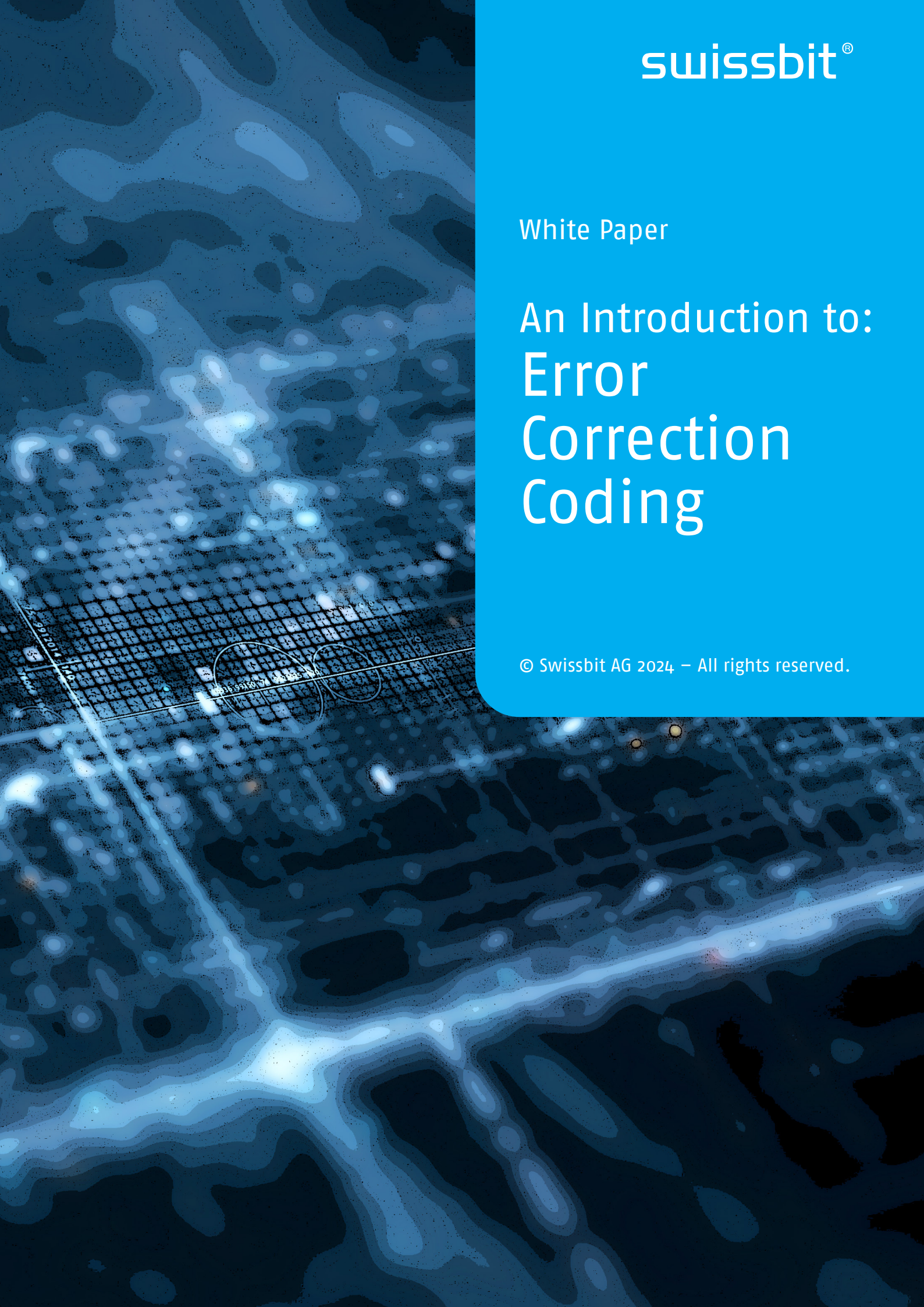

swissbit®

White Paper

An Introduction to: Error Correction Coding

swissbit®

1. Abstract

This paper explains the basic concepts behind error correction and highlights a brief history of different error correction codes discussing and alaysing the advantages and disadvantages of the following three methods most common to NAND flash technology.

The Bose-Chaudhuri-Hocquenghem (BCH) codes are discussed in detail, as they are used in flash memory controllers for SLC and MLC NAND flash. Low-Density Parity-Check (LDPC) codes are used today for TLC and QLC flash, which requires more powerful codes. Finally Generalized Concatenated (GC) codes are dicussed as this method found exclusively in Swissbit modules is especially suited for industrial storage or applications requiring high reliability.

Table of Contents

2. Introduction

As the integration size of NAND flash memory shrinks and more bits can be stored in a cell, the error correction requirements for flash memory controllers are increasing with each new generation of flash.

Error correction has become one of the most important tasks of flash memory controllers. For quite a number of years, error correction based on Bose-Chaudhuri-Hocquenghem (BCH) codes was the prevalent choice. Yet, as flash error rates continue to grow with higher density flashes, the error correction capability of the controllers has to be increased. This has become increasingly difficult when considering cost, area and power limitations.

3. History of Error Correction Codes

In the 1940s Richard Hamming developed the Hamming Code. At the time the computers stored information on punch cards. However, this system was error-prone, because it was common for cards to get bent or miss-punched, causing flipped bits and a failing system.

Hamming searched for a solution with parity bits. A parity bit (or check bit), is a bit added to a string of binary code to ensure that the total number of 1-bits in the string is even or odd. In the event of a single bit error, one parity bit added to the end of the message allows the system to detect the error because the parity bit no longer matches the sequence. While this allows detecting an error, it could not identify the exact error location and correct it.

Hamming found a method to also correct a single bit error by adding three parity bits to four bits of data, introducing the so-called 7:4 Hamming code. In Fig. 1 the basic principle is outlined. Each parity bit (P1, P2, P3) is related to three data bits, shown in the same circle. An incorrect data bit will cause at least two parity bits not to match the data. By considering which parity bits do not fit, the incorrect data bit can be located and corrected. In case only one parity bit does not match, the parity bit itself is the one bit that has flipped during transmission and therefore should be corrected to its inverse value.

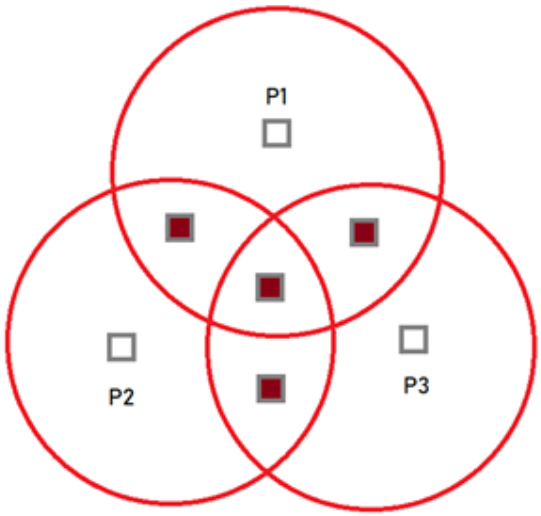

Figure 1: Basic outline of Hamming's 7:4 ECC method to identify and correct bit errors by using 3 parity bits for each block of 4 bits.

Further milestones through the history of error correction were:

1949 - Golay Code
The binary Golay code is a type of linear error correcting code used in digital communications which was introduced by the work of Golay [1].

1954 - Reed-Muller Code
The Reed-Muller codes are a family of linear error-correcting codes developed by Irving S. Reed and David E. Muller [2].

1959-1961 - Bose–Chaudhuri–Hocquenghem (BCH)
This code was invented by French Mathematician Alexis Hocquenghem and independently by Raj Bose and Dr. Ray-Chaudhuri. These codes are the most important cyclic block codes.

1960 - Reed-Solomon (RS) Code
Reed-Solomon codes (RS codes for short) are a class of cyclic block codes. They are used in channel coding to detect and correct transmission or memory errors as part of a forward error correction. They are a special case of no binary BCH-Codes [3].

1960 - Low-Density Parity-Check (LDPC) Code
Also known as the Gallager Codes which were developed by Robert G. Gallager in his doctoral dissertation [4].

1967-1969 - Berlekamp Massey Algorithm
A decoding algorithm for large distance RS code invented by Elwyn Berlekamp and James Massey.

1990-1991 - Turbo Codes
Turbo codes were developed in the early 90s and were the first practical codes to closely approach the channel capacity, a theoretical maximum for the code rate at which reliable communication is still possible given a specific noise level. Revolutionary for the times, many experts in the field did not believe reported results. Only when the performance was confirmed did engineers start considering other types of iterative signal processing.

Before Turbo Codes emerged, the most efficient ECC was a serial concatenation code RSV based on an outer Reed-Solomon ECC combined with an inner Viterbi decoded short constraint length convolutional code. Turbo codes are used extensively in telecommunications, specifically 3G, 4G and LTE as well as NASA missions.

4. Abbreviations and Concepts

The bit error rate (BER) is defined as

$$BER = \frac{Bit\ errors\ read\ from\ flash}{Total\ number\ of\ bits\ to\ be\ read}$$

gle bit being ıending on the flash memory, its technology, temperature, age and use. The uncorrectable bit error rate is defined as

$$UBER = \frac{Total\ number\ data\ errors}{Total\ number\ bits\ read} = \frac{UBLER\ x\ (number\ LBAs\ per\ ECC\ frame)}{ECC\ frame\ size}$$

uncorrectable block error rate (UBLER), which again is also the probability of a block or frame error is defined as:

$$UBLER = \frac{Number\ of\ uncorrectable\ ECC\ frames(or\ blocks\ or\ codewords)}{Total\ number\ of\ read\ frames}$$

performance of the error correction.

5. Bose-Chaudhuri-Hocquenghem Codes

In coding theory, Bose-Chaudhuri-Hocquenghem (BCH) codes form a class of cyclic codes that are constructed using polynomials over a finite field. It is the best algorithm with respect to correcting binary-symmetric channel (BSC) errors.

A key feature of BCH codes is that they allow control over the number of symbol errors correctable by the code. It is even possible to design binary BCH codes that can correct multiple bit errors without extreme computing power. BCH error correction decodes with ease through an algebraic method known as syndrome decoding. This not only simplifies the design of the decoder but also allows for small low power electronic hardware.

With a relatively standard implementation, BCH codes were most commonly found in satellite communications, compact disc players, DVDs, and Solid State Drives until TLC and QLC technology became widespread with 3D flash.

Technical Background

The Decoding of BCH code is performed in three main steps:

- The Syndrome is calculated from the received codeword.
- Error location polynomial is determined by solving the so-called key equation. The equation can be solved by the BMA-Algorithm. (Berlekamp-Massey Algorithm)
- Error locations are identified by exhaustive search. The most commonly used algorithm is called Chien Search.

After these steps, the errors can be corrected. Essentially, the minimum code distance determines the correction capability.

Advantages of BCH Codes

- BCH codes have a relatively straight forward implementation
- They allow for control over the number of symbol errors correctable by the code
- Decode with ease, which in turn, allows for low power electronic hardware

Disadvantages of BCH Codes

- Soft decoding algorithms (e.g. Viterbi, stack algorithm) for BCH codes have a very high computational complexity
- Bad code rate for higher error correction capabilities

5. Low-Density Parity Check Code

Developed in the 1960s, low-density parity-check (LDPC) codes, which are also known as Gallager codes, in honor of Robert G. Gallager, who developed the concept in his doctoral dissertation at the Massachusetts Institute of Technology, were impractical to implement at that time [4].

LDPC codes are a class of linear block codes for error correction. After they had been almost forgotten, a publication in the year 2001 demonstrated that they could operate close to the Shannon limit and that they could be efficiently implemented as irregular LDPC codes [5]. This resulted in an increase in popularity leading to their presence in several communication standards, like e.g. DVB-S2 and DV-T2 where an LDPC code is wrapped by an outer error correction – for the two mentioned standards that is a BCH code, to correct sporadic errors that are made by the LDPC decoder.

Technical Background

LDPC has become the flash industry´s term for the most powerful error correction codes and it is assumed that a flash memory controller featuring an LDPC-based error correction will always be able to provide the maximum performance in terms of correction. This is simply not the case. Just as it is the case for BCH codes, an LDPC code is not a single code but a family or class of codes.

In these classes of codes, there are hundreds of different codes that can remarkably differ in their correction capability. For example, BCH codes can be designed to correct 20 bit-errors in a 2K block of data, which is a rather poor performance, and BCH codes can be designed to correct 120 bit-errors in a 2K block of data, a very respectable performance. This illustrates that it is not sufficient to mention just the class of code, BCH, but also the performance in respect to what amount of erroneous bits can be corrected in a single codeword. The same applies for an LDPC code.

The class of code highlights the underlying principles and its construction, but it does not reveal its performance. This can be instantly recognized in Fig. 2, where the performance of different LDPC codes is shown for a limited signal-to-noise ratio via simulation.

Fig. 2 indicates that there are LDPC codes with excellent performance and LDPC codes with poor performance. Therefore, just relying on the term LDPC to expect a stellar performance is a misinterpretation of the term.

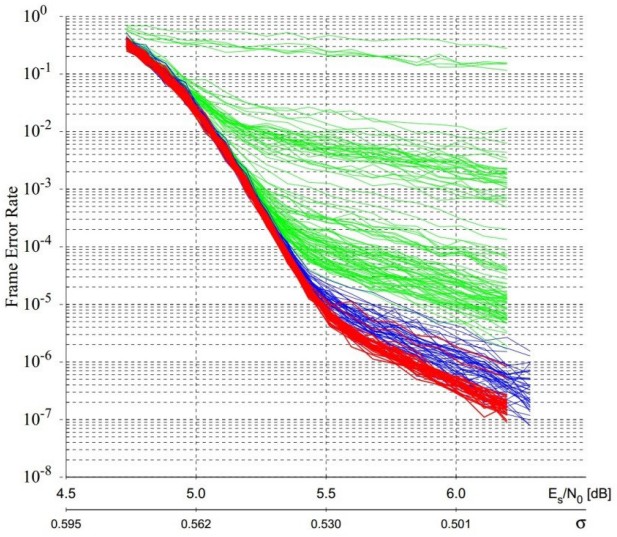

Figure 2: Frame error rates over signal-to-noise ratio for different LDPC implementations for short codewords. The colors show different optimizations: the random graphs (green), girth optimized (blue), and neighborhood optimized (red) graphs are all of identical degree structure [6]..

In contrast to a BCH code, the performance in terms of correctable bits in a codeword is very hard to specify. For the BCH code, this value can be calculated and is valid for all codewords – regardless of the error distribution. Unlike for the BCH code, the amount of errors an LDPC code can correct in a codeword is varying from codeword to codeword. As a consequence, there might be codewords where an LDPC code is able to correct 200 bit-errors in a 2K codeword and there might be a codeword where an LDPC code can only correct 40 bit-errors in a 2K codeword. Not only can the number of correctable errors in a codeword vary, but this number cannot be determined analytically. The implications of that are significant. It translates directly into the fact that the correction performance cannot be guaranteed through mathematical calculation.

Instead of analytically determining the numbers of correctable errors for every codeword, the performance is measured by simulation. Every performance indication of an LDPC code in any product today is based on simulation. As has been pointed out earlier, the varying number of correctable errors in codewords leads to the fact that only a simulation of every single codeword will make it possible to determine an exact amount of correctable errors. This simulation is highly time-consuming. In chapter 5.2 the time needed for a full simulation of all possible codewords for a standard LDPC implementation is calculated. As it is clearly obvious, this exceeds every development-time of any known system.

In order to simulate the performance nevertheless, several assumptions are taken. One of them is the assumption that the structure of the code and especially of the decoder allows the prediction of certain weaknesses in the decoder. An important step to modeling iterative decoding is to represent codes in a graphical way. Therefore, decoding in most LDPC implementations is done iteratively on a bipartite graph by a so-called message-passing algorithm. In the graph, information is exchanged between neighboring nodes by passing messages on the edges [5]. The performance of LDPC codes is simulated by measuring the performance on a set of so-called "error-prone" patterns that are applied to the decoder or its graph respectively.

The reason for this is that the performance of LDPC codes under iterative decoding algorithms in the error floor region is closely related to the structure of the code's Tanner graph. These "error-prone" patterns are different for the type of channel: They are called "stopping sets" for the binary erasure channel (BSE) [7], while they are called "trapping sets" [8], "near codewords" [9] or "pseudo codewords" [10] in case of the Binary Symmetric Channel (BSC) and the Additive White Gaussian Noise (AWGN) channel. The effect of trapping sets can be seen for all curves in Fig. 2 to the right of the value 5.5 on the SNR, the x-axis. After the LDPC code is simulated with these patterns, it is assumed that the worst correction capability, i.e. the codewords where the LDPC code can correct the fewest errors, is known and that the correction capability in all other codewords is better. There is a certain likelihood that this is true, maybe even for the majority of codewords, but it is definitely not a certainty.
In scientific terms, in the field of statistics, this method of estimating is called importance sampling:

It is a general technique for estimating properties of a particular distribution, while only having samples generated from a different distribution than the distribution of interest [11]. The result of this can be seen in Fig. 3. Towards the right side of the figure, the solid line represents the codewords that have actually been simulated. As the slope of the curve is steep, this region is called the waterfall region – the region where LDPC codes outperform any other codes. Yet underneath the solid line, starting from a UBLER of 10^{-10} and less, it is not certain which performance the code will deliver. As mentioned, the JEDEC defines a frame error rate less than 10^{-16} for enterprise-class flash storage. From Fig. 3, it is easily understandable that it is not possible to guarantee less than 10^{-16} when it´s only possible to simulate down to a frame error rate of 10^{-10} and the rest is based on estimation.

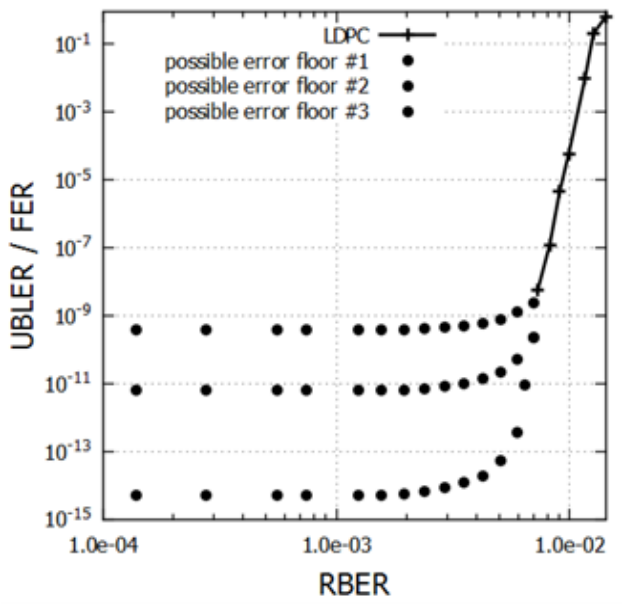

Figure 3: LDPC performance

In summary, the performance of an LDPC code is determined by simulating the codewords to a certain limit and then applying a very limited set of test-patterns that are suspected to be the most difficult to decode. After these have been decoded, it is assumed that this is the minimum error correction performance that can be expected in all of the other untested codewords as well. As this is only an estimate and there is no proof that this is correct, the complete performance cannot be guaranteed.

Calculation of LDPC Simulation Time

This chapter will go through a rough calculation of the time needed to verify an LDPC implementation and its error correction capability in order to guarantee that a certain amount of errors can be corrected in every single codeword.

The 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 [12]. Therefore, we require a frame error rate of 10^{-16}.

$$FER = 10^{-16}$$

To verify this frame error rate, it is necessary to simulate at least 40 x 10^{16} codewords. Assuming a code rate of

$$R = 0.9$$

For each 4K codeword results in a data size of 3.6 x 10^4bits. The total data volume to be simulated is then:

$$40 * 10^{16} * 3.6 * 10^4 \text{bit} = 1.44 * 10^{22} \text{bit}$$

If we then assume a simulation throughput of ~4$\frac{Gbits}{s}$ the time needed for the simulation can be obtained by dividing the data by the data-throughput. The total time to simulate all codewords is:

$$t = \frac{(1.44 * 10^{22} bit)}{(4 * 10^9 \frac{bit}{s})} = 3.6 * 10^{12} s = 114{,}079\ years$$

Advantages of LDPC

- LDPC codes can be very performant in terms of error correction

Disadvantages of LDPC

- The location of the error floor can only be estimated, therefore the performance of the code can only be estimated
- The class of code, LDPC, is often the only reference given. As a result, it is hard to distinguish which LDPC code with what level of performance is actually implemented.

5. Generalized Concatenated Code

It was mentioned earlier that Digital Video Broadcast standard uses a concatenation of an LDPC code and a BCH code. This is one of the two main methods of code concatenation, where the system of the inner encoder, transmission channel, and the inner decoder are seen as a super channel for an outer code. The outer code is designed to correct the residual errors of the inner code [13].

The other method is the so-called generalized code concatenation. It combines several outer and inner codes and optimizes the properties of the overall code. As a result, it is possible to combine the advantages of the inner and outer codes in order to reduce decoding complexity for example [13].

The GC code uses a combination of an inner BCH code and an outer Reed-Solomon (RS) code. The immense advantage that this code-construction offers is in one particular aspect very similar to a BCH-code: the number of correctable errors in each codeword can be analytically determined. This implies that for every codeword, the error correction can guarantee a level of correction performance. As for all of the available flash memories a guaranteed bit error rate is specified, it is possible to guarantee a reliable operation within specified parameters.

Technical Background

The GC codewords are arranged in a matrix. The rows of the matrix are protected by nested BCH codes, whereas the columns are protected by a RS code [13]. The first one is the inner code whereas the latter one is the outer code. A similar construction was presented in [14] for magnetic storage systems. Generalized concatenated codes are suited for high code rate applications with low-complexity decoding algorithms [15].

In contrast to LDPC codes, the residual error rates for GC codes can be determined analytically [16], which is important for applications where a low probability of failure has to be guaranteed.

This means that it is possible to mathematically guarantee a correction performance for every single codeword which implies in return that it is possible to guarantee a defined performance regarding uncorrectable block error rates on system level.

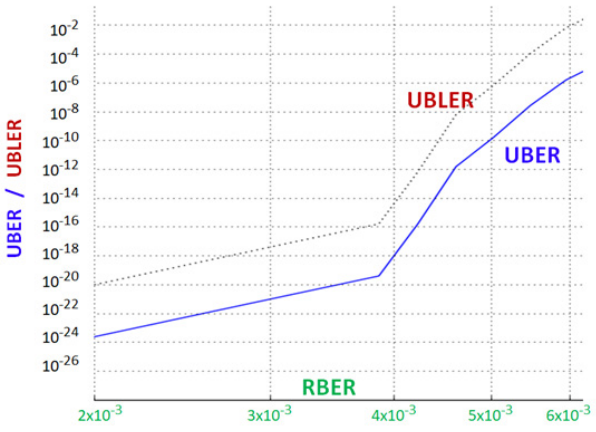

Figure 4: GCC performance simulation (4K codeword size, 0.9 code rate)

Soft Decoding

The soft-decoding of the GCC utilizes so-called logarithmic likelihood ratio (LLR) tables that contain probabilistic reliability information for the bits of a specific NAND flash device at a specific lifecycle condition. LLR tables are used by the soft decoder of the error correction engine to increase the probability to successfully correct false bits.

To obtain LLR-values of a specific type of flash memory, a sample is first brought to the desired lifecycle (for instance the maximum program- and erase-cycle number and the maximum data retention time). Then by applying three different readings starting from a calibrated reference voltage the population of bits is divided into 5 zones, as can be seen in Fig. 5. Readings at R+/- are normally called soft readings.

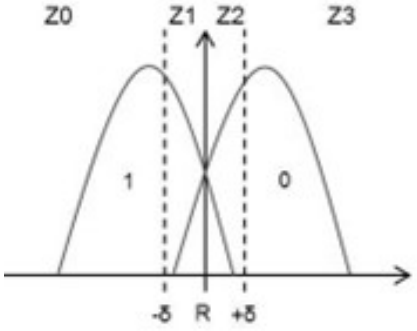

Figure 5: Cell threshold voltages for single-level cell (SLC) flash memory

Z0 and Z3 are the areas with higher reliability and Z1 and Z2 those with lower reliability (i.e. where more likely error bits are located). For each zone the ratio between the logarithmic of the probability to have 0s or 1s is calculated

$$LLR = \frac{\log(p(0))}{\log(p(1))}$$

So that a table like the following can be filled

Z0	Z1	Z2	Z3
-15	-4	4	15

In MLC, TLC or QLC based devices where multiple references are used the same process is applied to each reference.

Depending on the architecture used by the error correction, additional readings can be performed to gain even more information about the area with lower reliability.

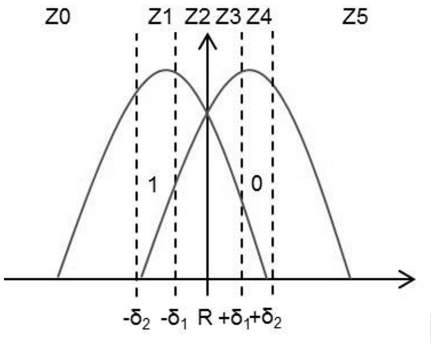

Figure 6: Cell threshold voltages

When the soft decoder tries to recover a failing page, the flash controller first performs several soft readings for that page. Using the additional information, it is now possible to identify which zone each bit belongs to and to assign it to its specific LLR value. With this additional reliability information, the soft decoder can significantly enhance the correction power of an ECC.

Different strategies can be implemented for the generation and usage of LLR tables.

LLR tables can be generated during a characterization phase in which flashes of the same technology are tested. Another approach is to generate and update LLR tables during operation within the storage device by using dedicated sample data. That approach should track accurately the lifecycle of the device. Sample data is a dedicated portion of flash data in which a known pattern is written. By definition, to build LLR tables, the information about which are the failing bits should be available thus they cannot be extracted by reading the data stored in the application.

Both approaches have pros and cons and depending on the application one or the other or even both can be used.

Since LLR tables provide reliability information for the flash device used in the application many different tables should be generated to cover the entire lifecycle of the device. For this reason tables at different cycling conditions, data retention, read disturb and temperature are collected and stored.

Calibration

During the operating lifetime and in case of uncorrectable errors read from the flash memory, the controller executes a calibration which shifts read-threshold levels of the memory.

On the transistor level, reading data from a NAND flash memory cell means to compare the threshold voltage of the cell with a reference voltage. The result of this operation is "0" if the threshold is higher than the reference voltage or "1" otherwise. There are many disturbs that impact the threshold voltage of a cell: program- and erase-cycling, read disturb, data retention, temperature variation, etc. The threshold shift due to above disturbs can cause the cell threshold to cross the reference voltage thus resulting in an error during a read operation. The operation that tries to restore the best reading conditions by changing the reference voltage to reduce error bits is called calibration.

As depicted in Fig. 7, calibration changes the reference voltage to eliminate, or reduce as much as possible, the number of error bits. Since different blocks or pages can experience different disturbs, optimal calibration result found for one page can not necessarily be applied to another page.

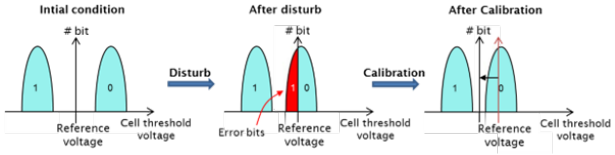

Figure 7: Calibration process

If that is the case, the calibration process should be repeated. NAND flash devices do not automatically track threshold variations. Instead, the flash controller is expected to decide when calibration is needed and to execute the suitable sequence of operations. The effects can be seen in Fig. 8.

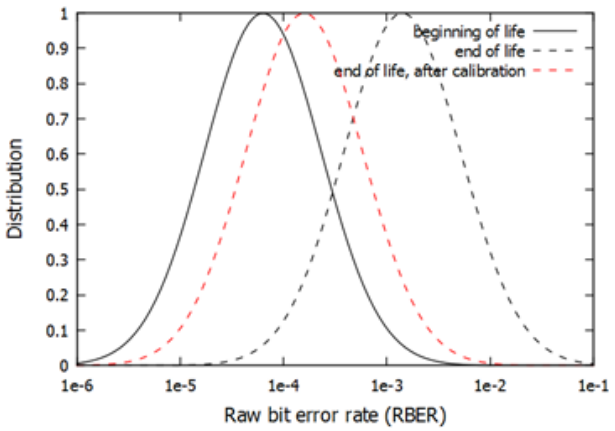

Figure 8: Effects of Calibration

While at the beginning of the lifetime of a flash memory the expectancy value of the raw bit error rate lies below $1e^{-04}$, it moves over the lifetime of the memory towards $1e^{-03}$ and more. The calibration process counters this and moves the expectancy value of the raw bit error rate back to a lower value.

Calibration is a time-consuming task and could limit read performance if used too frequently. Hence, it is normally done when the number of error bits is close or above the correction capability of the error correction.

The frequency of this situation is increasing towards the NAND's end of life.

Advantages of the GCC

- Hard- and soft-decision error correction
- Analytical determination of correction performance
- LLR based soft-decoding (LLR tables obtained from flash characterization or from in-system data based on pilot data)
- Correction strength is a residual from (calibration, GCC hard/soft decoding, tuned read parameters, flash channel model)

Disadvantages of the GCC

- Some LDPC codes show better results in for higher bit error rates while showing worse results for lower bit error rates

Conclusion

This paper led through the history of error correction to the codes that are relevant in the field of NAND flash memory today.

For a long time, the BCH codes were predominantly found in the error correction modules. In recent years, the market has mainly moved towards LDPC based error correction. Although these codes can be powerful, they can have drawbacks for industrial applications or applications with higher demands for reliability. To overcome these drawbacks, GC codes found in certain Swissbit modules boast guaranteed correctional performance.

List of Sources

[1] M. Golay, "Notes on Digital Coding," Proceedings of the IEEE, p. 657, 1949.
[2] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," Transactions of the IRE Professional Group on Information Theory, pp. 38-49, September 1954.
[3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields. In: Journal of the Society for Industrial and Applied Mathematics," SIAM J., pp. 300-304, 1960.
[4] R. G. Gallager, "Low-Density Parity-Check Codes," IRE Transactions on Information Theory, pp. 21-28, 1962.
[5] R. L. Urbanke and T. J. Richardson, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," IEEE Transactions on Information Theory, pp. 599-617, February 2001.
[6] T. Richardson, "Error Floors of LDPC Codes," [Online]. Available: https://web.stanford.edu/class/ee388/papers/ErrorFloors.pdf. [Accessed 02 11 2023].
[7] C. Di, D. Proietti, E. Telatar, T. Richardson and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," IEEE Transactions on Information Theory, pp. 908-917, June 2002.
[8] T. Richardson, "Error floors of LDPC codes," in Proc. 41th Annual Allerton Conference on Communication, Control and Computing, Monticello, 2003.
[9] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanunjan-Margulis low-density parity-check codes," Electronic Notes in Theoretical Computer Science, no. vol. 74, 2003.
[10] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," IEEE Transactions of Information Theory, 2005.
[11] Wikipedia, "Importance sampling," [Online]. Available: https://en.wikipedia.org/wiki/Importance_sampling. [Accessed 22 Decemeber 2023].
[12] d. JESD218, "Solid-State Drive (SSD) Requirements Endurance Test Method,," JEDEC Solid State Technology Association, 2010.
[13] J. Freudenberger, U. Kaiser and J. Spinner, "Concatenated code constructions for error correction in non-volatile memories," International Symposium on Signals, Systems and Electronics (ISSSE), pp. 1-6, Oct 2012.
[14] J. Spinner and J. Freudenberger, "Design and Implementation of a Pipelined Decoder for Generalized Concatenated Codes Format," in SBCCI '14 Proceedings of the 27th Symposium on Integrated Circuits and Systems Design Article No. 35, Aracaju, Brazil, 2014.
[15] A. Fahrner, H. Griesser, R. Klarer and V. Zyablov, "Low-complexity GEL codes for digital magnetic storage systems," IEEE Transactions on Magnetics, vol. 40, no. 4, pp. 3093-3095, July 2004.
[16] I. Dumer, "Concatenated codes and their multilevel generalizations," in Handbook of Coding Theory vol. 2, Amsterdam, The Netherlands, Elsevier, 1998.

swissbit®

Do you have any questions? Get in touch!

Europe
+49 (30) 936 954 400
sales@swissbit.com

USA
+1 (978) 490 3252
salesna@swissbit.com

About Swissbit

Swissbit AG is the leading European manufacturer of storage, security and embedded IoT solutions for demanding applications. As trusted partner, Swissbit empowers the digital and connected world by reliably storing and protecting data in industrial, security and IoT applications.

www.swissbit.com