Rino Micheloni
Luca Crippa
Alessia Marelli

# Inside NAND Flash Memories

Springer

Vervain Ex. 2003 p. 1 of 36
Vervain v. Phison
PGR2024-00048

# Inside NAND Flash Memories

Rino Micheloni • Luca Crippa • Alessia Marelli

# Inside NAND Flash Memories

Springer

Vervain Ex. 2003 p. 3 of 36
Vervain v. Phison
PGR2024-00048

Rino Micheloni
Integrated Device Technology
Agrate Brianza
Italy
rino.micheloni@ieee.org

Luca Crippa
Forward Insights
North York
Canada
luca.crippa@ieee.org

Alessia Marelli
Integrated Device Technology
Agrate Brianza
Italy
alessiamarelli@gmail.com

ISBN 978-90-481-9430-8 e-ISBN 978-90-481-9431-5
DOI 10.1007/978-90-481-9431-5
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2010931597

© Springer Science+Business Media B.V. 2010
No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

*Cover design*: eStudio Calamar S.L.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

However, NAND exhibits fast page writes due to the ability to write 4–8 kB simultaneously resulting in very high sequential write throughput. The serial architecture and small cell size make NAND Flash optimized for low cost mass storage whereas NOR Flash is optimized for performance code storage and execution.

## 1.3 Multi-bit per cell storage

### 1.3.1 Memories scaling

Where other semiconductor memories were on a 2 year cadence for new process technology introduction, NAND Flash memories have historically been on a 1 year cadence. This accelerated process scaling resulted in the bit size of SLC NAND overtaking MLC NOR in 2005 as shown in Fig. 1.6. MLC NAND is by far the lowest cost semiconductor memory with none of the memory technologies even close to being cost competitive. This is mainly due to the very small cell size combined with multi-level cell capability.

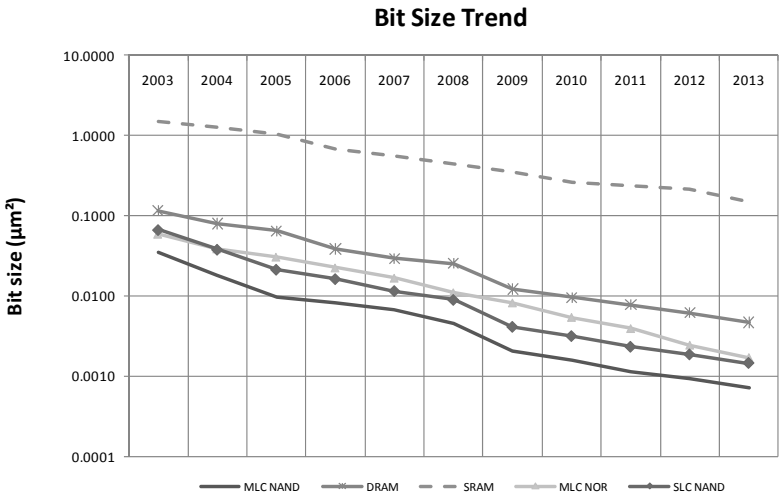


**Fig. 1.6.** Scaling (Source: Forward Insights)

### 1.3.2 Multi-level cell concept

Figure 1.7 illustrates the concept of multi-level cell storage in Flash memories. Conventional SLC or single-level cell storage distinguishes between a '1' and '0' by having no charge or charge present on the floating gate of the Flash memory cell. By increasing the number of charge or voltage threshold (Vt) levels, more

than 1-bit per cell may be stored. Two bits per cell (MLC) storage is enabled by increasing the number of Vt levels to four representing 11, 10, 01 and 00. Similarly by increasing the number of voltage threshold levels to eight and 16, 3-bit per cell and 4-bit per cell storage is enabled.

The benefit of multi-level cell storage is that storage capacity may be increased without a corresponding increase in process complexity. The same fab equipment used to manufacture silicon wafers for SLC products may be used to manufacture MLC, 3-bit per cell and 4-bit per cell devices. However, multi-level cell storage requires accurate placement of the Vt charge levels so that the charge distributions don't overlap as well as accurate sensing of the different charge levels. As the number of Vt levels increases the time it takes for accurate programming and sensing increases. Additional circuitry and programming algorithms are necessary to ameliorate the degradation of the performance and endurance of such devices.

However in effect, transitioning from SLC to MLC to 3-bit to 4-bit per cell technology is equivalent to a partial shrink of the device from one process technology generation to the next without additional capital investment.

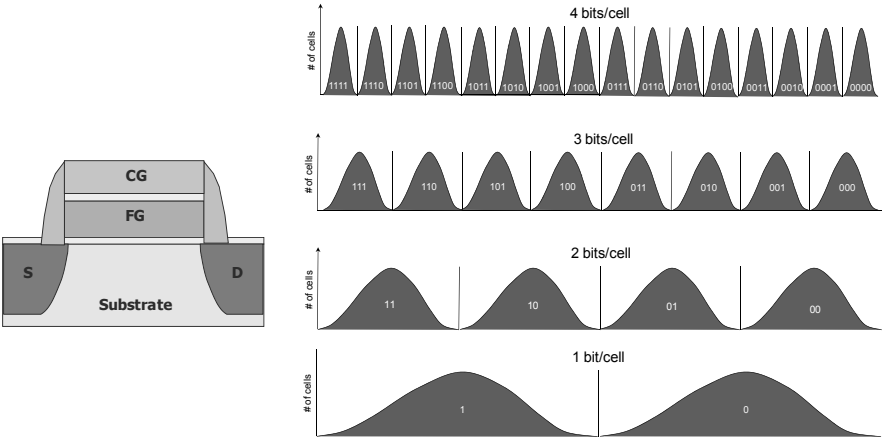


**Fig. 1.7.** Multi-level storage in floating gate NAND Flash memory

### 1.3.3 NAND scaling

Figure 1.8 shows the bit size trend for SLC, MLC, 8LC (3-bit per cell) and 16LC (4-bit per cell) technologies. The bit size is a proxy for the cost. The first MLC NAND Flash chip was introduced at the end of 2001 by SanDisk and Toshiba. It was a 1 Gb chip based on 0.16 µm process technology. Subsequent MLC generations were introduced on an almost yearly basis.

The first commercial production of 8LC began in 2008 also by SanDisk and Toshiba. The device, a 16 Gb product based on 56 nm process technology was introduced at a process technology one generation behind the mainstream MLC

products. As a result, the device was short-lived since the MLC products based on 43 nm technology were more cost-competitive than the 8LC product. However in 2010, the cost benefits of 8LC technology are expected to be realized as 8LC is manufactured on the same process technology as MLC.

As with 8LC, 16LC also from SanDisk and Toshiba was introduced in 2009 on a mature process technology – 43 nm. As a result, the first product – a 64 Gb chip – is not cost competitive with the mainstream 32 nm MLC products. It is mainly being used as a learning vehicle. Only when 16LC can be manufactured on the leading edge process technology will the full cost benefits be realized.

Note that the cost per bit reduction becomes progressively smaller as one transitions from SLC to MLC to 8LC to 16LC. An approximate 40–50% reduction can be obtained moving from SLC to MLC but this figure drops to 20% for MLC to 8LC and 10% for 8LC to 16LC. As a result, the economic benefit of 16LC may not be enough to justify the additional design efforts to implement it.

One of the main scaling challenges is that the number of electrons stored in the floating gate is decreasing significantly with each process generation. This has consequences for the sensing of the data value stored and data retention. These issues combined with inter-cell interference will make 16LC a less scalable technology. To overcome these challenges, innovative programming algorithms and signal processing techniques filter the signal from the noise will be required for future generations of NAND Flash.

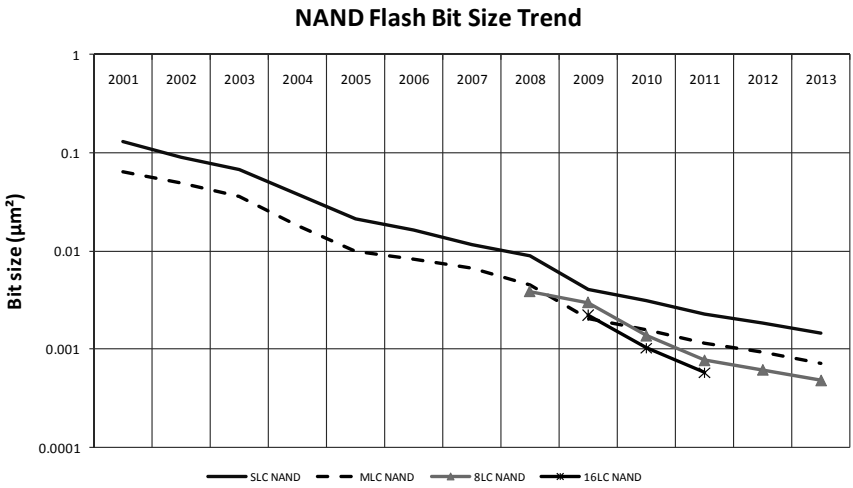


**Fig. 1.8.** NAND Flash bit size trend (Source: Forward Insights)

### 1.3.4 Capacity

Until 2006, MLC chip density was doubling every year. This doubling of chip density is being extended to 18 months to 2 years as the scaling challenges

increase. As MLC became mainstream, SLC began to lag behind in process migration and density transition. For example, it will take 3 years for the transition from 8 to 16 Gb SLC chips.

The benefit of 8LC and 16LC is the ability to enable monolithic chip densities which would not be possible with SLC and MLC. For example, the first 16LC device was the first monolithic chip with a capacity of 64 Gb. It is expected that 8LC and 16LC will continue to lead the introduction of the highest capacity devices (Fig. 1.9).

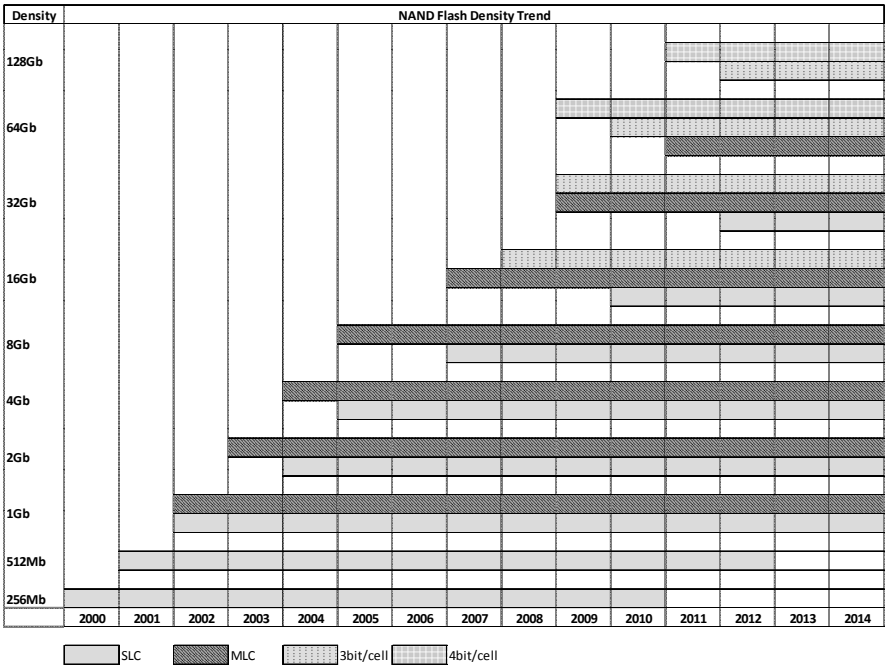


**Fig. 1.9.** NAND Flash density roadmap (Source: Forward Insights)

### 1.3.5 Device characteristics

The lower costs associated with multi-level cell technology does not come for free. The lower cost comes at the expense of reduced performance and endurance. A comparison of the device characteristics for SLC, MLC, 3- and 4-bit per cell is summarized in Fig. 1.10. Significant degradation in endurance, retention and write performance occurs with increasing number of bits per cell. Read performance and operating current are also negatively impacted.

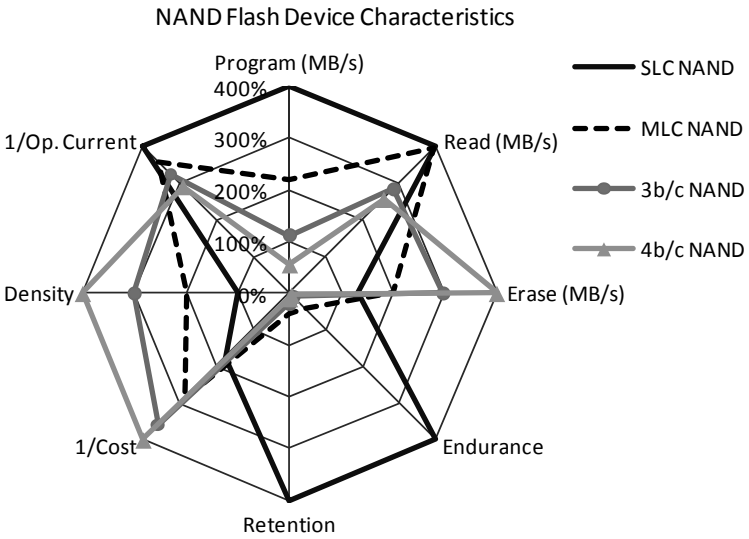


**Fig. 1.10.** NAND device characteristics (Source: Forward Insights)

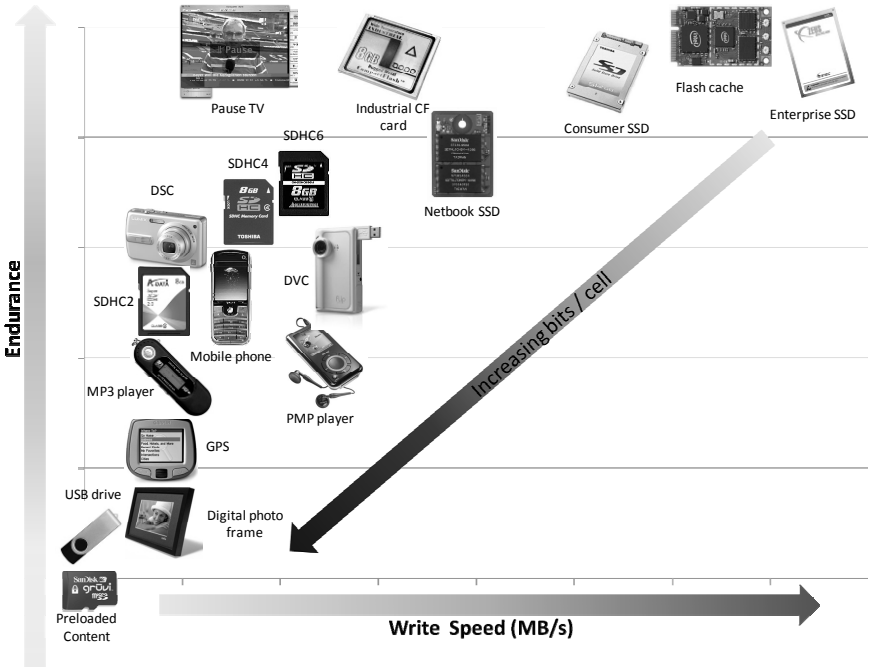


**Fig. 1.11.** Applications for multi-bit per cell NAND Flash memories (Source: Forward Insights)

The most popular Flash memory cell is based on the *Floating Gate* (FG) technology, whose cross section is shown in Fig. 2.1. A MOS transistor is built with two overlapping gates rather than a single one: the first one is completely surrounded by oxide, while the second one is contacted to form the gate terminal. The isolated gate constitutes an excellent "trap" for electrons, which guarantees charge retention for years. The operations performed to inject and remove electrons from the isolated gate are called program and erase, respectively. These operations modify the threshold voltage $V_{TH}$ of the memory cell, which is a special type of MOS transistor. Applying a fixed voltage to cell's terminals, it is then possible to discriminate two storage levels: when the gate voltage is higher than the cell's $V_{TH}$, the cell is on ("1"), otherwise it is off ("0").

It is worth mentioning that, due to floating gate scalability reasons, charge trap memories are gaining more and more attention and they are described in Chap. 5, together with their 3D evolution.

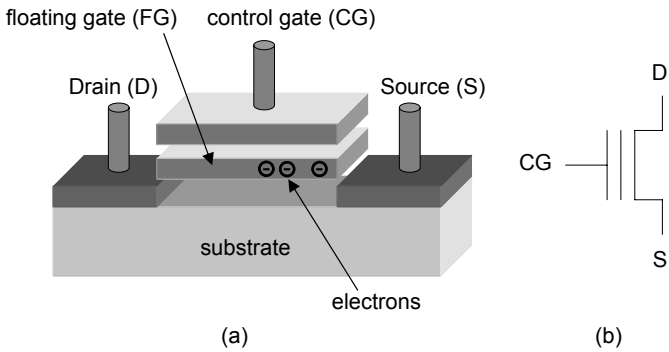


**Fig. 2.1.** (a) Floating gate memory cell and (b) its schematic symbol

## 2.2 NAND memory

### 2.2.1 Array

The memory cells are packed to form a matrix in order to optimize silicon area occupation. Depending on how the cells are organized in the matrix, it is possible to distinguish between NAND and NOR Flash memories. This book is about NAND memories as they are the most widespread in the storage systems. NOR architecture is described in great details in [1].

In the NAND string, the cells are connected in series, in groups of 32 or 64, as shown in Fig. 2.2. Two selection transistors are placed at the edges of the string, to ensure the connections to the source line (through $M_{SL}$) and to the bitline (through $M_{DL}$). Each NAND string shares the bitline contact with another string. Control gates are connected through wordlines (WLs).

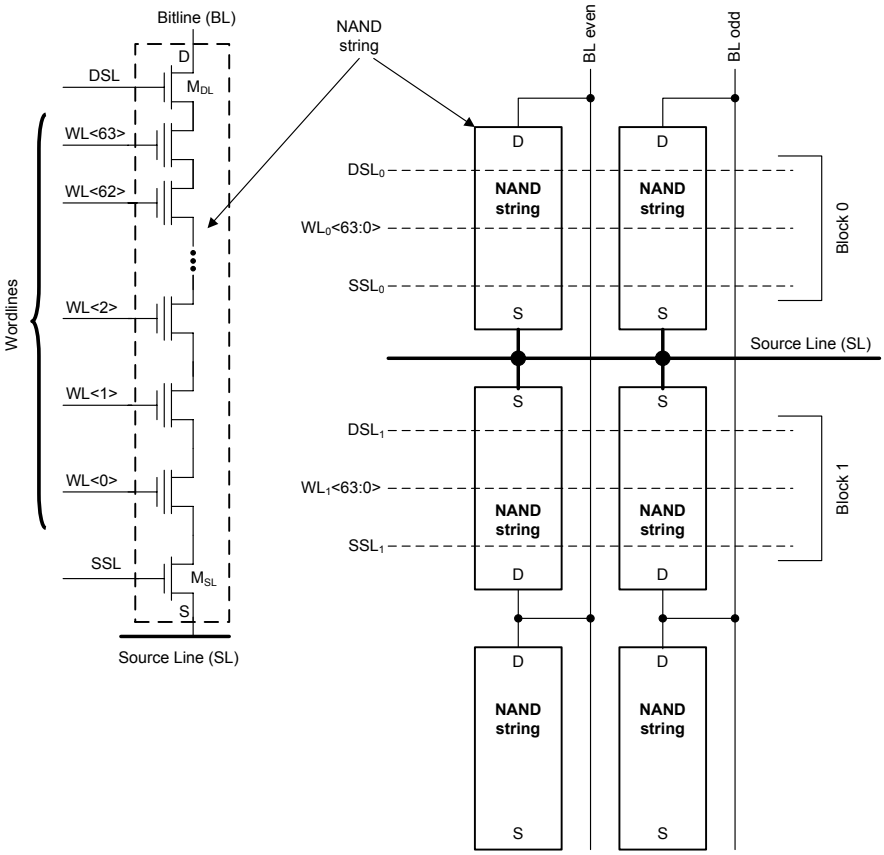


**Fig. 2.2.** NAND string (left) and NAND array (right)

Logical pages are made up by cells belonging to the same wordline. The number of pages per wordline is related to the storage capabilities of the memory cell. Depending on the number of storage levels, Flash memories are referred to in different ways: SLC memories store 1 bit per cell, MLC memories (Chap. 10) store 2 bits per cell, 8LC memories (Chap. 16) store 3 bits per cell and 16LC memories (Chap. 16) store 4 bits per cell.

If we consider the SLC case with interleaved architecture (Chap. 8), even and odd cells form two different pages. For example, a SLC device with 4 kB page has a wordline of 65,536 cells.

Of course, in the MLC case there are four pages as each cell stores one *Least Significant Bit* (LSB) and one *Most Significant Bit* (MSB). Therefore, we have:

– MSB and LSB pages on even bitlines
– MSB and LSB pages on odd bitlines

All the NAND strings sharing the same group of wordlines are erased together, thus forming a Flash block. In Fig. 2.2 two blocks are shown: using a bus representation, one block is made up by $WL_0$<63:0> while the other one includes $WL_1$<63:0>.

NAND Flash device is mainly composed by the memory array. Anyway, in order to perform read, program, and erase additional circuits are needed. Since the NAND die must be inserted in a package with a well-defined size, it is important to organize all the circuits and the array in the early design phase, i.e. it is important to define a floorplan.

In Fig. 2.3 an example of a floorplan is given. The Memory Array can be split in different planes (two planes in Fig. 2.3). On the horizontal direction a Wordline is highlighted, while a Bitline is shown in the vertical direction.

The Row Decoder is located between the planes: this circuit has the task of properly biasing all the wordlines belonging to the selected NAND string (Sect. 2.2.2). All the bitlines are connected to sense amplifiers (Sense Amp). There could be one or more bitlines per sense amplifier; for details, please, refer to Chap. 8. The purpose of sense amplifiers is to convert the current sunk by the memory cell to a digital value. In the peripheral area there are charge pumps and voltage regulators (Chap. 11), logic circuits (Chap. 6), and redundancy structures (Chap. 13). PADs are used to communicate with the external world.

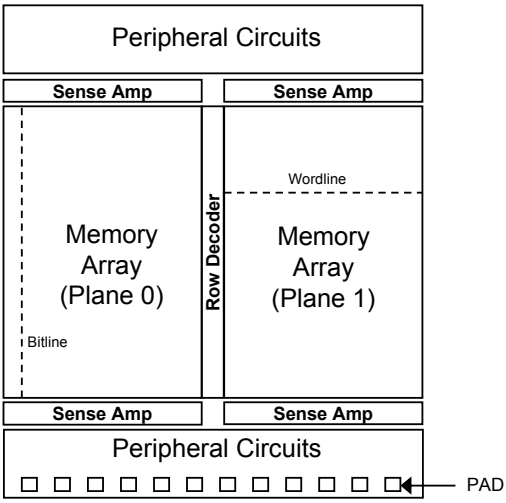


**Fig. 2.3.** NAND Flash memory floorplan

## 2.2.2 Basic operations

This section briefly describes the basic NAND functionalities: read, program, and erase.

It can be noted that only the DQS pin has been added to the standard ASI. In this case, higher speeds are achieved increasing the toggling frequency of RE#.

Figure 2.30 shows a “Cmd” sequence, followed by “Dout” cycles for Toggle-Mode interface.

## 2.4 NAND-based systems

Flash cards, USB sticks and *Solid State Disks* (SSDs) are definitely the most known examples of electronic systems based on NAND Flash.

Several types of memory cards are available on the market [3–5], with different user interfaces and form factors, depending on the needs of the target application: e.g. mobile phones need very small-sized removable media like µSD. On the other hand, digital cameras can accept larger sizes as memory capacity is more important (CF, SD, MMC). Figure 2.31 shows different types of Flash cards.

The interfaces of the Flash cards (including USB sticks) support several protocols: parallel or serial, synchronous or asynchronous. Moreover, the Flash cards support hot insertion and hot extraction procedures, which require the ability to manage sudden loss of power supply while guaranteeing the validity of stored data.

For the larger form factors, the card is a complete, small system where every component is soldered on a PCB and is independently packaged. For example, the NAND Flash memories are usually available in TSOP packages. It is also possible to include some additional components: for instance, an external DC-DC converter can be added in order to derive an internal power supply (CompactFlash cards can work at either 5 or 3.3 V), or a quartz can be used for a better clock precision. Usually, reasonable filter capacitors are inserted for stabilizing the power supply. Same considerations apply to SSDs; the main difference with Flash cards is the system capacity as shown in Fig. 2.32 where multiple NANDs are organized in different channels in order to improve performances.

For small form factors like µSD, the size of the card is comparable to that of the NAND die. Therefore, the memory chip is mounted as bare die on a small substrate. Moreover, the die thickness has to be reduced in order to comply with the thickness of µSD, considering that several dies are stacked, i.e. mounted one on top of each other.

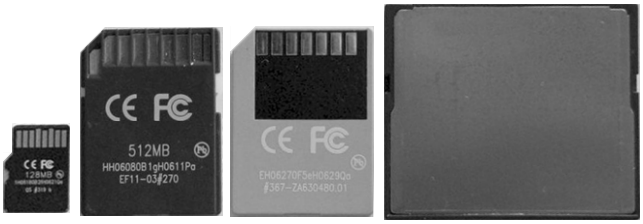


**Fig. 2.31.** Popular Flash card form factors

All these issues cause a severe limitation to the maximum capacity of the card; in addition external components, like voltage regulators and quartz, cannot be used. In other words, the memory controller of the card has to implement all the required functions.

The assembly stress for small form factors is quite high and, therefore, system testing is at the end of the production. Hence, production cost is higher (Chap. 15).

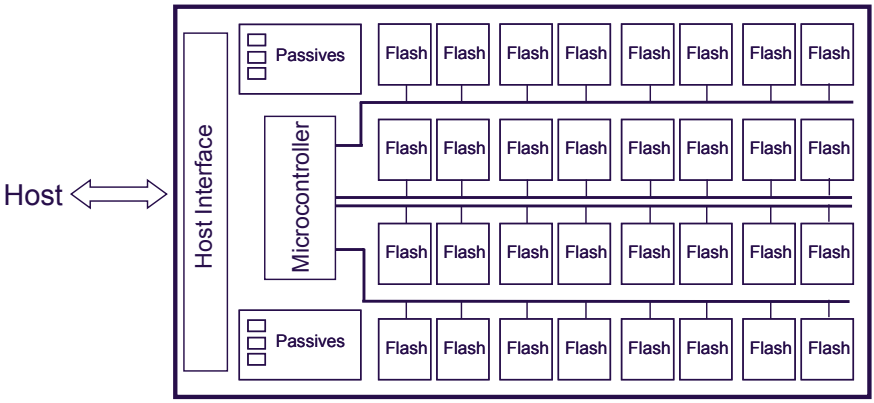


**Fig. 2.32.** Block diagram of a SSD

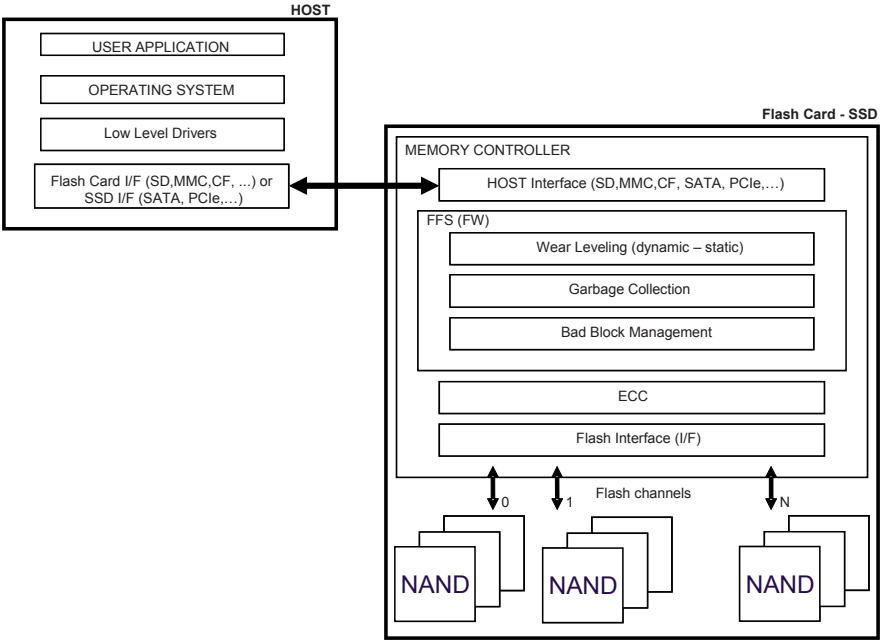


**Fig. 2.33.** Functional representation of a Flash card (or SSD)

For a more detailed description of Flash cards, please, refer to Chap. 17. SSDs are described in Chap. 18.

Figure 2.33 shows a functional representation of a memory card or SSD: two types of components can be identified: the memory controller and the Flash memory components. Actual implementation may vary, but the functions described in the next sections are always present.

### 2.4.1 Memory controller

The aim of the memory controller is twofold:

1. To provide the most suitable interface and protocol towards both the host and the Flash memories
2. To efficiently handle data, maximizing transfer speed, data integrity and information retention

In order to carry out such tasks, an application specific device is designed, embedding a standard processor – usually 8–16 bits – together with dedicated hardware to handle timing-critical tasks.

For the sake of discussion, the memory controller can be divided into four parts, which are implemented either in hardware or in firmware. Proceeding from the host to the Flash, the first part is the host interface, which implements the required industry-standard protocol (MMC, SD, CF, etc.), thus ensuring both logical and electrical interoperability between Flash cards and hosts. This block is a mix of hardware – buffers, drivers, etc. – and firmware – command decoding performed by the embedded processor – which decodes the command sequence invoked by the host and handles the data flow to/from the Flash memories.

The second part is the Flash File System (FFS) [6]: that is, the file system which enables the use of Flash cards, SSDs and USB sticks like magnetic disks. For instance, sequential memory access on a multitude of sub-sectors which constitute a file is organized by linked lists (stored on the Flash card itself) which are used by the host to build the File Allocation Table (FAT).

The FFS is usually implemented in form of firmware inside the controller, each sub-layer performing a specific function. The main functions are: *Wear leveling Management*, *Garbage Collection* and *Bad Block Management*. For all these functions, tables are widely used in order to map sectors and pages from logical to physical (Flash Translation Layer or FTL) [7, 8], as shown in Fig. 2.34.

The upper block row is the logical view of the memory, while the lower row is the physical one. From the host perspective, data are transparently written and overwritten inside a given logical sector: due to Flash limitations, overwrite on the same page is not possible, therefore a new page (sector) must be allocated in the physical block and the previous one is marked as invalid. It is clear that, at some point in time, the current physical block becomes full and therefore a second one (Buffer) is assigned to the same logical block.

The required translation tables are always stored on the memory card itself, thus reducing the overall card capacity.

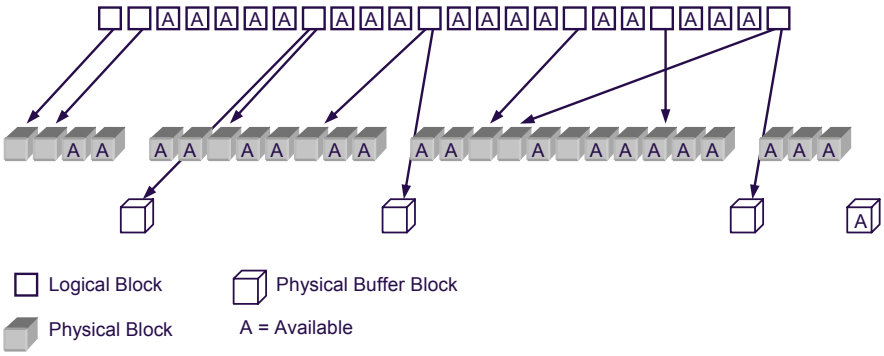


**Fig. 2.34.** Logical to physical block management

### *Wear leveling*

Usually, not all the information stored within the same memory location change with the same frequency: some data are often updated while others remain always the same for a very long time – in the extreme case, for the whole life of the device. It's clear that the blocks containing frequently-updated information are stressed with a large number of write/erase cycles, while the blocks containing information updated very rarely are much less stressed.

In order to mitigate disturbs, it is important to keep the aging of each page/block as minimum and as uniform as possible: that is, the number of both read and program cycles applied to each page must be monitored. Furthermore, the maximum number of allowed program/erase cycles for a block (i.e. its endurance) should be considered: in case SLC NAND memories are used, this number is in the order of 100 k cycles, which is reduced to 10 k when MLC NAND memories are used.

Wear Leveling techniques rely on the concept of logical to physical translation: that is, each time the host application requires updates to the same (logical) sector, the memory controller dynamically maps the sector onto a different (physical) sector, keeping track of the mapping either in a specific table or with pointers. The out-of-date copy of the sector is tagged as both invalid and eligible for erase. In this way, all the physical sectors are evenly used, thus keeping the aging under a reasonable value.

Two kinds of approaches are possible: Dynamic Wear Leveling is normally used to follow up a user's request of update for a sector; Static Wear Leveling can also be implemented, where every sector, even the least modified, is eligible for re-mapping as soon as its aging deviates from the average value.

### *Garbage collection*

Both wear leveling techniques rely on the availability of free sectors that can be filled up with the updates: as soon as the number of free sectors falls below a given threshold, sectors are "compacted" and multiple, obsolete copies are deleted.

This operation is performed by the Garbage Collection module, which selects the blocks containing the invalid sectors, copies the latest valid copy into free sectors and erases such blocks (Fig. 2.35).

In order to minimize the impact on performance, garbage collection can be performed in background. The equilibrium generated by the wear leveling distributes wear out stress over the array rather than on single hot spots. Hence, the bigger the memory density, the lower the wear out per cell is.

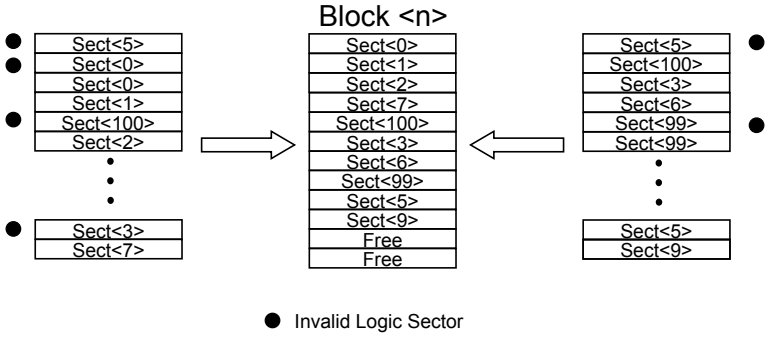


**Fig. 2.35.** Garbage collection

### *Bad block management*

No matter how smart the Wear Leveling algorithm is, an intrinsic limitation of NAND Flash memories is represented by the presence of so-called *Bad Blocks* (BB), i.e. blocks which contain one or more locations whose reliability is not guaranteed.

The *Bad Block Management* (BBM) module creates and maintains a map of bad blocks, as shown in Fig. 2.36: this map is created during factory initialization of the memory card, thus containing the list of the bad blocks already present during the factory testing of the NAND Flash memory modules. Then it is updated during device lifetime whenever a block becomes bad.

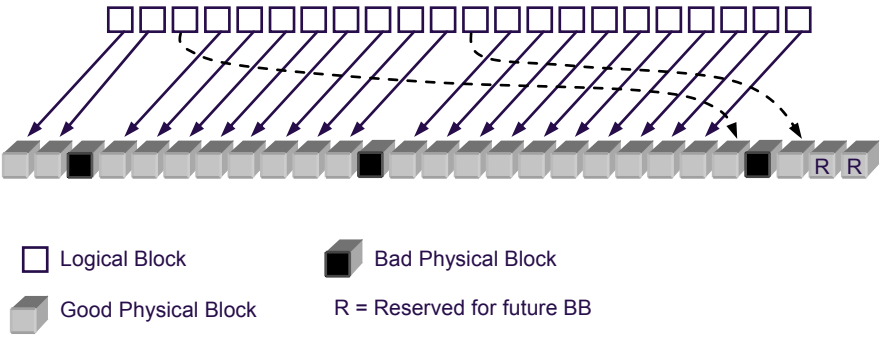


**Fig. 2.36.** Bad Block Management (BBM)

### *ECC*

This task is typically executed by a specific hardware inside the memory controller. Examples of memories with embedded ECC are also reported [9–11]. Most popular ECC codes, correcting more than one error, are Reed–Solomon and BCH [12]. While the encoding takes few controller cycles of latency, the decoding phase can take a large number of cycles and visibly reduce read performance as well as the memory response time at random access.

There are different reasons why the read operation may fail (with a certain probability):

- Noise (e.g. at the power rails)
- $V_{TH}$ disturbances (read/write of neighbor cells)
- Retention (leakage problems)

The allowed probability of failed reads after correction is dependent on the use case of the application. Price sensitive consumer application, with a relative low number of read accesses during the product life time, can tolerate a higher probability of read failures as compared to high-end applications with a high number of memory accesses. The most demanding applications are cache modules for processors.

The reliability that a memory can offer is its intrinsic error probability. This probability could not be the one that the user wishes. Through ECC it is possible to fill the discrepancy between the desired error probability and the error probability offered by the memory (Chap.14).

The object of the theory of error correction codes is the addition of redundant terms to the message, such that, on reading, it is possible to detect the errors and to recover the message that has most probably been written.

Methods of error correction are applied for purpose of data restoration at read access. Block code error correction is applied on sub-sectors of data. Depending on the used error correcting schemes, different amount of redundant bits called parity bits are needed.

Between the length $n$ of the code words, the number $k$ of information bits and the number $t$ of correctable errors, a relationship known as Hamming inequality exists, from which it is possible to compute the minimum number of parity bits:

$$\sum_{i=0}^{t} \binom{n}{i} \leq 2^{n-k} \tag{2.3}$$

It is not always possible to reach this minimum number: the number of parity bits for a good code must be as near as possible to this number. On the other hand, the bigger the size of the sub-sector is, the lower the relative amount of spare area (for parity bits) is. Hence, there is an impact in Flash die size.

BCH and Reed–Solomon codes have a very similar structure, but BCH codes require less parity bits and this is one of the reasons why they were preferred for an ECC embedded in the NAND memory [11].

### 2.4.2 Multi-die systems

A typical memory system is composed by several NAND memories. Typically, an 8-bit bus, usually called channel, is used to connect different memories to the controller (Fig. 2.32). It is important to underline that multiple Flash memories in a system are both a means for increasing storage density and read/write performance [13].

Operations on a channel can be interleaved which means that another memory access can be launched on an idle memory while the first one is still busy (e.g. writing or erasing). For instance, a sequence of multiple write accesses can be directed to a channel, addressing different NANDs, as shown in Fig. 2.37: in this way, the channel utilization is maximized by pipelining data load, while the program operation takes place without requiring channel occupation. A system typically has two to eight channels operating in parallel (or even more).

As shown in Fig. 2.38, using multiple memory components is an efficient way to improve data throughput while having the same page programming time.

The memory controller is responsible for scheduling the distributed accesses at the memory channels. The controller uses dedicated engines for the low level communication protocol with the Flash.

Moreover it is clear that the data load phase is not negligible compared to the program operation (the same comment is valid for data output): therefore increasing I/O interface speed is another smart way to improve general performance: high-speed interfaces, like DDR, have already been reported [14] and they are discussed in more details in Chap. 7. Figure 2.39 shows the impact of DDR frequency on program throughput. As the speed increases, more NAND can be operated in parallel before saturating the channel. For instance, assuming a target of 30 MB/s, 2 NANDs are needed with a minimum DDR frequency of about 50 MHz. Given a page program time of 200 µs, at 50 MHz four NANDs can operate in interleaved mode, doubling the write throughput. Of course, power consumption has then to be considered.

There are also hybrid architectures which combine different types of memory. Most common is usage of DRAM as memory cache. During write access the cache is used for storing data before transfer to the Flash. The benefit is that data updating, e.g. in tables, is faster and does not wear out the Flash.

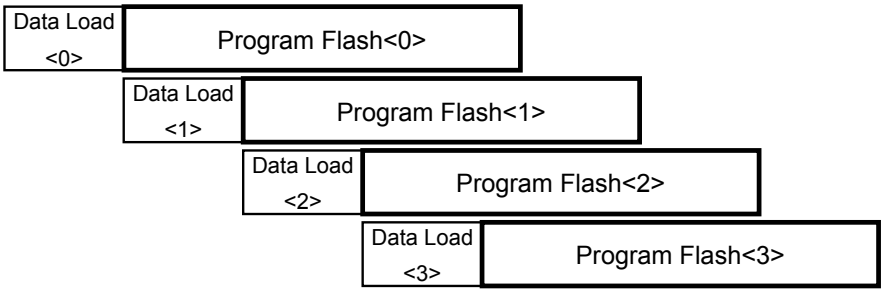


**Fig. 2.37.** Interleaved operations on one Flash channel

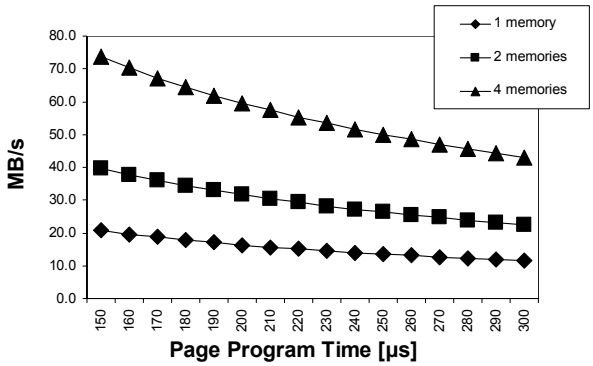


**Fig. 2.38.** Program throughput with an interleaved architecture as a function of the NAND page program time

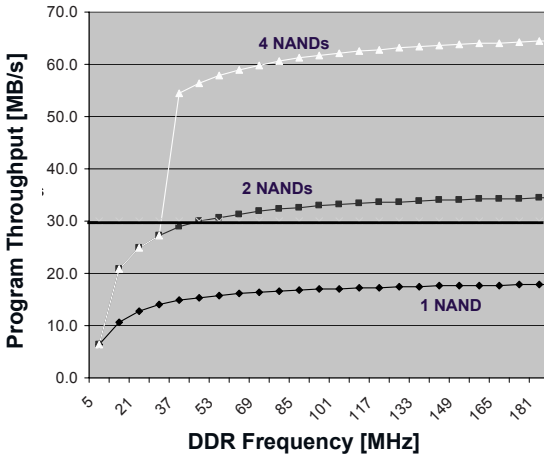


**Fig. 2.39.** Program throughput with an interleaved architecture as a function of the channel DDR frequency. 4 kB page program time is 200 µs

Another architecture uses a companion NOR Flash for purpose of "in-place execution" of software without pre-fetch latency. For hybrid solutions, a multi-die approach, where different memories are packaged in the same chip, is a possibility to reduce both area and power consumption.

### 2.4.3 Die stacking

Reduced form factor has been one of the main drivers for the success of the memory cards; on the other hand, capacity requirement has grown dramatically to the extent that standard packaging (and design) techniques are no longer able to

In fact, DRAM are the main drivers for TSV technology as, with the standard bonding technology, they cannot stack more than two dies in the same package.

The solutions presented so far exploit advances in stacking techniques, eventually requiring changes in chip design floorplan. Quite recently, advanced design and manufacturing solutions have been presented, where the 3D integration [20] is performed directly at chip level.

### 2.4.4 3D memories and XLC storage

The 3D concept is simple: instead of stacking several dies, each of which being a fully-functional memory component, it is possible to embed in the same silicon die more than one memory array. In this way, all the control logic, analog circuits and the pads can be shared by the different memory arrays. In order to keep the area to the minimum, the memory arrays are grown one on top of the other, exploiting the most recent breakthroughs in silicon manufacturing technology.

Two different solutions have been recently presented for NAND Flash memories: in one case [21, 22], the topology of the memory array is the usual one, and another array is diffused on top of it, as shown in Fig. 2.48, so that two layers exist. Therefore the NAND strings (i.e. the series of Flash memory cells which is the basic building block of the array) are diffused on the X–Y plane. Around the arrays, all the peripheral circuitry is placed in the first (i.e. lower) layer. The only exception is the wordline (WL) decoder. To avoid Fowler–Nordheim (FN) erasing of the unselected layer, all WLs in that layer must be floating, just like the WLs of unselected blocks in the selected layer. This function is performed by a layer-dedicated WL decoder.

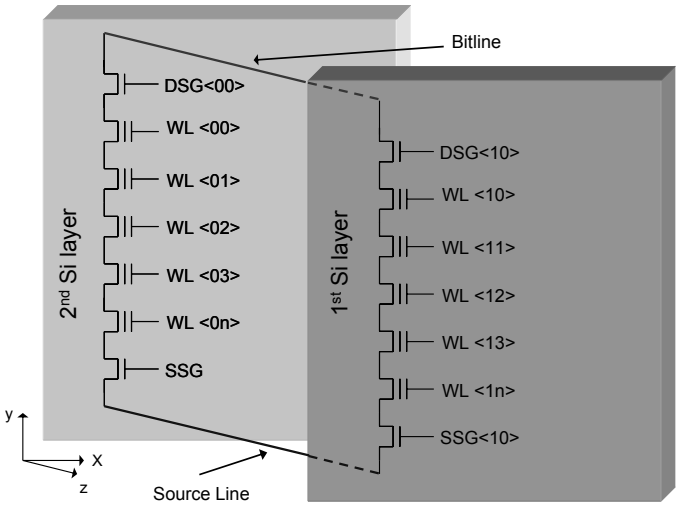


**Fig. 2.48.** Three-dimensional horizontal memory array

The second approach [23, 24] is shown in Figs. 2.49 and 5.9: in this case the NAND strings are orthogonal to the die (along the Z direction). NAND string is on the plugs located vertically in the holes punched through whole stack of the gate plates. Each plate acts as control gate except the lowest plate which takes a role of the lower select gate. 3-D charge trap memories are described in Sect. 5.3.

The density of the memory can also be increased acting at cell level: in its simplest form, a non-volatile memory cell stores one bit of information: '1' when the cell is erased and '0' when it is programmed: Sensing techniques for measuring the amount of electrical charge stored in the floating gate are described in Chap. 8. This kind of storage is referred to as Single Level Cell (SLC).

The concept can be extended by having four different charge levels (corresponding to logic values 00, 01, 10, 11) inside the floating gate, thus leading to the so-called Multi Level Cell (MLC) approach, i.e. 2 bit/cell. Chapter 10 is entirely dedicated to MLC NAND devices.

Several devices implementing the 2 bit/cell technology are commercially available, and indeed MLC has become a synonym for 2 bits/cell. Almost all the Flash cards contain MLC devices as they are cheaper than SLC.

The circuitry required to read multiple bits out of a cell is of course more complex than in the case of single bit, but the saving in term of area (and the increase in density) is worth the complexity. The real disadvantage lies in reduced endurance and reliability.

In terms of endurance, as already mentioned previously, a SLC solution can withstand up to 100,000 program/erase cycles for each block, while a MLC solution is usually limited to 10,000. For this reason, wear leveling algorithms must be used, as already outlined in Sect. 2.4.1. In terms of reliability, it is clear that the more levels are used, the more read disturbs can happen, and therefore the ECC capability must be strengthened.

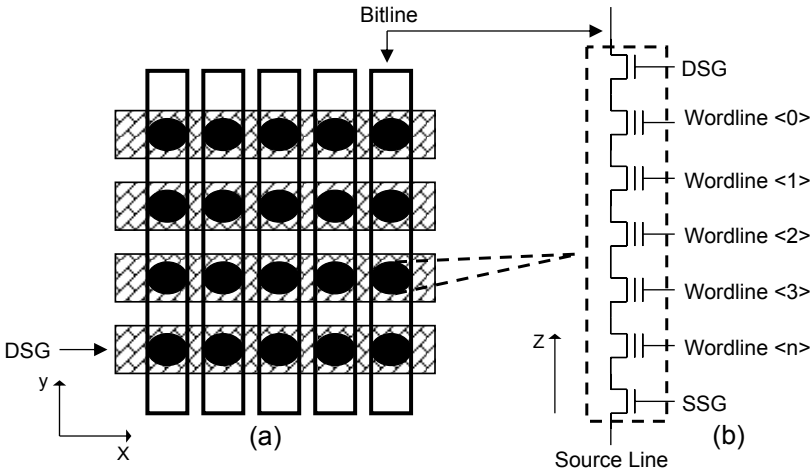


**Fig. 2.49.** Three dimensional vertical memory array: (a) top down view of 3-D vertical memory array; (b) equivalent circuit of the vertical NAND string

The concept has been extended recently to 3 bit/cell and 4 bit/cell, where 8 and 16 different charge levels are stored inside the same cell. This storage approach is known as XLC and it is described in Chap. 16.

## References


1. G. Campardo, R. Micheloni, D. Novosel, "VLSI-Design of Non-Volatile Memories", Springer-Verlag, 2005.
2. R. H. Fowler and L. Nordheim, "Electron Emission in Intense Electric Fields," Proceedings of the Royal Society of London, Vol. 119, No. 781, May 1928, pp. 173–181.
3. www.mmca.org
4. www.compactflash.org
5. www.sdcard.com
6. A. Kawaguchi, S. Nishioka, and H. Motoda. "A Flash-Memory Based File System", Proceedings of the USENIX Winter Technical Conference, pp. 155–164, 1995.
7. J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho. "A Space-Efficient Flash Translation Layer for Compactflash Systems," IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, May 2002, pp. 366–375.
8. S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S.-W. Park, and H.-J. Songe. "FAST: A Log-Buffer Based FTL Scheme with Fully Associative Sector Translation", 2005 US-Korea Conference on Science, Technology, and Entrepreneurship, August 2005.
9. T. Tanzawa, T. Tanaka, K. Takekuchi, R. Shirota, S. Aritome, H. Watanabe, G. Hemink, K. Shimizu, S. Sato, Y. Takekuchi, and K. Ohuchi, "A Compact On-Chip ECC for Low Cost Flash Memories," IEEE Journal of Solid-State Circuits, Vol. 32, May 1997, pp. 662–669.
10. G. Campardo, R. Micheloni et al.,"40-mm$^2$ 3-V-only 50-MHz 64-Mb 2-b/cell CHE NOR Flash memory," IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, Nov. 2000, pp. 1655–1667.
11. R. Micheloni et al., "A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput", IEEE International Solid-State Circuits Conference Dig. Tech. Papers, pp. 142–143, Feb. 2006.
12. R. Micheloni, A. Marelli, R. Ravasio, "Error Correction Codes for Non-Volatile Memories", Springer-Verlag, 2008.
13. C. Park et al., "A High Performance Controller for NAND Flash-based Solid State Disk (NSSD)", IEEE Non-Volatile Semiconductor Memory Workshop NVSMW, pp. 17–20, Feb. 2006.
14. D. Nobunaga et al., "A 50nm 8Gb NAND Flash Memory with 100MB/s Program Throughput and 200MB/s DDR Interface", IEEE International Solid-State Circuits Conference Dig. Tech. Papers, pp. 426–427, Feb. 2008.
15. K. Kanda et al., "A 120mm$^2$ 16Gb 4-MLC NAND Flash Memory with 43nm CMOS Technology", in IEEE International Solid-State Circuits Conference Dig. Tech. Papers, pp. 430–431, Feb. 2008.
16. Chang Gyu Hwang, "New Paradigms in the Silicon Industry, International Electron Device Meeting (IEDM), 2006, pp. 1–8.
17. M. Kawano et al., "A 3D Packaging Technology for a 4Gbit Stacked DRAM with 3Gbps Data Transfer, International Electron Device Meeting (IEDM), 2006, pp. 1–4.

their variations. As in all microelectronics industry, however, this is a fast changing and evolving condition. Formerly very widespread cards were the CompactFlash and the SmartMedia, although now rather superseded. More vendor specific cards are Memory Stick and xD. USB Flash Drive are yet another type of card very popular today because they have no restrictions on their size like all the aforementioned cards and can thus be very sizeable in available memory.

The following Table 17.1 summarizes the dimensions of the most widespread memory cards.

**Table 17.1.** Dimensions of selected memory cards

| Card | Common varieties | Dimensions (mm) |
|---|---|---|
| CompactFlash | I | 36 × 43 × 3.3 |
| | II | 36 × 43 × 5.0 |
| SmartMedia | | 37 × 45 × 0.76 |
| MultiMediaCard | MMC | 24 × 32 × 1.4 |
| | Reduced size | 24 × 18 × 1.4 |
| Secure Digital | SD | 24 × 32 × 2.1 |
| | microSD | 11 × 15 × 1.0 |
| Memory Stick | Standard | 21.5 × 50 × 2.8 |
| | Pro | 21.5 × 50 × 2.8 |
| | Duo | 20 × 31 × 1.6 |
| xD | Standard | 25 × 20 × 1.7 |
| | Type M | 25 × 20 × 1.7 |
| | Type H | 25 × 20 × 1.7 |
| USB Flash Drive | | Arbitrary |

Host manufacturers have a multitude of cards to pick from for their applications. Each card may have its own advantages or disadvantages in terms of form factor, size, capacity, speed, protocol complexity or pin count. Sometimes, however, more mundane motives underlie the card choice. These might include licenses and royalties that come with some cards' types.

Security is also an increasing decisive factor that may tip the scale toward one card rather than another. Some cards implement protocol features specifically devised to limit somehow the access to the card content. These features are known as *digital rights management* (*DRM*) and are mostly used by copyright owners.

## 17.2 Memory card architecture and assembly

One of the foremost common characteristics of memory cards is their use of NAND Flash memories as storage devices, although this is not compulsory in their definition. Indeed, memory cards with NOR Flash memories have been designed and in the near future even other futuristic memories like the Phase Change Memories (PCM) might be used, too.

All memory cards have a similar architecture. A simple block diagram is represented in Fig. 17.1. The core of the card is a Micro-Controller, which masters all the card's activities. A stack of NAND memories constitutes the memory space where data are actually stored. The Micro-Controller communicates with the stack of all the NAND memories through a *Memory Interface*. The Memory Interface implements the NAND protocol to exchange information with the memory stack. It must comply with the specifications dictated by the NAND manufacturer, so a different NAND type usually necessitates a different Memory Interface.

The Micro-Controller must also communicate with the external world. This task is mandated to the *Protocol Interface*. Each type of card (MMC, SD, Compact Flash...) has to obey to its own protocol.

The three blocks Protocol Interface, Micro-Controller and Memory Interface constitute a single die, the Controller. The NAND memories are separate dice. All these dice must be assembled in the same package. This is challenging because the thickness of the card package is very thin (1–3 mm depending on the card). Different strategies can be applied to the package assembly.

One method is to assemble the Flash, typically in a TSOP or WSOP package, and the Controller, typically in an LGA package, together through surface mount technology (SMT). A simple schematic diagram is shown in Fig. 17.2.

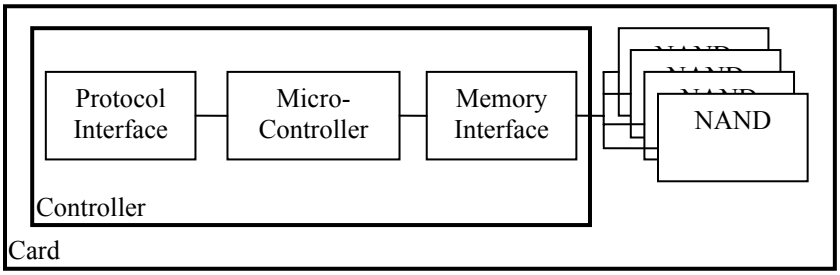


**Fig. 17.1.** Memory card's architecture

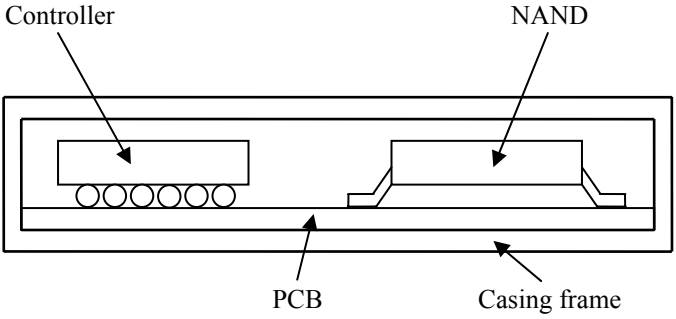


**Fig. 17.2.** Surface mount technology (SMT)

A *Multi-Chip Package* (*MCP*) is a package containing several dice of the same type stacked vertically and connected in such a way to form a single bigger device. In Fig. 17.2, the NAND can be a single die or an MCP of stacked NAND dice. A *System-in-Package* (*SiP*) is a package combining different devices (digital, analog, passives...) forming a system. The whole card package is thus a SiP.

Another possible approach is to bond the bare chips directly to the substrate, without putting them in their own package first. This technique is dubbed Chip on Board (COB) and has the obvious advantage of letting more room within the card package.

## 17.3 Memory card specifications

As an example of card protocol and specifications, we can look in deeper details at a particular card type, the Embedded MultiMediaCard (eMMC). In particular, we will refer to the JESD84-A44 datasheet (MMCA 4.4), but all that will be described here still holds true with minor changes for any other version of MMC. Secure Digital cards are very similar to MMC cards, too.

### 17.3.1 Pinout

The pin out of the eMMC card is very simple. Apart from the supply voltage pins there is a reset pin, the ever-present clock (*CLK*), one command line (*CMD*) and eight data lines (*DAT<7:0>*). The CMD line is used by the host to issue commands to the card and by the card to send responses to the host. The DAT lines are used to exchange data between the host and the card.

### 17.3.2 Commands and responses

Commands are sent by the use of command tokens, whose coding scheme is 48 bit long and sent via the CMD line one bit at a time:

| | Start Bit | Transmission Bit | Command Index | Argument | CRC7 | End Bit |
|---|---|---|---|---|---|---|
| Width (bits) | 1 | 1 | 6 | 32 | 7 | 1 |

The Start Bit, the Transmission Bit and the End Bit have a fixed pattern and their purpose is simply to signal the start and the end of the command token. The *Command Index* is 6 bit long, thus providing 64 possible commands. Further commands are made available through a special sequence of two command tokens. The *Argument* field has a different meaning according to the command issued.

The *Driver Stage Register* selects the power strength of the output buffer. This permits to adapt the card's output stage to the transfer rate and to the bus capacitance (which varies for instance according to the bus length and the number of connected cards).

## 17.4 Flash translation layer

Operating systems usually write data in sectors (e.g. 512 bytes in size). Unfortunately, Flash memories lack the so-called *update-in-place* capability. This means they cannot be altered on a sector (page in NAND nomenclature) basis, because the erase operation involves a whole block (made up of several sectors/pages). Therefore, file systems cannot write ones back to zeros in a single addressed sector. This is awkward for a high-level file system to manage.

In order to alleviate the workload of the operating system, software routines can be implemented to emulate a NAND Flash behavior where single sectors (or NAND pages) can be modified independently. Such software is called *Flash Translation Layer* (*FTL*), which is an intermediate layer between the file system and the storage media (the NAND Flash). The operating system can then write the NAND memory on a page basis without worrying about the details of its physical implementation. Usually it is the Controller in Fig. 17.1 taking care of the FTL.

The FTL achieves its task through a *logical to physical re-mapping*. Hence, when the file system sends the command to write or update a certain logical page, the FTL actually writes it in a different available physical page and updates a table storing the logical to physical mapping (*mapping table*). The page containing the older data is marked as invalid; Fig. 17.6 shows an example.

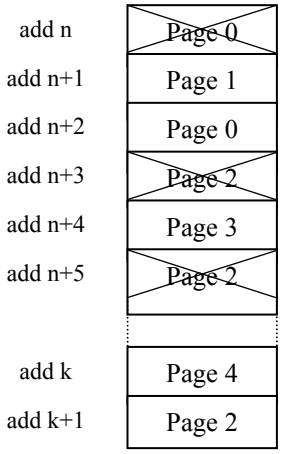


**Fig. 17.6.** Logical to physical re-mapping

Here page 0 has been written first at address n, followed by page 1. Then page 0 has been updated, but instead of overwriting it at address n, the FTL writes it in another location (n + 2) and marks address n as invalid.

As it is apparent from Fig. 17.6, after a while a great portion of the memory will be filled with invalid data taking precious unusable space. Eventually, all these locations will have to be erased to make this memory space available to new data. This procedure is not trivial, because the erase operation involves a whole block, comprising several pages, some with valid data, while others with invalid data. Therefore, valid data must be copied first to a new location, before the erase operation takes place. All this procedure is called *garbage collection*. An example is shown in Fig. 17.7.

All valid pages are copied into new locations (the order does not matter because the mapping table takes care of the correct correspondence between the logical and physical address). Then all the memory locations containing the old data (valid and invalid) are erased, thus creating space accessible to new data.

Garbage collection is very critical for many reasons. Let us start to better illustrate this point by introducing the notion of garbage collection efficiency.

*Garbage collection efficiency* is defined as the ratio of the number of invalid pages in a block to the total number of pages in that block. The higher the invalid pages within a block, the higher the efficiency. Indeed, when the number of invalid pages within a block is high we have a twofold benefit. First, only a few pages must be copied into new locations, thus limiting the program time and reducing the cycling efforts (greater lifetime) of the memory cells. Moreover, when erasing such block, a lot of space will be available for new data, because only few data were valid and still taking memory space.

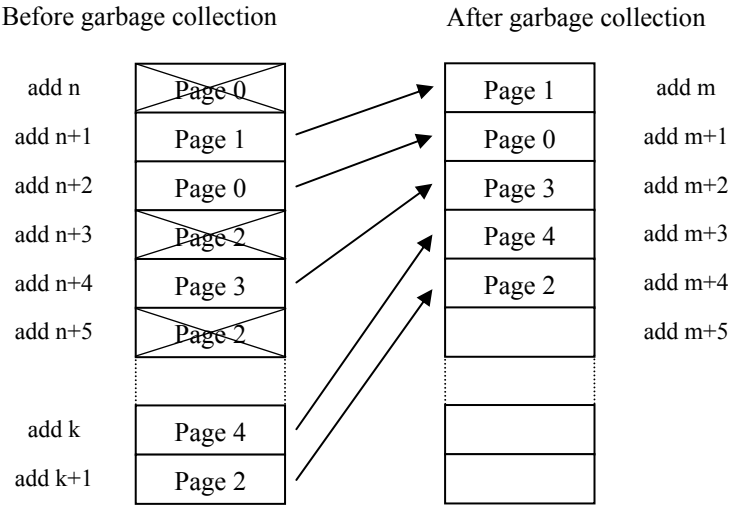


**Fig. 17.7.** Garbage collection

When should the garbage collection be run? A high program throughput calls for a lot of free space. If the host requires more space than readily available, it will have to halt its operation and wait for the garbage collection to kick in and erase enough invalid pages. This will limit the sustainable program throughput. So apparently, the garbage collection should be run as often as possible. However, this in turn will limit the garbage collection efficiency. Therefore, an accurate trade-off will have to be determined to establish the best performance of garbage collection.

Flash memories' blocks can only be programmed and erased a limited number of times. This is the reason why a good FTL should also implement a *wear leveling* algorithm. A wear leveling software module makes sure all NAND blocks are cycled an identical number of times, avoiding some blocks ending their cycle life while others being cycled only a few times. Were not a wear leveling algorithm present, blocks with frequently changed data would endure much more cycles than blocks containing long-lived data.

There are two levels of wear leveling. In the *first level wear leveling* new data is programmed into a free block with the fewest write/erase cycles. Two techniques to achieve this are the chain method and the array method.

In the *chain method,* the pool of available blocks is organized in a chain (Fig. 17.8).

A simple round robin algorithm selects one block after another. When a block is erased, it is added to the chain.

In the *array method* an age vector stores the number of each block's program/erase cycles. This information can be used by the wear leveling algorithm to ensure a uniform exploitation of all the blocks.

In the *second level wear leveling* from time to time blocks containing long-lived data are moved (copied) into other blocks even though they never received an update command. The original block can then be used to store new incoming data. The second level wear leveling is triggered when the difference between the maximum and the minimum number of cycles per block reaches a predetermined threshold.

Devising an efficient FTL is a taxing job. Two main concerns are the clustering and the cleaning policy. A few examples of both will be reported, but let start off by introducing the concepts of hot and cold data.

*Hot data* are the most frequently updated data. *Cold data* (or non-hot data) are seldom updated data. In a given block, it is advantageous to gather (cluster) all hot data or all cold data, because blocks with hot data will soon become garbage. Cleaning such blocks is beneficial because they have high garbage collection efficiency.

How to effectively cluster hot data then? One method is the *Dynamic dAta Clustering* (*DAC*) depicted in Fig. 17.9.

**Fig. 17.8.** Chain method

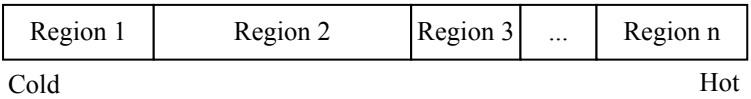


**Fig. 17.9.** Dynamic dAta Clustering

In the DAC method, the Flash memory is divided into *n* regions, whose size is not necessarily identical. Each region is not even necessarily made up of contiguous memory locations. Region 1 contains the coldest data, Region n the hottest. The hotter a block becomes, the more it shifts towards Region n. Conversely, the colder a block becomes, the more it shifts towards Region 1. When a block is written for the first time, it is put in Region 1. When a block is updated, if it has been into its Region k for a time shorter than a $t_{threshold}$ it is written in Region k + 1, otherwise it is written again in Region k. The $t_{threshold}$ is introduced because a block's hotness degrades as it ages. Upon the cleaning request of a block, if its valid data have been in their Region k longer than $t_{threshold-2}$ they are copied in Region k − 1, otherwise they are still copied in Region k. Each block has associated with it the number of the Region it belongs to.

The real advantage of this technique is twofold. Firstly, the algorithm does not require complex calculations and secondly the classification is fine grained (which in turn improves the performance).

Another clustering method is the *Dynamic Striping*, illustrated in Fig. 17.10. In the Dynamic Striping methodology, two registers called LRU (least recently used) are utilized. Only logical block addresses belonging to the hot list are hot, all the others are cold. The candidate list contains the logical block addresses recently written. If a logical block address belongs to the candidate list and is updated after a short while, it is promoted to the hot list. The last element of the hot list is demoted to the candidate list.

As a further improvement of the Dynamic Striping, hot data are written in the bank with the lowest number of erase cycles to enhance the wear leveling.

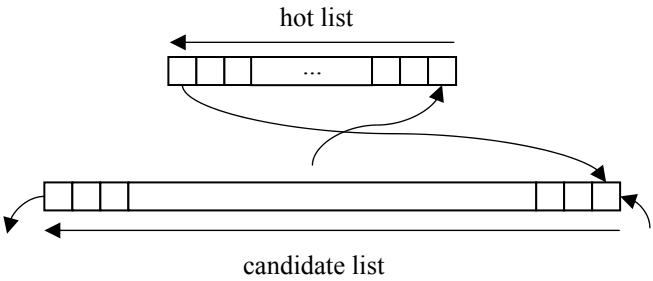


**Fig. 17.10.** Dynamic striping

Cold data are written in the bank with the lowest capacity utilization (i.e. the ratio of the number of live pages to the number of total pages) in order to make the capacity utilization more uniform (cold data will remain longer in their locations).

*Cleaning policies* relate to the criterion used to pick up the block to be erased for garbage collection purposes.

The *greedy policy* is one of the simplest cleaning policies, in that it selects for erasing the block with the greatest number of invalid data. Though simple, this policy is not very efficient because it does not take into account the cycling the blocks have been subjected to. It has also been proved ineffective in case of localities of reference (i.e. the fact that if a certain memory location is referenced at a given time, it is probable that the neighboring memory locations will be referenced in the near future).

A better cleaning policy is the *cost-benefit policy*. The block chosen for erase is the one with the greatest value of the following expression:

$$\frac{age \cdot (1 - valid)}{2 \cdot valid} \tag{17.1}$$

where *valid* is the percentage of valid data in the block, and *age* is the time elapsed since the last modification. Note that (1-*valid*) is the percentage of invalid data, i.e. the space that ultimately will be freed. The term 2·*valid* can be conceptually regarded as the cost of reading the valid data and to copy them in another location (2·*valid* = *valid* + *valid* and the first term is proportional to the time required to read the valid data while the second term is proportional to the time required to copy them in a new location). Therefore, the ratio (1-*valid*)/2·*valid* in the above expression can be considered as a benefit over cost ratio. This policy is better than greedy's because it takes into account the age of the block.

Yet another cleaning policy is the *Cost Age Time* (*CAT*) *policy*. The block chosen for erase is the one with the minimum value of the following expression:

$$\frac{valid}{1 - valid} \cdot \frac{1}{age} \cdot n_C \tag{17.2}$$

where *valid* and *age* are defined as in the previous case and $n_C$ is the number of cleaning, i.e. the number of times a block has been erased. This cleaning policy is endowed even with a sort of wear leveling thanks to the term $n_C$.

A good FTL must also take care of *bad blocks*. Bad blocks contain not-working or unreliable bits and therefore cannot be used. They may be present since the manufacturing of the device or may develop during its lifetime. The FTL must store all the necessary information about which blocks can be used and which cannot. When a program operation in a certain block fails, the FTL acknowledges that through the NAND Status Register and acts copying the valid data of the block into a new good block; the block where the fail occurred is then be marked as bad.

Finally, a FTL must ensure the reliability of data in the event of sudden power loss. In order to guarantee this, the FTL must wait for the program operation to

complete successfully before marking the old page as invalid or before erasing a block. In such a way, if the new data is lost due to a power failure, the old data can still be recovered.

## 17.5 Cryptography

Security has become an important part of cards' characteristics. New protocols implement commands handling confidential data with the utmost caution. Let us mention the standard JESD84-A44 (eMMC), which states that every card complying with it must implement a special memory area called *Replay Protected Memory Block* (*RPMB*), which must be protected against replay attacks. In a *replay attack* a valid data transmission is intercepted first, recorded and then played back again later on by an attacker. Hence, cryptography has become fundamental even in card design and usage. To protect a portion of the memory against replay attacks the eMMC protocol employs a key (MAC) and a nonce. To better understand these concepts, let us briefly review a few pivotal definitions and notions about cryptography.

A *cipher* is an algorithm to *encrypt* and *decrypt* a *plaintext* into and from a *ciphertext* (Fig. 17.11).

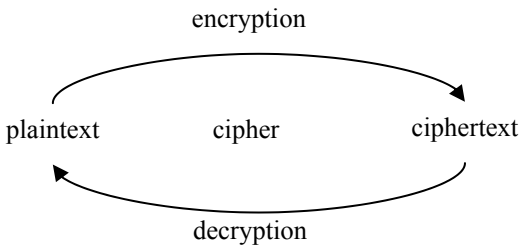


**Fig. 17.11.** Cipher

Note that in the everyday vernacular language sometimes the terms cipher and code are used interchangeably. However, in cryptography a *code* has a *codebook* associating each word or phrase of the plaintext with one or more codewords. Therefore, the plaintext set is somewhat limited and must be anticipated a priori. Codes operate according to the meaning while ciphers operate according to a single letter or bit.

*Block ciphers* work on blocks whereas *stream ciphers* work on continuous streams of data. Each cipher needs a *key* or *cryptovariable*. *Symmetric key algorithms* have the same key for encryption and decryption. *Asymmetric key algorithms* have different keys for encryption and decryption. In symmetric key algorithms, the key must be known only by the sender and by the recipient.

A *cryptographic hash function* is a function mapping a *message* into a message *digest* (a.k.a. *hash value*) as depicted in Fig. 17.12.

it is downloaded into a RAM and then executed from the RAM. This approach has two main disadvantages. The first is high power consumption due to the refresh operation required by RAM memories. The second drawback is the time lost for the downloading of the content from the Flash to the RAM, which slows down all the system.

In the Store and Download methodology, if all the code in the non-volatile memory is downloaded to the RAM, the architecture is termed *Fully Shadowed*. In this case, the boot phase is very sluggish. Conversely, in a *Demand Paged* architecture, only a subset of the code is copied from the Flash to the RAM, not all of it. Obviously, the part copied is the one needed at that specific moment.

Sometimes, in the Demand Paged architecture, the code is also stored in the Flash in a compress form and it is uncompressed when copied into RAM. On the contrary, compression is used rarely in Fully Shadowed solutions. Evidently, compression is incompatible with an Execute in Place technique.

Running the code of an application directly from the Flash memory eludes all the shortcomings of the Store and Download methodology listed thus far. Even in this case, however, a small amount of RAM is still needed for variables and the stack. Only the code and read-only data (constants) are read from the Flash. The RAM size is thus considerably reduced. This implies both a smaller footprint and lower power consumption (the refresh takes place in a much smaller memory). Another contribution to lower power consumption is given by the fact that a Flash memory can be put in stand-by state when not in use without losing its content, while this is impossible for a RAM. Usually, the Flash is divided in two areas, one for the code and the other for the data.

Unfortunately, all that glitters is not gold. In order to implement the Execute in Place, the Flash must possess a few definite characteristics. The foremost attribute is random read timing. This holds true in particular for the system code. For this reason, NOR type memories are ideal for this application. NAND type memories, on the contrary, are not very apt to it because of their high latency and for this reason are mostly used with the Store and Download method. A high-speed buffer, though, can still render NAND memories liable to Execute in Place.

Furthermore, Flash memories are slower than RAM. However, if a microprocessor's cache is present it helps to counteract this effect.

## 17.7 Managed memory

Until few years ago, NAND Flash memories increased their capacity size by means of technology shrink, even more than Moore's law could foresee in 1965. Today, we are getting closer to lithography limits, supposed to be close to 20 nm. To maintain the current trend and double the number of bit stored per square millimeter, in an 18–24 months timeframe, a different solution must be adopted to pack more bits into the same cell. The choice is a 2-bit/cell device, so called MLC (Multi Level Cell), a 3-bit/cell device, so called TLC (Triple Level Cell), or more in general, an n-bit per cell.

### 17.7.1 Multibit and shrink technology issues

The increase in the number of bits per cell involves a corresponding increase in the issue related to the cell size reduction.

A first aspect to be taken into account is that we need to put a greater number of threshold distributions almost in the same voltage range; as an example, in a 2-bit/cell device we need to put three distributions in a range of about 6 V (cells erased are on the left of 0 V), while in a 3-bit/cell seven distributions are inside the same range. In a MLC device, using all the available range between 0 and 6 V, we have 6 V/3 = 2 V for each distribution, while if the device is a TLC the space available for each distribution is less than one half of the MLC: in fact 6 V/7 = 0.86 V (see Fig. 17.14).

The distributions shown in Fig. 17.14 are ideal, because they are still distinct entities, each of them taking up a well defined space; we can therefore associate every cell, whatever its Vth value is, with the logic value of the information contained inside.

In reality, due to multiple phenomena that will be discussed below, distributions tend to enlarge and to move, producing an overlap of the distributions themselves (see Fig. 17.15).

These phenomena, as we have seen, are becoming more evident with the technology shrink and especially with the increase of the number of bits stored within each cell, as it involves an exponential increase in the number of distributions to be placed on the axis of the threshold voltages.

This uncertainty about the value contained into some cells involve NAND memory devices usage; a solution has to be put in place, aimed at eliminating, or at least minimizing, the causes responsible for the Vth change in respect of the value assigned during the programming process. The same approach should also ensure that, even with a Vth change, at least for a limited number of cells, it can be possible to retrieve the information stored during the program operation.

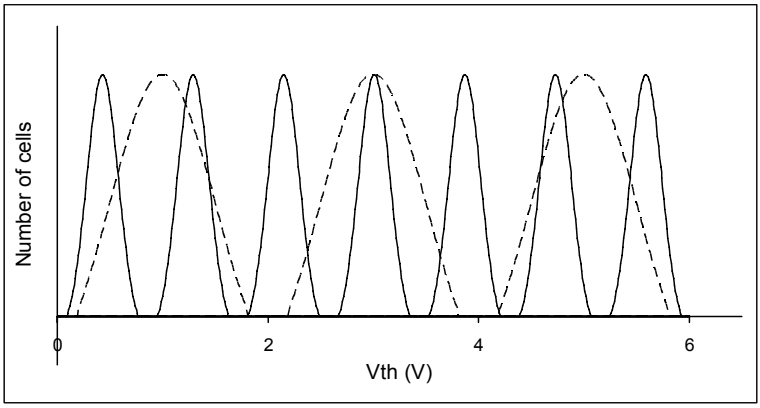


**Fig. 17.14.** Ideal distribution of positive Vth in 2 bits/cell case (dotted line) and 3 bits/cell (solid line)

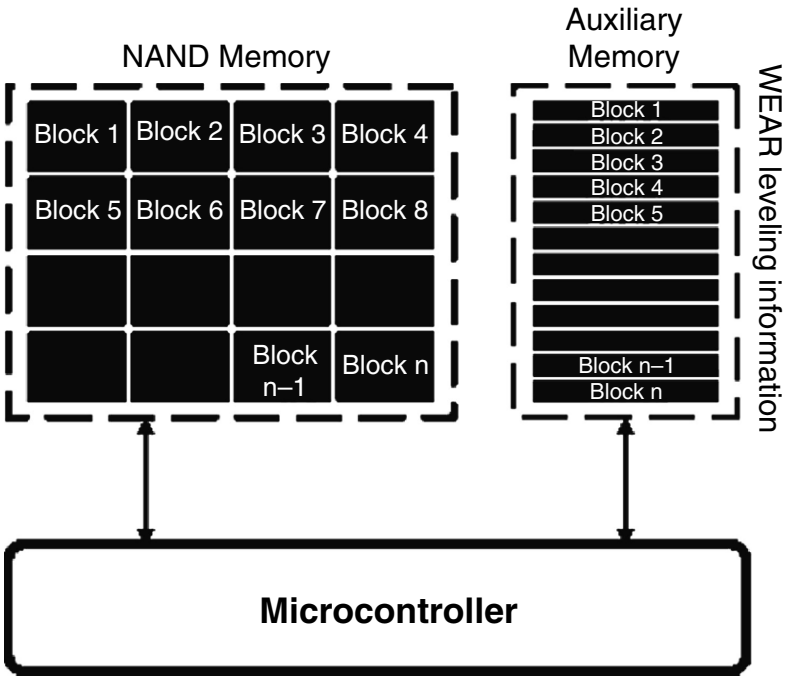


**Fig. 17.22.** Auxiliary memory for wear-leveling

### 17.7.3 Flexible solution

In non volatile memory environment, NOR and NAND addressed so far different application segments, where different performances and memory capacity were requested.

Random Read performances of NOR Flash are particularly useful to implement code Execute In Place (XiP): this kind of memory is used in a system as execution memory of code or as boot memory.

NAND write throughputs and high densities are the best solution for Mass Storage applications but the range of applicability has progressively increased covering also code storage: a NAND device is used to guarantee code (operating system code, software applications) persistence that is usually downloaded in an execution memory (typically a RAM) more often in a multichip solution.

As we saw before, NAND technology is classified mainly into two sub-types: Single Level Cell and Multi Level Cell. Table 17.2 summarizes some differences between the two classes related to the same cell lithography.

The basic differentiation points highlight that SLC is more suitable to store and download code thanks to the higher read performances and endurance, while MLC NAND is an optimized solution for all applications requiring high densities of storage.

**Table 17.2.** NAND Flash: SLC versus MLC

| | SLC | MLC |
|---|---|---|
| Sequential read access time | 25 ns | 25 ns |
| Random read access time | 25 µs | 60 µs |
| Program time | 200 µs | 800 µs |
| Erase time | 2 ms | 2.5 ms |
| Power supply – Vdd | 1.8/3.0 V | 3.0 V |
| Program/Erase cycles | 100 k | 10 k |
| Data retention | 10 years | 10 years |

Many market applications require both SLC features to reliably store system code and data and MLC features, to save high density data while requiring lower reliability

A new solution, we'll call it flexible, is realized thanks to the microcontroller and consists on the partitioning of the single NAND cell in two sub sections, the first of them to be used as SLC, the other one as MLC (see Fig. 17.23).

The partitioning allows the microcontroller to use different algorithms and settings in memory usage depending on the features of the selected portion. The partition is dynamic; the space assigned to the code can be modified by the microcontroller. In the SLC partition program (PV) and read (RV) references are more relaxed and the pulses can have a wider size due to a single distribution in the positive side of the threshold voltage values. This minor precision in Vth values allows speeding up program and read operations, getting closer to native SLC memories. Additionally, in SLC partition, there is not the need of a high ECC capability, affecting as well performances and data area to store parity bits.

The same approach can be applied to 3 or more bits per cell devices, where a portion is reserved for a SLC-mode usage to get better performances in terms of throughput and reliability.

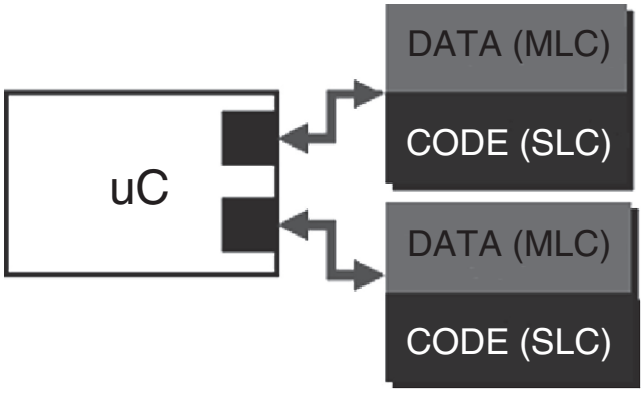


**Fig. 17.23.** Flexible solution scheme