Error Control Schemes for Modern Flash Memories

Solutions for Flash deficiencies.

By Frederic Sala, Kees A. Schouhamer Immink, and Lara Dolecek

Flash, already one of the dominant forms of data storage for mobile consumer devices, such as smartphones and media players, is experiencing explosive growth in cloud and enterprise applications. Flash devices offer very high access speeds, low power consumption, and physical resiliency.

Flash technology has improved at a dramatic rate over the course of the last ten years. Flash prices have fallen from US$3/GB in 2006 to less than US$1/GB for consumer drives. Costs have decreased to US$2/GB for enterprise-class Flash-based drives. The data storage density has increased to the point where 2-TB Flash-based solid-state drives (SSDs) are commonly available. Furthermore, the advantages of Flash have been recognized by the data center and cloud storage communities. For example, Facebook, Amazon, and Dropbox have begun to replace their traditional magnetic-storage drives with Flash-based SSDs.

Increased storage density and lower prices, however, have come at the cost of decreased Flash reliability and lifetime. Device lifetimes are rated in program-erase (P/E) cycles. Traditional Flash devices, which store a single bit per Flash cell (SLC), are rated at approximately 10^5 P/E cycles. Storing 2 bits per cell lowers the rating to roughly 10^4 P/E cycles. One further bit, yielding triple-level cells in triple-level cell (TLC) devices, further lowers the rating to 10^3 P/E cycles. This is a disastrous trend, especially in applications that require frequent writing. The device lifetime is also negatively affected by decreasing feature size. Similar trends hold for device reliability, which is measured by the average bit-error rate (BER) at the rated lifetimes [1].

The solution to this discouraging trend is to apply error-correcting codes (ECCs). ECCs are frequently used in many types of computer memories. However, because of the particularly significant deficiencies present in Flash, error correction plays an especially important role. Traditional ECCs, such as Bose–Chaudhuri–Hocquenghem (BCH) codes and Reed–Solomon (RS) codes, have often been used by Flash designers. The very popular low-density parity-check (LDPC) codes are currently widely studied for Flash applications. Furthermore, an entire series of advanced coding techniques has been developed to take advantage of the unique properties of Flash memories.

Our goal in this article is to provide a high-level overview of error correction for Flash. We will begin by discussing Flash functionality and design. We will introduce the nature of Flash deficiencies. Afterwards, we describe the basics of ECCs. We discuss BCH and LDPC codes in particular and

Digital Object Identifier 10.1109/MCE.2014.2360965
Date of publication: 17 December 2014

wrap up the article with more advanced methods and future directions for Flash coding.

FUNCTIONALITY

Flash memories are built from floating-gate transistors. Electrons can be trapped in the floating gate of the device. Information is represented by modulating the amount of charge present on this gate [2]. Each floating-gate transistor is referred to as a Flashcell. A Flash memory is a large array of cells; the cells are organized into pages, which are further grouped into blocks. A typical Flash block contains approximately one million cells. Figure 1(b) shows a Flash cell, while Figure 1(a) shows the array of cells composing a block.

Different types of Flash memories are categorized according to the amount of information that can be stored in a cell. SLC memories store 1 bit per cell. The term "single level" refers to the fact that one level beyond the zero voltage level is needed to distinguish among two states and, thus, store a single bit of information. MLC memories store ≥ 2 bits of information per cell. The name MLC is sometimes used to refer specifically to cells that store 2 bits of information; TLC refers to memories storing 3 bits per cell. Flash chips with 4 bits per cell are the densest production devices available today.

Reading and writing Flash cells works in the following way. A cell can be written to by inserting charge into the floating gate. The way this is done depends on the specific type of Flash device. In general, writing specifically refers to adding charge to a cell. Removing charge from a cell requires a different operation: Fowler–Nordheim tunneling. Note, however, that while writing to a cell can be done individually, the erasure operation must be done over an entire block, which contains numerous cells. As a result, erasing is a much more expensive operation than writing. We shall return to this issue when discussing Flash problems.

Reading a Flash cell is simpler. First, the correct cell is selected according to its location in a block. A threshold voltage is applied onto the appropriate page and bit lines (the vertical and horizontal black lines, respectively, shown in Figure 1) to determine whether the corresponding cell contains charge. This is the case with an SLC cell. In the MLC case, several thresholds must be applied to determine the level of the device.

MEMORY IMAGE COURTESY OF FREEIMAGES.COM/LEONLAI.
FLASH IMAGE LICENSED BY GRAPHIC STOCK.

FLASH ISSUES

Despite the many advantages previously discussed, Flash memories also experience a diverse set of issues. We will briefly discuss the key problems. These problems are exacerbated by the increase in the per-cell storage density. Recall that information is represented in Flash cells according to the amount of charge present on a gate; each possible value that cells represent is assigned a range of voltage levels, and each cell's voltage is checked to determine in which value's range it falls. As devices move from SLC to MLC, and from less- to more-dense MLCs, the ranges of voltage that each value may occupy decrease. These smaller intervals increase the potential for error when cells deviate from the ideal voltage levels, regardless of the source of these deviations. This notion is depicted in Figure 2. Here, we see the distributions for an MLC and TLC cell. The TLC distributions must occupy a smaller interval in the voltage range and have a higher overlap, resulting in a larger potential for error.

One of the major issues that Flash cells experience is charge leakage. Charge leakage occurs when electrons trapped in the cell slowly leak out over time. As a result, the voltage levels of the transistors in the cells decrease slowly. Once this decrease is sufficiently large, the cell's voltage will fall into an incorrect interval and the cell will be read incorrectly. Note that such errors are asymmetric: charge leakage causes cell voltages to decrease, but not to increase. Such asymmetric errors are unusual: the typical communications channel operates on the assumption that the value to be transmitted is equally likely to increase or decrease during transmission. For this reason, error correction for Flash is a unique problem. A different example of the asymmetry of errors in Flash can be seen in Figure 2. The distribution for the zero value is much larger than that of other values, and, as a result, errors involving a stored value of zero are much more probable.

Another concern, briefly mentioned earlier, is also asymmetric: the difference between reading and writing. Recall that a Flash cell can be written to individually, while a cell cannot be erased without erasing the entire block of cells. Such blocks contain many cells, and, therefore, an erasure is a major event. Moreover, the erasure process has a long duration and permanently wears out the device. For this reason, Flash manufacturers

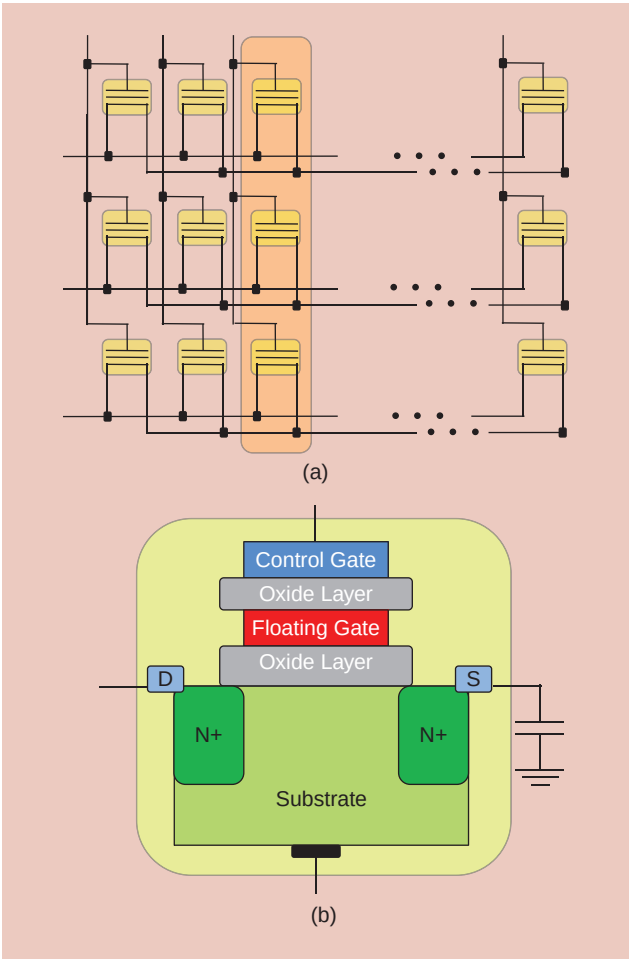

FIGURE 1. Flash organization for an SLC device. (a) A Flash block in which each yellow unit represents one cell. Each vertical collection of cells, one of which is highlighted in orange, makes up a page. (b) A single Flash cell made up of a floating gate transistor. The floating gate, in red, is isolated between two oxide layers.

rate their products by P/E cycles. Flash devices are only capable of undergoing a finite amount of P/E cycles before the devices have been damaged to such a degree that they are no longer reliable. For example, an SLC device is capable of withstanding roughly 10^5 P/E cycles. However, a modern TLC device can only undergo about 10^3 P/E cycles. It is clear that erasures must be avoided at all costs. As we will see, rewriting schemes have been developed that minimize the required number of erasures.

Although a Flash cell may be written to individually, the process is not entirely independent. That is, when inserting charge into a particular cell, due to parasitic capacitances between physically adjacent cells, the neighboring cells' floating gates may also gain electrons, raising their voltage levels. This behavior is known as intercell coupling, or cell-to-cell interference. It is particularly prominent when a cell is stuck between two cells whose levels are significantly increased during writing. This problem is depicted in Figure 3.

Another type of error related to inserting charge into cells is overwriting. Overwriting occurs when too much charge is inserted into a cell. In this case, the cell's voltage level has

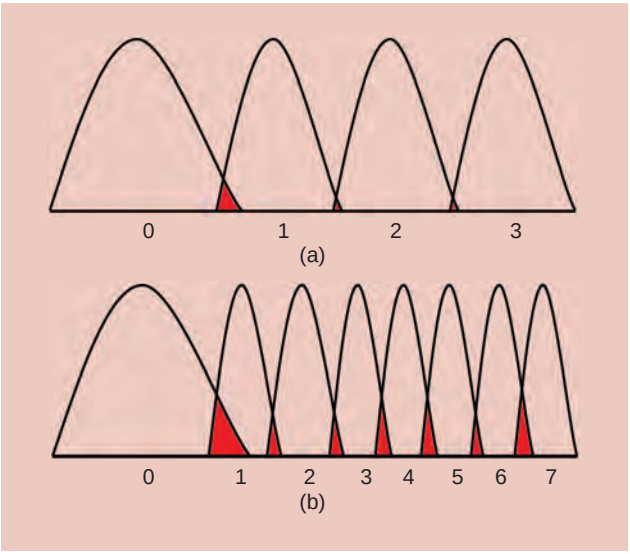

FIGURE 2. Flash sources of error. The distribution of the voltage in a cell for various values is shown (a) for the MLC (2 bits per cell) case and (b) for the TLC (3 bits per cell) case. The areas highlighted in red are potential errors. Note that the distributions in the TLC case must be much thinner to fit the same voltage interval and, even so, have a much larger potential for error.

been pushed up beyond the desired target. When overwriting has occurred, a costly erasure is needed. Therefore, it is crucial to avoid overwriting. For this reason, cell programming is a gradual process: a small amount of charge is inserted, the cell is read back, another small amount of charge is inserted, and so on. This gradual charge-injection process allows for the avoidance of overwriting.

ECCs AND CODING SCHEMES

All of the problems detailed in the previous section have significant impacts on the reliability and lifetime of Flash memories. As a result, many schemes based on coding theory have been introduced to ameliorate these effects. In this section, we broadly classify these schemes into two categories: ECCs, which correct cells that are read incorrectly, and rewriting codes, whose goal is to maximize information storage while avoiding expensive block erasures. We note that there are schemes capable of both correcting errors and minimizing

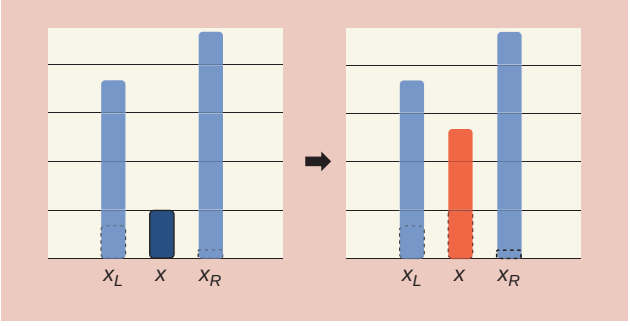

FIGURE 3. Intercell coupling in Flash. The levels of the cells x_L and x_R are increased in programming. As a result, the neighboring cell x's level is inadvertently increased.

erasures, thus falling into both categories. In the "Advanced Coding Techniques" section, we will discuss such schemes.

ECCs

We will briefly introduce some basic topics from coding theory. ECCs are a way of protecting transmitted information against corruption due to noise. The idea is to insert redundancy into the transmitted message in such a way that the receiver is capable of retrieving the original, uncorrupted intended message from the noisy version it receives.

Let us say we wish to transmit a single bit, for example, to a receiver, knowing that the bit may be flipped to the incorrect value ($0 \rightarrow 1$ or $1 \rightarrow 0$) with some probability p. One way to add redundancy to protect our bit is to transmit 000 instead of 0 and 111 instead of 1. We call this procedure coding, and the transmitted sequences are called codewords. The original values, in this case 0 and 1, are called messages. The set of codewords forms the code. In our toy example, which is a type of repetition coding, the code is $C = \{000, 111\}$. The process is then to encode our information by mapping each message to a codeword, to communicate our selected codeword, then to decode the received sequence to a message at the other end. The medium over which the codeword is transmitted is called the channel, as shown in Figure 4.

In our example, we know that some bits may be flipped during transmission, so we may receive a sequence not made up entirely of 0s and 1s. A simple decoding rule is to decode a received sequence with more 1s than 0s as a 1, and one with more 0s than 1s as a 0. This rule is not guaranteed to catch all errors: if two bits are flipped in a codeword, such as $111 \rightarrow 100$, the decoding rule will produce an incorrect output. However, the probability of error is reduced. In fact, if we wish to reduce the probability of error even further, we may map each bit to an even longer sequence: k 0s or k 1s. However, there is a major disadvantage: we require k bits to transmit a single bit of information, so our rate of communication is just $1/k$. For our code C, we have $k = 3$, so the rate is 1/3. If we wish to reduce our error probability to zero, the value of k must go to infinity, and the rate likewise drops to zero. For this reason, repetition codes are not efficient.

However, simple examples, such as repetition codes, serve to demonstrate the key concepts in coding theory. We wish to encode information efficiently by maintaining a high rate. We wish to correct as many errors as possible. Finally, we also seek efficient decoding rules. The last aspect is particularly tricky. There is a natural decoding technique [known as maximum-likelihood (ML) decoding] where the decoder computes the probability that each possible codeword was transmitted conditioned on the received sequence. However, doing so requires performing a number of computations for each codeword. In a large code, with millions of codewords, this is impossible. Therefore, for our codes to be practical, we require low-complexity decoders.

The earliest and most famous example of an ECC is the Hamming (7, 4) single-bit ECC. In this code, the transmitted sequences (codewords) are 7 bits long and contain 4 bits

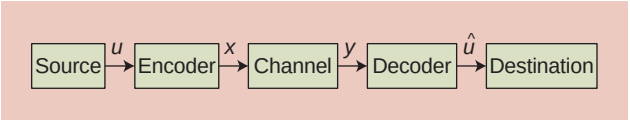

FIGURE 4. A coding block diagram.

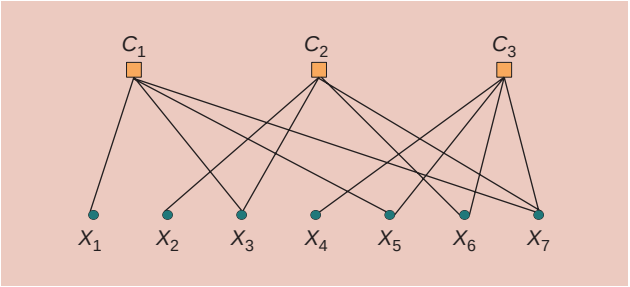

FIGURE 5. A Tanner graph for the (7,4) Hamming code. There are seven variables nodes (circles) and three check nodes (squares).

of information (and, thus, 3 bits of redundancy). If any single bit in the codeword is flipped during transmission, the receiver can detect the position of this flip and reverse it, retrieving the correct intended message.

Hamming codes are a specific family of linear block codes. Linear codes form a linear (vector) space so that the sum of any two codewords in the code is also a codeword. The term "block" simply refers to the fact that all codewords have the same length. Just as in the previous Hamming code example, linear block codes are described in terms of the three parameters $[n, k, d]$, where n is the codeword length, k is the dimension of the space spanned by the codewords (for a binary code, dimension k simply means that each codeword conveys k bits of information), and d is the minimum distance. The minimum distance parameter describes what type of and how many errors the code is capable of dealing with. A code with a minimum distance d can correct $\lfloor (d-1)/2 \rfloor$ errors and can detect $d-1$ errors. The Hamming code corrects a single bit error so that $d = 3$.

Since linear block codes form a linear space, each such code can be described by the null space of a matrix, which is referred to as the parity-check matrix H. A sequence $\mathbf{c}$ is a codeword in the code C if the equation $\mathbf{c}H^T = 0$ is satisfied. For example, the Hamming [7, 4, 3] code can be derived from the parity-check matrix

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}. \quad (1)$$

Notice that the dimensions of H are 3×7. This implies that the codewords have a length of $n = 7$ and that there are three parity-check equations, given by each row. For example, the first row tells us that the first, third, fifth, and seventh bits in each codeword must sum (in this case, XOR) to zero. Such constraints on the sums of parities of the codeword bits are why linear block codes are sometimes referred to as parity-check codes.

To determine how powerful the type of ECC must be, Flash manufacturers examine the underlying error rates of their

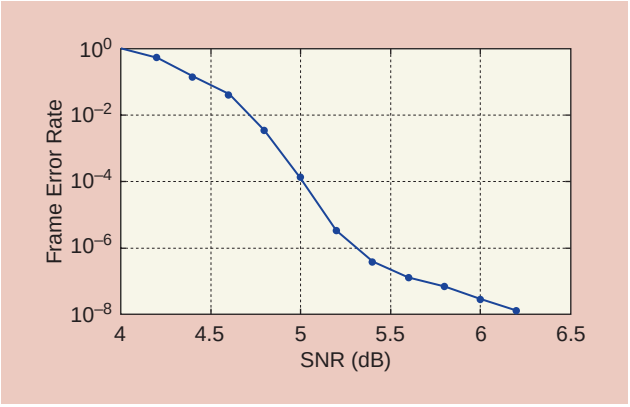

FIGURE 6. A performance curve for an LDPC code. At low values of SNR, the output BER falls dramatically with a slight increase in SNR. This is the waterfall regime. As the SNR increases, the BER curve begins to level off; this effect is known as the error floor.

devices. The average probability of error due to all sources is summarized by the BER of the device. The BER, which is determined by rigorous testing, determines the expected number of errors in a group of Flash cells. In earlier SLC-based Flash devices, the typical number of bits in error among a number of cells was less than one, so, therefore, single-bit error-correcting Hamming codes were sufficient. However, as devices have more density and higher complexity, the BER has increased. For example, some current TLC Flash devices have BERs in excess of 10^{-4}, which means that the number of expected errors is much higher. At the same time, Flash devices must be efficient, so we seek rates of 0.9 and higher. The codeword length n is typically 8,192 bits (1 kB) or longer. There are several classes of ECCs used in Flash that fit these target parameters.

BCH CODES

Flash designers have long relied on a class of codes known as BCH codes. BCH codes are a family of algebraic linear-block codes. In other words, they are based on algebraic spaces with certain properties and have codewords of equal length forming a linear (vector) space [3]. It is possible to view Hamming codes as a specific subclass of BCH codes. BCH codes, however, are capable of correcting more than one codeword error. In fact, it is possible to design BCH codes with any desired target error-correcting capability (that is, a target minimum distance d). Of course, the larger the error-correcting capability, the lower the code rate for a given codeword length and the higher the overhead that is used for redundancy.

Recall that designers must select the code parameters $[n, k, d]$. An example of a BCH code that is useful for Flash is the [8,262, 8,192, 5] code. Here, each codeword is 8,262 bits long, contains 8,192 bits (that is, 1,024 kB) of information, and has a minimum distance of $d = 5$, so it is possible to correct two bit errors among the 8,262 bits of the codeword. The rate, in this case, is $r = k/n = 0.9915$. A BCH code with a higher error-correcting capability (and higher overhead) is the [8,360, 8,192, 12] code, which can correct five errors or detect 11 errors.

BCH codes have a long history, having been initially discovered more than 50 years ago. In addition to being able to correct several errors, BCH codes have several advantages. Specifically, efficient low-complexity encoders and decoders exist for BCH codes. Two famous classes of BCH decoders are based on the Peterson–Gorenstein–Zierler and the Berlekamp–Massey algorithms. A popular subclass of nonbinary BCH codes are RS codes.

LDPC CODES

Despite the simplicity of BCH codes, other more powerful classes of codes exist. In this subsection, we introduce the most popular such class, known as LDPC codes. These codes were originally introduced by Robert Gallager in the 1960s [4] and were subsequently rediscovered in the 1990s. LDPC codes have been the subject of enormous research and application interest in the last two decades.

Like BCH codes, LDPC codes are also linear block codes. The "low-density" terminology refers to the fact that the parity-check matrix of an LDPC code is sparse. That is, each codeword bit is involved in relatively few parity checks. It is possible to visualize the parity-check matrix H by associating it with a graph, known as the Tanner graph. This is a bipartite graph with two types of nodes: variable nodes, corresponding to the bits in each codeword, and check nodes, corresponding to the parity-check equations (the rows in H). There is an edge placed between a variable node and a check node if the corresponding bit in the codeword is included in the appropriate parity-check equation. We illustrate the idea for the Hamming (7, 4) code with parity-check matrix H given by (1) in Figure 5.

This graph-based representation naturally lends itself to a form of decoding performed on the Tanner graph itself. For this reason, LDPC codes are sometimes described as graph-based codes. Decoding is performed by a technique called message passing. The idea is to communicate messages back and forth between the variable and check nodes. The initial messages are given by the values of the received sequence. The messages reflect beliefs (probabilities) regarding the values of the original intended codeword bits. For this reason, LDPC decoders are called belief-propagation decoders. After a number of rounds of back-and-forth communication among the nodes in the Tanner graph, the beliefs converge to produce the output. The process requires several iterations, so LDPC decoders are a class of iterative decoders.

This decoding procedure is known to be efficient in terms of complexity. It is in fact optimal (equivalent to ML decoding) in the case where the Tanner graph does not contain any cycles. This is not the case in practical situations, so iterative decoding is generally suboptimal. Nevertheless, it can be shown that iterative decoding performs very close to the optimal, impractical ML decoder. We note that, although iterative decoding is low complexity in a mathematical sense, the hardware required for LDPC decoders is nevertheless more involved compared to that for BCH decoders.

LDPC codes are also famous for their near-capacity performance at very long block lengths (very large values of n).

The capacity is a quantity that refers to the maximum rate at which information can be reliably transmitted through a particular channel. Shannon's channel coding theorem (1948) showed that there exist codes with a vanishingly small probability of error for any rate below channel capacity, but no such codes for any rate above capacity. The goal for coding theorists is to develop codes that operate at rates near or equal to capacity while still retaining low-complexity encoding and decoding features. Algebraic codes, including BCH codes, however, do not operate near capacity.

LDPC codes also offer the benefit of soft-decision decoding, while BCH codes are limited to hard-decision decoding. "Hard-decision" refers to the fact that the received sequence, which forms the input to the decoder, must be binary (in the case of a binary code). That is, we must feed the decoder 0s or 1s. On the other hand, with soft-decision decoding, the inputs may be a range of values in between. Soft-decision decoding yields better performance.

Previously, when designing BCH codes, we focused on the minimum distance parameter d. This is the parameter of interest for all algebraic codes. However, with LDPC codes, we are less interested in d. LDPC codes ensure a very low overall probability of error. Correcting a particular class of errors (e.g., two bit errors in each codeword) is not as important. LDPC codes can be evaluated either theoretically, for code lengths that are asymptotic (where the code length n goes to infinity), or through simulation, where the error rate of the code is evaluated for various input BERs. What we typically observe when examining the performance of LDPC codes through simulation is that the performance is initially excellent at low signal-to-noise ratios (SNRs), but, as the SNR is increased, the performance curve BER begins to level off. These two regimes are known as the "waterfall" area and the "error floor." The origin of these names can be understood from Figure 6. The error floor is known to be caused by subsets of vertices in the Tanner graph that meet certain combinatorial conditions. These are called absorbing sets. Removing absorbing sets is a critical issue in LDPC design.

In summary, LDPCs offer both low-complexity, efficient decoding and very high rates. For this reason, LDPC codes have become extremely attractive for application to Flash memories. The main challenge is designing good LDPC codes at high rates ($r > 0.9$) and short to moderate block lengths. This is a more difficult task compared to BCH code design. A popular and efficient class of LDPC codes are so-called quasicyclic codes, which can be constructed based on techniques from algebraic geometry.

REWRITING SCHEMES

As we described earlier, a major issue in Flash is avoiding erasures. Therefore, we are interested in representing information in Flash in such a way that it can be rewritten several times without the need for block erasures. Coding schemes (with or without error correction) that have this capability are known as rewriting schemes.

A particularly famous type of rewriting scheme for Flash is based on write-once memory (WOM) codes. These codes

Tensor-product codes are a scheme that uses linear-algebraic operations to combine multiple algebraic codes.

were introduced by Rivest and Shamir in the 1980s in the context of optical media, which can only be programmed once [5]. The idea in WOM codes is to allow information to be written in a block of memory multiple times in such a way that the stored values can only be increased, never decreased. This scenario is perfectly suited to erasure avoidance in Flash.

WOM codes are characterized by how many times data can be rewritten to a block and how much total data may be stored at the stages following these writes. A code allowing for t writes is a called a t-write WOM code. The sum of the individual rates on each of the t writes is known as the sum rate.

The most basic WOM code, introduced by Rivest and Shamir in [5], allows for two writes, storing 2 bits each in a memory consisting of 3 bits of storage. This gives a sum rate of 4 bits. The table in Figure 7 describes the operation of the code. The left column shows the 2-bit message x we seek to store. The middle column shows the encoding of this message (using 3 bits) for the first write. The rightmost column depicts the codewords used for the second write. Note that changing the message stored from the first to the second write does not require changing any 1 to a 0, as desired.

Floating codes are a generalization of WOM codes [6]. In a t-write WOM, we store one codeword (that is, a single sequence or variable). We may rewrite this variable up to t times, but after the tth write, the memory block must be erased. Floating codes allow for the storage of k variables with $w = kt$ writes in such a way that each of the variables may be rewritten any number of times (possibly more than t), so long as there have been no more than kt total writes.

The idea underlying floating codes is to increase flexibility: rather than requiring each of the k variables to be written at most t times, any number of writes are allowed as long as the total number of writes does not exceed the parameter $w = kt$. Like WOM codes, floating codes are characterized by their total information storage and the write parameter w. We also note that recent research in the area of rewriting

x	$c(x)$	$c'(x)$
00	000	111
01	100	011
10	010	101
11	001	110

FIGURE 7. An encoding table for two-write WOM code storing 2 bits of information in 3-bit codewords.

> Flash devices are only capable of undergoing a finite amount of P/E cycles before the devices have been damaged to such a degree that they are no longer reliable.

codes has led to the development of WOM and floating codes that integrate error-correction capability.

CONSTRAINED CODES

A third category of codes for Flash memories are constrained codes. Specifically, balanced codes, where codewords have an equal number (or a fixed ratio) of zeros and ones, have been proposed and used to counter the detrimental effects of charge offset uncertainty [7]. Zhou et al. [8] and Sala et al. [9] investigated constrained codes that enable dynamic reading thresholds in nonvolatile memories such as Flash. Immink and Weber [10] advocated a new detection method based on the Pearson distance, which is intrinsically resistant to offset uncertainty.

Constrained codes are also useful when dealing with intercell coupling. Recall that this effect takes place when a cell with a small level is sandwiched between two cells whose levels are dramatically increased. The result is that the middle cell level is inadvertently increased. Constraints codes have been introduced that exclude programming adjacent cells as high–low–high [11]. Various two-dimensional codes have also been proposed to counter the effects of intercell coupling [7].

ADVANCED CODING TECHNIQUES

The two classes of ECCs described in the "ECCs" section, BCH codes and LDPC codes, are general-purpose codes. Although they are quite useful for Flash applications, they cannot fully take advantage of the idiosyncrasies of these memories. More advanced coding techniques exist that attempt to fully exploit Flash characteristics.

First, the LDPC codes that we described are binary codes. However, it is not necessary to limit LDPC codes to binary alphabets. Nonbinary LDPC codes are known to have dramatically improved performance at the cost of additional decoding complexity [12]. Designing powerful nonbinary LDPC codes for application to Flash is a particularly challenging problem. As in the binary case, absorbing sets create the error-floor phenomenon at high SNRs. In the nonbinary case, advanced algorithms are needed to design codes with a limited number of absorbing sets. A powerful class of nonbinary LDPC codes are based on objects known as protographs [13].

Tensor-product codes are a scheme that uses linear-algebraic operations to combine multiple algebraic codes [14]. The idea is to create a code that is capable of correcting very specific types of errors. BCH codes correct any type of symbol errors. Consider a nonbinary BCH code that operates on the alphabet $\{0, 1, \ldots, 7\}$. Here, the codeword is made up of symbols from this 8-ary alphabet. Each symbol contains 3 bits of information. This code is useful for TLC Flash because each codeword symbol represents the value of a single cell (which contains 3 bits in TLC). A BCH code can correct errors from any value to any other (e.g., $2 \rightarrow 6$, $1 \rightarrow 4$, etc.). In other words, it corrects errors symmetrically. To maintain the flexibility to correct so many different types of errors, a larger amount of overhead is required. In practice, however, only a small subset of these errors is known to occur in Flash. Tensor product codes allow for the creation of asymmetric codes, which correct only these particular errors, allowing for a smaller overhead and a higher rate. Tensor-product codes designed specifically for TLC Flash were introduced in [15] and extended in [16]. We also note that other asymmetric error-correcting schemes exist, such as, in the context of Flash, those given in [17].

Another class of codes is based on the concept of rank modulation [18]. The idea in rank modulation is to represent information in the rankings of cells in an entire block rather than on the absolute amount of charge in a particular cell. Therefore, information is stored in permutations and is read by comparing the values of different cells in blocks. This is advantageous for several reasons, including the fact that charge leakage, which affects all cells at roughly equal rates, will not change the relative ranks of cells, only their absolute values. Therefore, rank modulation offers lower error rates.

On the other hand, we still wish to further protect Flash devices from error by using ECCs. Since in rank modulation information is stored in permutations, these codes no longer use binary or nonbinary alphabets, but rather permutations. Such codes are quite challenging to design. Recent advances have shown how to take an algebraic code with a certain minimum distance d and transform it into a permutation code with corresponding properties [19].

CONCLUSION

As we have seen, the explosive growth of Flash as a storage medium along with the ever-growing demand for more fast data storage has led the Flash industry in two directions: devices have become, on the one hand, faster, denser, and higher capacity and, on the other hand, less reliable, more error-prone, and suffering from vastly reduced lifetimes. This trend is likely to continue in the foreseeable future.

To deal with the lack of reliability in Flash devices, designers and manufacturers have turned to tools from coding theory. These powerful tools have traditionally been applied to communications technology, but they are quite useful when applied to memories. Thus far, Flash designers predominantly use algebraic ECCs, such as simple single-error-correcting Hamming codes, or the more general BCH codes. These codes have a structure that is easy to understand and analyze, along with easily implemented encoders and decoders.

Coding theory was revolutionized by the (re-) discovery of LDPC codes. These codes, capable of reaching capacity, offer performance superior to any algebraic code. Flash designers are currently seeking to apply LPDC codes to Flash devices; the challenges here include finding codes of

appropriate length and structure and implementing LDPC decoders in hardware.

Coding also presents solutions to unique Flash problems, such as the disastrous effects of needing to erase cells, which can be mitigated through rewriting codes. An entire series of codes has been developed to deal with specific Flash deficiencies. Inevitably, coding will form a critical component of all future Flash devices. Future Flash codes will combine error correction with rewriting abilities and will be tailored to deal with device-specific issues. Developing such codes will remain a central challenge and a unique opportunity for both the coding theory community and the Flash industry.

ABOUT THE AUTHORS

Frederic Sala (fredsala@ucla.edu) earned the B.S.E. degree in electrical engineering from the University of Michigan, Ann Arbor, in 2010. He is currently pursuing the Ph.D. degree in electrical engineering at the University of California, Los Angeles (UCLA), where he is associated with the LORIS lab. He is a recipient of the NSF Graduate Research Fellowship, the 2013 Distinguished Masters Thesis Award—Signals and Systems Track Electrical Engineering Department, UCLA, and the 2013 Edward K. Rice Outstanding Masters Student Award from the Henry Samueli School of Engineering, UCLA.

Kees A. Schouhamer Immink (schouhamerimmink@gmail.com) earned his Ph.D. degree from the Eindhoven University of Technology in 1985. He founded and became president of Turing Machines, Inc. in 1998. Since 1994, he has been an adjunct professor at the Institute for Experimental Mathematics, Essen University, Germany. He contributed to digital audio and video recording products, such as the CD, CD-ROM, CD-video, digital audio tape recorder, digital compact cassette system, digital versatile disk, and Blu-ray disc. He received widespread recognition for his pioneering contributions to the technologies of video, audio, and data recording. He received a Knighthood in 2000, a personal Emmy Award in 2004, the 1996 IEEE Masaru Ibuka Consumer Electronics Award, the 1998 IEEE Edison Medal, the 1999 AES Gold Medal, and the 2004 SMPTE Progress Medal. He received an honorary doctorate from the University of Johannesburg and was named a fellow of the IEEE, AES, and SMPTE. He was inducted into the Consumer Electronics Hall of Fame and elected into the Royal Netherlands Academy of Arts and Sciences and the U.S. National Academy of Engineering. He served the profession as president of the Audio Engineering Society Inc., New York, in 2003.

Lara Dolecek (dolecek@ee.ucla.edu) earned her B.S. (with honors), M.S., and Ph.D. degrees in electrical engineering and computer sciences, as well as her M.A. degree in statistics, all from the University of California (UC), Berkeley. She is an assistant professor with the Electrical Engineering Department at the University of California, Los Angeles (UCLA), where she also codirects the UCLA Center on Development of Emerging Storage Systems. She received the 2007 David J. Sakrison Memorial Prize for the most outstanding doctoral research in the Department of Electrical Engineering and Computer Sciences at UC Berkeley. She received the Northrop Grumman Excellence in Teaching Award (2013), the Intel Early Career Faculty Award (2013), the University of California Faculty Development Award (2013), the Okawa Research Grant (2013), the NSF CAREER Award (2012), and the Hellman Fellowship Award (2011). She is an associate editor for coding theory of *IEEE Transactions on Communications* and *IEEE Communication Letters* and is the lead guest editor of the 2014 *IEEE Journal on Selected Areas in Communications* special issue on emerging data storage.

REFERENCES

[1] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND Flash memory," in *Proc. 10th USENIX Conf. File and Storage Technologies,* San Jose, CA, 2012.

[2] R. Bez, E. Camerlanghi, A. Modelli, and A. Visconti, "Introduction to Flash memory," *Proc. IEEE*, vol. 91, no. 4, pp. 489–502, Apr. 2003.

[3] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ: Pearson Prentice Hall, 2004.

[4] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[5] R. L. Rivest and A. Shamir, "How to reuse a write-once memory," *Inform. Contr.*, vol. 55, no. 1–3, pp. 1–19, Dec. 1982.

[6] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Nice, France, June 2007, pp. 1166–1170.

[7] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, "Codes for digital recorders," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[8] H. Zhou, A. Jiang, and J. Bruck, "Error-correcting schemes with dynamic thresholds in non-volatile memories," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, St. Petersburg, Russia, July–Aug. 2011, pp. 2143–2147.

[9] F. Sala, R. Gabrys, and L. Dolecek, "Dynamic threshold schemes for multi-level non-volatile memories," *IEEE Trans. Commun.*, vol. 61, no. 7, pp. 2624–2634, July 2013.

[10] K. A. S. Immink and J. H. Weber, "Minimum Pearson distance detection for multi-level channels with gain and/or offset mismatch," *IEEE Trans. Inform. Theory*, vol. 60, no. 10, pp. 5966–5974, Oct. 2014.

[11] A. Berman and Y. Birk, "Constrained flash memory programming," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, St. Petersburg, Russia, July–Aug. 2011, pp. 2128–2132.

[12] M. C. Davey and D. J. C. MacKay, "Low-density parity check codes over GF (q)," *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, June 1998.

[13] L. Dolecek, D. Divsalar, Y. Sun, and B. Amiri, "Non-binary protograph-based LDPC codes: Enumerators, analysis, and designs," *IEEE Trans. Inform. Theory*, vol. 60, no. 7, pp. 3913–3941, July 2014.

[14] J. K. Wolf, "On codes derivable from the tensor product of check matrices," *IEEE Trans. Inform. Theory*, vol. 11, no. 2, pp. 281–284, Apr. 1965.

[15] R. Gabrys, E. Yaakobi, and L. Dolecek, "Graded bit error correcting codes with applications to flash memory," *IEEE Trans. Inform. Theory*, vol. 59, no. 4, pp. 2315–2327, Apr. 2013.

[16] R. Gabrys, F. Sala, and L. Dolecek, "Coding for unreliable Flash memory cells," *IEEE Commun. Lett.*, vol 18, no. 9, pp. 1491–1494, July 2014.

[17] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multi-level flash memories," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1582–1595, Apr. 2010.

[18] A. Jiang, R. Mateescu, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inform. Theory*, vol. 55, no. 6, pp. 2659–2673, June 2009.

[19] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Trans. Inform. Theory*, vol. 56, no. 7, pp. 3158–3165, July 2010.