Shaping Codes for Structured Data

Yi Liu* and **Paul H. Siegel***
*Electrical and Computer Engineering Dept., University of California, San Diego, La Jolla, CA 92093 U.S.A
{*yil333, psiegel*}*@ucsd.edu*

***Abstract*—In this work, we study data shaping codes for flash memory. We introduce data shaping codes for SLC flash memory. It reduces wear by minimizing the average fraction of programmed level. Then we extend this algorithm to MLC flash memory. It makes use of a page-dependent cost model and is designed to be compatible with the standard procedure of row-by-row, page-based, wordline programming. We also give simulation results demonstrating the performance of the direct shaping method when applied to English language text. Finally, we use random walk model to analysis its performance and error propagation property.**

I. Introduction

NAND flash memory has become a widely used data storage technology. It uses floating-gate transistor (commonly referred to as a cell) to store information. As mentioned in [1], NAND flash memory cells gradually wear out with program/erase (P/E) cyclling, but the damage caused by P/E cycling is dependent on the level programmed to the cell. The author then proposed a coding technique called Adaptive Endurance Coding which increases the number of P/E cycles that a Flash device and endure. In SLC flash memory, each cell has two different states, representing '1' or '0'. It is shown that storing '1' causes less damage than storing '0' in the cell. In order to increase the lifetime of the flash memory, one can reduce the average fraction of '0' in the encoded sequence. Data shaping for structured data can be viewed as a technique for combining lossless compression and endurance coding into a single encoding operation. The intent is to efficiently transform the structured data sequence directly into a sequence that induces less wear when programmed into a NAND flash memory. For binary (SLC) flash, this would translate into reducing the average fraction of symbols corresponding to the higher programmed cell level. (We will follow the usual convention of associating 1 with the erased level and 0 with the programmed level.)

In this paper, we consider the problem of finding data shaping codes for SLC and MLC Flash memory. In section II we introduce a shaping codes especially designed for SLC flash memory (Direct Shaping Codes). A simulation result is given using English Novel *Gone with the Wind*. In section III, we introduce the structure of MLC flash memory and we build a content-dependent cost model to reveal the damage of each level. Based on this, we extend shaping codes introduced in section II to MLC flash memory. Then in section IV, we analysis the error propagation property of Direct Shaping Codes. We find out if a sufficiently large number of words have been read correctly, we may avoid error propagation.

II. Shaping Codes for SLC Flash

A. Encoder and Decoder

change some 'mathbf' in this section to 'mathcal' Shaping Codes for SLC flash memory, or Direct shaping codes was first introduced by Sharon et al. [2]. It makes use of an adaptive encoding dictionary $\mathbf{D}$ with length m. $\mathbf{D}$ contains two lists, input list $\mathbf{X}$ and output list $\mathbf{Y}$. $\mathbf{X}$ is consist of data words $\mathbf{w_k} \in \{0,1\}^m$ and its count n_k in previous data sequence, denoted by $x_k = \{\mathbf{w_k}, n_k\}$, $k = 1, 2, \ldots, 2^m$. $\mathbf{X}$ is dynamically ordered according to n_k. Output list is consist of codewords $\mathbf{y_k} \in \{0,1\}^m$ and is ordered by increasing cost of $\mathbf{y_k}$ (i.e., increasing number of 0 symbols in $\mathbf{y_k}$). When a data words $\mathbf{w}$ of length m is encountered in the sequence, it is mapped to the binary length-m output word $\mathbf{y}$ that occupies the same position in output list $\mathbf{Y}$. Then the frequency count n of the word $\mathbf{w}$ is increased by 1 and the ordering of input list is updated accordingly.

Example 1. Consider encoding dictionary $\mathbf{D}$ with parsing length $m = 2$. The ordered output list $\mathbf{Y}$ is $\{11, 10, 01, 00\}$. After data sequence $\mathbf{W} = 101100101110$, which contain 3 $\mathbf{w_1} = 10$, 2 $\mathbf{w_2} = \mathbf{11}$ and 1 $\mathbf{w_3} = \mathbf{00}$. The dictionary is shown in Table I(a). The next data words $\mathbf{w} = 00$, which is mapped to $\mathbf{y_3} = 01$ in Table I(a). Then we add 1 to the count n_3 and update the dictionary, as shown in Table I(b).

TABLE I: Length-2 dictionary when encoding data words $\mathbf{w} = 00$

(a) Before Encoding

w	y	n
10	11	3
11	10	2
00	01	1
01	00	0

(b) After Encoding

w	y	n
10	11	3
00	10	2
10	01	2
01	00	0

The decoder will dynamically reconstructs the dictionary and inverts the encoder mapping. When a codeword $\mathbf{y}$ is encountered in the encoded sequence, it is mapped to the binary length-m data words $\mathbf{w}$ that occupies the same position in input list $\mathbf{X}$. Then the frequency count n of the word $\mathbf{w}$ is increased by 1 and the ordering of input list is updated accordingly. This rate-1 shaping code incurs no rate penalty and can be implemented with low complexity.

B. Simulation Results

We now present simulation results quantifying the endurance gain which can be achieved by the use of SLC data shaping codes. The structured data we used was English Novel *Gone with the Wind*. It was written in ASCII Code and had size about 2^{24} bits. We chose the parsing length of encoder to be 4 and 8. Fig. 1 shows the fraction of 0 symbols corresponding to the first 2^m bits in the original file and in the encoded files. The fraction of 0's in the original file is approximately 0.52. With parsing length equal to 4 bits, the fraction of 0's reduces to about 0.28, and with parsing length of 8 bits, the fraction

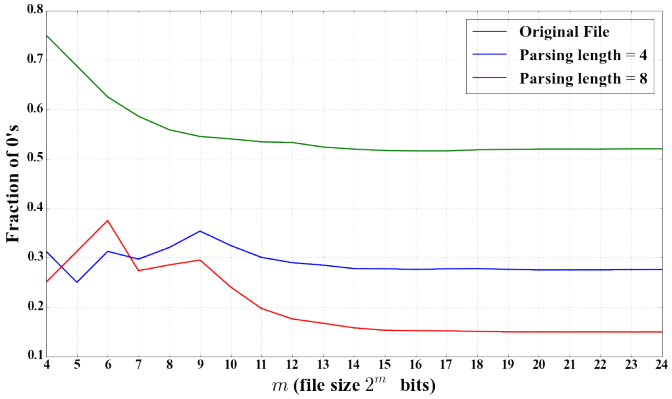

Fig. 1: Direct Shaping Codes applied to *Gone with the Wind*

is reduced to about 0.15. For data that is large enough, direct shaping codes is a optimal coding algorithm that can reach least cost.

III. Data Shaping Codes for MLC Flash

A. Cost Model for MLC Flash

Every cell in MLC flash memory can be charged to 4 different levels (2 bits/cell respectly.) They are denoted by $0, 1, 2, 3$ from lowest to highest and the corresponding binary representations in terms of lower-page and upper-page bits are '11', '10', '00', '01'. To program the MLC flash cell, the controller first charges the cell based on lower-page bit, then it charges the cell again for upper-page bit, as shown in Fig. 2.

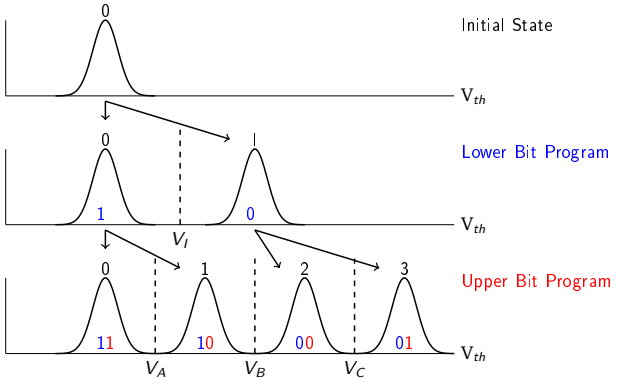

Fig. 2: Schematic of MLC flash cell programming.

To characterize and quantify the damage caused by different level in MLC Flash, we perform a experiment in which we repeatedly program MLC flash to a constant level. After every 100^{th} P/E cycles, we program MLC flash with pseudo-random data and record its bit error rate. The result is shown in Fig. 3. From Fig. 3, we see that cell damage caused by content '11', '10', '00' and '01' monotonically increase. We now introduce *cost model*, in the form of a vector of cell-level costs $[c_0, c_1, c_2, c_3]$, to quantifies the relative amount of device wear associated with each of the cell levels. In practice, these costs will satisfy $c_0 \leqslant c_1 \leqslant c_2 \leqslant c_3$, reflecting the increased damage induced by higher programmed levels.

A proper way to to calculate c_i,which was proposed by Li et al. in [3], is by measuring number of cycles it takes for

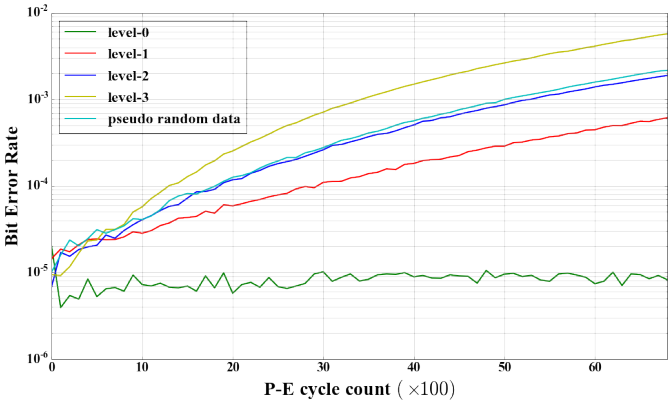

Fig. 3: Bit Error Rate of MLC Flash for different programming levels, while repeatedly programming a constant level

a certain level to reach the maximum tolerable bit error rate. Assume the designed lifetime for a MLC Flash Memory to be T_0 cycles. The maximum tolerable bit error rate BER_{max} is defined as the bit error rate after we program random data to flash memory for T_0 cycles. Let $\Phi^i(T)$ denote the bit error rate after we keep programming flash memory with the same content $i \in [0, 1, 2, 3]$. Let T^i_{max} denote the P/E cycle number under which the $\Phi^i(T)$ equals to BER_{max}. Hence we can estimate that the damage caused by programming i is proportional to $1/T^i_{max}$ and the damage caused by programming random content is proportional to $1/T_0$. So we can set the cost of each level to be

$$c_i = \frac{T_0}{T^i_{max}} \tag{1}$$

Example 2. We calculate the cost model given by Fig. 3. Assuming designed lifetime $T_0 = 4000$ cycles. The bit error rate of erasure state stays the same, so we can set $c_0 = 0$. Based on this Fig., we find that $T^1_{max} = 6800$, $T^2_{max} = 4300$ and $T^3_{max} = 2800$. So the cost model is $[0, 0.59, 1.07, 1.43]$

B. Shaping Codes for MLC Flash

Application of the SLC shaping code independently to lower and upper pages will not be effective in reducing the average cost. This can be seen by referring to the schematic of the MLC flash cell programming process in Fig. 2. As shown in the schematic, the cost associated with an upper bit 1 or 0 depends on the value of the corresponding lower bit in the cell. The proposed MLC shaping encoder achieves improved wear reduction by using lower-page dependent dictionaries when encoding the upper pages, as we now describe.

First, fix a parsing length m. Encoding and programming of wordlines is done in a row-by-row, sequential manner. For the lower pages, the encoder simply uses the direct shaping code with parsing length n. When programming an upper page, however, the encoder first reads the corresponding, previously programmed lower page.

Now, suppose the lower page that has been programmed and presumably correctly recovered by the encoder consists of the sequence of length-m codewords $\mathbf{y}^{(\mathbf{k})}$, $k = 0, \ldots, L$. Consider the sequence of length-m data words $\mathbf{w}^{(\mathbf{k})}$, $k = 0, \ldots, L$ that need to be encoded for the upper page. To encode the k th data

words $\mathbf{w}^{(\mathbf{k})}$, we encode using a direct shaping code that is based upon an adaptively-built dictionary that depends on the corresponding lower page codeword $\mathbf{v}^{(\mathbf{k})}$. The only difference in the operation of each lower-page dependent shaping encoder is that the ordering of the length-m encoder output words in terms of increasing cost depends specifically on the lower-page codeword $\mathbf{v}$ and the cost model $[c_0, c_1, c_2, c_3]$.

To illustrate the design of the encoder, we will use the cost model $[c_0, c_1, c_2, c_3] = [0, 1, 1, 2]$. This simple cost model is motivated as follows. Consider the standard lower-upper page binary representation of the cell levels: 0=11, 1=10, 2=00, 3=01. We assign a cost of 0 to lower bit value 1, and a cost of 1 to a lower bit value of 0, in accordance with the MLC cell programming schematic. When the lower bit value is 1, we assign a cost of 0 to upper bit value of 1, and a cost of 1 to upper bit value of 0. On the other hand, when the lower bit value is 0, we assign a cost of 0 to upper bit value of 0, and a cost of 1 to upper bit value of 1. Again, these cost assignments are consistent with the MLC cell programming schematic. The cost associated with a cell level is then defined as the sum of the corresponding lower bit and upper bit costs.

Choose parsing length $n = 4$ and assume the lower page codeword is $\mathbf{v} = \mathbf{1110}$. For the first three bits, where the corresponding lower bit is a 1, programming the upper bit to 1 is better than to 0. For the last bit, where the corresponding lower bit is a 0, programming the upper bit to 0 is better than to 1. This implies that the lowest cost output word in the dictionary is 1110, corresponding to cell level word 0002, which has total cost 1. Similar reasoning leads to the ordered list of output words shown in Table II. The corresponding list of cell level words is given in Table III. It is easy to verify that the costs of the corresponding cell level words are non-decreasing: cost 2 for words 1 to 4, cost 3 for words 5 to 10, cost 4 for words 11 to 14, and cost 5 for word 15. In general, the encoder uses 2^m direct shaping encoders operating in a sequence determined by the sequence of lower page codewords $\mathbf{v}^{(\mathbf{k})}$, $k = 0, \ldots, L$.

TABLE II: Ordered list of upper page words for lower page codeword 1110.

index	Output List	index	Output List
0	1110	8	1000
1	1111	9	0100
2	1100	10	0010
3	1010	11	1001
4	0110	12	0101
5	1101	13	0011
6	1011	14	0000
7	0111	15	0001

C. Encoding Algorithm for Shaping Codes

For a given length-m codewords $\mathbf{y}$, we denote the cost of every bit by $u_i \in [c_0, c_1, c_2, c_3]$, $i = 1, 2, \ldots, m$. The total cost is denoted by $U_{\mathbf{y}} = \sum_{i=1}^{m} u_i$. The output words must be ordered by increasing cost and easily indexed. This can be done using enumerative coding introduced by [4]. In his paper, Cover denoted by $n(x_1, x_2, \cdots x_k)$ the number of codewords $\mathbf{k}$ for which the first k coordinates are given by $(x_1, x_2, \cdots, x_k)$.

TABLE III: Corresponding list of cell level words for lower page codeword 1110.

index	Output List	index	Output List
0	0002	8	0112
1	0003	9	1012
2	0012	10	1102
3	0102	11	0113
4	1002	12	1013
5	0013	13	1103
6	0103	14	1112
7	1003	15	1113

To calculate $n(x_1, x_2, \cdots, x_k)$, we can calculate the polynomial $(x^{c_0} + x^{c_1})^{n_1}(x^{c_2} + x^{c_3})^{n_0}$, where n_b, $b \in \{0, 1\}$ is the number of bits equal to b in the remaining lower page codeword $\mathbf{v}(\mathbf{k}+\mathbf{1}, \cdots, \mathbf{n})$. The possible costs and the number of output words with each cost are given by the exponents and coefficients. Example 3 shows how to calculate $n(x_1, x_2, \cdots, x_k)$ and algorithm for building the encoding dictionary is given in algorithm 1.

Example 3. We want to calculate $n(1, 1)$ given that lower page codewords is $\mathbf{v} = \mathbf{1110}$ and total cost is 2. The cost of first two bits are 0 so the remaining cost is 2. We calculate the polynomial $(x^{c_0} + x^{c_1})^{n_1}(x^{c_2} + x^{c_3})^{n_0} = (1 + x)(x + x^2) = x + 2x^2 + x^3$. The coefficient of x^2 is 2, meaning there are two possible codewords whose total cost $U = 2$ and ther first two bits are 11.

Algorithm 1 Building output list based on lower page codeword

Require: Parsing length n and lower page codeword $\mathbf{v_0}$
Ensure: Encoding dictionary

1: Decide the number of 1's and 0's in the lower page, denoted as n_1 and n_0
2: Calculate polynomial $(x^{c_0} + x^{c_1})^{n_1}(x^{c_2} + x^{c_3})^{n_0}$
3: Decide all possible weight $w_i$and the number of codeword that has this weight n_i.
4: If index $I > \sum_{m=1}^{k-1} n_i$ and $I \leqslant \sum_{m=1}^{k} n_i$, the codeword $\mathbf{v_I}$ has weight K.
5: If $I > n(0)$ see $x_1 = 1$ and set $I = I - n(0)$, otherwise see $x_1 = 0$
6: For i th bit, if $I > n(x_1, \cdots, x_i - 1, 0)$ see $x_i = 1$ and set $I = I - n(x_1, \cdots, x_i - 1, 0)$, otherwise see $x_i = 0$

D. Simulation Results

Fig. 5 can be used to assess the performance of the MLC shaping encoder when applied to the English novel *Gone with the Wind*. We first divided it into two consecutive pieces of size 2^{23} bits; the first was used for lower page encoding and the second for upper page encoding. We applied MLC shaping encoders corresponding to the first 2^m bits in both file. For each file size, the plot in Fig. 5 shows the corresponding fractions when the proposed MLC shaping encoder is applied. Using these results, we see that for file size 2^{24}, the average cost after MLC shaping is 0.53. In contrast, when no coding is applied, as shown in Fig. **??**, the average cost is 0.89, and when SLC shaping is applied independently to lower and upper pages, the average cost is 0.62. For more realistic cost models in which level 2 is more costly than level 1, the relative gain from MLC shaping will be even larger.

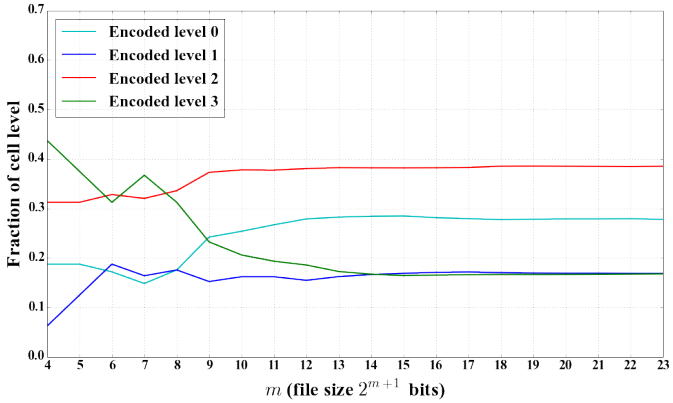

Fig. 4: Fractions of MLC cell levels for segments of *Gone with the Wind* without MLC shaping code.

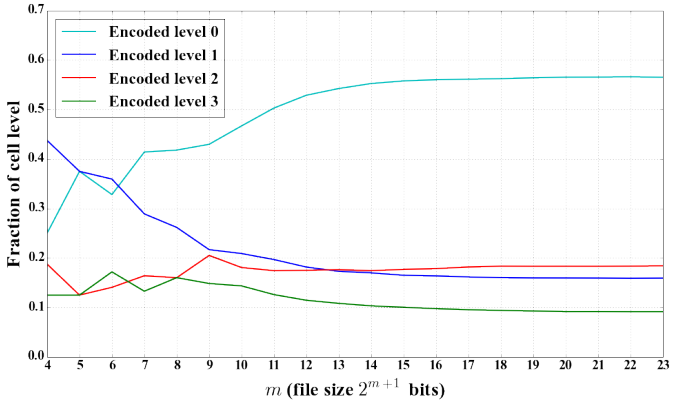

Fig. 5: Fractions of MLC cell levels for segments of *Gone with the Wind* with MLC shaping code.

IV. Performance and Error Propagation Analysis

change all p in this section to P. To analysis the performance and Error Propagation of direct shaping codes, we consider a length-m dictionary with 2^m symbols. We denote by t the total number of data words been encoded. Let $\mathbf{N}(t) = \{n_1(t), n_2(t), n_3(t), \ldots, n_{2^m}(t)\}$ be the word counts and $N_i(t) = n_i(t) - n_{i+1}(t)$ denotes the *distance* between the counts for the ith and $(i+1)$st words. Let $P = \{P_1, P_2, \ldots, P_{2^n}\}$ be the true probability distribution of words, and assume $P_1 \geqslant P_2 \geqslant \ldots \geqslant P_{2^n}$. We denote by $H(\mathcal{X})$ the entropy of the input words, $H(\mathcal{X}) = -\sum_i p_i \log p_i$. We say $N(t)$ is *stable* if $n_1(t) > n_2(t) > \ldots > n_{2^m}(t)$. The cost of codewords is $\mathcal{U} = [U_1, U_2, \ldots U_{2^m}]$ with $U_1 \leqslant U_2 \leqslant \ldots \leqslant U_{2^m}$.

A. Performance Analysis

Given the distribution P, we want to know the minimum cost we can get with rate-1 codes. This can be achieved by applying data compression and endurance codes. We have the following Theorem.

Theorem 1. *Given the distribution P and cost of codewords $\mathcal{U}$, the minimum cost we can get with rate-1 codes is $\sum_i \bar{P}_i U_i / m$. Where $\bar{P}_i = \frac{1}{Z} e^{-\mu U_i}$, μ is a positive constant selected such that $-\sum_i \bar{P}_i \log \bar{P}_i = H(\mathcal{X})$ and Z is a normalization constant.*

Proof: First we do a data compression and get the unstructured source, the compression ratio is $H(\mathcal{X})/m$. Then we do a endurance codes with expansion factor $f = m/H(\mathcal{X})$. From [1], we know the optimal distribution is $\bar{P}_i = \frac{1}{z} e^{-\mu c_i}$ where μ is a positive constant selected such that $-f \sum_i \bar{p}_i \log \bar{p}_i = m$, which means $-\sum_i \bar{P}_i \log \bar{P}_i = H(\mathcal{X})$, and Z is a normalization constant. So the best endurance gain we can get is $\sum_i \bar{P}_i U_i / m$ ∎

Theorem 2. *Given the distribution P and cost of codewords $\mathcal{U}$, the average cost we can get from direst shaping codes is $\sum_i P_i U_i / m$*

Proof: For any two symbols i j in the dictionary, first we assume $P_i > P_j$. Consider a sequence of i.i.d random variables $X_1^{ij}, X_2^{ij}, \ldots$ with $P(X_1^{ij} = 1) = p_i$, $P(X_1^{ij} = -1) = p_j$ and $P(X_1^{ij} = 0) = 1 - p_i - p_j$. The expected value of X_1^{ij} is $\mu^{ij} = p_i - p_j$ Consider $S_t^{ij} = \sum_{k=1}^{t} X_k^{ij}$. $S_t^{ij} > 0$ means $n_i(t) > n_j(t)$. By law of large numbers, $S_t^{ij}/t - \mu^{ij} \to 0$ almost surely. This means for any $\epsilon > 0$, $\{S_t^{ij}/t\}_0^\infty$ is within ϵ of $p_i - p_j > 0$ all but finitely many times. This means for any 2 symbols $i > j$, $n_i(t) > n_j(t)$ for all but finitely many times. So the dictionary is stable for all but finitely many times. Since we want to calculate the average cost, we only need to consider the situation when the dictionary is stable.

Consider a sequence of i.i.d random variables $Y_1, Y_2, \ldots$ with $P(Y_1 = U_k) = p_k$. The total cost after t steps is $W_t = \sum_{i=1}^{t} Y_i$, by law of large numbers, $W_t/t - \sum_{i=1}^{2^m} U_k P_k \to 0$ almost surely. So the average cost is $\sum_{i=1}^{2^m} U_k P_k / m$. It is easy to check that this statement is still true when $P_i = P_j$. ∎

From theorem 1 and 2, we can find out that direct shaping codes are sub-optimal. When $m = 1$, direct shaping codes can achieve minimum cost. When $m \geqslant 2$, direct shaping codes can achieve minimum cost only if $\frac{U_i}{\log p_i} = \text{const}$.

B. Error Propagation Property

The shaping decoder reproduces the dynamic construction of the encoding table. Errors in reading the flash memory can lead to incorrect word frequency counts that, in turn, can cause decoding errors if word counts are not sufficiently separated. We now consider the dictionary that contains only 2 symbols.

Choosing parsing length $m = 1$, the input word will be 1 or 0. $P = \{P_1, P_2\}$ is the true probability distribution of words 1 and 0, and $P_1 \geqslant P_2$.. Suppose that, at time t_0, the count of input words $n_1(t_0) \neq n_2(t_0)$. Since there are only 2 symbols, the dictionary after steps t can be represented by the symbol that has more counts, denoted by $s(t) \in \{0, 1\}$, and the distance $N(t) = n_1(t) - n_2(t)$. $N(t)$ acts like a one dimension random walk, shown in Fig. 6. Blue line denotes encoding process starting at t_0. If an error happens when reading the flash memory, for example at point A. The codeword is 0 and distance is $N^A(6) = 3$. Because of the error, the decoder reads the wrong codeword 1 and its distance becomes $N^{A'}(6) = 5$. After point A, the decoding dictionary, presented by red dash line, is separated from the encoding dictionary. The following three codewords is decoded correctly. After point B, the status of the encoder has changed, $s_e^B(t) = 0$, but the decoder stays the same, $s_d^{B'}(t) = 1$. The next input word is 0 and codeword is 1, but the decoded word is 1. This means every point after point D will be decoded incorrectly. This example shows that we want to avoid the situation when $N(t) = 0$ because

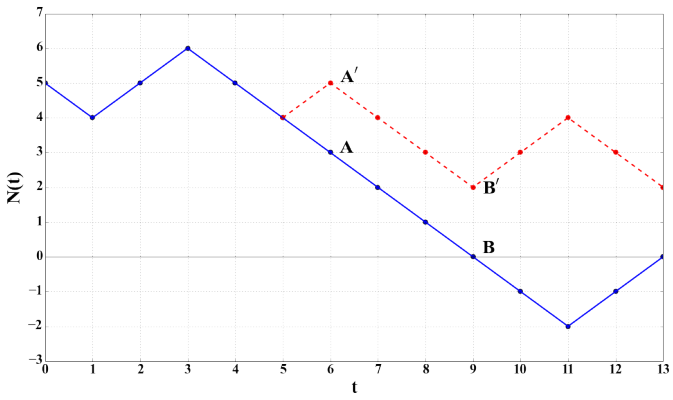

Fig. 6: Random walk in two symbols' dictionary

it may cause error propagation. For a length-m dictionary, at time t_0 $n_i(t_0) \neq n_j(t_0)$ for some symbols i and j. We say a *recurrence* occurs if, at some future time $t > t_0$, $n_i(t) = n_j(t)$.

Now we examine the recurrence property between these words 1 and 2. Assuming at some time t_0 the distance between words 1 and 2 is $N(t_0) = n_1(t_0) - n_2(t_0) \neq 0$, we want to know the probability that a recurrence occurs in the future. We use the theorem from [5].

Theorem 3. *If $P_1 \geqslant P_2$ and $N(t_0) \neq 0$ at some time t_0, the probability that a recurrence involving two words 1 and 2 will occur in the future is*

$$P = \begin{cases} \frac{P_2}{P_1}^{N(t_0)} & \text{if } N(t_0) > 0 \\ 1 & \text{if } N(t_0) \leqslant 0 \end{cases} \tag{2}$$

From theorem 3, we know that if we make sure $N(t_0)$ for some t_0 is large enough (the dictionary is stable), the probability that $N(t) = 0$ in the future is small and we can avoid the error propagation. We therefore need to know how many input words it takes for a dictionary to achieve stability, as well as the probability that the dictionary will subsequently become unstable. To study this problem, we reformulated the problem in terms of recurrence properties of a multidimensional random walk.

C. Recurrence of a multidimensional random walk

If the dictionary is not stable, theorem 3 tells that there must be a recurrence in the future. So in this part, we first assume that $\mathbf{N}(t_0)$ is stable for some t_0. Using theorem 3, we can build the upper bound of the recurrence probability of the whole dictionary. We denote by $\{i, i+1\}$ the event *symbol i, $i+1$ ever meet*, and by $\{\overline{i, i+1}\}$ its compliment. Notice that $i, i+1$ and $i+1, i+2$ is not independent, but $\{i, i+1\}$ and $\{i+2, i+3\}$ are independent. The probability that *a recurrence involving any two adjacent symbols in the dictionary* is $P\left\{(\bigcup_{i=1}^{2^{m-1}-1}\{i, i+1\})^C\right\}$.

Lemma 4. *Let $\mathbf{D}$ be a direct shaping dictionary with parsing length m. Assume it is stable at some t, ($\mathbf{N}(t)$ is stable). The probability that a recurrence involving any two symbols in the dictionary, denoted as P_W, is equal to the probability that a recurrence involving any two adjacent symbols in the dictionary.*

$$P_W = P\left\{(\bigcup_{i=1}^{2^{m-1}-1}\{i, i+1\})^C\right\} \tag{3}$$

Proof: We want to proof these two events are the same. Any recurrence condition is a permutation of set $\{1, 2, \ldots, 2^m\}$. So the set of all possible recurrence condition is the symmetric group S_{2^m}. Using cycle notation shown in [6] to represent the element in S_{2^m}, any recurrence between two adjacent symbols is $(i, i+1)$. Since S_{2^m} can be generated by $\langle (1,2), (2,3), \ldots, (k, k+1), \ldots, (2^m - 1, 2^m)\rangle$. These two events are the same, both are $S_{2^m}/\{\mathbb{1}\}$, where $\mathbb{1}$ is the identity element. ■

Lemma 5. *The probability recurrence happens between any symbols is bound by:*

$$P_W \leqslant 2 - \prod_{i=1}^{2^{m-1}} (1 - \frac{P_{2i}}{P_{2i-1}}^{N_{2i-1}(t)}) - \prod_{i=1}^{2^{m-1}-1} (1 - \frac{P_{2i+1}}{P_{2i}}^{N_{2i}(t)}) \tag{4}$$

Proof: Since $\{i, i+1\}$ and $\{i+1, i+2\}$ are not independent and $\{i, i+1\}$ and $\{i+2, i+3\}$, we can separate $\bigcup_{i=1}^{2^{m-1}-1}\{i, i+1\}$ into two events.

$$\begin{aligned} P_W &= P\left\{\bigcup_{i=1}^{2^{m-1}}\{i, i+1\}\right\} \\ &= P\left\{(\bigcup_{i=1}^{2^{m-1}}\{2i-1, 2i\})\bigcup(\bigcup_{i=1}^{2^{m-1}-1}\{2i, 2i+1\})\right\} \\ &\leqslant P\left\{\bigcup_{i=1}^{2^{N-1}}\{2i-1, 2i\}\right\} + P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{2i, 2i+1\}\right\} \end{aligned} \tag{5}$$

Since $\{i, i+1\}$ and $\{i+2, i+3\}$ are independent.

$$\begin{aligned} &P\left\{(\bigcup_{i=1}^{2^{m-1}}\{2i-1, 2i\})^C\right\} = P\left\{(\bigcap_{i=1}^{2^{m-1}}\{\overline{2i-1, 2i}\})\right\} \\ &= \prod_{i=1}^{2^{m-1}} P\{\overline{2i-1, 2i}\} = \prod_{i=1}^{2^{m-1}} (1 - \frac{P_{2i}}{P_{2i-1}}^{N_{2i-1}(t)}) \end{aligned} \tag{6}$$

$$\begin{aligned} &P\left\{(\bigcup_{i=1}^{2^{m-1}-1}\{2i, 2i+1\})^C\right\} = P\left\{(\bigcap_{i=1}^{2^{m-1}-1}\{\overline{2i, 2i+1}\})\right\} \\ &= \prod_{i=1}^{2^{m-1}-1} P\{\overline{2i, 2i+1}\} = \prod_{i=1}^{2^{m-1}-1} (1 - \frac{P_{2i+1}}{P_{2i}}^{N_{2i}(t)}) \end{aligned} \tag{7}$$

Combine (5), (6), (7), we have

$$\begin{aligned} P_W &\leqslant P\left\{\bigcup_{i=1}^{2^{m-1}}\{2i-1, 2i\}\right\} + P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{2i, 2i+1\}\right\} \\ &= 2 - P\left\{(\bigcup_{i=1}^{2^{m-1}}\{2i-1, 2i\})^C\right\} - P\left\{(\bigcup_{i=1}^{2^{m-1}-1}\{2i, 2i+1\})^C\right\} \\ &= 2 - \prod_{i=1}^{2^{m-1}} (1 - \frac{P_{2i}}{P_{2i-1}}^{N_{2i-1}(t)}) - \prod_{i=1}^{2^{m-1}-1} (1 - \frac{P_{2i+1}}{P_{2i}}^{N_{2i}(t)}) \end{aligned} \tag{8}$$

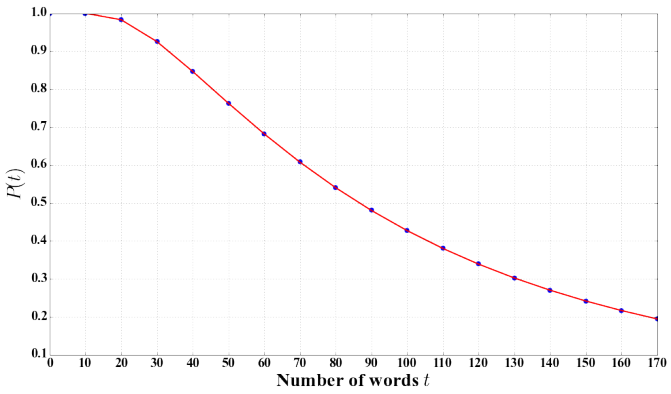

Fig. 7: An example of upper bound of $P(t)$ when $P = \{0.4, 0.3, 0.2, 0.1\}$

■

We denote by $A(\mathbf{N}(t))$ the right side of equation (8). Notice that $A(\mathbf{N}(t))$ will decrease if $N_i(t)$ increase, $i = 1, 2, \ldots, 2^m - 1$. $P(\mathbf{N}(\mathbf{t}))$ denotes the probability that after t steps the word counts is $N(t)$. Clearly

$$P(\mathbf{N}(t)) = \binom{t}{n_1(t), \ldots, n_{2^m}(t)} P_1^{n_1(t)} \ldots P_{2^m}^{n_{2^m}(t)} \quad (9)$$

Combine Lemma 5 and law of total probability, we have the following theorem.

Theorem 6. *After t data words be encoded, the probability that the recurrence occurs in the future is*

$$P(t) \leqslant \sum_{\substack{\mathbf{N}(t) \text{ is} \\ \text{stable}}} A(\mathbf{N}(t)) P(\mathbf{N}(t)) + \sum_{\substack{\mathbf{N}(t) \text{ is not} \\ \text{stable}}} P(\mathbf{N}(t)) \quad (10)$$

Fig 7 shows the upper bound of $P(t)$ when we set $m = 2$ and $P = \{0.4, 0.3, 0.2, 0.1\}$. From this figure we can find out $P(t)$ will decrease when t increase. This tells us that if we make sure the first t data words is decoded correctly, for example combining error correction codes with shaping codes, we can reduce the likelihood of decoder's error propagation.

V. Conclusion

In this paper, we first introduce a data shaping codes for SLC flash memory. Based on the structure of MLC flash memory, we build the cost model and expand this algorithm to MLC flash. Then we examine the error propagation property for direct shaping codes. Future work may contains simulation on different types of data on flash memory and combining shaping codes with error correction codes.

References

[1] Jagmohan, Ashish, et al. "Adaptive endurance coding for NAND Flash." GLOBECOM Workshops (GC Wkshps), 2010 IEEE. IEEE, 2010.

[2] E. Sharon, et al.,"Data Shaping for Improving Endurance and Reliability in Sub-20nm NAND," presented at Flash Memory Summit, Santa Clara, CA, August 4-7, 2014.

[3] Li, Jiangpeng, et al. "How much can data compressibility help to improve nand flash memory lifetime." Proceedings of 13th USENIX Conference on File and Storage Technologies (FAST). 2015.

[4] T.M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. 19, no. 1, pp. 73-77, Jan. 1973.

[5] Doyle, Peter G., and J. Laurie Snell. "Random walks and electric networks." Carus mathematical monographs 22 (2000).

[6] Dummit, David Steven, and Richard M. Foote. Abstract algebra. Vol. 1999. Englewood Cliffs: Prentice Hall, 1991.