Solid-State Drive (SSD): A Nonvolatile Storage System

BY RINO MICHELONI
Microsemi Corporation, Sunnyvale, CA, USA

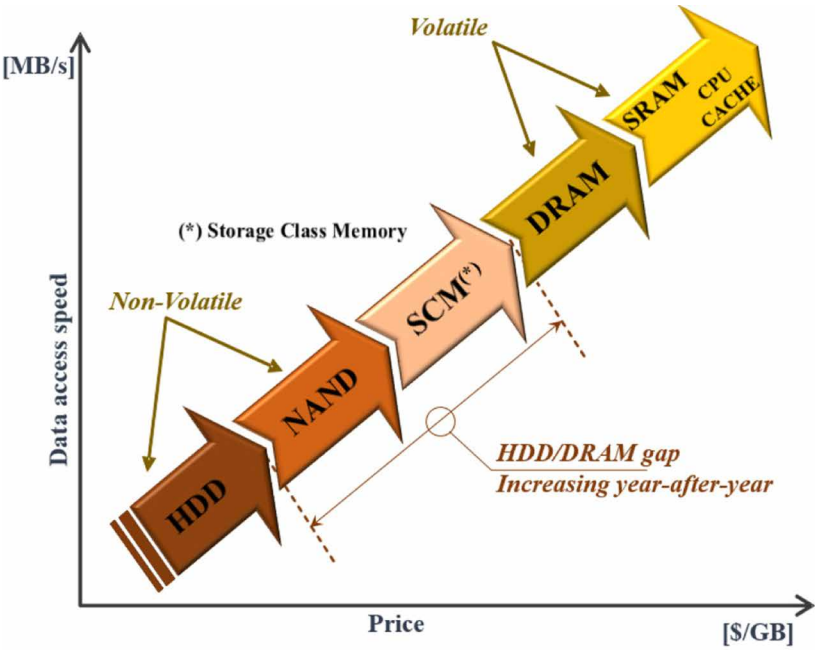

Fig. 1. ***Memory hierarchy.***

I. WHY SOLID-STATE DRIVES

Over the last 15 years, NAND Flash memories have changed our lives: Flash cards [mainly in the secure digital (SD) form factor] have almost completely replaced photographic films, and USB keys have driven floppy disks to extinction. Lately, thanks to a great tradeoff between cost and performance (i.e., write/read speed), NAND Flash technology has begun fighting against hard disk drives (HDDs) in the form of solid-state drives (SSDs).

In a nutshell, HDDs [1] can be seen as electromechanical devices because the information is stored on a spinning disk, covered with ferromagnetic material. A motor drives the spinning disk while a moving actuator arm has to tightly control the position of the magnetic head in charge of writing and reading to/from the storage media. The simple fact that there is a rotating disk implies that random access is limited by the mechanical movement of the disk; reaching a different area of the spinning plate in less than a millisecond is definitely tough. Modern applications such as financial transactions, data mining, machine learning, and cloud computing need very fast access to stored data and HDDs are not the best fit for them. Moreover, the mechanical parts pose a major constraint on reducing the HDD form factor and they also represent a major source of power consumption.

Smartphones and tablets have played a key role in looking for something different from HDDs because portable applications absolutely need less power-hungry and lighter storage devices. But this is not the only reason. Historically, if we focus only on access speed to stored data, DRAMs have greatly outpaced HDDs, thus creating a big gap in the so-called memory hierarchy, which is shown in Fig. 1. It was exactly this gap that opened the door to newcomers in the storage infrastructure, which, in the old days, was an exclusive domain of HDDs (and tapes). The aforementioned gap in read and write performances is now so big that even NAND Flash memories cannot fill it entirely. The gap between DRAM and NAND is supposed to be covered by a new class of memories called storage class memories (SCMs). Both industry and academia are placing a lot of effort in identifying and developing these new memories. Magnetic RAM (MRAM), Resistive RAM (ReRAM), carbon nanotubes, and 3-D XPoint are some of the leading SCM candidates.

Digital Object Identifier: 10.1109/JPROC.2017.2678018

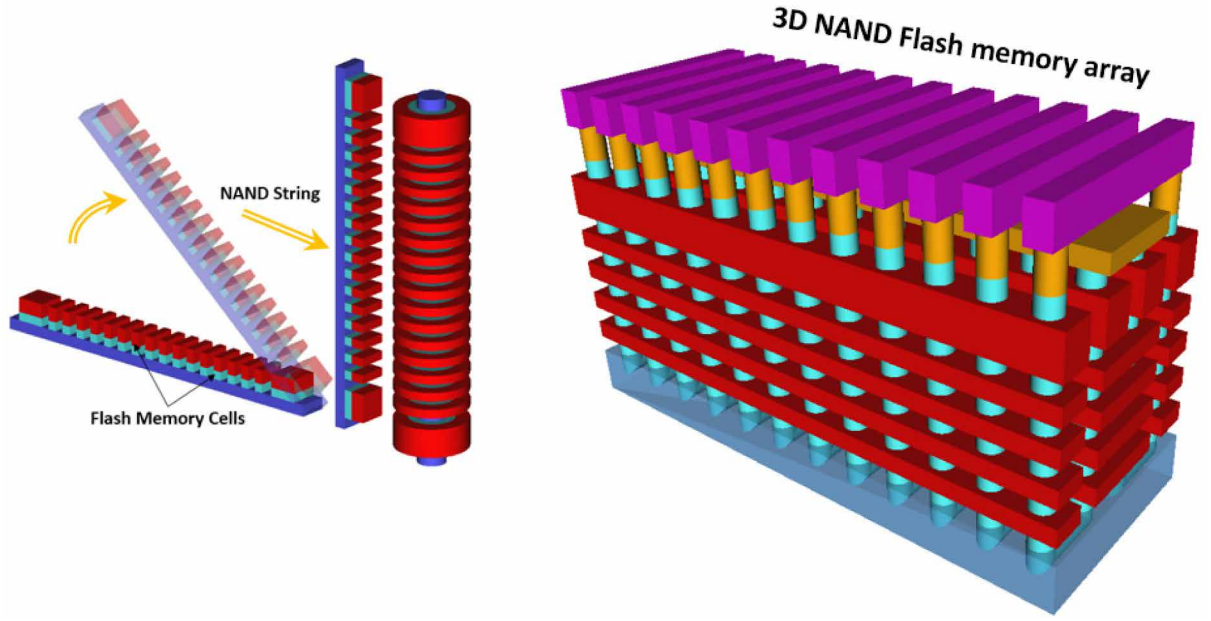

Fig. 2. ***Three-dimensional NAND Flash memory array: NAND strings go from planar (left) to vertical (right).***

II. FLASH TECHNOLOGY

Let us now take a closer look at NAND technology [2]. Flash memories are solid-state devices; in other words, they are "simple" pieces of silicon without any moving mechanical parts. Just because of that, there is no need for a motor, which greatly improves access speed to stored data by itself. As a rule of thumb, a NAND memory can be read in less than 100 μs. More importantly, there is no difference between sequential and random access as there is no sensing head that needs to move across the silicon die. Being much faster than HDDs, SSDs have moved the speed bottleneck from the storage side to the host side. Legacy storage interfaces such as serial ATA (SATA) and serial attached SCSI (SAS) are now running out of steam and this is why faster "computing" interfaces such as PCI Express (PCIe) are gaining momentum in the storage market.

In terms of capacity, a single NAND die can now store up to 512 Gb and a single package can contain up to 16 dies in a 12-mm ×18-mm footprint. At this point, SSDs can really challenge HDDs in most of the applications. This massive storage density improvement has been enabled by two main technologies.

- 3-D (vertical) integration (Fig. 2) [3]. NAND memory cells can be vertically stacked to form multiple memory layers within the same silicon die. The most recent devices have 64 layers but memories with more than 100 layers are expected to come in the near future.
- Multilevel storage (Fig. 3). Flash storage is built around the ability of trapping and detrapping electrons inside a metal–oxide–semiconductor (MOS) transistor. In practice, the population of trapped electrons acts as an electrostatic shield and it ends up modifying the transistor's threshold voltage. By carefully modulating the amount of electrons, multiple threshold voltages can be generated and translated into the digital domain. For instance, eight voltage values will result in 3 b of digital information. Based on the number of voltage levels, NAND memories can be classified as follows:
  - SLC: two threshold voltages, 1 b per memory cell;
  - MLC: four threshold voltages, 2 b per memory cell;
  - TLC: eight threshold voltages, 3 b per memory cell;
  - QLC: 16 threshold voltages, 4 b per memory cell.

The aforementioned 512-Gb devices are based on TLC storage but QLC devices are under development and they are expected to reach the market in the coming few years.

III. SSDs BLOCK DIAGRAM

When we open the case of an SSD, we find a complete system inside; human eyes can just see part of it, the hardware (HW) one, but the firmware (FW) part is as important. Let us start from what we can immediately see. A simplified block diagram of a typical SSD's HW is shown in Fig. 4. Of course, there are plenty of NAND Flash memories, but the microcontroller is definitely the brain of the system. It is also common to find other components such as:

- direct-current-to-direct-current (dc–dc) converters to derive all the necessary internal power supplies;

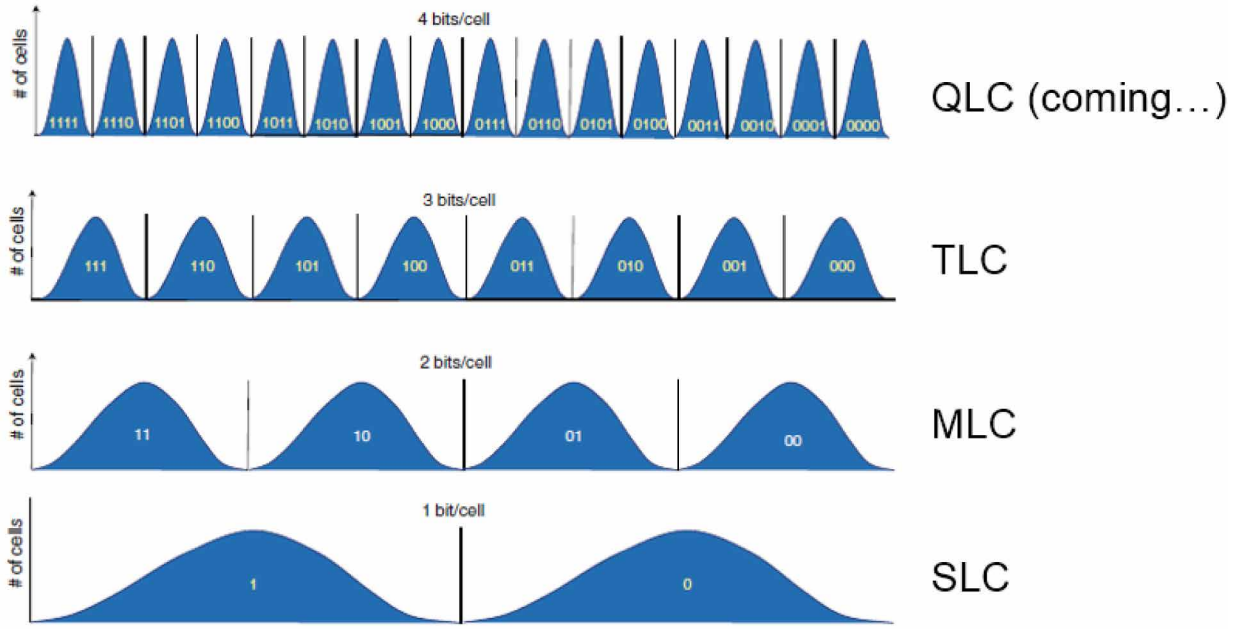

Fig. 3. *NAND classification based on how many bits are stored per physical memory cell.*

- quartz crystals for high precision clocks;
- filter capacitors for filtering power supplies;
- a network of temperature sensors for power management (for instance, if the temperature becomes too high, performances can be throttled not to exceed SSD's power budget);
- fast DRAM components are used for data caching: when the system host issues a write operation to the drive, data are actually first cached to reduce the transfer time seen by the host, and then copied to the Flash subsystem.

At a very high level, the SSD's microcontroller (or simply Flash controller) needs to take care of the following tasks [4]:

- communication to/from the system host;
- communication to/from the Flash subsystem by using the selected electrical interface and protocol (e.g., ONFI or Toggle);
- communication to/from DRAM subsystem;
- read/write performances;
- data integrity during all data transfers, and retention of the stored nonvolatile information (which is very sensitive to temperature).

Generally speaking, activities of Flash controllers can be grouped into six modules, which can be implemented either in hardware or in firmware, depending on design choices and target performances (Fig. 5).

The first module connects the drive to the host system (host interface in the block diagram of Fig. 5). In other words, it enables the physical connection between host and SSD based on the selected protocol (e.g., PCIe, SAS, SATA, etc.), thus ensuring both logical and electrical interoperability. Usually, this block is made of HW (e.g., buffers, drivers, etc.) and FW (e.g., one of the Cores is used to decode the command sent by the Host). When host commands are decoded, the second module, the *Flash interface*, kicks in. In essence, this second module translates all the decoded commands into low-level instructions for the NAND subsystem. Again, the controller needs to guarantee the electrical interoperability with NAND devices. The two most popular commercial NAND protocols are called ONFI and Toggle, and they are capable of transferring data in DDR mode up to 800 MB/s (1 GB/s and beyond might be possible in future generations). Another electrical interface (the third module, DRAM interface) that needs to be handled by the Flash controller is the one toward DRAM components which is mainly used for data caching and for storing the mapping tables required by the FTL (see below).

The fourth module is the Flash file system (FFS) [5]; the main goal here is to make an SSD look like a standard HDD to the host, main reason being the possibility of reusing all the existing applications, not necessarily developed having in mind the specific properties of the solid-state storage. Typical FFS implementation is FW based, as sketched in Fig. 5. There are four main FW layers: Flash translation layer (FTL), wear leveling, garbage collection, and bad block management.

In order to understand why there is a need for such a complex FW

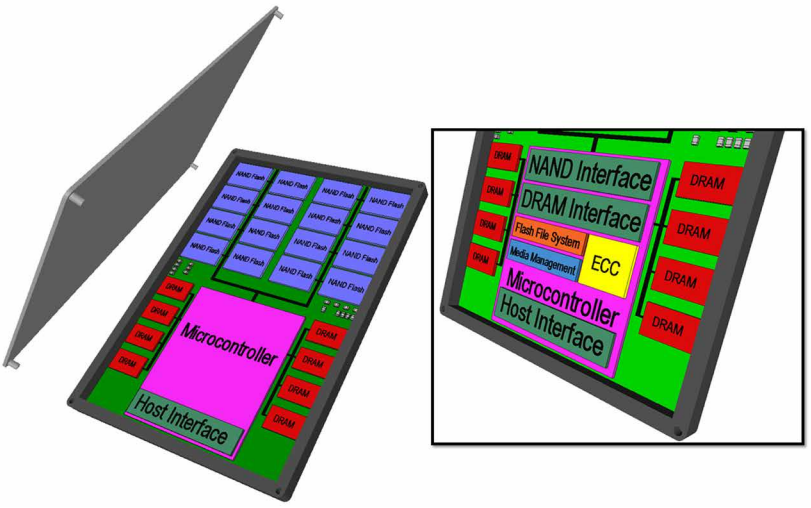

Fig. 4. ***Solid-state drive: Block diagram.***

infrastructure, we first need to dig a little bit deeper in how Flash memories store data. Flash arrays start from the so-called "erased" state where all bits are set to "1." Write (also known as Program) operations can change the state of each single bit to "0"; in other words, writing is selective at the bit level. Unfortunately, we cannot state the same for the "erase" operation, i.e., the operation that brings the digital value back from "0" to "1": erase can only act on group of cells called "blocks." The whole Flash memory array is split into thousands of blocks, and each of them is made by hundreds (or thousand) of pages. Today, each page is 16 kB long: read and write operations work on pages, in the sense that the user read and write data patters of 16 kB in parallel. Because of this very unique storage functionality, a "simple" page update is actually not that simple. In fact, a page update implies changing some of the bits from "0" to "1" and this operation, in Flash terms, means erasing. The point is that a single erase operation involves multiple pages and it can take several milliseconds to complete. Because it would take too long, what actually happens is that the updated page gets written to a different memory location and the original page (i.e., the page that needed to be updated) gets invalidated. As a consequence, there is a mismatch between physical and logical page addresses. This misalignment can be fixed by using tables to store the logical-to-physical mapping, and this is the so-called Flash translation layer (FTL). The number of tables can be huge and it can have a significant impact on the effective SSD's storage capacity, if it is not carefully designed.

Operation after operation (especially write and erase operations), NAND Flash memories wear out in the sense that it becomes more and more difficult to precisely control the number of electrons trapped inside memory cells; in other words, it becomes harder to generate the number of threshold voltages required for multilevel storage (MLC, TLC, or QLC). Therefore, it is critical to spread operations across the memory array as much as possible. Wear leveling algorithms are designed to accomplish this goal by leveraging the above-described concept of logical-to-physical translation. When the system host wants to update a specific page within a specific block, the Flash controller dynamically maps the new content to a different block. Wear leveling algorithms are in charge of deciding which of the available blocks to pick. There are two possible strategies. Dynamic wear leveling looks for the block with the lowest erase count,

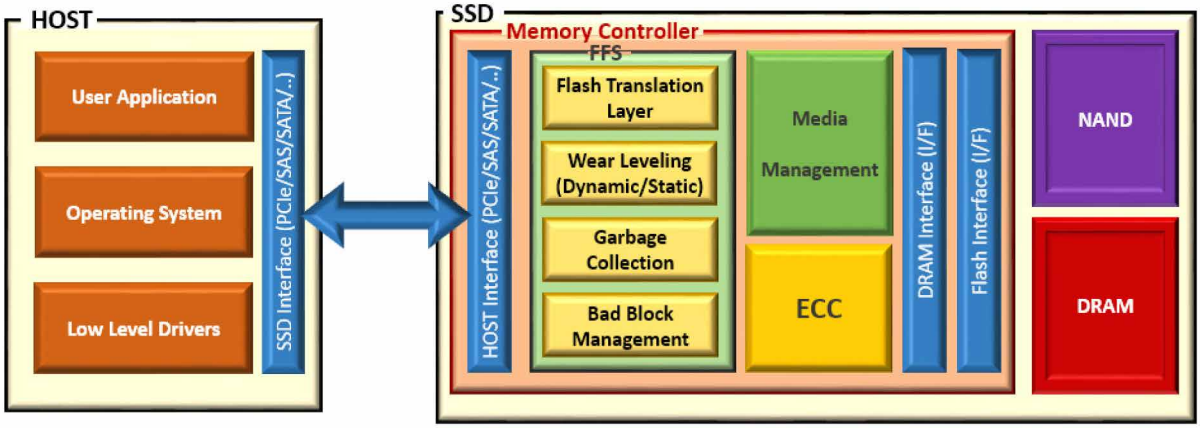

Fig. 5. ***Functional view of a Flash controller.***

while static wear leveling chooses among blocks whose erase count deviates from the average, even if they have been recently erased. Wear leveling needs a pool of "free" (i.e., erased) blocks. When the population of this pool goes below a threshold, a FW layer called garbage collection takes over. This algorithm selects the block that needs to be erased based on a predefined cost function; it copies the entire content to a different block, and then it triggers the erase operation such that the block can be moved to the list of available blocks. Usually, garbage collection is a background operation to avoid any performance (throughput and latency) hit; in other words, write and especially read operations have higher priorities compared to erase operations, which take a much longer time to complete. Of course, given the same SSD's workload, the bigger is the memory capacity, the lower is the number of operations that each cell has to experience.

The fourth FW layer, the bad block management (BBM), takes care of the so-called bad blocks (BB): these blocks contain unreliable cells and, therefore, cannot be used to store data. NAND Flash devices contain BBs when they are shipped from the factory, but new bad blocks can pop during SSD's lifetime as a result of failures during either programming or erasing. BBM keeps track of BBs in a dedicated list, which has to be nonvolatile. In fact, this list has to be retrieved at every boot of the drive (power-up) to avoid storing user data inside unreliable memory cells.

Let us now go back to Fig. 4. The fifth module inside the Flash controller is the one performing the error recovery, i.e., the error correction code (ECC) [6]. Historically, Bose–Chaudhuri–Hocquenghem (BCH) code has been the code used to enhance NAND reliability, mainly because its HW implementation is relatively simple. Most recently, low-density parity-check (LDPC) codes [7] have caught a lot of attention because they can get much closer to the Shannon limit. With Flash technology moving to high 3-D stacks and QLC coming in few years, the possibility of correcting more errors becomes very attractive. LDPC codes leverage complex algorithms, require more logic gates, and, therefore, consume more power. Therefore, there is a big research activity in this space trying to find the right tradeoff between correction performances and HW cost.

Last but not least, in Fig. 4 we have the media management module. By media we mean either NAND Flash or any other emerging nonvolatile memories (e.g., ReRAM, MRAM, etc.). Given the fact that LDPC codes are approaching the Shannon limit, there is not so much space left for improving the SSD's lifetime by "simply" increasing the number of errors that the Flash controller can recover. As a result, there is the need for reducing the BER growth rate such that the ECC maximum correction capability is reached at a higher count of program/erase cycles, as sketched in Fig. 6. When looking at Flash technology, all the techniques used to mitigate the NAND raw BER fall under the term Flash signal processing (FSP) [8]: data randomization and read oversampling (also known as read retry) are popular examples of these techniques. Therefore, the media management module is in charge of executing FSP.

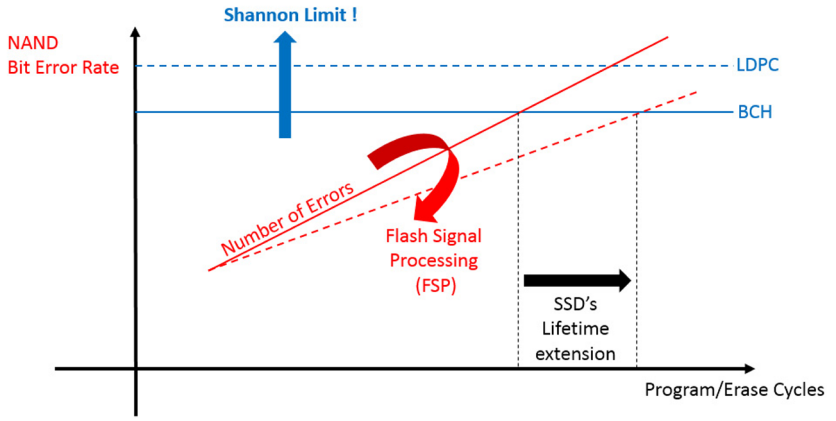

Fig. 6. ***Flash signal processing is used to mitigate NAND BER growth.***

IV. HYBRID SSDS

Most recently, both industry and academia have increased their research effort in the hybrid memory management space, developing a wide variety of systems. Actually, "hybrid" is a generic term because it can have different meanings depending on the context. For instance, at the system level, storage can be hybrid because it combines HDDs and SSDs together. A single SSD can be hybrid because of two reasons:

- it embeds different types of NAND memories: SLC and TLC, SLC and QLC, etc.;
- it combines different nonvolatile memories such as NAND and ReRAM, MRAM, PCM, etc.

Of course, the combination of different memories in the same system boosts the complexity to a completely different level, both in terms of firmware and data management. Indeed, in order to exploit all the benefits of the different memories, applications running on the system host side have to carefully decide where it is more convenient to store a particular set of data. This opens the box to the concept of "data temperature": data are defined as "hot," "warm," and "cold"

depending on how frequently they are updated and accessed.

V. CONCLUSION

Solid-state drives are changing the way people store and process data, but SSDs are very complex systems to build because they require a sophisticated mix of hardware, software, and firmware. On top of that, nonvolatile memories can be of different types, involving totally different storage mechanisms, each of them with its own reliability challenges. All of the above considerations imply tens of billions of dollars spent in R&D worldwide each year, with engineers from all over the places scratching their heads to solve very complex problems: mathematics, physics, circuit design, process technology, manufacturing, lithography, signal processing, and testing techniques are all called to give their contribution to drive the evolution of SSDs even further. ■

REFERENCES

[1] T. Zhang, G. Mathew, H. Zhong, and R. Micheloni, "Modern hard disk drive systems: Fundamentals and future trends," in *Memory Mass Storage*, G. Campardo, F. Tiziani, M. Iaculo, Eds. New York, NY, USA: Springer-Verlag, 2011, ch. 4.

[2] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*. New York, NY, USA: Springer-Verlag, 2010.

[3] R. Micheloni, Ed., *3D Flash Memories*. New York, NY, USA: Springer-Verlag, 2016.

[4] R. Micheloni, A. Marelli, and K. Eshghi, *Inside Solid State Drives (SSDs)*. New York, NY, USA: Springer-Verlag, 2013.

[5] R. Micheloni, M. Picca, S. Amato, H. Schwalm, M. Scheppler, and S. Commodaro, "Nonvolatile memories for removable media," *Proc. IEEE*, vol. 97, no. 1, pp. 148–160, Jan. 2009.

[6] R. Micheloni, A. Marelli, and R. Ravasio, *Error Correction Codes for Non-Volatile Memories*. New York, NY, USA: Springer-Verlag, 2008.

[7] N. Xie, W. Xu, T. Zhang, E. F. Haratsch, and J. Moon, "Concatenated low-density parity-check and BCH coding system for magnetic recording read channel with 4 kB sector format," *IEEE Trans. Magn.*, vol. 44, no. 12, pp. 4784–4789, Dec. 2008.

[8] B. Shin, C. Seol, J.-S. Chung, and J. J. Kong, "Error control coding and signal processing for flash memories," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seoul, South Korea, May 2012, pp. 409–412.