[Article] FLARES: an aging aware algorithm to autonomously adapt the error correction capability in NAND Flash memories

A

FLARES: an aging aware algorithm to autonomously adapt the error correction capability in NAND Flash memories[1]

STEFANO DI CARLO, SALVATORE GALFANO, MARCO INDACO, PAOLO PRINETTO,
Politecnico di Torino
DAVIDE BERTOZZI, PIERO OLIVO and CRISTIAN ZAMBELLI, Università di Ferrara

With the advent of solid-state storage systems, NAND flash memories are becoming a key storage technology. However, they suffer from serious reliability and endurance issues during the operating lifetime that can be handled by the use of appropriate error correction codes (ECC) in order to reconstruct the information when needed.. Adaptable ECCs may provide the flexibility to avoid worst-case reliability design thus leading to improved performance. However, a way to control such adaptable ECCs strength is required. This paper proposes FLARES, an algorithm able to adapt the ECC correction capability of each page of a flash based on a flash RBER prediction model and on a measurement of the number of errors detected in a given time window.

FLARES has been fully implemented within the YAFFS 2 filesystem under the Linux operating system. This allowed us to perform an extensive set of simulations on a set of standard benchmarks that highlighted the benefit of FLARES on the overall storage subsystem performances.

Categories and Subject Descriptors: B.3.4 [**MEMORY STRUCTURES**]: Reliability, Testing, and Fault-Tolerance—*Error-checking*; B.8.m [**PERFORMANCE AND RELIABILITY**]: Miscellaneous

General Terms: Design, Algorithms, Performance, Reliability

Additional Key Words and Phrases: adaptable ECC, BCH codes, error correcting codes, NAND flash memory

1. INTRODUCTION

Driven by the ever increasing demand for high-performance data storage, NAND flash memory has become one of the fastest growing segments in the global semiconductor industry. Developers have successfully scaled NAND flash to sub-20-nm technology and moved from the single-level cell (SLC) technology, in which each cell is able to store a single bit of information, to the multi-level cell (MLC) technology, able to store more than one bit per cell (e.g., 2-4 bits) [Cai et al. 2013a].

As NAND flash technology scales down and increases the number of levels per cell, system management algorithms need to face serious issues to maintain product reliability, while continuing to address reduced endurance and demand for increased performance [Li and Quader 2013; Micheloni et al. 2010; Ielmini 2009; Jae-Duk et al. 2002; Mincheol et al. 2009; Cooke 2007]. Designers use error correction code (ECC) to guarantee target reliability levels by providing multiple-bit corrections to stored data [Li and Quader 2013]. If data is deteriorated by aging or read/program disturbs, the ECC can correct the errors and ensure access to error-free data. Reed-Solomon (RS) codes [Reed and Solomon 1960] and Bose-Chaudhuri-Hocquenghem (BCH) codes [Bose and Ray-Chaudhuri 1960] are well-known solutions used to improve NAND flash reliability. Nevertheless, choosing the ECC correction capability is one of the key design choices in the development of a NAND flash storage system. It implies to trade-off between reliability and performance. High correction capability guarantees high reliability but, as a drawback, it introduces larger ECC encoding/decoding latency, information overhead, implementation overhead and power overhead. A wrong choice may either underestimate or overestimate the required redundancy, with the risk of missing or overkilling the target failure rate and unnecessarily penalizing the memory performance. Moreover, the reliability of a NAND flash is not constant. It continuously

[1]This research has been partly supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404 and through the vIrtical project under GA 288574.

decreases with time due to the aging effect caused by program and erase operations on floating gate transistors [Chen 2011]. ECC with programmable correction capability are now widely implemented in the same flash controller to adapt the error rate changes over program and erase cycles. This trend is confirmed by an increased number of publications proposing hardware implementations of adaptable BCH and RS codecs for NAND flash that guarantee low hardware overhead compared to worst-case designs that implement a fixed correction capability [Song et al. 2002; Atieno et al. 2006; Chen et al. 2009; Caramia et al. 2010; Cherukuri 2010; Zambelli et al. 2012; Di Carlo et al. 2012; Fabiano et al. 2013]. However, for a realistic application of an adaptable ECC to a NAND flash, a strategy to decide which correction capability to use at run-time is required. This is still an open question that need to be properly explored.

This paper tries to answer this question proposing FLARES, an algorithm able to predict, at run-time, the optimal correction capability for each page of a NAND flash. FLARES performs predictions based on a combination of data obtained from a NAND flash bit error rate (BER) estimation model and real measurements of the number of errors detected in a given time window. The BER estimation model exploited in FLARES considers the combined effect of program erase cycles and retention time. This represents an important improvement when compared with previous publications that in general consider these two contributions in isolation. This prediction model, combined with the use of real BER data collected at run-time, represents a key instrument to implement an effective ECC adaptation strategy in a real flash controller. FLARES is generic enough to work with several types of ECC able to provide programmable error correction capability. This includes well established RS, BCH codes and Low Density Parity Check Codes (LDPC) that are gaining importance for future NAND flash controllers. FLARES only requires that the ECC enables to program the target correction capability, and that it is able to provide information about the number of errors detected and corrected in a codeword. Without loss of generality, this paper considers its application to the adaptable BCH ECC subsystem presented by Zambelli et al. [2012]. FLARES has been first evaluated by a MATLAB® implementation used to validate its accuracy in terms of selection of the most suitble ECC correction capability. Moreover, the FLARES algorithms have been implemented in a real environment by instrumenting the YAFFS 2 (Yet Another Flash File System version 2) filesystem [yaffs.net 2007], one of the most used NAND flash based filesystems. This implementation has been used to perform an extensive simulation campaign based on a set of standard benchmarks that allowed us to clearly evaluate the impact of FLARES on the throughput, power consumption and write amplification of the NAND flash.

The paper is organized as follows. Section 2 introduces the flash memory wear-out model exploited in Section 3 to define the proposed ECC adaptation heuristic. Section 4 introduces the proposed YAFFS 2 implementation and Sections 5 provides results related to the validation and application of the proposed algorithm to a set of benchmarks. Finally Section 6 summarizes the main contributions of the work and concludes the paper.

2. MODELING THE BIT ERROR RATE IN A NAND FLASH MEMORY

The Bit Error Rate (BER) of a page, i.e., the fraction of bits that contain incorrect data, is the key factor to quantify the NAND flash reliability and hence to select the ECC correction capability [Mielke et al. 2008]. In this paper we focus on retention errors and program errors caused by cell aging that are the two dominant types of errors in MLC NAND flash [Cai et al. 2012]. Two values of BER must be considered when characterizing a NAND flash. The Raw Bit Error Rate (RBER) is the BER before applying the error correction. The RBER is technology/environment dependent and is

not constant; it increases with page aging [Brewer and Gill 2008; Mielke et al. 2008; Cai et al. 2012]. The Uncorrectable Bit Error Rate (UBER) is instead the BER after the application of the ECC. The UBER is application dependent and sets the target reliability of the storage system. Manufacturers of NAND flash storage systems often report UBER values on their data-sheets, in the range of 10^{-11} to 10^{-16} [Gray and van Ingen 2011]. The definition of UBER is a very useful reliability metric for mass-storage devices because a bit error that damages one file out of many is not equivalent to a functional failure that destroys the drive. UBER is therefore used to specify the data-corruption rate that is accepted in the target application [Mielke et al. 2008].

Considering an ECC with correction capability of p errors, UBER is the probability of having $E > p$ errors in the page divided by the number n of bits in the page [Cooke 2007]. Several studies reported that program and retention errors in a page are in general non-correlated [Yaakobi et al. 2009; Mielke et al. 2008; Micheloni et al. 2008; Micheloni et al. 2010; Yang et al. 2012; Yaakobi et al. 2012; Tanakamaru et al. 2013]. UBER can therefore be computed according to eq. (1) that considers a binomial distribution of randomly occurred bit errors:

$$\mathrm{UBER} = \overbrace{\frac{1}{n}\sum_{i=p+1}^{n}\binom{n}{i}\cdot \mathrm{RBER}^{i}\cdot(1-\mathrm{RBER})^{n-i}}^{\frac{P(E>p)}{n}} \tag{1}$$

From eq. (1) it is clear that, in order to properly model the NAND flash reliability, it is mandatory to model its RBER. RBER modeling for NAND flash is still a matter of debate since there are multiple approaches to derive a consistent model valid for different NAND technologies. Nevertheless, from an empirical and statistical characterization of several devices [Mielke et al. 2008; Micheloni et al. 2010] it is possible to infer a set of equations describing the RBER behavior under different memory operating points (i.e., writing, erasing, disturbing, or retaining the data).

In a n-bit MLC NAND flash, the threshold voltage (V_{th}) of each cell can be programmed to 2^{n} separate states. Each state corresponds to a non-overlapping threshold voltage window. Cells programmed to the same n-bit value have their threshold voltages falling into the same window, but their exact threshold voltages could be different. It is therefore possible to identify a V_{th} distribution for each available state. As an example, in a 2-bit MLC memory, each cell can assume $2^2 = 4$ separate states and therefore four V_{th} distributions L0-L3 are used (see Fig. 1). The non-overlap space between the distributions is called the distribution margin. Four predefined read reference voltages ($\mathrm{V_{R0}}$-$\mathrm{V_{R3}}$ in Fig.1) are used to discriminate between the four possible cell states. These read reference voltages are located in the distribution margins of the threshold voltage distributions.

An erase operation sets all cells of a block at level L0. L0 is the starting point for each program operation that moves the threshold voltages of the selected cells to one of the L1-L3 levels. A standard algorithm named incremental step pulse programming (ISPP) is usually exploited to accomplish this operation [Micheloni et al. 2010]. A voltage step of predefined amplitude and duration is applied to the gate of each programmed cell. Afterwards, a verify operation takes place. It verifies whether the V_{th} of each cell falls into the target V_{th} distribution. All cells that have reached the desired distribution level are excluded from the following pulses. For the remaining cells another cycle of ISPP is applied after incrementing the programming voltage. In fact, the V_{th} distributions of the L1-L3 levels significantly deviate from an ideal Gaussian shape. They often cross the distribution read levels ($\mathrm{V_{R0}}$-$\mathrm{V_{R3}}$) and cause bit errors. Therefore, the RBER of a MLC flash memory strictly depends on the shape of

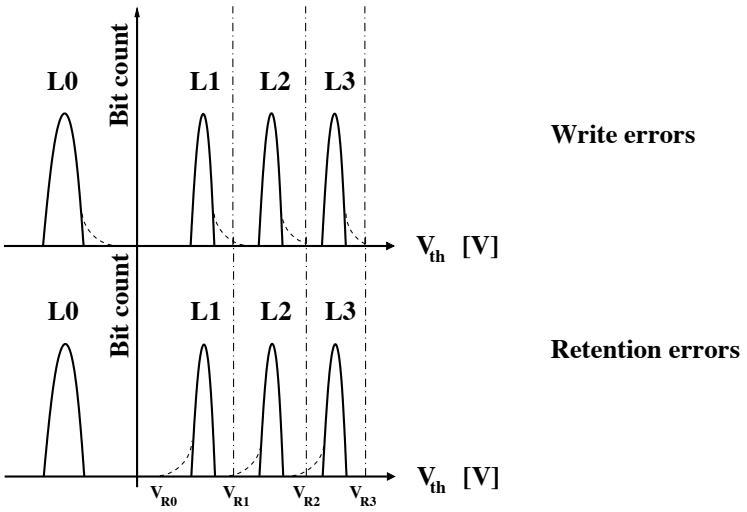


Fig. 1. Threshold voltage distribution of a 4-levels MLC NAND Flash. The effects of both retention and write errors are highlighted as distribution tails crossing the reference voltages ($V_{R0..3}$) used for discrimination between levels. The erased distribution L0 is affected only by write errors [Micheloni et al. 2010].

the V_{th} distributions associated to the different cell states (see Fig. 1). The upper tail of a distribution is mainly caused by over-programmed cells and disturbed cells (i.e., unintended program errors), whereas the lower tail is mainly due to charge losses over time (i.e., retention errors). These, represent the two main source of errors considered in this paper [Mielke et al. 2008].

A simple NAND flash RBER model has been proposed by Sun et al. [2011]. The RBER of different MLC NAND flash parts from a variety of vendors and different NAND technologies (3x nm, 4x nm, and 5x nm) has been quantified as a function of the number of program/erase cycles. Empirical data have been then curve-fit by an exponential growth model. The resulting fitting equation takes the following form:

$$\mathrm{RBER}(PE) = A \cdot e^{B \cdot PE} + C \tag{2}$$

where PE is the instantaneous program/erase count, and A, B, and C are fitting constants computed with a 95% confidence bound least-square regression. The main benefit of this model lies in its simplicity. However, it has some drawbacks. First, the RBER may be under/over-estimated for low PE values (i.e., at the beginning of the memory life-time). Second, the flash RBER is not actually independent from the memory operating conditions. Eq. (2) does not include any time-related term, introducing an approximation that prevents its usage in real scenarios.

An alternative to the model of eq. (2) is described in [JEDEC 2011]. A power-law model is adopted to describe the NAND flash error-rate as a function of the elapsed time and of the program/erase cycles:

$$\mathrm{RBER}\left(PE, t_{ret}\right) = \mathrm{RBER}_{wr} + B_o \left(PE^n \cdot t_{ret}\right)^m \tag{3}$$

where t_{ret} is the page retention time measured in hours, PE is the instantaneous program/erase cycles count of the page, m is a coefficient whose numerical value is usually between 1 and 2, n is a power-law coefficient for program/erase cycles,RBER_{wr} is the error rate observed at $t_{ret} = 0$, and B_o is a scale factor, which depends on the target technological process. The coefficients m, n and B_o can be derived through curve-fitting of experimental data retrieved from the flash technology under investigation. RBER_{wr}

mainly corresponds to errors due to program/erase cycling. The RBER contribution modeled by eq. (3) increases as retention time increases. Moreover, this increment increases with larger program/erase cycles. This models the fact that fresh devices have less retention problems, while, in worn-out devices, retention errors increase. Actually, it has been observed that the RBER does not increase monotonically [Cai et al. 2012]. The retention charge loss shifts back the V_{th} distributions. Therefore, for a short time this phenomenon partially mitigates the distribution upper tails effect (Fig. 1) and reduces the RBER. However, after a short time, the retention effects start introducing far more errors than the ones that can be mitigated by the V_{th} distributions shift and the RBER starts increasing. Therefore, eq. (3) slightly overestimates the retention RBER in fresh devices, representing an acceptable worst-case estimation.

The model proposed by eq. (3) has the main drawback that it does not give any provision on how to estimate RBER_{wr}, which represents the RBER at the beginning of the retention time. However, RBER_{wr} can be estimated resorting to the model presented in eq. (2) that takes into account program/erase cycles effects as:

$$\mathrm{RBER}_{wr}(PE) = A \cdot e^{B \cdot PE} + C \tag{4}$$

The two models can therefore be combined in order to obtain a more precise estimation of the RBER. To summarize, the accurate RBER model exploited in this paper, that combines the models presented in eqs. (2) and (3) and considers program/erase cycling impact ($\mathrm{RBER}_{wr}(PE)$) along with retention loss effects ($\mathrm{RBER}_{rd}(PE, t_{ret})$), is represented by the following equation:

$$\mathrm{RBER}\left(t_{ret}, PE\right) = \overbrace{\left[A \cdot e^{B \cdot PE} + C\right]}^{\mathrm{RBER}_{wr}(PE)} + \overbrace{B_o \left(PE^n \cdot t_{ret}\right)^m}^{\mathrm{RBER}_{rd}(PE, t_{ret})} \tag{5}$$

In the next section we present how the aforementioned analytical model can be exploited to dynamically select the proper correction capability for a flash page.

3. SELECTING THE CORRECTION CAPABILITY FOR A FLASH PAGE

This section introduces the FLARES algorithm. FLARES estimates the best ECC correction capability to apply to a NAND flash page in order to meet the target UBER, while maximizing the NAND flash performance. The basic idea beyond FLARES is to estimate the RBER of the flash page resorting to the model introduced in eq. (5), coupled with error rate information collected at run-time. To perform RBER run-time estimation, FLARES requires to associate the following information items, denoted as Page run-time PROFile (PPROF), to each page of the NAND flash:

— $\mathrm{p_{cur}}$: the correction capability that has been used to encode the content of the page,
— $\mathrm{p_{next}}$: the correction capability to use at the next page programming to sustain the selected UBER,
— $\mathrm{pecycles}$: the number of program/erase cycles applied to the page,
— $\mathrm{writestamp}$: the timestamp of the last program operation, required to compute the page retention time,
— errc: a counter accumulating the total number of errors detected when reading the page, and
— failc: a counter counting the number of times the ECC decoding fails.

The PPROF of each NAND flash page must be constantly updated at run-time. Before presenting FLARES we therefore present the upPage function reported in Alg. 1 that introduces the PPROF update strategy. The $\mathrm{operation}$ parameter determines the performed memory access (i.e., read or programming).

In case of read operation (rows 2-7 - Alg. 1) the ECC decoder provides the fail flag indicating whether the ECC decoding was successful or not and the number deterr of detected and corrected errors. If the decoding was successful ($\mathrm{fail} = \mathrm{false}$), errc is updated adding the detected number of errors (row 3 - Alg. 1). If not, the page failure count failc is incremented (row 5 - Alg. 1). In this case, the decoder is unable to correctly compute the actual number of errors in the page. errc is therefore updated adding a number of errors approximated to the current correction capability ($\mathrm{p_{cur}}$) plus one error (row 6 - Alg. 1). It is worth to remember here that, in this case, the page must be invalidated. Its content cannot therefore be used anymore. Instead, in case of programming operation (rows 9-10 - Alg. 1), the number of program/erase cycles of the page is incremented (row 9 - Alg. 1) and the page writestamp is generated and saved (row 10 - Alg. 1).

ALGORITHM 1: $\mathrm{upPage}(\mathrm{operation}, \mathrm{deterr}, \mathrm{fail}, \mathrm{p_{cur}})$

```
1  if operation = read then
2      if fail = false then
3          errc = errc + deterr
4      else
5          failc = failc + 1
6          errc = errc + p_cur + 1 {The number of errors in the page is higher than the correction
           capability of the code. The page must be invalidated.}
7      end
8  else
9      pecycles = pecycles + 1
10     writestamp = current_time
11 end
```

The PPROF contains enough information to perform RBER estimation for a NAND flash page according to the model introduced in eq. (5). However, a fixed estimation model such as the one proposed in eq. (5) is unable to take into account error rate variations due to specific run-time and environmental conditions (e.g., temperature stress) that may arise in real applications. As a consequence the reliability of the flash could be either overestimated or underestimated, with potential risk for the data integrity. To avoid this, we introduce the FLARES estimation algorithm reported in Alg. 2. FLARES complements theoretical RBER estimation with run-time RBER estimation obtained resorting to information provided by the ECC subsystem.

The correction capability $\mathrm{p_{next}}$ that must be applied to the page at the next programming operation is computed by observing windows of `WSIZE` operations. The contribution of the retention time to the RBER of the page is first evaluated. The page retention time is computed as the difference between the current time and the $\mathrm{writestamp}$ that has been saved in the PPROF during the last write operation (Alg. 2, row 1). If the current retention time is approaching the maximum page retention time, a rewrite alarm is issued to inform the flash management system that the page must be rewritten to avoid loss of information due to retention errors (Alg. 2, rows 2-4). It is worth to mention here that, with the introduction of a variable ECC correction capability, the maximum retention time that enables to sustain the selected UBER is not constant but is a function of the selected ECC correction capability. It can be computed by substituting eq. (5) into eq. (1) and inverting the obtained equation. Since this operation involves the computation of complex binomials, a static table of maximum retention times associated to different values of $\mathrm{p_{cur}}$ and $\mathrm{pecycles}$ ($\mathrm{max\_ret\_time}(\mathrm{p_{cur}}, \mathrm{pecycles})$) is precomputed and searched every time the maximum retention time of a page must be evaluated.

ALGORITHM 2: FLARES(PPROF)

```
   Require: const WSIZE, PAGESIZE, SAFERANGE, MAXFAIL, MAXCRITICAL
   Require: MAXOVER, MIX, REQ_RET_TIME
   Require: var criticalc, overc, writestamp, current_time
 1 ret_time = current_time − PPROF.writestamp
 2 if ret_time > search(max_ret_time (p_cur, PPROF.pecycles)) then
 3 |   alarm("page rewrite required")
 4 else
 5 |   meas_rber = (PPROF.errc/PAGESIZE)/WSIZE−RBER_rd(PPROF.pecycles, ret_time)
 6 |   model_rber = RBER_wr(PPROF.pecycles)
 7 |   avg_rber =MIX∗meas_rber + (1−MIX) ∗ model_rber
 8 |   proj_rber = avg_rber + RBER_rd (PPROF.pecycles, REQ_RET_TIME)
 9 |   (min_rber, max_rber, p) = search(corr_table, proj_rber)
10 |   if PPROF.failc >MAXFAIL then
11 |   |   invalidate page
12 |   |   PPROF.p_next = max(PPROF.p_cur + 1, p) {failure zone}
13 |   |   PPROF.failc = 0
14 |   else if p > PPROF.p_cur then
15 |   |   PPROF.p_next = p {fast zone}
16 |   else if p < PPROF.p_cur then
17 |   |   overc = overc + 1 {overcorrection zone}
18 |   |   if overc >MAXOVER then
19 |   |   |   PPROF.p_next = PPROF.p_cur − 1
20 |   |   |   overc = 0, criticalc = 0, errc = 0
21 |   |   end
22 |   else if proj_rber > max_rber − max_rber ∗ (1−SAFERANGE) then
23 |   |   criticalc = criticalc + 1 {critical zone}
24 |   |   if criticalc >MAXCRITICAL then
25 |   |   |   PPROF.p_next = PPROF.p_cur + 1
26 |   |   |   overc = 0, criticalc = 0, errc = 0
27 |   |   end
28 |   else
29 |   |   PPROF.p_next = PPROF.p_cur; {safe zone}
30 |   end
31 |   errc = 0
32 end
```

If the retention time is still within an acceptable interval, the contribution of the program/erase cycles to the page reliability is evaluated (Alg. 2, rows 5-30). The number of errors detected by the ECC during the selected window of operations (errc) is used to obtain an empirical measure (meas_rber) of the page RBER (Alg. 2, row 5). meas_rber is computed as the number of detected errors divided by the page size and averaged over the window size. Since this measurement also includes the contribution of retention errors, the theoretical value of this contribution ($\mathrm{RBER}_{\mathrm{rd}}$ from eq. (5)) is removed. This is motivated by the fact that the error rate contribution due to retention errors is transitory, i.e., it vanishes once the page is rewritten. Therefore, any control action (beside the page refresh upon retention check) must not depend on such component. Following, model_rber, the theoretical contribution of program/erase cycles to the page RBER ($\mathrm{RBER}_{\mathrm{wr}}$ from eq. (5)) is computed (Alg. 2, row 6). The two measures are then combined to obtain an average RBER estimation (avg_rber) for the page (Alg. 2, row 7). The MIX parameter is a internal parameter between 0 and 1, used to tune the contribution of meas_rber and model_rber to the final RBER estimation.

According to the JEDEC standard JESD47G.01 [JEDEC 2010], a NAND flash must retain data for a maximum retention time (REQ_RET_TIME) of 1 year when cycled at maximum endurance. If the correction capability is chosen according to avg_rber, which is related to a device with no retention, the minimum retention requirement could not

be met. The correction capability must be instead chosen using the RBER value after 1 year of retention. An estimation of this value, $\mathrm{proj\_rber}$, is obtained by projecting $\mathrm{avg\_rber}$ at 1 year retention. This is done (Alg. 2, row 8) by adding to $\mathrm{avg\_rber}$ the RBER contribution due to the minimum required retention ($\mathrm{RBER_{rd}}$). Given $\mathrm{proj\_rber}$, eq. (1) can be reversed to obtain the minimum correction capability p required to sustain the target UBER. Again, since this operation involves the calculation of large sums and binomials, a static table ($\mathrm{corr\_table}$) in which each row is associated to a correction capability p and a given range of RBER ([$\mathrm{min\_rber}$, $\mathrm{max\_rber}$]) is computed and searched when required (Alg. 2, row 8).

Given the page profile and the computed correction capabilities, five conditions may arise:

(1) $\mathrm{proj\_rber}$ imposes a correction capability p higher than the current correction $\mathrm{p_{cur}}$ (Alg. 2, rows 14-15). In this situation, called *fast zone* (FS), $\mathrm{p_{next}}$ is updated to the computed p to immediately follow the change in the requested correction capability.

(2) $\mathrm{proj\_rber}$ imposes a correction capability p lower than the current correction $\mathrm{p_{cur}}$ (Alg. 2, rows 17-21). In this situation, called *overcorrection zone* (OC), the current correction capability is overestimated. However, it is risky to immediately lower the correction capability of the page since there might be fluctuations of $\mathrm{avgrber}$ when considering different windows. A counter (overc) is used to record the number of times this situation arises. If this number becomes greater than a predefined threshold (`MAXOVER`), we assume that the correction capability can be safely lowered of one unit (Alg. 2, rows 18-20).

(3) $\mathrm{proj\_rber}$ imposes a correction capability p equal to the current correction $\mathrm{p_{cur}}$, but its value falls in a neighborhood of $\mathrm{maxrber}$ defined by a `SAFERANGE` constant (Alg. 2 rows 22-23). In this situation, called *critical zone* (CR), the RBER of the page is approximating the limit manageable by $\mathrm{p_{cur}}$. A counter ($\mathrm{criticalc}$) counts the number of occurrences of this situation. If this number becomes higher than a predefined threshold (`MAXCRITICAL`) we increase the correction capability of one unit. In this way we try to anticipate the need for higher correction capability in pages that are approaching the correction limit of the current code (Alg. 2, rows 24-27).

(4) the number of detected failures in the window is higher than a given threshold `MAXFAIL` (Alg. 2, rows 10-13). This situation, called *failure zone* (FA), is the most critical condition, since the selected code has been repeatedly unable to handle the amount of errors detected in the page. The correction capability is therefore immediately incremented in order to deal with the new situation. The increment is performed even if the computed p is not actually higher than $\mathrm{p_{cur}}$ (Alg. 2, row 12). It is worth to note that this zone is considered for safety reasons, only. If the proper retention margins are considered, the probability of entering this zone must be negligible. In fact, a page refresh should be issued before reaching a critical retention condition of the page.

(5) if none of the previous conditions is true (Alg. 2, row 29), the page is in the *safe zone* (SZ). The current correction capability is proper and no action must be taken.

The different thresholds introduced in FLARES provide several degrees of freedom to precisely tune the ECC adaptation to the characteristic of the flash and to the memory access profile of the target application. FLARES must be scheduled every time a page is accessed, but it must be also scheduled periodically in order to monitor those pages that are rarely accessed but whose retention time might become critical for the stored information. It is worth to mention here that the use of a variable ECC correction capability for each page makes the information stored in the PPROF critical for the correct behavior of the NAND flash. In particular the correction capability $\mathrm{p_{cur}}$ used to encode the page is critical in order to properly retrieve the stored informa-

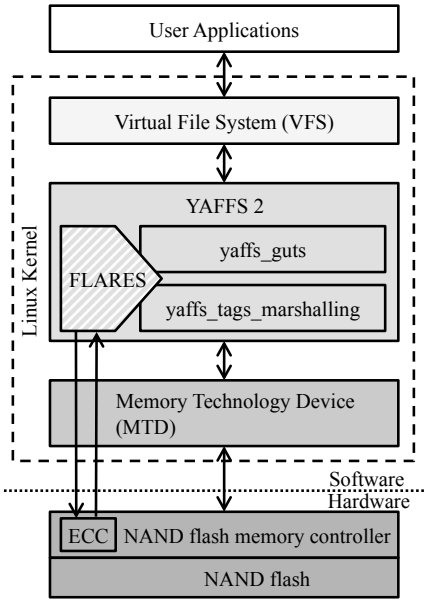


Fig. 2. FLARES Development environment

tion. The PPROF must therefore be stored in a reliable way but without impacting the flash endurance and the flash performance. A solution to this problem in a selected implementation framework will be discussed in the next section of this paper.

4. IMPLEMENTATION

There are several options to implement FLARES in a real environment. Given its low computational demand it can be implemented in the flash translation layer (FTL), i.e., into the firmware of the NAND flash memory controller, or as part of the flash file system [Di Carlo et al. 2011]. In this paper, FLARES has been integrated within YAFFS 2 (Yet Another Flash File System version 2) [yaffs.net 2007], one of the most famous open-source flash memory file systems. This choice is motivated by the availability of the file system source code that can be analyzed and modified in order to implement the FLARES algorithms. Moreover, implementing FLARES at the filesystem level, makes it independent from the specific flash memory subsystem.

4.1. Environment

Figure 2 reports the software and hardware stack in which FLARES has been implemented. The whole environment runs under the Linux operating system. User applications are decoupled from YAFFS 2 by means of the Linux Virtual File System (VFS) layer. Therefore, they do not require any modification to work with FLARES.

FLARES is implemented as an additional YAFFS 2 module as will be better explained later in this section. It therefore communicates with the Linux Memory Technology Device (MTD) that acts as a device driver interfacing the file system with the NAND flash controller. In a real environment, the NAND flash controller manages the flash operations and performs ECC encoding and decoding exploiting fast ECC hardware structures such as the ones presented in [Zambelli et al. 2012; Di Carlo et al. 2012; Fabiano et al. 2013; Atieno et al. 2006; Chen et al. 2009]. To perform controlled experiments in which errors can be easily emulated and realistic performance pre-

cisely measured, the full hardware layer reported in Figure 2 has been emulated at the MTD level.

The MTD has been instrumented to emulate the presence of a NAND flash memory by storing flash pages in RAM. Following Cai et al. [2012] we emulated a real 2-bit per cell 4-levels MLC NAND memory featuring a 3x nm manufacturing process. The memory includes 4,096 blocks of 128 pages. The page size is 4KB plus 224B of spare area. Table Ia reports the flash performance for read and write operations emulated by the MTD as well as information about power consumption and endurance (i.e., maximum number of PE cycles) that will be used in the experiments to evaluate the benefit of FLARES. Fast RAM read/write access, compared to the flash access time, allows us to emulate the NAND flash behavior in software, without introducing any timing overhead.

Table I. Parameters used to tune FLARES to the specific experimental setup used in this paper to show its benefit and performance on the system performance.

Page write time (AVG) @ cycle 1	800us
Page read time	75us
Write operation power consumptions	0.164 W
Read operation power consumptions	40 mW
Maximum considered P/E cycles	10,000
Page Size	4 kB + 224B

(a) NAND Flash simulation parameters. Programming timings are provided at cycle 1

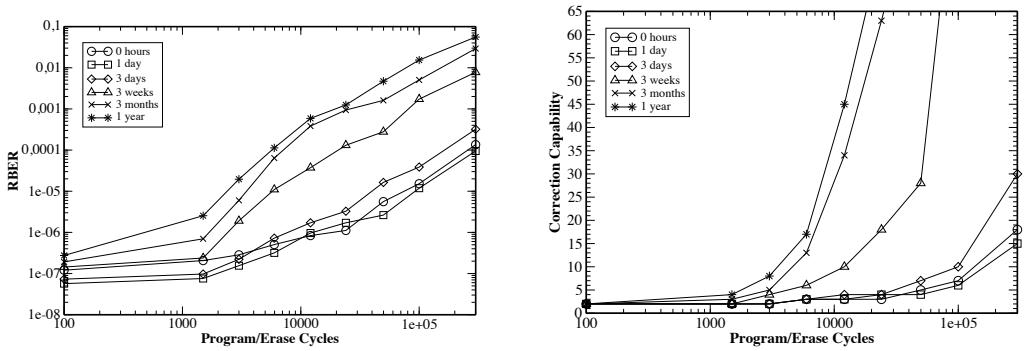
Constant	Value
SAFERANGE	5%
MAXFAIL	3
MAXCRITICAL	5
MAXOVER	15

(b) Algorithm constants.

From the electrical simulations and physical experiments performed in Cai et al. [2012] we have been able to extrapolate physical flash parameters that have been used to characterize the NAND flash reliability and, therefore, to properly tune and configure FLARES for a target experimental setup. Figure 3a reports the measured flash RBER as a function of the PE cycles, for different retention times ranging from 0 hours to 1 year. Computed data refer to 25°C equivalent temperature conditions. The ECC correction capability required to sustain an UBER of 10^{-11} is instead reported in Figure 3b. Experimental measurements report that for large program/erase cycles and retention values, the RBER becomes very high. This makes error correction infeasible due to unacceptable ECC complexity and latency (the ECC correction capability would exceed 100 bits). For this reason, following other publications analyzing the endurance of MLC memories [Yaakobi et al. 2010; Grochowski and Fontana 2012], we considered a maximum endurance of 10,000 PE cycles per page.

The MTD has been also instrumented to emulate the ECC encoding/decoding activity performed by NAND flash controller. When emulating the ECC activity, the main problem to address is to introduce a mechanism to inject errors in the flash pages, and to precisely emulate the ECC encoding/decoding latency introduced by the NAND flash controller. FLARES is independent from the selected ECC. In this paper we considered the BCH ECC architecture proposed by Zambelli et al. [2012]. This architecture has two main advantages. First it implements a very fast parallel hardware core for BCH ECC encoding and decoding with programmable correction capability. It is designed for state-of-the-art MLC memories thus providing a realistic case study. Second, a VHDL model of the full encoding/decoding system is available. This enables precise RTL simulations to obtain a precise characterization of the ECC activity in terms of timing and power consumption.

When instrumenting the MTD to emulate the ECC activity it is worth to remember that the BCH encoding time is fixed and depends on the parallelism of the encoder (8-bit in our architecture) and on the ECC codeword size (a full page of the flash in



(a) Measured RBER for a page of the flash at different PE cycles and retention times.

(b) Required correction capability for a page of the flash at different PE cycles and retention times.

Fig. 3. Flash memory measuerd RBER and required correction capability

our architecture). Similarly, the ECC decoding time depends on the parallelism of the decoder (8-bit in our architecture), on the codeword size, on the selected ECC correction capability, on the actual number of errors in the codeword, and on the error location within the codeword (errors in the last bits of the codeword require more clock cycles to be corrected). In order to emulate the impact of the ECC on the system's performance it is therefore enough to emulate the ECC latency according to the aforementioned parameters. The ECC power consumption is also affected by the same parameters. In our architecture the worst case ECC decoding time ranges between 83.9us for a correction capability equal to one and 194us for a correction capability equal to 50 that is the maximum correction capability required in our experimental setup.

We performed an extensive set of VHDL simulations on the architecture proposed by Zambelli et al. [2012] to characterize the ECC latency and power consumption for the encoding and decoding operation injecting variable number of errors on a large set of locations of a flash page. Latency results have been tabulated and instrumented in the MTD. Every time a memory page is accessed, depending on the target ECC correction capability and on the errors in the page the ECC encoding/decoding latency is emulated. Errors can be injected in a page through an injection function implemented in the MTD. This function receives as a parameter the minimum and the maximum number of errors that must be injected in a target page. It randomly generates the number of errors in this range and its location in the page randomly taken from the list of error locations simulated in the VHDL model. This information is recorded in a data structure instrumented in the MTD that is used every time the memory page is accessed to emulate the ECC activity.

Finally, without loosing generality, some of the constants required to tune FLARES have been fixed and reported in Table Ib.

4.2. FLARES Implementation Details

FLARES requires to write and to read information contained in the page run-time profile associated to each page of the flash. This is similar to what already happens with *tags* in YAFFS 2 that store page information required to build the filesystem. Hence, as reported in Fig. 2, FLARES has been implemented as an additional module within YAFFS 2 at the tags marshalling level (contained in the yaffs_tagsmarshall.c YAFFS source file).

One of the most critical activities performed by FLARES is maintaining updated PPROF information for each page of the flash. Following the policy implemented by YAFFS 2 to manage its tags, each PPROF must be stored in flash to store the informa-

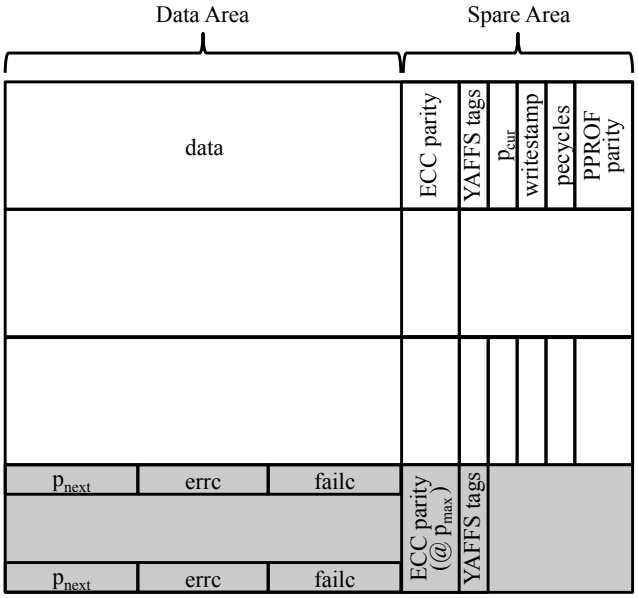


Fig. 4. PPROF data organization

tion across system reboots, but it must also be cached in RAM to enable fast access and to minimize page programming that may impact the flash endurance. The coherence of the two copies must be always guaranteed.

When considering the PPROF of a page two sets of information items with different impact on FLARES and different storage requirements can be identified.

$\mathrm{p_{cur}}$, $\mathrm{pecycles}$ and $\mathrm{writestamp}$ are high-critical information items. They are essential to decode the page and to keep track of the page wear-out. These items change their value when a page is programmed and remain constant when the page is read. We therefore store this portion of the PPROF in flash every time a page is programmed resorting to the page spare area (together with the ECC parity bits and other YAFFS 2 tags as reported in Fig. 4). This write policy provides high robustness. The PPROF stored in flash is always updated and consistent with the cached copy in RAM with no penalties on the flash endurance (no additional page programming operations required). Nevertheless, it is important to avoid the corruption of this portion of the PPROF due to errors that may arise in the flash. The sequence of bits composing the three variables is protected by a dedicated ECC (a BCH code in our implementation) whose correction capability is fixed and designed for worst case conditions. The parity bits of this ECC (PPROF parity in Fig. 4) are stored together with the PPROF in the flash spare area. The introduction of this extra ECC (PPROF ECC) has low impact on the system's performance. Even if designed for worst case conditions, it is applied on a small block of data. It generates a small number of parity bits and it introduces a negligible encoding time. Moreover, the ECC decoding phase, which is the ECC most complex operation, is performed only once when the PPROF is cached, again introducing a minor impact on the overall system's performance.

The remaining part of the PPROF containing $\mathrm{p_{next}}$, errc and failc is instead continuously updated at run-time. It is therefore impossible to guarantee the continuous consistency between cached information and flash content. This portion of the PPROF is therefore cached and updated in RAM during normal operations and flushed only when the filesystem is unmounted resorting to a few dedicated reserved pages (Fig-

ure 4 gray pages). This is in line with the YAFFS 2 architecture that exploits reserved flash pages for storing file system related information. To guarantee the integrity of the file system, all reserved pages are not controlled by FLARES and always protected with worst case ECC. Again, since these pages are rarely accessed the use of the worst case ECC has a reduced impact on the overall system's performance. Given the inconsistency between cached information and flash information for this portion of the PPROF, problems may arise if the cache is not properly flushed when the filesystem is unmounted (e.g., due to a power loss). Nevertheless, this information is not critical for the proper behavior of the file system. Even if lost, data in the flash can still be properly accessed and the lost PPROF information can be re-computed by FLARES at run-time after the analysis of a few windows of operations.

The PPROF cache policy is important to guarantee high system's performance. Loading all PPROFs in RAM when the file system is mounted is time-consuming and therefore not feasible. While the content of the reserved pages can be cached when the filesystem is mounted (only few pages are used), the remaining portion of the PPROFs that is spread in the spare area of each page is cached on *on-demand*. When a page is accessed for the first time after mounting the filesystem, two situations may arise:

— *Read operation*. Both the page content and the spare area where the PPROF is stored are read from the flash. The PPROF can therefore be cached without introducing additional operations.
— *Program operation*. An additional read operation is performed before the write operation to access the page spare area and to read the page PPROF. $\mathrm{pecycles}$ is then used to properly update $\mathrm{p_{next}}$ and therefore to decide the correction capability to apply.

It is worth to highlight here that, with the increase of the flash size, the cost of caching the PPROF of all pages may increase. Following traditional caching algorithms one solution to this problem is to use a small cache and maintain only a subset of the PPROF using a typical cache replacement algorithm (e.g., LRU, pseudorandom, etc.). This is a very simple solution that, however, introduces overhead in terms of performance and wear-out. Another possibility is instead to let group of pages share the PPROF data so as to reduce the cache size. This solution makes the PPROF caching easier, nevertheless it reduces the fine tuning capabilities of FLARES since within a group the worst case correction capability must be always considered.

Together with the cache policy, the way FLARES is scheduled is another important aspect to optimize both flash performance and reliability. FLARES is scheduled every time a flash page is either read or programmed. However, this is not enough to guarantee the flash integrity. Pages encoded with low correction capability that are rarely accessed may become critical due to retention errors. In order to properly issue the page rewrite alarms available in FLARES, rarely accessed pages can be checked in background during the system's idle time, thus constantly monitoring the full flash content.

In the remaining of this section we try to quantify the flash memory overhead introduced by FLARES to store the PPROFs. As previously mentioned we consider a target UBER of 10^{-11}. Starting from the RBER reported in Figure 3a, according to eq. (1) and to the model proposed in Section 2 fitted on the experimental data, we require an ECC with correction capability (i.e., $\mathrm{p_{cur}}$ and $\mathrm{p_{next}}$) ranging from 3 to 50 errors. Both $\mathrm{p_{cur}}$ and $\mathrm{p_{next}}$ can therefore be represented on 6 bits. In our implementation, the ECC parity bits are computed on a full flash page (i.e., 4KB). According to the BCH theory (see Micheloni et al. [2008]) a Galois field $GF(2^{13})$ can be used to build the BCH code. The number of parity bits to store ranges from $3 \cdot 13 = 39\mathrm{bit} \cong 5\mathrm{B}$ when the minimum correction capability is selected to $50 \cdot 13 = 650\mathrm{bit} \cong 81\mathrm{B}$ when the worst case correction

capability is selected. In the considered technology, $\mathrm{pecycles}$ ranges from 0 to 10,000. It can therefore be represented on 14 bits. The $\mathrm{writestamp}$ is instead represented as a default Linux 32 bits unsigned integer timestamp. The maximum errc value is given by the product of the maximum number of errors that can be detected at each read operation (i.e., $\mathrm{p}_{\max} + 1$, in case of failure) and `WSIZE`. In our implementation, considering $\texttt{WSIZE} = 100$, this leads to a maximum value of 5,100 that can be represented on 13 bits to be stored. Finally failc is limited by the `MAXFAIL` value (3, in this implementation). Two bits are enough for its representation. Table II summarizes the memory occupation of the PPROF fields for our specific implementation.

Table II. PPROF memory occupation

Quantity	Occupation
$\mathrm{p}_{\mathrm{cur}}$	6 bits
$\mathrm{p}_{\mathrm{next}}$	6 bits
$\mathrm{pecycles}$	14 bits
$\mathrm{writestamp}$	32 bits
errc	13 bits
failc	2 bits

According to data reported in Table II, the ECC required to protect the critical portion of the PPROF stored in the page spare area works on a block of $k = 52$ bits. According to the BCH theory (see Micheloni et al. [2008]), and considering RBER information reported in Figure 3a, a Galois field $GF(2^7)$ can be used to build the ECC code and a correction capability $\mathrm{p}_{\mathrm{PPROF}} = 5$ is required to achieve an UBER of 10^{-11} in the worst case scenario. This introduces 35 parity bits that must be stored in the spare area.

According to the previous discussion, in the worst cast each page must store in the spare area 81B for ECC parity, $\sim$7B to store $\mathrm{p}_{\mathrm{cur}}$, $\mathrm{pecycles}$ and $\mathrm{writestamp}$, $\sim$4B to store the PPROF parity plus 36B to store YAFFS tags for a total of $\sim$128B. This leaves enough space in the spare area (224B) to implement system level management policies such as wear leveling and bad block management.

Finally, each page must store two counters (failc and errc, of 13 and 2 bits, respectively) plus 6 bits for $\mathrm{p}_{\mathrm{next}}$ in the extra reserved pages. With the considered flash technology each reserved page is able to store 1,560 partial PPROFs leading to a total of 337 pages dedicated to FLARES. This represents the 0.06% of the flash storage capacity, which can be acceptable when considering the gain in terms of performance and power consumption provided by FLARES.

To properly estimate $\mathrm{RBER}_{\mathrm{wr}}$ and $\mathrm{RBER}_{\mathrm{rd}}$ (Alg. 2, rows 5-8), floating point calculations of complex functions must be performed. However, working at the Linux kernel level, floating point calculations are not permitted. Fixed point calculations have been therefore exploited. Splines (piecewise cubic polynomials) have been off-line interpolated on the data presented in Subsection 4.1. These splines are then evaluated at run-time to approximate complex functions. Using long numeric representations and proper number of spline pieces (or knots), it is possible to guarantee a representation error significantly lower than the RBER measurements resolution. Similarly, fixed tables have been used to avoid complex computations in the choice of the correction capability according to the RBER and the retention time. In the current implementation of FLARES, all fixed tables have been hardcoded in the FLARES source code. Nevertheless, this is not the optimal solution. Since these tables are technology dependent, in a real environment they should be moved within the flash driver thus enabling to manage different devices with appropriate parameters.

Extensive experimental campaigns have been performed to quantify benefits provided by Flares. Such results are discussed in the next section.

5. EXPERIMENTAL RESULTS

This section reports the results of two sets of experiments performed on FLARES. The first set of experiments aims at validating the proposed approach demonstrating the capability of FLARES to correctly adapt the ECC correction capability to the error rate of the flash. The second set of experiments shows the impact of FLARES on the NAND flash wear-out and on the system performance. This enables to quantify the trade-off capability between performance and wear-out.

The FLARES RBER prediction model described in Section 2 has been fitted to the measured RBER data for the target flash technology reported in Figure 3a. The parameters A, B and C of eq. (2) have been estimated using the MATLAB® Curve Fitting tool against data related to 0 hours retention, while parameters from eq. 5 have been fitted using the MATLAB® Surface Fitting tool. Table III reports fitted and calculated data and some related fitting output. The correlation coefficients from the two fittings (i.e., $R_1^2 = 0.9979$ and $R_2^2 = 0.9876$) guarantee a good fitting of the model with only a small deviation from experimental data.

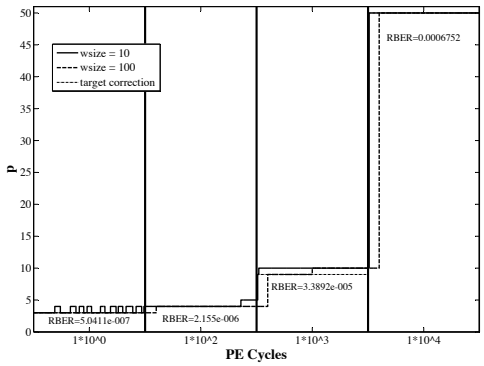
Table III. RBER model fitting output

Parameter	Value
A	1.059e-005
B	8.634e-006
C	-1.009e-005
R_1^2	0.9984
B_o	1.691e-011
m	0.6027
n	2.167
R_2^2	0.9879

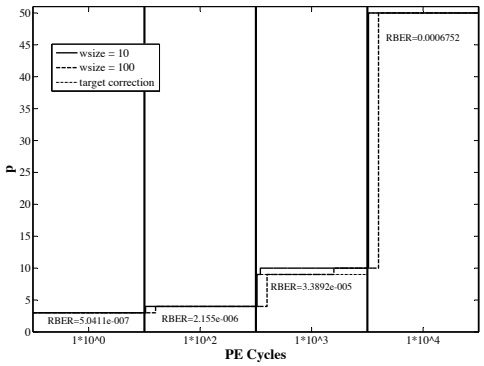
5.1. Validation results

The ability of FLARES to select the ECC error correction capability for a flash page has been validated by simulating memory operations on a single page of the flash memory, thus emulating the wear-out of the page and the occurrence of errors. Performing this simulation campaign running the full FLARES implementation would require a considerable amount of simulation time due to the need of emulating the wear-out of the flash across a large amount of operations. Nevertheless, it is worth to consider that FLARES performances are not important in this validation campaign. Therefore, the software and the emulated hardware stack reported in Fig. 2 do not require to be fully implemented and the validation effort can concentrate on the behavior of the FLARES algorithms (Alg. 1 and Alg. 2). For this reason, the two algorithms have been implemented in MATLAB® and the error occurrence has been emulated by providing proper values to the deterr and fail parameters of Alg. 1 as better explained later in this section. Using this simplified MATLAB® model, we have been able to perform simulations that emulate the behavior of FLARES on a single page of the flash with different execution parameters. Results are summarized in Fig.5a, 5b, 5c, and 5d.

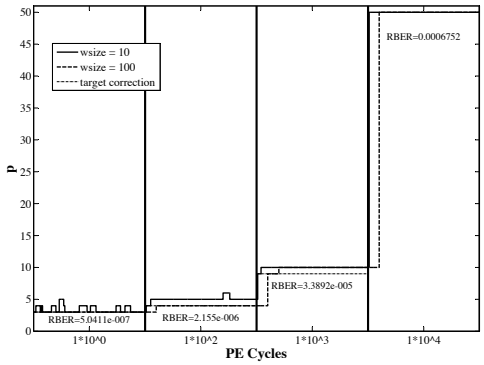
Even with a simplified MATLAB® implementation, simulating the full life of the page would be too time consuming. We therefore sampled five representative operating points corresponding to different wear-out conditions of the page (i.e., PE equal to 10, 10^2, 10^4, and 10^5). Each operating point, corresponding to one of the vertical sections of Fig. 5a, 5b, 5c, and 5d, sets a fixed 1-year-retention RBER (reported in the figure) and



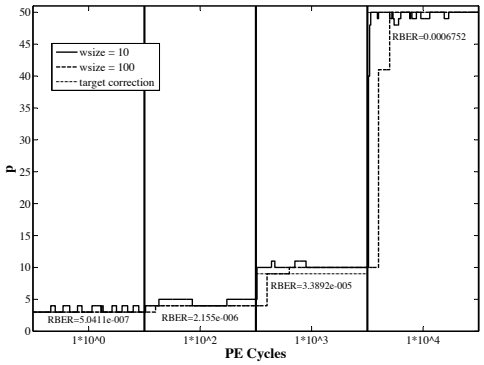
(a) FLARES adaptation strategy at works for different values of WSIZE. Simulations reported in this figure are performed considering $\mathrm{MIX} = 0.5$



(b) FLARES adaptation strategy at works for different values of WSIZE. Simulations reported in this figure are performed considering $\mathrm{MIX} = 0$



(c) FLARES adaptation strategy at works for different values of WSIZE. Simulations reported in this figure are performed considering $\mathrm{MIX} = 1$



(d) FLARES adaptation strategy at works for different values of WSIZE. Simulations reported in this figure are performed considering $\mathrm{MIX} = 0.5$ and 1 year minus 1 hour retention

Fig. 5. FLARES accuracy experimental results

therefore a different target ECC correction capability reported using a dotted line in the figures.

For each operation point we analyzed a sequence of 1,000 random flash operations for a total of 5,000 operations. During each operation errors have been introduced in the page in order to emulate the target RBER of the selected operation point. Furthermore, to model a certain level of variability in the injected error rate, the target RBER has been corrected with a value obtained sampling a Gaussian distribution centered in the target RBER with a standard deviation of $5 \cdot 10^{-7}$. This value has been chosen to be large enough to introduce variations in the number of errors, but small enough to guarantee that the order of magnitude of the target RBER remains the same. We repeated the analysis considering two different window sizes and three values of the MIX parameter to highlight the effect of these parameters on FLARES.

Let us consider Fig. 5a. It shows the ECC correction capability predicted by FLARES with MIX equal to 0.5. Looking at the figure, one can notice that, when the size of the window is small, the adaptation is very sensible to changes in the estimated RBER, with the drawback that the correction capability is in general slightly overestimated especially at the beginning of the flash life-time. This problem can be mitigated by enlarging the window size. When considering windows of 100 operations, the prediction

becomes more accurate, even if a certain delay in the adaptation can be observed. In general, the window size parameter allows the designer to trade-off between fast response and precision of the adaptation. It is also worth to note that FLARES is able to adapt even to sharp changes of the flash RBER as the one applied when moving from one operation point to the next one.

The contribution of the MIX parameter that sets whether the flash RBER estimation have to rely more on the measured RBER or on the modeled RBER can be appreciated looking at Fig. 5b and Fig. 5c. When MIX is set to zero (Fig. 5b), FLARES is able to properly predict the error correction capability of the flash. However, it looses the ability to react to changes in the error rate due to environmental conditions that have not been considered in the original model. Differently, when MIX is set to one, only the measured RBER is considered (Fig. 5c), FLARES becomes too sensible to local changes in the measured RBER leading to cases in which the correction capability is slightly underestimated. This clearly demonstrates the importance of combining the modeled RBER with the measured RBER as performed in FLARES.

Overall, considering our specific experimental setup, $\mathrm{MIX} = 0.5$ is the best option to obtain good adaptation and precise results. Nevertheless, the different parameters represent a valuable instrument to let designers carefully tuning FLARES to the specific technological and operative conditions of the target system. Looking at Fig. 5a the reader may note that, even when $\mathrm{WSIZE} = 10$ is selected, the target ECC correction capability can be underestimated for a certain time when moving from one operating point to the next one. While at a first analysis this may represent a degradation of the target UBER of the system, one have to consider that, in our simulation, we consider a sharp change of one order of magnitude of the target RBER when moving from one operating point to the next one. This represents a very critical condition for FLARES that, however, is very unlikely to happen in a real environment in which the RBER in general increases according to a continuous function. To better investigate this situation we performed an additional simulation in which we considered 1,000 operating points starting from PE equal to 10^3, and increasing PE of 10^4 when moving to an operation point to the next one. With this more realistic condition, considering both $\mathrm{WSIZE} = 10$ and $\mathrm{WSIZE} = 100$, FLARES never underestimated the target correcting capability, thus enabling to meet that target UBER of the application.

Finally, we performed a simulation in which the retention of the memory is stressed. This result (Fig. 5d) shows that FLARES is capable of adapting the correction capability also with very large retention times.

5.2. Wear-out and performance results

The impact of FLARES on the NAND flash wear-out and performance has been evaluated by estimating the impact it has on a set of workloads designed to perform intensive I/O operations. FLARES has been deployed within a virtual machine running the Linux operating system and the YAFFS 2 filesystem. The use of a virtual machine is required to execute the benchmarks in a controlled environment where only those processes required for the correct execution of the experiments are enabled.

Several file system benchmarks are available on the Internet (e.g. IOzone [iozone.org 2001], Postmark [Katcher 1997], SPEC benchmarks [spec.org 2001], Filebench [Wilson 2008], etc.). We selected the Filebench benchmark suite. Filebench is an open source File System benchmark managed by the File systems and Storage Lab group of the Computer Sciences Department of the Stony Brook University. It provides a large variety of benchmarks whose behavior can be specified using the Workload Model Language (WML). They either perform simple file I/O operations, or emulate complex I/O activities. We selected three Filebench workloads representative of three typical behaviors for the flash activity: (1) a read intensive *videoserver* application that reads a set of

video files from the flash; (2) a balanced read/write *webserver* emulating a web server application that opens, reads and closes files while writing a log file, and (3) a write intensive *varmail* application emulating a mail server performing create-append-sync, read-append-sync, read and delete operations on emails.

5.2.1. wear-out. FLARES introduces a mechanism to issue a page refresh command for those pages that are reaching their maximum retention time (see Alg 2, rows 2-4). While this approach enables to reduce the impact of retention errors on the flash, it may introduce additional programming operations that have an impact on the endurance of the flash. According to Cai et al. [2012] there are two options to refresh a flash page: (i) remap (copy/move) the page to a different location, or (ii) re-program it in its original location by recharging the floating gates.

The first solution is the one traditionally used in commercial controllers to implement wear leveling algorithms. Unlike DRAM cells, which can be refreshed in-place, flash cells generally must first be erased before they can be programmed. To avoid the slow erase operation, current wear leveling algorithms remap the data to another physical location rather than erasing the data and then programming in-place. Remap the page to be refreshed is therefore a simple solution already available in commercial NAND flash controllers. Nevertheless, the copy operation introduces and additional write operation whose impact on the flash wear-out must be carefully quantified.

The second solution starts from the assumption that cells with retention problems can be reprogrammed to the value they had before the floating gate lost charge by re-charging additional electrons onto the floating gate through the ISPP algorithm. Only those cells that lost their charge are actually affected but the ISPP algorithm, thus making the impact of this operation on the flash wear-out negligible. One of the main drawbacks of this approach is that, when a flash cell is reprogrammed, additional electrons may be injected into the floating gates of its neighbor cells due to coupling capacitance, thus injecting additional program errors that may increase the overall RBER of the flash. Nevertheless, according to [Cai et al. 2013b] only a large number of accumulated in-place recharging will actually cause problems and at that time remap needs to be triggered. This makes the re-program technique a very valuable solution whenever it is available in the target NAND flash controller.

Both refresh approaches can work with FLARES. If page re-programming is selected, negligible wear-out effects are introduced by FLARES. However, the contribution of the additional program errors introduced by the refresh operation must be carefully analyzed at the technological level in order to properly design the ECC required to sustain the target UBER. In the remaining of this section we will try to quantify the effect of FLARES on the memory endurance if the page remap option is selected.

As a measure of the memory wear-out, we use the *Write Amplification* of an application [Hu et al. 2009]. It is defined as the total number of page write operations performed on the flash (i.e., all write operations including the additional write operations generated by FLARES) divided by the number of write operations issued by the application (i.e., the write operations that would be generated without using FLARES). To evaluate the Write Amplification introduced by FLARES the three selected benchmarks have been first executed in the target virtual environment for a period of 2 days tracking the number read/write/erase operations issued by the application on the flash. It is worth to note here that simulating the benchmarks for a period long enough to generate page refresh commands and to wear-out the memory would be impossible in a limited time frame. To reduce the simulation time, first of all we performed the experiments emulating in the Linux MTD a set of small flash memories ranging from 32MB to 256MB. Small flash memories are easier to wear-out and information

tracked by this execution can be used to extrapolate the behaviour of the application on the target 2GB memory. The list of operations tracked during the execution of the benchmarks has been then used as input for the MATLAB model of FLARES under the assumption of continuous repetitive (steady-state) application behaviour (i.e., the application continuously performs its operation into an infinite loop) and in the worst-case where all pages remain always valid. This enabled us to to project the simulation results over the full flash life-time. Fig. 6a, 6b and 6c show the resulting values of Write Amplification for the three applications.

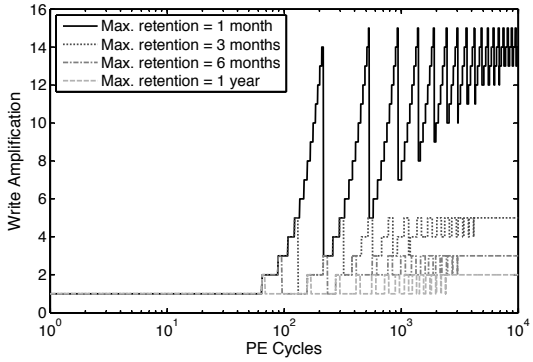
If we consider 1-year retention time, for write intensive or balanced applications (Fig. 6b and 6c), flash pages have a high probability to be rewritten by the application, they therefore in general do not reach the 1 year maximum retention time that would trigger the FLARES page refresh. FLARES therefore does not introduce any Write Amplification[2]. Instead, when considering read intensive applications (Fig. 6a), the average page retention time is slightly larger than the 1-year limit thus triggering some refresh operation. Our experimental data highlighted a low Write Amplification of 2. At the beginning of the memory life, this phenomenon does not happen since the retention contribution on RBER is lower and, related to Eq. 1 characteristics, low correction capability can successfully correct larger RBER ranges.

By analyzing more in detail the obtained results, one can notice that, considering the webserver and varmail applications, the maximum retention time of 1 year is definitely over-designed. Cai et al. [2012] demonstrated that designers can use the maximum retention time coupled with page refresh as an additional design space to trade-off among ECC complexity, flash performance and flash endurance. Following what proposed in Cai et al. [2012] we evaluated the Write Amplification introduced by the page refresh mechanism of FLARES for different retention times lower than 1-year. Results are reported as well in Fig. 6a, 6b and 6c. For often-writing applications (webserver and varmail, in our case), the retention limit can be lowered down to 1 month by only introducing a very limited Write Amplification overhead. Similarly, for seldom-writing applications (videoserver application in our case), an acceptable Write Amplification value can be obtained lowering the maximum retention time down to 3 months. These results are important when compared to the gain of performance one can obtain from this reduced retention constraint.

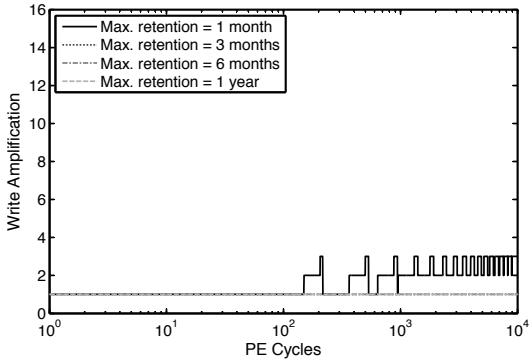
5.2.2. Performances. To conclude the FLARES analysis we evaluated the impact FLARES has on the NAND flash throughput and power consumption.

Fig. 7 reports the throughput of the three considered benchmarks in terms of number of flash operations performed per unit of time (read or program operations). Results are provided for different target retention times and are compared to the performance obtained executing the application without FLARES, using a fixed ECC with correction capability $\mathrm{p} = 50$. Every simulation point reported in Fig. 7 corresponds to data obtained by executing and profiling the benchmark for a period of 6 hours considering different wear-out points. The simulation has been performed resorting to the FLARES architecture implemented in Section 4. During the execution of each benchmark, similarly to what has been done for the validation experiments, errors have been randomly injected in the accessed pages in order to emulate the target RBER for the selected operation point. The injection has been performed resorting to the injection function instrumented in the Linux MTD. Moreover, to take into account the loss of performance due to the FLARES page refresh operations, refresh commands have been randomly generated according to the statistics collected during the wear-out analysis phase reported in the previous section. In order to automate this massive campaign of

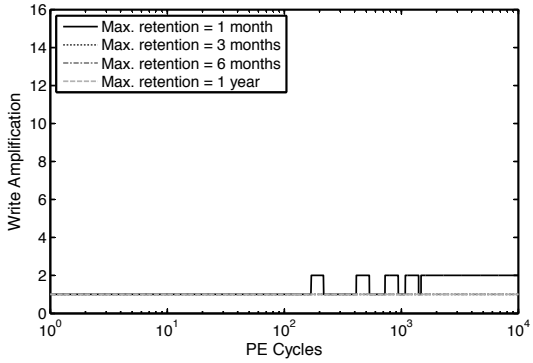
[2]since Write Amplification is a ratio, no contribution means Write Amplification = 1



(a) FLARES Write Amplification for Videoserver application



(b) FLARES Write Amplification for Webserver application



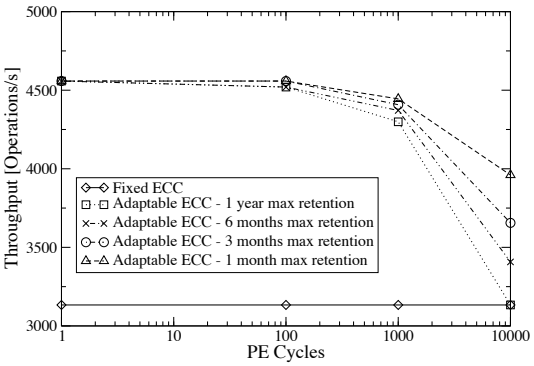
(c) FLARES Write Amplification for Varmail application

Fig. 6. FLARES wear-out impact

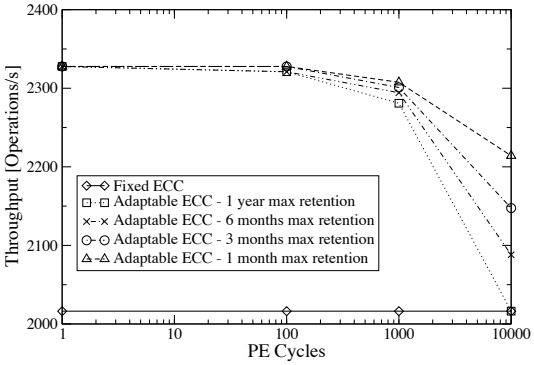
experiments we resorted to EF^{3}S, a framework to evaluate performance of flash based storage systems [**?**]. EF^{3}S, allows to efficiently describe the set of experiments to perform and to automate their execution, providing a set of traces of the performed flash operations with related timing that can be used to evaluate the memory performance.

Exploiting the execution traces collected during the estimation of the throughput of the benchmarks we also estimated the impact of FLARES on the power consumption of the flash subsystem as reported in Fig. 8. The estimated power consumption includes both the contribution of the NAND Flash array and the one of the ECC subsystem estimated after synthesizing its VHDL model in a STM-45nm technology library [cmp.imag.fr 2013].

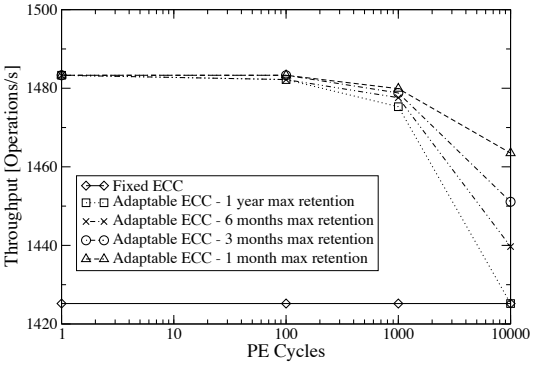
From Fig. 7 and Fig. 8 we can elicit the benefits of FLARES on the throughput and power consumption of all benchmarks when compared to the use of a fixed correction capability ECC. Considering standard 1 year maximum retention, advantages are especially achieved in the early stage of the flash life-time when the flash memory manifests a reduced RBER. As expected, the maximum benefit is observed for the Videoserver application (Fig. 7a) that performs read intensive activities on the flash. ECC decoding is the most computational demanding activity compared to ECC encoding. Reducing the correction capability of the code reduces the ECC decoding time with a significant positive impact on the throughput (which improves of about 50%) and power (which improves of about 10%) of read intensive applications. Neverthe-



(a) Videoserver application throughput



(b) Webserver application throughput



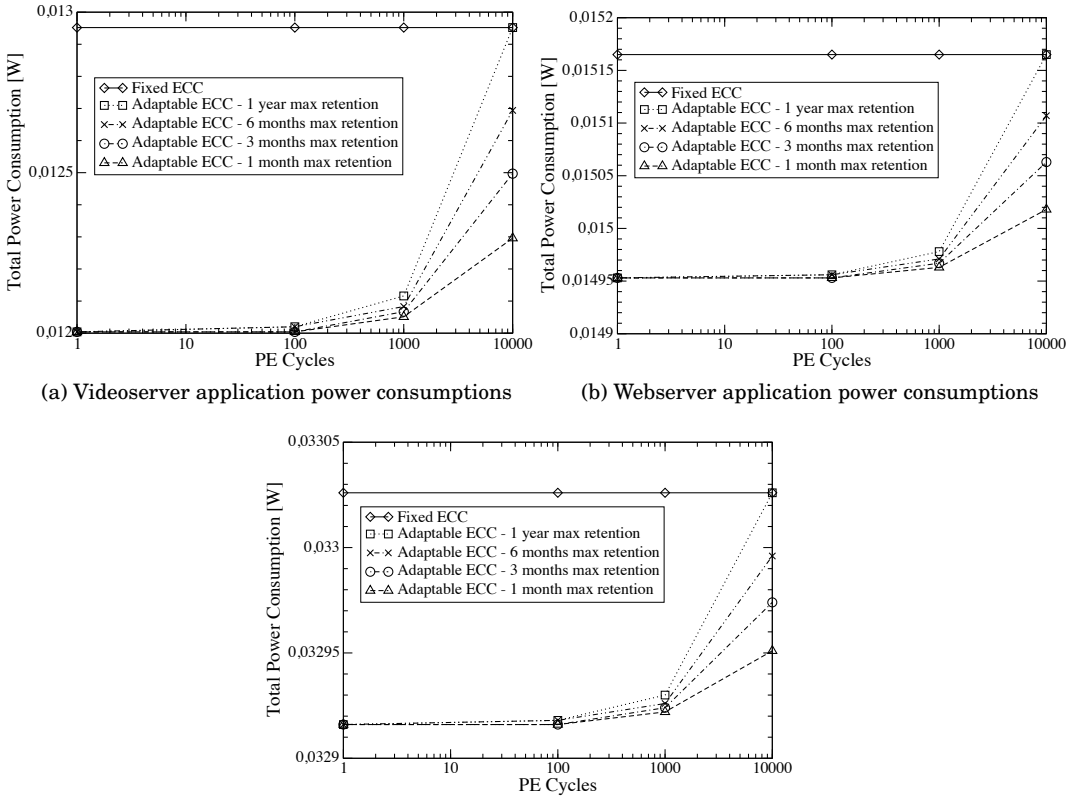
(c) Varmail application throughput

Fig. 7. FLARES throughput experimental results

less, even in the case of write intensive applications such as the varmail benchmark, the throughput at the beginning of the flash life-time is about 5% higher when using FLARES compared to the fixed ECC.

Considering the possibility of reducing the maximum retention time limit, the advantages for all the applications further grow along the whole memory life and especially at the end of the life. Improvements, both in terms of throughput and power consumption, at the end of life are comparable to those at the beginning of life. Given the high performance gain, which comes at very low cost in terms of wear-out, lowering the retention time limit, used together with FLARES, represents an outstanding memory performances boost technique.

Finally, the ability of tuning the ECC correction capability at run-time has also the potential of improving the reliability of the NAND flash in case of unforeseen stress conditions that may rise the flash RBER of a page to unexpected high error rates. In a fixed ECC design the ECC correction capability is designed to meet the target UBER of the application. Over-designing the ECC is always avoided since it would kill the performance of the target application. When introducing FLARES, if the spare area contains enough space to store additional ECC parity bits, designer become free to slightly over-designed the ECC correction capability to account for the possibility of pages that experience very high error rate. This would enable additional reliability that is however only selected when really required without penalizing the performance on pages that behave as expected. The only overhead of this approach stems in the ad-



(a) Videoserver application power consumptions

(b) Webserver application power consumptions

(c) Varmail application power consumptions

Fig. 8. FLARES total power consumptions experimental results

ditional complexity of the ECC hardware, that is however acceptable given the strong scaling of current technology nodes.

6. CONCLUSIONS

In this paper we presented FLARES, an heuristic able to estimate at run-time the best ECC correction capability to apply to each page of a flash based storage system.

FLARES has been fully implemented within the YAFFS 2 filesystem under the Linux operating system. It is therefore ready to be applied in real applicative scenarios. When put at work for real-life workloads, experimental results showed strong improvement in the overall application throughput thus confirming the added value of using FLARES adaptation techniques. Moreover, simulation results also highlighted that FLARES predictions are in general accurate, thus enabling to fit the reliability requirements imposed by real applications.

The capability of FLARES, coupled with an underlying adaptable ECC subsystem holds promise of improving the performance of the flash by carefully tuning the reliability level to the actual wear-out conditions of the flash.

REFERENCES

Lilian Atieno, Jonathan Allen, Dennis Goeckel, and Russell Tessier. 2006. An adaptive Reed-Solomon errors-and-erasures decoder. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. ACM, 150–158.

Raj C. Bose and Dijen K. Ray-Chaudhuri. 1960. On a class of error correcting binary group codes. *Information and Control* 3, 1 (March 1960), 68–79.

Joe E. Brewer and Manzur Gill. 2008. *Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices*. IEEE Press.

Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2012. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*. 521–526. DOI:http://dx.doi.org/10.1109/DATE.2012.6176524

Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2013a. Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '13)*. EDA Consortium, San Jose, CA, USA, 1285–1290. http://dl.acm.org/citation.cfm?id=2485288.2485597

Yu Cai, O. Mutlu, E.F. Haratsch, and Ken Mai. 2013b. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. 123–130. DOI:http://dx.doi.org/10.1109/ICCD.2013.6657034

Yu Cai, G. Yalcin, O. Mutlu, E.F. Haratsch, A. Cristal, O.S. Unsal, and Ken Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*. 94–101. DOI:http://dx.doi.org/10.1109/ICCD.2012.6378623

M. Caramia, M. Fabiano, A. Miele, R. Piazza, and P. Prinetto. 2010. Automated synthesis of EDACs for FLASH memories with user-selectable correction capability. In *Proceedings of IEEE High Level Design Validation and Test Workshop (HLDVT), 2010*. 113 –120. DOI:http://dx.doi.org/10.1109/HLDVT.2010.5496653

Te-Hsuan Chen, Yu-Ying Hsiao, Yu-Tsao Hsing, and Cheng-Wen Wu. 2009. An Adaptive-Rate Error Correction Scheme for NAND Flash Memory. In *Proceedings of 27th IEEE VLSI Test Symposium, 2009 (VTS '09)*. 53 –58. DOI:http://dx.doi.org/10.1109/VTS.2009.24

Yuan Chen. 2011. Flash Memory Reliability NEPP 2008 Task Final Report. (2011). http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41262/1/09-9.pdf

Raghunath Cherukuri. 2010. Agile encoder architectures for strength-adaptive long BCH codes. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. 1900–1904. DOI:http://dx.doi.org/10.1109/GLOCOMW.2010.5700273

cmp.imag.fr. 2013. CMP Project. (2013). Retrieved Dec. 2013 from http://cmp.imag.fr/

Jim Cooke. 2007. The Inconvenient Truths of NAND Flash Memory. In *Flash Memory Summit*. http://download.micron.com/pdf/presentations/events/flash_mem_summit_jcooke_inconvenient_truths_nand.pdf

Stefano Di Carlo, Michele Fabiano, Marco Indaco, and Paolo Prinetto. 2012. ADAGE: An Automated Synthesis tool for Adaptive BCH-based ECC IP-Cores. In *IEEE International Test Conference*. 15.

Stefano Di Carlo, Michele Fabiano, Paolo Prinetto, and Maurizio Caramia. 2011. *Design Issues and Challenges of File Systems for Flash Memories*. inTech, Chapter 1, 3–30.

M. Fabiano, M. Indaco, S. Di Carlo, and P. Prinetto. 2013. Design and optimization of adaptable BCH codecs for NAND flash memories. *Microprocessors and Microsystems* 37, 4–5 (2013), 407 – 419.

J. Gray and C. van Ingen. 2011. Empirical Measurements of Disk Failure Rates and Error Rates. (2011). http://arxiv.org/ftp/cs/papers/0701/0701166.pdf

Edward Grochowski and Robert E. Jr. Fontana. 2012. Future Technology Challenges For NAND Flash And HDD Products. In *Flash Memory Summit*. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_S102A_Grochowski.pdf

Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. 2009. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference (SYSTOR '09)*. ACM, New York, NY, USA, Article 10, 9 pages.

D. Ielmini. 2009. Reliability issues and modeling of Flash and post-Flash memory. *Microelectronic Engineering* 86, 7–9 (2009), 1870–1875.

iozone.org. 2001. IOzone file system benchmark. (2001). Retrieved 2013 from www.iozone.org

Lee Jae-Duk, Hur Sung-Hoi, and Choi Jung-Dal. 2002. Effects of floating-gate interference on NAND flash memory cell operation. *IEEE Electron Device Letters* 23, 5 (May 2002), 264–266. DOI:http://dx.doi.org/10.1109/55.998871

JEDEC. 2010. *Stress-Test-Driven Qualification of Integrated Circuits (JESD47G.01)*. Technical Report. JEDEC Solid State Technology Association.

JEDEC. 2011. *Failure Mechanisms and Models for Semiconductor Devices (JEP122G)*. Technical Report. JEDEC Solid State Technology Association.

Jeffrey Katcher. 1997. PostMark: a new file system benchmark. Network Appliance Tech Report TR3022. (Oct. 1997).

Yan Li and K.N. Quader. 2013. NAND Flash Memory: Challenges and Opportunities. *Computer* 46, 8 (August 2013), 23–29. DOI:http://dx.doi.org/10.1109/MC.2013.190

Rino Micheloni, Luca Crippa, and Alessia Marelli. 2010. *Inside NAND flash memories*. Springer Verlag.

Rino Micheloni, Alessia Marelli, and Roberto Ravasio. 2008. *Error Correction Codes for Non-Volatile Memories*. Springer Publishing Company, Incorporated.

Neal Mielke, Todd Marquart, Ning Wu, Jeff Kessenich, Hanmant Belgal, Eric Schares, Falgun Trivedi, Evan Goodness, and Leland R. Nevill. 2008. Bit error rate in NAND Flash memories. In *Proceedings of the IEEE International Reliability Physics Symposium*. Phoenix, AZ, USA, 9–19. DOI:http://dx.doi.org/10.1109/RELPHY.2008.4558857

Park Mincheol, Kim Keonsoo, Park Jong-Ho, and Choi Jeong-Hyuck. 2009. Direct Field Effect of Neighboring Cell Transistor on Cell-to-Cell Interference of nand Flash Cell Arrays. *IEEE Electron Device Letters* 30, 2 (feb. 2009), 174–177. DOI:http://dx.doi.org/10.1109/LED.2008.2009555

I. S. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960), 300–304.

Moon Kyou Song, Hee-Sun Won, and Min Han Kong. 2002. Architecture for decoding adaptive Reed-Solomon codes with varying block length. In *Consumer Electronics, 2002. ICCE. 2002 Digest of Technical Papers. International Conference on*. 298–299. DOI:http://dx.doi.org/10.1109/ICCE.2002.1014038

spec.org. 2001. SPEC Standard Performance Evaluation Corporation. (2001). Retrieved 2013 from http://www.spec.org

Hairong Sun, Peter Grayson, and Bob Wood. 2011. Qualifying Reliability of Solid-State Storage from Multiple Aspects. In *7th IEEE International Workshop on Storage Network Architecture and Parallel I/O*.

S. Tanakamaru, Y. Yanagihara, and K. Takeuchi. 2013. Error-Prediction LDPC and Error-Recovery Schemes for Highly Reliable Solid-State Drives (SSDs). *Solid-State Circuits, IEEE Journal of* 48, 11 (Nov 2013), 2920–2933. DOI:http://dx.doi.org/10.1109/JSSC.2013.2280078

Andrew Wilson. 2008. The New and Improved FileBench. In *File and Storage Technologies (FAST), 2008. 6th USENIX Conference on*.

E. Yaakobi, L. Grupp, P.H. Siegel, S. Swanson, and J.K. Wolf. 2012. Characterization and error-correcting codes for TLC flash memories. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*. 486–491. DOI:http://dx.doi.org/10.1109/ICCNC.2012.6167470

E. Yaakobi, J. Ma, A. Caulfield, L. Grupp, S. Swanson, P.H. Siegel, and Wolf J.K. 2009. Error Correction Coding for Flash memories. In *Flash Memory Summit*. http://cmrr.ucsd.edu/research/documents/Number31Winter2009_000.pdf

Eitan Yaakobi, Jing Ma, Laura Grupp, Paul H. Siegel, Steven Swanson, and Jack K. Wolf. 2010. Error characterization and coding schemes for flash memories. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. 1856–1860. DOI:http://dx.doi.org/10.1109/GLOCOMW.2010.5700263

yaffs.net. 2007. YAFFS: A Flash file system for embedded use. (2007). Retrieved Feb. 2013 from http://www.yaffs.net/

Chengen Yang, Y. Emre, and C. Chakrabarti. 2012. Product Code Schemes for Error Correction in MLC NAND Flash Memories. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 20, 12 (Dec 2012), 2302–2314. DOI:http://dx.doi.org/10.1109/TVLSI.2011.2174389

C. Zambelli, M. Indaco, M. Fabiano, S. Di Carlo, P. Prinetto, P. Olivo, and D. Bertozzi. 2012. A cross-layer approach for new reliability-performance trade-offs in MLC NAND flash memories. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*. 881 –886.