Design of on-chip error correction systems for multilevel NOR and NAND flash memories

F. Sun, S. Devarajan, K. Rose and T. Zhang

Abstract: The design of on-chip error correction systems for multilevel code-storage NOR flash and data-storage NAND flash memories is concerned. The concept of trellis coded modulation (TCM) has been used to design on-chip error correction system for NOR flash. This is motivated by the non-trivial modulation process in multilevel memory storage and the effectiveness of TCM in integrating coding with modulation to provide better performance at relatively short block length. The effectiveness of TCM-based systems, in terms of error-correcting performance, coding redundancy, silicon cost and operational latency, has been successfully demonstrated. Meanwhile, the potential of using strong Bose–Chaudhiri–Hocquenghem (BCH) codes to improve multilevel data-storage NAND flash memory capacity is investigated. Current multilevel flash memories store 2 bits in each cell. Further storage capacity may be achieved by increasing the number of storage levels per cell, which nevertheless will correspondingly degrade the raw storage reliability. It is demonstrated that strong BCH codes can effectively enable the use of a larger number of storage levels per cell and hence improve the effective NAND flash memory storage capacity up to 59.1% without degradation of cell programming time. Furthermore, a scheme to leverage strong BCH codes to improve memory defect tolerance at the cost of increased NAND flash cell programming time is proposed.

1 Introduction

Driven by the ever increasing demand for on-chip/board non-volatile data storage, flash memory has become one of the fastest growing segments in the global semiconductor industry [1]. Flash memories are categorised into two families, NOR flash and NAND flash [2]: NOR flash memories are mainly used for code storage and have relatively short block length, for example, 16 or 64 user bits per block, whereas NAND flash memories are mainly used for massive data storage and have relatively long block length, for example, 8192 or 16,384 user bits (i.e. 1024 or 2048 user bytes) per block. With its well-demonstrated effectiveness for increasing flash memory storage capacity, the multilevel concept, that is, to store more than 1 bit in each cell (or floating-gate MOS transistor) by programming the cell threshold voltage into one of $l > 2$ voltage windows, is being widely used in both NOR and NAND flash memories [3–7]. Owing to the inherently reduced operational margin, multilevel flash memories are increasingly relying upon on-chip error correction to ensure the storage reliability [8–10]. In current practice, most multilevel NOR and NAND flash memories store 2 bits in each memory cell and employ classical linear block error correction codes (ECCs) such as Hamming and Bose–Chaudhuri–Hocquenghem (BCH) codes to realise on-chip error correction.

This work is interested in the design of multilevel flash memory on-chip error correction systems that may outperform the current practice by realising superior reliability and/or enabling higher effective storage capacity. Because of the significant difference on the block length between NOR and NAND flash memories, we consider these two types of flash memories separately. In the context of NOR flash, we investigate the use of trellis coded modulation (TCM) [11] technique to realise on-chip error correction. The motivation is 2-fold: (1) The more-than-two-levels-per-cell storage capacity makes the modulation process non-trivial and an integral part of the on-chip ECC. (2) TCM can effectively integrate ECC with modulation to realise better error correction performance when the block length is relatively small. We note that, although the use of TCM in multilevel memory has been first proposed in [12], the incurred hardware implementation cost and latency overhead have not been addressed, which leaves its practical feasibility a missing link. Furthermore, TCM-based approach is only applicable to NOR flash because its advantage over conventional linear block codes quickly diminishes as the block length increases, which however was not pointed out in [12]. To evaluate the silicon cost of using TCM-based on-chip error correction, we implemented the read datapath consisting of high-precision sensing circuits and TCM decoder. The results suggest that TCM-based systems can achieve encouraging memory cell savings at small operational latency and silicon cost.

In the context of NAND flash, we investigated the use of very strong BCH codes to enable higher storage capacity. Currently, most multilevel NAND flash memories store 2 bits (or four levels) in each cell, for which a weak ECC code that can only correct few (e.g. one or two) errors is typically used [13]. Higher storage capacity may be realised by further increasing l, which will make it increasingly more difficult to ensure storage reliability. In this regard, solutions may be pursued along two directions, including: (i) improve the

doi:10.1049/iet-cds:20060275

Paper first received 3rd September 2006 and in revised form 7th April 2007

The authors are with the ECSE Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

E-mail: sunf@rpi.edu

programming scheme to accordingly tighten each threshold voltage window and (ii) use much stronger ECC. Along the first direction, researchers have developed high-accuracy programming techniques to realise 3bits/cell and even 4bits/cell storage capacity [14, 15], which however complicates the design of the peripheral programming mixed signal circuits and degrades the programming throughput.

To the best of our knowledge, the potential of using much stronger ECC to improve NAND flash storage capacity has not been addressed in the open literature. This work attempts to fill this gap by investigating the use of strong BCH codes to enable a relatively large l (6, 8 and 12 in this work). With the advantages of simplifying the programming circuits and maintaining or even increasing the programming throughput, the use of strong ECC is subject to two main drawbacks: (i) strong ECC requires a higher coding redundancy that will inevitably degrade the storage capacity improvement gained by a larger l and (ii) the ECC decoder may incur non-negligible silicon area overhead and increase read latency. In general, to realise the same error correction performance (or to achieve the same coding gain), the longer the ECC code length, the less will be the relative coding redundancy (or higher code rate). Therefore strong ECCs are only suitable for NAND flash memories that have long data block length and hence may tolerate longer read latency. Using 2 bits/cell NAND flash memories that employ single-error-correcting Hamming codes as a benchmark, we investigated the effectiveness of using strong BCH codes to ensure storage reliability when increasing the value of l to 6, 8 and 12, respectively. With the same programming scheme, and hence the same threshold voltage distribution characteristics as the 2 bits/cell benchmark, the larger value of l will result in a worse raw storage reliability and demands a stronger BCH code. To investigate the trade-off between design complexities and storage capacity improvements, we designed BCH decoders using 0.13 μm complementary metal–oxide–semiconductor (CMOS) standard cell and static random access memory (SRAM) libraries. The results show that strong BCH codes can enable a relatively large increase of the number of storage levels per cell and hence a potentially significant memory storage capacity improvement. Finally, a scheme is proposed to leverage strong BCH codes to improve NAND flash memory defect tolerance by trading off the memory cell programming time.

The paper is organised as follows: We briefly present the basics of multilevel flash memories in Section 2. The proposed TCM-based on-chip error correction systems for multilevel NOR flash memories is presented in Section 3, and Section 4 discusses the use of strong BCH codes in multilevel NAND flash memories. Conclusions are drawn in Section 5.

2 Multilevel flash memories

This section briefly presents some basics of multilevel flash memory programming/read and the memory cell threshold voltage distribution model to be used in this work. Interested readers are referred to [3] for a comprehensive discussion on multilevel flash memories. Multilevel flash memory programming is typically realised by combining a program-and-verify technique with a staircase V_{pp} ramp as illustrated in Fig. 1. The tightness of each programming threshold voltage window is proportional to V_{pp}, whereas the cell programming time is roughly proportional to $1/V_{pp}$. The read circuit in l levels/cell NAND flash memories usually has a serial sensing structure that takes

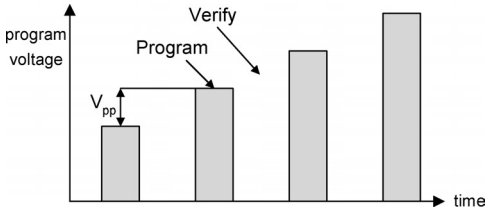

Fig. 1 *Schematic illustration of program-and-verify cell programming*

$l-1$ cycles to finish the read operation. Higher read speed can be realised by increasing the sensing parallelism at the cost of silicon area, which is typically preferred in latency-critical NOR flash memories.

On the basis of the results published in [16] for 2 bits/cell NOR flash memory, the cell threshold voltage approximately follows a Gaussian distribution as illustrated in Fig. 2: the two inner distributions have the same standard deviation, denoted as σ; the standard deviations of the two outer distributions are 4σ and 2σ, respectively. The locations of the means of the two inner distributions are determined to minimise the raw bit error rate. Let V_{max} denote the voltage difference between the means of the two outer distributions. We assume that this model is also valid for NAND flash memories.

3 TCM-based on-chip error correction for NOR flash

3.1 TCM system structure

The basic idea of TCM is to jointly design trellis codes (i.e. convolutional codes) and signal mapping (i.e. modulation) processes to maximise the free Euclidean distance (Similar to the Hamming distance of linear block codes, free Euclidean distance determines the error correction capability of convolutional codes, that is, a convolutional code with free Euclidean distance of d_{free} can correct at least $\lfloor (d_{free}-1)/2 \rfloor$ code symbol errors) between coded signal sequences. As illustrated in Fig. 3, given an l-level/cell memory core, an m-dimensional TCM encoder receives a sequence of n-bit input data, adds r-bit redundancy and hence generates a sequence of $(n+r)$-bit data, where each $(n+r)$-bit data are stored in m memory cells and $2^{n+r} \leq l^m$. The encoding process can be outlined as follows: (1) A convolutional encoder convolves the input k bits sequence with r linear algebraic functions and generates $k+r$ coded bits. (2) Each $k+r$ coded bits select one of the 2^{k+r} subsets of an $m-D$ signal constellation, where each subset contains 2^{n-k} signal points. (3) The additional $n-k$ uncoded bits select an individual $m-D$ signal point from the selected subset.

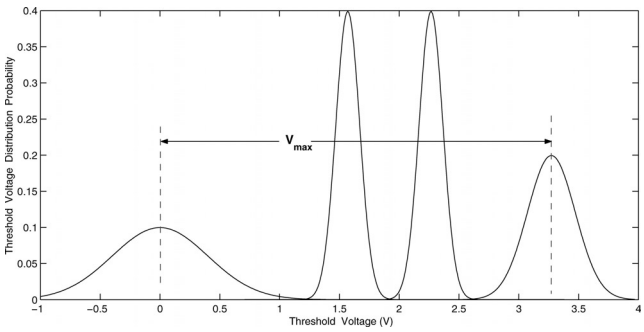

Fig. 2 *Approximate cell threshold voltage distribution model in 2 bits/cell memory*

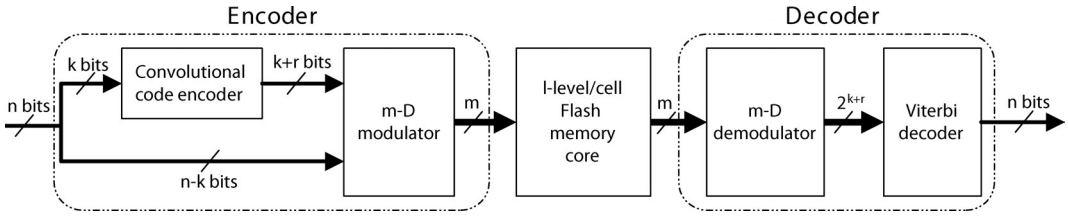

Fig. 3 *Block diagram of TCM-based on-chip error correction system*

Let s denote the memory order of the convolutional code encoder. To protect an N-bit data block, the TCM encoder totally receives $N+s$ bits including s zero bits for convolutional code termination. If $N+s$ is not divisible by n, the last input to the encoder will contain less than n bits, for which the $m-D$ modulation may be simplified to a modulation with a lower dimension. As illustrated in Fig. 3, the TCM decoder contains an $m-D$ demodulator that provides 2^{k+r} branch metrics and branch symbol decisions to the Viterbi decoder for trellis decoding.

3.2 System design and performance evaluation

Targeting 2 bits/cell NOR flash memories, we designed three TCM-based systems that protect 16-bit, 32-bit and 64-bit user data in one codeword. These three systems share the same system design parameters (referred to Fig. 3): $n=7$, $k=2$, $r=1$, $m=4$ and the memory order of the convolutional code $v=3$. The signal read from each memory cell is quantised by 12 levels. We decided to use 12-level quantisation mainly based on our finite-precision computer simulations, which suggested 12-level quantisation appears to provide a good trade-off between implementation cost and error correction performance. To realise 4D modulation, we use the scheme proposed by Wei [17] that hierarchically partitions the 4D rectangular lattice formed by four memory cells into eight 4D sub-lattices. Each coded 3-bit data from the convolutional code encoder selects one out of the eight 4D sub-lattices. The 4D signal space partition is described as follows: First, we partition each 1D signal constellation (corresponding to one memory cell) into two subsets E and F, as shown in Fig. 4, where the signal points labelled as e and f belong to the subsets E and F, respectively.

Next, we partition each 2D signal constellation into four subsets $A=(E,F)$, $B=(F,E)$, $C=(E,E)$ and $D=(F,F)$, as shown in Fig. 4, where the signal points labelled as a, b, c and d belong to the subsets A, B, C, and D, respectively. Finally, we partition the 4D signal constellation into $2^{k+r}=8$ 4D subsets formed as listed in Table 1.

To protect 16-bit user data, the TCM encoder receives 19 bits (including 3 zero bits for termination) and finishes encoding in three steps: during each of the first two steps, it receives 7 bits and maps the coded 8 bits onto four memory cells through 4D modulation; in the last step, it receives 5 bits and maps the coded 6 bits onto three memory cells through 3D modulation that is obtained by collapsing one 2D constellation in the original 4D modulation into a 1D constellation. Therefore this system is denoted as (11, 8) TCM, that is, one codeword occupies 11 memory cells and protects $2\times 8=16$-bit user data (notice that each memory cell stores 2 bits). For the purpose of comparison, we considered two other ECC schemes using linear block codes: (i) a (11, 8, 1) 4-ary shortened Hamming code and (ii) a (13,8,2) 4-ary shortened two-error-correcting BCH code [18].

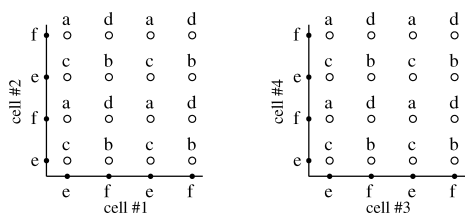

Fig. 4 *16-point 2D signal constellation partition*

Fig. 5*a* shows the performance comparison of these three schemes. Although the performance curves of the two linear block codes can be analytically derived, we have to rely on extensive computer simulations to obtain the performance curve of the (11, 8) TCM system, for which the solid part is obtained by computer simulation and the dashed part is estimated following the trend of the simulation results. With the same coding redundancy, (11, 8) TCM can achieve about five orders of magnitude better performance than (11, 8, 1) Hamming code. Compared with (13, 8, 2) BCH code, (11, 8) TCM can achieve almost the same performance while realising a saving of 2/13 (15.4%) memory cells.

To protect 32-bit user data, the TCM encoder receives 35 bits and finishes encoding in five steps, each step maps 8 bits onto four memory cells through 4D modulation. Hence, this is denoted as (20, 16) TCM, that is, one codeword occupies 20 memory cells and protects 32-bit user data. The (20, 16) TCM is compared with two linear block codes: (i) a (19, 16, 1) 4-ary shortened Hamming code and (ii) a (23, 16, 2) 4-ary shortened BCH code. Fig. 5*b* shows their performance comparison.

To protect 64-bit user data, the TCM encoder receives 67 bits and finishes encoding in 10 steps: during each of the first nine steps, it receives 7 bits and maps the coded 8 bits onto four memory cells through 4D modulation; in the last step, it receives 4 bits that bypass the convolutional code encoder and directly map onto two memory cells through 2D modulation that is a constituent of the original 4D modulation. Hence, this is denoted as (38, 32) TCM. We compared it with two linear block codes: (i) a (36, 32, 1) 4-ary shortened Hamming code and (ii) a (39, 32, 2) 4-ary shortened BCH code. Fig. 5*c* shows their performance

Table 1: Partition of the 4-D signal constellation

4D subset	Concatenation form
$\mathcal{P}_1$	$(A, A)\cup(B, B)$
$\mathcal{P}_2$	$(C, C)\cup(D, D)$
$\mathcal{P}_3$	$(A, B)\cup(B, A)$
$\mathcal{P}_4$	$(C, D)\cup(D, C)$
$\mathcal{P}_5$	$(A, C)\cup(B, D)$
$\mathcal{P}_6$	$(C, B)\cup(D, A)$
$\mathcal{P}_7$	$(A, D)\cup(B, C)$
$\mathcal{P}_8$	$(C, A)\cup(D, B)$

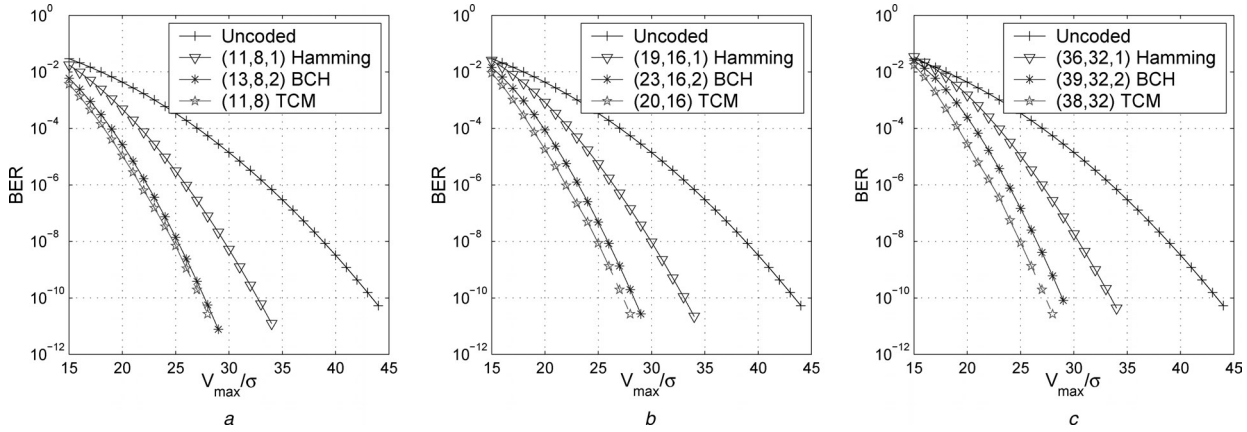

Fig. 5 *BER performance when protecting 16-bit, 32-bit and 64-bit user data*
In TCM schemes, the signal read from each memory cell is quantised by 12 levels

comparison. Table 2 summarises the comparison between the TCM and the other ECC schemes discussed above in terms of coding redundancy and error-correcting performance. Notice that the positive and negative numbers in the third column mean positive and negative saving of memory cells, respectively, and the performance gain in the fourth column is measured when the bit error rate(BER) of TCM-based system approaches 10^{-14}.

3.3 Silicon implementation

The above result shows the effectiveness of TCM-based on-chip error correction in terms of coding redundancy and error-correcting performance. However, to be a promising candidate for multilevel NOR flash memory, it should be able to achieve small latency and negligible silicon area compared with the overall memory die size. In the following, we present proof-of-concept implementation results for the above three TCM-based systems for protecting 16-bit, 32-bit and 64-bit user data. Clearly, TCM encoders are very simple and can easily achieve very small latency with negligible silicon cost. Hence we only focus on TCM decoders.

The TCM decoding datapath contains high-precision sensing circuits, 4D demodulator and Viterbi decoder. The sensing circuit realises 12-level quantisation, instead of 4-level quantisation as in the conventional linear-block-code-based ECC scheme. Using Cadence tool with IBM 0.18 μm 7WL technology, we designed a current-mode 12-level parallel sensing circuit following the structure proposed in [19]. Fig. 6 shows the general structure of a 12-level current-mode parallel sensing circuit that mainly contains 11 current comparators. 12-level quantisation is realised by comparing the current from the selected memory cell with the reference currents from the 11 reference cells which are appropriately programmed. The silicon area of one 12-level current-mode parallel sensing circuit is estimated as 0.006 mm^2. The simulation results show that the worst-case sensing latency (i.e. the input current is equal to one of the reference currents) is about 300 ps.

Upon receiving the data from four 12-level sensing circuits, the 4D demodulator finds the most likely point in each 4D signal subset and calculates the corresponding log-likelihood metric as the branch metrics sent to the Viterbi decoder. The output branch metrics are represented with 6 bits. The 4D demodulator receives a set of data denoted as $\hat{Z} = \{\hat{z}_1, \hat{z}_2, \hat{z}_3, \hat{z}_4\}$ from the four analogue-to-digital converters (ADCs), where each $\hat{z}_i$ is the digitised data read from one cell. Given $\hat{Z}$, the 4D demodulator should calculate

$$\text{4D\_Metric\_}j = \max_{p \in \mathcal{P}_j} (\log P(\hat{Z}|p)),$$
$$\text{for } j = 1, 2, \ldots, 8 \qquad (1)$$

where each 4D_Metric_j represents the log-likelihood value of one most likely point in each 4D subset and p represents the point in each 4D subset $\mathcal{P}_j$. As each point in each 4D subset is represented by 5 bits, the 4D demodulator should also generate eight 5-bit data representing the most likely points in the eight 4D subsets. Leveraging the hierarchical structure of the 4D modulation, as shown in Fig. 7, the demodulation is realised in a hierarchical manner. The 4D demodulator starts with finding the

Table 2: Comparison between TCM and the other ECC schemes based on linear block code

TCM	Competing ECC	Savings of cells, %	Performance gain
(11, 8)	(11, 8, 1) Hamming	0	$\sim 10^5$
	(13, 8, 2) BCH	15.4	~ 1
(20, 16)	(19, 16, 1) Hamming	−5.3	$\sim 10^5$
	(23, 16, 2) BCH	13	~ 10
(38, 32)	(36, 32, 1) Hamming	−5.6	$> 10^5$
	(39, 32, 2) BCH	2.6	> 10

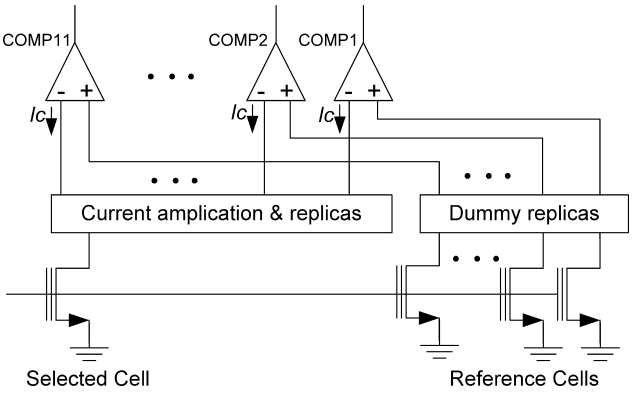

Fig. 6 *Structure of a 12-level parallel sensing circuit*

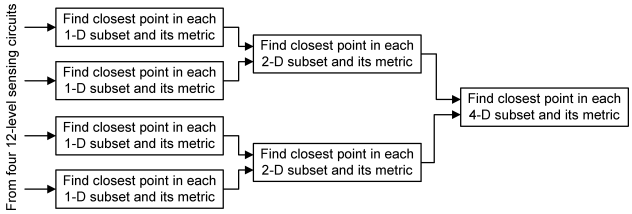

Fig. 7 *Data flow of the 4D demodulator*

closest point in each 1D subset and its metric. Each 1D demodulator receives the data $\hat{z}_i$ read from one cell and calculates

$$\begin{aligned} \text{Metric\_}E &= \max_{p\in E}(\log P(\hat{z}_i|p)), \\ \text{Metric\_}F &= \max_{p\in F}(\log P(\hat{z}_i|p)) \end{aligned} \tag{2}$$

where Metric_E and Metric_F represent the log-likelihood value of one most likely point in two 1D subsets E and F, respectively. The 1D demodulator also generates 1 bit (note that we have two signal points in each 1D subset) to represent the closest point in each 1D subset. This operation can be implemented using a simple look-up table.

As discussed in Section 3.2, each 2D signal constellation is partitioned into four subsets $A = (E, F)$, $B = (F, E)$, $C = (E, E)$, and $D = (F, F)$. Upon the received data $\{\hat{z}_i, \hat{z}_{i+1}\}$ for $i = 1, 3$, the 2D demodulator generates

$$\begin{aligned} \text{Metric\_}A &= \max_{p\in A}(\log P(\{\hat{z}_i, \hat{z}_{i+1}\}|p)) \\ &= \text{Metric\_}E + \text{Metric\_}F \\ \text{Metric\_}B &= \max_{p\in B}(\log P(\{\hat{z}_i, \hat{z}_{i+1}\}|p)) \\ &= \text{Metric\_}F + \text{Metric\_}E \\ \text{Metric\_}C &= \max_{p\in C}(\log P(\{\hat{z}_i, \hat{z}_{i+1}\}|p)) \\ &= \text{Metric\_}E + \text{Metric\_}E \\ \text{Metric\_}D &= \max_{p\in D}(\log P(\{\hat{z}_i, \hat{z}_{i+1}\}|p)) \\ &= \underbrace{\text{Metric\_}F}_{\text{1st 1D}} + \underbrace{\text{Metric\_}F}_{\text{2nd 1D}} \end{aligned} \tag{3}$$

In a similar way, the 4D demodulator generates eight demodulation metrics as follows

$$\begin{aligned} \text{4D\_Metric\_1} &= \max_{p\in\mathcal{P}_1}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}A + \text{Metric\_}A, \\ &\quad \text{Metric\_}B + \text{Metric\_}B) \\ \text{4D\_Metric\_2} &= \max_{p\in\mathcal{P}_2}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}C + \text{Metric\_}C, \\ &\quad \text{Metric\_}D + \text{Metric\_}D) \\ \text{4D\_Metric\_3} &= \max_{p\in\mathcal{P}_3}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}A + \text{Metric\_}B, \\ &\quad \text{Metric\_}B + \text{Metric\_}A) \\ \text{4D\_Metric\_4} &= \max_{p\in\mathcal{P}_4}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}C + \text{Metric\_}D, \\ &\quad \text{Metric\_}D + \text{Metric\_}C) \\ \text{4D\_Metric\_5} &= \max_{p\in\mathcal{P}_5}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}A + \text{Metric\_}C, \\ &\quad \text{Metric\_}B + \text{Metric\_}D) \\ \text{4D\_Metric\_6} &= \max_{p\in\mathcal{P}_6}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}C + \text{Metric\_}B, \\ &\quad \text{Metric\_}D + \text{Metric\_}A) \\ \text{4D\_Metric\_7} &= \max_{p\in\mathcal{P}_7}(\log P(\hat{Z}|p)) \\ &= \max(\text{Metric\_}A + \text{Metric\_}D, \\ &\quad \text{Metric\_}B + \text{Metric\_}C) \\ \text{4D\_Metric\_8} &= \max_{p\in\mathcal{P}_8}(\log P(\hat{Z}|p)) \\ &= \max\left(\underbrace{\text{Metric\_}C}_{\text{1st 2D}} + \underbrace{\text{Metric\_}A}_{\text{2nd 2D}}, \right. \\ &\quad \left. \underbrace{\text{Metric\_}D}_{\text{1st 2D}} + \underbrace{\text{Metric\_}B}_{\text{2nd D}}\right) \end{aligned} \tag{4}$$

The generated eight metric values will be sent to Viterbi decoder as the branch metrics for final decoding.

The last block on the decoding datapath is a Viterbi decoder. To minimise the decoding latency, we use a state-parallel register-exchange Viterbi decoder architecture. As Viterbi decoder implementation has been extensively addressed in the open literature, we will not elaborate on the decoder architecture details. Interested readers are referred to [20]. Here we note that, for the scenario of protecting 16-bit user data, as the Viterbi decoder finishes the decoding in only three steps, we directly unrolled the recursive datapath of the original Viterbi decoder and fully optimise the circuit's structure, which reduces both silicon area and decoding latency. Table 3 summarises the read datapath implementation metrics of the three TCM systems. We note that the TCM systems protecting 32-bit and 64-bit user data contain four sensing circuits and one 4D demodulator, whereas the TCM system protecting 16-bit user data contains 11 sensing circuits and two 4D and one 3D demodulators in order to match the parallelism of the unrolled Viterbi decoder, and hence the silicon area of the TCM system protecting 16-bit user data is comparable to the other two scenarios, even with the least block length.

4 BCH-based on-chip error correction for NAND flash

4.1 Binary BCH codes

Binary BCH code construction and encoding/decoding are based on binary Galois fields. A binary Galois field with degree of m is represented as GF(2^m). For any $m \geq 3$ and $t > 2^{m-1}$, there exists a primitive binary BCH code over GF(2^m), which has the code length $n = 2^{m-1}$ and information bit length $k \geq 2^m - mt$ and can correct up to (or slightly more than) t errors. A primitive t-error-correcting (n, k, t) BCH code can be shortened (i.e. eliminate a

Table 3: Summary of implementation metrics

	Silicon area, mm^2	Latency[a], ns
(11, 8) TCM	0.12	8.3
(20, 16) TCM	0.10	24.3
(38, 32) TCM	0.12	44.3

[a]Includes latency of sensing circuits and TCM decoding

certain number, say s, of information bits) to construct a t-error-correcting $(n - s, k - s, t)$ BCH code with less information bits and code length but the same redundancy. Given the raw BER p_{raw}, an (n, k, t) binary BCH code can achieve a codeword error rate of

$$P_{\text{e}} = \sum_{i=t+1}^{n} \binom{n}{i} p_{\text{raw}}^{i} (1 - p_{\text{raw}})^{n-i} \tag{5}$$

Binary BCH encoding can be realised efficiently using linear shift registers, whereas binary BCH decoding is much more complex. Various BCH decoding algorithms have been proposed [21]. In Section 4.3, we will elaborate on the binary BCH decoding algorithm and decoder architecture used in this work.

4.2 BCH codes for multilevel NAND flash

We first investigate the potential storage capacity improvement by increasing l from 4 to 6, 8 and 12, respectively. Assuming the same programming scheme (i.e. the same step-up voltage V_{pp} and hence same cell programming time) as the 2 bits/cell memory, we have the cell threshold voltage distributions for $l = 6$, 8 and 12 as illustrated in Fig. 8 and described as follows: the $l - 2$ inner distributions have the same standard deviation σ; the standard deviations of the two outer distributions are 4σ and 2σ, respectively. The locations of the means of the $l - 2$ inner distributions are determined to minimise the raw BER. It should be pointed out that, as the value of l increases, some factors such as floating-gate interference [22] and source line noise [23] might degrade the threshold voltage distribution (or increase the standard deviation). As currently no data are available in the open literature to model such possible deviation degradation and we expect that such degradation should not be significant, we assume that the standard deviation is independent of l in this work.

We set $V_{\max}$, the voltage difference between the means of the two outer distributions, as 6.5 V [24] and σ as 1. For l of 6, 8 and 12, we store 5 bits per two cells, 3 bits per cell and 7 bits per two cells, respectively. Accordingly, the raw BER are about 8×10^{-12} $(l = 4)$, 5×10^{-7} $(l = 6)$, 5×10^{-5} $(l = 8)$ and 2×10^{-3} $(l = 12)$, respectively. Because the cell programming time remains the same as the 2 bits/cell benchmark, the programming throughput may approximately increase by 25, 50, and 75%, respectively.

To protect 8192 and 16,384 user bits per codeword with a target codeword error rate of lower than 10^{-14}, single-error-correcting Hamming codes will be sufficient to ensure the storage reliability for $l = 4$. For larger values of l, binary BCH codes are constructed by shortening primitive binary BCH codes under GF(2^{14}) and GF(2^{15}), respectively. Table 4 lists the BCH code parameters and the corresponding codeword error rates. Table 4 also shows the percentages of the user bits storage gain over the 2 bits/cell benchmark, given the same number of memory cells.

4.3 BCH code decoder architecture and ASIC design

To evaluate decoder silicon implementation metrics for the above BCH codes, we carried out application–specific integrated circuit(ASIC) design using 0.13 μm CMOS standard cell and SRAM libraries. In the following, we first briefly describe the BCH decoder architecture and then present the silicon implementation results. A syndrome-based binary BCH code decoder consists of three blocks, as shown in Fig. 9. For an (n, k, t) binary BCH code constructed under a Galois field with the primitive element α, the overall decoder architecture is described as follows.

4.3.1 Syndrome computation: Given the received bit vector $\boldsymbol{r}$, it computes $2t$ syndromes as $S_i = \sum_{j=0}^{n-1} r_j \alpha^{ij}$ for $i = 0, 1, \ldots, 2t - 1$. As pointed out in [25] for binary BCH codes, we have $S_{2j} = S_j^2$, so only t parallel syndrome generators are required to explicitly calculate the odd-indexed syndromes, followed by much simpler square circuits. For a decoder with parallelism of p (i.e. the syndrome computation block receives p input bits in each

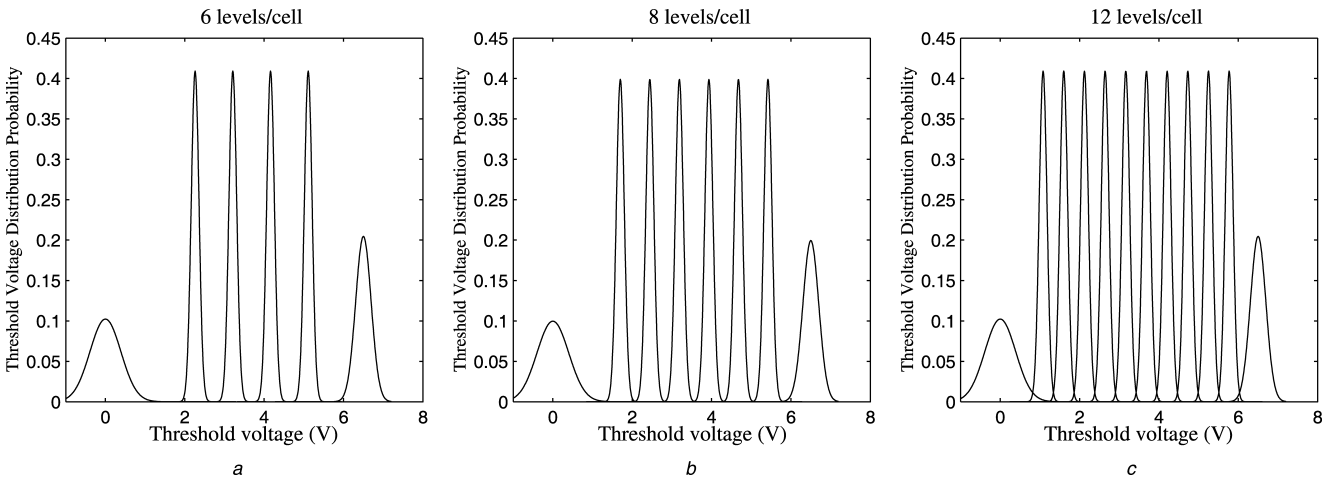

Fig. 8 *Approximate flash memory cell threshold voltage distribution model*
a $l = 6$
b $l = 8$
c $l = 12$

Table 4: BCH codes parameters and performance

l	(n, k, t) BCH codes	Codeword error rate	User bits storage gain, %
6	(8262, 8192, 5)	1.1×10^{-17}	24.0
8	(8360, 8192, 12)	3.1×10^{-15}	47.0
12	(9130, 8192, 67)	2.8×10^{-15}	57.0
6	(16 459, 16 384, 5)	7.0×10^{-16}	24.5
8	(16 609, 16 384, 15)	3.2×10^{-15}	48.0
12	(17 914, 16 384, 102)	7.2×10^{-15}	60.1

clock cycle), each syndrome generator has the structure as shown in Fig. 10.

4.3.2 Error locator calculation: Based on the $2t$ syndromes, we calculate the error locator polynomial $\Lambda(x) = 1 + \Lambda_1 x + \Lambda_2 x^2 + \cdots + \Lambda_t x^t$ using the inversion-free Berlekamp–Massey algorithm [26]. To minimise the silicon area cost, a fully serial architecture is used, which takes $t(t+3)/2$ clock cycles to finish the calculation. It mainly contains three Galois field multipliers and two first-input first-output (FIFO) buffers with lengths of t and $t+1$, respectively.

4.3.3 Chien search: Upon receiving the error locator polynomial $\Lambda(x)$, it exhaustively examines whether α^i is the root of $\Lambda(x)$ for $i = 0, 1, \ldots, n-1$, that is, check whether $\Lambda(\alpha^i) = \sum_{j=1}^{t} \Lambda_j \alpha^{ij} + 1$ is zero or not. It outputs an error vector $\boldsymbol{e}$ in such a way that, if α^i is a root, then $e_{n-i} = 1$, otherwise $e_{n-i} = 0$. The overall decoder output is obtained by $\boldsymbol{r} + \boldsymbol{e}$ as illustrated in Fig. 9. Fig. 11 shows the Chien search architecture with the parallelism factor of p that generates a p-bit output each clock cycle.

4.3.4 Decoder ASIC design: For the BCH codes listed in Table 4, we designed decoders with the following configurations: the syndrome computation and Chien search blocks have a parallelism factor of 4; the error locator calculation block is fully serial and takes $t(t+3)/2$ clock cycles. Therefore the syndrome computation and Chien search blocks always have the same latency (in terms of the number of clock cycles), whereas the latency of error locator calculation block depends on the value of t. To improve the decoding throughput and minimise the decoding latency, these BCH decoders support pipelined operation summarised as follows:

- For $l = 6, 8$: The BCH codes have relatively small values of t, so the corresponding error locator calculation blocks have much less latency than the other two blocks. Therefore we use a one-stage pipelined decoder structure in which the syndrome computation and error locator calculation blocks operate on one codeword, whereas the Chien search block operates on the other codeword in parallel.
- For $l = 12$: The BCH codes have relatively large values of t, so the corresponding error locator calculation blocks have similar or even slightly longer latency than the other two blocks. Therefore we use a two-stage pipelined decoder structure in which these three blocks operate in parallel on three consecutive codewords.

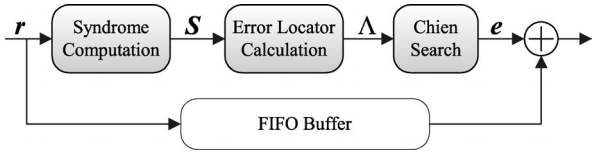

Fig. 9 *Binary BCH code decoder structure*

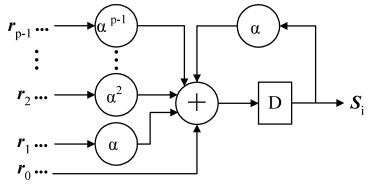

Fig. 10 *Structure of one syndrome generator with parallelism factor of p*

Furthermore, the decoder FIFO as shown in Fig. 9 is realised by SRAMs to minimise the silicon area cost. These BCH decoders are designed with Chartered 0.13 μm CMOS standard cell and SRAM libraries, where Synopsys tools are used throughout the design hierarchy down to place and route. We set the power supply as 1.08 V and the number of metal layers as four in the place and route. Post-layout results verify that the decoders can operate at 400 MHz and hence support about 1.6 Gbps decoding throughput because of the decoder parallelism factor of 4. Such throughput appears to be sufficient in real-life applications [27]. The silicon area and decoding latency are listed in Table 5.

To demonstrate the overall NAND flash memory storage capacity improvement potential, we carried out the following estimation for 70-nm CMOS technology: The effective NAND flash memory cell size is 0.024 μm^2 at 70-nm CMOS technology [7]. We scale the BCH decoder silicon area by $(130/70)^2 = 3.4$ to estimate the decoder silicon area at 70-nm CMOS technology. Accordingly, Table 6 shows the estimated total numbers of user bits that can be stored in a NAND flash memory core of 100 mm^2 while considering the BCH decoder silicon area cost. The effective storage capacity improvement is obtained by comparing against the 2 bits/cell benchmark.

4.4 Integration with defect tolerance

We assumed above that the cell programming time (and hence threshold voltage distribution) remains the same for various l and BCH codes are solely used for compensating threshold voltage distribution-induced (TVDI) errors. It is intuitive that, if we improve the programming accuracy by reducing the step-up programming voltage V_{pp} at the cost of increased programming time, the TVDI error rates will correspondingly reduce. This will leave a certain degree of BCH code error correction capability available for compensating memory defects. This can be considered as a trade-off between programming time and defect tolerance.

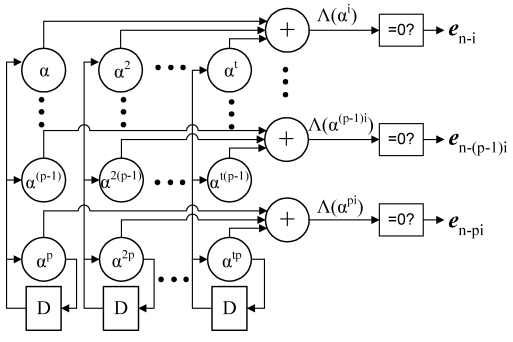

Fig. 11 *Structure of Chien search with the parallelism factor of p*

Table 5: BCH decoder ASIC design post-layout results

l	(n, k, t) BCH codes	Silicon area, mm^2	Latency, μs
6	(8262, 8192, 5)	0.21	10.4
8	(8360, 8192, 12)	0.32	10.9
12	(9130, 8192, 67)	1.43	17.6
6	(16 459, 16 384, 5)	0.25	20.7
8	(16 609, 16 384, 15)	0.38	21.4
12	(17 914, 16 384, 102)	2.14	40.2

Table 6: Estimated storage capacity for a 100 mm^2 NAND flash memory core

	(n, k, t) BCH codes	Stored user bits, Gbits	Effective storage capacity improvement
4		8.33	–
6	(8262, 8192, 5)	10.32	23.9
8	(8360, 8192, 12)	12.24	46.9
12	(9130, 8192, 67)	13.03	56.4
6	(16 459, 16 384, 5)	10.36	24.4
8	(16 609, 16 384, 15)	12.32	47.9
12	(17 914, 16 384, 102)	13.25	59.1

Following this intuition, we investigate such a trade-off in NAND flash memories with l of 6, 8 and 12, respectively. On the basis of the cell threshold voltage distribution model as presented above, if we reduce the step-up programming voltage V_{pp} to improve the programming accuracy, the standard deviation of the threshold voltage distribution will accordingly reduce. In this work, we simply assume that the standard deviation σ is inversely proportional to the cell programming time. If a t-error-correcting binary BCH code needs to compensate up to d_{def} defective memory cells, it will only be able to correct up to $t_{TVDI} = t - d_{def}$ TVDI errors. To

Table 7: Trading TVDI error correction capability for defect tolerance

l	(n, k, t) BCH codes	Standard deviation σ	d_{def}	t_{TVDI}
6	(8262, 8192, 5)	0.833	1	2
8	(8360, 8192, 12)	0.930	1	9
		0.719	3	3
12	(9130, 8192, 67)	0.950	4	53
		0.900	7	42
		0.800	12	24
		0.571	17	6
6	(16 459, 16 384, 5)	0.800	1	2
8	(16 609, 16 384, 15)	0.950	1	12
		0.700	4	3
12	(17 914, 16 384, 102)	0.950	6	81
		0.900	12	60
		0.800	20	32
		0.571	27	6

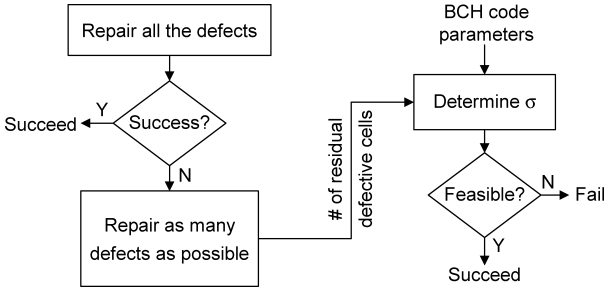

Fig. 12 *Flow diagram using BCH codes for defect tolerance*

accommodate such TVDI error correction capability loss, we have to accordingly reduce the TVDI error rates by improving the programming accuracy and hence reducing the standard deviation parameter σ. For the BCH codes listed in Table 4, we have to shown the trade-offs between σ and d_{def} as in Table 7.

Based on the above discussion, we further propose a modified multilevel flash memory defect-tolerant strategy by combining the conventional spare rows/columns repair and BCH codes. As illustrated in Fig. 12, we first check whether the available spare rows/columns can repair all the defects in one memory block, if not, then we carry out a certain repair algorithm to use the spare rows/columns to repair as many defects as possible so that the number of residual defective cells can be minimised. Then we calculate how to adjust the threshold voltage distribution deviation parameter σ in order to compensate the TVDI error correction capability loss. Finally, we check whether the target σ is feasible, subject to some practical constraints such as circuit precision and minimum allowable cell programming time.

5 Conclusions

This paper presented on-chip error correction system design approaches for multilevel code-storage NOR flash and data-storage NAND flash memories. We applied the TCM concept to design an on-chip error correction system for multilevel NOR flash memories. Compared with the conventional practice using linear block codes, the TCM-based design solution can provide better coding redundancy against error-correcting performance trade-offs. Targeting 2 bits/cell NOR flash, we designed TCM-based systems for three scenarios where the number of user bits per block is 16, 32 and 64, respectively. Compared with the systems using two-error-correcting BCH codes, the TCM-based systems can achieve ~1 order of magnitude better BER while saving 15.3% (16-bit), 13.0% (32-bit) and 2.6% (64-bit) memory cells, respectively. Cadence and Synopsys tools were used to implement the read datapath including mixed signal sensing circuits and digital TCM demodulation and decoding circuits. The latency and silicon area are 8.3 ns and 0.12 mm^2 (16-bit), 24.3 ns and 0.10 mm^2 (32-bit), and 44.3 ns and 0.12 mm^2 (64-bit), respectively.

In the context of NAND flash memory, we demonstrated the promise of using strong BCH codes to further improve multilevel data-storage NAND flash memory capacity without degrading memory programming time. Targeting the codeword error rate lower than 10^{-14}, we constructed BCH codes with 8192 and 16384 user bits per codeword, respectively. It shows that, given the same number of memory cells, up to 60% more user bits can be stored compared with the 2 bits/cell benchmark. To evaluate decoder

silicon area and achievable decoding throughput/latency, we implemented BCH decoders using 0.13 μm CMOS standard cell and SRAM libraries. Post-layout results verify that the decoders occupy (much) less than 2.5 mm^2 silicon area and achieves (much) less than 41 μs decoding latency and 1.6 Gbps decoding throughput. On the basis of the published results for NAND flash effective cell area and a simple scaling rule, we estimate that, under 70-nm CMOS technology and 100 mm^2 core area, up to 59.1% effective storage capacity improvement can be realised compared with 2 bits%cell benchmark.

Furthermore, we propose a design strategy that can leverage the large error correction capability of strong BCH codes to improve memory defect tolerance by trading off the memory cell programming time.

6 Acknowledgments

The authors thank the anonymous reviewers for their valuable comments and suggestions, which have largely improved the quality and presentation of this paper.

7 References

1 Hwang, C.: 'Nanotechnology enables a new memory growth model', *Proc. IEEE*, 2003, **91**, pp. 1765–1771
2 Bez, R., Camerlenghi, E., Modelli, A., and Visconti, A.: 'Introduction to Flash memory', *Proc. IEEE*, 2003, **91**, pp. 489–502
3 Ricco, B., *et al.*: 'Nonvolatile multilevel memories for digital applications', *Proc. IEEE*, 1998, **86**, pp. 2399–2423
4 Lee, S., *et al.*: 'A 3.3 V 4 Gb four-level NAND flash memory with 90 nm CMOS technology'. Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), 2004, pp. 52–513
5 Sim, S.-P., *et al.*: 'A 90 nm generation NOR flash multilevel cell (MLC) with 0.44 μm^2/bit cell size'. IEEE VLSI-TSA Int. Symp. on VLSI Technology, April 2005, pp. 35–36
6 Servalli, G., *et al.*: 'A 65nm NOR flash technology with 0.042 μm^2 cell size for high performance multilevel application'. IEEE Int. Electron Devices Meeting, December 2005, pp. 849–852
7 Hara, T., *et al.*: 'A 146-mm^2 8-Gb multi-level NAND flash memory with 70-nm CMOS technology', *IEEE J. Solid-State Circuits*, 2006, **41**, pp. 161–169
8 Gregori, S., Cabrini, A., Khouri, O., and Torelli, G.: 'On-chip error correcting techniques for new-generation Flash memories', *Proc. IEEE*, 2003, **91**, pp. 602–616
9 Silvagni, A., Fusillo, G., Ravasio, R., Picca, M., and Zanardi, S.: 'An overview of logic architectures inside Flash memory devices', *Proc. IEEE*, 2003, **91**, pp. 569–580
10 Rossi, D., Metra, C., and Ricco, B.: 'Fast and compact error correcting scheme for reliable multilevel flash memories'. Proc. Eighth IEEE Int. On-Line Testing Workshop, July 2002, pp. 221–225
11 Ungerboeck, G.: 'Trellis-coded modulation with redundant signal sets. Parts I and II', *IEEE Commun. Mag.*, 1987, **25**, pp. 5–21
12 Lou, H.-L., and -Sundberg, C.E.W.: 'Increasing storage capacity in multilevel memory cells by means of communications and signal processing techniques', *IEE Proc., Circuits Devices Syst.*, 2000, **147**, pp. 229–236
13 Tanzawa, T., *et al.*: 'A compact on-chip ECC for low cost flash memories', *IEEE J. Solid-State Circuits*, 1997, **32**, pp. 662–669
14 Nobukata, H., *et al.*: 'A 144-Mb, eight-level NAND flash memory with optimized pulsewidth programming', *IEEE J. Solid-State Circuits*, 2000, **35**, pp. 682–690
15 Grossi, M., Lanzoni, M., and Ricco, B.: 'A novel algorithm for high-throughput programming of multilevel flash memories', *IEEE Trans. Electron Devices*, 2003, **50**, pp. 1290–1296
16 Atwood, G., Fazio, A., Mills, D., and Reaves, B.: 'Intel StrataFlash™ memory technology overview' Intel Technology Journal', 4th Quarter 1997, pp. 1–8
17 Wei, L.F.: 'Trellis-coded modulation with multidimensional constellations', *IEEE Trans. Inf. Theory*, 1987, **33**, pp. 483–501
18 Sun, F., Devarajan, S., Rose, K., and Zhang, T.: 'Multilevel flash memory on-chip error correction based on trellis coded modulation'. IEEE Int. Symp. Circuits and Systems (ISCAS), May 2006
19 Calligaro, C., Gastaldi, R., Manstretta, A., and Torelli, G.: 'A high-speed parallel sensing scheme for multi-level nonvolatile memories'. Proc. Int. Workshop on Memory Technology, Design and Testing, August 1997, pp. 96–101
20 Fettweis, G., and Meyr, H.: 'High-speed parallel Viterbi decoding: algorithm and VLSI-architecture', *IEEE Commun. Mag.*, 1991, **29**, pp. 46–55
21 Blahut, R.E.: 'Algebraic codes for data transmission' (Cambridge University Press, 2003)
22 Lee, J.-D., Hur, S.-H., and Choi, J.-D.: 'Effects of floating-gate interference on NAND flash memory cell operation', *IEEE Trans. Electron Devices*, 2002, **23**, pp. 264–266
23 Takeuchi, K., Tanaka, T., and Nakamura, H.: 'A double-level-V_{th} select gate array architecture for multilevel NAND flash memories', *IEEE J. Solid-State Circuits*, 1996, **31**, pp. 602–609
24 Micheloni, R., *et al.*: 'A 0.13-μm CMOS NOR flash memory experimental chip for 4-b/cell digital storage'. Proc. 28th European Solid-State Circuits Conf., September 2002, pp. 131–134
25 Chen, Y., and Parhi, K.K.: 'Area efficient parallel decoder architecture for long BCH codes'. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, May 2004, pp. V-73–V-76
26 Burton, H.O.: 'Inversionless decoding of binary BCH codes', *IEEE Trans. Inf. Theory*, 1971, **17**, (4), pp. 464–466
27 Micheloni, R., *et al.*: 'A 4 Gb 2b/cell NAND flash memory with embedded 5b BCH ECC for 36 MB/s system read throughput'. IEEE Int. Solid-State Circuits Conf., February 2006, pp. 497–506