3

Error Correction Codes and Signal Processing in Flash Memory

Xueqiang Wang[1], Guiqiang Dong[2], Liyang Pan[1] and Runde Zhou[1]
[1]*Tsinghua University,*
[2]*Rensselaer Polytechnic Institute,*
[1]*China*
[2]*USA*

1. Introduction

This chapter is to introduce NAND flash channel model, error correction codes (ECC) and signal processing techniques in flash memory.

There are several kinds of noise sources in flash memory, such as random-telegraph noise, retention process, inter-cell interference, background pattern noise, and read/program disturb, etc. Such noise sources reduce the storage reliability of flash memory significantly. The continuous bit cost reduction of flash memory devices mainly relies on aggressive technology scaling and multi-level per cell technique. These techniques, however, further deteriorate the storage reliability of flash memory. The typical storage reliability requirement is that non-recoverable bit error rate (BER) must be below 10^{-15}. Such stringent BER requirement makes ECC techniques mandatory to guarantee storage reliability. There are specific requirements on ECC scheme in NOR and NAND flash memory. Since NOR flash is usually used as execute in place (XIP) memory where CPU fetches instructions directly from, the primary concern of ECC application in NOR flash is the decoding latency of ECC decoder, while code rate and error-correcting capability is more concerned in NAND flash. As a result, different ECC techniques are required in different types of flash memory.

In this chapter, NAND flash channel is introduced first, and then application of ECC is discussed. Signal processing techniques for cancelling cell-to-cell interference in NAND flash are finally presented.

2. NAND flash channel model

There are many noise sources existing in NAND flash, such as cell-to-cell interference, random-telegraph noise, background-pattern noise, read/program disturb, charge leakage and trapping generation, etc. It would be of great help to have a NAND flash channel model that emulates the process of operations on flash as well as influence of various program/erase (PE) cycling and retention period.

2.1 NAND flash memory structure

NAND flash memory cells are organized in an array->block->page hierarchy, as illustrated in Fig. 1., where one NAND flash memory array is partitioned into many blocks, and each

block contains a certain number of pages. Within one block, each memory cell string typically contains 16 to 64 memory cells.

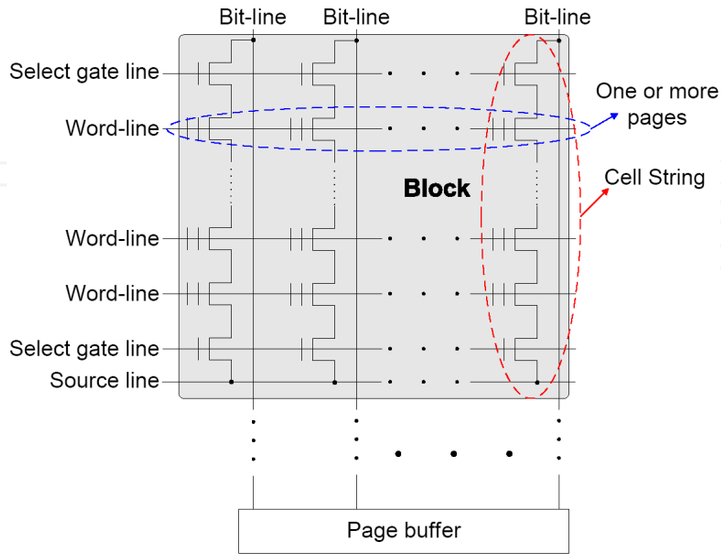


Fig. 1. Illustration of NAND flash memory structure.

All the memory cells within the same block must be erased at the same time and data are programmed and fetched in the unit of page, where the page size ranges from 512-byte to 8K-byte user data in current design practice. All the memory cell blocks share the bit-lines and an on-chip page buffer that holds the data being programmed or fetched. Modern NAND flash memories use either even/odd bit-line structure, or all-bit-line structure. In even/odd bit-line structure, even and odd bit-lines are interleaved along each word-line and are alternatively accessed. Hence, each pair of even and odd bit-lines can share peripheral circuits such as sense amplifier and buffer, leading to less silicon cost of peripheral circuits. In all-bit-line structure, all the bit-lines are accessed at the same time, which aims to trade peripheral circuits silicon cost for better immunity to cell-to-cell interference. Moreover, relatively simple voltage sensing scheme can be used in even/odd bit-line structure, while current sensing scheme must be used in all-bit-line structure. For MLC NAND flash memory, all the bits stored in one cell belong to different pages, which can be either simultaneously programmed at the same time, referred to as full-sequence programming, or sequentially programmed at different time, referred to as multi-page programming.

2.2 NAND flash memory erase and program operation model

Before a flash memory cell is programmed, it must be erased, i.e., remove all the charges from the floating gate to set its threshold voltage to the lowest voltage window. It is well known that the threshold voltage of erased memory cells tends to have a wide Gaussian-like distribution. Hence, we can approximately model the threshold voltage distribution of erased state as

$$p_e(x) = \frac{1}{\sigma_e \sqrt{2\pi}} e^{-\frac{(x-\mu_e)^2}{2\sigma_e^2}} \tag{1}$$

where μ_e and δ_e are the mean and standard deviation of the erased state.
Regarding memory programming, a tight threshold voltage control is typically realized by using incremental step pulse program (ISPP), i.e., memory cells on the same word-line are recursively programmed using a program-and-verify approach with a stair case program word-line voltage V_{pp}, as shown in Fig.2.

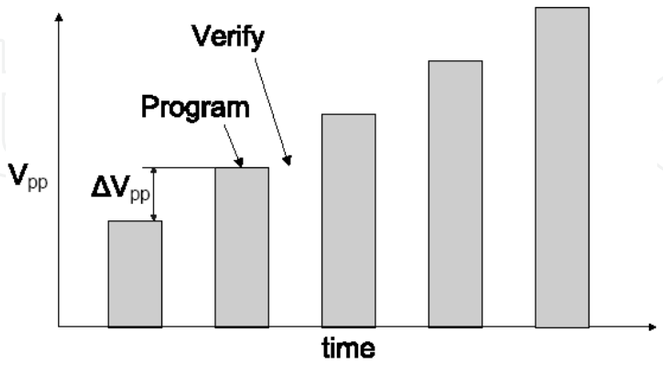


Fig. 2. Control-gate voltage pulses in program-and-verify operations.

Under such a program-and-verify strategy, each programmed state (except the erased state) associates with a verify voltage that is used in the verify operations and sets the target position of each programmed state threshold voltage window. Denote the verify voltage of the target programmed state as V_p, and program step voltage as ΔV_{pp}. The threshold voltage of the programmed state tends to have a uniform distribution over $[V_p, V_p + \Delta V_{pp}]$. Denote V_p and $V_p + \Delta V_{pp}$ for the *k*-th programmed state as V_l^k and V_r^k. We can model the ideal threshold voltage distribution of the *k*-th programmed state as:

$$p_p^{(k)}(x) = \begin{cases} \frac{1}{\Delta V_{pp}}, & \text{if } V_l^{(k)} \le x \le V_r^{(k)} \\ 0, & \text{else} \end{cases} \tag{2}$$

The above *ideal* memory cell threshold voltage distribution can be (significantly) distorted in practice, mainly due to PE cycling effect and cell-to-cell interference, which will be discussed in the following.

2.3 Effects of program/erase cycling

Flash memory PE cycling causes damage to the tunnel oxide of floating gate transistors in the form of charge trapping in the oxide and interface states, which directly results in threshold voltage shift and fluctuation and hence gradually degrades memory device noise margin. Major distortion sources include

1. Electrons capture and emission events at charge trap sites near the interface developed over PE cycling directly result in memory cell threshold voltage fluctuation, which is referred to as random telegraph noise (RTN);
2. Interface trap recovery and electron detrapping gradually reduce memory cell threshold voltage, leading to the data retention limitation.

RTN causes random fluctuation of memory cell threshold voltage, where the fluctuation magnitude is subject to exponential decay. Hence, we can model the probability density function $p_r(x)$ of RTN-induced threshold voltage fluctuation as a symmetric exponential function:

$$p_r(x) = \frac{1}{2\lambda_r} e^{-\frac{|x|}{\lambda_r}} \tag{3}$$

Let N denote the PE cycling number, λ_r scales with N in an approximate power-law fashion, i.e., λ_r is approximately proportional to N_a.

Threshold voltage reduction due to interface trap recovery and electron detrapping can be approximately modeled as a Gaussian distribution $\mathcal{N}(\mu_d, \delta_d^2)$. Both μ_d and δ_d^2 scale with N in an approximate power-law fashion, and scale with the retention time t in a logarithmic fashion. Moreover, the significance of threshold voltage reduction induced by interface trap recovery and electron detrapping is also proportional to the initial threshold voltage magnitude, i.e., the higher the initial threshold voltage is, the faster the interface trap recovery and electron detrapping occur and hence the larger threshold voltage reduction will be.

2.4 Cell-to-cell interference

In NAND flash memory, the threshold voltage shift of one floating gate transistor can influence the threshold voltage of its adjacent floating gate transistors through parasitic capacitance-coupling effect, i.e. one float-gate voltage is coupled by the floating gate changes of the adjacent cells via parasitic capacitors. This is referred to as cell-to-cell interference. As technology scales down, this has been well recognized as one of major noise sources in NAND flash memory. Threshold voltage shift of a victim cell caused by cell-to-cell interference can be estimated as

$$F = \sum_k (\Delta V_t^{(k)} \cdot \gamma^{(k)}) \tag{4}$$

where $\Delta V_t^{(k)}$ represents the threshold voltage shift of one interfering cell which is programmed after the victim cell, and the coupling ratio is defined as

$$\gamma^{(k)} = \frac{C^{(k)}}{C_{total}} \tag{5}$$

where $C^{(k)}$ is the parasitic capacitance between the interfering cell and the victim cell, and C_{total} is the total capacitance of the victim cell. Cell-to-cell interference significance is affected by NAND flash memory bit-line structure. In current design practice, there are two different bit-line structures, including conventional even/odd bit-line structure and emerging all-bit-line structure. In even/odd bit-line structure, memory cells on one word-line are alternatively connected to even and odd bit-lines and even cells are programmed ahead of odd cells in the same wordline. Therefore, an even cell is mainly interfered by five neighboring cells and an odd cell is interfered by only three neighboring cells, as shown in Fig. 3. Therefore, even cells and odd cells experience largely different amount of cell-to-cell interference. Cells in all-bit-line structure suffers less cell-to-cell inference than even cells in odd/even structure, and the all-bit-line structure can effectively support high-speed current sensing to improve the memory read and verify speed. Therefore, throughout the remainder of this paper, we mainly consider NAND flash memory with the all-bit-line structure. Finally, we note that the design methods presented in this work are also applicable when odd/even structure is being used.

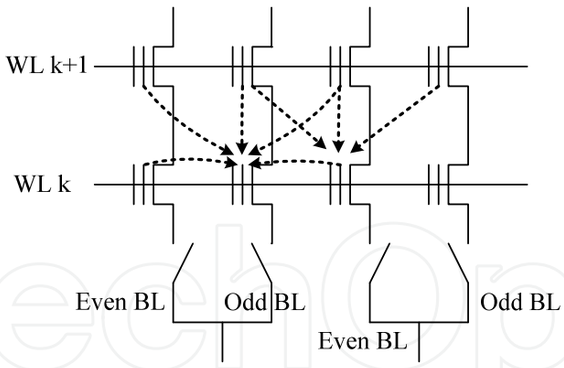


Fig. 3. Illustration of cell-to-cell interference in even/odd structure: even cells are interfered by two direct neighboring cells on the same wordline and three neighboring cells on the next wordline, while odd cells are interfered by three neighboring cells on the next wordline.

2.5 NAND flash memory channel model

Based on the above discussions, we can approximately model NAND flash memory device characteristics as shown in Fig. 4, using which we can simulate memory cell threshold voltage distribution and hence obtain memory cell raw storage reliability.

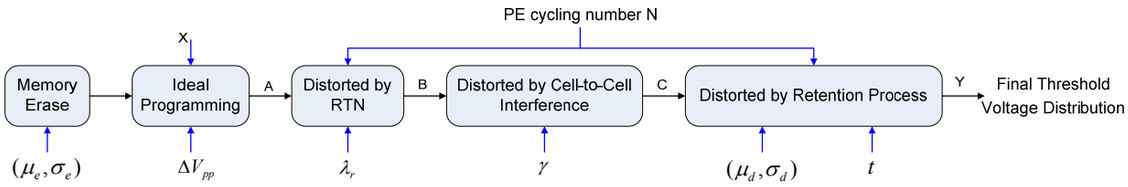


Fig. 4. Illustration of the approximate NAND flash memory device model to incorporate major threshold voltage distortion sources.

Based upon the model of erase state and ideal programming, we can obtain the threshold voltage distribution function $p_p(x)$ right after ideal programming operation. Recall that $p_{pr}(x)$ denotes the RTN distribution function, and let $p_{ar}(x)$ denote the threshold voltage distribution after incorporating RTN, which is obtained by convoluting $p_p(x)$ and $p_r(x)$, i.e.,

$$p_{ar}(x) = p_p(x) \bigotimes p_r(x) \tag{6}$$

The cell-to-cell interference is further incorporated based on the model of cell-to-cell interference. To capture inevitable process variability, we set both the vertical coupling ratio and diagonal coupling ratio as random variables with tailed truncated Gaussian distribution:

$$p_c(x) = \begin{cases} \frac{c_c}{\sigma_c\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}, & \text{if } |x-\mu_c| \le w_c \\ 0, & \text{else} \end{cases} \tag{7}$$

where μ_c and δ_c are the mean and standard deviation, and C_C is chosen to ensure the integration of this tail truncated Gaussian distribution equals to 1. In all the simulations in this section, we set $w_c = 0.1\mu_c$ and $\delta_c = 0.4\mu_c$.

Let $p_{ac}(x)$ denote the threshold voltage distribution after incorporating cell-to-cell interference. Denote the retention noise distribution as $p_t(x)$. The final threshold voltage distribution $p_f(x)$ is obtained as

$$p_f(x) = p_{ac}(x) \bigotimes p_t(x) \tag{8}$$

The above presented approximate mathematical channel model for simulating NAND flash memory cell threshold voltage is further demonstrated using the following example.

Example 1: Let us consider 2bits/cell NAND flash memory. Normalized μ_e and σ_e of the erased state are set as 1.4 and 0.35, respectively. For the three programmed states, the normalized program step voltage ΔV_{pp} is 0.2, and the normalized verify voltages V_p are 2.6, 3.2 and 3.93, respectively. For the RTN distribution function, we set the parameter $\lambda_r = K_r N^{0.5}$, where K_r equals to 0.00025. Regarding to cell-to-cell interference, we set the ratio between the means of γ_y and γ_{xy} as 0.08 and 0.0048, respectively. For the function $\mathcal{N}(\mu_d, \sigma_d^2)$ to capture trap recovery and electron detrapping during retention, we set that μ_d scales with $N^{0.5}$ and σ_d^2 scales with $N^{0.6}$, and both scale with $\ln(1 + t/t_0)$, where t denotes the memory retention time and t_0 is an initial time and can be set as 1 hour. In addition, as pointed out earlier, both μ_d and σ_d also depend on the initial threshold voltage. Hence we set that both approximately scale $K_s(x - x_0)$, where x is the initial threshold voltage, and x_0 and K_s are constants. Therefore, we have

$$\begin{cases} \mu_d = K_s(x - x_0) K_d N^{0.5} \ln(1 + t/t_0) \\ \sigma_d^2 = K_s(x - x_0) K_m N^{0.6} \ln(1 + t/t_0) \end{cases} \tag{9}$$

where we set $K_s = 0.38$, $x_0 = 1.4$, $K_d = 4 \times 10^{-4}$, and$K_m = 4 \times 10^{-6}$. Accordingly, we carry out Monte Carlo simulations to obtain

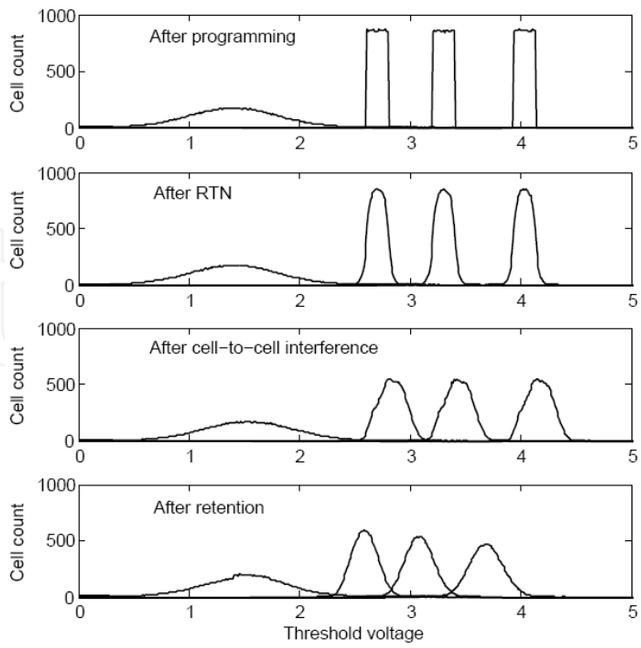


Fig. 5. Simulated results to show the effects of RTN, cell-to-cell interference, and retention on memory cell threshold voltage distribution after 10K PE cycling and 10-year retention.

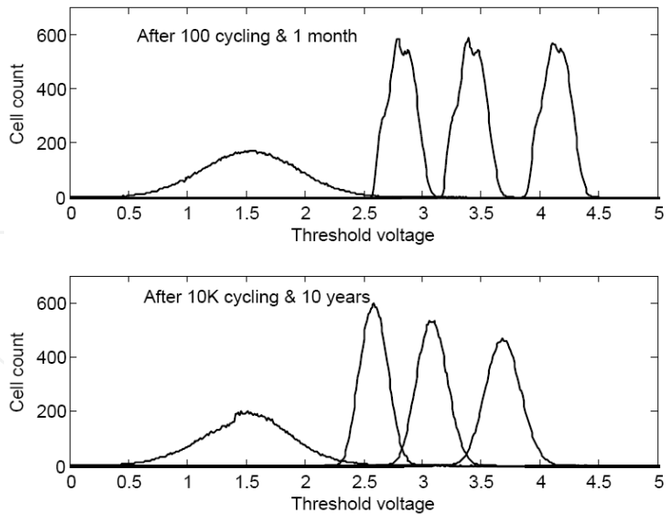


Fig. 6. Simulated threshold voltage distribution after 100 PE cycling and 1-month retention and after 10K PE cycling and 10-year retention, which clearly shows the dynamics inherent in NAND flash memory characteristics.

Fig. 5 shows the cell threshold voltage distribution at different stages under 10K PE cycling and with 10-year storage period. The final threshold voltage distributions after 100 PE cycling and 1 month storage and after 10K PE cycling and 10 years storage are shown in Fig. 6. Fig. 7 presents the evolution of simulated raw BER with program/erase cycling.

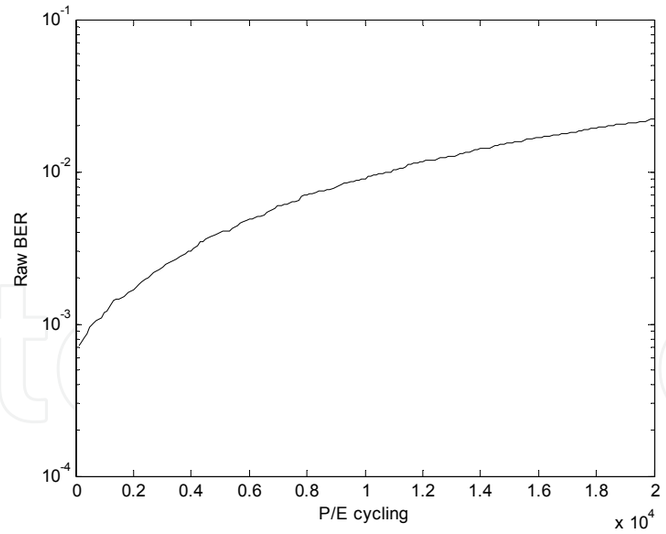


Fig. 7. The evolution of raw BER with program/erase cycling under 10-year storage period.

3. Basics of error correction codes

In the past decades, error correction codes (ECC) have been widely adapted in various communication systems, magnetic recording, compact discs and so on. The basic scheme of ECC theory is to add some redundancy for protection. Error correction codes are usually

divided into two categories: block codes and convolution codes. Hamming codes, Bose-Chaudur-Hocquenghem(BCH) codes, Reed-Solomon(RS) codes, and Low-density parity-check (LDPC) codes are most notable block codes and have been widely used in communication, optical, and other systems.

The encoding/decoding scheme of a block code in a memory is shown in Fig. 8. When any *k*-bit information data is written to flash memory, an encoder circuit generates the parity bits, adds these parity bits to the *k*-bit information data and creates a *n*-bit codeword. Then the whole codeword is written in and stored on a page of the memory array. During the reading operation, a decoder circuit searches errors in a codeword, and corrects the erroneous bits within its error capability, thereby recovering the codeword.

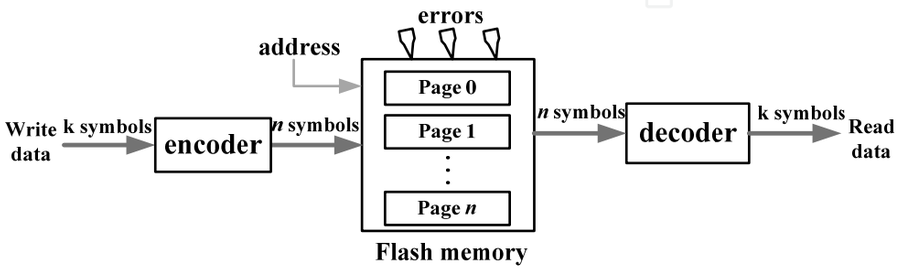


Fig. 8. ECC encoding and decoding system in a flash memory

Current NOR flash memory products use Hamming code with only 1-bit error correction. However, as raw BER increases, 2-bit error corretion BCH code becomes a desired ECC. Besides, in current 2b/cell NAND flash memory. BCH codes are widely employed to achieve required storage reliability. As raw BER soars in future 3b/cell NAND flash memory, BCH codes are not sufficient anymore, and LDPC codes become more and more necessary for future NAND flash memory products.

3.1 Basics of BCH codes

BCH codes were invented through independent researches by Hocquenghen in 1959 and by Bose and Ray-Chauduri in 1960. Flash memory uses binary primitive BCH code which is constructed over the Galois fields GF(2^m). Galois field is a finite field in the coding theory and was first discovered by Evariste Galois. In the following, we will recall some algebraic notions of GF(2^m).

Definition 3.1 Let α be an element of GF(2^m), α is called primitive element if the smallest natural number *n* that satisfies $\alpha^n=1$ equals 2^m -1, that is, $n=2^m$ -1.

Theorem 3.1 Every none null element of GF(2^m) can be expressed as power of primitive element α, that is, the multiplicative group GF(2^m) is cyclic.

Definition 3.2 GF(2^m)[*x*] is indicated as the set of polynomials of any degree with coefficients in GF(2^m). An irreducible polynomial *p(x)* in GF(2^m)[*x*] of degree *m* is called primitive if the smallest natural number *n*, such that x^n–1 is a multiple of *p(x)*, is $n=2^m$–1.

In fact, if $p(x)=x^m+a_{m-1}x^{m-1}+\ldots+a_1x+a_0$ is a primitive polynomial in GF(*p*)[*x*] and α is one of its roots, then we have

$$a^m = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1} \tag{10}$$

Equation (10) indicates that each power of α with degree larger than m can be converted to a polynomial with degree m-1 at most. As an example, some elements in the field GF(2^4), their binary representation, and according poly representation forms are shown in Table 1.

Element	Binary representation	Polynomial representation
0	0000	0
α^0	1000	1
α^1	0100	α
α^3	0001	α^3
α^4	1100	1+α
α^5	0110	$\alpha+\alpha^2$
α^6	0011	$\alpha^2+\alpha^3$

Table 1. Different representations of elements over GF(2^4)

Based on the Galois fields GF(2^m), the BCH(n, k) code is defined as
Codeword length: $n = 2^m-1$
Information data length: $k \geq 2^m-mt$
In a BCH code, every codeword polynomial $c(x)$ can be expressed as $c(x)=m(x)g(x)$, where $g(x)$ is the generator polynomial and $m(x)$ is the information polynomial.
Definition 3.3 Let α be the primitive element of GF(2^m). Let t be the error correction capability of BCH code. The generator polynomial $g(x)$ of a primitive BCH code is the minimal degree polynomial with root: α, α^2,…α^t. g(x) is given by

$$g(x) = LCM\{\psi_0(x), \psi_1(x), \cdots \psi_{d-2}(x)\} \tag{11}$$

Where Ψ_i is the minimal polynomial of α^i.
Generally, the BCH decoding is much more complicated than the encoding. A typical architecture of BCH code application in a flash memory is presented in Fig. 9.

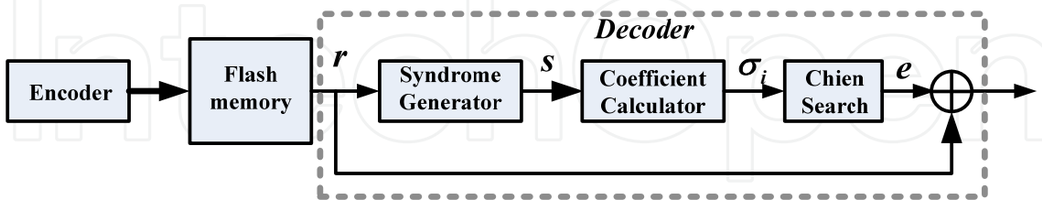


Fig. 9. Architecture of BCH code application in a flash memory

3.1.1 BCH encoding

For a BCH(n,k) code, assuming its generator polynomial is $g(x)$, and the polynomial of the information to be encoded is $m(x)$ with degree of k-1. The encoding process is as follows:
First, the message $m(x)$ is multiplied by x^{n-k}, and then divided by $g(x)$, thereby obtaing a quotient $q(x)$ and a remainder $r(x)$ according to equation (12). The remainder $r(x)$ is the polynomial of the parity information; hence the desired parity bits can be obtained.

$$\frac{m(x)\cdot x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)} \tag{12}$$

As mentioned above, any codeword of BCH code is a multiple of the generator polynomial. Therefore, an encoded codeword $c(x)$ can be expressed as:

$$c(x) = m(x)\cdot x^{n-k} + r(x) \tag{13}$$

3.1.2 BCH decoding

Generally the decoding procedure for binary BCH codes includes three major steps, which is shown in Fig. 9.

- Step 1: Calculating the syndrome ***S***.
- Step 2: Determining the coefficients of the error-location polynomial.
- Step 3: Finding the error location using Chien Search and correcting the errors.

During the period of data storage in flash memory, the repeated program/erase (P/E) cycles may damage the stored information; thereby some errors occur in the read operation. The received codeword can be expressed as $r(x) = c(x) + e(x)$ with the error polynomial representation $e(x) = e_0 + e_1x + \ldots e_{n-1}x^{n-1}$

The first step in the BCH decoding is to calculate $2t$ syndromes with the received r(x). The computation is given by

$$\frac{r(x)}{\psi_i(x)} = Q_i(x) + \frac{S_i(x)}{\psi_i(x)} \quad for \;\; 1 \le i \le 2t \tag{14}$$

Where Ψ_i is the minimal polynomial of element α^i, t is the error numbers in codeword. $S_i(x)$ is called syndrome. Since $\Psi_i(a^i)$=0, the syndrome can also be obtained as $S_i(\alpha^i) = r(\alpha^i)$.

$$S_i(a^i) = r(a^i) \tag{15}$$

From equation (14), it can be seen that the syndrome calculation in the BCH decoding is similar to the encoding process in equation (12). Hence, they both employ the linear feedback shift register (LFSR) circuit structure.

The next step is to compute the coefficients of the error-location polynomial using the obtained syndrome values. The error-location polynomial is defined as

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots \sigma_t x^t \tag{16}$$

Where and σ_i ($1 \le i \le t$) is the required coefficient.

There are two main methods to compute the coefficients, one is Peterson method and the other is Berlekamp-Massey algorithm. In the following sections, we will discuss and employ both methods for error correction in different types of flash memory.

The last step of BCH decoding is Chien search. Chien search is employed to search for the roots of the error locator polynomial. If the roots are found $\sigma(\alpha^i) = 0$ for $0 \le i \le n-1$, then the error location is n-1-i in the codeword. It should be noted that the three modules of a BCH decoder is commonly designed with three pipeline stages, leading to high throughput of the BCH decoding.

3.2 LDPC code

Low-density parity-check (LDPC) codes can provide near-capacity performance. It was invented by Gallager in 1960, but due to the high complexity in its implementation, LDPC codes had been forgotten for decades, until Mackey rediscovered LDPC codes in the 1990s. Since then LDPC codes have attracted much attention.

A LDPC code is given by the null space of a sparse *mxn* 'low-density' parity-check matrix H. Regular LDPC codes have identical column weight and identical row weight. Each row of H represents one parity check. Define each row of H as check node (CN), and each column of H as variable node (VN). A LDPC code can be represented by Tanner graph, which is a bipartite graph and includes two types of nodes: n variable nodes and m check nodes. In Tanner graph, the i-th CN is connected to j-th VN, if $h_{i,j}$=1.

Consider a (6, 3) linear block code with H matrix as

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The corresponding Tanner graph is shown in Fig. 10.

The performance of LDPC code depends heavily on parity-check matrix H. Generally speaking, LDPC code with larger block length, larger column weight and larger girth trends to have better performance. In Tanner graph, a cycle is defined as a sequential of edges that form a closed path. Short cycles degrade the performance of LDPC codes, and length of the shortest cycle in Tanner graph is named as girth.

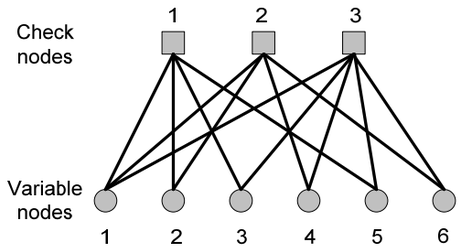


Fig. 10. The Tanner graph for the given (6, 3) linear block code.

There have been lots of methods to construct parity-check matrix. To reduce the hardware complexity of LDPC encoder and decoder, quasi-cyclic (QC) LDPC code was proposed and has widely found its application in wireless communication, satellite communicate and hard-disk drive.

As for LDPC decoding, there are several iterative decoding algorithms for LDPC codes, including bit-flipping (BF) like decoding algorithms and soft-decision message-passing decoding algorithms. Among all BF-like decoding, BF and candidate bit based bit-flipping (CBBF) can work with only hard-decision information. Other BF-like decoding require soft-decision information, which incurs large sensing latency penalty in flash memory devices as discussed later, though they may increase the performance a little bit.

Soft-decision message passing algorithm, such as Sum-product algorithm (SPA), could provide much better performance than BF-like decoding, upon soft-decision information. However, the complexity of SPA decoding is very high. To reduce the decoding complexity, min-sum decoding was proposed, with tolerable performance loss. Readers can refer to "Channel Codes: Classical and Modern" by Willian E. Ryan and Shu Lin.

4. BCH in NOR flash memory

Usually NOR flash is used for code storage and acts as execute in place (XIP) memory where CPU fetches instructions directly from memory. The code storage requires a high-reliable NOR flash memory since any code error will cause a system fault. In addition, NOR flash memory has fast read access with access time up to 70ns. During read operation, an entire page, typical of 256 bits, is read out from memory array, and the ECC decoder is inserted in the critical data path between sense amplifiers and the page latch. The fast read access imposes a stringent requirement on the latency of the ECC decoder (required <10% overhead), and the ECC decoder has to be designed in combinational logic. As a result, decoding latency becomes the primary concern for ECC in NOR Flash memory.

Traditionally, hamming code with single-error-correction (SEC) is applied to NOR flash memory since it has simple decoding algorithm, small circuit area, and short-latency decoding. However, in new-generation 3xnm MLC NOR flash memory, the raw BER will increase up to 10^{-6} while application requires the post-ECC BER be reduced to 10^{-12} below. From Fig. 11, it is clear that hamming code with t=1 is not sufficient anymore, and double-error-correction (DEC) BCH code gains more attraction in future MLC NOR flash memory. However, the primary issue with DEC BCH code applied in NOR flash is the decoding latency. In the following, a fast and adaptive DEC BCH decoding algorithm is proposed and a high-speed BCH(274,256,2) decoder is designed for NOR flash memory.

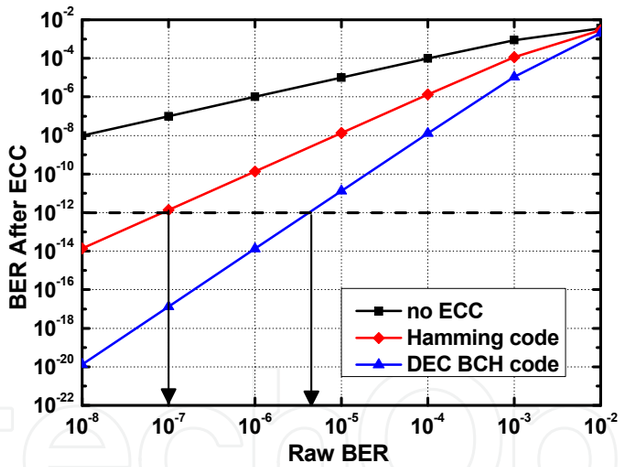


Fig. 11. BER curves of different ECC in NOR flash memory with 256-bit page size

4.1 High-speed DEC BCH decoding algorithm

First we employ equation equation (15) for high-speed syndrome computation. The entire expression of syndromes is

$$(S_1, S_2, \cdots S_{2t}) = r \cdot H^T = (r_0, r_1 \ \ldots \ r_n) \cdot \begin{pmatrix} 1 & 1 & . & 1 \\ (\alpha) & (\alpha^2) & . & (\alpha^{2t}) \\ . & . & . & . \\ (\alpha)^{n-1} & (\alpha^2)^{n-1} & . & (\alpha^{2t})^{n-1} \end{pmatrix} \tag{17}$$

Here ***r*** is the received codeword and ***H*** is defined as the parity matrix

Each element of GF(2^m) α^i can be represented by a m-tuples binary vector, hence each element in the vector can be obtained using mod-2 addition operation, and all the syndromes can be obtained with the XOR-tree circuit structure. Furthermore, for binary BCH codes in flash memory, even-indexed syndromes equal the squares of the other one, i.e., $S_{2i}=S_i^2$, therefore, only odd-indexed syndromes (S_1, S_3 …S_{2t-1}) are needed to compute.

Then we propose a fast and adaptive decoding algorithm for error location. A direct solving method based on the Peterson equation is designed to calculate the coefficients of the error-location polynomial. Peterson equation is show as follows

$$\begin{bmatrix} S_{t+1} \\ S_{t+2} \\ . \\ . \\ S_{2t} \end{bmatrix} = \begin{bmatrix} S_1 & S_2 & . & . & . & S_t \\ S_2 & S_3 & . & . & . & S_{t+1} \\ . & . & & & & . \\ . & . & & & & . \\ S_t & S_{t+1} & . & . & . & S_{2t-1} \end{bmatrix} \begin{bmatrix} \sigma_t \\ \sigma_{t-1} \\ . \\ . \\ \sigma_1 \end{bmatrix} \tag{18}$$

For DEC BCH code t=2, with the even-indexed syndrome S_1, S_3, the coefficient σ_1, σ_2 can be obtained by direct solving the above matrix as

$$\sigma_1 = S_1,\ \sigma_2 = S_1^2 + S_3 / S_1 \tag{19}$$

Hence, the error-locator polynomial is given by

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 = 1 + S_1 x + (S_1^2 + \frac{S_3}{S_1})x^2 \tag{20}$$

To eliminate the complicate division operation in above equation, a division-free transform is performed by multiplying both sides by S_1 and the new polynomial is rewritten as (21). Since it always has $S_1 \neq 0$ when any error exists in the codeword, this transform has no influence of error location in Chien search where roots are found in $\sigma(x)$ =0, that is also $\sigma'(x)$ =0.

$$\sigma'(x) = \sigma_0' + \sigma_1' x + \sigma_2' x^2 = S_1 + S_1^2 x + (S_1^3 + S_3)x^2 \tag{21}$$

The final effort to reduce complexity is to transform the multiplications in the coefficients of equation (21) to simple modulo-2 operations. As mentioned above, over the field GF(2^m), each syndrome vector (S[0], S[1], $\cdots$ S[m-1]) has a corresponding polynomial $S(x)$ = S[0] + S[1]x+ $\cdots$ + S[m-1]x^{m-1}. According to the closure axiom over GF(2^m), each component of the coefficient σ_1' and σ_2' is obtained as

$$\begin{aligned} \sigma_1'[i] &= \sum S_1[j] \text{ for } 0 \le i, j \le m-1 \\ \sigma_2'[i] &= S_3[i] + \sum S_1[j] \bullet S_1[k] \text{ for } 0 \le i, j, k \le m-1 \end{aligned} \tag{22}$$

It can be seen that only modulo-2 additions and modulo-2 multiplications are needed to calculate above equation, which can be realized by XOR and AND logic operations, respectively. Hardware implementation of the two coefficients in BCH(274, 256, 2) code is

shown in Fig. 12. It can be seen that coefficient σ_1' is implemented with only six 2-input XOR gates and coefficient σ_2' can be realized by regular XOR-tree circuit structure. As a result, the direct solving method is very effective to simplify the decoding algorithm, thereby reduce the decoding latency significantly.

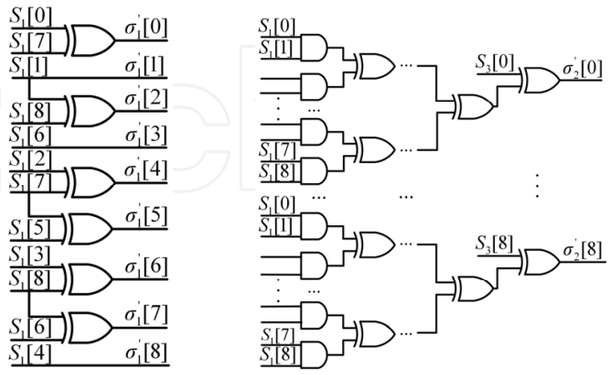


Fig. 12. Implementation of the two coefficient in BCH(274,256,2)

Further, an adaptive decoding architecture is proposed with the reliability feature of flash memory. As mentioned above, flash memory reliability is decreased as memory is used. For the worst case of multi-bit errors in flash memory, 1-bit error is more likely happened in the whole life of flash memory (R. Micheloni, R. Ravasio & A. Marelli, 2006). Therefore, the best-effort is to design a self-adaptive DEC BCH decoding which is able to dynamically perform error correction according to the number of errors. Average decoding latency and power consumption can be reduced.

The first step to perform self-adaptive decoding is to detect the weight-of-error pattern in the codeword, which can be obtained with Massey syndrome matrix.

$$L_j = \begin{bmatrix} S_1 & 1 & 0 & \cdots & 0 \\ S_3 & S_2 & S_1 & \cdots & 0 \\ S_{2j-1} & S_{2j-2} & S_{2j-1} & \cdots & S_j \end{bmatrix} \tag{23}$$

where S_j denotes each syndrome value ($1 \le j \le 2t-1$).

With this syndrome matrix, the weight-of-error pattern can be bounded by the expression of $\det(L_1)$, $\det(L_2)$, …, $\det(L_t)$. For a DEC BCH code in NOR flash memory, the weight-of-error pattern is illustrated as follows

- If there is no error, then det(L1) = 0, det(L2) = 0, that is,

$$S_1 = 0,\ S_1^3 + S_3 = 0 \tag{24}$$

- If there are 1-bit errors, then $\det(L_1) \neq 0$, $\det(L_2) = 0$, that is

$$S_1 \neq 0,\ S_1^3 + S_3 = 0 \tag{25}$$

- If there are 2-bit errors, then $\det(L_1) \neq 0$, $\det(L_2) \neq 0$, that is

$$S_1 \neq 0,\ S_1^3 + S_3 \neq 0 \tag{26}$$

Let define $\boldsymbol{R}= S_1^3 + S_3$. It is obvious that variable R determines the number of errors in the codeword. On the basis of this observation, the Chien search expression partition is presented in the following:

- Chien search expression for SEC

$$\Lambda_{SEC}(\alpha^i) = S_1 + S_1^2(\alpha^i) \qquad \text{for } 2^m - n \le i \le 2^m - 1 \tag{27}$$

- Chien search expression for DEC

$$\Lambda_{DEC}(\alpha^i) = \Lambda_{SEC}(\alpha^i) + R(\alpha^i)^2 \qquad \text{for } 2^m - n \le i \le 2^m - 1 \tag{28}$$

Though above equations are mathematically equivalent to original expression in equation (21), this reformulation make the Chien search for SEC able to be launched once the syndrome S_1 is calculated. Therefore, a short-path implementation is achieved for SEC decoding in a DEC BCH code. In addition, expression (27) is included in expression (28), hence, no extra arithmetic operation is required for the faster SEC decoding within the DEC BCH decoding. Since variable R indicates the number of errors, it is served as the internal selection signal of SEC decoding or DEC decoding. As a result, self-adaptive decoding is achieved with above proposed BCH decoding algorithm reformulation.

To meet the decoding latency requirement, bit-parallel Chien search has to be adopted. Bit-parallel Chien search performs all the substitutions of (28) of n elements in a parallel way, and each substitution has m sub-elements over GF(2^m). Obviously, this will increase the complexity drasmatically. For BCH(274, 256, 2) code, the Chien search module has 2466 expression, each can be implemented with a XOR-tree. In (X. Wang, D. Wu & C. Hu, 2009), an optimization method based on common subexpression elimination (CSE) is employed to optimize and reduce the logic complexity.

4.2 High-speed BCH decoder implementation

Based on the proposed algorithm, a high-speed self-adaptive DEC BCH decoder is design and its architecture is depicted in Fig. 13. Once the input codeword is received from NOR flash memory array, the two syndromes S_1, S_3 are firstly obtained by 18 parallel XOR-trees. Then, the proposed fast-decoding algorithm is employed to calculate the coefficients of error location polynomial in the R calculator module. Meanwhile, a short-path is implemented for SEC decoding once the syndrome value S_1 is obtained. Finally, variable R determines whether SEC decoding or DEC decoding should be performed and selects the according data path at the output.

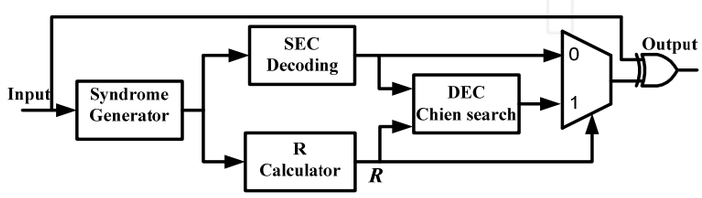


Fig. 13. Block diagram of the proposed DEC BCH decoder.

The performance of an embedded BCH (274,256,2) decoder in NOR flash memory is summarized in Table 2. The decoder is synthesized with Design Compiler and implemented in 180nm CMOS process. It has 2-bit error correction capability and achieves decoding

latency of 4.60ns. In addition, it can be seen that the self-adaptive decoding is very effective to speed up the decoding and reduce the power consumption for 1-bit error correction. The DEC BCH decoder satisfies the short latency and high reliability requirement of NOR flash memory.

Code Parameter	BCH(274, 256) codes	
Information data	256 bits	
Parity bits	18 bits	
Syndrome time	1.66ns	
Data output time	1-bit error	3.53ns
	2-bit errors	4.60ns
Power consumption (Vdd=1.8V, T=70ns)	1-bit error	0.51mW
	2-bit error	1.25mW
Cell area	0.251 mm2	

Table 2. Performance of a high-speed and self-adaptive DEC BCH decoder

5. LDPC ECC in NAND flash memory

As raw BER in NAND flash increases to close to 10^{-2} at its life end, hard-decision ECC, such as BCH code, is not sufficient any more, and such more powerful soft-decision ECC as LDPC code becomes necessary. The outstanding performance of LDPC code is based on soft-decision information.

5.1 Soft-decision log-likelihood information from NAND flash

Denote the sensed threshold voltage of a cell as V_{th}, the distribution of erase state as $p_0(x)$, the distribution of programmed states as $p_k(x)$, where k is the index of programmed state. Denote S_i as the set of the states whose i-th bit is 0. Thus, given the V_{th}, the LLR of i-th code bit in one cell is:

$$L(b_i) = \log \frac{\sum_{k \in S_i} p^{(k)}(V_{th})}{\sum_k p^{(k)}(V_{th}) - \sum_{k \in S_i} p^{(k)}(V_{th})} \tag{29}$$

Clearly, LLR calculation demands the knowledge of the probability density functions of all the states, and threshold voltage of concerned cells.

There exist many kinds of noises, such as cell-to-cell interference, random-telegraph noise, retention process and so on, therefore it would be unfeasible to derive the closed-form distribution of each state, given the NAND flash channel model that captures all those noise sources. We can rely on Monte Carlo simulation with random input to get the distribution of all states after being interrupted by several noise sources in NAND flash channel. With random data to be programmed into NAND flash cells, we run a large amount of simulation on the NAND flash channel model to get the distribution of all states, and the obtained threshold voltage distribution would be very close to real distribution under a large amount of simulation. In practice, the distribution of V_{th} can be obtained through fine-grained sensing on large amount of blocks.

In sensing flash cell, a number of reference voltages are serially applied to the corresponding control gate to see if the sensed cell conduct, thus the sensing result is not the exact target threshold voltage but a range which covers the concerned threshold voltage. Denote the sensed range as $[R_l, R_r)$ (R_l and R_r are two adjacent reference voltages). There is be $R_l \leq V_{th} < R_r$.

Example 2: Let's consider a 2-bit-per-cell flash cell with threshold voltage of 1.3V. Suppose the reference voltage starts from 0V, with incremental step of 0.3V. The reference voltages applied to the flash cell is: 0, 0.3V, 0.6V, 0.9V, 1.2V, 1.5V ... This cell will not be open until the reference voltage of 1.5V is applied, so the sensing result is that the threshold voltage of this cell stays among (1.2, 1.5].

The corresponding LLR of i-th bit in one cell is then calculated as

$$L(b_i) = \log \frac{\int_{R_l}^{R_r} \sum_{k \in \mathcal{S}_i} p^{(k)}(x)dx}{\int_{R_l}^{R_r} \sum_{k} p^{(k)}(x)dx - \int_{R_l}^{R_r} \sum_{k \in \mathcal{S}_i} p^{(k)}(x)dx} \tag{30}$$

5.2 Performance of LDPC code in NAND flash

With the NAND flash model presented in section 2 and the same parameters as those in Example 1, the performances of (34520, 32794, 107) BCH code and (34520, 32794) QC-LDPC codes with column weight 4 are presented in Fig. 14, where floating point sensing is assumed on NAND flash cells. The performance advantage of LDPC code is obvious.

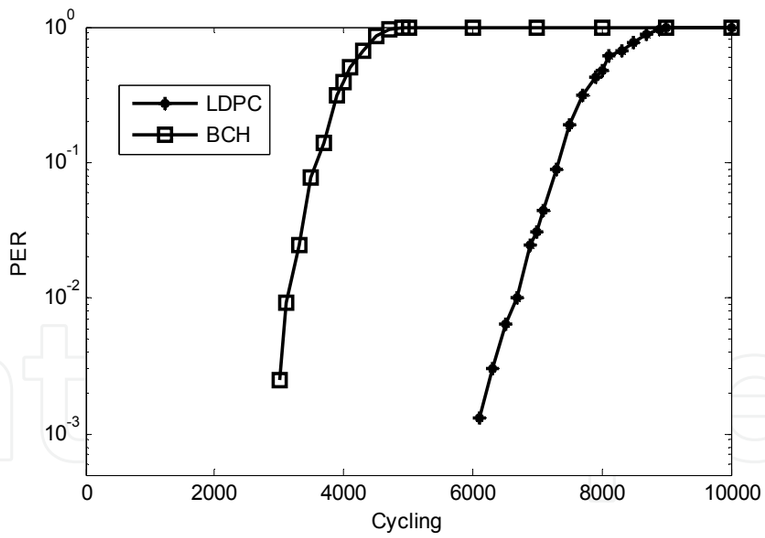


Fig. 14. Page error rate performances of LDPC and BCH codes with the same coding rate under various program/erase cycling.

5.3 Non-uniform sensing in NAND flash for soft-decision information

As mentioned above, sensing flash cell is performed through applying different reference voltages to check if the cell can open, so the sensing latency directly depends on the number of applied sensing levels. To provide soft-decision information, considerable amount of sensing levels are necessary, thus the sensing latency is very high compared to hard-decision sensing.

Soft-decision sensing increases not only the sensing latency, but also the data transfer latency from page buffer to flash controller, since these data is transferred in serial.

Example 3: Let's consider a 2-bit-per-cell flash cell with threshold voltage of 1.3V. Suppose the hard reference voltages as 0, 0.6V and 1.2V respectively. Suppose sensing one reference voltage takes 8us. The page size is 2K bytes and I/O bus works as 100M Hz with 8-bit width. For hard-decision sensing, we need to apply all three hard reference voltages to sense it out, resulting in sensing latency of 24us. To sense a page for soft-decision information with 5-bit precision, we need $2^5 * 8 = 256$ us, more than ten times the hard-decision sensing latency. With 5-bit soft-decision information per cell, the total amount of data is increased by 2.5 times, thus the data transfer latency is increased by 2.5 times, from 20.48 us to 51.2us. The overall sensing and transfer latency jumps to 51.2+256=307.2 us from 20.48+24=44.48 us.

Based on above discussion, it is highly desirable to reduce the amount of soft-decision sensing levels for the implementation of soft-decision ECC. Conventional design practice tends to simply use a uniform fine-grained soft-decision memory sensing strategy as illustrated in Fig. 15, where soft-decision reference voltages are uniformly distributed between two adjacent hard-decision reference voltages.

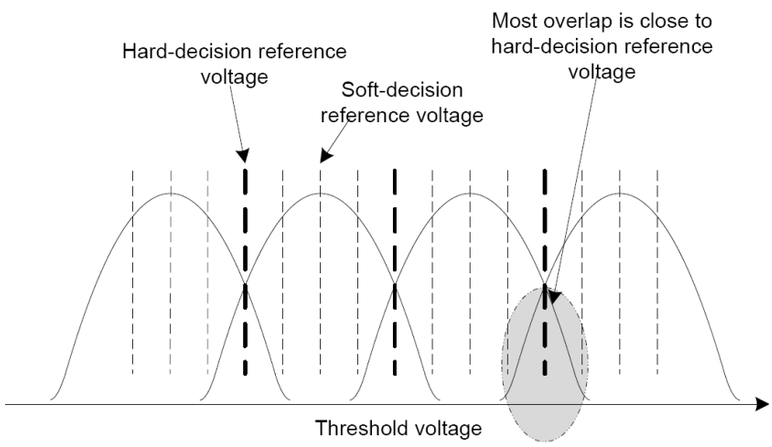


Fig. 15. Illustration of the straightforward uniform soft-decision memory sensing. Note that soft-decision reference voltages are uniformly distributed between any two adjacent hard-decision reference voltages.

Intuitively, since most overlap between two adjacent states occurs around the corresponding hard-decision reference voltage (i.e., the boundary of two adjacent states) as illustrated in Fig. 15, it should be desirable to sense such region with a higher precision and leave the remainder region with less sensing precision or even no sensing. This is a non-uniform or non-linear memory sensing strategy, through which the same amount of sensing voltages is expected to provide more information.

Given a sensed threshold voltage V_{th}, its entropy can be obtained as

$$H(V_{th}) = \sum_{k} P(\text{state} = k|V_{th}) \log \frac{1}{P(\text{state} = k|V_{th})} \tag{31}$$

Where

$$P(\text{state} = k|V_{th}) = \frac{p^{(k)}(V_{th})}{\sum_v p^{(v)}(V_{th})} \tag{32}$$

For one given programmed flash memory cell, there are always just one or two items being dominating among all the $P(\text{state} = k|V_{th})$ items for the calculation of $H(V_{th})$. Outside of the dominating overlap region, there is only one dominating item very close to 1 while all the other items being almost 0, so the entropy will be very small. On the other hand, within the dominating overlap region, there are two relatively dominating items among all the $P(\text{state} = k|V_{th})$ items, and both of them are close to 0.5 if V_{th} locates close to the hard-decision reference voltage, i.e., the boundary of two adjacent states, which will result in a relatively large entropy value $H(V_{th})$. Clearly the region with large entropy tends to demand a higher sensing precision. So, it is intuitive to apply a non-uniform memory sensing strategy as illustrated in Fig. 16. Associated with each hard-decision reference voltage at the boundary of two adjacent states, a so-called *dominating overlap region* is defined and uniform memory sensing is executed only within each dominating overlap region.

Given the sensed V_{th} of a memory cell, the value of entropy $H(V_{th})$ is mainly determined by two largest probability items, and this translates into the ratio between the two largest probability items. Therefore, such a design trade-off can be adjusted by a probability ratio R, i.e., let $[B_l^{(k)}, B_r^{(k)}]$ denote the dominating overlap region between two adjacent states, we can determine the border $B_l^{(k)}$ and $B_r^{(k)}$ by solving

$$\frac{p^{(k)}(B_l^{(k)})}{p^{(k+1)}(B_l^{(k)})} = \frac{p^{(k+1)}(B_r^{(k)})}{p^{(k)}(B_r^{(k)})} = R \tag{33}$$

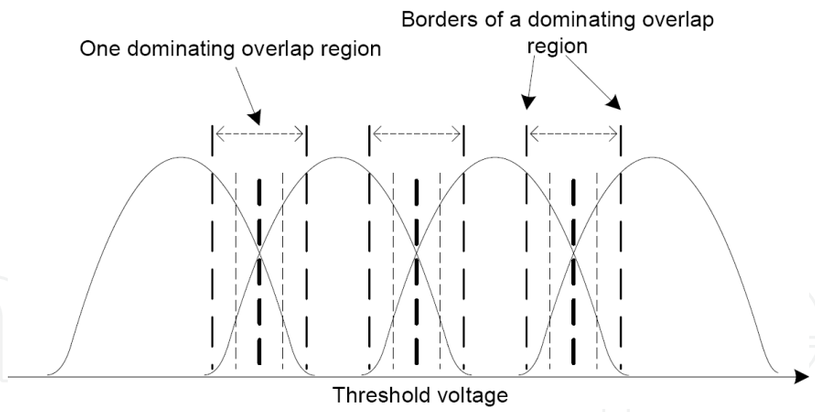


Fig. 16 Illustration of the proposed non-uniform sensing strategy. Dominating overlap region is around hard-decision reference voltage, and all the sensing reference voltages only distribute within those dominating overlap regions.

Since each dominating overlap region contains one hard-decision reference voltage and two borders, at least $3(K-1)$ sensing levels should be used in non-uniform sensing. Simulation results on BER performance of rate-19/20 (34520, 32794) LDPC codes in uniform and non-uniform sensing under various cell-to-cell interference strengths for 2 bits/cell NAND flash are presented in Fig. 17. Note that at least 9 non-uniform sensing levels is required for non-uniform sensing for 2 bits/cell flash. The probability ratio R is set as 512. Observe that

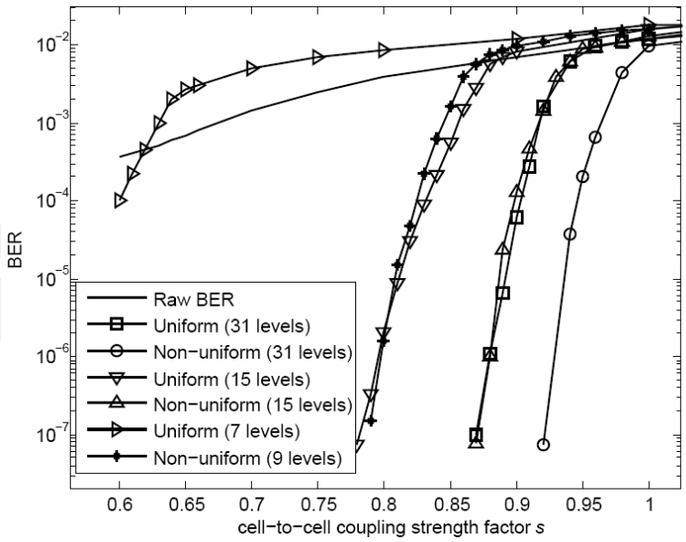


Fig. 17. Performance of LDPC code when using the non-uniform and uniform sensing schemes with various sensing level configurations.

15-level non-uniform sensing provides almost the same performance as 31-level uniform sensing, corresponding to about 50% sensing latency reduction. 9-level non-uniform sensing performs very closely to 15-level uniform sensing, corresponding to about 40% sensing latency reduction.

6. Signal processing for NAND flash memory

As discussed above, as technology continues to scale down and hence adjacent cells become closer, parasitic coupling capacitance between adjacent cells continues to increase and results in increasingly severe cell-to-cell interference. Some study has clearly identified cell-to-cell interference as the major challenge for future NAND flash memory scaling. So it is of paramount importance to develop techniques that can either minimize or tolerate cell-to-cell interference. Lots of prior work has been focusing on how to minimize cell-to-cell interference through device/circuit techniques such as word-line and/or bit-line shielding. This section presents to employ signal processing techniques to tolerate cell-to-cell interference.
According to the formation of cell-to-cell interference, it is essentially the same as inter-symbol interference encountered in many communication channels. This directly enables the feasibility of applying the basic concepts of post-compensation, a well known signal processing techniques being widely used to handle inter-symbol interference in communication channel, to tolerate cell-to-cell interference.

6.1 Technique I: Post-compensation

It is clear that, if we know the threshold voltage shift of interfering cells, we can estimate the corresponding cell-to-cell interference strength and subsequently subtract it from the sensed threshold voltage of victim cells. Let $\tilde{V}_t^{(k)}$ denote the sensed threshold voltage of the k-th interfering cell and $\bar{V}_e$ denote the mean of erased state, we can estimate the threshold voltage shift $\Delta\tilde{V}_t^{(k)}$ of each interfering cell as $(\tilde{V}_t^{(k)} - \bar{V}_e)$. Let $\bar{\gamma}^{(k)}$ denote the mean of the corresponding coupling ratio, we can estimate the strength of cell-to-cell interference as

$$\tilde{F} = \sum_{k} \left((\tilde{V}_t^{(k)} - \bar{V}_e) \cdot \bar{\gamma}^{(k)} \right) \tag{34}$$

Therefore, we can post-compensate cell-to-cell interference by subtracting estimated $\tilde{F}$ from the sensed threshold voltage of victim cells. In [Dong, Li & Zhang, 2010], the authors presents simulation result of post-compensation on one initial NAND flash channel with the odd/even structure. Fig. 18 shows the threshold voltage distribution before and after post-compensation. It's obvious that post-compensation technique can effectively cancel interference.

Note that the sensing quantization precision directly determines the trade-off between the cell-to-cell interference compensation effectiveness and induced overhead. Fig. 19 and Fig. 20 show the simulated BER vs. cell-to-cell coupling strength factor s for even and odd pages, where 32-level and 16-level uniform sensing quantization schemes are considered. Simulation results clearly show the impact of sensing precision on the BER performance. Under 32-level sensing, post-compensation could provide large BER performance improvement, while 16-level sensing degrades the odd cells' performance when cell-to-cell interference strength is low.

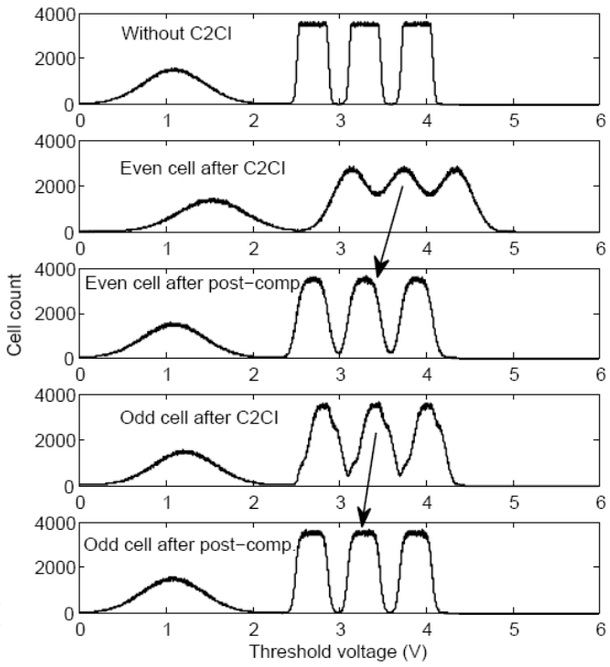


Fig. 18. Simulated victim cell threshold voltage distribution before and after post-compensation.

Reverse Programming for Reading Consecutive Pages

To execute post-compensation for concerned page, we need the threshold voltage information of its interfering page. When consecutive pages are to be read, information on the interfering pages become inherently available, hence we can capture the approximate threshold voltage shift and estimate the corresponding cell-to-cell interference on the fly during the read operations for compensation.

Since sensing operation takes considerable latency, it would be feasible to run ECC decoding on the concerned page first, and sensing the interfering page will not be started until that ECC decoding fails, or will be started while ECC decoding is running.

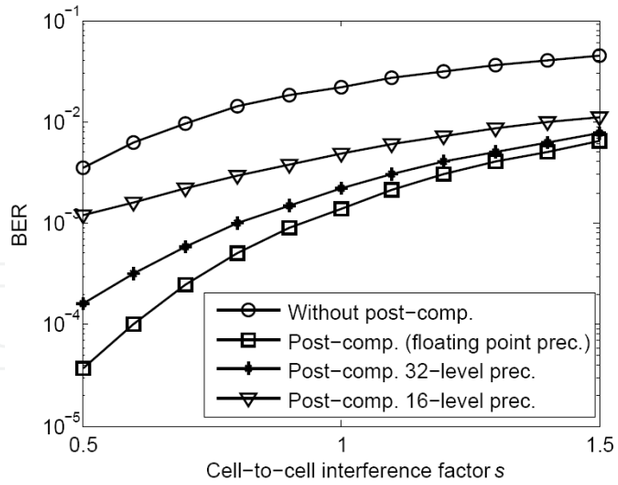


Fig. 19. Simulated BER performance of even cells when post-compensation is used.

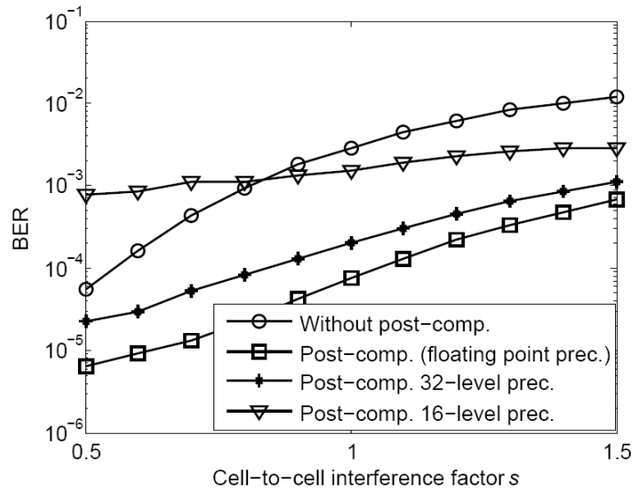


Fig. 20. Simulated BER performance of odd cells when post-compensation is used.

Note that pages are generally programmed and read both in the same order, i.e. page with lower index is programmed and read prior to page with higher index in consecutive case. Since later programmed page imposes interference on previously programmed neighbor page, as a result, one victim page is read before its interfering page is read in reading consecutive pages, hence extra read latency is needed to wait for reading interfering page of each concerned page. In the case of consecutive pages reading, all consecutive pages are concerned pages, and each page acts as the interfering page to the previous page and meanwhile is the victim page of the next page. Intuitively, reversing the order of programming pages to be descending order, i.e., pages with lower index are programmed latter, meanwhile reading pages in the ascending order can eliminate this extra read latency in reading consecutive pages. This is named as *reverse programming* scheme.

In this case, when we read those consecutive pages, after one page is read, it can naturally serve to compensate cell-to-cell interference for the page being read later. Therefore the extra sensing latency on waiting for sensing interfering page is naturally eliminated. Note that this reverse programming does not influence the sensing latency of reading individual pages.

6.2 Technique II: Pre-distortion

Pre-distortion or pre-coding technique widely used in communication system can also be used in NAND flash: Before a page is programmed, if its interfering pages are also known, we can predict the threshold voltage shift induced by cell-to-cell interference for each victim cell, and then correspondingly pre-distort the victim cell target programming voltage. Hence, after its interfering pages are programmed, the pre-distorted victim cell threshold voltages is expected to shift to its desired location by cell-to-cell interference.

Let $V_t^{(k)}$ denote the expected threshold voltage of the k-th interfering cell after programming and V_e denote the mean of erased state, we can predict the cell-to-cell interference experienced by the victim cell as

$$\hat{F} = \sum_k \left((V_t^{(k)} - \bar{V}_e) \cdot \bar{\gamma}^{(k)} \right) \tag{35}$$

Let V_p denote the target verify voltage of the victim cell in programming operation, we can pre-distort the victim cell by shifting the verify voltage from V_p to $V_p - \hat{F}$. The threshold voltage of the victim cell will be shifted towards its desired location after the occurrence of cell-to-cell interference. It should be emphasized that, since we cannot change the threshold voltage if the victim cell should stay at the erased state, this pre-distortion scheme can only handle cell-to-cell interference for those programmed states but is not effective for erased state.

Fig. 21 illustrates the process of pre-distortion, where the verify voltage V_p is assumed to be able to be adjusted with a floating-point precision. Clearly, this technique can be considered as a counterpart of the post-compensation technique.

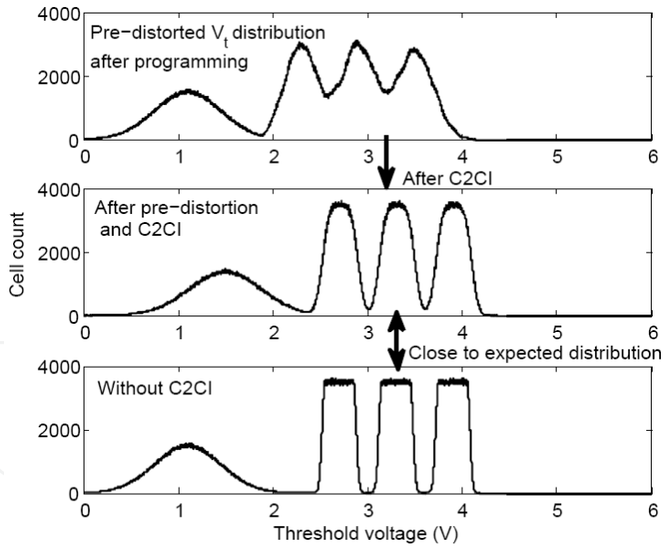


Fig. 21. Illustration of threshold voltage distribution of victim even cells in even/odd structure when data pre-distortion is being used.

Fig.22 shows the cell threshold distribution with the cell-to-cell interference strength factor $s = 0.8$ under the same initial NAND flash channel model as in above subsection, where the pre-distortion is assumed to be able to be adjusted with a floating-point precision.

Fig. 23 shows the simulated BER of even cells over a range of cell-to-cell interference strength factor s. Besides the ideal floating point precision, pre-distortion with finite precision is also shown, where the range of pre-distorted V_p is quantized into either 16 or 32

levels. Clearly, as the finite quantization precision of pre-distorted V_p increases, it can achieve a better tolerance to cell-to-cell interference, at the cost of increased programming latency, a larger page buffer to hold the data and higher chip-to-chip communication load.

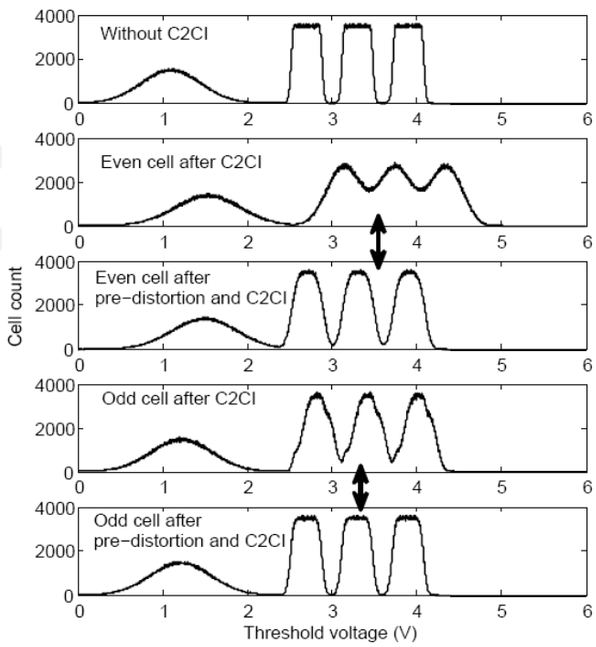


Fig. 22. Simulated threshold voltage distribution when using pre-distortion.

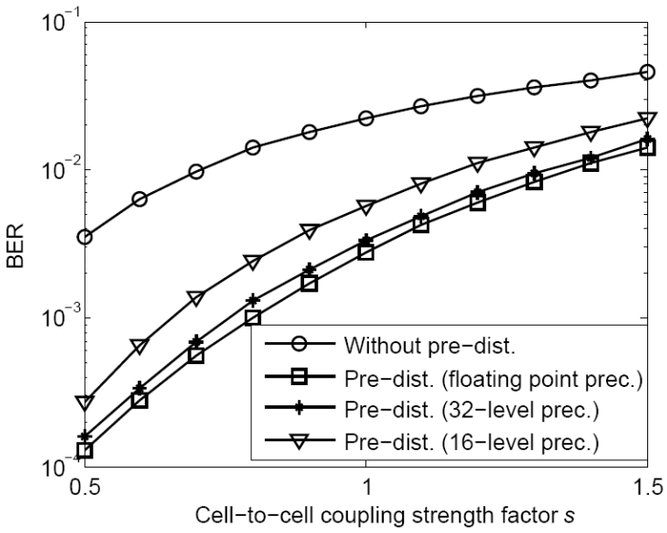


Fig. 23. The simulated BER of even cells with pre-distortion under various cell-to-cell strength factor.

7. Reference

K. Kim et.al, "Future memory technology: Challenges and opportunities," in *Proc. of International Symposium on VLSI Technology, Systems and Applications*, Apr. 2008, pp. 5–9.

G. Dong, S. Li, and T. Zhang, "Using Data Post-compensation and Pre-distortion to Tolerat Cell-to-Cell Interference in MLC NAND Flash Memory", *IEEE Transactions on Circuits and Systems I*, vol. 57, issue 10, pp. 2718-2728, 2010

Y. Li and Y. Fong, "Compensating for coupling based on sensing a neighbor using coupling," *United States Patent* 7,522,454, Apr. 2009.

G. Dong, N. Xie, and T. Zhang, "On the Use of Soft-Decision Error Correction Codes in NAND Flash Memory", *IEEE Transactions on Circuits and Systems I*, vol. 58, issue 2, pp. 429-439, 2011

E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, pp. 138–163, June 2005.

Y. Pan, G. Dong, and T. Zhang, "Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance", *USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2011

G. Dong, N. Xie, and T. Zhang, "Techniques for Embracing Intra-Cell Unbalanced Bit Error Characteristics in MLC NAND Flash Memory", *Workshop on Application of Communication Theory to Emerging Memory Technologies* (in conjection with IEEE Globecom), Dec. 2010

N. Mielke et al., "Bit error rate in NAND flash memories," in *Proc. of IEEE International Reliability Physics Symposium*, 2008, pp. 9–19.

K. Kanda et al., "A 120mm2 16Gb 4-MLC NAND flash memory with 43nm CMOS technology," in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2008, pp. 430–431,625.

Y. Li et al., "A 16 Gb 3-bit per cell (X3) NAND flash memory on 56 nm technology with 8 MB/s write rate," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 195–207, Jan. 2009.

S.-H. Chang et al., "A 48nm 32Gb 8-level NAND flash memory with 5.5MB/s program throughput," in *Proc. of IEEE International Solid-State Circuits Conference*, Feb. 2009, pp. 240–241.

N. Shibata et al., "A 70nm 16Gb 16-level-cell NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 43, pp. 929–937, Apr. 2008.

C. Trinh et al., "A 5.6MB/s 64Gb 4b/cell NAND flash memory in 43nm CMOS," in *Proc. of IEEE International Solid-State Circuits Conference*, Feb. 2009, pp. 246–247.

K. Takeuchi et al., "A 56-nm CMOS 99-mm2 8-Gb multi-level NAND flash memory with 10-mb/s program throughput," *IEEE Journal of Solid-State Circuits*, vol. 42, pp. 219–232, Jan. 2007.

G. Matamis et al., "Bitline direction shielding to avoid cross coupling between adjacent cells for NAND flash memory," *United States Patent* 7,221,008, May. 2007.

J. W. Lutze and N. Mokhlesi, "Shield plate for limiting cross coupling between floating gates," *United States Patent* 7,335,237, Apr. 2008.

H. Chien and Y. Fong, "Deep wordline trench to shield cross coupling between adjacent cells for scaled NAND," *United States Patent* 7,170,786, Jan. 2007.

S. Li and T. Zhang, "Improving multi-level NAND flash memory storage reliability using concatenated BCH-TCM coding," *IEEE Transactions on Circuits and Systems-I: Regular Papers*, vol. PP, pp. 1–1, 2009.

K. Prall, "Scaling non-volatile memory below 30 nm," in *IEEE 2nd Non-Volatile Semiconductor Memory Workshop*, Aug. 2007, pp. 5–10.

H. Liu, S. Groothuis, C. Mouli, J. Li, K. Parat, and T. Krishnamohan, "3D simulation study of cell-cell interference in advanced NAND flash memory," in *Proc. of IEEEWorkshop on Microelectronics and Electron Devices*, Apr. 2009.

K.-T. Park et al., "A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 40, pp. 919–928, Apr. 2008.

K. Takeuchi, T. Tanaka, and H. Nakamura, "A double-level-Vth select gate array architecture for multilevel NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 31, pp. 602–609, Apr. 1996.

K.-D. Suh et al., "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, pp. 1149–1156, Nov. 1995.

C. M. Compagnoni et al., "Random telegraph noise effect on the programmed threshold-voltage distribution of flash memories," *IEEE Electron Device Letters*, vol. 30, 2009.

A. Ghetti, et al., "Scaling trends for random telegraph noise in deca-nanometer flash memories," *in IEEE International Electron Devices Meeting*, 2008, 2008, pp. 1–4.

J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron. Device Letters*, vol. 23, pp. 264–266, May 2002.

K. Takeuchi et al., "A 56-nm CMOS 8-Gb multi-level NAND flash memory with 10-MB/s program throughput," *IEEE Journal of Solid-State Circuits*, vol. 42, pp. 219–232, Jan. 2007.

Y. Li et al., "A 16 Gb 3 b/cell NAND flash memory in 56 nm with 8 MB/s write rate," in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2008, pp. 506–632.

R.-A. Cernea et al., "A 34 MB/s MLC write throughput 16 Gb NAND with all bit line architecture on 56 nm technology," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 186–194, Jan. 2009.

N. Shibata et al., "A 70 nm 16 Gb 16-level-cell NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 43, pp. 929–937, Apr. 2008.

H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach," *IEEE Transactions on Circuits and Systems-I: Regular Papers*, vol. 52, no. 4, pp. 766–775, 2005.

I. Alrod and M. Lasser, "Fast, low-power reading of data in a flash memory," in *United States Patent* 20090319872A1, 2009.

Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results", *IEEE Trans. Inf. Theory*, vol. 47, pp. 2711-2736, Nov. 2001.

R. G. Gallager, "Low density parity check codes", *IRE Trans. Inf. Theory*, vol. 8, pp. 21-28, Jan. 1962.

G. Dong, Y. Li, N. Xie, T. Zhang and H. Liu, "Candidate bit based bit-flipping decoding algorithm for LDPC codes", *IEEE ISIT* 2009, pp. 2166-2168, 2009

J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes", *IEEE Commun. Lett.*, vol. 8, pp. 165-167, Mar. 2004.

F. Guo and L. Hanzo, "Reliability ratio based weighted bit-flipping decoding for low-density parity-check codes", *Electron. Lett.*, vol. 40, pp. 1356-1358, Oct. 2004.

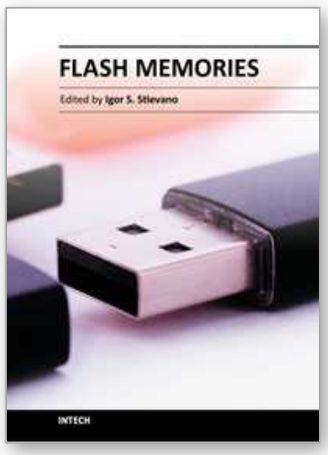
C.-H. Lee and W. Wolf, "Implementation-efficient reliability ratio based weighted bit-flipping decoding for LDPC codes", *Electron. Lett.*, vol. 41, pp. 755-757, Jun. 2005.

D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes", *Electron. Lett.*, vol. 32, pp. 1645–1646, Aug. 1997.

X. Wang, L. Pan, D. Wu et al., "A High-Speed Two-Cell BCH Decoder for Error Correcting in MLC NOR Flash Memories", *IEEE Trans. on Circuits and Systems II*, vol.56, no.11, pp.865-869, Nov. 2009.

X. Wang, D. Wu, C. Hu, et al., "Embedded High-Speed BCH Decoder for New Generation NOR Flash Memories" *Proc. IEEE CICC 2009*, pp. 195-198, 2009.

R. Micheloni, R. Ravasio, A. Marelli, et al., "A 4Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36MB/s system read throughput", *Proc. IEEE ISSCC*, pp. 497-506, Feb. 2006.



Flash Memories
Edited by Prof. Igor Stievano

ISBN 978-953-307-272-2
Hard cover, 262 pages
Publisher InTech
Published online 06, September, 2011
Published in print edition September, 2011

Flash memories and memory systems are key resources for the development of electronic products implementing converging technologies or exploiting solid-state memory disks. This book illustrates state-of-the-art technologies and research studies on Flash memories. Topics in modeling, design, programming, and materials for memories are covered along with real application examples.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Xueqiang Wang, Guiqiang Dong, Liyang Pan and Runde Zhou (2011). Error Correction Codes and Signal Processing in Flash Memory, Flash Memories, Prof. Igor Stievano (Ed.), ISBN: 978-953-307-272-2, InTech, Available from: http://www.intechopen.com/books/flash-memories/error-correction-codes-and-signal-processing-in-flash-memory

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen