Journal of Electronic Testing (2023) 39:487–500
https://doi.org/10.1007/s10836-023-06075-6

E^3C Techniques for Protecting NAND Flash Memories

Shyue-Kung Lu[1] · **Zeng-Long Tsai**[1]

Received: 2 January 2023 / Accepted: 2 June 2023 / Published online: 1 July 2023

Abstract
Due to the rapid technology scaling and increasing program/erase cycles, the *raw bit error rate* (RBER) in NAND flash memory keeps increasing rapidly. This dilemma seriously incurs reliability issues. The traditional *error correction codes* (ECCs) with stronger protection capability are usually equipped for all flash pages as a solution to maintain the mandatory yield and reliability levels. However, the growth of RBER is exponentially proportional to the number of induced P/E cycles and the distributions of errors are usually uneven. Therefore, the conventional *uniform* ECC protection based on the worst scenario of fault distributions might incur unnecessary hardware and latency overhead. Moreover, the overlong ECC check bits might be stored in two different flash pages. Therefore, two flash read/program operations are required for reading/programming a codeword. To cure these drawbacks, the *ECC Caching (E^3C)* techniques are proposed in this paper. The main idea is to upgrade the ECC protection levels for flash pages when their *correction slack* is below the specified threshold. An ECC Cache containing the ECC CAM and the ECC SRAM is used for accessing and storing extra check bits, respectively. The corresponding repair flow and hardware architecture are also proposed. According to simulation results, we can enhance the reliability and yield of flash memories significantly with negligible hardware cost.

Keywords NAND flash memory · Reliability · Yield · Fault tolerance · Repair

1 Introduction

NAND flash memory is the most prevalent non-volatile memory widely used in modern consumer electronic products, such as personal digital assistants (PDAs), laptop computers, digital audio players, and data centers. In the past decades, the advanced fabrication technologies have kept increasing the capacity of NAND flash memories and more bits can be stored in a flash memory cell. Several types of flash cells have been developed, such as a *single-level cell (SLC)*, a *multi-level cell (MLC),* and a *triple-level cell (TLC)* [3, 20, 24] The scaling trend not only significantly increase the storage density and thus reduce the bit cost. This trend is expected to continue in the future. However, a smaller cell incurs a lower noise margin for each threshold voltage level and then cause serious endurance and reliability issues [2, 4, 8, 18, 22].

For example, about 10^5 program-erase (P/E) cycles are available for a 34 nm SLC. The number of P/E cycles is declined to about 3,000 times for a 16 nm MLC [4]. This limited number of P/E cycles means that flash cells cannot reliably store data after running out of the limited P/E cycles. Furthermore, there are also data retention issues, disturbances, and permanent faults that may exist in flash memory. If a flash memory cell cannot store a data bit for the specified retention time, then the *data retention fault* (DRF) occurs in this cell. Previous researches have shown that most flash memories experience a sharp increase in raw bit error rate after a number of P/E cycles [4, 22]. In fact, the RBER due to data retention faults is exponentially proportionally to the number of P/E cycles. The data retention faults are considered as the most dominant faults of bit errors [4]. Moreover, as the bit density of flash cells grows, the RBER also increases significantly because similar voltage levels are assigned to consecutive logic states [4].

Besides the endurance and data retention issues that may incur reliability threats, there are also a variety of permanent faults which will impact the fabrication yield. The most likely functional faults in flash memory is the disturb faults. Traditional disturb fault models include *program*

Responsible Editor: M. Taouil

✉ Shyue-Kung Lu
sklu@mail.ntust.edu.tw

[1] National Taiwan University of Science and Technology, Taipei, Taiwan

disturbance fault (PDF), *erase disturbance fault* (EDF), and *read disturbance fault* (RDF) [8, 22]. PDF fault can be further categorized into the *WPDF* (Wordline PDF) and the *BPDF* (Bitline PDF). EDF also includes the *WEDF* (Wordline EDF) and the *BEDF* (Bitline EDF). RDF includes the *read program disturbance faults* (RPDF) and the *read erase disturbance faults* (REDF). As the count of P/E cycles increases, effects of these three noises also aggravate.

There are also March-like test algorithms [27] proposed for testing such faults. According to the foundry statistics, disturb faults and data retention faults appear more frequent than other fault types [27]. The occurrence probabilities of these faults also depends on the number of P/E cycles. As the P/E cycles increase, the incurred number of disturbance and data retention faults also significantly increase. It is difficult to correct these big amount of RBER using current fault-tolerant techniques with acceptable hardware overhead and performance penalties.

To conquer the reliability and yield challenges, three types of fault-tolerant techniques can be applied. The first one is the *fault avoidance* technique which encodes the data bits to be programmed (increasing the number of 1's) such that flash cells can stay at more reliable states [10, 26]. The second type is the *hardware redundancy-based* technique which uses spare flash blocks and/or columns to bypass the detected faulty flash cells [6, 9, 13, 15, 16]. The most popularly used technique is the *built-in self-repair* (BISR). The conventional BISR consists of a *built-in self-test* (BIST) module, a *built-in redundancy analysis* (BIRA) module and an *on-chip repair* module. The BIST module perform the adopted test algorithms for testing the flash array. When faulty cells are detected by the BIST operations, the BIRA module tries to run redundancy analysis algorithms for optimal or near optimal spare allocation. The *on-chip repair* module practically conduct the reconfiguration or remapping for bypassing the faulty memory cells. The main drawback of BISR is that the hardware overhead and some timing penalty should be compromised. Moreover, as the RBER increases significantly with the increasing P/E cycles, the huge faulty cells cannot be repaired with affordable hardware cost and timing impacts [14, 16].

The last type of fault-tolerant techniques is by using the error correction codes. The most popularly used ECCs include the Hamming code [11], the Bose-Chaudhuri-Hocquenghem (BCH) [7, 23] code, and the low-density parity code (LDPC) [17, 25, 29]. ECC detects and corrects a pre-defined number of faulty cells by adding parity redundancy. The covered faults include permanent faults, endurance- and disturbance-induced faulty cells, and data retention faults. However, the number of faulty bits that can be corrected by the adopted ECC schemes is usually *pre-defined* in the design stage. If the number of faulty bits exceeds the correction capability, these faulty bits cannot be properly corrected. Therefore, the bit error rate after ECC correction will increase with the number of P/E cycles and cannot meet the mandatory reliability specifications.

As the RBER increases, we can simply equip *uniform* protection capabilities for all codewords based on the worst case fault situations to achieve the required reliability requirement. However, the equipped stronger protection capabilities usually incur significant hardware overhead and encoding/decoding latency. There are also adaptive ECC techniques proposed [1, 5, 12, 19, 28, 30] for solving the drawbacks of uniform protection. The main ideas of these previous works include adaptively managing the length of the codeword, adaptively increasing the correction capability or scrambling faulty cells into different codewords so that the number of faulty bits in each codeword does not exceed the limit for ECC protection [19]. These studies achieve a significant increase in yield and reliability at a reasonable cost.

However, the conventional adaptive ECC techniques still has three main shortcomings which should be conquered. First, the data words and their extra check bits should be stored in two different flash pages. Therefore, additional read/program operations are required for flash page accessing and the performance will be significantly degraded. In other words, we should use two read operations to get the data words and the extra check bits and then combine them for decoding. Similarly, two program operations are required to program the data words and the extra check bits.

Second, due to the frequent programing operations for flash pages storing the extra check bits, it is very possible to run out of the limited P/E cycles and leave these flash pages unusable. Third, the management of protection levels by using previous adaptive ECC schemes are basically based on the elapsed P/E cycles, which cannot reflect the real faulty status of codewords.

To cure these drawbacks, *ECC Caching* (E^3C) techniques are proposed in this paper. For each flash page, we provide N protection levels ($PL_0 - PL_{N-1}$) based on the *correction slack (CS)* defined as the difference between the protection capability of the adopted ECC and the number of faulty bits which have been corrected for a codeword. PL_0 (PL_{N-1}) has the weakest (strongest) protection capability. PL_0 is the default protection level. Therefore, the *check bits* (*CBs*) of PL_0 are stored in the same flash pages with their corresponding data words. Alternately, a specific flash memory space is used for storing the *extra check bits* (*ECBs*) when the protection capabilities of flash pages are upgraded above PL_0. To ease the descriptions of the E^3C techniques, we assume $N=4$ in this manuscript.

Similar to the retrieving of the *page mapping table* (*PMT*), the *ECB*s are retrieved from the flash memory array and stored in the equipped *ECC Cache* which contains an *ECC CAM* and an *ECC SRAM*. The ECC CAM stores the physical page numbers which have been upgraded to higher

protection levels. The outputs of the ECC CAM is used as the addresses for accessing ECBs stored in the ECC SRAM. Owing to the accessing mechanisms of the ECC Cache, we do not have to access the flash memory array twice and thus the performance can be greatly improved. Moreover, the wear out of the flash pages for storing *ECBs* can also be greatly released. During power off, the *ECBs* in the ECC Cache are then written back to the assigned flash memory space for future usage. The corresponding hardware architecture and repair flow are also proposed. According to simulation results, the proposed E^3C techniques can improve reliability and yield of flash memories significantly.

The main contributions of this work include:

1. Based on the real fault distributions of flash pages, ECC caching (E^3C) techniques are proposed for adaptively correcting faulty bits in flash pages.
2. The concept of the correction slack is first used to determine if a flash page should be upgraded to a higher protection level. It can reflect the real faulty status of codewords.
3. Read and program operation flows are proposed for implementing the E^3C techniques.
4. The architectures of the ECC CAM and the ECC SRAM are proposed for accessing and storing extra check bits, respectively.
5. Hardware architectures for implementing the proposed E^3C techniques and a simulator are developed to evaluate the repair rate, hardware overhead, yield, and reliability.

The proposed techniques can be applied to SLC, MLC, TLC, or QLC flash memories since each flash page can be encoded with the proposed E^3C techniques. The rest of this paper is organized as follows. The fundamentals of flash memory are presented in Section 2. The basic concepts of the proposed E^3C techniques are described in Sect. 3. Novel read and program flows are presented in Section 4. The corresponding hardware architectures are presented in Section 5. Simulation results are provided in Section 6. Finally, some conclusions are given in Section 7.

2 Preliminaries of Flash Memory

The typical cross-sectional view of a floating-gate flash memory cell is shown in Fig. 1. A floating gate is inserted between the control gate (CG) and the substrate (B). Depending on the amount of charges stored on the floating gate, the stored logic values can be discriminated by using a current sense amplifier. Owing to the rapid progress of process technologies, a flash memory cell can store a single bit (SLC), two bits (MLC), or three bits (TLC) of information.

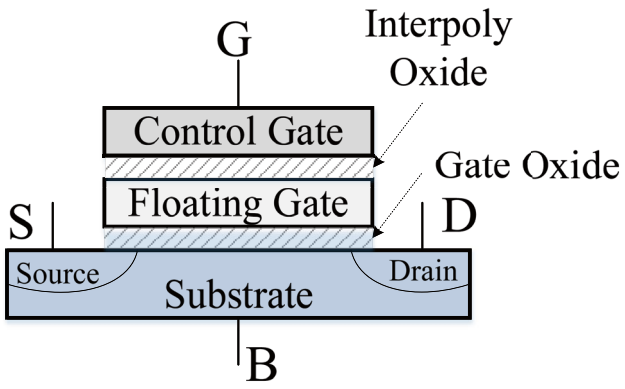

Fig. 1 The cross-sectional view of a flash cell

Flash cells which can stored 4 bits (QLC) are also recently reported [3].

For a flash cell, there are three basic operations—*program*, *read*, and *erase* operations. By using the *channel hot-electron* (CHE) injection and the Fowler-Norheim (FN) tunneling effects, charges can be injected to the floating gate. For SLC, the value of the stored bit is then logic 0. If the charges stored on the floating gate are extracted by applying a high voltage at the source and the induced FN tunneling effect, the logic value of the cell becomes logic 1. The read operation applies a read voltage which can distinguish the stored states at the control gate. The logic value is determined by the sensed current through the peripheral sensing circuitries.

Figure 2 shows the basic architecture of a typical $P \times Q$ flash block which contains P rows and Q columns. There are approximately 32 K – 64 K cells in each row controlled by the same wordline ($WL_0 - WL_{P-1}$). Each row in the flash block form a flash page if the SLC cells are used in the array. The cells in each column are chained in series to form the bit line BL_i, $0 \le i \le Q-1$, and is called a *string*. The basic unit for the erase operation is a flash block which consists of thousands of flash pages. Alternately, the basic unit for the read and program operations is a flash page. The detailed operations of the read, program, and the erase operations for the NAND flash array can be found in [3].

The architecture of a flash memory equipped with ECC is shown in Fig. 3. Since the I/O bus is much narrower than the page size, a *page buffer* is inserted between the ECC module and the flash memory array to temporarily store the data words and check bits. The ECC module is used for encoding and decoding based on the the adopted ECC codes. Before the data words from the I/O bus entering the flash memory array, data words should be encoded by the ECC module to sequentially generate codewords and then the generated codewords are stored into the page buffer. When the page buffer is full of codewords, the entire contents of the page

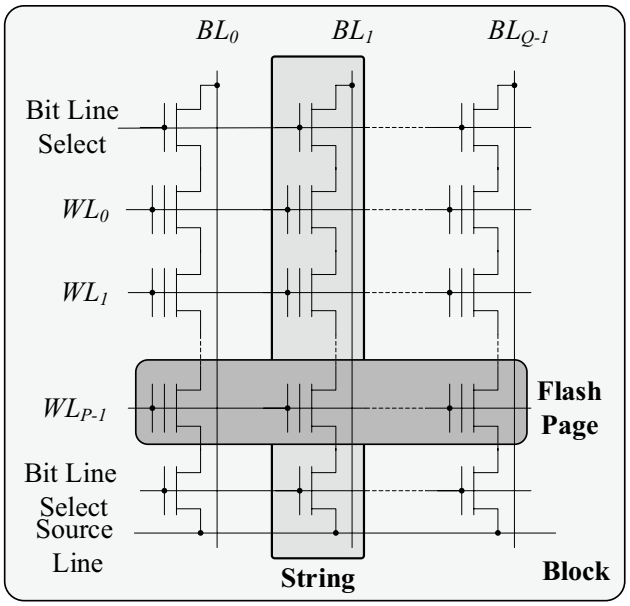

Fig. 2 The architecture of a flash block

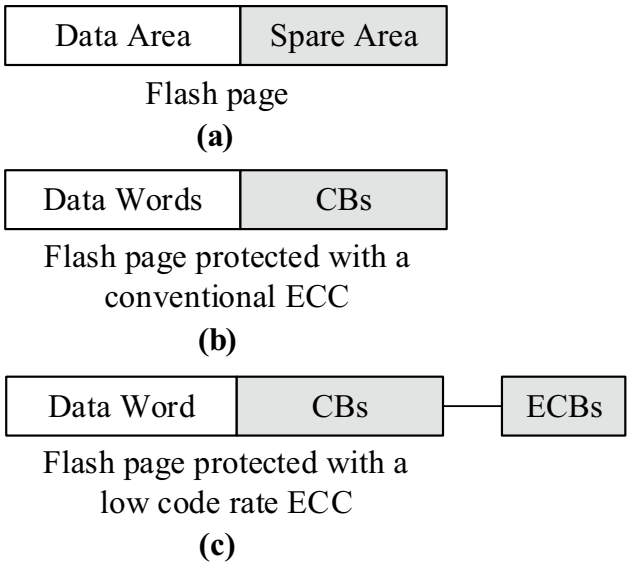

Fig. 4 The data layout of a flash page

buffer are programmed into the flash page simultaneously. To read a flash page, all codewords of an entire flash page are first read out and stored into the page buffer. They are then decoded by the ECC module sequentially and then sent out to the I/O bus for usage.

The data layout of a typical flash page is shown in Fig. 4(a). It can be seen that the flash page is partitioned into the *data area* and the *spare area* and their sizes are predefined by the flash memory designers. If a conventional uniform ECC technique is equipped, the *CBs* of the data words are stored in the spare area as shown in Fig. 4(b). The *code rate* of an ECC is defined as the ratio between the length of the data word and the codeword length (data word + check bits). The size of the spare area might be limited and cause constraints to the effective ECC code rate and then the error correction capability. If we want to achieve sufficient and complicated reliability requirements for flash memories, we have to boost the error correction capability of the adopted ECC. That is, we can adopt a low code rate ECC to increase the protection strength. However, the low code rate ECC might incur more check bits which cannot accommodate in the spare area as shown in Fig. 4(c). The exceeding part of the *CBs* is called the *ECBs*.

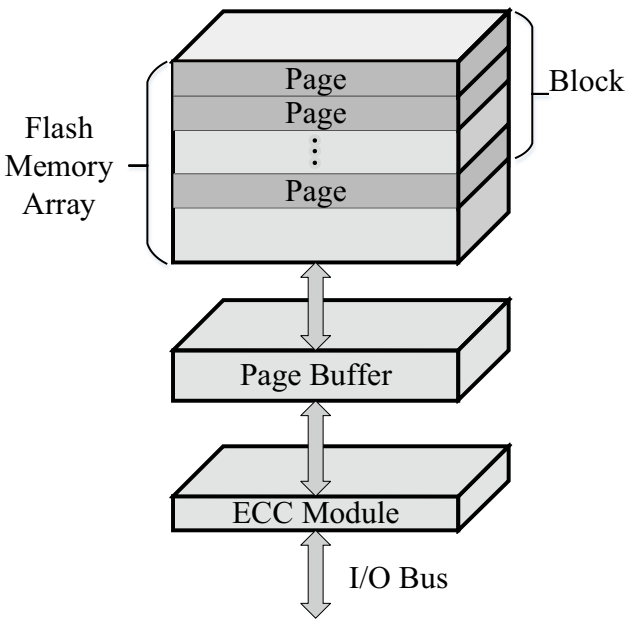

Fig. 3 Architecture of a flash memory equipped with ECC

The locations for storing *ECBs* have significant impacts on the performance and lifetime of flash memories. The conventional ECC techniques store data words and *ECBs* in different flash pages. Therefore, two read or program operations are required when read or program a flash page, respectively. It is evident that the performance will be seriously degraded. Moreover, the lifetime of flash pages for storing *ECBs* will also be shortened since these pages should be programmed more frequently.

An intuitive method to cure this problem is to extend the spare area of each flash page for accommodating ECBs. However, if we allocate sufficient spare area based on the worst case scenario, the incurred hardware cost will be intolerable. There are two categories of solutions for managing the *ECBs*—the *coupling schemes* and the *decoupling schemes* [30]. Coupling schemes try to store data words, *CBs*, and *ECBs* as a whole. For example, one possible way is by shrinking the area for storing data words for each flash

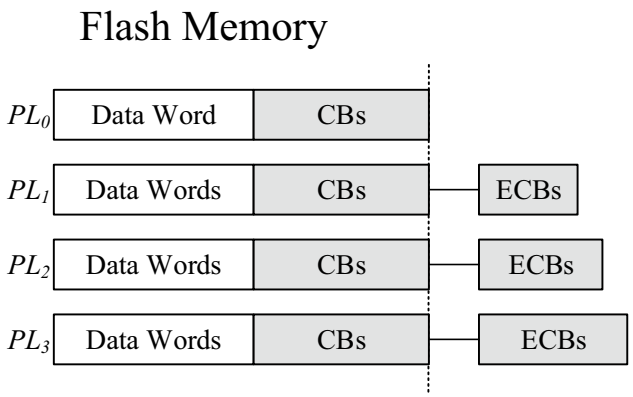

Fig. 5 An example of the E^3C technique

page. Alternately, decoupling schemes store the *ECBs* separately. Unfortunately, the decoupling schemes suffer from significant read and write performance degradation.

3 Basic Concept of the E^3C Techniques

For the proposed ECC caching *(E^3C)* techniques, *N* protection levels ($PL_0 - PL_{N-1}$) are adaptively provided for flash pages. PL_0 is the default protection level as shown in Fig. 4(b). For protection levels $PL_1 - PL_{N-1}$, *ECBs* are required to achieve stronger ECC protection. The higher the protection level, the longer the *ECBs*. The number of faulty bits can be corrected by PL_i is denoted as t_i, $0 \le i \le N-1$. When a current protection level is upgraded, extra t_e faulty bits can be corrected. In other words, $t_i = t_{i-1} + t_e$, $1 \le i \le N-1$. The E^3C architecture for a flash page with four protection levels ($N=4$) is shown in Fig. 5.

Similar to the decoupling schemes, specific flash pages are reserved and used for storing these *ECBs*. However, in order to maintain the read performance and reduce the write amplification problem incurred in the conventional decoupling schemes, an ECC Cache is used for temporarily storing *ECBs* retrieved from the reserved flash pages during power on. The ECC Cache contains two modules—the ECC SRAM module and the ECC CAM (Content-addressable memory) module. The ECC SRAM module contains *N—1* SRAMs ($SRAM_1 - SRAM_{N-1}$) which provide storage space for temporarily storing the *ECBs* of PL_i, $1 \le i \le N-1$. Similarly. ECC CAM module includes $N-1$ CAM modules ($CAM_1 - CAM_{N-1}$) which store the mapping information for accessing the corresponding SRAMs. We do not have to equip SRAM and CAM modules for PL_0 since the *CB*s of PL_0 should be stored with their corresponding data words in the same flash pages and can be accessed simultaneously.

An example flash memory equipped with two protection levels ($N=2$) is shown in Fig. 6. The ECC Cache contains only one SRAM module ($SRAM_1$) and one CAM module (CAM_1), which has r_1 entries, respectively. That is, r_1 flash pages are allowed to be upgraded to protection level PL_1. The CAM module mainly stores the *PPN* of flash pages which have been upgraded to PL_1. In this figure, *PPN* and

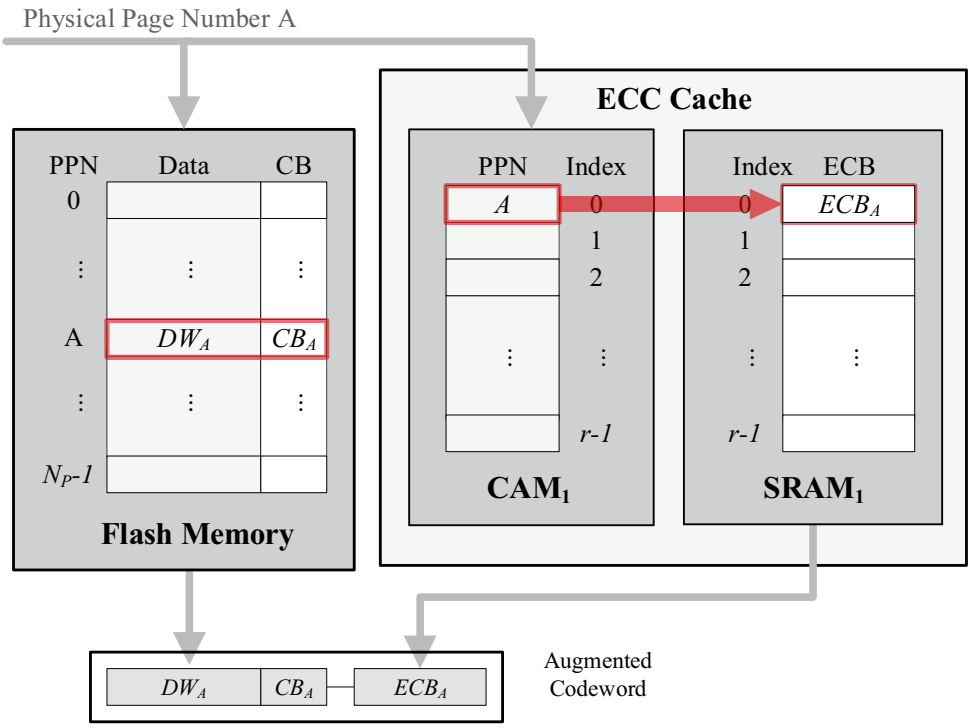

Fig. 6 An example for accessing the ECBs stored in $SRAM_1$

N_P denote the physical page number and the total number of pages in the flash memory array, respectively. If we want to access the physical page number A generated by the *flash translation layer* (FTL) based on the *page mapping table* (PMT), the physical page number A is sent to the flash memory array for accessing the data word A (DW_A) and the default incorporated PL_0 check bits (CB_A). The physical page number A is also sent to the CAM_1 for comparison purposes. Since the physical page number A has been stored in CAM_1, it means that we have already equipped PL_1 for physical page number A. Therefore, the generated match signal from the ECC CAM activates the read operation for the extra check bits (ECB_A) stored in $SRAM_1$. After the accessing operations, we can concatenate the data word DW_A, CB_A, and ECB_A to form the augmented codeword.

In this work, we assume that the BCH code is adopted for protecting NAND flash memories. Let H denote the parity-check matrix of the BCH code and the elements of H are in $GF(2^m)$. The maximum codeword length $(DW + CB)$ is $2^m - 1$. Moreover, the length of the check bits should be less than tm, where t denotes the number of faulty bits which can be corrected by the adopted BCH code. For example, if the length of DW is 340 bits and we want to construct the BCH code with $t = 20$ based on $GF(2^9)$. The maximum codeword length is $2^9 - 1 = 511$ bits. The length of the CBs can be 171 since it is less than mt which is 180. If we want to enhance the protection capability (increase the value of t), similar procedures can be used to evaluate the number of CBs. After enhancing the protection capability, the number of ECBs than can be determined.

The augmented codeword is then sent to the ECC decoder for decoding to generate the original data word. In this example, we only provide two protection levels (PL_0 and PL_1). Therefore, only $SRAM_1$ and CAM_1 should be added for caching *ECBs*. As shown in Fig. 7, if N protection levels are provided, the ECC SRAM module contains N-1 SRAMs ($SRAM_1$ – $SRAM_{N-1}$). For $SRAM_i$, r_i entries are provide for storing *ECBs*. That is, r_i physical pages are allowed to be upgraded to PL_i, $1 \le i \le N-1$. In this figure, $ECB_{i,j}$ denotes the extra check bits stored in the j^{th} entry of $SRAM_i$, $1 \le i \le N-1$, $0 \le j \le r_i-1$.

The CAM module CAM_i is used for storing the PPNs of flash pages which have been upgraded to protection level PL_i, $1 \le i \le N-1$. The comparison results can determine if the protection level of a physical flash page has been upgraded. The architecture of the ECC CAM module is shown in Fig. 8 where N protection levels are provided. There are $N-1$ CAMs (CAM_1 – CAM_{N-1}) in the ECC CAM module. Similarly, there are r_i entries in CAM_i. Each CAM entry contains two fields $PPN_{i,j}$ and $V_{i,j}$,which store the physical page number and the *valid* bit, respectively, $1 \le i \le N-1$, $0 \le j \le r_i-1$.

It should be noted that the flash memory cannot update data at the same place. If we upgrade the protection level during the read operation, the upgrading can only be feasible during the next program operation. All the read operations before the next program should still use the original protection level for decoding. Therefore, even we store the *PPN* of the flash page to be upgraded in an entry of the higher CAM module, the valid bit should be reset to 0 until the next program operation occurs.

4 Repair Flow of the E^3C Techniques

The read operation flow of the proposed E^3C technique is shown in Fig. 9. When a read request is received, the *PPN* of the requested page is retrieved from the *PMT* in the FTL first. The *PPN* is then sent to the ECC CAM module to check if the PPN of the flash page to be read has been equipped with a higher protection level by comparing the *PPN* with all the entries in the CAM modules. If a match occur and the valid bit is 1, the corresponding *ECB* entry in the corresponding ECC SRAM will be read out and sent

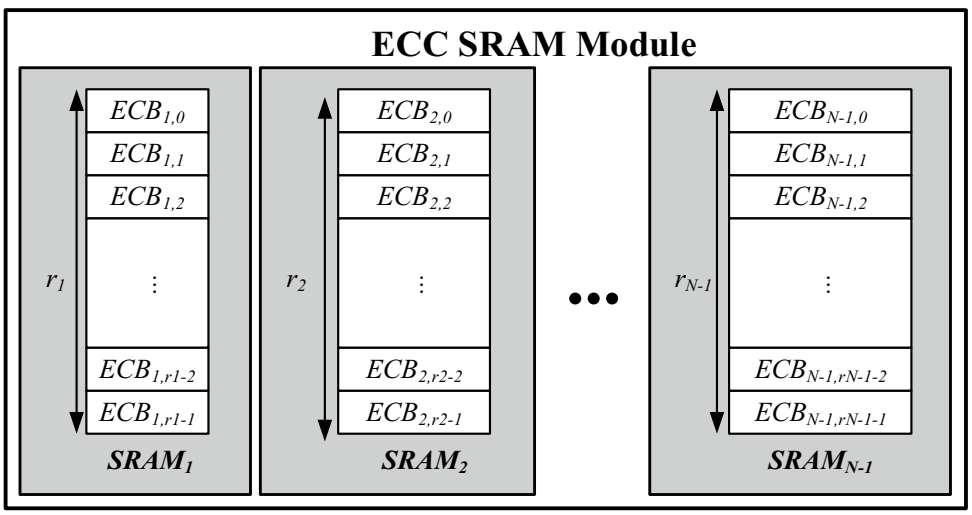

Fig. 7 Architecture of the ECC SRAM module

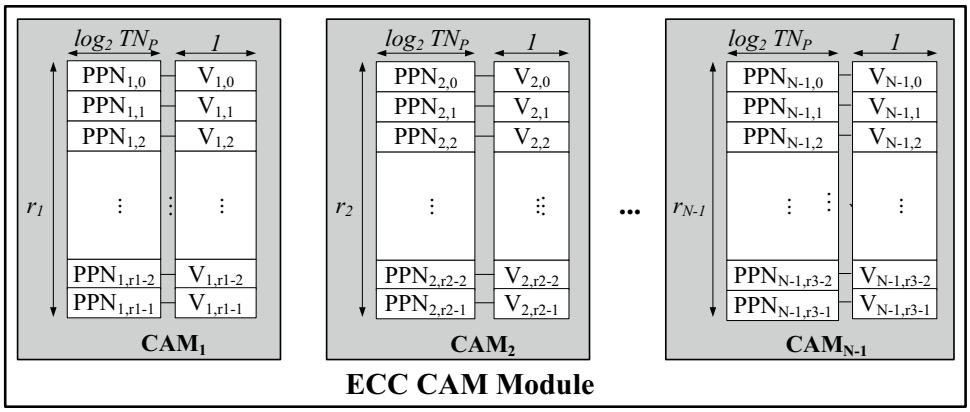

Fig. 8 Architecture of the ECC CAM

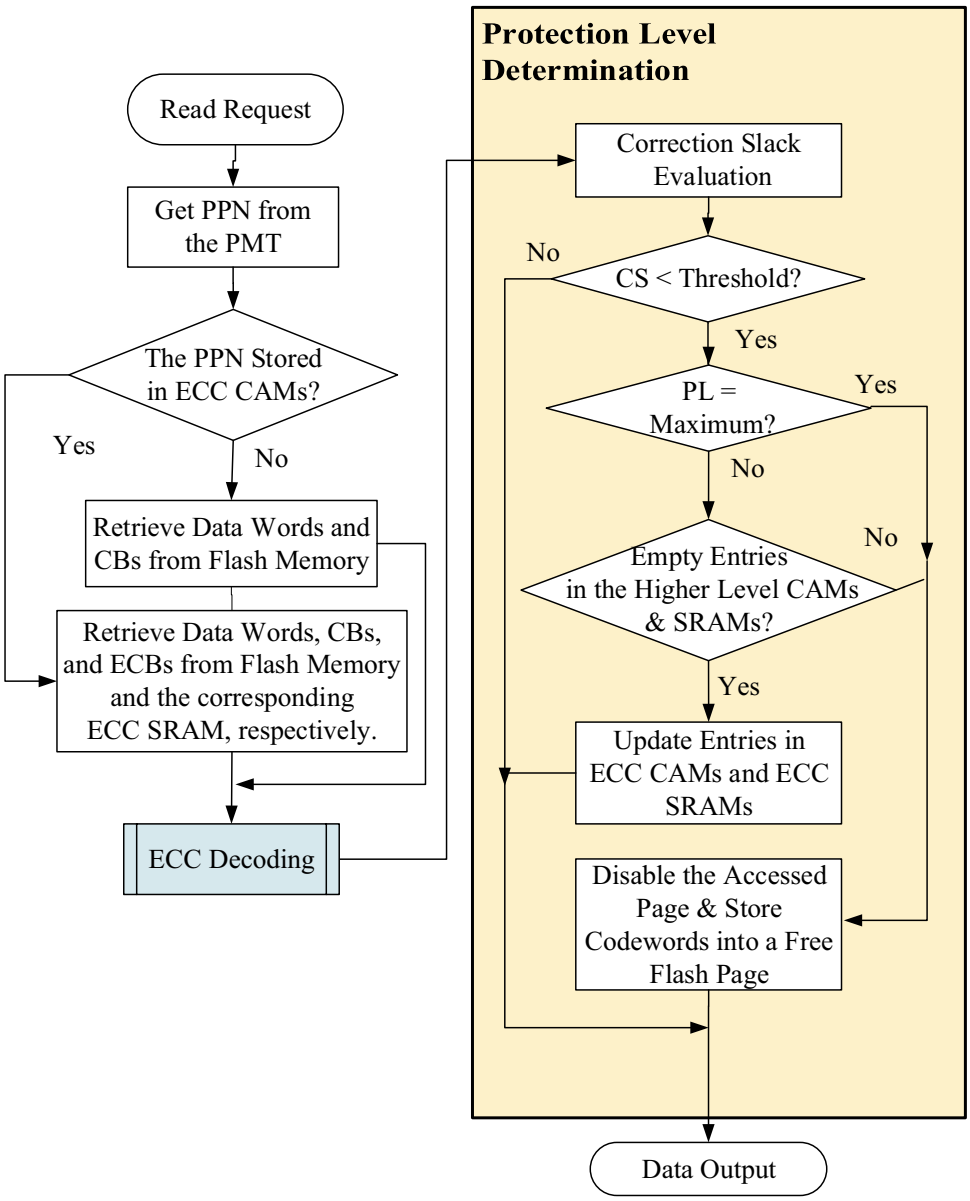

Fig. 9 Read operation flow

to the ECC decoder for decoding. If there are no any match entries in the ECC CAM, we can retrieve the data words and its corresponding *CBs* from the flash memory and send to the decoder for decoding directly.

After the ECC decoding, we then enter the *protection level determination (PLD)* phase. The *correction slack (CS) evaluation* operations are used to determine the correction slack by subtracting the number of erroneous bits in the faulty codeword from the correction capability of the current ECC protection level. It is evident that the number of faulty bits can be determined by the decoder circuitries if it does not exceed the protection capability. If the correction slack is lower than the pre-defined threshold value, it means that the accessed flash page is not reliable for correcting more faulty bits. Therefore, if we have not run out of all protection levels (PL = Maximum), we can upgrade the protection level for this flash page. An available *ECB* entry in the higher level SRAM module is allocated for storing extra check bits for the upgraded protection level. Similarly, the *PPN* value is also stored in the *PPN* field of an empty entry in the higher level CAM module. The valid flag *V* of the new allocated CAM entry should remain 0 until the next program operation is applied to this physical page.

Alternately, if the PL of the accessed flash page has already in the highest protection level, we do not have the opportunity to upgrade its protection level. Therefore, the accessed physical flash page should be disabled and its contents are moved to a fault-free flash page. If the currently accessed flash page still has enough correction slack, we do not have to upgrade the protection level and the read out contents can be outputted directly.

The program operation flow is shown in Fig. 10. When a program request is received, the *PPN* of the requested page is retrieved from the *PMT* in the FTL and sent to the ECC CAM module to check if a higher ECC protection level has been equipped for the requested page. If a match occurs, the match signal of ECC CAM module is used to determine the required ECC protection level for the page. The *ECB*s of the corresponding protection level will be stored into the corresponding *ECB* entries in the ECC SRAM. If the valid bit of the requested page is 0, it will then be set to one. It means the entry of this protection level is activated. Meanwhile, the CAM entry in the original CAM module will be reset to release the space in the ECC SRAM.

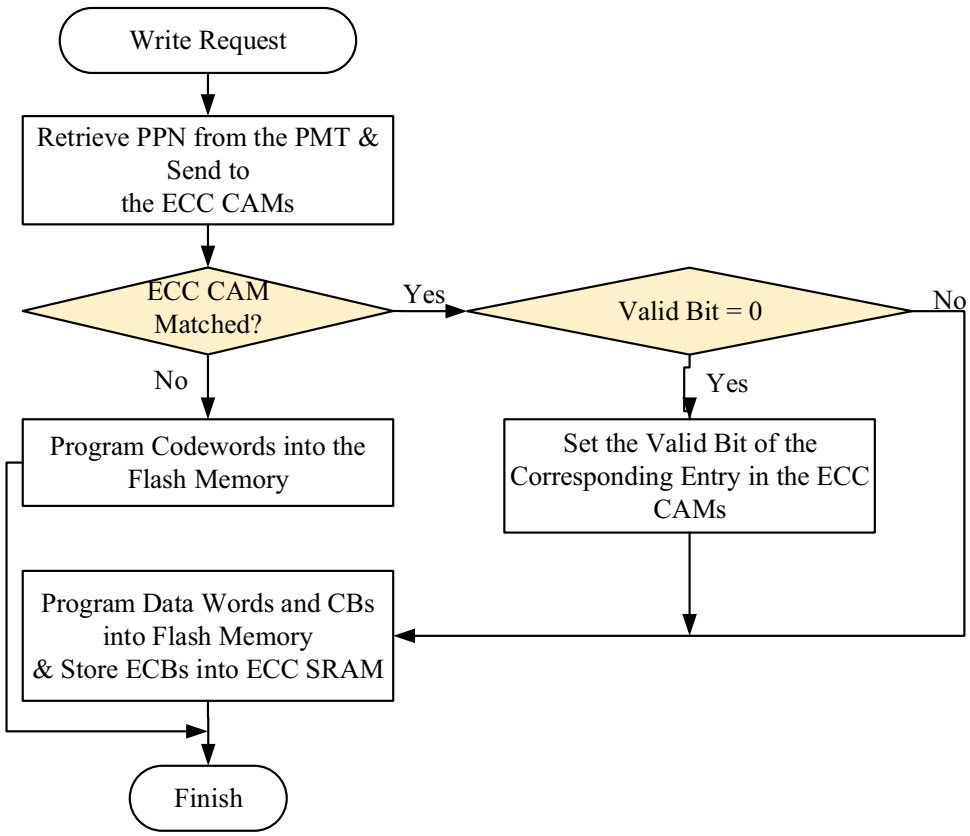

Fig. 10 Program operation flow

5 Hardware Architecture

The hardware architecture which can perform the proposed E^3C techniques is shown in Fig. 11. It mainly consists of the *BIST* module, the BCH encoder/decoder, the ECC CAM module, the ECC SRAM module, and the control logic module. The *BIST* module executes the adopted March test algorithms [27] to test the flash memory array. If there are faults detected when testing the memory array, the conventional hard repair techniques [15] can be activated to repair these faulty cells (not shown in this figure). The roles of the ECC CAM module and the ECC SRAM modules are described in Section 3. The control logic module is used to orchestrate the read and program flows. We assume that the BCH error correction code is adopted. Therefore, the BCH encoder and decoder are also included in the hardware architecture.

6 Simulation Results

6.1 Hardware Overhead

The hardware overhead models of the proposed techniques are derived for the estimation. We use the number of extra transistors required for the estimation. Let N_B and N_P denote the number of blocks in the flash memory and the number of pages in a block, respectively. We assume that each page contains S codewords and the wordlength is WL. The number of transistor count (TC_M) of the flash memory array can be express as:

$$TCM = NB \times NP \times S \times WL.$$

If we use 6-transistor SRAM cells to implement the ECC SRAM module, the transistor count of the ECC SRAM is:

$$TC_{SRAM} = 6 \times \sum_{i=1}^{N-1} \sum_{j=0}^{r_i-1} ECB(i,j).$$

During power off, the contents stored in the ECC SRAM module should be restored into the flash memory for future usage. Therefore, the extra required number of flash cells (and transistor count for SLC) TC_{ECB} for storing the extra check bits is:

$$TC_{ECB} = \sum_{i=1}^{N-1} (i \times t_e) \times r_i.$$

If 10-transistor CAM cells are used to implement the CAM modules within the ECC CAM module and each CAM entry contains $\log(N_B \times N_P)$-bit *PPN* field and 1-bit valid flag, the transistor count of the ECC CAM is:

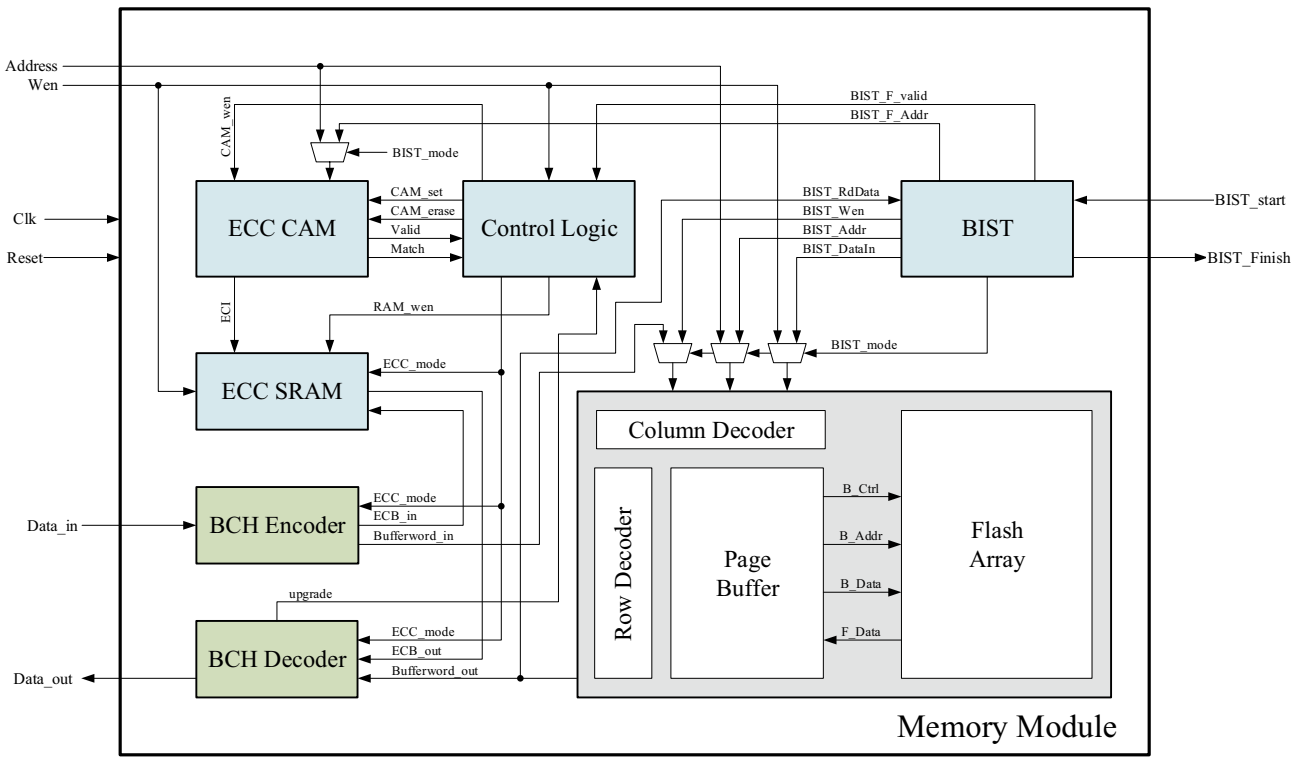

Fig. 11 Hardware architecture for implementing the E^3C technique

Table 1 Hardware overhead of a 1–Gb flash memory

r_1	r_2	r_3	HO
16	8	4	0.0520%
64	8	4	0.1092%
256	16	8	0.3711%
1024	64	32	1.4854%

$$TC_{CAM} = 10 \times \log_2(N_B \times N_P) \times \sum_{i=1}^{N-1} r_i.$$

Based on the transistor counts described above, the hardware overhead (HO) of the proposed techniques can be expressed as:

$$HO = \frac{TC_{SRAM} + TC_{CAM} + TC_{ECB}}{TC_M}.$$

Table 1 shows the hardware overhead of a 1-Gb flash memory. The default protection level PL_0 has $t_0=4$ and $t_e=5$. We assume that the flash memory contains 128 blocks and each block contains 64 pages. The size of each flash page is 16 KB. Moreover, 4 protection levels ($N=4$) are used in this example. From this table we can see that the hardware overhead is very low. Moreover, as the size of the flash memory increases, the HO can be further reduced. For the estimation of hardware overhead, we use the number of extra transistors required for the evaluation. Therefore, the improvements of repair rate and reliability are independent of the used architecture, technology, and design methodology.

Based on the models for evaluating hardware overhead, it is very easy to derive HO if we change the assumptions. Based on our evaluations, the hardware overhead is also almost negligible.

6.2 Reliability Analysis

Let λ denote the failure rate of a flash cell. The reliability of a cell then can be expressed as

$$R_C(t) = e^{-\lambda t}.$$

The reliability of a codeword with t-bit correction capability can be expressed as

$$R_W(t) = \sum_{i=0}^{t} \binom{WL}{i} \left(e^{-\lambda t}\right)^{WL-i} \left(1 - e^{-\lambda t}\right)^{i}.$$

Thus, the reliability of a flash page can be derived as

$$R_P(t) = \left[R_W(t)\right]^S.$$

For a flash memory with N_B blocks and N_P pages per block, the reliabilities of a block ($R_B(t)$) and the flash memory ($R_M(t)$) can be expressed as

$$R_B(t) = [R_P(t)]^{N_P},$$

$$R_M(t) = [R_B(t)]^{N_B}.$$

For the proposed E^3C technique, the reliability of a codeword with protection level PL_i, $0 \le i \le N\text{-}1$ can be expressed as:

$$R_{W\_Ln}(t) = \sum_{j=0}^{t_0 + i \times t_e} \binom{WL}{j} \left(e^{-\lambda t}\right)^{WL-j} \left(1 - e^{-\lambda t}\right)^{j}.$$

Thus, the reliability of a page with PL_i protection can be derived as

$$R_{P\_Li}(t) = \left[R_{W\_Li}(t)\right]^S.$$

As shown in Fig. 8, there are r_i entries in CAM_i, therefore, r_i flash pages can be equipped with PL_i protection, $0 \le i \le N\text{-}1$. The reliability of a flash memory incorporated with the E^3C ($R_{E^3C}(t)$) technique then can be expressed as

$$R_{E^3C}(t) = \prod_{i=0}^{N-1} R_{P\_Li}(t)^{r_i},$$

where

$$r_0 = N_B \times N_P - \sum_{i=1}^{N-1} r_i.$$

The reliabilities of a 512-Mb flash memory with different ECC Cache configurations are shown in Fig. 12. We assume $\lambda = 10^{-9.5}$/hour [21] and four protection levels are provided ($N=4$). The wordlength of data words is 4096 bits. In this figure, ECC_4 denotes that the memory is equipped with

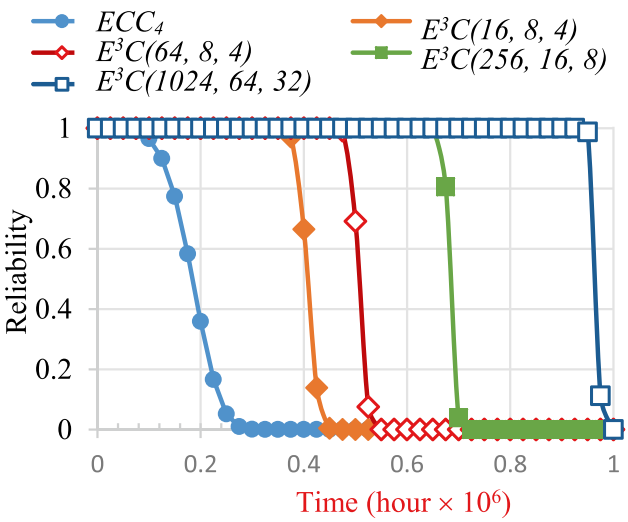

Fig. 12 Reliabilities of a 512-Mb flash memory

uniform 4-bit error correction capability for all pages. For the E^3C techniques, additional 5 bits of correction capability can be provided after upgrading a protection level, i.e., $t_0=4$, $t_e=5$. $E^3C(r_1, r_2, r_3)$ denotes that there are r_1, r_2, and r_3 *CAM* entries in CAM_1, CAM_2, and CAM_3, respectively. The number of entries in the CAM module of each ECC level can be adjusted for the tradeoff between hardware overhead and reliability. The results show that the reliabilities are greatly improved with very small hardware overhead as shown in Table 1.

The reliability evaluation described above is based merely on flash memory cell failures. In a typical flash memory chip, there are some other circuits added such as the BIST/BISR, ECC encoder/decoder, and the control circuit. For the proposed E^3C techniques, we have to further include the ECC CAM and the ECC SRAM modules. Fortunately, the incurred hardware overhead is almost negligible as shown in Table 1. Therefore, the faults in these circuits are not taken into account for the evaluation of reliability. Although these added circuits do have slight impact on the final achieved reliability, however, the reliabilities of the proposed techniques are much higher than that of the conventional uniform ECC technique as shown in Fig. 12. Therefore, if we take the extra circuits into account, the reliability trends are almost the same as shown in this figure.

6.3 Repair Rate

Repair rate is defined as the probability of correcting all faulty cells. A simulator is developed for evaluating the achieved repair rate. In the simulation, we assume that only the uniform ECC techniques and the E^3C techniques are used without incorporating any spares into the flash memory for a more fair comparison. We assume that the injected defects follow the Poisson distribution and the average defect number is 0.6. The injected fault types include *single cell faults* (SCF), *faulty rows* (FR), *faulty columns* (FC), and *cluster faults* (CF) as shown in Fig. 13. Simulation results for a 512-Mb flash memory are shown in Table 2. The number of simulation instances is 50,000. We can see that the proposed E^3C techniques have much higher repair rates than the uniform ECC_4 technique and are very close to the uniform ECC with 19-bit correction capability (ECC_{19}). In other words, we can have higher code rates than by using ECC_{19} and achieve nearly the same repair rates.

6.4 Yield Analysis

Let Y_O denote the original yield of a flash memory without using any fault-tolerant techniques. Let RR_{E^3C} represent the repair rate using the proposed E^3C techniques. The *effective yield* (*EY*) than can be expressed as:

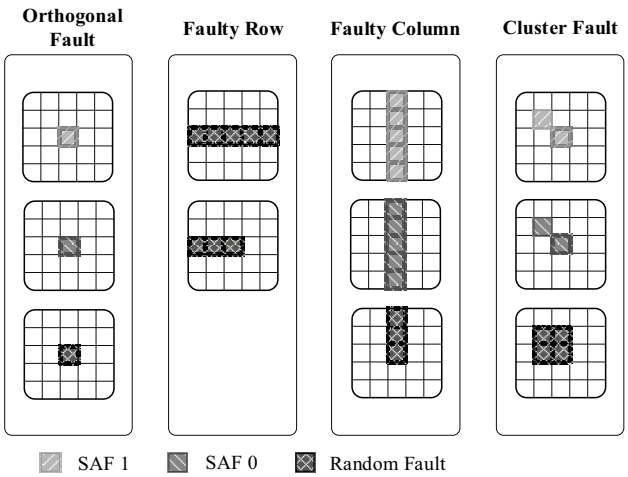

Fig. 13 Injected fault types

$$EY_{E^3C} = Y_0 + \left(1 - Y_0\right) \times RR_{E^3C}.$$

Similarly, EY_{ECC_t} denotes the effective yield if the uniform ECC techniques with t-bit correction capability is used. Therefore, EY_{ECC_t} can be expressed as

$$EY_{ECC_t} = Y_0 + \left(1 - Y_0\right) \times RR_{ECC_t}.$$

We can get the repair rates with the simulator and the results shown in Table 2 to evaluate the *EY* values. The effective yields of a 512-Mb flash memory with $Y_0=0.95$ are shown in Fig. 14. From this figure we can see that E^3C(1024, 64, 32) and E^3C(16, 8, 4) have almost the same effective yield levels as ECC_{19} when the single faulty cell ratio is high. As the single faulty cell ratio decreases, E^3C(1024, 64, 32) and E^3C(16, 8, 4) have slightly less effective yield than ECC_{19}. We can conclude that even we use less ECC resources, the proposed techniques can also maintain sufficient effective yield.

6.5 VLSI Implementation

We use TSMC Artisan 90 nm 1P9M technology to implement the hardware architecture shown in Fig. 11 (the flash memory array is not included). The related materials of this process technology can be easily accessed through the support of *Taiwan Semiconductor Research Institute (TSRI)*. The ECC SRAM and the page buffer are implemented with Artisan memory compiler. The page size is 8 KB. The entries in the CAM module and the SRAM module are 1024、64、32 for PL_1, PL_2, and PL_3, respectively. According to the synthesis results, the operation frequency can achieve up to 200 MHz. The ECC SRAM and ECC CAM occupy 26% and 18% of the total chip area, respectively. If we also consider the large flash memory area, the hardware overhead can be significantly reduced as shown in Table 1. The core

Table 2 Repair rates of a 512-Mb flash memory

Fault Type (%)				Repair Rate					
SCF	FC	FR	CF	ECC_4	ECC_9	ECC_{14}	ECC_{19}	E^3C (16, 8, 4)	E^3C (1024, 64, 32)
80	10	5	5	0.788	0.858	0.921	0.959	0.957	0.959
70	15	10	5	0.606	0.723	0.830	0.917	0.909	0.913
60	15	15	10	0.472	0.604	0.748	0.868	0.861	0.863
50	20	20	10	0.370	0.509	0.676	0.826	0.825	0.826
40	25	25	10	0.296	0.434	0.606	0.803	0.785	0.788
30	25	30	15	0.240	0.373	0.555	0.768	0.752	0.752
20	30	35	15	0.197	0.325	0.501	0.720	0.702	0.719

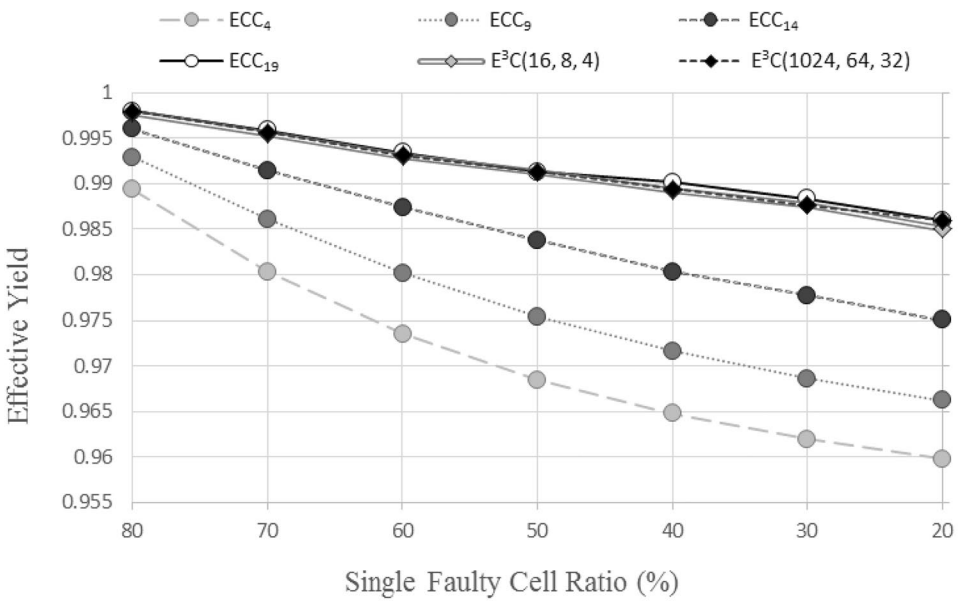

Fig. 14 The effective yields of a 512-Mb flash memory ($Y_0 = 0.95$)

area and the chip area are 2,698,050 μm^2 and 4,788,239 μm^2, respectively. The detailed specifications of the designed chip are shown in Table 3. Besides the internal core circuitries, the chip area also includes the area of I/O pads.

Latency and power consumption evaluations are also important for the proposed techniques. Fortunately, the encoding/decoding latency will not be sacrificed since the highest protection level can be designed with the required protection capability of the uniform ECC. Moreover, the latency of the ECC cache is much lower than program/read the flash memory. The ECC cache does not located on the critical path for accessing the flash memory. Therefore, the original latency can be maintained. For power consumption, the encoding/decoding circuitries will not incur extra power consumption. The reason is the same as for latency evaluation.

Table 3 Specifications of the designed chip

Process	TSMC 90 nm 1P9M
Operation Frequency	200 MHz
Core Voltage	1.0 V
Total Gate Count	865,814 (100%)
Page Buffer Gate Count	175,128 (20%)
ECC Encoder/Decoder Gate Count	255,953 (30%)
ECC CAM Gate Count	226,367 (26%)
ECC SRAM Gate Count	157,797 (18%)
Miscellaneous	50,570 (6%)
Core Area	2,698,050 μm^2
Chip Area	4,788,239 μm^2

7 Conclusion

E^3C techniques are proposed for reliability and yield enhancement of flash memory. Instead of applying a uniform and strong ECC for each flash page, we adaptively increase the protection capability by using the inherent on-line monitoring ability of ECC. In other words, we on-line evaluate the correction slack for each flash page. When the correction slack is below the specified threshold, we then upgrade the protection level until the highest level is reached. Experimental results show that the reliability and repair rate can be improved significantly with negligible hardware overhead. The proposed techniques can also solve the write amplification problem incurred in the conventional adaptive ECC techniques. Moreover, the performance penalty can also be eliminated.

Availability of Data and Materials The datasets supporting the conclusions of this article are included within the article.

Declarations

Conflict of Interest Statement The authors have no relevant financial or non-financial interests to disclose.

References

1. Basak A, Paul S, Park J, Park J, Bhunia S (2013) Reconfigurable ECC for Adaptive Protection of Memory. In Proc Int'l Midwest Symp Circuits and Systems (MWSCAS) pp. 1085–1088
2. Baumann RC (2001) Soft errors in advanced semiconductor devices—Part I: The three radiation sources. IEEE Trans Device Mater Reliab 1(1):17–22
3. Bez R, Camerlenghi E, Modelli A, Visconti A (2003) Introduction to flash memory. Proc IEEE 91(4):489–502
4. Cai Y, Ghose S, Haratshc EF, Luo Y, Mutlu O (2017) Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. Proc IEEE 105(9):1666–1704
5. Chen TH, Hsiao YY, Hsing YT, Wu CW (2009) An Adaptive-Rate Error Correction Scheme for NAND Flash Memory. In Proc VLSI Test Symp pp. 53–58
6. Corcoran PM, Bigioi P, Nanu F (2014) Detection and Repair of Flash-eye in Handheld Devices. In Proc IEEE Int'l Conf on Consumer Electronics (ICCE) pp. 213–216
7. Forney G (1965) On Decoding BCH Codes. IEEE Tran Inf 11(4):549–557
8. Ginez O, Portal JM, Aziza H (2008) Reliability Issues in Flash Memories: An On-line Diagnosis and Repair Scheme for Word Line Drivers. In Proc Int'l Workshop on Mixed-Signals, Sensors, and Systems Test, pp. 1–6
9. Ginez O, Portal JM, Aziza H (2009) An On-Line Testing Scheme for Repairing Purposes in Flash Memories. In Proc IEEE Int'l Symp Des Diagnostics Electronic Circ Syst (DDECS) pp. 213–216
10. Guo J, Chen Z, Wang D, Shao Z, Chen Y (2014) DPA: A Data Pattern Aware Error Prevention Technique for NAND Flash Lifetime Extension. In Proc Asia South Pac Design Autom Conf (ASP-DAC) pp. 592–597
11. Hamming RW (1950) Error detecting and correcting codes. Bell Syst Tech J 29:147–160
12. Hsieh JW, Chen CW, Lin HY (2015) Adaptive ECC Scheme for Hybrid SSD's. IEEE Trans Comp 64(12):3348–3361
13. Hsiao YY, Chen CH, Wu CW (2006) A Built-In Self-Repair Scheme for NOR-type flash memory. In Proc IEEE VLSI Test Symp (VTS), Berkeley pp. 114–119
14. Hsiao YY, Chen CH, Wu CW (2006) A built-in self-repair scheme for NOR-type flash memory. In Proc IEEE VLSI Test Symp (VTS) pp. 114–119
15. Hsiao YY, Chen CH, Wu CW (2010) "Built-In Self-Repair Schemes for Flash Memories. IEEE Trans Comput Aided Des Integr Circuits Syst 29(8):1243–1256
16. Huang CT, Yeh JC, Shih YY, Huang RF, Wu CW (2005) On test and diagnostics of flash memories. In Proc IEEE Asian Test Symp (ATS) pp. 260–265
17. IEEE Computer Society (2019) IEEE Standard for Error Correction Coding of Flash Memory Using Low-Density Parity Check Codes. In IEEE Std 1890-2018, vol., no., pp. 1–51, 28 Feb. 2019. https://doi.org/10.1109/IEEESTD.2019.8654228
18. Kuo W, Chien WTK, Kim T (1998) Reliability, yield, and stress burn-in. Kluwer Academic Publishers, Boston
19. Lu SK, Zhong SX, Hashizume M (2016) Adaptive ECC Techniques for Yield and Reliability Enhancement of Flash Memories. In Proc IEEE Asian Test Symp (ATS) pp. 287–292
20. Micheloni R, Picca M, Amato S, Schwalm H, Scheppler M, Commodaro S (2009) Non-volatile Memories for Removable Media. Proc IEEE 97(1):148–160
21. Mielke N et al (2008) Bit error rate in NAND Flash memories. In Proc IEEE Int'l Reliability Physics Symposium, Phoenix, AZ, USA pp. 9–19
22. Mielke NR, Frickey RE, Kalastirsky I, Quan M, Ustinov D, Vasudevan VJ (2017) Reliability of Solid-State Drives Based NAND Flash Memory. Proc IEEE 105(9):1725–1750
23. Ning S (2018) Advanced Bit Flip Concatenates BCH Code Demonstrates 0.93% Correctable BER and Faster Decoding on (36 864, 32 768) Emerging Memories. IEEE Trans Circuits and Systems I 65(12):4404–4412
24. Park Y, Lee J, Cho SS, Jin G, Jung E (2014) Scaling and Reliability of NAND Flash Devices. In Proc IEEE 52nd Int Reliab Phys Symp (IRPS) pp. 2E.1.1–2E.1.4
25. Tanakamaru S, Yanagihara Y, Takeuchi K (2013) Error-prediction LDPC and Error-recovery Schemes for Highly Reliable Solid-state Drives (SSDs). IEEE J Solid-State Circuits 48(11):2920–2933
26. Wei D, Deng L, Zhang P, Qiao L, Peng XY (2016) "NRC: A Nibble Remapping Coding Strategy for NAND Flash Reliability Extension. IEEE Trans Comput Aided Des Integr Circuits Syst 35(11):1942–1946
27. Yeh JC, Cheng KL, Chou YF, Wu CW (2007) "Flash memory testing and built-in self-diagnosis with march-like test algorithms. IEEE Trans Comput Aided Des Integr Circuits Syst 26(6):1101–1113
28. Yuan L, Liu H, Jia P, Yang Y (2015) Reliability-based ECC System for Adaptive Protection of NAND Flash Memories. In Proc 5th Int'l Conf. Commun Syst Netw Technol (CSNT) pp. 897–902
29. Zambelli C, Cancelliere G, Riguzzi F, Lamma E, Olivo P, Marelli A, Micheloni R (2017) Characterization of TLC 3D-NAND Flash Endurance through Machine Learning for LDPC Code Rate Optimization. In Proc Intt'l Memory Workshop (IMW) pp. 1–4
30. Zhou Y, Wu F, Lu Z, He X, Huang P, Xie C (2018) SCORE: A Novel Scheme to Efficiently Cache Overlong ECCs in NAND Flash Memory. ACM Trans Arch Code Opt n15(4):60:1–60:25

Shyue-Kung Lu received the MS degree in 1991 from National Tsing Hua University, Hsinchu, Taiwan, and PhD degree in 1996 from National Taiwan University, Taipei, Taiwan, both in electrical engineering. Since 1998, he has been with the Department of Electronics Engineering, Fu Jen Catholic University, Taipei, Taiwan, where he is a professor. He also served as the Chair of the EE Department of FJU from 2006 to 2009. From 2009, he joined the Department of Electrical Engineering, National Taiwan University of Science and Technology. His research interests include the areas of VLSI testing and fault-tolerant computing, test and repair techniques for 3D stacked ICs, artificial intelligence (AI) deep learning HW/SW designs, video coding techniques and architectures design. Dr. Lu is a Life Member of the Taiwan IC Design Society and a member of IEEE. He is also the treasurer of IEEE Taipei Section. He received the Best Paper Award of the IEEE 2009 International Conference on Test and Diagnosis (ICTD), the Young Scholar Award of the IEEE 2014 CPMT Symposium, the Best Paper Award of the 2017 IEEE Asian Test Symposium (ATS), the Best Paper Award of the 2017 IEEE CASS Shikoku Chapter, the 2018 IEEE TTTC/JETTA Best Paper Award, and the Student Poster Award of the 2019 IEEE International 3D Systems Integration Conference.

Zeng-Long Tsai received the B.S. and M.S. degrees from National Taiwan University of Science and Technology, Taipei, Taiwan, in 2018 and 2020, both in Electrical Engineering, respectively. His research interests include VLSI design and testing, as well as the testing and self-repair of semiconductor memory.