ratio of $E_b/N_0 = 4.7$ dB which is only 1 dB from the channel capacity of the quantized AWGN channel. For $P_e = 10^{-10}$, the code rate $R = 0.8$ is sufficient for a signal to noise ratio $E_b/N_0 \geq 4.6$ dB. Note that soft-input decoding of the first and second level is sufficient for all SNR values above $E_b/N_0 = 5.5dB$.

6.3 Other Soft Decoders

The complexity of the stack decoding algorithm is very high and thus not suitable for large inner codes n_b. In literature, several soft-decoding algorithms exist. A alternative is the Chase decoder, which is described and investigated in this section. A special case of the Chase decoder is the SPC.

The Chase decoding algorithm in [11] is a reliability based decoding procedure that generates a list of candidate codewords by flipping bits $\mathcal{T}$ in the received word $\mathbf{r}$. The test patterns for the bit flipping are based on the least reliable positions of the received word. For each test pattern v_i, algebraic hard-input decoding Ψ is employed. Finally, the resulting codeword candidate $\hat{x}$ from the list is obtained by minimizing the Euclidean distance between the candidate codewords and the received word. Chase devised three different algorithms. The main difference between the algorithms is the number of test patterns. The complexity of Chase decoding depends on the number of candidates.

Algorithm 5: Chase-II decoder [6]

```
begin
    L ← {∅}
    T ← generate test vectors(r,⌊(d − 1)/2⌋)
    i ← 0
    while i < |T| do
        x̂, failure ← decode(r ⊕ v_i ∈ T) with Ψ
        if !failure then
            if ⟨r, y⟩ > max then
                max ← ⟨r, y⟩
                L = {x̂}
        i ← i + 1
    return L
```

Figure 6.6 shows the WER and erasure rate for a BCH code $B(n = 109, k = 101, d = 4)$. There is a coding gain between hard information only BMD and the continuous (cont.) channel. The Flash memory quantization (quant.) shows a relatively small loss regarding the continuous channel.

The SPC code uses one additional bit to keep the codeword weight even. In the case of hard decoding, the SPC decoder is only able to detect errors by checking if the weight is

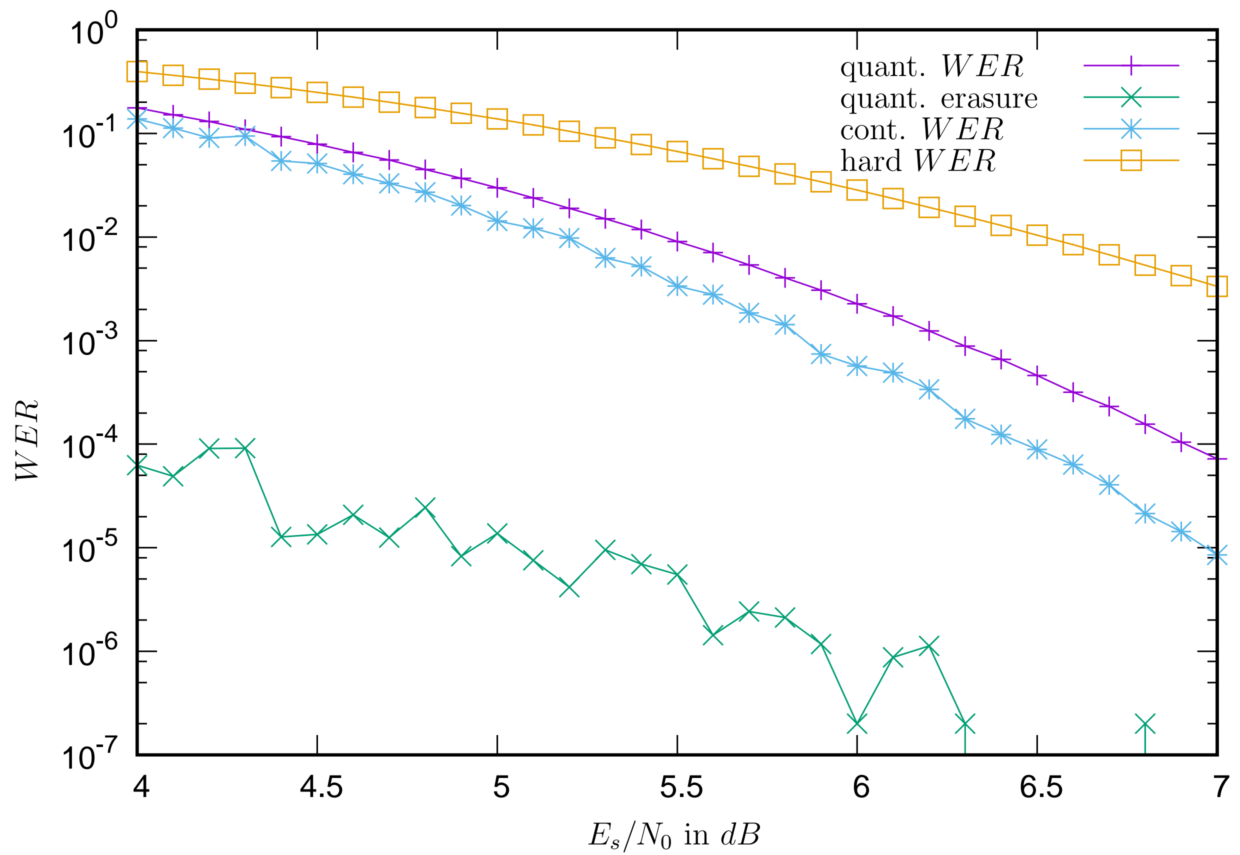


Fig. 6.6: Chase2 error curve.

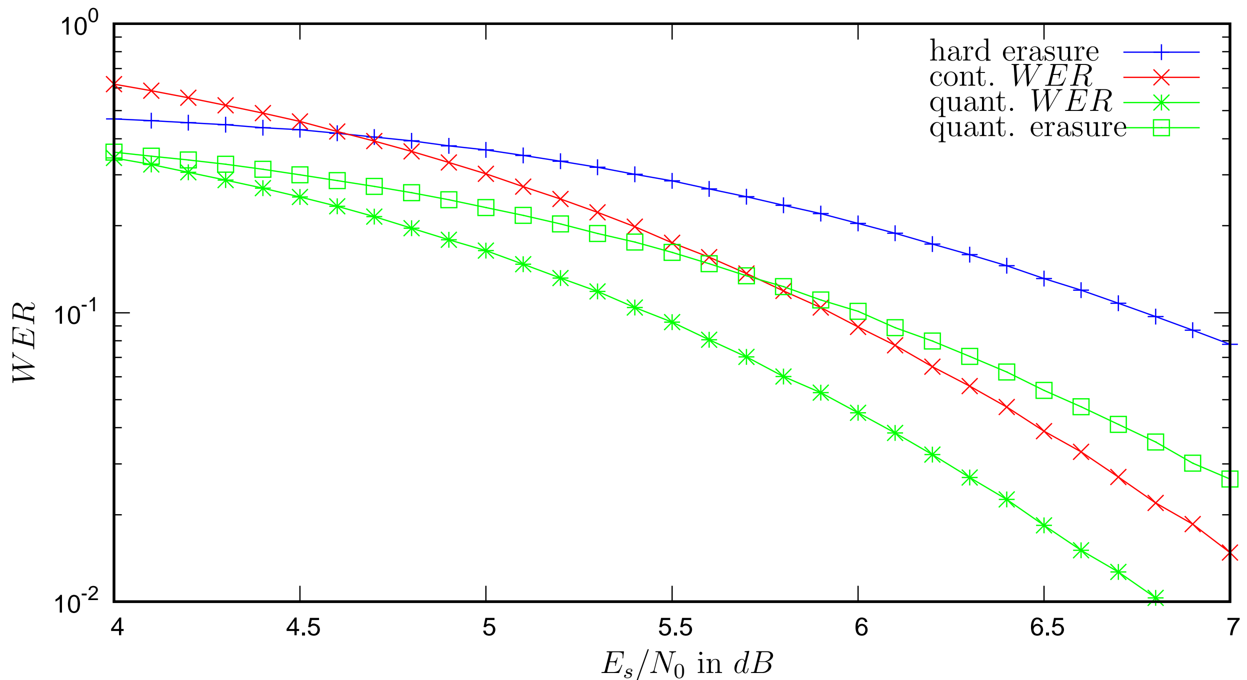


Fig. 6.7: Chase2 error curve.

odd. In the case of soft-decoding, the SPC code can correct up to a single bit error using the bit-flipping algorithm. If the parity of the column is odd, the least reliable bit position in this column is flipped. In the context of Flash memories, this bit position corresponds to the LLR value with the smallest magnitude. If the parity is odd and the least reliable bit position is not unique, which can occur frequently in the quantized channel, a decoding failure is signaled to the outer decoder. This code is a maximum-likelihood (ML) decoder.

Figure 6.7 shows the hard, quantized and continuous decoding cases. The hard decoding case can only detect errors and has in comparison to the other with soft-decoding the worst error correction performance. In the continuous case the decoder cannot detect errors because it is quasi impossible that two equal values occur.

6.4 Summary

In this chapter, we have presented several soft-decoding algorithms for the inner decoder of the GCC. Soft-decoding promises an improvement of error correction capability. We also showed how the inner codes of the GCC can be decoded using supercode decoder and thus a much smaller complexity can be achieved. Additional soft-decoding algorithms like Chase-II and its special case, the SPC bit-flipping algorithm were also investigated. It is possible to simulate these soft-decoders performances of the inner codes and obtain an overall FER as discussed in 4.2, which is essential. Soft-decoding is highly interesting for the lower GCC levels because they have a low complexity and their error correction improvement is relatively high in comparison to the BMD opponent.

A remaining open question is the influence of quantization for these decoding algorithms. Besides the quantization issue, the Flash memory channel has asymmetries that must be investigated. Furthermore, a proper quantization threshold has a influence on the overall error correction performance.

Chapter 7

Implementation of an Algebraic BCH Decoder

In this chapter, we investigate improvements of the common algebraic BCH decoder implementation. This single-block BCH code is the state-of-the-art ECC of nowadays Flash memory controllers, because it has good performance in correcting independent errors. An issue of this code is, that the decoder complexity increases at least quadratically with the error correction capability. There are two significant parameters that lead to this increased implementation complexity. First the degree m of the $GF(2^m)$ increases to cover the entire data block in comparison to the component codes of the GCC which are shorter. Besides of the degree m the choice of the primitive polynomial has a significant impact on the implementation complexity. In section 7.2 the impact of the primitive polynomial is investigated and an example is given. Second, the error correction capability leads to a higher degree of the error-location polynomial $\sigma(x)$, which is determined by the BMA. Several implementations are proposed in [57, 72]. Its hardware realization is scalable in throughput or logic. After the BMA has found a solution, the degree of the error-location polynomial is known. Thus, a multi degree processing Chien search can be constructed that processes the polynomial with different speeds and with that a trade-off between throughput versus die area can be realized. An approach to a mixed solution is presented in 7.1.

The novelties in this chapter are:

- A mixed parallel and serial BCH implementation design and its analysis. (Published in [82, 95])
- Investigations on Galois field optimizations and its application in mixed GF BCH decoder implementations.

7.1 A Mixed Parallel and Serial BCH Implementation

In this section, we provide an overview of a BCH implementation. It exists several modules that can be scaled differently. Because the decoding complexity grows quadratically with the error correction capability t, modifications in the state-of-the-art BMA and Chien search are discussed to trade-off between speed and die area.

Algebraic decoding of BCH codes consists of the following four steps, which are usually realized in modules in a hardware implementation:

1. Calculating the Syndrome
2. Calculation of the error-location polynomial (BMA)
3. Evaluation of the error-location polynomial (Chien search)
4. Correcting the erroneous positions

Steps 2 and 3 dominate the computational complexity and therefore the area that is required for a hardware implementation. Both decoding steps can be realized with serial and parallel implementations. A parallel implementation achieves larger throughput but also requires more area than a serial implementation.

7.1.1 Syndrome Computation

In practice the codeword is read word-wise, i.e. 8 or 16 bit per cycle. The decoder calculates the syndrome values in parallel for each syndrome S_i using Horner's rule, which calculates n codeword bits per iteration. Using Horner's rule means processing with the most significant bit or degree first.

In common the calculation of the syndrome $\mathbf{S}$ can be expressed as a matrix multiplication of the check matrix $\mathbf{H}$ and the received codeword $\mathbf{r}$, by

$$\mathbf{S} = \mathbf{H}\mathbf{r}\,. \tag{7.1}$$

In order to scale the syndrome calculation between logic complexity and throughput, the partial syndrome vector S' is determined by a reduced $k' \times i$ checkmatrix.

$$\mathbf{S}' = \begin{pmatrix} \alpha^0 & \alpha^1 & \dots & \alpha^{k'-1} \\ \alpha^0 & \alpha^2 & \dots & \alpha^{2(k'-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^0 & \alpha^i & \dots & \alpha^{i(k'-1)} \end{pmatrix} \cdot \begin{pmatrix} r_0 \\ \vdots \\ r_{k'-1} \end{pmatrix} \tag{7.2}$$

$\mathbf{S}'$ represents the fraction of the Horner's rule. This is then added to a feedback register, which is then multiplied by $\alpha^{ik'}$ where i is the index of the syndrome value and k' is the size of the portion of the entire codeword.

In polynomial form it can be expressed as

$$\begin{aligned} S_i(\alpha^i) = (\dots(r_{n-0}\alpha^0 + \dots + r_{n-k'-1}\alpha^{i(k'-1)})\alpha^{ik'} + \\ r_{n-k'} \dots r_{n-2k'-1}\alpha^{i(k'-1)})\alpha^{ik'} + \\ r_{n-2k'} \dots r_{n-3k'-1}\alpha^{i(k'-1)} + \dots)\alpha^{ik'} + \\ r'_k\alpha^0 + \dots r_0\alpha^{i(k'-1)} \end{aligned} \tag{7.3}$$

where the characteristic of the Horner scheme can be recognized.

Due to the symmetry of the binary BCH code the even indexed syndrome values are linear dependent on the odd indexed syndrome values. This property can be used to reduce the implementation complexity. All even indexed syndrome values can be calculated by the square or multiple square of an odd indexed syndrome value, when all odd indexed syndrome values are ready. This technique reduces the number of registers to store the intermediate syndrome values but leads to a higher latency in the case of multiple squaring, which affects only the last cycle and thus can be solved by a multi-cycle path. Alternatively, the even syndrome values S_{2i} can be calculated in the *i-th* iteration of the BMA with a single multiplier but additional multiplexers and thus area and timing are reduced.

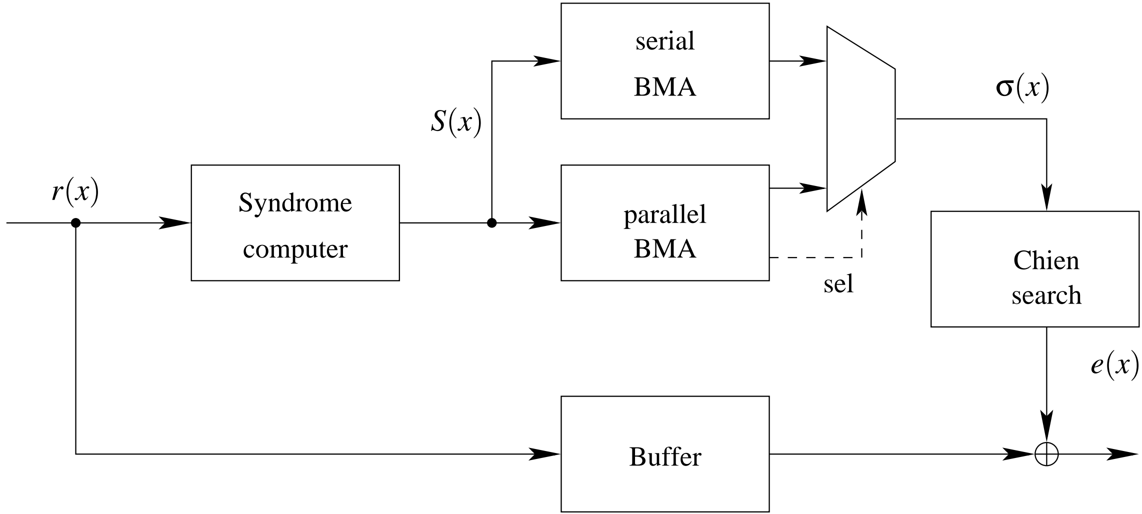


Fig. 7.1: BMA Serial/Parallel Decoder Scheme

7.1.2 Mixed Serial and Parallel BMA

Commonly, hardware implementations for BCH decoding employ the BMA using many GF multiplications. An overview of different parallel implementations is provided in reference [44], which also presents a comparison of the computational complexity of the different realizations. Accordingly, a parallel implementation for a t error-correcting code requires at least $2t$ multipliers and t iterations.

Serial implementations of the BMA require a significantly smaller number of multipliers, but increase the number of iterations [4, 10]. For instance, the serial implementation according to reference [10] requires only three multipliers, but t^2 iterations.

In the following we present a concept for a BMA implementation that enables a better trade-off between throughput and area. This concept requires both a parallel and a serial implementation of the BMA, but the parallel implementation is only used to correct up to $t_1 < t$ errors (see Figure 7.1). Hence, it requires only $2t_1$ multipliers and t_1 iterations. However, if the number of errors is larger than t_1, a serial implementation is used for error correction. A carefully chosen value of t_1 allows reducing the required area without a significant effect on the average throughput.

The error correction capability t determines the iteration number of a parallel or serial BMA and is therefore constant. However, in the mixed parallel and serial implementation the iteration number depends on the effective number of errors. Thus, the average number can be calculated. Let $p(t_1, \varepsilon)$ be the conditional probability of the number of errors is more than t_1 while at least one error has occurred. Hence, the conditional probability depends on t_1 and the bit error rate ε. Thus, the average iteration number $\overline{N}$ comprises the constant parallel part $t + t_1$ and the serial effort $2t^2$ that is weighted by the probability $p(t_1, \varepsilon)$.

$$\overline{N} = \left\lfloor \frac{t + t_1}{2} \right\rfloor + p(t_1, \varepsilon)t^2 \tag{7.4}$$

for binary BCH codes and

$$\overline{N} = t + t_1 + 2p(t_1, \varepsilon)t^2 \tag{7.5}$$

for non-binary codes. In case of BSC the probability that the serial decoder has to be deployed

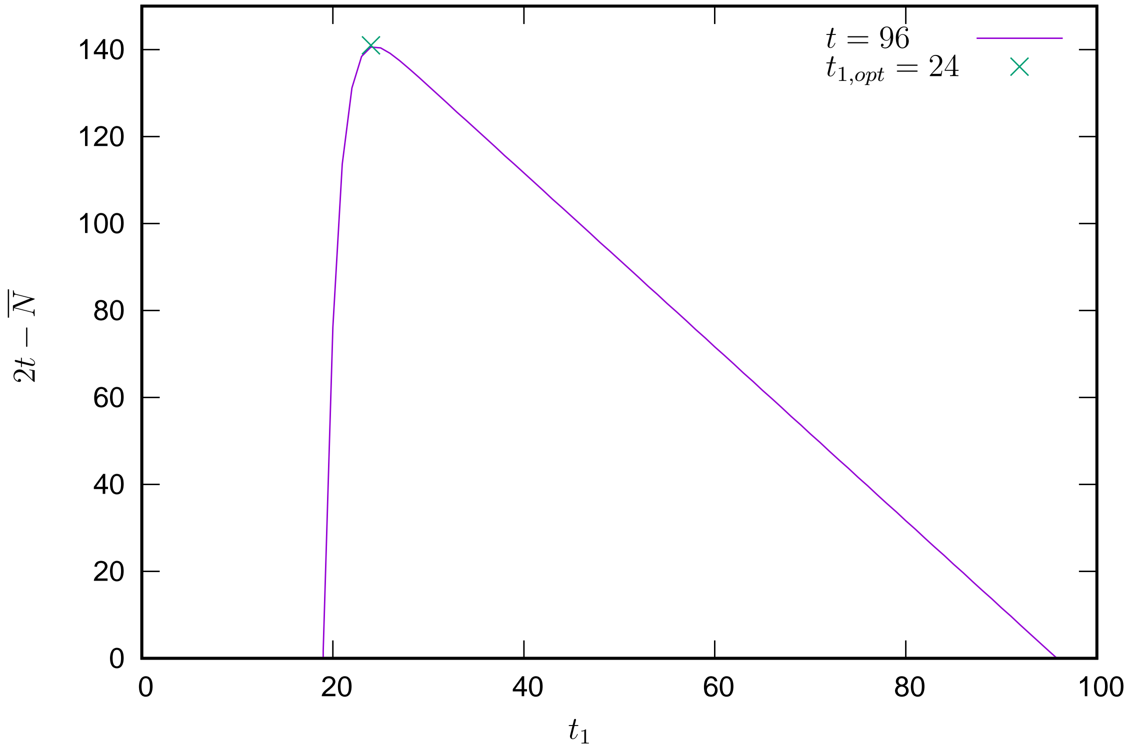


Fig. 7.2: $t - \overline{N}$ for parallel/serial BMA with $t = 96$

is $p(t_1, \varepsilon)$ which can be approximated by

$$p(t_1, \varepsilon) \approx \frac{\sum_{i=t_1+1}^{n} \binom{n}{i} \varepsilon^i (1-\varepsilon)^{n-i}}{1 - (1-\varepsilon)^n} \tag{7.6}$$

where the numerator term is the probability that the number of errors is larger than t_1 and the denominator term is the probability that at least one error occurs in a word of n bits. To guarantee that the average throughput is not reduced compared with a fully parallel implementation, $\overline{N} \leq 2t$ should be fulfilled.

Example 15. *We consider a typical Flash memory to determine the performance of a mixed parallel/serial BMA implementation. In order to support a 1-kByte sector size with additional meta information, a code with* $k = 8288$ *is required. Because* ε *increases during the life cycle of a Flash memory, we consider the highest tolerable bit error rate* $\varepsilon_t = 1.13 \cdot 10^{-3}$ *to achieve* $FER \leq 10^{-16}$. *Thus a* $t = 96$ *bit-error-correcting code is required and the codeword size is* $n = 9623$. *The value* $t_{1,opt} = 24$ *is then the optimal point to switch from the* t_1 *parallel processing BMA to the serial one. The improvement is depicted in Figure 7.2 which shows the reduction of the number of iterations* $t - \overline{N}$.

7.1.3 Multi Speed Chien Search

In [82] this concept of a mixed serial/parallel implementation was only applied to the BMA. However, this approach can also be applied to the following decoding step, i.e. the Chien search [12].

Chien search evaluates the error-location polynomial and determines its roots. The state-of-the-art implementation is similar to the syndrome calculation using the Horner's rule. Likewise, the correction of the bit errors is applied byte- or word-wise. In a parallel Chien search,

the errors for m positions can be checked in one cycle. Therefore the hardware has to evaluate m times

$$\sigma(\alpha^{-l}) = \sum_{i=0}^{t} \sigma_i \alpha^{-li} \tag{7.7}$$

This needs tm multiplications and $(t+1)m$ additions, where t is the error correction capability of the code. Here, a Chien search is proposed that uses two different parallelization degrees. A faster Chien search is used if the detected number of errors is less or equal t_1. This fast search calculates positions in parallel. Moreover, a slower calculation with error capability t and with parallelization degree $m < m_1$ is used if the number of errors is greater than t_1.

We illustrate this concept with a simple example with $t = 4$ and $t_1 = 2$. For instance, for a parallel implementation with parallelization degree $m_1 = 4$ the evaluation of $\sigma(\alpha^0)$, $\sigma(\alpha^{-1})$, $\sigma(\alpha^{-2})$, $\sigma(\alpha^{-3})$ has to be realized in hardware. For the next iteration, the coefficients of the error-location polynomial are multiplied by powers of α^{-4}, which results in the new polynomial

$$\tilde{\sigma}(x) = \sigma_0 + \sigma_1 \alpha^{-4} x + \sigma_2 \alpha^{-8} x^2 \,. \tag{7.8}$$

This polynomial is used as an error-location polynomial in the next iteration. Therefore, in the second iteration $\sigma(\alpha^{-4})$,$\sigma(\alpha^{-5})$,$\sigma(\alpha^{-6})$,$\sigma(\alpha^{-7})$ are evaluated. In slow mode with parallelization m degree, only two are evaluated. The basic concept is illustrated in Figure 7.3. The modification of the error-location polynomial for the next iteration is implemented for $m_1 = 4$. Hence the mode for $m = 2$ evaluates only one half of all possible positions. This is resolved by running the Chien search twice: First, with the initial error-location polynomial. In the second run with a modified error-location polynomial, where all coefficients are multiplied with powers of α^{-2}

$$\tilde{\sigma}(x) = \sigma_0 + \sigma_1 \alpha^{-2} x + \sigma_2 \alpha^{-4} x^2 + \sigma_3 \alpha^{-6} x^3 + \sigma_4 \alpha^{-8} x^4 \tag{7.9}$$

Therefore, the Chien search for $m = 2$ requires twice the number of iterations. Alternatively, a mode selection for the modification of the error-location polynomial can be implemented by multiplexing the required powers of α^{-m_1} or α^{-m}, respectively. Note that in this example the multiplications with α^{-3} and α^{-4} are used in both modes, but with different coefficients of the error-location polynomial. In order to utilize the corresponding multipliers in both modes, the coefficients can be selected by using multiplexers. The mode selection and the multiplexing of the coefficients are also illustrated in Figure 7.3.

Because the number of errors is already estimated by the BMA, the decoder can select the fast or slow Chien search based on the degree of the error-location polynomial. This mode selection is indicated in Figure 7.1. Therefore, the number of cycles depends on the actual number of errors. The Chien search requires $\frac{n}{m_1}$ iterations in fast mode and $\frac{n}{m}$ iterations in slow mode. Thus, we obtain the average number of iterations for the Chien search as

$$N = \frac{(1 - p(t_1, \varepsilon))n}{m_1} + \frac{p(t_1, \varepsilon)n}{m} \,. \tag{7.10}$$

Here ε denotes the bit error probability of the channel and $p(t_1, \varepsilon)$ the conditional probability that the number of errors is greater than given that at least one error occurred. This conditional probability can be approximated by Formula 7.6 [82].

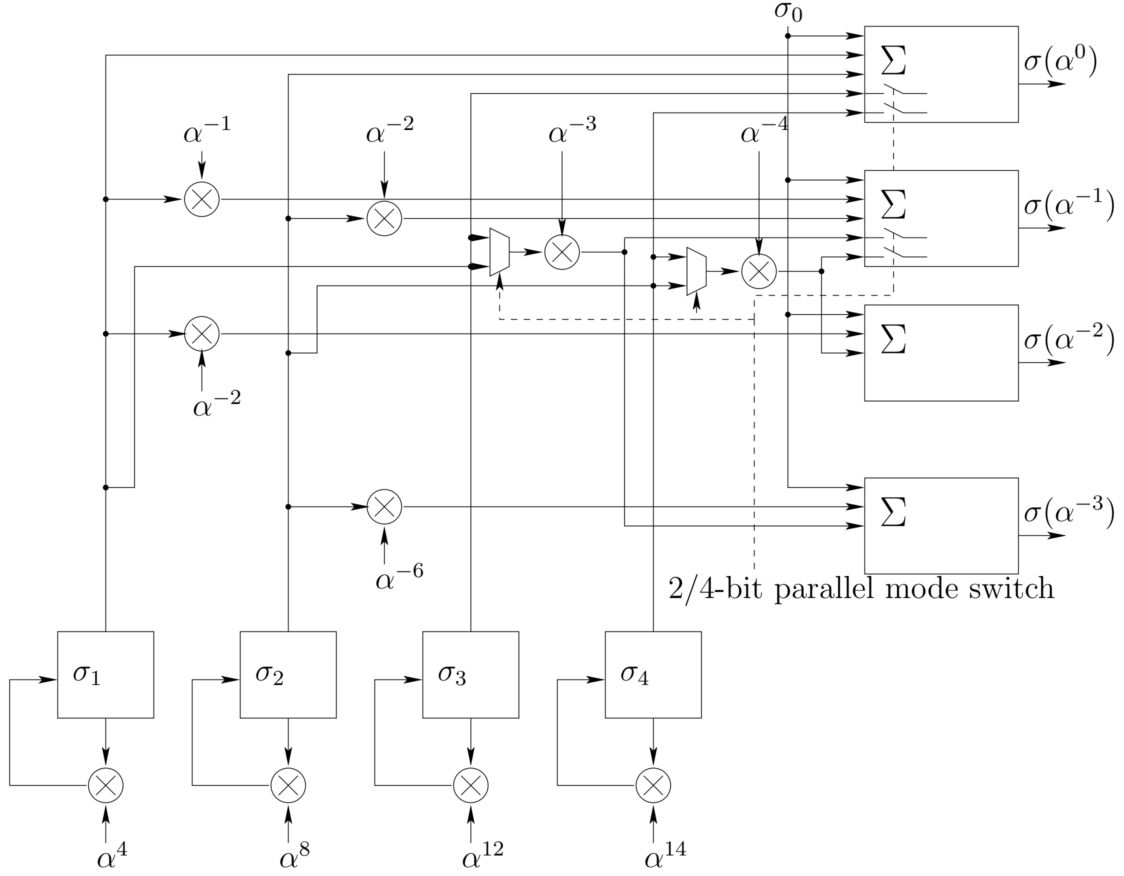


Fig. 7.3: Chien-search reused multipliers

7.1.4 Overall Considerations

The average number of iterations for both decoding steps (BMA and Chien search) is therefore

$$N = \lfloor \frac{t+t_1}{2} \rfloor + \frac{n}{m_1} + p(t_1, \varepsilon)(t^2 + \frac{n}{m} - \frac{n}{m_1}) \tag{7.11}$$

for binary BCH codes and

$$N = t + t_1 + \frac{n}{m_1} + p(t_1, \varepsilon)(2t^2 + \frac{n}{m} - \frac{n}{m_1}) \tag{7.12}$$

for non-binary codes (cf. [82] for the average number of iterations for the BMA).

In order to illustrate the performance of the new approach we consider a typical example of error correction in flash memories. To provide a sector size of 1 Kbyte, a code dimension of $k = 8288$ bits is required which includes the data bits plus some additional bits used by the flash controller. The error correction should ensure a FER less than 10^{-16}. Three different BCH codes with different error correction capabilities are considered. The parameters of the codes are summarized in Table 7.1.

Because the crossover probability of flash memories deteriorates with increasing number of read/write cycles, we consider the maximum tolerable crossover probability. In Table 7.1 the value p_t denotes the maximum crossover probability so that the FER is less than 10^{-16} for a given error correction capability.

The value $t_{1,opt}$ is the value of t_1 that achieves an average number of iterations similar to the number of iterations required by a fully parallel BMA implementation and an eight-bit-parallel

t	24	48	96
ε	$3 \cdot 10^{-4}$	$1.3 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$
$p(t, \varepsilon)$	$1.4 \cdot 10^{-3}$	$6.5 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$
$t_{1,opt}$	8	22	55
average number of cycles (combined)	1096	1162	1296
number of cycles (parallel)	1102	1168	1300
reusable multipliers	17	39	80
total number of multipliers (combined)	111	241	524
total number of multipliers (parallel)	192	384	768
ratio (combined/parallel)	0.58	0.63	0.68
total 4 input LUTs (combined)	10122	21831	44841
total 4 input LUTs (parallel)	13186	29293	56786
ratio (combined/parallel)	0.77	0.75	0.79

Tab. 7.1: Comparison of a fully eight-bit parallel Chien search with the proposed approach

Chien search. This value is calculated for the maximum crossover probability p_t. Moreover, Table 7.1 provides the number of multipliers for a fully eight-bit-parallel Chien search as well as for the combined Chien search with $m_1 = 8$ and $m = 4$. Table 7.1 demonstrates that a reduction of the number of multipliers in the range of $30-40\%$ is possible without deterioration of the average decoding time.

However, the number of gates required to realize a multiplier varies. Hence, the number of multipliers allows only a rough estimation of the actual area reduction. Therefore, the whole design was implemented with Verilog HDL and verified on a Xilinx Virtex4 (xc4vlx200) FPGA. For this target a 22 bit parallel BMA requires 11069 LUTs while the serial implementation for 48 bits only requires 3462 LUT. Overall, 14531 LUT are required to implement the mixed serial/parallel BMA for 48 bit errors, whereas an implementation of the parallel BMA for 48 bit errors requires 29655 LUT. Similarly, for the Chien search the number of LUTs is reduced from 29293 to 21831 (cf. Table 7.1). Hence, with the proposed implementation the number of look-up tables is reduced by 38% without any reduction of the average throughput compared with a fully parallel implementation.

7.2 GF Optimization

Hardware implementations of finite fields are explained in [18]. It can be distinguished between serial multipliers using a single LFSR and parallel multipliers. This enables a variable scaling between area and time complexity. The focus in this work is on parallel multipliers. Later in this section, the impact of the complexity is discussed dependent on the primitive polynomial. Finally, the improvement is demonstrated based on a BCH code using variable GF multipliers to enable configurable block length.

As already described in Chapter 3, an extension field is generated by a primitive polynomial $p(x)$. Based on this, valid operations, which meet the conditions of a field, like addition and multiplication can be implemented using boolean algebra. Remember, addition is realized using a bitwise XOR-logic. On the other hand, a GF multiplication can be expressed as a circular bitwise convolution with additional modulo operation over the primitive polynomial $p(x)$.

First, we discuss the properties of primitive polynomials with respect to their weight distribution. The weight of a polynomial is defined as the number of all coefficients unequal to zero.

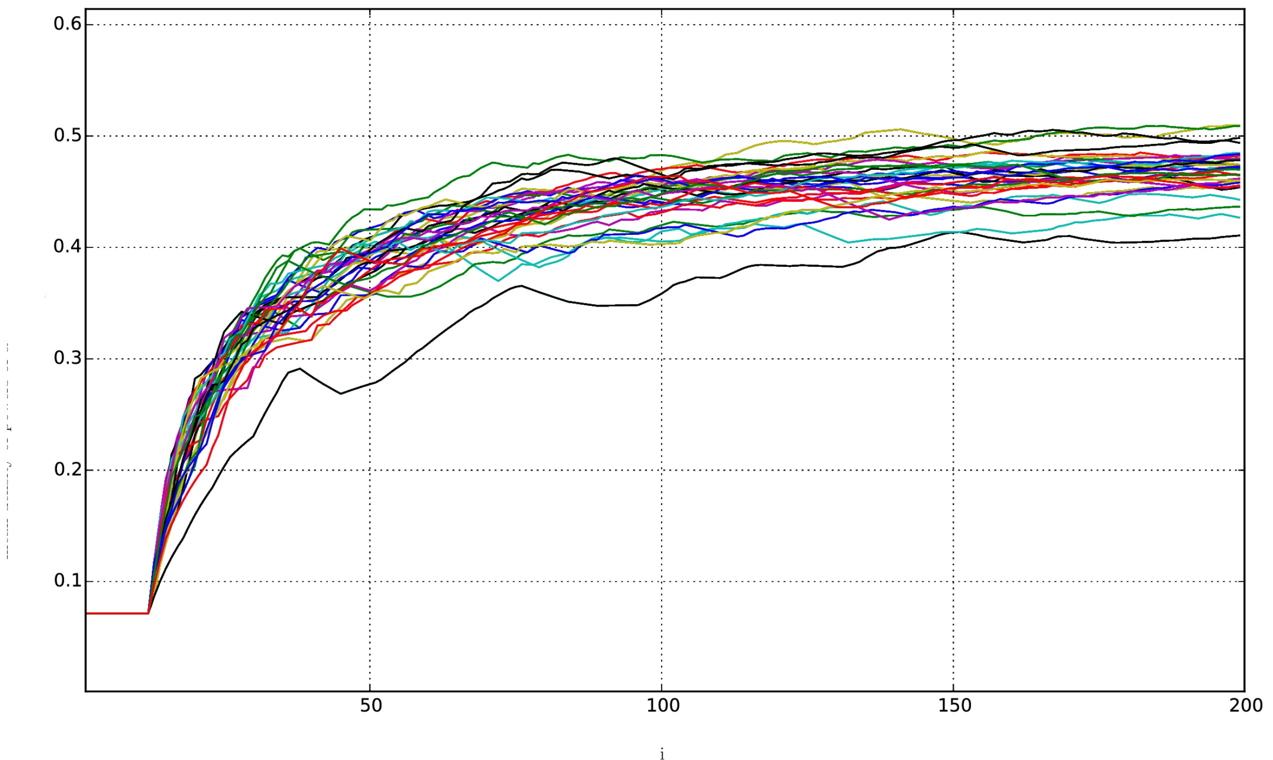


Fig. 7.4: mean density distribution for different primitive polynomials of degree 14

For all possible primitive polynomials, the GF has the property that all appearances of zeros and ones in the coefficients α^i for $i \in 0 \dots 2^m - 1$ is equal and thus its mean density converge to 0.5. However, if the distribution up to a value i is considered much smaller than $2^m - 1$, then there exist a major difference in the mean density between different primitive polynomials. In Figure 7.4 the mean density distribution for different primitive polynomials of degree 14 is shown. The first m GF elements have a mean density of $\frac{1}{14}$. After the 14-th element is a variance between different primitive polynomials. In Figure 7.4 a reduced selection of those primitive polynomials is shown.

Primitive polynomials $p(x)$ determine the partial products in the reduction stage. A coefficient unequal to zero means an addition, which is implemented as a XOR-logic element. However, the primitive polynomial also describes the coefficients used in constant values.

Figure 7.5 shows the schematic of a GF multiplier. The interface comprises two inputs, the multiplicands $a(x)$ and $b(x)$, and one output, the product $y(x)$. Internally the multiplier can be divided into two parts. The multiplication stage that can be understood as a circular convolution of $z(x) = a(x)b(x)$ and the reduction stage $y(x) = z(x)\ mod\ p(x)$.

As can be seen in the Verilog listing 7.1, the multiplication stage comprises element wise multiplication of a_n and b_m using a logical AND. The partial product results are added together using the logical XOR. Finally the reduction stage is applied.

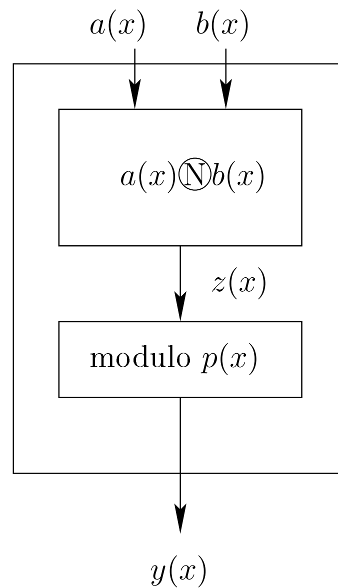


Fig. 7.5: GF multiplier scheme

Listing 7.1: degree 4 GF multiplier in Verilog

```
module multiply_m4(y, a, b);
   input  [3:0] a, b;
   output [3:0] y;
   reg    [3:0] y;

   wire Z0, Z1, Z2, A3, Z4, Z5, Z6;

   /* multiplication stage */
   assign Z0 = (a[0]&b[0]);
   assign Z1 = (a[0]&b[1])^(a[1]&b[0]);
   assign Z2 = (a[0]&b[2])^(a[1]&b[1])^(a[2]&b[0]);
   assign Z3 = (a[0]&b[3])^(a[1]&b[2])^(a[2]&b[1])^(a[3]&b[0]);
   assign Z4 = (a[1]&b[3])^(a[2]&b[2])^(a[3]&b[1]);
   assign Z5 = (a[2]&b[3])^(a[3]&b[2]);
   assign Z6 = (a[3]&b[3]);

   /* reduction stage */
   always@(Z0 or Z1 or Z2 or Z3 or Z4 or Z5 or Z6)
   begin
      y[0]=Z0^          Z4;
      y[1]=   Z1^       Z4^Z5;
      y[2]=      Z2^       Z5^Z6;
      y[3]=         Z3^       Z6;
   end
endmodule
```

Multiplier with a variable input $a(x)$ and a constant input $b(x)$ undergo a optimization in the multiplication stage. There are two possible reductions, which can be explained using boolean algebra. Assume, $b[y_0]$ is a coefficient equal 1, then $a[x_i] \wedge b[y_0] = a[x_i] \wedge 1 = a[x_i]$

thus a logical AND is diminished. If the coefficient $b[n_0]$ is a 0 then the output of the product is $a[x_i] \wedge b[y_0] = a[x_i] \wedge 0 = 0$. Thus the product and the summation is removed. In the listing four products of $a[x_i] \wedge b[y_0]$ can be seen. This means that each of these influences is effected m times in a GF multiplier.

In the reduction stage are $m \cdot 2 - 1$ additions of the partial products of $z_i \cdot \alpha^i$ for $i \in 0 \dots m \cdot 2 - 1$. This means that the complexity depends on the weight of all coefficients α^i for $i \in 1 \dots 2m - 1$.

Up to now only GF multipliers are considered that provide a fixed primitve polynomial. In the application there exists also the demand on variable block sizes. This can be achieved by using multi mode GF multipliers that can switch between different predefined primitive polynomials with different degree m. The advantage of combining the multiplier implementation is, that the multiplication stage can be reused. Only the reduction stage has to be extended by a second modulo operation. In the case of a constant multiplier, the coefficients are multiplexed. At each coefficient degree where the constant value is the same, the reduction rules as described are persistent. In the case of a difference in the coefficient value, logic elements cannot be reduced. Based on this metric extension a primitive polynomial pair can be searched.

In order to discuss the potential of this optimization approach a combined BCH code is considered. The implementation can correct up to 96-bit errors in a 1-kByte block size using a $GF(2^{14})$ and up to 80-bit errors in 2-kByte block size using a $GF(2^{15})$. Three cases are examined to describe its influence in the implementation complexity.

Best primitive polynomial pair:

$$\begin{aligned} p_{14}(x) &= x^{14} + x^{11} + x^{10} + x^9 + x^6 + x^5 + x^3 + x^1 + 1 \\ p_{15}(x) &= x^{15} + x^{14} + 1 \end{aligned} \tag{7.13}$$

Standard MATLAB primitive polynomial pair:

$$\begin{aligned} p_{14}(x) &= x^{14} + x^{10} + x^6 + x + 1 \\ p_{15}(x) &= x^{15} + x + 1 \end{aligned} \tag{7.14}$$

Worst primitive polynomial pair:

$$\begin{aligned} p_{14}(x) &= x^{14} + x^{12} + x^8 + x^7 + x^4 + x^3 + 1 \\ p_{15}(x) &= x^{15} + x + 1 \end{aligned} \tag{7.15}$$

Based on these primitive polynomials, the implementation complexity in the unit of LUTs for the described BCH decoder is listed in Table 7.2. For the implementation Protocompiler and the HAPS-DX evaluation board with a Virtex-7 FPGA was used.

	Syndrom	BMA	Chien	Sum
best	14674	14079	44779	73532
MATLAB	15171	13982	48296	77449
worst	17576	14218	59904	91698

Tab. 7.2: Virtex7 Synthesis LUTs results

As can be seen from this table, the complexity reduction between the best and the worst primitive polynomial pair is about 19.7% for the overall decoder system. The syndrome calculation comprise constant multipliers and has a gain of 17.6% between the best and the

worst primitive polynomial pair. A lower gain can be seen in the BMA because it consists only of full multipliers where takes only an effect in the modulo stage. It can be seen from Table 7.2 that the Chien search has the largest number of constant multipliers, with a gain of 26.5%.

7.3 Summary

This chapter described the improvements for a state-of-the-art flash controller BCH decoder. In Section 7.1 a combination of a mixed parallel and serial BMA and a multi speed Chien search was discussed. Its goal was to handle a trade-off between implementation size and throughput that depends on the RBER. It was shown how an optimal ratio between parallel and serial calculation is determined. We have also shown the area reduction of 38% for an exemplar 48 error-correcting BCH FPGA implementation.

In Section 7.2 the optimization of the GF arithmetic was discussed. The primitive polynomial has an impact on the necessary logic in constant and full multipliers. By means of an example, it was shown that an FPGA synthesis area results between the best and worst case scenario makes a difference of 19.7%.

These optimizations are applicable in the GCC. The BMA of the inner GCC code is a candidate of a mixed parallel and serial implementation. At least one error is occurring relatively often. The highest level of the nested BCH code then determines the t_1 of the parallel BMA. E.g. $t_{b,L-1} = 13$ then $t_1 = 7$ is able to correct all single errors in the later described pipelined BMA. If more errors occur in a column, then the pipeline is stopped and the remaining degrees of the error-location polynomial are calculated in a serial BMA implementation.

Because the component codes rely on the GF arithmetic, the GF optimization also affects the size of the overall GCC implementation.

Chapter 8

Implementation of GCC with Hardinformation

In this chapter, the focus is placed on the hardware implementation of the proposed pipelined GCC decoder from Chapter 4. Chapter 7 was a starting point for the implementation of the component BCH codes and in parts of the RS codes. The main difference for the GCC inner BCH codes are the error correction capability, the bit width of the syndrome, Chien search, and the pipelined BMA. Moreover, the outer RS codes need a key equation solver that considers erasures and calculates an additional error-value polynomial. We also describe the GCC encoder whose BCH encoder model is similarly used in the re-imageing and re-encoding step of the decoder. In Section 8.1 we describe a realization of the decoding scheme from Figure 8.1 in hardware. Beside the question of the logic complexity, the requirements for buffers are investigated in Section 8.2. In Section 8.3 a comparison between a state-of-the-art BCH code and the GCC is presented. Finally, in Section 8.4 an outlook of improvements that require further investigations is provided.

The novelties in this chapter are:

- A GCC hardware design for hard information decoding and
- its analysis regarding throughput and area consumption.

Parts of this chapter are published in [88, 91, 92, 99].

8.1 GCC Data Matrix and Pipelined Decoder

In this section, we draw a hardware design of the GCC decoder that is refined step-by-step. Figure 8.1 shows the decoding scheme which serves as implementation guideline. In this figure repeating tasks for the levels can be identified which differ in the code distance parameter. Thus, a received codeword is processed several times by the same decoder modules that are configurable. From this starting point we develop a decoder architecture. In order to achieve a high throughput, a pipelined approach is developed.

Due to the small GF alphabet and the small error correction capability of the BCH codes, the implementation allows an efficient pipelining structure as proposed in Figure 8.1. Except the BMA algorithm for the outer RS codes all decoding components are arranged in a pipelining structure. We explain Figure 8.1 from left to the right.

In order to store the received word the decoder has a data matrix buffer which is implemented as a shift register. This shift register has $n_a + D_{BCH}$ columns where each column is

Fig. 8.1: Implementation Scheme of A Pipelined GCC Decoder. Reproduced by permission of the Institution of Engineering & Technology [92]

components	cycles
BCH syndrome calc.	1
BCH iBMA	$t_{b,L-1}$
BCH Chien search	1
BCH re-image	$L-l$, (1)
RS syndrome	1

Tab. 8.1: First pipeline latencys

n_b bits wide. The parameter D_{BCH} is the delay caused by the syndrome calculation and the BMA. This value depends on the error-correcting capability of the code. We choose a constant value for D_{BCH}, i.e. the delay is determined by the maximum error-correcting capability of the BCH codes. $n_a \times n_b$ bits are required to store the entire codeword. The additional D_{BCH} columns have a delay function to keep the data synchronized with the decoding process to prevent additional alignment cycles.

In the first iteration the switch $sw1$ connects the decoder with the input line and the data is loaded column-wise. Each new input column is fed into the buffer and the BCH decoder. In every pipeline cycle the data matrix buffer is shifted one column to the right. The data matrix buffer loads the columns into the leftmost column and all columns are right shifted until the rightmost column ($n_a + D_{BCH}$). The second branch leads into the decoder. Table 8.1 lists the decoder components and the cycle times in the same order as they are used in the first part of the pipeline. The re-image component in this first example uses a register in every level and thus it has a latency of $L-l$ cycles. The re-image can also be implemented using only combinatorial logic.

8.1.1 BCH Decoding

First, the BCH syndrome calculation is applied calculating all $S_i, i = 0, \ldots, 2t-1$ syndromes in one cycle. The result is forwarded to the inversionless Berlekamp-Massey algorithm (iBMA). We follow the notation of [35] for the description of the BMA.

The iBMA is a sequential algorithm, where the error-location polynomial is updated within t iterations. In order to speed up the decoding process, we use a pipelined implementation, i.e. we use an instance of the algorithm for each iteration. In the j-th instance, the iBMA first calculates the discrepancy Δ according to

$$\Delta_j = \sum_{i=0}^{M^{(j-1)}} \sigma_i^{(j-1)} S_{2j-i-1} \,. \tag{8.1}$$

$M^{(j-1)}$ is the degree of the error-location polynomial $\sigma^{(j-1)}(x)$ in instance $j-1$. Next the iBMA checks whether the error-location polynomial has to be increased

$$\delta = \begin{cases} 1 & , \Delta_j \neq 0 \text{ and } 2M^{(j-1)} \leq j-1 \\ 0 & , \text{otherwise} \,. \end{cases} \tag{8.2}$$

Finally, the iBMA updates $\sigma^{(j-1)}(x)$ according to

$$\begin{bmatrix}\sigma^{(j)}(x)\\ \nu^{(j)}(x)\end{bmatrix} = \begin{bmatrix}\theta^{(j-1)} & -\Delta_j x\\ \delta & (1-\delta)x\end{bmatrix} \cdot \begin{bmatrix}\sigma^{(j-1)}(x)\\ \nu^{(j-1)}(x)\end{bmatrix}$$
$$\theta^{(j)} = \delta\Delta_j + (1-\delta)\theta^{(j-1)} \text{ and}$$
$$M^{(j)} = \delta(k - M^{(j-1)}) + (1-\delta)M^{(j-1)}\,. \tag{8.3}$$

Here, $\nu^{(j)}(x)$ is a supporting polynomial that is required for the update of the error-location polynomial. The variable $\theta^{(j)}$ controls the multiplication to avoid inversions.

From these equations follows that the number of multiplications for the discrepancy and the update of $\sigma(x)$ and $\nu(x)$ are growing with the degree $M^{(j)}$ of $\sigma(x)$. Hence, the first stages require fewer finite field multipliers. Moreover, the iBMA has to be configurable to solve the key equation for all possible error-correcting capabilities of the inner codes.

In the next pipeline step, the Chien search checks all roots for each error position in parallel. The evaluation of the error-location polynomial is implemented to solve all multiplications and additions in one cycle. In the same cycle the column codeword is corrected by loading the column D_{BCH} and flipping the bits at the detected error positions. The corrected column codeword is then fed into the BCH re-imaging stages.

The iBMA using the update rule in Eq. (8.3) does not change for even syndrome value due to their linear dependencies. Thus the update rule can be modified such that it takes the even $2i$ and odd $2i+1$ syndrome values into account

$$\begin{bmatrix}\sigma^{(j)}(x)\\ \nu^{(j)}(x)\end{bmatrix} = \begin{bmatrix}(1-\delta)\theta^{2(j-1)} + \delta\Delta_j\theta^{(j-1)} & -\delta\Delta_j^2\nu^{(j-1)}(x)\\ \delta x & (1-\delta)x^2\end{bmatrix} \begin{bmatrix}\sigma^{(j-1)}(x)\\ \nu^{(j-1)}(x)\end{bmatrix}$$
$$\theta^{(j)} = (1-\delta)\theta^{(j-1)} + \delta\Delta_j\theta^{(j-1)}$$
$$L^{(j)} = (1-\delta)L^{(j-1)} + \delta(k - L^{(j-1)})\,. \tag{8.4}$$

Using Eq. (8.4), the intermediate steps need no registers and thus this solution is smaller and needs half the number of cycles as Eq. (8.3)

8.1.2 Re-imaging

The re-imaging process is the inverse mapping of the encoding of a column codeword which is depicted in Figure 4.6. Each re-image stage comprises a direct encoder implementation for the corresponding inner code. Hence, the re-imaging stage is equivalent to the encoding of the inner codes in the GCC encoder. In each clock cycle the codeword is inferred for one level until the desired level is reached. Figure 8.2 shows how the pipelined encoders are implemented in a cascaded form, where the information $a_{j,i}$ of level i is first encoded and then the parity $\mathbf{p}_{j,i}$ is subtracted from the column codeword.

After a latency of $L-l$ cycles, which is caused by the re-imaging process, the current RS codeword row i is selected by a de-multiplexer. Subsequently, the RS symbol $a_{j,i}$ is stored in the RS buffer, which stores an entire outer codeword $\mathbf{a}_i$. The symbol $a_{j,i}$ is loaded into the next cycle of the RS syndrome calculator.

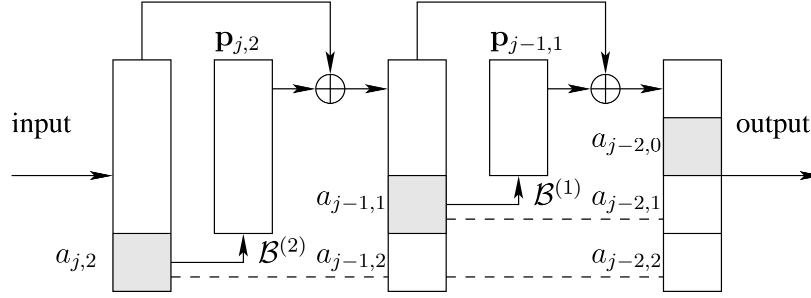


Fig. 8.2: BCH re-image cascade. Reproduced by permission of the Institution of Engineering & Technology [92]

8.1.3 RS decoding

The syndrome for the RS code is calculated using Horner's rule

$$S_i(\alpha^i) = (\dots(r_n\alpha^i + r_{n-1})\alpha^i + r_{n-2}) + \dots)\alpha^i + r_0\,. \tag{8.5}$$

This has three advantages:

- The RS codeword is arriving symbol-wise and the syndroms can be calculated symbol-wise.
- The RS codeword is read in reverse order which simplifies the Chien search and the Forney algorithm.
- Horner's method only needs a single multiplier and a single adder per syndrome value.
- The Horner's method can also be partially parallelized such that n symbols can be calculated at once (see Equation 7.2).

Once the last column has reached the RS syndrome calculation, the pipeline is paused and the RiBMA [57] starts to calculate the error-location polynomial $\sigma(x)$ and error-value polynomial $\Omega(x)$ of the outer RS code based on the RS syndrome values. The RiBMA needs $6t$ cycles. When $\sigma(x)$ and $\Omega(x)$ are ready the second part of the pipeline can be started. First the RS Chien search is loaded with $\sigma(x)$ and checks one location per cycle. Because the Forney algorithm needs the derivative $\sigma'(x)$, which can be retrieved from the Chien search, it starts with one cycle delay with respect to the Chien search. If the Chien search has found an error position, the error value is calculated by the Forney algorithm. This value is then applied synchronously to the output of the RS buffer.

After decoding the RS codeword, the information symbols can be shifted to the output. Furthermore, the corrected outer code symbols are re-encoded with the inner code of the actual level and subtracted synchronously from the GCC data matrix buffer. In the next iteration, the switch $sw1$ is switched to the feedback line and the result is fed back into the GCC data matrix buffer and the BCH decoder pipeline which continues with the decoding process.

Except for the RiBMA all components of the RS decoder have one clock cycle latency. The total number of cycles can be calculated based on the cycle latency from Table 8.1 as well as Equations (8.6), (8.7), and (8.8). The number N_{pipeline} of pipeline cycles comprises five cycles for the decoder operations that occur in each iteration and the entire number of columns which are processed in each iteration. This is followed by the iBMA which depends

on the error correction capability of the inner code. Additionally, the pipeline has $\frac{(L-1)L}{2}$ BCH re-image and $L-1$ BCH re-encoding cycles.

$$N_{\text{pipeline}} = (5+n_a)L + 2\sum_{i=0}^{L-1} t_{b,i} + \frac{(L-1)L}{2} + L - 1 \quad (8.6)$$

The RiBMA depends on the error correction capability of the outer code.

$$N_{\text{RiBMA}} = 6\sum_{i=0}^{L-1} t_{a,i} \quad (8.7)$$

The total number of cycles is

$$N_{\text{total}} = N_{\text{pipeline}} + N_{\text{RiBMA}} \quad (8.8)$$

The pipeline can be instantiated multiple times. Accordingly, the implementation can be either optimized in speed or in the cell area. The RiBMA can furthermore be optimized with its pipelined version pRiBMA [57] that needs only $3t$ cycles. In [72], several approaches are investigated. A good choice for error and erasure RS decoder is the universal parallel inversionless Blahut algorithm (UPIBA). It considers erasures and inherently computes the error-value polynomial $\hat{\Omega}(x)$.

8.1.4 Hardware Architecture

Figure 8.3 shows the hard-decoder module. It comprises two inner decoder modules and one outer decoder module. The inner decoder modules are loaded in an alternating sequence at one cycle. b_0 comprises all columns with an odd index and b_1 of all columns with an even index. The interaction between the inner decoders and the outer decoder comprises a feedback loop, where the inner decoders estimate RS code symbols $a_{j,l}$ and receive the corrected symbols $\hat{a}_{j,l}$ for each level. Because we use the parity-check code, the inner code can detect errors. This erasure information era is passed to the outer decoder.

While the input b is read column-wise, the output $\hat{a}$ is processed row-wise. This means, that m bits of a column are written per cycle. In each level the column grows by m bits until $k_b^{(0)}$ bits are reached. As we use two instances of the inner decoder, we can process two symbols per cycle and thus can fill up two different columns per cycle.

The reassembling comprises θ_b buffers, that also can be implemented as a double buffer to reduce latency. For ASIC memories exists the possibility with the bit write enable (BWE) lines to write only dedicated bits within a memory word of width $k_b^{(0)}$. This mechanism does not exist in FPGA block memory models but can be solved by using $L \cdot \theta_b$ parallel memories with a width of m and depth of $\lceil \frac{n_a}{\theta_b} \rceil$. In most cases a bit width conversion from n_b for b respectively $k_b^{(0)}$ for $\hat{a}$ to some power of 2 has to be applied.

The inner decoder is shown in Figure 8.4. This module comprises all parts necessary for decoding BCH codes and for the re-encoding that provides the interface between the BCH and RS codes. In the first level, all columns from the input $b^{(0)}$ are loaded through the demultiplexer into the GCC buffer and the decoding unit that comprises syndrome calculation, BMA and Chien search. The GCC buffer is a static random-access memory (SRAM) arrangement such that parallel access is enabled [92]. If a resulting error vector is available, the

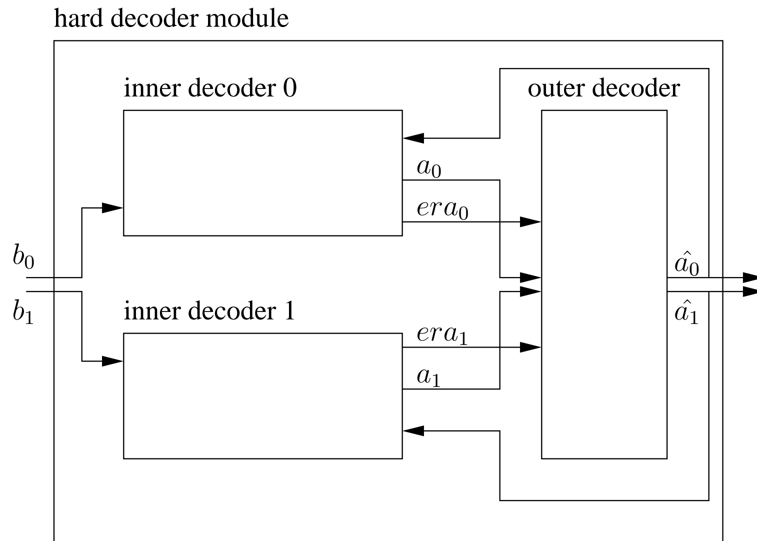


Fig. 8.3: Data flow of the hard decoder module. Reproduced by permission of the Institution of Engineering & Technology [101]

corresponding erroneous column is loaded from the GCC buffer and the correction is applied. The result is re-imaged regarding the current level and the outer code symbol $a_{j,l}$ is extracted. In the levels $i > 0$ the resulting outer codeword symbol $\hat{a}_{j,l}$ of the previous level is re-encoded to BCH codeword and subtracted from the corresponding column of the GCC buffer. The resulting column is used in the next decoding stage. All of the elements of the inner decoder are arranged in a pipelined structure such that one column can be decoded per cycle. All necessary syndrome values are calculated per cycle. The pipelined BMA needs $t_{b,l}$ cycles to calculate the error-locator polynomial, where $t_{b,l}$ is the error-correcting capability of the inner code in level i. Similar to the syndrome calculation the Chien search determines all error positions within one cycle. Moreover, the numbers of estimated error is determined for failure detection. This failure detection is needed for erasure decoding of the RS codes.

If the soft-decoding mode is activated, the inner decoder receives already corrected columns from the first soft-decoding stage. These columns are stored in the GCC buffer and re-imaged for the outer decoder. In the subsequent levels the columns are passed to the soft-decoder module if the syndrome is non-zero. The resulting column codeword is inserted to the re-image stage.

Because the inner codes are binary BCH codes, the even syndrome values are linear dependent on the odd ones. Thus, the inner syndrome calculation module only calculates all odd syndromes and the zero syndrome and needs $t+1$ multipliers, where t is the largest error-correcting capability of the inner codes. The calculation of all necessary even syndrome values is shifted into the pipelined BMA where t additional multipliers are needed. This shifting saves registers that would be needed to store the syndrome values. The inner Chien search evaluates the error-location polynomial $\sigma_b(x)$ and checks all possible roots in parallel in a single cycle. The re-image and re-encoding modules are similar and deliver their result in one cycle. In the re-encoding stage, $\hat{a}^{(l)}$ is re-encoded with $\mathcal{B}^{(l)}$.

The implementation of the outer decoder is an RS decoder unit with erasure decoding capability. Thus this module comprises the RS symbol input that expects θ_b symbols per cycle. Like the inner decoder, these symbols are passed to an RS buffer and the syndrome calculation that calculates all syndrome values. Similar to the encoder, the syndrome calculation is usually implemented using an LFSR [66, 54, 35]. While the pipeline is running, erasure signals era

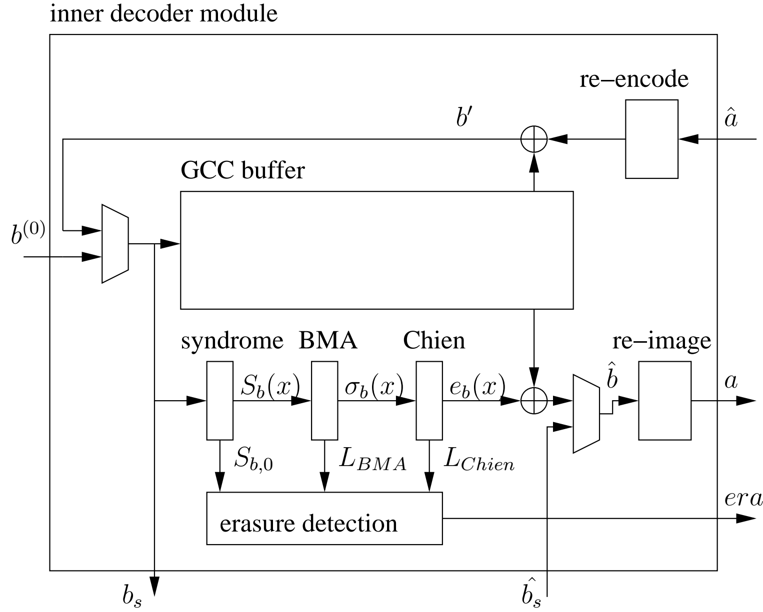


Fig. 8.4: Data flow of the inner decoder module. Reproduced by permission of the Institution of Engineering & Technology [101]

can occur. A column position j that contains an erasure is stored as $\alpha^{(2^m - n_a + j)}$ for each inner decoder instance in a separate stack, where α is the primitive element of $GF(2^m)$.

Based on the syndrome values and the erasure values, the error-locator polynomial $\sigma_a(x)$ and complementary error-value polynomial $\hat{\Omega}_a(x)$ are calculated with the so-called Blahut algorithm [72]. In particular, we used the UPIBA that calculates $\sigma_a(x)$ and $\hat{\Omega}_a(x)$ in two iterations and decreases the number of GF multipliers. As we use shortened outer codes, $\sigma_a(x)$ and $\hat{\Omega}_a(x)$ have to be adjusted such that the Forney and Chien search start with the correct position. This is achieved by an additional cycle in the UPIBA when $\sigma_a(x)$ and $\hat{\Omega}_a(x)$ are determined, respectively.

Once $\sigma_a(x)$ and $\hat{\Omega}_a(x)$ are available, the pipeline process can continue by correcting the RS code. In parallel the Chien search checks and counts the error positions whereas the Forney module calculates the error magnitude. A high-speed implementation of this processing is described in [39].

A non-pipelined GF multiplier is sufficiently fast, whereby fewer registers are needed. In the outer decoder module, the GF multipliers of the Chien search, Forney and UPIBA are identical but not used at the same time. Because the UPIBA is idle while the Forney and Chien search are processing and vice versa, these multipliers are shared between these modules by using multiplexers. This is efficient because the Forney algorithm needs $2t + 1$ and the Chien search $2t$ multipliers which corresponds well with the $4t + 2$ multipliers that are needed in the UPIBA module [72].

A special requirement for the RS buffer is that it loads new values while it is unloading the symbols for the last level. This is implemented using a dual-port static random-access memory (DP-SRAM). The gap between the read and write process corresponds to the pipeline latency.

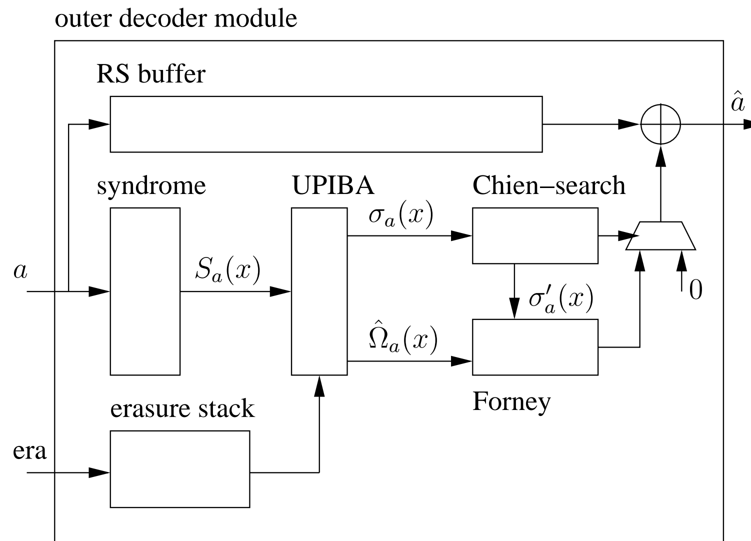


Fig. 8.5: Data flow of the outer decoder module. Reproduced by permission of the Institution of Engineering & Technology [101]

8.1.5 Encoder

The encoder performs two tasks: First each row is encoded with the desired RS code then each column is encoded with the nested BCH code. At the beginning the data is loaded into the RS encoder and simultaneously into the correct row of the GCC data matrix. Typically, the encoder is implemented as an LFSR where the coefficients of $g(x)$ represent the taps of the feedback [44, 66, 54, 35]. Because the GCC consists several RS codes, it comprises multiple LFSR instances. When finally a row is encoded the multiplexer switches to the next row. If all rows are encoded the GCC data matrix can be read column-wise while each column is encoded with the nested BCH code. Instead of using an LFSR the BCH encoders uses a direct full-parallel parity calculation.

8.2 Codeword Buffer

In this section, we describe an improvement of the GCC codeword buffer that was previously described by a shift register which has major impact on the die area of the ASIC implementation. We first address the problem that comes from the flip-flop circuit shown in Figure 8.1. We show different solutions that can be made more efficient with SRAM modules. The most efficient implementation will be described in further detail. Synthesis results can be found in Section 8.3.

Above a certain size, a register cell requires more area as a memory cell in an SRAM. The requirements for the access of the GCC buffer are:

- column-wise access
- incremental addressing
- two read and one write access with constant column distances

We consider several SRAM constellations with different access mechanisms. There exists single-port static random-access memorys (SP-SRAMs) where in each cycle only one single

word can be either read or written. By contrast, DP-SRAMs can access read and write multiple words from or to the SRAM.

Single SP-SRAM A single-port SRAM for the GCC buffer requires a data width of n_b bits. Its length is n_a rows. The column positions of the GCC codeword are static with its addressing. This solution is the simplest and smallest. Its disadvantage is that this method requires three independent memory accesses that have to be made in three different clock cycles to serve one pipeline cycle. As the pipeline decoder logic can be clocked with the same frequency as the SRAM this clocking would lead to a performance degradation.

Single DP-SRAM A single dual port SRAM is slightly larger as an SP-SRAM but smaller than shift registers. The columns of the GCC codeword are still static with respect to their physical memory addresses. The three accesses that are needed can be divided in two cycles. For example, one write access in the first cycle and two read accesses in the second cycle. The number of access cycles can be reduced from three down to two cycles with this solution.

Two DP-SRAM A third solution is to divide the shift register into two DP-SRAM. The shift register is split at the column D_{BCH}. Each part of the shift register is then replaced by a DP-SRAM.

In this scenario the functionality is reproduced by two ring buffers. The columns of the GCC codeword are first loaded into the first ring buffer. The tail of this ring buffer is read out and stored in the second ring buffer. Additionally, this column is forwarded into the correction part after the BCH Chien search. The corrected column is fed into the RS syndrome calculation module. When the second part of the pipeline becomes active, the columns are subtracted from the GCC codeword, which are available at the tail of the second ring buffer. The result is inserted into the first ring buffer and the BCH syndrome calculation.

This solution enables each access to be made in the same clock cycle. The disadvantage is that more area is needed than with the SP-SRAM solution.

Three SP-SRAM The solution with three SP-SRAM is investigated in further detail. It uses three SP-SRAM that are simultaneous accessed for reading and writing of the three pipeline access points in a circular scheme. Likewise, the columns of the GCC codeword are distributed in this circular scheme. The mapping between the column number j and the physical memory page access can be described in the following modulo function.

$$SRAM_{id} = j\%3 \tag{8.9}$$

The mapping from the column number into the SRAM address can be undertaken by:

$$SRAM_{address} = \frac{j_{current}}{3} \tag{8.10}$$

0	3	6	9	12	15	...	SRAM 0
1	4	7	10	13	16	...	SRAM 1
2	5	8	11	14	17	...	SRAM 2

Fig. 8.6: GCC column distribution for each SRAM. Reproduced by permission of the Institution of Engineering & Technology [92]

Figure 8.6 depicts the distribution of the GCC columns. With this construction simultaneous access to the memory composition is available if the accesses have a distance of three plus a specific value.

In the case that the distances of the accesses from the decoding logic cannot comply with this rule, additional buffers can be used. In the worst case four additional column registers are needed. The shift register is replaced by the modules in Figure 8.7. These modules are the SRAM iterator module and the three SRAM modules.

The SRAM iterator module controls the addressing of the three SRAM pages. It does the initialization and increments those addresses which comprise a minor and major field. The minor address is incremented each cycle and runs from 0 to 2. Depending on the minor address a corresponding major address is incremented. This module contains three major address registers. These major address registers have to be initialized properly to reconstruct the access distances from the shift register.

The SRAM memory composition comprises three SRAM instances. This module evaluates the minor address and simultaneously forwards the control, data and address lines to each SRAM module.

This solution avoids the memory overhead which is produced by the shift register and the divided DP-SRAM. The overhead disappears because the positions of the GCC columns are static. Furthermore, the overhead which comes from the additional logic and wiring of the DP-SRAM is avoided. All three pipeline memory accesses can be served in simultaneously and without collision in a single clock cycle.

The size of this control logic like the address decoder for each of the three memories is small. This version also requires logic for the multiplexing of the data, address and control lines. Moreover, a relatively small addressing logic is needed.

8.3 Comparison

The GCC construction is well known in the coding theory community. However, to the best of our knowledge there are no published hardware architectures of GCC encoders and decoders. Therefore, we compare the GCC decoder implementation with a long BCH decoder that has comparable error correction performance.

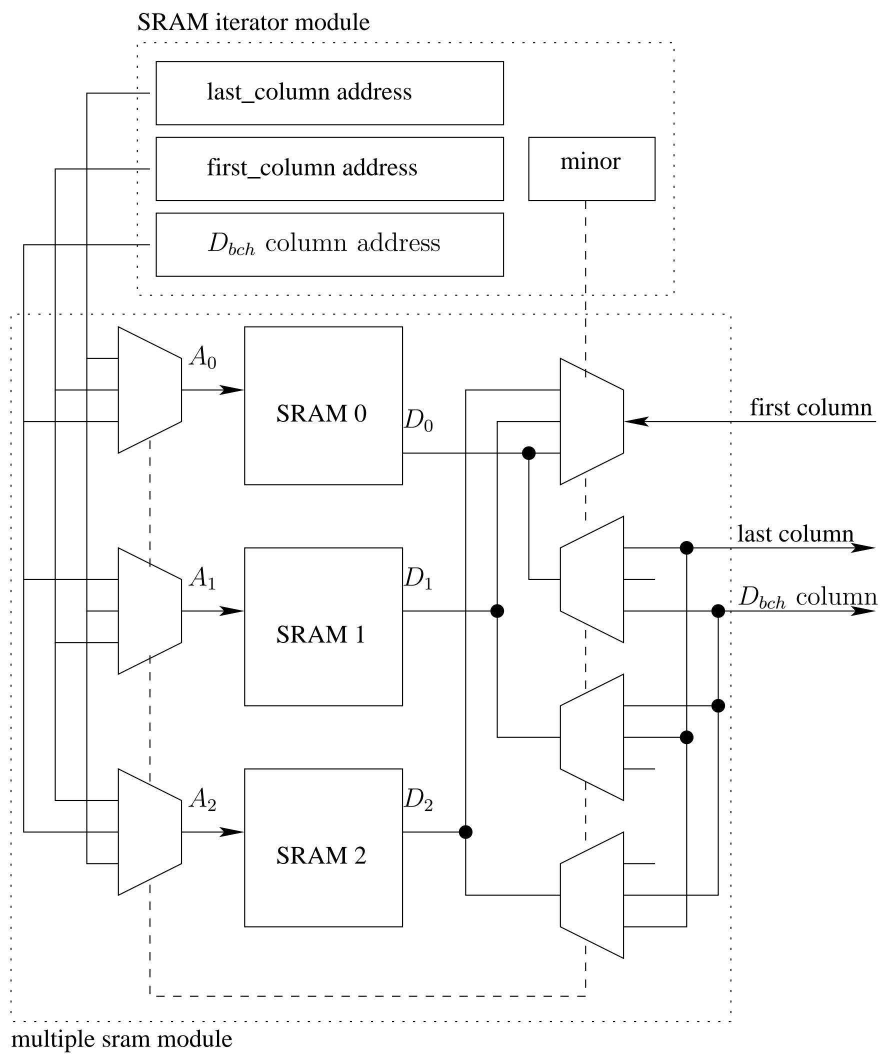


Fig. 8.7: SRAM access control. Reproduced by permission of the Institution of Engineering & Technology [92]

Module	Slacktime (ns)
BCH syndrome	4.66
BCH iBMA	6.20
BCH Chien search	8.80
BCH re-image	8.67
RS syndrome	7.87
RS Chien search	7.39
RS Forney alg.	4.52
RS RiBMA	5.26

Tab. 8.2: Slack time @100 MHz

The first example is an implementation of the proposed GCC code. First we describe the code parameters which meet the constraints of FER and BER. Subsequently, we present cell area size and clock frequency measurements. In the second example, a long BCH code is presented, where we explain the code parameters and offer a brief overview of the decoder implementation. Both decoders are implemented in Verilog and are synthesized with the Design Compiler from Synopsys using a 40nm technology with high density standard cells (HVT + SVT).

Example 16. *For the BCH code we use a 1-kByte information sector. The target FER of 10^{-16} at $\varepsilon = 3.8 \cdot 10^{-3}$ requires a 96-bit-error-correcting BCH code. The long BCH code has the parameters $\mathcal{A}(2^{14}; 9609, 8280, t = 96)$. The implementation was synthesized with the same tool as in Example 4. In order to build a codeword with $n = 9609$ we need a Galois field $GF(2^m)$ with $m = 14$. This is a disadvantage as the complexity of a multiplier is $O(m^2)$ [18]. The implementation is based on a mixed serial/parallel implementation as proposed in [83].*

The smaller Galois fields used in the GCC lead to a much smaller and faster implementation compared to the BCH decoder with an arithmetic in $GF(2^{14})$. The BCH decoder achieves a clock frequency of 100 MHz. Table 8.2 shows the slack times for each module of the GCC decoder. The first seven modules are the pipeline. The pipeline modules and the RS RiBMA can be put in different clock domains. Using a common clock domain the GCC decoder achieves a clock frequency of 180 MHz.

From Equation (8.8) follows that the GCC implementation requires 2 286 cycles. Hence, the decoder achieves a throughput of 1.3 Gb/s, whereas the BCH decoder has a throughput of about 0.6 Gb/s. The worst slack times have the modules BCH syndrome and RS Forney algorithm. In the BCH syndrome module the shorter slack time comes from the calculation of the even syndrome values. These even syndrome values are calculated in the same cycle as squares or higher powers of odd syndrome values. As the iterative iBMA requires only the first syndrome value in the first instance, the calculation of the even syndrome values can be moved in a later cycle without increasing the pipeline latency. Another bottleneck is the RS Forney algorithm. Equation (3.18) shows that first $\Omega(x)$ and $\sigma'(x)$ are evaluated. This needs a parallel multiplication and summation of all products. The final division leads to a second multiplication and causes the small slack time. In order to obtain a shorter critical path the Forney algorithm can be divided into multiple cycles.

The cell area sizes for all modules are listed in Tab. 8.3, where we use a GE as a unit of measure which is the area of a two-input drive-strength-one NAND gate. The decoder logic requires 147 678 GE. Tab. 8.4 lists the cell area sizes for the BCH decoder logic. From these values we observe that the logic of the GCC decoder requires only 30% of the cell area that is required for the BCH decoder.

module name	cell area
BCH syndrome	3 084
BCH pipeline BMA	28 017
BCH Chien search	16 153
BCH re-image	2 025
RS syndrome	3 083
RS RiBMA	82 330
RS Chien search	5 391
RS Forney alg.	7 595
Subtotal (logic only)	**147 678**
RS buffer (shift reg.)	37 126
GCC data matrix (shift reg.)	171 535
Total a) (shift reg.)	**356 339**
RS buffer (SRAM)	30 870
GCC data matrix n_a (SRAM)	31 039
Total b) (SRAM)	**199 587**

Tab. 8.3: GCC ASIC area in GE

module name	cell area
BCH syndrome	153 683
BCH mixed BMA [82]	145 131
BCH Chien search	197 582
Total	496 396

Tab. 8.4: 96-bit BCH ASIC area in GE

However, the values of Tab. 8.3 show that the buffers, which are implemented as shift registers, deploy a cell size of 208 661 GE. This represents 58% of the entire GCC implementation (value a) in Tab. 8.3. In order to reduce the cell area, the GCC data matrix can be implemented using SRAM. A significant cell area reduction can be achieved by the approach presented in Section 8.2. The total cell area size for this improved GCC decoder is 199 587 GE (value b) in Tab. 8.3.

8.4 Improvements

Thus far, a single-mode GCC has been considered. The described decoding flow is very exhaustive in the case of error-free codewords, given that all steps have to be decoded to detect an error. In this section, we first propose an improvement that enables a very low latency in the error-free case and a processing reduction in the case of few erroneous columns.

8.4.1 Speed

First, a brief overview of a common controller channel stream is provided. Figure 8.8 shows the flash channel, which has two components that need to communicate with each other. The flash memory is depicted on the left side and on the right side the internal program/date memory. In between are the ECC logic and additional cryptography and/or compression modules. The communication between the flash controller and the flash memory is critical. Due to the internal arrangement of the data in pages, it is necessary to read a complete page.

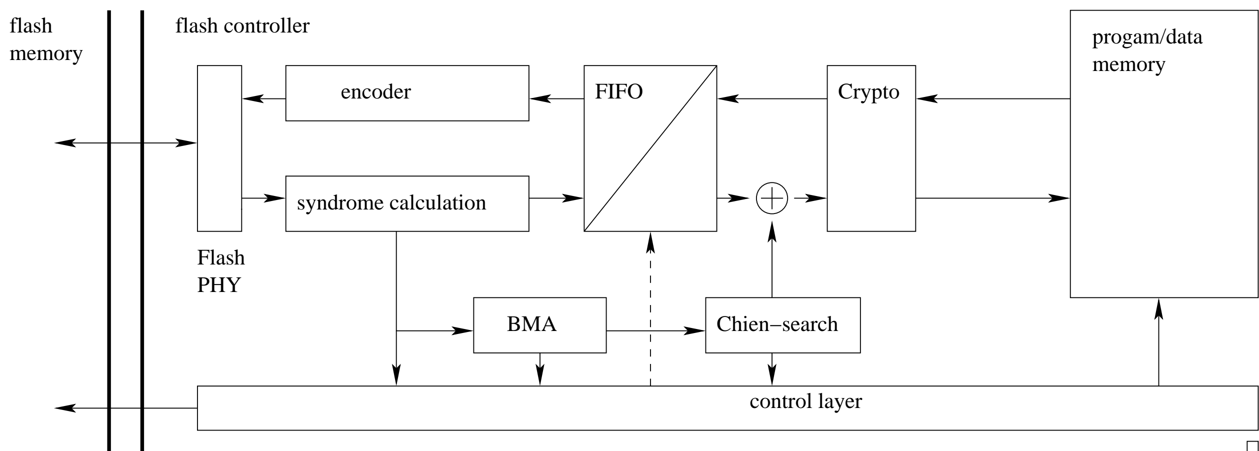


Fig. 8.8: Flash controller channel scheme for BCH codec.

The difficulty is that the data must be transmitted and received across the flash data PHY as a stream, which cannot be interrupted.

The encoding process can satisfy this condition. The information data can be written to the flash memory and the parity be calculated simultaneously. Once the information is sent to the flash and encoder, the parity is ready and can be written to the flash data PHY seamlessly. In the case of reading from the flash memory, the data stream passes the syndrome calculation first and is passed to the internal FIFO. If the data is error less, then the syndrome value calculation is reset and starts to calculate the syndromes for the next block. In the erroneous case, the error correction starts by deploying the BMA and holds the FIFO. Once the error-location polynomial is passed to the Chien search, the data from the FIFO can be corrected and further processed to store this block in the program/data memory. In this case, the data beyond the corrected block is neglected and has to be reread that leads to a significant loss of throughput.

A GCC encoder can easily be designed such that the output stream is seamless. The information data is aligned column-wise. Once a codeword matrix column is filled, it can be encoded using the nested BCH codes. All outer RS codes are processed simultaneously in parallel. The only critical issue occurs in the case that the column size is not a multiple of the data stream from the FIFO.

In contrast to BCH codewords, the GCC codeword is not systematic, which means that it has to be processed by the re-imaging stage to obtain the original information. Thus, with the previously-described GCC decoder, a seamless error correction is not possible, regardless of whether the block is erroneous or correct. This is the case because the correctness of the codeword is only known if all syndromes are checked. In the iterative processing scheme, it is recommended to process each level. A solution is the simultaneous syndrome calculation of all levels. If this correctness indicator is available, the information can be re-imaged seamlessly by using the output of the re-image stage.

If the syndrome values of all outer levels are stored, an improvement can be achieved by marking the erroneous columns for later processing. All syndrome values are known from the first iteration but need to be updated in the case of a corrected column in a later iteration. Instead of recalculating the syndrome values using the entire row, only the syndrome value of the changed symbol is calculated and added to the existing overall syndrome value of the complete row. This is possible due to the linearity property.

The proposed improvements are shown in Figure 8.9. Additional elements are needed to store the number of the erroneous column and the syndrome values of all levels. The column

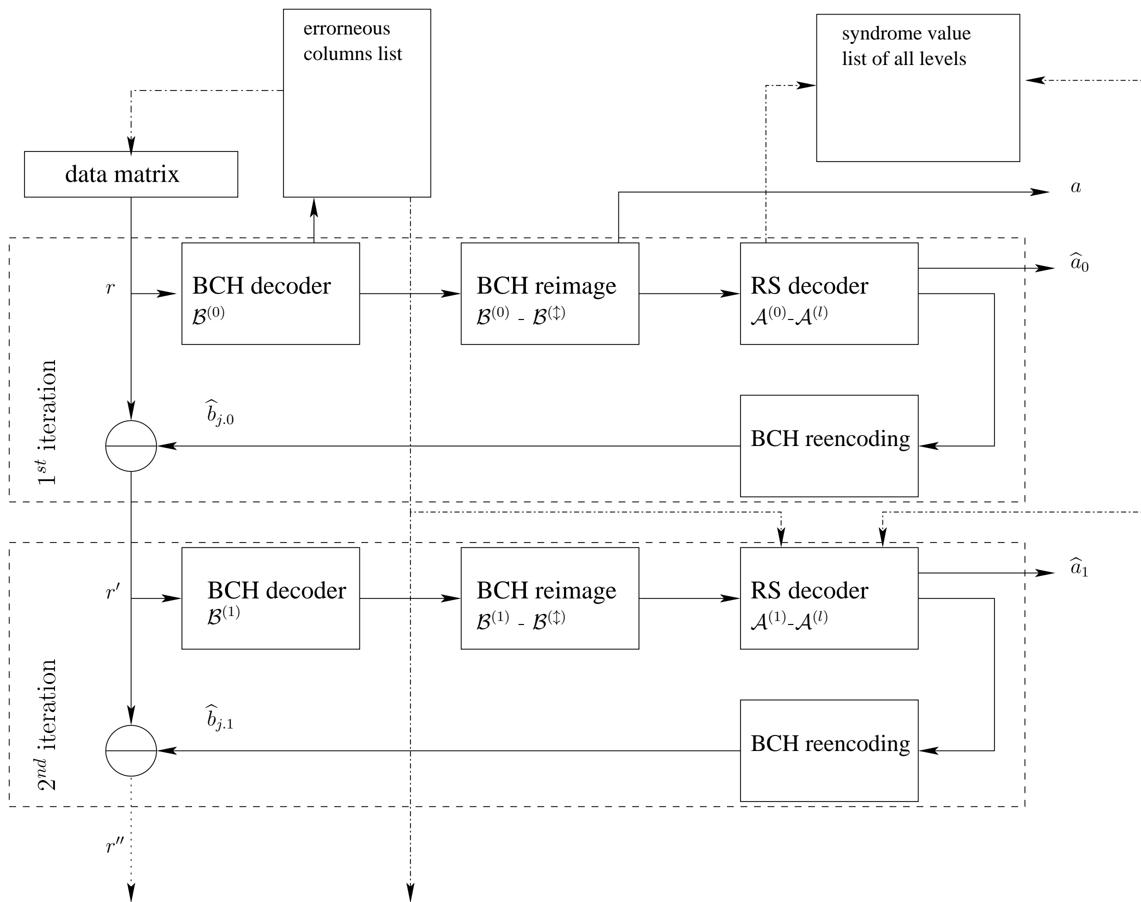


Fig. 8.9: improved GCC decoding scheme

number is used to load and decode only the erroneous columns and as the coefficient for the outer syndrome calculation.

8.5 Summary

In this chapter, we have shown how the GCC decoding described in Chapter 4 can be implemented in hardware. Based on the decoding scheme shown in Figure 4.7, we derived a data flow shown in Figure 8.1, which is the draft for the hardware modules and offers a first overview of the necessary calculation and buffers. From that point, the hardware modules were designed as depicted in Figures 8.3, 8.4 and 8.5. It is clear that the GCC codeword buffer depicted in 8.1 is not suitable to be implemented as a shift register due to the large area. Instead, an alternative buffer was developed. This was possible due to the sequence of column codes that are read. This GCC implementation then was compared to a BCH. By dividing a large algebraic code into several smaller ones the complexity was reduced mainly through the GF arithmetic. The GCC then led to a higher code rate. It also shows that an accurate comparison is difficult due to design criteria like achievable throughput, code rate or decoding performance.

Chapter 9

Implementation of GCC with Softinformation

9.1 Introduction

In this chapter, we propose a decoder architecture extension for a GCC soft-input decoder. In Chapter 6 several soft-decoder approaches were discussed. In this chapter a possible implementation approach is presented and analyzed. First we discuss in Section 9.2 the stack algorithm as an inner decoder of the GCC decoder presented in [91]. Then the stack algorithm implementation for supercode decoding with its subsystems is presented and discussed. Similar for the previously presented SPC and Chase-II, a decoder is described in Section 9.3. In Section 9.3.1 the integration of the soft-decoder modules into the hard-decoder architecture described in Chapter 8 is described. This integration is similar for the stack decoding algorithm and the Chase decoder. We discuss in both cases the throughput and area consumption.

The novelties in this chapter are:

- A soft information decoder implementation design for GCC,
- investigation on several soft-decoding algorithms (e.g. Chase-II, Stack algorithm) as component codes and
- its analysis regarding throughput and area consumption.

Parts of this chapter are published in [94, 101]

9.2 Stack Algorithm and Supercode Decoding

The original hard-input GCC decoder implementation in [91] uses algebraic syndrome decoding. In this implementation the first levels of $\mathcal{B}$ can correct $t_{b,0} = 1$ and $t_{b,1} = 2$ errors. Thus high error correction capabilities of the outer codes $\mathcal{A}^{(0)}$ and $\mathcal{A}^{(1)}$ are required. This leads to lower code rates and a high decoding complexity of those outer codes. On the other hand the soft-decoding complexity of the column codes increases significantly with each code level. Hence soft-decoding is of interest for the lower levels.

Subsequently the algebraic decoding logic for the column code remains in the implementation. Therefore it is possible to check whether the syndrome is zero. In this case the codeword can be assumed to be correct, i.e. neither algebraic decoding nor sequential decoding result in a different codeword.

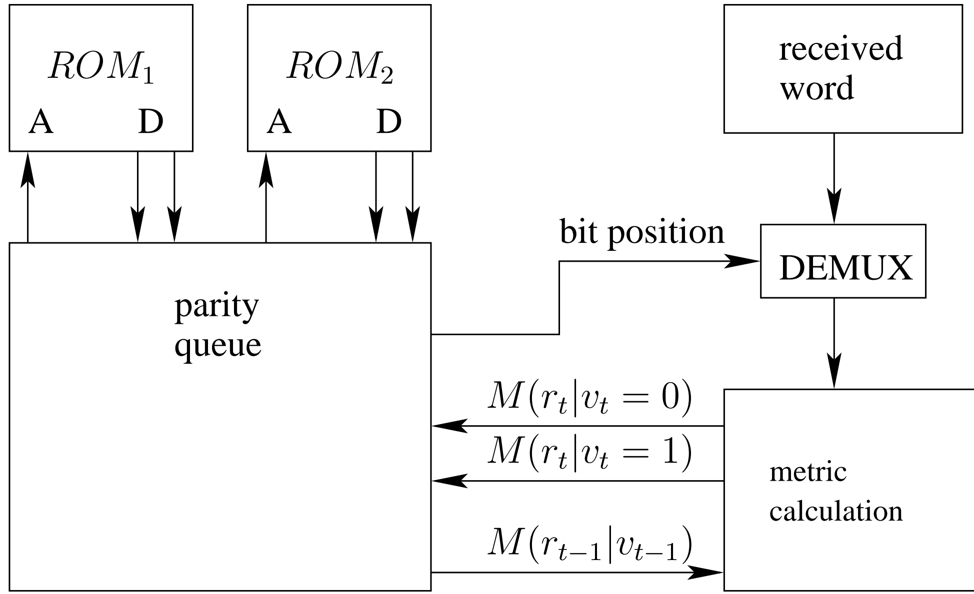


Fig. 9.1: Block diagram of the sequential decoder

9.2.1 Decoding Logic

A brief system overview is depicted in Figure 9.1. The system comprises a word array of size n_b and a desired width which stores the q-ary word. Furthermore, a de-multiplexer selects the currently processed bit position depending on the top path of the stack and delivers this value to the metric calculator. Based on the received codeword symbol r_i and the previous metric $M(r_{t-1}|v_{t-1})$ the metric module returns $M(r_t|v_t)$ to the priority queue block. To represent the supercode trellis asynchronous ROM is used. Each word of the ROM represents a trellis node $\sigma_{t,i}$. The data comprises two pointers for the successor nodes $v_{t+1} = 0$ and $v_{t+1} = 1$.

Depending on the top entry of the priority queue the desired codeword symbol is selected and the next branches for the actual nodes $\sigma_{t,1}$ and $\sigma_{t,2}$ are loaded from the trellis ROM. The priority queue unloads the top entry and loads the new paths in a single clock cycle.

Each entry of the priority queue contains several elements. The first element is the metric value. The trellis path, its length, and a pointer to the current node are stored. All entries have to be ordered by the metric values such that the top entry has the highest value.

The process of the priority queue starts with its initialization. The starting node, its initial metric value and the path length are set. Each update cycle begins with the load phase in which the next node pointers are loaded from the trellis ROM. Simultaneously, the next codeword symbol is loaded based on the path length index. The next metric value can be determined based on the code symbol and the available branches.

With binary codes there exists at least one possible branch and at most two branches. The resulting branches are pre-sorted using combinatorial logic. In the following we call these two entries the major and the minor entries, where the major entry has the better metric value.

All priority queue elements are successively ordered in a chain. Each element can exchange its date with its previous or next neighbor. Furthermore, each element can decide whether it keeps its own data, take the data from its neighbor, load the new major data or the new minor data. In each element the metric value is compared with the new value. The result of this comparison is signaled to its predecessor and successor elements. If the signal of a predecessor is false and the major metric value comparator gives a positive signal the new major value

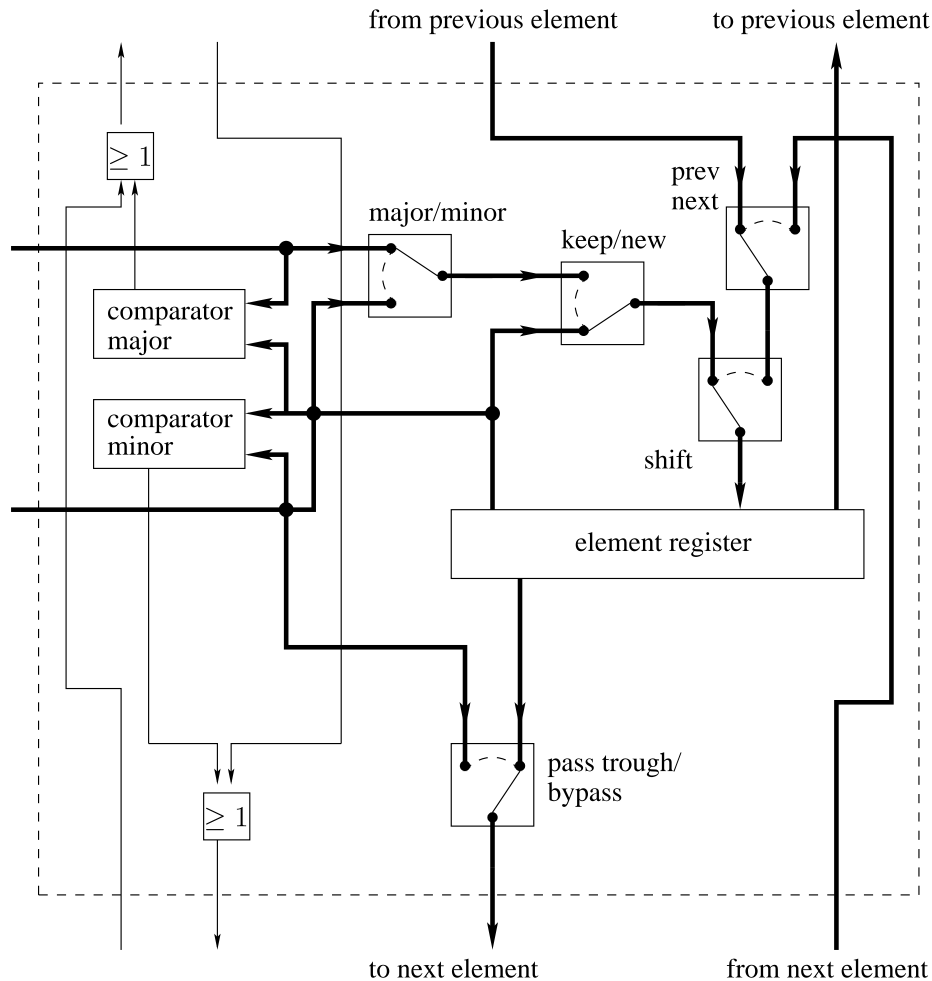


Fig. 9.2: Priority queue element

will be stored. Likewise, if an element receives a false signal from its successor and the minor metric value comparator signals a new metric value that is less than the current value, the new minor data is stored. In the case that an element receives a signal from its neighbors, space for the new data has to be created by shifting all entries to next element.

Two special cases exist that have to be taken into account. The first special case occurs if a node has only a single outgoing branch. In this case, the shifting of elements has to be prevented by signaling. The second special case occurs if the new major and the new minor elements are designated to be inserted into the same entry register. This case can be detected and preventing by passing this value to the next element.

The algorithm terminates if the maximum possible path length is reached. The stored path in the top element is the decoded codeword. In the practical implementation an iteration counter will terminate after a determined maximum number of iterations. This abort can be used to mark this decoded GCC column as a erasure symbol for the outer RS code.

In order to decode supercodes, the following extensions have to be implemented. First for each supercode a distinct ROM is needed which represents its trellis. The metric calculation has to take all trellis branches of each supercode into account. Furthermore, all node pointers have to be stored in the priority queue elements.

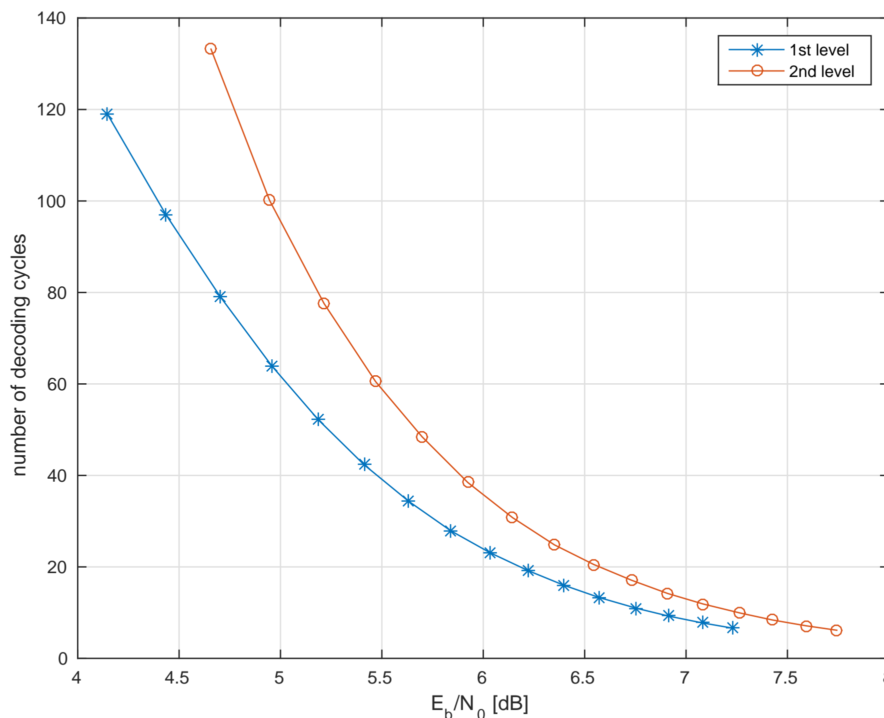


Fig. 9.3: Average number of iterations for the first and second level

9.2.2 Area Comparison and Throughput Estimation

In this section, we present an FPGA implementation of the proposed soft-input decoder and compare it with the hard-input decoder presented in [91]. The hard-input decoder uses algebraic decoding. It comprises of the syndrome calculation, the BMA, and the Chien search module. The soft input decoder is implemented as proposed in Section 6.1.2. It has two limitations. First, the length of the priority queue is limited to 64 elements. Furthermore, the accuracy of the metric calculation is limited to sixteen bits, and we use three-bit quantization of the input symbols.

The stack algorithm has a variable execution time depending on the error pattern. This algorithm needs at least 61 cycles to traverse the entire trellis if no error occurred. This case can be omitted by checking whether the syndrome of a column word is zero. If no error is detected the soft-decoding can be avoided and thus only a single cycle is needed.

Figure 9.3 depicts the average number of cycles needed for the stack algorithm. It shows the dependency between the channel bit error rate and the computational complexity, i.e. fewer errors lead to fewer decoding cycles. Note that the algebraic hard-input decoder needs four cycles for the first and six cycles for the second level. Hence, for high SNR ratios the stack algorithm requires an average number of cycles that is comparable with hard-input decoding.

Next we present FPGA synthesis result for the stack algorithm. The synthesis was performed with Xilinx Vivado and a Virtex-7 target device. Table 9.1 shows the number of slices and look-up tables (LUT) of the hard- and soft-input decoder with three-bit quantization. From these results we observe that the number of logic elements required for the stack algorithm is about 82% of the number of logic gates required for the GCC hard-input decoder.

Module	LUT	Slices
RS module (t=78)		
Syndrome	1 701	1 395
BMA	21 701	6 662
Forney alg.	1 046	729
Chien search	854	712
BCH Module (n=60,t=8)		
Syndrome	184	46
BMA	2 006	732
Chien search	1 557	240
re-image	148	336
Total	**29 197**	**10 852**
Stack alg.	**23 896**	**9 885**

Tab. 9.1: Results of an FPGA synthesis

9.2.3 Concluding Remarks and Future Directions

In this section, we have presented a soft input decoder for generalized concatenated codes. We have proposed a sequential decoding algorithm based on supercode trellises to reduce the complexity of the soft-input decoding compared to ordinary trellis based decoding. This implementation improves the error correction capability significantly compared with hard-input decoding. Consequently, the proposed method is a promising approach for soft-input decoding of GCC. The implementation of the stack algorithm is nine times large than the algebraic decoder. This complexity can be reduced by moving the majority of the register entries into read only memory. Nevertheless, the proposed soft-input decoding increases the overall complexity of the GCC decoder by 82% compared to the decoder presented in [92]. Other soft-input decoding methods for binary BCH codes may reduce the overall decoding complexity.

The proposed coding scheme is well suited for applications that require very low residual error rates, e.g. it might be appropriate for flash memories that provide soft information about the state of the memory cells. For flash memories, various concatenated coding schemes were proposed to enable soft-input decoding, e.g. product codes [75] and concatenated coding schemes based on trellis coded modulation and outer BCH oder RS codes [64, 41, 55]. For the presented simulation results we assumed a quantized additive white Gaussian noise channel. For practical flash memories, this model is not accurate. With multi-level cell and triple-level cell technologies, the reliability of the bit levels and cells varies and coding schemes were proposed that take these error characteristics into account [74, 73, 27, 26]. We believe that the adaptation of the proposed GCC coding scheme for flash memories is a promising direction for further research.

9.3 Chase Decoder

At this point we discuss the alternative to the stack decoding algorithm as mentioned in Chapter 6. In this scenario a SPC, a special case of the Chase algorithm, is applied in the first decoding level. The second and third levels are decoded using the Chase-II decoder.

The LLR values comprise one bit representing the sign and four bits that indicate the reliability. The least reliable bit position is determined by a sorting stage for the LLR values as

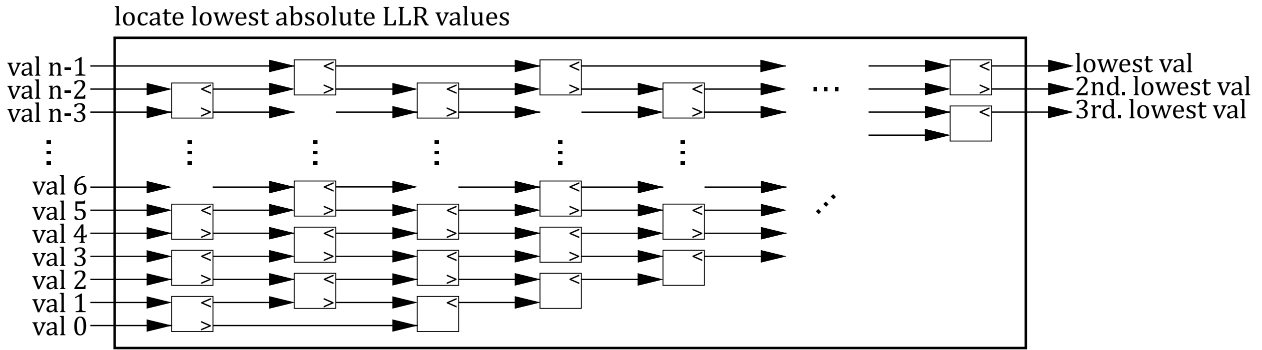


Fig. 9.4: Sorting stage to determine the positions of the least reliable bits. Reproduced by permission of the Institution of Engineering & Technology [101]

depicted in Figure 9.4. Because only the n minimal values are necessary, this sorting algorithm can be relaxed in two ways. First, the full sort is reduced in a partial and second the partial results are not necessarily sorted. Furthermore, for an efficient sorting the stream-like delivery of the LLR values in a real system should be considered. In [61] a partial sorting algorithm for streamed data is described that promises in conjunction with a more sophisticated sorting algorithm a much lower implementation complexity. The sorting stage determines the three least reliable bit positions (LRPs) which are required by the subsequent decoding rounds. Hence, the LRP values are also stored in the SRAM. The sorting stage comprises cascaded comparator devices that compare the magnitude of the two input LLR values and output the smaller value at the first output. To determine the bit positions each LLR value is associated with an index value. The output of the sorting stage are the three smallest LLR magnitudes and their indexes, i.e. the LRP values.

The second part of the soft-decoder is only required for levels $l > 0$. To generate the test patterns for the Chase decoding, the LRP values corresponding to the current column $b^{(l)}$ are read from the RAM. The pattern generation module generates eight test patterns that correspond to all possible bit flips for the three LRP values. The bit flipping is applied to the vector $b^{(l)}$ and the result is fed into the pipelined BCH hard decoder. The metric calculation stage calculates a metric value for each test pattern. The metric is the sum of all LLR magnitudes for all flipped bit positions, where bits may be flipped by the pattern generator as well as by the BCH decoder. The smallest metric value and the corresponding positions of the flipped bits are stored in the metric calculation module. These values are updated if a candidate with smaller metric value is found. The metric calculation considers only valid BCH codewords, where invalid codewords are detected by the erasure detection module based on the SPC and the degree of the error-locator polynomial. After all test patterns are decoded, the error correction module determines the final candidate codeword $\hat{b}^{(l)}$ by applying bit flipping to the bit positions that are stored in the metric module.

This Chase decoding procedure requires several cycles and is much slower than the hard-decoding of a column vector. However, only columns with non-zero syndrome have to be processed. If the soft-decoding mode is activated, the inner decoder receives already corrected columns from the first soft-decoding stage. These columns are stored in the codeword buffer and re-imaged for the outer decoder. In the subsequent levels the columns are passed to the soft-decoder module if the syndrome is non-zero. The resulting column codeword is inserted to the re-image stage. Therefore, the impact of the soft-decoding of levels $l > 0$ on the throughput is relatively low. Moreover, the reading process for the soft data is much slower than with hard-input data.

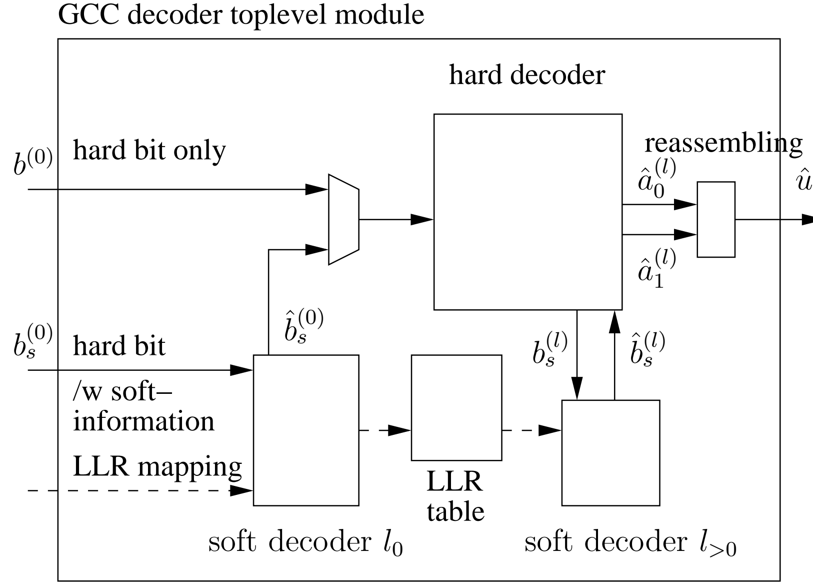


Fig. 9.5: Data flow of the GCC top level module. Reproduced by permission of the Institution of Engineering & Technology [101]

9.3.1 GCC Soft-decoder Architecture

In this section, we describe the overall hardware architecture for the proposed soft-decoder. Figure 9.5 shows the decoder top-level module. The decoder has two input data paths, $b^{(0)}$ is used for hard-decoding only and $b_s^{(0)}$ for codewords with soft information. The port width of the output $\hat{u}$ depends on the parallelization degree n_p of the inner decoder and is $n_p \cdot n_b$. It reads n_p columns every cycle. If the decoder is in hard-decoding mode, the data passes directly through the de-multiplexer into the hard decoder and finally results in estimated information $\hat{u}$. The hard decoder is loaded column-wise but the output is read row-wise and has to be reassembled in the correct order. This can be achieved by loading an intermediate buffer where in each level the column codes are extended with bit write enable lines. In the final level, the columns in the buffer are complete and the memory can be read column-wise.

The soft-decoding requires several cycles per column depending on the level. $b_s^{(0)}$ has a port width of $(n_s + n_i + 1) \cdot n_b$, where n_s is the number of soft bits per code bit and n_i the number of threshold identification bits. With TLC and MLC flash memories, the bit error probability varies from charge level to charge level and hence depends on the decision threshold. For MLC one bit and for TLC two bits are required to indicate the threshold.

In the soft-decoding mode the data passes through the first soft-decoder level $l = 0$. During this first decoding stage the soft and identification bits from the flash memory have to be mapped to LLR. These LLR values are needed for levels with soft-decoding. Hence, the LLR values are stored in a RAM. Soft-decoding of the single parity-check code in the first level is a simple bit-flipping procedure. If the parity of the column is odd, the least reliable bit position in this column is flipped. This bit position corresponds to the LLR value with the smallest magnitude. The LLR values have a resolution of $n_s = 4$ bits per code bit. Hence, the position of the least reliable bit might not be unique. Consequently, the bit flipping is only applied if the parity is odd and the least reliable position is unique. If the position is not unique and the parity is odd, a failure is declared. The hard decoder for BCH codes is only required for levels $l > 0$. Moreover, only columns with non-zero syndrome have to be processed.

module name	LUT
decoder total	92 509
2x inner decoder	16 422
syndrome calculation	732
pipelined BMA	9 350
Chien search	3 590
re-image	511
re-encoder	416
GCC buffer control logic	833
outer decoder	50 114
syndrome calculation	3 001
UPIBA	10 835
Chien search	4 359
Forney	3 059
erasure stack control logic	28
error counter	10
multiplier pool	28 725
soft-decoder	8 899
control logic	923

Tab. 9.2: FPGA utilization in LUT

purpose	type	width	depth
6x GCC buffers	SP	$n_b = 109$	$\left\lceil \frac{\lceil \frac{n_a}{\theta_b} \rceil}{3} \right\rceil = 56$
RS buffer	DP	$\theta_b m_a = 18$	$\lceil \frac{n_a}{\theta_b} \rceil = 167$
2x erasure stack	SP	$m_a = 9$	$max(d_a) = 222$
UPIBA	SP	$m_a = 9$	$max(d_a) = 222$
2x reassembling	SP&bwe	$k_b = 108$	$\lceil \frac{n_a}{\theta_b} \rceil = 167$

Tab. 9.3: GCC buffer dimensions and type

9.3.2 Synthesis Result and Throughput Estimation

The proposed decoder GCC with Chase-II soft-decoder was implemented in Verilog and synthesized using Synopsys Protocompiler for a Xilinx Virtex 7 FPGA xc7vx690t. Tab. 9.2 shows the number of LUTs for the complete decoder and its submodules. There is a small difference in the number of LUTs for the inner codes, and thus we list the arithmetic mean between the two instances.

Additionally, buffers are needed that are implemented either as SP-SRAM or DP-SRAM (see Section 8.2). Each of the two GCC buffers comprises three SP-SRAM, where two are in read mode while one can be written. The buffer for the UPIBA and the two erasure stack buffers are SP-SRAM, the RS buffer is the only DP-SRAM. The relation between code parameters and SRAM dimensions is shown in Tab. 9.3.

Example 17. *Tab. 9.2 and Tab. 9.3 show the decoder implementation size for a code that can decode in hard-input mode a channel down to* $SNR \geq 6.5dB$ *and in soft mode down to* $SNR \geq 5.5$*. In both cases* $FER \leq 10^{-16}$ *can be achieved. The code has* $L = 12$ *levels and* $n_a = 334$ *columns.*

At this point, we describe the number of cycles for the modules that are necessary to

decode a codeword. Subsequently, we explain the mean number of cycles per block depending on the error probability and provide an example.

The cycles that are necessary to decode a codeword can be divided into two components which vary depending on the inner and outer codeword distances $d_b^{(l)}$ and $d_a^{(l)}$ of each level. The first component is the pipeline formation consisting of the complete inner decoder and parts of the outer decoder. First we consider the number of cycles of the pipeline formation. The latency of the column decoder is one cycle for each step: the syndrome calculation, Chien search, re-imaging and re-encoding. In addition, $d_b^{(l)}$ cycles are needed to determine $\sigma_b(x)$. The RS decoder components that are part of this pipeline are the RS syndrome calculation with a latency of one cycle as well as the error position and error magnitude calculation with a latency of three cycles. The processing of all columns takes n_a cycles plus the pipeline latency.

The Blahut algorithm is operating while the pipeline is stopped. It can only start when all syndrome values $S_a(x)$ are ready. Depending on the level l, UPIBA needs $2(d_a^{(l)}+2)$ cycles for the calculation of $\sigma_a(x)$ and $\hat{\Omega}_a(x)$ polynomials if $S_a(x)$ is non-zero. Otherwise the UPIBA is skipped and the pipeline process is continued seamlessly. Based on this estimation we can determine the number of cycles τ_h for the hard-decoding.

$$\tau_h = \left\lceil \frac{n_b(L+1)}{\theta_b} \right\rceil + 2\sum_{l=0}^{L-1}(d_a^{(l)}+2)p_{b,l} + \sum_{l=1}^{L-1} d_b^{(l)} + 8(L-2) \tag{9.1}$$

where $p_{b,l}$ is the probability that $S_a(x)$ is non-zero in level l.

The proposed soft-input decoder comprises two different modules that have different time properties. The first parity-check bit-flipping algorithm adds a constant of seven cycles per column. In the next two levels, the soft-decoder takes twelve cycles, which are only needed if $S_b(x)$ is non-zero. In the soft decoded levels, the BMA and Chien search are skipped. Thus, we can determine the soft-decoder cycle time τ_s.

$$\begin{aligned}\tau_s \;=\; & \left\lceil \frac{n_b(L+1)}{\theta_b} \right\rceil + \sum_{l=0}^{L-1}(d_a^{(l)}+2)p_{b,l} \\ & +7(L-2) + 7n_b + 12n_a(1-(1-P_b)^{n_b}) \end{aligned} \tag{9.2}$$

Figure 9.6 presents the achievable throughput in Gbit/s versus SNR.

9.4 Summary

In this chapter, a architecture has been shown that discusses how a hard-decoder can be extended by a soft-decoder. As mentioned, it is only useful to decode the lower inner-code levels. We showed a solution using the stack decoding algorithm that is suitable to decode short column codes. For larger column codes the Chase decoder is the better choice. In both cases, the LLR values are necessary to form the metric. For 3D Flash memories high code rates R are expected and thus GCC codes with larger column sizes are required. Subsequently, a Flash controller will demand the Chase decoder solution.

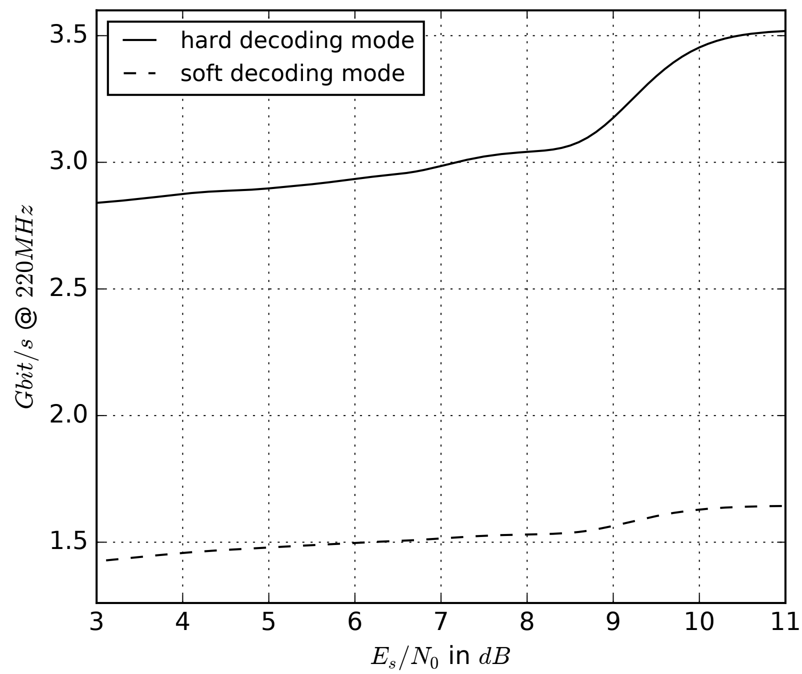


Fig. 9.6: Throughput versus SNR. Reproduced by permission of the Institution of Engineering & Technology [101]

Chapter 10

Conclusion

Flash memories are important non-volatile storage media that have many applications from consumer products to industrial storage solutions. However, flash memories require error correction methods to guarantee data reliability. This thesis has investigated error correction techniques suitable for flash memories focusing on applications that require very low residual error rates, e.g. word error rates below 10^{-16}.

The advent of flash memories that provide reliability information about the cell status led to the introduction of LDPC codes and soft-input decoding for flash memories. However, guaranteeing word error rates as low as 10^{-16} with LDPC codes is extremely difficult or infeasible. In this thesis, we have proposed a different approach that is based on generalized concatenated codes. These codes enable low complexity decoding methods, soft-input decoding, and an analysis of the error correction performance for arbitrary low residual error probabilities.

We have explained the GC code construction and have developed an approach to determine suitable code parameters. A major result of this work is a detailed description of the decoding scheme of GC codes. We have presented a complete design for the ECC unit in a flash memory controller for industrial applications. The decoder achieves a high throughput in the case where only hard decoding is performed. If a decoding failure occurs, the received word can be decoded by a soft-decoder unit in a second pass.

Today's flash controllers must support different code types, e.g. the decoding of long BCH codes is required to support older flash technologies. Future research could exploit potential synergistic effects of the algebraic decoders for BCH and GC codes. BCH and GC codes use similar algorithms. Multipliers, adders, and registers can be shared between the BCH decoder and the outer RS component decoders of the GC code. To quantify these effects further investigations on such a reconfigurable hardware are necessary.

The results of this thesis led into two patents [US20170331498A1, US20170331499A1]. Furthermore, Hyperstone GmbH decided to implement the GCC as an additional ECC in their micro controller product X1.

Bibliography

[1] *Solid-State Drive (SSD) Requirements and Endurance Test Method (JESD218)*. JEDEC SOLID STATE TECHNOLOGY ASSOCIATION, 2010.

[2] L.E. Aguado and P.G. Farrell. On hybrid stack decoding algorithms for block codes. *Information Theory, IEEE Transactions on*, 44(1):398–409, Jan 1998.

[3] Rohit P Ambekar, David B Bogy, Qing Dai, and Bruno Marchon. Critical clearance and lubricant instability at the head-disk interface of a disk drive. *Applied Physics Letters*, 92(3):033104, 2008.

[4] R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[5] M. Blaum, J. Bruck, and A. Vardy. Interleaving schemes for multidimensional cluster errors. *IEEE Transactions on Information Theory*, 44(2):730–743, 1998.

[6] Martin Bossert. *Channel coding for telecommunications*. Wiley, 1999.

[7] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, 105(9):1666–1704, Sept 2017.

[8] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1285–1290, March 2013.

[9] Hsie-Chia Chang and C.B. Shung. New serial architecture for the Berlekamp-Massey algorithm. *IEEE Transactions on Communications*, 47(4):481–483, April 1999.

[10] Hsie-Chia Chang and C.B. Shung. New serial architecture for the Berlekamp-Massey algorithm. *IEEE Transactions on Communications*, pages 481– 483, April 1999.

[11] D Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, pages 170–182, 1972.

[12] R. T. Chien. Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes. In *IEEE Trans. Inform. Theory*, 1964.

[13] S. Cho, Daesung Kim, Jinho Choi, and Jeongseok Ha. Block-wise concatenated BCH codes for NAND flash memories. *IEEE Transactions on Communications*, 62(4):1164–1177, April 2014.

[14] C. De Almeida and Jr. Palazzo, R. Efficient two-dimensional interleaving technique by use of the set partitioning concept. *Electronics Letters*, 32(6):538–540, 1996.

[15] D Divsalar and JH Yuen. Performance of concatenated reed-solomon/viterbi channel coding. In *The Telecommunications and Data Acquisition Report*, 1982.

[16] L. Dolecek and Y. Cassuto. Channel coding for nonvolatile memory technologies: Theoretical advances and practical considerations. *Proceedings of the IEEE*, 105(9):1705–1724, Sept 2017.

[17] Guiqiang Dong, Ningde Xie, and Tong Zhang. On the use of soft-decision error-correction codes in NAND Flash memory. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(2):429–439, Feb 2011.

[18] Wolfram Drescher. *Schaltungsanordnung zur Galoisfeld-Arithmetik.* Dissertation TU-Dresden, 2005.

[19] I. Dumer. *Concatenated codes and their multilevel generalizations.* in Handbook of Coding Theory, Vol. II, Elsevier, Amsterdam, 1998.

[20] A. Fahrner, H. Griesser, R. Klarer, and V.V. Zyablov. Low-complexity GEL codes for digital magnetic storage systems. *IEEE Transactions on Magnetics*, 40(4):3093–3095, July 2004.

[21] Achim Fahrner. *On Signal Processing and Coding for Digital Magnetic Storage Systems.* VDI-Verlag, 2005.

[22] G David Forney. Concatenated codes, 1966.

[23] J. Freudenberger, F. Ghaboussi, and S. Shavgulidze. New coding techniques for codes over Gaussian integers. *IEEE Transactions on Communications*, 61(8):3114–3124, Aug 2013.

[24] J. Freudenberger, F. Ghaboussi, and S. Shavgulidze. Set partitioning and multilevel coding for codes over Gaussian integer rings. In *9th International ITG Conference on Systems, Communications and Coding (SCC), Munich*, pages 1–5, Jan 2013.

[25] J. Freudenberger, M. Rajab, and S. Shavgulidze. Estimation of channel state information for non-volatile flash memories. In *IEEE 7th International Conference on Consumer Electronics (ICCE)*, Sept 2017.

[26] R. Gabrys, F. Sala, and L. Dolecek. Coding for unreliable flash memory cells. *IEEE Communications Letters*, 18(9):1491–1494, Sept 2014.

[27] R. Gabrys, E. Yaakobi, and L. Dolecek. Graded bit-error-correcting codes with applications to flash memory. *IEEE Transactions on Information Theory*, 59(4):2315–2327, April 2013.

[28] S. Gerardin, M. Bagatin, A. Paccagnella, K. Grürmann, F. Gliem, T. R. Oldham, F. Irom, and D. N. Nguyen. Radiation effects in flash memories. *IEEE Transactions on Nuclear Science*, 60(3):1953–1969, June 2013.

[29] S.W. Golomb, R. Mena, and Wen-Qing Xu. Optimal interleaving schemes for two-dimensional arrays. *IEEE Transactions on Information Theory*, 52(9):4223–4229, 2006.

[30] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen. Flexlevel nand flash storage system design to reduce ldpc latency. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(7):1167–1180, July 2017.

[31] D. h. Lee and W. Sung. Estimation of nand flash memory threshold voltage distribution for optimum soft-decision error correction. *IEEE Transactions on Signal Processing*, 61(2):440–449, Jan 2013.

[32] K. Haymaker and C. A. Kelley. Structured bit-interleaved LDPC codes for MLC flash memory. *IEEE Journal on Selected Areas in Communications*, 32(5):870–879, May 2014.

[33] K Huber. Codes over Gaussian integers. *IEEE Transactions on Information Theory*, pages 207–216, 1994.

[34] A. Huebner, J. Freudenberger, R. Jordan, and M. Bossert. Irregular turbo codes and unequal error protection. In *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No.01CH37252)*, pages 142–, 2001.

[35] Yuan Jiang. *A practical guide to error-control coding using Matlab*. Artech House, 2010.

[36] D. Kim and J. Ha. Quasi-primitive block-wise concatenated bch codes with collaborative decoding for nand flash memories. *IEEE Transactions on Communications*, 63(10):3482–3496, Oct 2015.

[37] D. Kim and J. Ha. Serial quasi-primitive bc-bch codes for nand flash memories. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[38] S. Korkotsides, G. Bikas, E. Eftaxiadis, and T. Antonakopoulos. BER analysis of MLC NAND flash memories based on an asymmetric pam model. In *2014 6th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 558–561, May 2014.

[39] Hanho Lee. A high-speed low-complexity Reed-Solomon decoder for optical communications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(8):461–465, 2005.

[40] H. Li. Modeling of threshold voltage distribution in NAND flash memory: A monte carlo method. *IEEE Transactions on Electron Devices*, 63(9):3527–3532, Sept 2016.

[41] S. Li and T. Zhang. Improving multi-level NAND flash memory storage reliability using concatenated BCH-TCM coding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(10):1412–1420, Oct 2010.

[42] Shu Lin and Daniel J. Costello. *Error Control Coding*. Upper Saddle River, NJ: Prentice-Hall, 2004.

[43] Wei Lin, Shao-Wei Yen, Yu-Cheng Hsu, Yu-Hsiang Lin, Li-Chun Liang, Tien-Ching Wang, Pei-Yu Shih, Kuo-Hsin Lai, Kuo-Yi Cheng, and Chun-Yen Chang. A low power and ultra high reliability LDPC error correction engine with digital signal processing for embedded nand flash controller in 40nm coms. In *Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2, June 2014.

[44] Wei Liu, Junrye Rho, and Wonyong Sung. Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories. In *IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS)*, pages 303 –308, oct. 2006.

[45] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. North Holland Publishing Co., June 1988.

[46] Alan B. Marchant. *Optical recording: a technical overview*. Addison-Wesley, Reading, Mass. [u.a.], 1990.

[47] C. Martinez, R. Beivide, and E. Gabidulin. Perfect codes for metrics induced by circulant graphs. *IEEE Transactions on Information Theory*, 53(9):3042–3052, 2007.

[48] J Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 1969.

[49] J. Massey. Variable-length codes and the Fano metric. *IEEE Transactions on Information Theory*, 18(1):196–198, 1972.

[50] Erwin R. Meinders. *Optical data storage: phase-change media and recording*, volume 4. Springer, Dordrecht, 2006.

[51] R. Micheloni, A. Marelli, and R. Ravasio. *Error Correction Codes for Non-Volatile Memories*. Springer, 2008.

[52] Rino Micheloni and Luca Crippa. *Solid-State-Drives (SSDs) Modeling*. Springer, 2017.

[53] V Yasuyuki Goto V Norikazu Nakamura, V Akiyo Mizutani, and V Hiroshi Chiba V Keiji Watanabe. Head disk interface technologies for high recording density and reliability. *Fujitsu Sci. Tech. J*, 42(1):113–121, 2006.

[54] A. Neubauer, J. Freudenberger, and V. Kühn. *Coding Theory: Algorithms, Architectures and Applications*. John Wiley & Sons, 2007.

[55] J. Oh, J. Ha, J. Moon, and G. Ungerboeck. Rs-enhanced TCM for multilevel flash memories. *IEEE Transactions on Communications*, 61(5):1674–1683, May 2013.

[56] ONFI. Open nand flash interface specification revision 3.0. *ONFI Workgroup, Published Mar*, 15:288, 2011.

[57] Jeong-In Park, Kihoon Lee, Chang-Seok Choi, and Hanho Lee. High-speed low-complexity Reed-Solomon decoder using pipelined Berlekamp-Massey algorithm and its folded architecture. *Journal of semiconductor technology and science*, 10(3):193–202, 2010.

[58] V. Pavlushkov, R. Johannesson, and V. V. Zyablov. Unequal error protection for convolutional codes. *IEEE Transactions on Information Theory*, 52(2):700–708, Feb 2006.

[59] S. Plass, A. Dammann, G. Richter, and M. Bossert. Channel correlation properties in OFDM by using time-varying cyclic delay diversity. *Journal on Communications*, 3(3):19–26, 2008.

[60] Antonios Prodromakis, Stelios Korkotsides, and Theodore Antonakopoulos. MLC NAND flash memory: Aging effect and chip/channel emulation. *Microprocessors and Microsystems*, 39(8):1052 – 1062, 2015.

[61] A. Rjabov. Hardware-based systems for partial sorting of streaming data. In *2016 15th Biennial Baltic Electronics Conference (BEC)*, pages 59–62, Oct 2016.

[62] V. Sorokine and F.R. Kschischang. A sequential decoder for linear block codes with a variable bias-term met ric. *IEEE Transactions on Information Theory*, 44(1):410–416, 1998.

[63] Alessandro S. Spinelli, Christian Monzio Compagnoni, and Andrea L. Lacaita. Reliability of NAND flash memories: Planar cells and emerging issues in 3D devices. *Computers*, 6(2), 2017.

[64] F. Sun, S. Devarajan, K. Rose, and T. Zhang. Design of on-chip error correction systems for multilevel NOR and NAND flash memories. *IET Circuits, Devices Systems*, 1(3):241 –249, June 2007.

[65] B. Tepekule, U. Yavuz, and A.E. Pusane. On the use of modern coding techniques in QR applications. In *21st Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2013.

[66] Chenxu Wang, Yuhong Gao, Liang Han, and Jinxiang Wang. The design of parallelized BCH codec. *3rd International Congress on Image and Signal Processing (CISP)*, 2010.

[67] H. Wang, T. Y. Chen, and R. D. Wesel. Histogram-based flash channel estimation. In *IEEE International Conference on Communications (ICC)*, pages 283–288, June 2015.

[68] Jiadong Wang, K. Vakilinia, Tsung-Yi Chen, T. Courtade, Guiqiang Dong, Tong Zhang, H. Shankar, and R. Wesel. Enhanced precision through multiple reads for ldpc decoding in flash memories. *IEEE Journal on Selected Areas in Communications*, 32(5):880–891, May 2014.

[69] L. Weiburn and J.K. Cavers. Improved performance of Reed-Solomon decoding with the use of pilot signals for erasure generation. In *Vehicular Technology Conference, 1998. VTC 98. 48th IEEE*, volume 3, pages 1930–1934 vol.3, May 1998.

[70] J. Wolf. Efficient maximum likelihood decoding of linear block codes using a trellis. *IEEE Transactions on Information Theory*, 24(1):76–80, January 1978.

[71] C. J. Wu, H. T. Lue, T. H. Hsu, C. C. Hsieh, W. C. Chen, P. Y. Du, C. J. Chiu, and C. Y. Lu. Device characteristics of single-gate vertical channel (SGVC) 3D NAND flash architecture. In *IEEE 8th International Memory Workshop (IMW)*, pages 1–4, May 2016.

[72] Yingquan Wu. New scalable decoder architectures for Reed-Solomon codes. *IEEE Transactions on Communications*, 63(8):2741–2761, 2015.

[73] E. Yaakobi, L. Grupp, P.H. Siegel, S. Swanson, and J.K. Wolf. Characterization and error-correcting codes for TLC flash memories. In *International Conference on Computing, Networking and Communications (ICNC)*, pages 486–491, Jan 2012.

[74] E. Yaakobi, Jing Ma, L. Grupp, P.H. Siegel, S. Swanson, and J.K Wolf. Error characterization and coding schemes for flash memories. In *IEEE GLOBECOM Workshops*, pages 1856–1860, December 2010.

[75] C. Yang, Y. Emre, and C. Chakrabarti. Product code schemes for error correction in MLC NAND flash memories. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(12):2302–2314, Dec 2012.

[76] Xinmiao Zhang and K K Parhi. High-speed architectures for parallel long BCH encoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(7):872–877, 2005.

[77] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 243–256, San Jose, CA, 2013. USENIX.

[78] I. Zhilin and A. Kreschuk. Generalized concatenated code constructions with low overhead for optical channels and nand-flash memory. In *2016 XV International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY)*, pages 177–180, Sept 2016.

[79] L. Zuolo, C. Zambelli, A. Marelli, R. Micheloni, and P. Olivo. LDPC soft decoding with improved performance in 1x-2x MLC and TLC NAND flash-based solid state drives. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017.

[80] Victor Zyablov, Sergo Shavgulidze, and Martin Bossert. An introduction to generalized concatenated codes. *European Transactions on Telecommunications*, 10(6):609–622, 1999.

Publications of the author

[81] J. Freudenberger, U. Kaiser, and J. Spinner. Concatenated code constructions for error correction in non-volatile memories. In *Int. Symposium on Signals, Systems, and Electronics (ISSSE), Potsdam*, pages 1–6, Oct 2012.

[82] J. Freudenberger and J. Spinner. Mixed serial/parallel hardware implementation of the Berlekamp-Massey algorithm for BCH decoding in flash controller applications. In *Int. Symposium on Signals, Systems, and Electronics (ISSSE), Potsdam*, pages 1–5, Oct 2012.

[83] J. Freudenberger and J. Spinner. A configurable Bose-Chaudhuri-Hocquenghem codec architecture for flash controller applications. *Journal of Circuits, Systems, and Computers*, 23(2):1–15, Feb 2014.

[84] J. Freudenberger, J. Spinner, and S. Shavgulidze. Generalized concatenated codes for correcting two-dimensional clusters of errors and independent errors. In *Int. Conference on Communication and Signal Processing (CSP), Castelldefels-Barcelona*, pages 1–5, February 2014.

[85] J. Freudenberger, J. Spinner, and S. Shavgulidze. Set partitioning of Gaussian integer constellations and its application to two-dimensional interleaver design. In *Int. Conference on Communication and Signal Processing (CSP), Castelldefels-Barcelona*, pages 1–5, February 2014.

[86] J. Freudenberger, J. Spinner, and S. Shavgulidze. Set partitioning of Gaussian integer constellations and its application to two-dimensional interleaving. *IET Communications*, 8(8):1336–1346, May 2014.

[87] J. Freudenberger, T. Wegmann, and J. Spinner. An efficient hardware implementation of sequential stack decoding of binary block codes. In *IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 135–138, Sept 2015.

[88] J. Spinner and J. Freudenberger. Hardware-Entwurf einer flexiblen Fehlerkorrektureinheit für Flashspeicher. In *MPC-Workshop, Aalen*, pages 33–40, Jul 2012.

[89] J. Spinner and J. Freudenberger. Generierung von Code-Komponenten für BCH Encodierer. In *MPC-Workshop, Offenburg*, pages 43–46, Feb 2012.

[90] J. Spinner and J. Freudenberger. Semi-automatic source code generation for the hardware-implementation of a parallelizable and configurable error correction unit. In *8th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Aachen*, pages 1–4, Jun 2012.

[91] J. Spinner and J. Freudenberger. Design and implementation of a pipelined decoder for generalized concatenated codes. In *Proceedings of 27th Symposium on Integrated Circuits and Systems Design (SBCCI), Aracaju, Brazil*, pages 1–16, Sept 2014.

[92] J. Spinner and J. Freudenberger. Decoder architecture for generalized concatenated codes. *IET Circuits, Devices & Systems*, 9(5):328–335, 2015.

[93] J. Spinner and J. Freudenberger. Soft input decoding of generalized concatenated codes using a stack decoding algorithm. In *Proceedings of 2nd BW-CAR Symposium on Information and Communication Systems (SInCom)*, pages 1–5, Dec 2015.

[94] J. Spinner and J. Freudenberger. A decoder with soft decoding capability for high-rate generalized concatenated codes with applications in non-volatile flash memories. In *Proceedings of 30th Symposium on Integrated Circuits and Systems Design (SBCCI), Fortaleza, Brazil*, Sept. 2017.

[95] J. Spinner, J. Freudenberger, C. Baumhof, A. Mehnert, and R. Willems. A BCH decoding architecture with mixed parallelization degrees for flash controller applications. In *IEEE 26th International SOC Conference (SOCC), Erlangen*, pages 116–121, Sept 2013.

[96] J. Spinner, J. Freudenberger, and S. Shavgulidze. Sequential decoding of binary block codes based on supercode trellises. In *1st BW-CAR Symposium on Information and Communication Systems (SInCom)*, pages 25–28, Nov 2014.

[97] J. Spinner, J. Freudenberger, and S. Shavgulidze. A soft input decoding algorithm for generalized concatenated codes. *IEEE Transactions on Communications*, 64(9):3585–3595, Sept 2016.

[98] J. Spinner, M. Rajab, and J. Freudenberger. Construction of high-rate generalized concatenated codes for applications in non-volatile flash memories. In *2016 IEEE 8th International Memory Workshop (IMW)*, pages 1–4, May 2016.

[99] J. Spinner and J. Freudenberger. Eine effiziente Dekoderarchitektur für verallgemeinert verkettete Codes. *52. MPC-Workshop, Künzelsau*, pages 27–31, Jul 2014.

[100] J. Spinner and J. Freudenberger. Datenrettung für die Speicherkarte – Fehlerkorrekturverfahren für Flash-Speicher. *horizonte*, 45:18–20, 2015.

[101] J. Spinner, D. Rohweder, and J. Freudenberger. Soft input decoder for high-rate generalised concatenated codes. *IET Circuits, Devices & Systems*, 2018.

Patents

- US20170331498A1, J. Freudenberger, C. Baumhof, and J. Spinner, Method and device for error correction coding based on high-rate generalized concatenated codes, 2017
- US20170331499A1, J. Freudenberger, J. Spinner, and C. Baumhof, Method and decoder for soft input decoding of generalized concatenated codes, 2017