Neighborhood Level Error Control Codes for Multi-Level Cell Flash Memories

Shohei KOTAKI†, Nonmember and Masato KITAKAMI†, Member

SUMMARY  NAND Flash memories are widely used as data storages today. The memories are not intrinsically error free because they are affected by several physical disturbances. Technology scaling and introduction of multi-level cell (MLC) has improved data density, but it has made error effect more significant. Error control codes (ECC) are essential to improve reliability of NAND Flash memories. Efficiency of codes depends on error characteristic of systems, and codes are required to be designed to reflect this characteristic. In MLC Flash memories, errors tend to direct values to neighborhood. These errors are a class of $M$-ary asymmetric symbol error. Some codes which reflect the asymmetric property were proposed. They are designed to correct only 1 level shift errors because almost all of the errors in the memories are in such errors. But technology scaling, increase of program/erase (P/E) cycles, and MLC storing the large number of bits can cause multiple-level shift. This paper proposes single error control codes which can correct an error of more than 1 levels shift. Because the number of levels to be corrected is selectable, we can fit it into noise magnitude. Furthermore, it is possible to add error detecting function for error of the larger shift. Proposed codes are equivalent to a conventional integer codes, which can correct 1 level shift, on a certain parameter. Therefore, the codes are said to be generalization of conventional integer codes. Evaluation results show information lengths to respective check symbol lengths are larger than nonbinary Hamming codes and other $M$-ary asymmetric symbol error correcting codes.

key words: Integer codes, asymmetric symbol error, generalization, NAND Flash memory, neighborhood level error

1. Introduction

NAND Flash Memories [1] are widely used for data storages because of its compactness, low power consumption, low cost, high data throughput, and high reliability. Technology scaling and introduction of multi-level cell (MLC) have reduced the bit cost of NAND Flash memories rapidly. This allows even more memory applications, such as solid-state drive (SSD). NAND Flash memories are not intrinsically error free. There are many physical cases that lead errors to the memories. Error control codes (ECC) are essential technique to deal with such errors and to improve system reliability. These characteristics are stated in [2]–[4], for example.

Flash memory cells consist of MOS transistor with floating-gate [1]. Figure 1 shows flash memory cell structure. A cell has floating-gate, control gate, source and drain [5]. The amount of charge injected into floating-gate determines threshold voltage. When cells are programed, high voltage is applied to control gate. Then electrons are injected from the substrate into the floating gate. When cells are erased, high voltage is applied to substrate. Then electrons are removed from the floating gate. Data read out is performed by applying specific readout voltage to control gate and observing drain-source current. If the voltage is higher than threshold voltage, then drain-source current can be observed [5].

Errors in NAND Flash memories are due to threshold voltage variation, and some types of errors are considered for Flash memories. Write errors occur when target cells or adjacent cells are programmed. Such errors cause higher threshold voltage than intended. Retention errors occur when the cell loses charge over time, inducing lower threshold voltage. Read-disturb errors occur when neighboring cells are read, causing higher threshold voltage [3].

There are two types of memory cells, single-level cell (SLC) and multi-level cell (MLC). SLC contains 1 bit information in a cell. On the other hand, MLC contains $n$ bits in a cell by using $2^n$ threshold voltage levels, where $n > 1$ [1]. MLC improves data density, but requires more precise programming and tight threshold voltage [6]. This implies that MLC with the larger number of states is more vulnerable to errors. Errors in MLC Flash memories have a typical characteristic, that is, errors direct values to neighborhood. Such errors are considered as a class of $M$-ary asymmetric symbol errors, where errors occur with unequal probabilities [7].

Some codes which correct specific case of asymmetric error, where values shift only 1 level, are proposed. Sato and Kitakami realized the function by taking advantage of Gray code for MLC Flash memories [8]. Also, Kostadinov et al. did it by constructing codes over integer ring, called integer codes [9]. These codes are originally proposed for communication systems, but applicable for MLC Flash memories. However, considering bit error rate increase accompanied by technology scaling and increase of program/erase (P/E) cycles [10], or MLC storing the large

---

Manuscript revised April 28, 2013.

†The authors are with the Graduate School of Advanced Integration Science, Chiba University, Chiba-shi, 263–8522 Japan.
a) E-mail: s_kotaki@chiba-u.jp
DOI: 10.1587/transinf.E96.D.1926

Copyright © 2013 The Institute of Electronics, Information and Communication Engineers
number of bits, multiple-level shift is not negligible. For example, error characteristics of 3x-nm technology MLC NAND Flash memory are studied experimentally in [11]. Although their results indicate that dominant error in MLC Flash memories is 1 level shift, approximately 5% of errors are 2 levels shift. This is for relatively narrow threshold voltage window in the tested Flash memory. Also, a mathematical model which consider occurrence of multiple-level shift is proposed in [12]. They model threshold voltage with Rayleigh distribution, using the results from [3] and [13].

This research considers codes with function between general class of asymmetric error correcting codes [14] and 1 level shift error correcting codes. Conventional codes which correct 1 level shift cannot correct multiple-level shift. On the other hand, general class of asymmetric error correcting codes can correct multiple-level shift, but they are not efficient. Thus, new error correcting codes are required.

This paper proposes single error control codes which can correct multiple-level shift within a certain selected range. The codes is constructed over integer ring because integer codes can correct errors of a given type [9]. In other words, we can select neighborhood values as correctable error patterns. The ring is selected as all correctable error values are unit to gain large code length. Also, error detecting function for the larger error can be added if it is needed.

This paper includes 5 sections. Section 2 introduces mathematical preliminaries to construct codes. The proposed codes are shown, and evaluated in Sects. 3 and 4. Conclusion is stated in Sect. 5.

2. Preliminaries

2.1 Asymmetric Error

We define $e$ levels shift by using the description in [14].

**Definition 1:** Let $u, e \in \{0, 1, \ldots, M\}$. An additive error, where a code symbol $u$ is changed to $u+e$ and satisfies $u+e \leq M$, or a subtractive error, where $u$ is changed to $u - e$ and satisfies $0 \leq u - e$, is called $e$ levels shift.

The errors based on definition 1 exhibit asymmetric error in MLC Flash memories. We also introduce more generalized class of asymmetric error, which can also stand for multiple-level shift.

**Definition 2:** [14] Let $A = \{a_0, a_1, \ldots, a_{M-1}\}$ be a set of $M$-ary symbols. An asymmetric symbol error set $\varepsilon$ is defined as follows:

$$\varepsilon = \{(a_i \rightarrow a_j)| a_i, a_j \in A, \Pr(a_i|a_j) > T, 0 \leq i \neq j \leq M - 1\},$$

where $(a_i \rightarrow a_j)$ means an error of $a_i$ being changed into $a_j$, $\Pr(a_i|a_j)$ is the probability of $(a_i \rightarrow a_j)$ and $T$ is a threshold error probability given in advance.

**Definition 3:** [14] Let $(a_i \rightarrow a_j) \in \varepsilon$. An error where $a_i$ is changed into $a_j$ is called an $\varepsilon$-asymmetric symbol error.

2.2 Integer Codes

**Definition 4:** [15] Let $Z_m$ be the ring of integer modulo $m$. An integer code of length $N$ with check matrix $H \in Z_m^{m \times N}$, is referred to as a subset of $Z_m^N$, defined by

$$C(H, d) = \{c \in Z_m^N | c \cdot H^T = d \mod m\}$$

where $d \in Z_m^M$ and $H^T$ is the transpose of $H$.

Without loss of generality, we assume $d = 0$.

**Definition 5:** [9] The code $C$ is said to be a $t$-$(\pm e_1, \pm e_2, \ldots, \pm e_s)$ error correcting code if it can correct up to $t$-errors with values $\pm e_i, i = 1, 2, \ldots, s$.

From Definition 5, proposed codes are said to be single $(\pm 1, \pm 2, \ldots, \pm s)$ error correcting codes. If the number of bits contained in a memory cell, that is $n$ in Sect. 1, increases, the number of error levels can also increase. Then it is required to set $s$ large so as to correct such error.

**Theorem 1:** A code $C$ is single $(\pm 1, \pm 2, \ldots, \pm s)$ error correcting code if and only if following relation holds for any distinct single $(\pm 1, \pm 2, \ldots, \pm s)$ errors $e_a, e_b$:

$$0 \neq e_a \cdot H^T \neq e_b \cdot H^T \neq 0,$$

where $H$ is a parity check matrix of the code $C$.

(Proof) The nonzero syndrome implies capability for error detection, and discrimination of syndromes implies capability for error correction. (Q.E.D.)

**Lemma 1:** An element of $Z_m$ is unit if it is not a zero divisor.

(Proof) Let $k$ be an element of $Z_m$ and not a zero divisor. If there exist nonzero distinct elements $\alpha, \beta$ which satisfy $k\alpha = l$ and $k\beta = l$, then $k(\alpha - \beta) = 0$. This is a contradiction. Thus $k\alpha$ and $k\beta$ are distinct regardless of the selection of $\alpha$ and $\beta$. Then, it is possible to select $\alpha$ such that $k\alpha = 1$. Therefore $k$ is a unit. (Q.E.D.)

**Lemma 2:** If order $m$ of ring $Z_m$ is multiple of primes larger than $s$, then $\pm 1, \pm 2, \ldots, \pm s$, which are elements of the ring, are units.

(Proof) Let $p$ be the minimum prime factor of $m$. If we suppose $k (1 \leq k < p)$ is a zero divisor, then there exists a nonzero integer $\alpha$ such that $k\alpha = 0$ mod $m$. Because $k$ is not a factor of $m$, $\alpha$ has to be multiple of $m$ to satisfy the equation. Such $\alpha$ is 0 over $Z_m$. This is a contradiction. Therefore, $k$ is not a zero divisor. It follows from Lemma 1 that $k$ is an unit. (Q.E.D.)

3. Neighborhood Level Error Control Codes

3.1 Code Construction

Code construction consists of three steps. Step 1 determines
levels for error correction. Step 2 makes three sets by assigning elements of \( Z_m \). Step 3 constructs check matrix using results of step 2.

**Step 1:** In this step, we determine \( s \) which is the levels to be corrected. To obtain large code length, it is preferable to let each correctable error value be an unit. Therefore, order of integer ring \( m \) should be multiple of primes considering Lemma 2. That is,

\[
m = p_1^{e_1} \cdot p_2^{e_2} \cdots \cdot p_n^{e_n} \quad (s < p_1 < p_2 < \cdots < p_n, e_j \geq 1),
\]

where each \( p_i \) is a prime.

For \( M \)-ary systems, if \( M = m \), then encoding and decoding is straightforward. However, there are many cases where we cannot let \( m \) be equal to \( M \). For example, when target system has \( M = 2^q \), \((\pm 2)\) error correcting code cannot be constructed for \( m = 2^q \) because any prime factor of \( m \) should be larger than 2. In this case, first we should let \( m \) be largest possible number which satisfies \( m < M \), because when \( m > M \), check symbols cannot be used in the system. Check symbols calculation and syndrome calculation should be performed with information symbols modulo \( m \). In this way, all error values to be controlled remain under control function, and appropriate error correction can be achieved. The encoding procedure is shown in Fig. 2.

**Step 2:** In this step, we make three sets, \( L \), \( O \) and \( U \) by assigning elements of \( Z_m \).

Set \( L \): Let \( E \) be a set containing correctable error values, that is, \( E = \{\pm 1 \pm 2 \cdots \pm s\} \). We determine \( L \) to satisfy

\[
\begin{align*}
xa &\neq xb \\
xa &\neq ya \\
xa &\neq yb
\end{align*}
\]

for any distinct \( x, y \in L \) and any distinct \( a, b \in E \). The following algorithm can make \( L \) satisfy the above conditions.

[Algorithm to determine \( L \)]

For an integer set \( S = \{a_1, a_2, \ldots, a_n\} \) and an integer \( x \), we define multiplication as \( xS = \{xa_1, xa_2, \ldots, xa_n\} \). Calculation is executed over \( Z_m \).

1) Let \( A = \{1, 2, \ldots, m - 1\} \), \( E = \{\pm 1, \pm 2, \ldots, \pm s\} \) and \( L = \phi \).
2) If \( A \) is a null set, then finish this algorithm.
3) Let \( x \) be an arbitrary element in \( A \).
4) If there exists such \( a, b \) that \( xa = xb \) for \( a, b \in E, a \neq b \), then eliminate \( x \) from \( A \) and back to 2).
5) If \( xE \not\subseteq A \), then eliminate \( x \) from \( A \) and back to 2).
6) If \( xE \subseteq A \), then eliminate elements of \( xE \) from \( A \).
7) Add \( x \) to \( L \) as an element and back to 2).

Set \( O \): Let \( O \) be a set of prime factors of \( m \) and all their multiples including 0. Moreover, the elements should not be included in the set \( L \). If \( m \) is a prime, Let \( O = \{0\} \).

Set \( U \): Let \( U \) equal \( Z_m \).

**Step 3:** In this step, we make three sets obtained in Step 2. Let \( r \) be check symbol length, and

\[
\begin{align*}
h_{i,j} &\in [(0_1, 0_2, \ldots, 0_{i-1}, l, u_1, u_2, \ldots, u_{r-l-1})]^T \\
&\quad | a_\phi, l \in L, u_\phi \in U, \}
\end{align*}
\]

and

\[
H_i = [h_{i,1}, h_{i,2}, \ldots, h_{i, l}]
\]

\[
1 \leq j \leq |O|^{|L|/|U|^{|L|/r_i-1}}
\]

where \( |O|, |L| \) and \( |U| \) are the numbers of elements corresponding to each set. Here, \( H_i \)'s are called submatrices of the code. The check matrix of the code is constructed as follows:

\[
H = [H_1, H_2, \ldots, H_r].
\]

**Theorem 2:** The codes constructed by the above steps are single \((\pm 1, \pm 2, \ldots, \pm s)\) error correcting codes.

(Proof) Let \( e \) and \( e' \) be error values of single \((\pm 1, \pm 2, \ldots, \pm s)\) error. Let \( s \) be a syndrome where error value \( e \) occurs in position corresponding to \( h_{i,j} \). Let \( s' \) be another syndrome where \( e' \) occurs in position corresponding to \( h_{r, j} \). Then the syndromes are calculated as \( s = eh_{i,j}, s' = e'h_{r,j} \).

1) For \( i \neq i' \)

We assume \( i < i' \). It is clear that \( i \)-th element of the syndrome \( s \) denoted \( s_i \) is multiplication of an element of \( L \) denoted \( l \), and error value \( e \). Then, the syndrome element \( s_i \) is not included in \( O \) because of definition of the set. That is,

\[
s_i = el \notin O.
\]

Also, \( i \)-th element of syndrome \( s' \) denoted \( s'_i \) is multiplication of an element of \( O \) denoted \( o \) and error value \( e' \). Then the syndrome element \( s'_i \) is an element of \( O \). That is,

\[
s'_i = e'o \in O.
\]

Since \( s_i \neq s'_i \), \( s \neq s' \) is satisfied.

2) For \( i = i' \)

2.1) For \( e = e' \)
Because any values of \((\pm 1, \pm 2, \ldots, \pm s)\) errors are units, there exist inverse elements. If \(s = s'\) is supposed, following relation is obtained by multiplying inverse element to the equation:

\[
h_{i,j} = e^{-1}s = e^{-1}s' = h_{i',j}.
\]

Equivalence of column vectors contradict code construction. Therefore \(s \neq s'\).

2-2) For \(e \neq e'\)

It is clear that \(s_i\) is multiplication of an element of \(L\) denoted \(l\) and error values \(e\). Also, \(s_i'\) is multiplication of an element of \(L\) denoted \(l'\) and error values \(e'\). Because of definition of the set \(L\), we obtain

\[
s_i = el \neq e'l' = s_i'.
\]

Therefore, \(s \neq s'\).

It follows from 1) and 2) that \(s \neq s'\) is satisfied when \((i, j, e) \neq (i', j', e')\). From theorem 1, the codes defined by \(H\) are single \((\pm 1, \pm 2, \ldots, \pm s)\) error correcting codes. (Q.E.D.)

The number of column vectors in respective submatrices \(H_i\) is

\[
|O|^{l-1}|L| |U|^{-i}.
\]

Therefore, code symbol length \(N\) is calculated by summation of (2), that is,

\[
N \leq \sum_{i=1}^{s} |O|^{l-1}|L| |U|^{-i} = |L| |U|^r - |O|^r
\]

\[
|U| - |O|,
\]

(3)

There are some remarks regarding the code construction. In step 2, if \(m < 2s + 1\), then there isn’t \(L\) that satisfies (1). But if \(m \geq 2s + 1\) is satisfied, then \(L\) cannot be null set because at least the element 1 can be contained in the set.

The algorithm to determine \(L\) doesn’t guarantee to maximize code length, but the code with check length at least \(N \geq (|U|^r - 1)/(|U| - 1)\) can be constructed. Also, if \(L\) contains 1, then check part of the code is clearly determined. This is easily realized by choosing first \(x\) as 1 in the algorithm.

Here we show some examples. All examples below choose minimum element of \(A\) as \(x\) when we make the set \(L\).

**Example 1:** We construct single \((\pm 1, \pm 2)\) error correcting code with 2 check symbols for 4 bit/byte systems. Let \(m\) be multiple of primes larger than 1. In this case, \(16 = 2^4\) cannot be used, but 3-5 = 15 is used for \(m\). Using algorithm, \(L = \{1, 3, 4, 5\}\) is obtained as follows:

1) Let \(A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}\), \(E = \{\pm 1, \pm 2\}\) and \(L = \phi\).
2) Let \(x = 1\), then \(xE = \{1, 2, 13, 14\}\).
3) Remove elements of \(xE\) from \(A\), then \(A = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}\).
4) Add \(x\) to \(L\), then \(L = \{1\}\).
5) Let \(x = 3\), then \(xE = \{3, 6, 9, 12\}\).
6) Remove \(xE\) from \(A\), then \(A = \{4, 5, 7, 8, 10, 11\}\).
7) Add \(x\) to \(L\), then \(L = \{1, 3\}\).
8) Let \(x = 4\), then \(xE = \{4, 7, 8, 11\}\).
9) Remove \(xE\) from \(A\), then \(A = \{5, 10\}\).
10) Add \(x\) to \(L\), then \(L = \{1, 3, 4\}\).
11) There remains no \(x\) which can be added to \(L\). Finally \(A\) becomes a null set, and then the Algorithm is finished.

The set \(O\) consists of 5 and its multiples, that is \(\{0, 5, 10\}\). Finally, code with length \(N = 54\) symbols is obtained as follows:

\[
H = \begin{bmatrix}
1 & \cdots & 1 & \cdots & 3 & \cdots & 4 & \cdots & 4 \\
0 & \cdots & 14 & \cdots & 0 & \cdots & 14 \\
0 & 0 & 5 & 5 & 10 & 10 & 10 \\
1 & 3 & 4 & 3 & 1 & 3 & 4
\end{bmatrix}
\]

**Example 2:** We construct single \((\pm 1)\) error correcting code with 2 check symbols for 3 bit/byte systems. Let \(m\) be multiple of primes larger than 1. In this case, \(8 = 2^3\) is used for \(m\). Using algorithm, \(L = \{1, 2, 3\}\) is obtained. The set \(O\) consists of 4 and its multiples, that is \(\{0, 4\}\). Finally, code with length \(N = 30\) symbols is obtained as follows:

\[
H = \begin{bmatrix}
1 & \cdots & 1 & \cdots & 2 & \cdots & 3 & \cdots & 3 \\
0 & \cdots & 7 & \cdots & 7 & \cdots & 7 & \cdots & 7 \\
0 & 0 & 4 & 4 & 4 & 4 & 4 & 4 & 4
\end{bmatrix}
\]

This code is equivalent to \((\pm 1)\) error correcting code in [9]. Actually, any codes in [9] can be constructed by proposed method when the minimum element of \(A\) is chosen as \(x\) in the algorithm to determine \(L\). Therefore, proposed codes are said to be generalization of the code.

3.2 Additional Error Detecting Function

Let \(p\) be minimum prime factor of \(m\). If \(p = s + 1\) then \((\pm(s + 1))\) error detecting function can be added. In this case, we initially determine the set \(O\) and use modified algorithm to determine \(L\). Let the sets obtained by the modified process be \(O'\) and \(L'\).

**[Algorithm to determine \(L'\)]**

Calculation is executed over \(Z_m\).

1) Let \(A = \{1, 2, \ldots, m - 1\}\), \(E = \{\pm 1, \pm 2, \ldots, \pm s\}\), \(L' = \phi\) and let \(O'\) be a set containing all zero divisors of \(Z_m\).
2) If \(A\) is a null set, then finish this algorithm.
3) Let \(x\) be an arbitrary element in \(A\).
4) If \(x\) is included in \(O'\), then eliminate \(x\) from \(A\) and back to 2).
5) If \(xE \notin A\), then eliminate \(x\) from \(A\) and back to 2).
Let $E$ be the combinations of $e$ memories have levels of eration.

3) Initially, $s_0 \leftarrow 1$.
4) If $s_i \in O$, then $i \leftarrow i + 1$.
5) Repeat 4) until $s_i \not\in O$ is satisfied. If $s_i \in O$ is satisfied for all $i \leq r$, then finish this algorithm and $(\pm(s+1))$ error is detected.
6) Search for $l \in L$ and $e \in E$ which satisfy $el = s_i$ over $Z_m$.
7) Calculate $s^* = e^{-1}s$.
8) Consider error $e$ is added to position where the corresponding column vector equals $s^*$. Calculate corrected word by $y' = y - e$ and finish this algorithm.

Considering (1), element $l$ and $e$ is uniquely determined. Also, because $|L||E| < m$ should be satisfied from (1), all combinations of $e$ and $l$ is at most $m$. Recent MLC Flash memories have levels of $M = 2^n$ where $n$ is at most 4, and $m \leq M$. Consequently, exhaustive search makes no problem in process 6) because it does not require so much calculation.

The decoding of code with error detecting function can be achieved similarly but replacing $L$ and $O$ with $l$ and $O$.

4. Evaluation

4.1 Information Length

Information symbol lengths to respective check symbol lengths for code with distinct function are listed in Tables 1-3. Table 1 is for 3 bit/byte, Table 2 is for 4 bit/byte and Table 3 is for 5 bit/byte. When making the set $L$, all codes in the tables, we choose the minimum element of $A$ as $x$. Each notation in these tables are as follows:

- C1: Conventional single $(\pm 1)$ error correcting codes using Gray code [8]
- C2: Conventional single $(\pm 1)$ error correcting integer codes [9]
- C3: Conventional nonbinary single error correcting Hamming codes [7]
- P0: Proposed single $(\pm 1)$ error correcting codes

| Table 1 | Information symbol lengths to respective check lengths for 3 bit/byte. |
|---------|--------------------------|
| $2$     | $3$     | $4$     | $5$     |
| C1      | 26      | 247     | 2036    | 16369   |
| C2      | 28      | 249     | 2036    | 16363   |
| C3      | 7       | 70      | 581     | 4676    |
| P0      | 28      | 249     | 2036    | 16363   |

| Table 2 | Information symbol lengths to respective check lengths for 4 bit/byte. |
|---------|--------------------------|
| $2$     | $3$     | $4$     | $5$     |
| C1      | 120     | 2036    | 32752   | 524268  |
| C2      | 124     | 2041    | 32756   | 524267  |
| C3      | 15      | 270     | 4365    | 69909   |
| P0      | 124     | 2041    | 32756   | 524267  |

| Table 3 | Information symbol lengths to respective check lengths for 5 bit/byte. |
|---------|--------------------------|
| $2$     | $3$     | $4$     | $5$     |
| C1      | 502     | 16369   | 524268  | 16777191|
| C2      | 508     | 16377   | 524276  | 16777195|
| C3      | 31      | 1054    | 33821   | 1082396 |
| P0      | 308     | 16377   | 524276  | 16777195|
| P1      | 382     | 14333   | 491516  | 16253923|
| P2(order 21) | 180 | 5955 | 184700 | 57258252 |
| P3(order 27) | 142 | 4209 | 116636 | 3175519  |
Table 4  Information symbol length comparison with codes in [14] for 4 bit/byte and \((\pm 1, \pm 2)\) error correction.

| Check Symbol Length | Conventional Code [14] | Proposed Code | Bound [14] |
|---------------------|------------------------|---------------|------------|
| 2                   | 23                     | 52            | 63         |
| 3                   | 438                    | 834           | 1023       |
| 4                   | 7221                   | 12632         | 16385      |
| 5                   | 116276                 | 189778        | 262143     |

Table 5  Information symbol length comparison with codes in [14] for 5 bit/byte and \((\pm 1, \pm 2)\) error correction.

| Check Symbol Length | Conventional Code [14] | Proposed Code | Bound [14] |
|---------------------|------------------------|---------------|------------|
| 2                   | 43                     | 190           | 255        |
| 3                   | 1530                   | 5935          | 8191       |
| 4                   | 49721                  | 184700        | 262143     |
| 5                   | 1596216                | 5725825       | 8386007    |

- P1: Proposed single \((\pm 1)\) error correcting \((\pm 2)\) error detecting codes
- P2: Proposed single \((\pm 2)\) error correcting codes
- P3: Proposed single \((\pm 2)\) error correcting \((\pm 3)\) error detecting codes

As shown, information lengths of proposed codes are larger than those of conventional single symbol error correcting Hamming code except for 3 bit/byte. For example, assuming 4 bit/byte and the error environment in which 2 levels shift occurs, conventional \((\pm 1)\) error correcting codes cannot correct such error. In this case, Hamming code (C3) or proposed \((\pm 1, \pm 2)\) error correcting code (P2) can correct the error. When codes with 2 check symbols are assumed, Table 2 shows that 15 information symbols can be controlled for Hamming code while 52 information symbols for proposed code. This reveals proposed codes are efficient for 4 bit/byte and 2 check symbols, and similar discussion also holds for 5 bit/byte and other check symbol lengths.

Also, Comparison of information lengths with another \(M\)-ary asymmetric symbol error correcting codes and theoretical bound [14] are listed in Tables 4 and 5. For example, considering 4 bit/byte and the error environment in which 2 levels shift occurs, conventional asymmetric error correcting code can control 23 information symbols while proposed code can do 52 symbols for 2 check symbols. Thus proposed codes are better for neighborhood error correction, although codes in [14] can deal with more generalized class of asymmetric error.

4.2 Bit Error Rate and Code Length

Uncorrectable bit error rate (UBER) toward some proposed \((\pm 2)\) error correcting codes for 4 bit/byte are listed in Table 6. In the table, we assume 5% of symbol errors are 2 levels shift and the rest is 1 level shift to calculate symbol error rate from raw bit error rate (RBER).

Considering the case of the 3x-nm technology product in [4] in which 2 levels shift is observed, RBER of the memory is less than \(10^{-4}\) through its life time of 3000 P/E cycles.

Table 6  UBER when \((\pm 2)\) error correcting codes for 4 bit/byte are applied.

| Error  | Detection | Miss Correction | Miss Detection |
|--------|-----------|-----------------|---------------|
| \((\pm 2)\) | 4.8%      | 95.2%           | 0%            |
| \((\pm 3)\) | 0%        | 100%            | 0%            |

Although MLC Flash products require UBER from \(10^{-13}\) to \(10^{-16}\) [16], proposed codes cannot achieve such UBER at any code length and RBER. Therefore, we should apply additional error correcting codes such as concatenated coding to gain practical UBER.

4.3 Decoding Error Probability

Decoding probability of some proposed codes calculated by examining all error pattern are listed in Tables 7-10. When the error value is an unit, miss correction certainly occur.
From first row in Tables 7 and 9, if a code doesn’t have \((\pm (s + 1))\) error detecting function, it is almost impossible to detect such errors.

5. Conclusion

In this paper, we have proposed neighborhood level error control code for MLC flash memory. The codes can correct single \((\pm 1, \pm 2, \ldots, \pm s)\) error where range of error correction \(s\) is selectable. In addition, we can add error detecting function toward \((\pm (s + 1))\) error if it is needed. The codes are generalization of conventional \((\pm 1)\) error correcting integer codes because they are equivalent on a certain parameter. Evaluation results show that the code length is larger than single symbol error correcting Hamming codes and conventional \(M\)-ary asymmetric error correcting codes. This shows that the proposed codes are more efficient when value range are limited to neighborhood.

Future study remains in devising coding method to achieve practical bit error rate, and extending the proposed codes to multiple error correction.

References

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, “Introduction to flash memory,” Proc. IEEE, vol.91, no.4, pp.489–502, April 2003.
[2] F. Masuoka, M. Monodomi, Y. Iwata, and R. Shirota, “New ultra high density EPROM and flash EEPROM with NAND structure cell,” Electron Devices Meeting, 1987 International, vol.33, pp.552–555, 1987.
[3] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Scharer, F. Trivedi, E. Goodness, and L.R. Nevill, “Bit error rate in NAND flash memories,” IEEE International Reliability Physics Symposium, pp.9–19, 2008.
[4] Y. Cai, E.F. Haratsch, O. Mutlu, and K. Mai, “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” Design, Automation & Test in Europe Conference & Exhibition, pp.521–526, March 2012.
[5] Y. Maeda and H. Kaneko, “Error control coding for multilevel cell flash memories using nonbinary low-density parity-check codes,” IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp.367–375, 2009.
[6] M. Grossi, M. Lanzoni, and B. Ricco, “Program schemes for multilevel flash memories,” Proc. IEEE, vol.91, no.4, pp.594–601, April 2003.
[7] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Wiley-Interscience, 2006.
[8] K. Sato and M. Kitakami, “A class of 1 level error correcting code for multilevel cell flash memories,” IEICE Technical Report, DC, no.53, pp.9–14, Dec. 2010.
[9] H. Kostadinov, H. Morita, and N. Manev, “On \((\pm 1)\) error correctable integer codes,” IEICE Trans. Fundamentals, vol.E93-A, no.12, pp.2758–2761, Dec. 2010.
[10] S. Tanakamaru, C. Hung, A. Esumi, M. Ito, K. Li, and K. Takeuchi, “95%-lower-BER 43%-lower-power intelligent solid-state drive (SSD) with asymmetric coding and stripe pattern elimination algorithm,” IEEE International Solid-State Circuits Conference, pp.204–206, 2011.
[11] Y. Cai, E.F. Haratsch, O. Mutlu, and K. Mai, “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” Design, Automation & Test in Europe Conference & Exhibition, pp.521–526, March 2012.
[12] C. Yang, Y. Emire, C. Chakrabarti, and T. Mudge, “Flexible product code-based ECC schemes for MLC NAND flash memories,” IEEE Workshop on Signal Processing Systems, pp.255–260, Oct. 2011.
[13] C.M. Compagnoni, A.S. Spinelli, R. Gusmeroli, A.L. Lacaia, B. Beltrami, A. Ghetti, and A. Visconti, “First evidence for Injection Statistic Accuracy Limitations in NAND Flash Constant-current Fowler-Nordheim Programming,” IEEE International Electron Devices Meeting, pp.165–168, Dec. 2007.
[14] H. Kaneko and E. Fujiwara, “A class of \(M\)-ary asymmetric integer error correcting codes for data entry devices,” IEEE Trans. Comput., vol.53, pp.159–167, Feb. 2004.
[15] A.J. Han Vinck and H. Morita, “Codes over the ring of integer modulo \(m\),” IEICE Trans. Fundamentals, vol.E81-A, no.10, pp.2013–2018, Oct. 1998.
[16] J. Gray and C.V. Ingen, “Empirical measurements of disk failure rates and error rates,” Microsoft Research Technical Report MSR-TR-2005-166, Dec. 2005.