EF^{3}S: an Evaluation Framework For Flash-based Systems

Di Carlo, S.; Galfano, S.; Indaco, M. and Prinetto, P.
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {*stefano.dicarlo*, *salvatore.galfano*, *marco.indaco*, *paolo.prinetto*}@polito.it

***Abstract*— NAND Flash memories are gaining popularity in the development of electronic embedded systems for both consumer and mission-critical applications. NAND Flashes crucially influence computing systems development and performances. EF^{3}S, a framework to easily assess NAND Flash based memory systems performances (reliability, throughput, power), is presented. The framework is based on a simulation engine and a running environment which enable developers to assess any application impact. Experimental results show functionality of the framework, analysing several performance-reliability trade-offs of an illustrative system.**

I. Introduction

It is expected that the market of electronic embedded systems, mainly fuelled by automotive, consumer electronics, and industrial control systems, will double in size over the next two years, reaching $158.6 billions by 2015 [1].

The non-volatile memory sub-system is a critical component when designing an embedded system, mainly due to strict requirements in terms of reliability, low power consumption, low cost, very fast access, light weight, and long endurance. Flash memories are today a consolidated technology, in terms of non-volatility solutions, for embedded systems development. They can replace power-hungry and unreliable hard disk drives (HDDs) which are impractical in high mechanic shock environments [2].

In the last decade, NAND flash technology faced a rapid improvement, resulting in an outstanding scaling down of the device physical structure up to 20-nm, and a significant reduction of the cost per GB. This has been favoured by the introduction of Double-level Cell (DLC) or Triple-level Cell (TLC) technologies, which store two and three bits per cell, respectively [3], [4]. However, due to the aggressive scaling down of the cell size, current NAND flash memories suffer from several reliability and endurance issues, which limit the device lifetime to about 3k program/erase cycles [5].

In order to overcome this limitation, a variety of combined approaches has been proposed. Error Correction Codes (ECCs), as Bose Ray-Chaudhuri Hocquenghem (BCH) codes [6], [7], [8] and Low-Density Parity-Check (LDPC) codes [9], [10], are systematically applied. Thought ECCs prolong device lifetime, they may be very complex [11], so their impact on write/read latency cannot be neglected. Several techniques for efficient Wear Leveling and Garbage Collection in the Flash Translation Layer (FTL) have been also explored [12], [13], [14]. FTL is typically implemented within specific Flash File Systems. Wear Leveling and Garbage Collection act at a higher level compared to ECCs. They can enhance the overall performances and provide more robust flash devices but, at the same time, they require additional computations to the system, which penalize the throughput.

All entities related to the flash memory, i.e., ECC schemas, FTL and the memory itself, constitute the so-called *Non-Volatile Memory (NVM) subsystem*. Several design alternatives are available to the designers for the implementation of each of these components, and each design choice has a specific impact on the characteristics of the NVM subsystem (e.g., performance, reliability, power consumption, etc.). Moreover, proper design strategies are strictly application dependent. Several trade-offs need to be investigated to best fit demanding application requirements. Designing a cost-efficient flash-based embedded system, and especially an application-specific one, therefore requires a difficult evaluation and faithful exploration of the enumerated design choices that requires the support of a dedicated powerful and flexible design environment.

This paper presents EF^{3}S (Evaluation Framework For Flash-based System), an EDA tool which aims at supporting the design of flash-based systems, targeting different kinds of applications. EF^{3}S offers the possibility of modeling: the physical NAND device, the memory controller, the NAND flash driver, the Flash File System (including wear leveling and garbage collection), and the application workloads. It offers the users a meaningful and synoptic vision (e.g., graphs, tables, punctual metrics) of the features (reliability, throughput, and other statistics) of the modeled NVM subsystem.

EF^{3}S is designed for a Linux based environment featuring a Flash File System (FFS), the Linux Memory Technology Device (MTD) and operating system support for raw flash management, including wear leveling and garbage collection. While the current version is designed to work with the YAFFS2 [15] Flash File System, it is foreseen, in future versions of the framework, to include additional choices such as UBIFS [16] or JFFS2 [17]. The whole framework combines Linux scripts, C programs and Matlab software modules. EF^{3}S enables designers to evaluate the NVM subsystem characteristics resorting to both synthetic workloads and real traces automatically extracted from real applications running in the environment.

Even thought more complex test-bed environments basically targeting Solid State Drives (SSDs) do exist (e.g., FlashSim [18]), the development of EF^{3}S is pushed by the unavailability of a simulation environment that specifically targets

flash-based embedded systems design. Differently from other existing solutions, and, at the best of our knowledge, EF^{3}S is the first tool supporting raw flash based NVM subsystems characterization using non-synthetic workloads.

The rest of this paper is organized as follows: Section II explores the dimensions of the issues of designing a NVM subsystem and overviews the main architectures and features of EF^{3}S. Experimental results from the application of EF^{3}S on a selected use case are reported in Section III. Finally, Section IV concludes the paper proposing future improvements.

II. EF^{3}S Framework

Figure 1 shows the EF^{3}S architecture. EF^{3}S is designed to run into a Linux box and comprises three main modules:

- the *System Configurator*, representing the interface with the user; it is in charge of collecting different design parameters in order to properly configure the target NVM subsystem;
- the *NVM Subsystem Software Stack*, that exploits the Linux kernel and file system stack to provide an execution environment for the target application workload in which operations on the flash memory can be properly profiled. The software characteristics of the target system are among the most difficult parts to model and usually significantly influence the overall system's behaviour. To precisely take all software layers into account, the whole NVM Subsystem Software Stack is set up in the Linux system hosting EF^{3}S, configuring it accordingly to the user inputs while the actual flash memory is emulated;
- the *Simulation Engine*, that elaborates information provided by the System Configurator and the NVM Subsystem Software Stack, simulates the behavior of the target flash memory, and finally outputs the desired statistics. The output of the framework includes information about throughput, power consumptions, reliability, and aging of the considered system. To assist users in design space exploration, combined and synoptic views of the previous features are provided. This helps evaluating trade-offs between NVM subsystem features in order to meet the demanding requirements set by the target applications.

In the next section, the different modules composing EF^{3}S will be analyzed in detail.

A. System Configurator

Designing a NVM subsystem requires the investigation of several design choices, whose characteristics must be specified to EF^{3}S in order to carry out the desired simulations. The *System Configurator* is in charge of collecting these information items. NVM characteristics can be classified into *Hardware Level* and *Software Level* parameters. Going into more details, Software Level design parameters include: the target application and workload, the selected Flash File System, the Wear Leveling algorithm, and the Garbage Collection strategy. Similarly, Hardware Level parameters can be further classified into flash memory device parameters, error correction schemas, and memory interconnection system (i.e., bus). These design choices fully characterize the NVM subsystem and can be provided in input to the system configuration through a set of dedicated configuration files. Some design choices have a limited set of possible options, while others may be freely configured by the user. The universe of the NVM subsystems which can be evaluated by the framework is delimited by all the possible configurations of the design choices. The remainder of this section will detail each specific configuration item available in EF^{3}S.

The workload, i.e., the system application interaction with the NVM subsystem, is one of the most critical elements to properly customize a system to the requirement of specific software applications. Two main parameters allow in EF^{3}S to configure the target workload:

- **Workload Type**. The user may select a set of internally generated workloads produced resorting to the FileBench [19], [20] software suite, or employ a custom workload generated profiling the execution of a custom software application reflecting the specific mission of the simulated system.
- **Workload Running Time**. It is the time for which the workload is executed and profiled. It has to be large enough to perform a significant amount of operations on the system thus collecting enough information about the system's behavior.

Together with the workload, at Software Level the System Configurator also enables to select among the following system's design choices:

- **Flash File System** (FFS). The selected Flash File System and its configuration parameters strongly affect the NVM performance. EF^{3}S is designed to enable simulations using different Linux based Flash Filesystems including YAFFS2 (Yet Another Flash File System 2), UBIFS (UBI File System) [16] or JFFS2 (Journaling Flash File System 2) [17] [1].
- **Garbage Collection** and **Wear Leveling** algorithms can be switched among a selection of available strategies that depend on the selected filesystem.
- **Caches**. In the NVM Subsystem Software Stack several level of caches at the Virtual File System (VFS) and Flash File System (FFS) level are used. These caches can be enabled or disabled to reflect the desired system configuration.

Software Level parameters are used by the System Configurator to properly instrument the NVM Subsystem Software Stack. The System Configurator also manages the execution of the chosen Workload, by executing and profiling it for the chosen Running Time.

At the Hardware Level, EF^{3}S enables the user to configure the NVM subsystem acting on the flash memory, the ECC and the flash interconnection type.

The target flash memory can be configured according to the following parameters:

[1] In the current release YAFFS2 is the only available filesystem, integration of UBIFS and JFFS2 is however under development

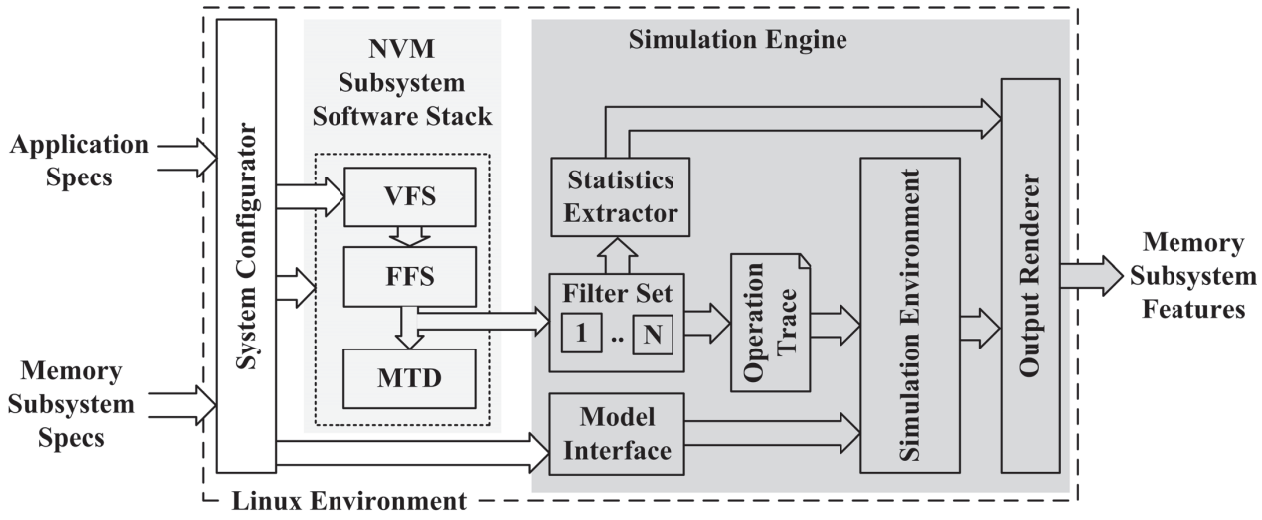


Fig. 1. EF^{3}S Architecture

- **Logical Parameters**. Include page size, block size (or pages per block), device partition size (or blocks per partitions) and total size (or number of device partitions).
- **Operational Parameters**, which may be found on the device datasheet. Such parameters include:
  - device clock frequency;
  - read/program/erase elementary physical operations specifications (e.g., program verify algorithm [21]);
  - elementary operation timing and power consumptions, with the possibility of defining time or aging dependent values;
- **Bare Memory Reliability Model** to be used for reliability status estimation of the memory. Raw Bit Error Rate (RBER) may be assumed as a measure of the reliability status of the bare flash memory. RBER is the fraction of bits that contain incorrect data before applying ECC [22]. An equation, correlating the RBER with the aging of the memory (i.e., the program/erase cycles of each block), is typically used to estimate the reliability status and to model thus physical and circuit level reliability model of the flash memory [7]. A possible choice is to use JEDEC standard [23]. The user may provide (RBER; program/erase cycles) couples, which will be automatically interpolated to fit the JEDEC model equation, or a completely new model equation.

All provided parameters may be time or memory aging dependent and can be specified as piecewise defined functions. This means that values are obtained from different functions, each one defined on different sub-intervals delimited either by program/erase cycles values or by time intervals (e.g., flash operation latencies strongly depend on the aging of the memory). Piecewise cubic polynomials may be specified by the user to accurately approximate them into the model.

Together with the target flash memory, the ECC subsystem can be also fully configured according to the following parameters:

- **ECC family**, chosen between BCH, LDPC, Reed Solomon (RS)[24] and Product Codes[25];
- **correction capability** or **target reliability**, measured as UBER (Uncorrectable Bit Error Rate) value, i.e., the error rate after applying ECC [22];
- **message length**, with the possibility to apply the code to portions of a page;
- **adaptability**, i.e., a fixed correction capability ECC schema versus a variable one can be chosen [21], [26];
- encoding/decoding **latency and power consumptions** or **implementation technology**, in terms of technology node used to synthesize the ECC, and possibly running clock frequency upper limit.

BCH solution has been characterized using a cycle accurate simulation of implemented codecs and offers higher flexibility, while other ECC families configuration relies on data extracted from literature. Further configurations will be characterized in following developments of EF^{3}S.

Finally, the Interconnection System is characterized by the Flash **Interconnection Type** with the rest of the system, i.e., the particular topology of interconnection and protocol.

B. NVM Subsystem Software Stack

The NVM Subsystem Software Stack comprises the basic Linux software modules required to deal with flash based storage systems: VFS, FFS and MTD.

The VFS role is to hide the specific Flash File System to the application thus offering a standard set of functions to the application. The Flash File System, is the core of the NVM Subsystem Software Stack in which stored information

consistency is granted. The Flash File System usually embeds specific Wear Leveling and Garbage Collection. Finally the MTD [27] is a driver, which provides a uniform interface to different (raw) flash chips. The MTD, if properly instrumented, emulates the presence of a raw NAND flash chip, providing thus the capability to run the NVM Subsystem Software Stack within the hosting Linux system, even when no physical memory devices are available.

In EF^{3}S, all elements of the NVM Subsystem Software Stack are configured and instrumented according to the configuration parameters provided to the System Configurator. In particular, the Flash File System is instrumented to produce a log file of all flash operations required by the target application workload. This log file is pivotal to analyze the system's behavior and provide output statistics.

C. Simulation Engine

The Simulation Engine is responsible for simulating the configured NVM subsystem.

The **Filter Set** is the main interface of the simulation engine with the NVM Subsystem Software Stack. It contains a set of filters, based on regular expressions, which are applied to the log file produced by the Flash File System and the Linux Kernel during the execution of the target application. The log file contains a huge amount of negligible information (e.g., information about the mounting of the flash memory, pointers to read/written data) that must be properly filtered before performing the required simulation. Moreover, several information items in the log that are written in a human readable way and must be removed. After filtering, a sequence of operations (hereinafter referred to as operation trace) is produced. Each operation is described according to the following formal:

$$\langle time \rangle \ \langle r/w\,(/e) \rangle \ \langle page\,(/block) \ address \rangle$$

where: $\langle time \rangle$ is the time when the operation was issued; $\langle r/w\,(/e) \rangle$ represents the operation type (r stands for read, w for write, e for erase); $\langle page\,(/block) \ address \rangle$ is the operation target page or block address.

The **Model Interface** acts as an interface between the System Configurator and the Simulation Environment. It elaborates the hardware specific configuration parameters in order to setup the simulation of the computed operation traces. In particular, this module evaluates the piecewise functions defining the Flash Operational Parameters: the right subinterval is identified and expressions are evaluated. The model interface also evaluates the reliability model equations. Finally, the Model Interface module sets the ECC characteristics and correction capability required to satisfy specifications and computes the ECC run-time parameters such as encoding/decoding latencies and power, resorting to previously characterized ECC schema.

Simulation Environment is the core module of the simulation engine. Here all data are combined. The operations to simulate are read from the operation trace. Each operation is emulated using the parameters that are concurrently updated by the Model Interface, according to the status of the simulated flash memory. At the same time, the Simulator Environment estimates and collects the relevant NVM subsystem features (e.g., average operation latency and throughput, average power and energy per operation, and aging of the pages) that are then collected and provided in output. Of course, produced output accuracy stems from the input models and data accuracy. The **Statistics Extractor** works in parallel to the Simulation Environment. By solely analyzing the input operation traces, this module extracts statistics about the Flash memory utilization, like number of read, write or erase operations per page or block, write/read intensity (ratio between read and write operations) and total number of operations.

Finally, the **Output Renderer** manages the data produced by the Simulator Environment and the Statistics Extractor, to offer a meaningful and synoptic vision of the reliability, performance and behavior of the simulated system. Primary outputs of the performed analysis are are Throughput, Power Consumptions, Reliability, and Aging. Rendered data, instead, include graphs and tables. Several interesting aspects/relations have been selected to be displayed, to better show the trade-offs involved in the design field:

- UBER vs. program/erase cycles;
- average latency or average read, write or application throughput or average power or average energy per operation vs. program/erase cycles;
- UBER vs. average latency or average read, write or application throughput or average power or average energy per operation;
- Aging vs. block address.

III. Experimental Results

In order to prove its effectiveness, EF^{3}S has been used to analyze a case study in which the impact of run-time ECC adaptation on the NVM performance and reliability must be evaluated. A NVM subsystem with fixed correction capability ECC is compared with one having a variable correction capability ECC.

A. Experimental Setup

In order to obtain the desired comparison, the framework has been used with two different configurations, differing in the ECC adaptivity setting. All remaining parameters have been configured in the same way, and two framework runs have been performed.

We used an internally generated workload based on FileBench benchmark behaviour emulating a Video Server, simulating the system for a Running Time of 10 minutes. The simulated system exploits YAFFS2 as FFS. Both VFS and YAFFS2 caches have been disabled.

We emulated a NAND Flash memory (with one partition), having 8192 blocks of 64 pages, with a page size of 4 kB. Its timings and power consumptions are summarized in Table I and are referred to a standard ISPP programming algorithm [21].

TABLE I. NAND Flash memory specifications

Operation	Timing (Average)	Power
Read	75µs	40mW
Program	680µs	813mW
Erase	3ms	813mW

The reliability model used to evaluate the RBER is the default one, i.e.:

$$RBER(peCycles) = A \cdot peCycles^{B} + C \qquad (1$$

where A, B and C were obtained by fitting the equation towards RBER values provided by the factory, and $peCycles$ are the program/erase cycles experienced by the page. Model was fitted to actual measures of RBER of a real memory chip (of the same kind of the emulated one) across its life.

For both system configurations a target UBER of 10^{-1} was set and a BCH code was applied to an entire page of the flash. For the fixed ECC schema configuration, the correction capability, t, was automatically set to meet the target UBER constraint even in the worst case, i.e., at the end of memory life (100k program/erase cycles). For the adaptable ECC schema, correction capability was instead chosen at every step of the memory life according to the punctual RBER estimation. Characterized default ECC schema were already implemented [21] and characterized considering a STM-45nm [28] Synopsys technology library and the ECC to be running at 100MHz clock frequency.

All simulations have been performed on a workstation equipped with an Intel Core i5 460m CPU, with 1GB RAM running Ubuntu 10.4 operating system.

B. Outputs Demonstration and Comparison

The Video Server behaves as a read intensive application i.e., the number of read operations prevails over the number of write operations (with a ratio of 1084.70). During 10 minutes, 457,138 read, 42,410 write, and 6 erase operations were performed.

Figure 2 compares the ultimate reliability (UBER) behaviour of the two solutions. Clearly, adaptable ECC solution follows more closely the target UBER, while fixed ECC

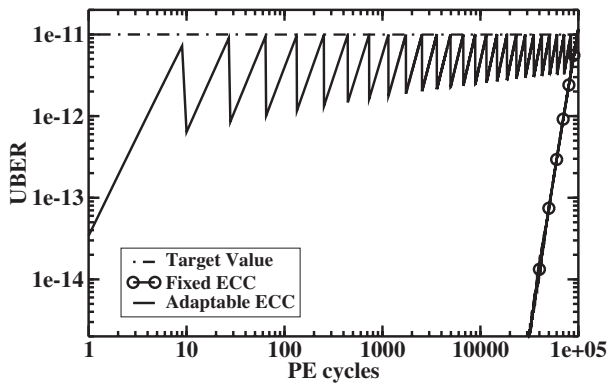
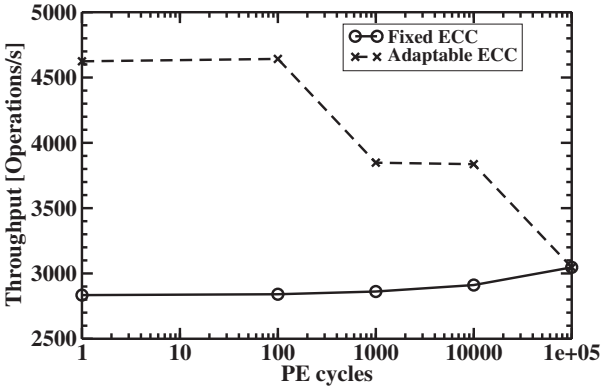


Fig. 2. UBER vs. program erase cycles

Figure 3 clearly shows the throughput advantages of adaptable ECC w.r.t. a fixed ECC schema. Figure 4 depicts the energy per operation trends across memory life, showing also power consumptions advantages of the adaptable solution.



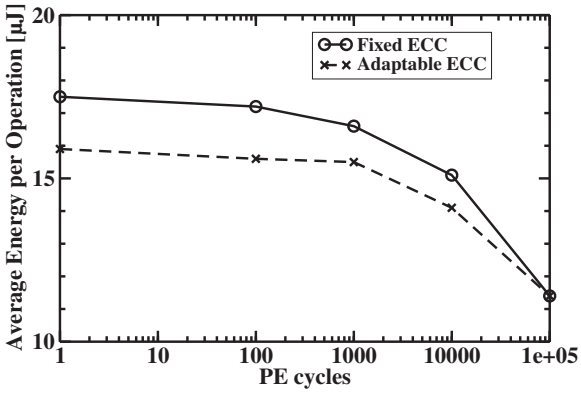


Fig. 4. Energy per Operation

Figure 5 represents a snapshot of the wear-out, i.e., program/erase (PE) cycles, of each flash memory block during the workload running time: used blocks are quite evenly distributed.

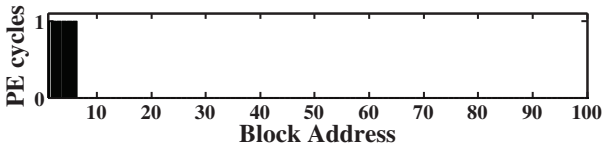


Fig. 5. Wear-out snapshot

C. Framework performance

The framework was developed considering to be as lightest as possible. Engine is the core of the computations. Trace filtering is the most data intensive operation. Such an elaboration, for a 10 minutes profiled trace of the above experiment, originates an output trace file of about 10 MB (it depends on the profiled application activity). For the rest of the engine

computations, instead, marginal time (less than one second) and memory resources (4MB) are employed.

IV. Conclusion

This paper presented EF^{3}S, an easy-to-use, highly configurable, and modular framework to assess a NAND flash based NVM subsystem performance (reliability, throughput, power, aging). The main architecture and characteristics of the framework have been presented and its application on a selected case study has been provided in order to show its effectiveness and the information the tool may provide.

EF^{3}S is continuously under development. Extended support for other several Flash File Systems is underway. Moreover, in its current release, the EF^{3}S is not aware of the specific data being stored in the flash memory. Adding data-awareness will enable to derive further insights about reliability and performances data dependency.

EF^{3}S is available for research purposes to interested readers and can be downloaded at http://www.testgroup.polito.it/index.php/component/k2/item/204-ef3s .

References

[1] M. King, "Embedded systems market to experience CAGR of 72013. [Online]. Available: http://www.companiesandmarkets.com/News/Information-Technology/Embedded-systems-market-to-experience-CAGR-of-7/NI3284

[2] M. Jedrak, "NAND flash memory in embedded systems," Accessed, Feb 2013. [Online]. Available: http://www.design-reuse.com/articles/24503/nand-flash-memory-embedded-systems.html

[3] G. Marotta, A. Macerola, A. D'Alessandro, A. Torsi, C. Cerafogli, C. Lattaro, C. Musilli, D. Rivers, E. Sirizotti, F. Paolini, G. Imondi, G. Naso, G. Santin, L. Botticchio, L. De Santis, L. Pilolli, M. Gallese, M. Incarnati, M. Tiburzi, P. Conenna, S. Perugini, V. Moschiano, W. Di Francesco, M. Goldman, C. Haid, D. Di Cicco, D. Orlandi, F. Rori, M. Rossini, T. Vali, R. Ghodsi, and F. Roohparvar, "A 3bit/cell 32Gb NAND flash memory at 34nm with 6MB/s program throughput and with dynamic 2b/cell blocks configuration mode for a program throughput increase up to 13MB/s," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, feb. 2010, pp. 444 –445.

[4] T. Futatsuyama, N. Fujita, N. Tokiwa, Y. Shindo, T. Edahiro, T. Kamei, H. Nasu, M. Iwai, K. Kato, Y. Fukuda, N. Kanagawa, N. Abiko, M. Matsumoto, T. Himeno, T. Hashimoto, Y.-C. Liu, H. Chibvongodze, T. Hori, M. Sakai, H. Ding, Y. Takeuchi, H. Shiga, N. Kajimura, Y. Kajitani, K. Sakurai, K. Yanagidaira, T. Suzuki, Y. Namiki, T. Fujimura, M. Mui, H. Nguyen, S. Lee, A. Mak, J. Lutze, T. Maruyama, T. Watanabe, T. Hara, and S. Ohshima, "A 113mm2 32Gb 3b/cell NAND flash memory," in *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, feb. 2009, pp. 242 –243.

[5] Z. Wang, M. Karpovsky, and A. Joshi, "Nonlinear multi-error correction codes for reliable MLC NAND flash memories," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 7, pp. 1221 –1234, july 2012.

[6] H. Choi, W. Liu, and W. Sung, "VLSI implementation of BCH error correction for multilevel cell NAND flash memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 843–847, 2010.

[7] R. Micheloni, A. Marelli, and R. Ravasio, *Error Correction Codes for Non-Volatile Memories*. Springer Publishing Company, 2008.

[8] M. Caramia, M. Fabiano, A. Miele, R. Piazza, and P. Prinetto, "Automated synthesis of EDACs for FLASH memories with user-selectable correction capability," *Proceedings of IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pp. 113 –120, june 2010.

[9] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in nand flash memory," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, no. 2, pp. 429 –439, feb. 2011.

[10] J. Kim, J. Cho, and W. Sung, "A high-speed layered min-sum LDPC decoder for error correction of NAND flash memories," in *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, aug. 2011, pp. 1 –4.

[11] S. Di Carlo, M. Fabiano, R. Piazza, and P. Prinetto, "EDACs and test integration strategies for NAND flash memories," in *Design Test Symposium (EWDTS), 2010 East-West*, 2010, pp. 218–221.

[12] S. Di Carlo, M. Fabiano, P. Prinetto, and M. Cramia, *Design Issues and Challenges of File Systems for Flash Memories*. InTech, 2011, ch. 1, pp. 3–30.

[13] S. Mylavarapu, S. Choudhuri, A. Shrivastava, J. Lee, and T. Givargis, "FSAF: File system aware flash translation layer for NAND flash memories," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 399 –404.

[14] M.-L. Chiao and D.-W. Chang, "ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation," *Computers, IEEE Transactions on*, vol. 60, no. 6, pp. 753 –766, june 2011.

[15] "YAFFS: A Flash file system for embedded use," Accessed, Feb 2013. [Online]. Available: http://www.yaffs.net/

[16] "UBIFS - UBI File-System," Accessed, Feb 2013. [Online]. Available: http://www.linux-mtd.infradead.org/doc/ubifs.html

[17] "JFFS: The Journalling Flash File System," Accessed, Feb 2013. [Online]. Available: http://linux-mtd.infradead.org/~dwmw2/jffs2.pdf

[18] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, sept. 2009, pp. 125 –131.

[19] "Filebench," web available resource - http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Filebench, 2012.

[20] Wilson, "The new and improved filebench," in *File and Storage Technologies (FAST), 2008. 6th USENIX Conference on*, 2008.

[21] C. Zambelli, M. Indaco, M. Fabiano, S. Di Carlo, P. Prinetto, P. Olivo, and D. Bertozzi, "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*, march 2012, pp. 881 –886.

[22] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill, "Bit error rate in NAND flash memories," in *Reliability Physics Symposium, 2008. IRPS 2008. IEEE International*, 27 2008-may 1 2008, pp. 9 –19.

[23] JEP122G, "Failure mechanisms and models for semiconductor devices," JEDEC Solid State Technology Association, Tech. Rep., 2011.

[24] B. Chen, X. Zhang, and Z. Wang, "Error correction for multi-level NAND flash memory using reed-solomon codes," in *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*. IEEE, 2008, pp. 94–99.

[25] C. Yang, Y. Emre, C. Chakrabarti, and T. Mudge, "Flexible product code-based ECC schemes for MLC NAND flash memories," in *Signal Processing Systems (SiPS), 2011 IEEE Workshop on*, 2011, pp. 255–260.

[26] M. Fabiano, M. Indaco, S. Di Carlo, and P. Prinetto, "Design and optimization of adaptable BCH codecs for NAND flash memories," *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, In Press, 2013.

[27] "Linux Memory Technology Devices," Accessed, Feb 2013. [Online]. Available: http://www.linux-mtd.infradead.org/index.html

[28] "CMP project," web available resource - http://cmp.imag.fr/, 2012.