Rewriting Codes for Flash Memories Based Upon Lattices, and an Example Using the E8 Lattice

Brian M. Kurkoski
University of Electro-Communications
Tokyo, Japan
kurkoski@ice.uec.ac.jp

***Abstract*—A rewriting code construction for flash memories based upon lattices is described, where the values stored in flash cells correspond to lattice points. This construction encodes information to lattice points in such a way that data can be written to the memory multiple times without decreasing the cell values. The construction partitions the flash memory's cubic signal space into blocks, which aids with encoding. The minimum number of writes is approximately linear in one of the code parameters. Using the E8 lattice as an example, the average number of writes can be increased by introducing randomization in the encoding.**

I. Introduction

Rewriting codes are a coding-theoretic approach to allow rewriting to memories which have some type of write restriction, typically values stored in memory may only be increased. While codes for binary media were proposed in the 1980s [1], [2], within the past few years, a large number of rewriting codes directed at flash memory have been described [3], [4], [5], [6], [7]. Most of these these floating codes or flash codes are designed for flash memory cells that can store one of q discrete levels, where the values can only increase on successive rewrites.

However, in the physical flash cell, charge is stored during write operations. Charge, read as a voltage, is an inherently continuous quantity. Commercial flash memory uses analog-to-digital conversion, and present $\log_2 q$ bits per cell of digital data externally. Currently, any coding, for error-correction and rewriting, must operate on these discrete values. However, one might expect that future coding schemes may have access to the continuous, or analog values stored in the flash memory cells.

This paper describes a rewriting code based upon lattices, and assumes that the analog values are available for coding. The values stored in flash cells correspond to lattice points. From a lattice perspective, conventional rewriting codes store data at the points $\{0, \ldots, q-1\}^n$ in a rectangular lattice. However, rectangular lattices are inefficient, and there exist lattices that have many desirable properties such as better packing efficiency.

Because the flash cell values are continuous quantities, this paper takes the signal-space viewpoint that has long been used for the AWGN channel. Among other results, it is now known that lattices can achieve the capacity of the AWGN channel [8] [9], and lattices appear to be a promising practical approach for bandwidth-constrained channels [10]. In fact, a related technique, trellis-coded modulation, has already been considered for error-correction in flash memories [11].

The power constraint and the associated shaping region are important considerations in designing lattice codes for both flash memories and AWGN channels. Shaping for the AWGN channel is computationally expensive, requiring lattice quantization in order to shape the codebook into a sphere-like set of lattice points [9]. But for flash memories, the power constraint is cubic, that is, all points are within the cube $(0, q-1)^n$, corresponding to the fact that the voltage on each cell has a minimum and maximum possible value. Fortunately, lattice quantization is not required for encoding, because there is an efficient encoding which results in a cubic shaping region, when the lattice generator matrix is triangular [12]. This paper includes a slight generalization of this method.

The flash memory model used in this paper has n cells which have values in the range $(0, q-1)$, and the values are continuous. Successive writes can only increase the values. For fixed constraints, the code construction should maximize the number of times that data can be written. Note that while many existing rewriting code constructions allow modifying a single information bit, the code presented here changes the entire information sequence with each write.

In the proposed code, the cell values are points of an n-dimensional lattice inside the cube $(0, q-1)^n$. To allow rewriting, there is a one-to-many mapping between from the information to the codebook. To encode an information sequence, the encoder searches over the candidate codewords and selects one. To aid this encoding and search, the codebook is partitioned into subcodebooks, most with a one-to-one mapping (some subcodebooks at the boundary may have fewer codewords than possible information sequences). Adding a random component to the encoding will improve the average number of writes. The proposed code construction is detailed in Section II. Since the average number of writes appears to be difficult to evaluated analytically, numerical evaluation is presented in Section III, where there is a clear tradeoff between the code rate and the average number of writes.

The paper concludes with Section IV, which discusses the

This research was supported in part by the Ministry of Education, Science, Sports and Culture; Grant-in-Aid for Scientific Research (C) number 21560388.

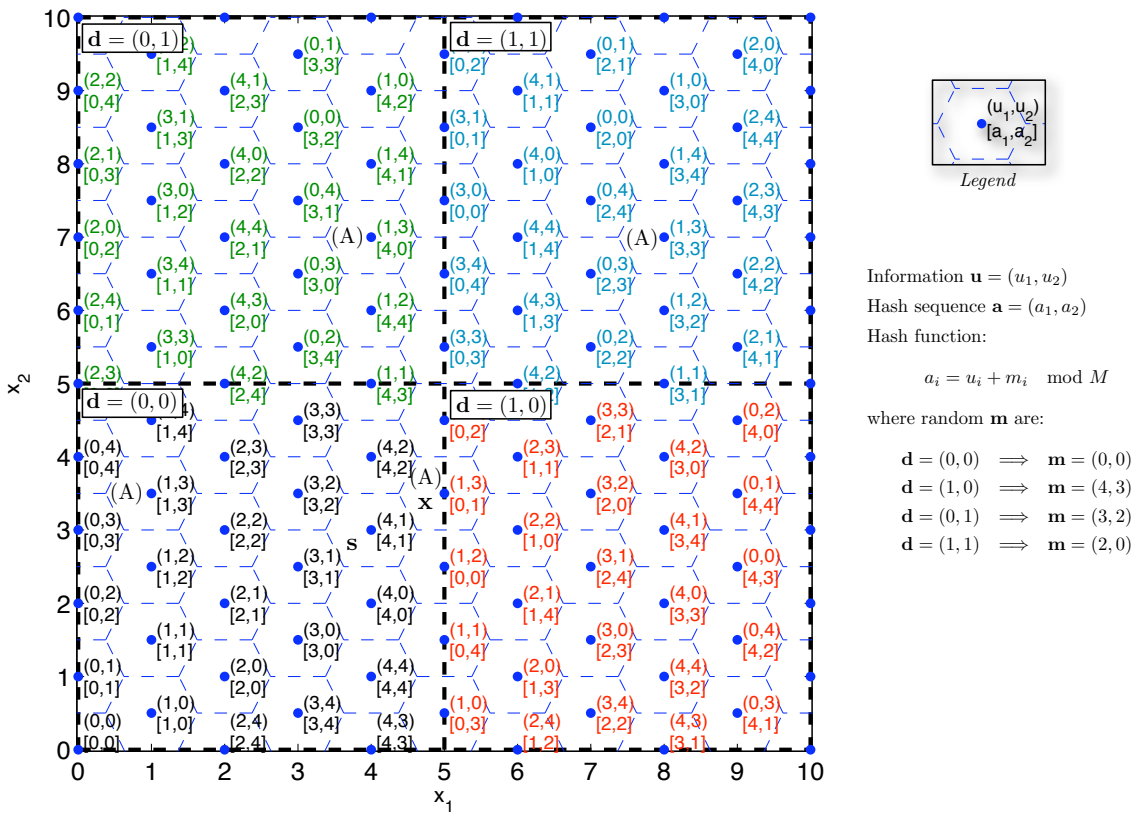

Fig. 1. Illustration of the proposed code for two dimensions, $n = 2, G = [1\ 0; \frac{1}{2}\ 1], M = 5, D = 2$. Refer to Subsection II-E for an example using this construction.

hypothesis that the number of rewrites depends upon the "depth" q of the cell more so than number of cells n. In addition, lattices have a significant error-correction capability, and the use of lattices for both rewriting and error-correction is discussed.

II. Code Construction

A. Lattices

An n-dimensional lattice Λ is defined by an n-by-n generator matrix G. The lattice consists of the discrete set of points $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$ for which

$$\mathbf{x} \quad = \quad G\mathbf{b}, \tag{1}$$

where $\mathbf{b} = (b_1, \ldots, b_n)^t$ is from the set of all possible integer vectors, $b_i \in \mathbb{Z}$. The Voronoi region is region of $\mathbb{R}^n$ which is closer to $\mathbf{x}$ than to any other point, and the volume of this region is the determinant of G:

$$V(\Lambda) \quad = \quad |\det G|. \tag{2}$$

The i, j entry of G is denoted g_{ij}.

B. Codebook

An overview of the codebook construction follows. The code is described by a lattice Λ with a lower-triangular generator matrix G, and two parameters D and M. The codebook consists of the intersection of Λ and a cube of volume $(DM)^n$. It is allowed that D is not an integer, and let $\overline{D} = \lceil D \rceil$. The codebook is partitioned into $\overline{D}^n$ subcodebooks, each the intersection of Λ and, if D is an integer, a cube of volume M^n. If D is not an integer, then some subcodebooks do not correspond to cubes. Also, M must satisfy the condition that that M/g_{ii} is an integer, where g_{ii} are the diagonal entries from the generator matrix, for $i = 1, \ldots, n$. Throughout this paper, it is assumed that:

$$D \cdot M = q - 1, \tag{3}$$

where q is an integer, which will satisfy the constraint that that each component of the lattice point is in the range $(0, q-1)$.

More precisely, the codebook construction is given as follows. Let B be an n-cube, given by:

$$0 \le x_i < D \cdot M, \tag{4}$$

for $i = 1, \ldots, n$, which has volume $(DM)^n$. Then the codebook of the proposed code is:

$$\mathcal{C} = \Lambda \cap B. \tag{5}$$

Subcodebooks are created by partitioning the cube B into $\overline{D}^n$ blocks. The blocks are indexed by $\mathbf{d}$, given by:

$$\mathbf{d} \quad = \quad \{d_1, d_2, \ldots, d_n\}, \text{ with } d_i \in \{0, 1, \ldots, \overline{D} - 1\}. \tag{6}$$

Each block $B_{\mathbf{d}}$ is given by the set of $\mathbf{x} \in \mathbb{R}^n$ such that:

$$d_i M \le x_i < (d_i + 1)M \text{ and } x_i < DM, \tag{7}$$

for $i = 1, \ldots, n$. If D is an integer, then each block has volume M^n. If D is not an integer, then blocks with one or more index components d_i equal to $\overline{D}-1$ will have volume less than M^n. Then the subcodebook $\mathcal{C}_{\mathbf{d}}$ consists of the lattice points inside each block:

$$\mathcal{C}_{\mathbf{d}} \quad = \quad \Lambda \cap B_{\mathbf{d}}. \tag{8}$$

The number of elements of the full codebook and the maximum number of elements of any subcodebook are:

$$\frac{(D \cdot M)^n}{|\det G|} \quad \text{and} \quad \frac{M^n}{|\det G|}, \tag{9}$$

respectively. Within each block with volume M^n, there is a one-to-one mapping from information to subcodewords, thus the rate of the code, expressed in information bits per cell is:

$$R \quad = \quad \frac{\log_2 \left(M^n/|\det G|\right)}{n} = \log_2 M, \tag{10}$$

if $|\det G| = 1$ is used. Also, there is a one-to-many mapping between information and the full codebook.

C. Encoding

An overview of the encoding scheme is as follows. An information sequence $\mathbf{u}$ is mapped randomly, but invertibly, to a "hashed" integer sequence $\mathbf{a}$. Then, $\mathbf{a}$ is encoded into multiple subcodebooks $\mathcal{C}_{\mathbf{d}}$, for various values of $\mathbf{d}$. Considering the current state of the memory, $\mathbf{s}$, the encoding from one of the subcodebooks, called $\mathbf{x}$, is selected and written to memory. The codebooks and the encoding is illustrated in Fig. 1 for $n = 2$, and an example is described in Subsec. II-E.

The details of the encoding follow. The information is the vector of integers $\mathbf{u} = (u_1, \ldots, u_n)$. The data range for each u_i depends on g_{ii}, with $a_i \in \{0, 1, \ldots, \frac{M}{g_{ii}} - 1\}$, so M/g_{ii} must be an integer. Note that the product of the diagonal entries of the triangular matrix G is equal to $|\det G|$.

A random "hash" h maps information $\mathbf{u}$ to hashed sequences $\mathbf{a} = (a_1, \ldots, a_n)$. This hash depends upon $\mathbf{d}$:

$$h(\mathbf{d}) : \mathbf{u} \to \mathbf{a}. \tag{11}$$

The purpose of the hash is to introduce randomness to the subcodes to increase the average number of writes. Suppose that the current state $\mathbf{s}$ is in block $\mathbf{d}'$ and that the information sequence $\mathbf{u}$ in this block maps to $\mathbf{x}[\mathbf{d}']$. If $\mathbf{x}[\mathbf{d}'] - \mathbf{s}$ has any negative components, then $\mathbf{x}[\mathbf{d}']$ cannot be selected. However, due to the nature of the encoding scheme, the encoding of the same $\mathbf{u}$ in some adjoining block, e.g. $\mathbf{d}''$, may also have negative components in $\mathbf{x}[\mathbf{d}''] - \mathbf{s}$, with some significant probability, and similarly $\mathbf{x}[\mathbf{d}'']$ cannot be selected. By introducing block-dependent randomization, this problem is avoided.

A simple hash is simply to add a random constant modulo M/g_{ii}:

$$a_i \quad = \quad u_i + m_{i,\mathbf{d}} \mod \frac{M}{g_{ii}}, \tag{12}$$

where $\mathbf{m_d}$ is a hash vector for block $\mathbf{d}$,

$$\mathbf{m_d} = \left(m_{1,\mathbf{d}}, m_{2,\mathbf{d}}, \cdots, m_{n,\mathbf{d}}\right). \tag{13}$$

The hash vector element $m_{i,\mathbf{d}}$ is selected with uniform probability from $\{0, 1, \ldots, \frac{M}{g_{ii}} - 1\}$.

The sequence $\mathbf{a}$ is then then encoded to a lattice point $\mathbf{x}[\mathbf{d}] \in \mathcal{C}_{\mathbf{d}}$ for any block $\mathbf{d}$, as follows. This method is a slight generalization of an existing method [12], in that non-unity elements are allowed on the diagonal of the generator matrix.

In general, $G \cdot \mathbf{a}$ is not in $B_{\mathbf{d}}$. Instead, the encoding finds

$$\mathbf{b} \quad = \quad \mathbf{a} + M\mathbf{k}, \tag{14}$$

with $\mathbf{k} = (\frac{k_1}{g_{11}}, \ldots, \frac{k_n}{g_{nn}})$ such that

$$\mathbf{x} = G \cdot \mathbf{b} \tag{15}$$

is in the cube $B_{\mathbf{d}}$. Define $\delta(d_i)$ as:

$$\delta(d_i) = \min\Big((d_i + 1)M, DM\Big). \tag{16}$$

Because the generator matrix is lower triangular, the k_i can be found by solving the inequality (7):

$$d_i M \leq \quad \sum_{j=0}^{i-1} g_{ji} b_j + g_{ii}\big(a_i + \tfrac{M}{g_{ii}} k_i\big) \quad < \delta(d_i) \tag{17}$$

for k_i, which is unique. First k_1, then $k_2, \ldots, k_n$ are found in sequence. In particular:

$$k_i \quad = \quad \Big\lceil \frac{d_i M - \sum_{j=0}^{i-1} g_{ji} b_j - g_{ii} a_i}{M} \Big\rceil, \tag{18}$$

where the computation at step i depends upon $b_1, \ldots, b_{i-1}$. Note that if D is not an integer, then some subcodebooks do not have images of all information sequences, and encoding in such cases is not possible.

It is necessary to find the codeword $\mathbf{x}$ to write to memory, from the various candidates $\mathbf{x}[\mathbf{d}]$. The current state of the memory is $\mathbf{s}$. So, for any such codeword $\mathbf{x}$, all components of $\mathbf{x} - \mathbf{s}$ must be positive. Since there is no a priori knowledge about future data sequences, it is reasonable that the codeword choice should maximize the number of codeword points that remain "available" to future writes, that is, the number of codewords in the positive direction should be maximized. While it is computationally difficult to count these points, a reasonable approximation is the volume that remains after the point is written. In particular, if $\mathbf{x}$ is to be written, then the remaining volume is:

$$\prod_{i=1}^{n} \big(D \cdot M - x_i\big) \tag{19}$$

and the encoder should write:

$$\mathbf{x} \quad = \quad \max_{\mathbf{d}:(\mathbf{s}-\mathbf{x}[\mathbf{d}]) \geq \mathbf{0}} \prod_{i=1}^{n} \big(D \cdot M - x_i[\mathbf{d}]\big). \tag{20}$$

This maximization is computationally complex as the lattice dimension n increases. Generally, however there will be a codeword in a neighboring block. Thus, the search can be performed not over all $\mathbf{d}$, but only over those positive neighbors of the block that contains the current state $\mathbf{s}$. This results in complexity proportional to 2^n.

Note that many conventional q-ary rewriting codes allow rewriting one *bit* at a time. For this lattice-based code, the entire *word* is rewritten.

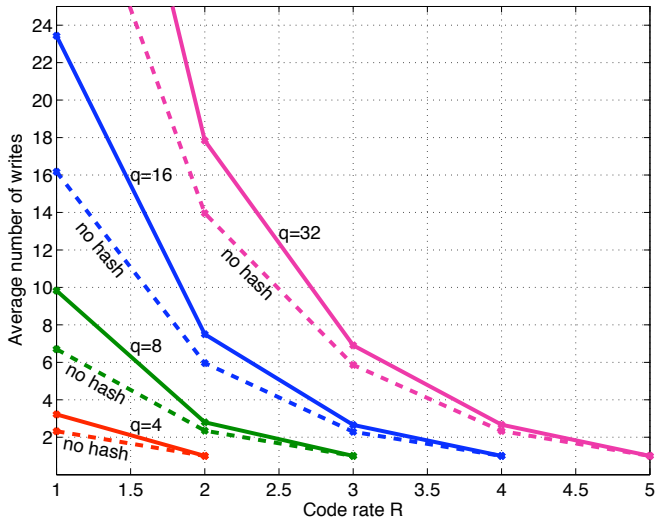

Fig. 2. Average number of word writes using the E8 lattice, with $q-1 = DM$ and code rate $R = \log_2 M$.

D. Decoding

Decoding is straightforward. Let the decoder input be the lattice point $\widehat{\mathbf{x}}$. The encoded integers are simply $\widehat{\mathbf{b}} = G^{-1}\widehat{\mathbf{x}}$, and from these, $\widehat{\mathbf{a}}$ is obtained as:

$$\widehat{a}_i \quad = \quad \widehat{b}_i \mod \frac{M}{g_{ii}}, \text{for } i = 1, \ldots, n. \tag{21}$$

The information sequence $\widehat{\mathbf{u}}$ is obtained by inverting the hash function:

$$\widehat{u}_i \quad = \quad \widehat{a}_i - m_{i,\mathbf{d}} \mod \frac{M}{g_{ii}}, \tag{22}$$

where $m_{i,\mathbf{d}}$ is defined as before.

E. Example

Here, an example of encoding is given, based upon the code in Fig. 1, which has $n = 2$ $G = [1\ 0; \frac{1}{2}\ 1]$, $M = 5$ and $D = 2$. This corresponds to $q = 11$. This code can write $\log_2 5$ bits per cell into 2 flash cells, where the cell can store values between 0 and 10. The information symbols $\mathbf{u}$ are from the alphabet $\{0,1,2,3,4\} \times \{0,1,2,3,4\}$.

Assume that the state of the memory is $\mathbf{s} = (4,3)$ (indicated in the figure), and the information to be written is $\mathbf{u} = (1,3)$. The hash vectors are $\mathbf{m}_{0,0} = (0,0)$, $\mathbf{m}_{1,0} = (4,3)$, $\mathbf{m}_{0,1} = (3,2)$ and $\mathbf{m}_{1,1} = (2,0)$.

For each $\mathbf{d}$, the hashed value is computed, and the candidate vector is found:

$$\begin{aligned}
&\mathbf{d} = (0,0) : \mathbf{u} = (1,3) \rightarrow \mathbf{a}[\mathbf{d}] = (1,3) \rightarrow \mathbf{x}[\mathbf{d}] = (1,3.5)\\
&\mathbf{d} = (0,1) : \mathbf{u} = (1,3) \rightarrow \mathbf{a}[\mathbf{d}] = (0,1) \rightarrow \mathbf{x}[\mathbf{d}] = (5,3.5)\\
&\mathbf{d} = (1,0) : \mathbf{u} = (1,3) \rightarrow \mathbf{a}[\mathbf{d}] = (4,1) \rightarrow \mathbf{x}[\mathbf{d}] = (4,7)\\
&\mathbf{d} = (1,1) : \mathbf{u} = (1,3) \rightarrow \mathbf{a}[\mathbf{d}] = (3,3) \rightarrow \mathbf{x}[\mathbf{d}] = (8,7).
\end{aligned}$$

These $\mathbf{x}[\mathbf{d}]$ are indicated in Fig. 1 by "(A)". For the first candidate $(1,3.5)$, the difference $\mathbf{x}[(0,0)] - \mathbf{s}$ is negative in the first component, and so this point cannot be written. For each of the remaining three, the product $\prod_i (DM - x_i[\mathbf{d}])$ is computed. Candidate point $\mathbf{x}[(0,1)]$ maximizes this product, and so it is selected as the point to be written, $\mathbf{x}$ (indicated in the figure).

III. Numerical Results

In order to make a fair normalization in the absence of noise, the scale of the lattice must be selected. For conventional q-ary rewriting codes, the rectangular lattice with integer spacing applies; the volume of the Voronoi region of this lattice is 1. The same scaling is applied to the lattice Λ. That is, a scalar α is selected such that:

$$|\det \alpha G| = \alpha^n |\det G| = 1. \tag{23}$$

It should be fairly clear that the minimum number of guaranteed writes is $\lfloor D \rfloor$. In a worst-case scenario, a codeword is written in block $\mathbf{d} = (0, \ldots, 0)$ followed by $\mathbf{d} = (1, \ldots, 1)$ until $\mathbf{d} = (D-1, \cdots, D-1)$. This may be visualized in Fig. 1 by first writing a codeword near the upper-right-hand corner of block $(0,0)$, and then $(1,1)$. Since a fractional number of minimum writes is not meaningful, $\lfloor D \rfloor$ is written.

To evaluate the average number of writes, the E8 lattice is used. This lattice, with dimension $n = 8$, has good packing properties, as well as an efficient decoding algorithm [13], and one possible generator is:

$$G \quad = \quad \begin{bmatrix}
1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1/2 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
1/2 & -1 & 1 & 0 & 0 & 0 & 0 & 0\\
1/2 & 0 & -1 & 1 & 0 & 0 & 0 & 0\\
1/2 & 0 & 0 & -1 & 1 & 0 & 0 & 0\\
1/2 & 0 & 0 & 0 & -1 & 1 & 0 & 0\\
1/2 & 0 & 0 & 0 & 0 & -1 & 1 & 0\\
1/2 & 0 & 0 & 0 & 0 & 0 & -1 & 2
\end{bmatrix} \tag{24}$$

It has a lower-triangular form, and so it is suitable for the proposed construction.

Naturally, there is a tradeoff between code rate and the average number of writes, and this is demonstrated in Fig. 2, obtained by computer simulation. Values of q were fixed, with $q - 1 = DM$; powers of two were chosen for q, but this is not a requirement. The code rate $R = \log_2 M$, and D was allowed to be a non-integer. The most striking feature is that the number of writes depends strongly upon q. Also, while not shown here, it was observed numerically that the average number of writes increased roughly linearly in D, much as the minimum number of writes is also linear in D.

Also shown in Fig. 2 is the average number of writes if hashing is not used, that is, the hash vector $\mathbf{m}$ is all-zeros. At low rates, the random hash increases the average number of writes. But as the rate increases, this advantage diminishes. Note that the hash has no influence on the minimum number of writes.

IV. Discussion

A point to note is that the rewriting capability of lattices presented in this paper does not appear to substantially depend

upon the dimension n. That is, the minimum number of writes is $\lfloor D \rfloor$, and there is a well-defined relationship between D, and R, q and M; but not n. However, the lack of dependence on n appears to not be surprising. In 1984, Fiat and Shamir, working with very general memory models, those based upon directed acyclic graphs (DAG), observed: "The significant improvement in memory capability is linear with the DAG depth. For a fixed number of states a 'deep and narrow' DAG cell is always preferable to a 'shallow and wide' DAG cell" [14]. That is, a deep cell has a large value of q, and a narrow cell has small n. Likewise, for the lattice-based codes presented in this paper, increasing q strongly increases the number of rewrites.

When q and n are both small, the lattice-based construction does have a weakness, because the maximum value cannot be written in all cells; conventional q-ary floating codes do not have this problem. This is a type of boundary problem, where there is an inefficiency at the boundary where some cells cannot be written. However, as either q or n increases, the efficiency increases, and the problem can become negligible.

Another point is that lattices have an inherent error-correction property, although that was not the subject of this paper. While rewriting codes for flash memories have received some research attention, error-correction coding for flash memories is of considerable practical importance [15] [16]. There have been only a few studies on dual-purpose codes which can both correct errors and allow rewriting [17] [18] [19]. Unfortunately, the simple concatenation of a rewriting code and an error-correction code appears to be problematic. Encoding the rewriting code followed by a systematic error-correction code means that parity bits are not rewritable. On the other hand, switching the concatenation results in no guarantees of minimum distance, since most rewriting codes do not appear to be systematic. However, lattices considered in this paper have a natural error-correction property, due do the Euclidean distance that separates the points. Thus, it appears that lattices may be used simultaneously for both rewriting and error-correction in flash memories. One lattice which appears to be particularly attractive for flash memories are low-density lattice codes [10], which have high coding gain and an efficient decoding algorithm [20].

REFERENCES

[1] R. L. Rivest and A. Shamir, "How to reuse a 'write-once" memory,' *Information and Control*, vol. 55, pp. 1–19, December 1982.

[2] G. D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Transactions on Information Theory*, vol. 32, pp. 697–700, September 1986.

[3] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1166–1170, June 2007.

[4] V. Bohossian, A. Jiang, and J. Bruck, "Buffer coding for asymmetric multi-level memory," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 1186–1190, June 2007.

[5] E. Yaakobi, A. Vardy, P. H. Siegel, and J. K. Wolf, "Multidimensional flash codes," in *Proceedings 46th Annual Allerton Conference on Communication, Control, and Computing*, (Monticello, IL, USA), pp. 392–399, September 2008.

[6] H. Finucane, Z. Liu, and M. Mitzenmacher, "Designing floating codes for expected performance," in *Proceedings 46th Annual Allerton Conference on Communication, Control, and Computing*, (Monticello, IL, USA), September 2008.

[7] A. Jiang and J. Bruck, "Information representation and coding for flash memories," in *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, pp. 920–925, August 2009.

[8] H.-A. Loeliger, "Averaging bounds for lattices and linear codes," *IEEE Transactions on Information Theory*, vol. 43, pp. 1767–1773, November 1997.

[9] U. Erez and R. Zamir, "Achieving $\frac{1}{2}\log(1 + \text{SNR})$ on the AWGN channel with lattice encoding and decoding," *IEEE Transactions on Information Theory*, vol. 50, pp. 2293–2314, October 2004.

[10] N. Sommer, M. Feder, and O. Shalvi, "Low-density lattice codes," *IEEE Transactions on Information Theory*, vol. 54, pp. 1561–1585, April 2008.

[11] F. Sun, S. Devarajan, K. Rose, and T. Zhang, "Design of on-chip error correction systems for multilevel nor and nand flash memories," *Circuits, Devices Systems, IET*, vol. 1, pp. 241–249, June 2007.

[12] N. Sommer, M. Feder, and O. Shalvi, "Shaping methods for low-denisty lattice codes," in *Proc. Information Theory Workshop, 2009*, pp. 238–242, October 2009.

[13] J. H. Conway and N. J. A. Sloane, "A fast encoding method for lattice codes and quantizers," *IEEE Transactions on Information Theory*, vol. 29, pp. 820–824, November 1983.

[14] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *IEEE Transactions on Information Theory*, vol. 30, pp. 470–480, May 1984.

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

[16] B. Chen, X. Zhang, and Z. Wang, "Error correction for multi-level NAND flash memory using Reed-Solomon codes," in *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pp. 94–99, October 2008.

[17] G. Zemor and G. D. Cohen, "Error-correcting WOM codes," *IEEE Transactions on Information Theory*, vol. 37, pp. 730–734, May 1991.

[18] A. Jiang, M. Schwartz, J., and Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, pp. 2112–2120, May 2010.

[19] F. Zhang, H. D. Pfister, and A. Jiang, "LDPC codes for rank modulation in flash memories," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 859–863, June 2010.

[20] B. Kurkoski and J. Dauwels, "Reduced-memory decoding of low-density lattice codes," *IEEE Communications Letters*, vol. 14, pp. 659–661, July 2010.