POLITECNICO DI TORINO

SCUOLA INTERPOLITECNICA DI DOTTORATO

Doctoral Program in Computer and Control Engineering

Final Dissertation

Service Oriented Non Volatile Memories

Marco Indaco

Tutor
prof. Paolo Prinetto

Co-ordinator of the Research Doctorate Course
prof. Pietro Laface

27/02/2014

Herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This thesis has not previously been presented in identical or similar form to any other Italian or foreign examination board. The thesis work was conducted from 01/2011 to 12/2013 under the supervision of Prof. Paolo Prinetto at Politecnico di Torino.

ACKNOWLEDGEMENTS

This thesis represents the culmination of three years of research. During this period many persons gave me suggestions and encouraged me. From all these persons I received a kind support, and for this reason I would like to thank all of them.

First and foremost, I wish to thank my advisor Prof. Paolo Prinetto, from Politecnico di Torino. This work would not be possible without his precious suggestions. His scientific driving guide helped me to find the right direction for my research activity. I wish to thank him also for gave me a real helpful hand in reviewing all the material I covered in this thesis. I also wish to thank Dr. Stefano Di Carlo, from Politecnico di Torino. He helped and inspired me with his valuable experience during the whole period of my PhD.

This thesis is also the result of the work I did with other research groups in Europe. In this sense I would like to thank people from the Universitat Polytecnica de Catalunya, and in particular Prof. Joan Figueras and Dr. Rosa Rodriguez-Montañés to share their time not only working together: thanks, I passed beautiful moments in Barcellona.

During these three years of research activity I spent the majority of the time with my colleagues in the laboratory of the Control and Computer Department, Politecnico di Torino. I shared with them good and the bad thinks of this life experience. Thanks all of you guys.

And last, but not least, I would like to really thank my family and my girlfriend for they unconditional love and patience. Even if they can not understand all the parts of my work, they always share all my problems, giving an unlimited and invaluable support. They always encouraged me during this adventure.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Stay hungry. Stay foolish

CHAPTER 1

INTRODUCTION

Contents of this chapter

According to Moore's law, advanced multifunctional computing systems realized in forthcoming manufacturing technologies hold the promise of a significant increase in device integration density, complemented by an increase in system performance and functionality.

In this scenario, a meaningful portion of the produced devices is represented by memories, one of the key components of any electronic systems to store both data and istructions.

Semiconductor memories can be split in terms of data persistance into two major categories: *volatile* and *non-volatile memories*. Random Access Memorys (RAMs), like Static RAM or Dynamic RAM are considered volatile memories. Although very fast in writing and reading (SRAM) or very dense (DRAM), they lose their content when power supply is switched off. Non Volatile Memorys (NVMs) such as Flash memories or Read-Only Memory (ROM) are able to balance the less-aggressive (with respect to SRAM and

DRAM) programming and reading performances with data persistance. Their content can be electrically altered but it is preserved when power supply is switched off.

The history of non-volatile memories began in the 1970s, with the introduction of the first EPROM memory (Erasable Programmable Read Only Memory). Since then, non-volatile memories have always been considered one of the most important families of semiconductors and, up to the 1990s, their interest was tied up more to their role as a product of development for new technologies than to their economic value.

Flash memories [21] are EEPROMs where the entire chip or subarray within may be erased at one time. There are many variants of Flash, but nowadays production is dominated by two types: NAND flash, which is oriented toward data-block storage applications, and common ground NOR flash, which is suited for code and word addressable data storage.

They both exploit the Floating Gate (FG) transistor, but differ in the way of performing operations and in the interconnections among cells. The NOR flashes support byte-addressable access, while the NAND flashes do not. The NOR flashes are usually used in systems that needs to boot out of flash, execute code from it, and store only small amounts of data because it supports byte-addressable operations and a fast read speed. Furthermore, since NAND flashes provide a higher capacity and a faster write speed than NOR flashes, the former ones are widely used for data storage applications. Table 1.1 briefly sums up the main characteristic of these types of flash-memory. This thesis work targets NAND flash memories, only.

	Standby/ Active Power	**Cost per bit**	**R/W/E Speed**	**Capacity**	**Endurance**	**Code Execution**	**Interface**
NAND	Med/Low	Low	Med/High/Med	High	Med	Hard	I/O-like
NOR	Low/Med	High	High/Low/Low	Low	Med	Easy	SRAM-like

Table 1.1: NAND Vs NOR flash-memory

The widespread usage of NAND flash memory technology [16] has faced a surprising increment, far beyond what it was originally expected, mainly thanks to the advances in the manufacturing processes (20nm NAND flash devices are currently available [105]). These advances are producing enormous gains in high data transfer rate, low power consumption, and shock-resistance when compared with traditional electro-mechanical

Hard Disks (HDs). The cell size of NAND memory is rapidly scaling down, approaching 4 F^2 where F is the minimum lithographic feature currently available in the manufacturer process. NAND flash technology results also to be cheaper than DRAM even if RAMs feature better read/write operation latencies. A flash-based memory experiences a prolonged endurance with respect to electro-mechanical technology which, due to mechanical parts, is more prone to failures [99] (Table 1.2).

	Present Density	**Cell Size (SLC)**	**MLC Capacity**	**Program Ener./Bit**	**Access Time (W/R)**	**Endurance**
HDD	750Gb/in^2	(2/3)F^2	No	0.3J	9.5/8.5ms	Low
DRAM	80Gb/in^2	$6F^2$	No	2pJ	10/10ns	High
NAND	550Gb/in^2	$4F^2$	4bits/cell	10nJ	200/25us	Med

Table 1.2: Comparison among memory technologies [95]

These motivations push NAND flash memories to be well suited for data storage in several domains that range from smartphones to mission-critical datacenter machines, and from desktops to automotive industry. At the same time, the increasing demand for high capacity and low cost led researchers to explore alternative solutions in the architecture of NAND flash memory. In order to both reduce the cost per bit and to increase the memory density, Multi Level Cell (MLC) technology has been applied to NAND flash devices. While the Single Level Cell (SLC) can store just a single bit per cell, an MLC device is able to store more than 1 bit per cell, resorting to different voltage levels.

A popular example that features the advent of MLC NAND flash memories are Solid State Drive (SSD) which are nowadays widely adopted in the market of mass memory devices. Flash-based SSDs typically consist of multiple NAND flash chips and a controller charged of data management and communication with the host machine. When compared to HDs, SSDs have several superior properties such as small form factor, light weight, low power consumption, and shock resistance. In terms of area density, HDs are still superior, even if NAND flash capacity can be further increased by exploiting the MLC technology (i.e., the capability to store more bits per cell).

NAND flash memories are also a mandatory component to provide high-density and low-cost data storage in the huge world of embedded systems. It is worth noticing here that NAND flash technology is mainly adopted as a storage device rather than boot or

main memory. This choice is due to its internal architecture and logical organization. In fact, NAND flash memories do not allow to access randomly any locations in the memory, but only groups of bytes, called *pages*. The page is, thus, the smallest portion of data that can be read/written. It is worth to point out here the asymmetric nature of the read and write operation. NAND flash technology implements the erase-before-programming approach, which requires to erase *block* (i.e., a group of pages) before programming the selected page. A block is the smallest area that can be erased. This impacts actual programming time of NAND memories, that results to be greater than reading one.

Since the flash memory storage capacity roughly doubles every 18 months [50], designers have to face additional challenging performance and reliability problems. MLC technology has further exacerbated these problems. MLC flash memories require higher programming time and provide reduced endurance when compared to old SLC devices. Reliability issues are critical as well because the more logical levels are stored in the cell, the more margins among voltage levels are reduced. This results in several sources of error such as overprogramming, program disturb, charge loss, charge leakage between neighboring cells or charge trap in floating gate oxide, that can more frequently affect memory functionalities and, as a consequence, cut down the number of program/erase cycles (P/E cycles) per block [98]. The *write endurance* is usually defined as the number of P/E cycles that can be applied to a block of flash memory before the storage media becomes unreliable.

The Raw Bit Error Rate (BER) i.e., the fraction of bits that contain incorrect data before applying fault tolerant mechanisms, is commonly used to estimate the inherent reliability of flash memories. The RBER of a MLC flash memory is around 10^{-6} [47], at least two orders of magnitude worse than those of a SLC device [51]. These problems can be further exacerbated by the file systems behavior, mainly due to the need of frequent writes of small amount of data, that can overbalance the memory wear-out upon specific areas of the memory [55].

All the aforementioned reliability issues must be tackled to increase endurance by typically resorting to different approaches at software, architecture, and device level.

At architectural level, fault tolerant mechanisms such as ECC are systematically applied. The choice of the most suitable error correcting schema is tightly application dependent. For this reason, practical ECC solutions are typically market segment-specific

and range from derivatives of the Hamming code [98] to the BCH [103] or the Reed-Solomon code [35]. Of course, by resorting to ECC, an acceptable reliability level is achieved while degrading read/write operation latency.

At device level, the choice of the proper high-voltage sub-system of the Flash memory device highly impacts in terms of reliability and performances. It is in charge of generating the voltage waveforms for flash cell read, program and erase operations, and for address decoding. Its operation is regulated by a microcontroller embedded in the flash device itself. In order to accurately fulfill the aforementioned operation, a standard algorithm is usually exploited in NAND Flash memories: the Incremental Step Pulse Programming (ISPP) [26]. Different ISPP versions exist and each one has a different impact on the reliability of the memory.

At the same time, *wear leveling* techniques [32, 34, 41, 49, 125] are used to distribute data evenly across each block of the entire flash memory, trying to level and to minimize the number of erasure cycles of each block.

It is straightforward that this is a huge design space in which each adopted solution and their combined usage differently moves the trade-off among performances, reliability, and power consumption.

State-of-the-art NAND Flash devices are tightly cost-optimized structures and the internal operations of the memory are mostly defined at design-time to cope with the industry standards (i.e., ONFI [9]). In this scenario, designers can just statically trade-off reliability and performance according to target application requirements.

However, the run-time reconfigurability, the adaptivity, and the resources optimization features requested by nowadays computing systems collide with this paradigm, which is therefore rapidly running out of steam[29]. New mobile usage models in today's complex embedded systems require the execution of multiple applications on the same device with seamless integration of safety- and time-critical functionalities with non-critical functionalities. This demands for increased run-time re-configurability of hardware blocks.

Automotive embedded systems represent a typical example. Sophisticated features such as autonomous driving and advanced driver assistance systems demand for high-level dependability and safety, while applications such as the anti-pinch control (i.e., a technique to prevent an electrically operated window or door from trapping a finger as it closes) [84] require mid-level reliability. Finally, several software applications enrich the so-called infotainment automotive systems that demand for relaxed requirements.

For instance, the feature-rich global positioning system providing traffic control information, or video streaming services featured by high-end cars require low-reliability, but are in general more demanding in terms of performance. Even for SSD devices, the need to compensate for performance and reliability degradation of MLC memories will necessarily call for optimized access modes. This can be exploited either by single applications, or by just specif threads running on the system.

We clearly see a trend toward augmenting memory controllers of MLC NAND flash devices to support differentiated access modes, each one setting a differentiated trade-off point in the performance-reliability optimization space. To this extent, NAND Flash vendors are introducing, in parallel with their legacy products, new devices which envision different levels of flexibility selectable by the user, through the memory controller, at boot-time. Most of these "tuning knobs" are based on speed/power consumption optimization (i.e., by changing the memory bus interface speed as in [1]), and storage paradigm adoption (i.e., choosing SLC or MLC writing schemes [10]).

Current research is unfortunately lagging far behind when it comes to systematic approaches for integrating different tuning knobs in the memory controller to materialize a fully on-the-fly adaptive non-volatile memory sub-system.

Introducing for the first time in the literature the concept of the *Service-Oriented Non Volatile Memories* (SONVMs), the goal of this PhD thesis is to enhance the degree of run-time reconfigurability of an MLC NAND Flash controller, through the provision of user-selectable differentiated memory access modes (i.e., services). Each mode implements a specific trade-off between read throughput, write throughput, reliability, and power.

So far, just few studies try to extend run-time adaptation to the NVM sub-system. [120] presents a first attempt of analyzing joined limited adaptation of the physical layer of the flash based on a programmable programming step voltage coupled with programmability of the ECC. Nevertheless, to the best of our knowledge, this is the first time that run-time adaptation is extended to the full NVM sub-system and a comprehensive study to analyze the effect of this adaptability considering a wide set of benchmarks is carried out. It pushes flash controllers beyond the poor flexibility of the current synthesis-time or boot-time tuning options made available by device manufacturers. Rather than focusing on implementing differentiated memory access modes with minor controller circuitry, this research work focuses on the characterization of the tuning range achievable with such modes.

In particular, the proposed solution envisions adaptivity at two different layers: architectural level and device level.

1.1 Architectural level adaptivity

The architecture layer adaptivity is based on adaptive ECC decoding structure. It implements a Bose-Chaudhuri-Hocquenhem (BCH) ECC with programmable correction capability [53]. BCH codes are a family of ECCs largely applied to NAND flash memories [38]. BCH codes are less complex than other ECCs and provide high code efficiency. Moreover, errors in flash memories are in general non-correlated and BCH codes are particularly efficient in this situation.

The construction of a BCH code is based on Galois field GF(2^m). Given a finite Galois field $GF(2^m)$ (with $m \geq 3$), a t-error-correcting BCH code, denoted as $BCH[n,k,t]$, encodes a k-bit message to a n-bit codeword by adding r parity bits to the original message. The number r of parity bits required to correct t errors in the n-bit codeword is computed by finding the minimum m that solves the inequality $k + r \leq 2^m - 1$, where $r = m \cdot t$.

The adaptable ECC block employed in this thesis is composed of both encoder and decoder modules and makes it possible to dynamically change its correction capability in a range between 1 and a maximum value.

The BCH encoder computes the r parity bits for a $k-$bit block of data. Parity bits correspond to the reminder of the division between the message and the code generator polynomial. This computation simply requires a $r-$bit linear feedback shift register (LFSR) with characteristic polynomial equal to the code generator polynomial. Programmability of the correction capability is achieved resorting to a parallel LFSR able to support different characteristic polynomials.

The BCH decoder identifies the position of erroneous bits of the codeword. This operation is more complex than the encoding. It requires three computational steps: (1) syndrome computation, (2) error-locator polynomial computation, and (3) error locations search. Each step is performed by a dedicated hardware block.

The *syndrome computation block* computes the $2xt$ syndromes of the codeword to decode. The computation of each syndrome requires a $m-$bit LFSR followed by a combinational network performing GF evaluations [101]. If all syndromes are null the codeword is error-free and the decoding stops. If not, the errors must be identified. To obtain

programmability, the syndrome block is designed to compute the maximum amount of syndromes. Depending on the programmed correction capability, only a proper subset of these computation blocks is then enabled.

The *error-locator polynomial computation block* computes the error-location polynomial whose roots represent the inverse of the error positions in the codeword. We implemented the inversion-less Berlekamp-Massey (iBM) algorithm proposed in [141] which is able to compute the error-locator polynomial in $2t$ iterations. The iBM algorithm is intrinsically programmable as long as one guarantees that internal buffers and hardware structures are sized to deal with the worst case design, and the number of iterations of the algorithm is set to $2t$.

Finally, the *error location search block* computes the roots of the error-locator polynomial that identify erroneous bits. This is the most complex and time intensive process of the decoder, since it basically requires evaluating the error-locator polynomial into each element of GF(2^m). We implemented a parallel *Chien machine* with fast skipping, able to perform more than one evaluation per clock cycle [44]. A Chien machine is mainly a grid of GF multipliers where the number of columns equals the correction capability t and the number of rows equals the parallelism. To obtain programmability, we designed this block for the worst case scenario, then enabling only a subset of the columns depending on the programmed value of t.

Given the enormous complexity to design an adaptable BCH-based ECC Intellectual Property Core (IP-Core), mainly due to the plethora of potential implementations, ranging from fully parallel to fully sequential ones, an architectural exploration task is required. The target is to design and implement an IP-Core able to efficiently scale in terms of power and latency when the required correction capability changes. Of course, design constraints such as area occupation and the total amount of generated parity bits have been taken into account to fit with available resources. To cope with this complexity, an advanced ESL tool called ADAGE [54] has been developed. ADAGE allows the automatic generation of BCH-based ECC IP-Cores with adaptable correction capability, supporting a systematic analysis and exploration of different architecture alternatives. This environment is strongly intended to be user-driven, automatic, and parametric. A complete framework for validating the correctness of the generated BCH hardware architecture is automatically generated, as well.

1.2 Physical level adaptivity

The physical layer adaptivity is based on an adaptive high-voltage sub-system of the Flash memory device. To better clarify the contribution of this thesis, it is worth briefly introducing here how the programming mechanism works. Commonly, to force the memory cell into different logical state, a predefined voltage level V_{th} is applied to the memory cell itself. In order to control the programmed V_{th} of the flash memory cell, a bit-by-bit program verify algorithm called ISPP is used. A voltage step (whose amplitude and duration are predefined) is applied to the gate of the cells. Afterwards, a Verify operation (i.e., threshold voltage Read) takes place in order to check if the cells V_{th} have exceeded a predefined voltage value V_{vfy} (in MLC architectures more than one Verify level is present) [98], [25]. If the Verify is successful, the cells have reached the desired distribution level and they are excluded from the following pulses. Otherwise, another cycle of ISPP is applied to the cells, where the programming voltage is incremented by a specified offset commonly called ΔISPP.

Due to technological variations, V_{th} is not perfectly related to the amplitude of the ISPP pulse. There are "fast" cells that reach the verify level with few program pulses and "slow" cells that require more pulses. Both behaviors represent a threat for the reliability of the program operation. A solution for increasing ISPP programming accuracy has been presented in [97], [98]. This algorithm exploits a Double Verify (DV) approach improving the RBER on scaled devices while reducing the write performance.

Another concern of MLC architectures is to decrease the write throughput performance mismatch against SLC memories. Both the ISPP-Standard Verify (ISPP-SV) and the ISPP-DV feature a large number of verify operations per single ISPP step, even if the memory cell is far from the V_{vfy} level. An interesting solution to avoid unnecessary verify operations is to use the Reduced Verify (RV) approach [98]. The reliability is now traded for increased programming speed, as this write methodology may be less robust against page-errors.

In current flash devices controllers, the programming algorithm is set at fabrication time, thus preventing run-time adaption. Our device layer is designed to be able to switch on-demand among the three ISPP versions. For this purpose, an extensive modeling, simulation and implementation framework has been set up for the analog part to capture how different program algorithms impact the RBER, the power consumption, and the write throughput of the memory.

The simulation environment is composed of two distinct modules: (a) the high-voltage subsystem of the memory, including the charge pumps, and the voltage regulators exploited for the generation of the voltages required for the programming algorithm (including the verify stage), and (b) a compact model for NAND Flash memories with array simulation capability.

Aforementioned parameters have been characterized by means of the developed simulation framework. Such parameters are derived as a function of the Program/Erase cycles of the memory, thus enabling lifetime-wide assessment of memory features.

1.3 Cross-layer optimization framework

After having considered the flexibility and the trade-offs in the physical layer and in the ECC sub-system in isolation, this thesis aims at acting upon their parameters at the same time to show unprecedented degrees of adaptivity to application requirements in the reliability/performance/power optimization space, thus identifying a set of differentiated access modes that can be configured in the memory controller [144], [20].

For this purpose, an extensive modeling, simulation and implementation framework has been set up: EF^{3}S, an easy-to-use, highly configurable, and modular tool [56]. EF^{3}S is an advanced EDA tool which aims at supporting the design of flash-based systems. It 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. This framework enables an accurate quantification of the trade-offs between the quality metrics of NVMs accesses on a set of real-life workloads and benchmark applications.

To appreciate the benefits of differentiated flash access modes on the execution of a set of real applications, the Filebench benchmark [7] has been selected for our analysis. It is an open source File System benchmark originally developed by Sun Microsystem and now by FSL (File systems and Storage Lab) group of the Computer Sciences Department of the Stony Brook University (USA). It provides a large variety of behaviors, also named *personalities*, specified using the Workload Model Language (WML) [134]. They either perform simple file I/O operations, or emulate complex I/O activities. Among the available personalities we selected three benchmark applications featuring different ratio between the number of read operations and the number of write operations.

Our analysis confirms that a wide range of access modes, each meeting highly differ-

entiated requirements across the embedded and the high-performance computing domains, can be achieved through a cross-layer approach. In particular we demonstrated that combining settings at physical and architectural levels in an MLC NAND flash subsystem holds promise of exposing unprecedented trade-offs between performance, reliability and power for memory access.

This opens up new perspectives for a NAND flash device in real-life systems.

Summarizing, the following key contributions can be identified as milestones of this thesis:

- an adaptive flash physical layer capable of different run-time selectable programming algorithms;
- the design of an adaptive ECC sub-system with run-time tunable correction capability;
- a definition of cross-layer memory access modes spanning the reliability-performance-power trade-off;
- a quantification of such trade-offs via physical-architectural co-simulation;
- an extended set of options to physically realize the proposed architecture in actual memory controllers.

1.4 Thesis organization

This thesis presents an innovative and never explored access modes for NAND flash memories and defines a new design paradigm called *Service-Oriented Non Volatile Memories*. It results in a valuable characterization of an unprecedented degree of flexibility through the combined approach of the architectural-level adaptivity and the physical-level adaptivity. The thesis is organized as follows.

Chapter 2
Overview on NAND flash memories: from technology to system level

Chapter 2 introduces the main technological features of NAND flash memories. From a technological standpoint, all NAND flashes are not created equal and may differ in cell types, architecture, performances, timing parameters, command set, etc. However,

they all share the following general organization. A NAND flash-memory is usually partitioned into blocks. Each block has a fixed number of pages of a fixed size. A block is the smallest unit for erase operations, while read and write operations are done in terms of pages. Therefore, a page can be erased only if the whole block it belongs to is erased.

Chapter 3
NAND flash memory reliability

Since NAND Flash memories are experiencing an even more faster technological scaling down, reliability requirements are becoming more and more difficult to guarantee. In order to meet these demanding requirements, designers adopt different strategies at design time to improve the overall reliability level whereas degrading memory performances. Basically, improving reliability results in prolong endurance (i.e., the allowed number of repeated program/erase cycles) and retention (i.e., the ability to maintain and retrieve the stored information). In this chapter we outline the NAND technology reliability issues. It mainly focuses on its intrinsic physical limits and it gives an overview of the most important reliability issues affecting the floating gate memory technology. At the end of this chapter it will be clear how many challenges designers have to face with and how each employed strategy definitely impacts on the reliability, power and performance.

Chapter 4
Adaptable flash physical management sub-system (physical-level adaptability)

The physical management sub-system of the memory controller interacts with the high-voltage analog circuitry of the NAND flash memory that generates the voltage waveforms for cells read/program/erase and for address decoding. A standard algorithm named ISPP is usually exploited to accomplish this operation [98]. In current flash device controllers, the programming algorithm is set at fabrication time, thus preventing run-time adaptation.

The programming algorithm and voltage waveforms affect both reliability and performance. Compared to the typical ISPP algorithm [106], the double verify algorithm can improve the RBER on the same device or sustain the RBER on scaled devices while reducing the write performance [98]. Differently, the reduced verify algorithm [98] is able to improve writing performance while increasing the RBER.

Chapter 4 first describes the programming algorithms that are commonly used in practice and then introduces an adaptive flash physical optimization approach capable of different run-time selectable programming algorithms.

Chapter 5
Adaptable BCH codecs for NAND flash memories (Architectural-Level Adaptivity)

The architectural adaptivity is got via an adaptive ECC sub-system. Chapter 5, thus, presents an adaptable BCH based design for NAND flash memory. It can dynamically adapt its correcting capability to the specific operational conditions of the memory. The number of parity bits and the decoding complexity can in fact be adapted depending on how many errors have to be corrected. The ECC sub-system is characterized to show the different trade-offs offered by its programmability.

A custom design environment, called ADAGE, has been developed to support the generation of such an architecture.

Chapter 6
Cross-layer optimized NAND flash access modes

While in the Chapter 5 and in Chapter 4 we have considered the gain achieved by resorting to an adaptable ECC and the flexibility in the physical layer, respectively, Chapter 6 presents a systematic approach able to provide an unprecedented degrees of adaptivity to application requirements in the reliability/performance/power optimization space, thus identifying a set of differentiated access modes that can be configured in the memory controller. To appreciate the benefits of differentiated flash access modes on the execution of a set of real applications, a sophisticated EDA tool called EF^{3}S (Evaluation Framework For Flash-based System) has been developed.

Appendixes A and B

Appendix A provides a generic overview of the NAND flash model employed in this thesis. Appendix B overviews the ECCs design dimensions and issues.

Conclusions

Our research work, summarized in this PhD thesis, demonstrated that combining settings at the physical and architectural level in an MLC NAND flash sub-system holds

promise of exposing unprecedented trade-offs between performance, reliability, and power for memory access. Our cross-layer optimization of NAND flash controllers includes the correction strength of an adaptive ECC framework and the programming algorithm of memory cells, thus yielding access modes for ultra-high performance, ultra-high reliability or intermediate trade-off requirements.

CHAPTER 2

OVERVIEW ON NAND FLASH MEMORIES: FROM TECHNOLOGY TO SYSTEM LEVEL

Contents of this chapter

This chapter aims at introducing the main concepts related to NAND flash memories, ranging from basic technological aspects to description of flash-based systems in a bottom-up fashion to provide readers the fundamental elements to understand the following chapters. Of course, the presentation will be based not on specific implementations but on general structure types, focusing on the main mechanisms and the main issues.

In the Section 2.1, first, a wide overview about NAND flash technology and expected trends in terms of capacity, cost, and performance are presented. Then the focus is placed on basic components making up NAND flash memories i.e., the memory cell. So, from a circuit level perspective, the whole NAND memory layout is provided, show-

ing how actual cells are grouped and connected (Section 2.2). The detailed description of the NAND flash memory architecture is followed by an analysis of basic NAND functionalities (Section 2.2). Considering the actual trend, MLC principles and characteristics are dealt with, as well (Section 2.4). Moving towards higher levels, Section 2.5 presents a logical organization, in a more organic view, of logic components composed of NAND flash memories. After having covered main topics related to NAND flash memory technology, the last Section 2.6 aims at summarizing challenges and issues when NAND flash memories are integrated into a complex system.

2.1 A Big Picture

The NAND Flash architecture was introduced by Toshiba in 1989. The increasing demand for high-speed storage capability both in consumer electronics (e.g., USB flash drives, digital cameras, MP3 players, solid state hard-disks, etc.) and mission critical applications, makes NAND flash memories a rugged, compact alternative to traditional mass-storage devices such as magnetic hard-disks. The NAND flash technology guarantees a non-volatile high-density storage support that is fast, shock-resistant and very power-economic. At higher capacities, however, flash storage can be much more costly than magnetic disks, and some flash products are still in short supply. Nevertheless, analysts predict that in the next years Flash memory revenues will rise to $22.4bn (see Fig. 2.1).

Figure 2.1: NAND flash memory revenue forecast

The efficient architecture of the NAND Flash allows denser layout with larger capacity on a given die size, in combination with a simpler production process. This enable the more cost effective NAND flash to replace other storage solutions, mainly based on legacy HDs. NAND Flash is organized into blocks and pages and erased on a block basis. A block consists of 64 or more pages [98]. However, caused by its internal organization, a NAND flash memory is not suitable for random access. This is the main reason to employ it as a data storage medium and not as main memory, where RAMs still play a key role.

Despite of other semiconductor memories that feature new process technology each 2 year, NAND flash technology is on 1 year cadence. The faster scaling down w.r.t. other memory technologies has thus resulted in the capability of integrating more bits per cell, replacing the older SLC technology with the more cost-effective MLC NAND memories (see Fig. 2.2). Currently, these motivations make MLC NAND technology the lowest cost semiconductor memory with none of the other memory technologies even close to being cost competitive.

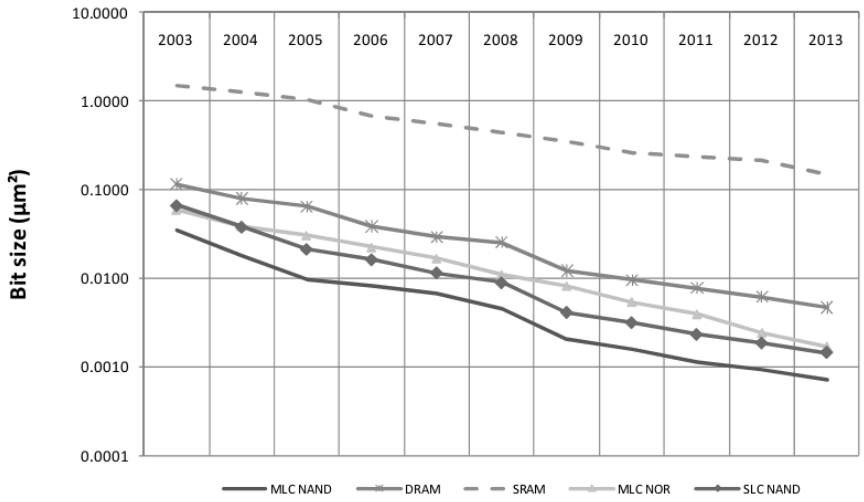


Figure 2.2: Bit Size Trend [98]

However, lower costs do not come for free. They come at the expense of reduced performance and endurance. The comparison between SLC and MLC, shown in the Fig. 2.3) exhibits that the more bits per cells, the more significant degradation in endurance and performances is experienced. The interested reader may refer to [36, 48] for more

detailed comparisons between SLC and MLC technology.

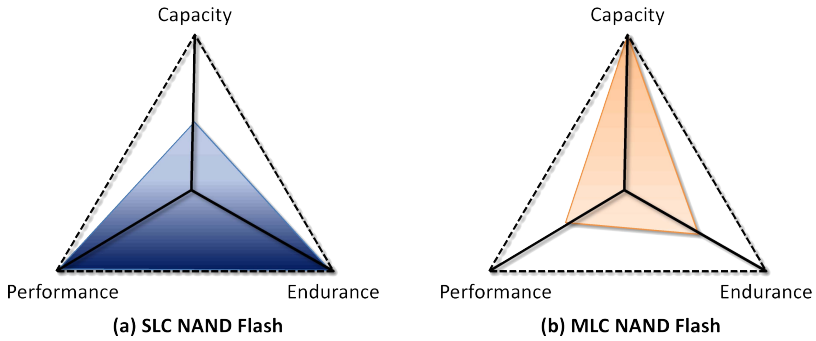


Figure 2.3: Comparison of SLC and MLC flash memories [58]

For these reasons, in actual memory controllers, several mechanisms at different abstraction layers are currently employed to ensure a desired level of reliability for a given application. At firmware-layer, sophisticated wear-leveling algorithms are in charge to evenly spread data into memory, while, at the same time, powerful ECCs IP-Cores protect data adding redundancy, making them more robust. Of course, even if the aforementioned mechanisms improve the overall reliability, they heavily impact on performances and power consumption. This is a multi-dimensional space that a NAND flash-based system designer have to face with, trading-off performances, reliability, and power according to application requirements.

2.2 Array Structure

In this section the key element to persistently store data in NAND flash memories is analyzed: the memory cell. Then a systematic view about the NAND flash memory array is given, focusing on circuit layout and interconnections of memory cells.

The most popular Flash memory cell is based on FG technology [98], whose cross section is shown in the Fig. 2.4. A MOS transistor is built with two overlapping gates rather than a single one: the former one is completely surrounded by oxide, while the latter 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 volt-

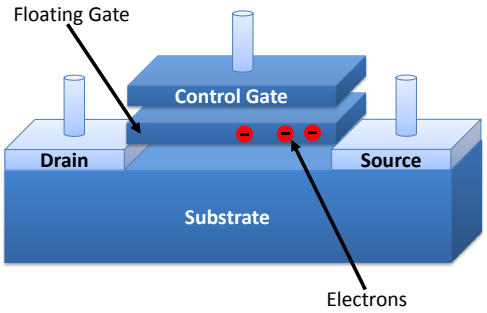


Figure 2.4: Floating gate memory cell

age to cell's terminals, it is then possible to discriminate two storage levels (e.g., whenever gate voltage is higher than the cell's V_{TH}, the cell is on ("1"), otherwise it is off ("0")).

The memory cells are grouped to form a matrix in order to reduce silicon area occupation. A fixed numbers of memory cells are connected in series to form a NAND string, as shown in the Fig 2.5. In the NAND string two selection transistors ensure the connections to the source line (by M_{SL}) and to the bitline (by M_{DL}). More NAND strings share the bitline contact. Control gates are linked by means of wordlines (WLs).

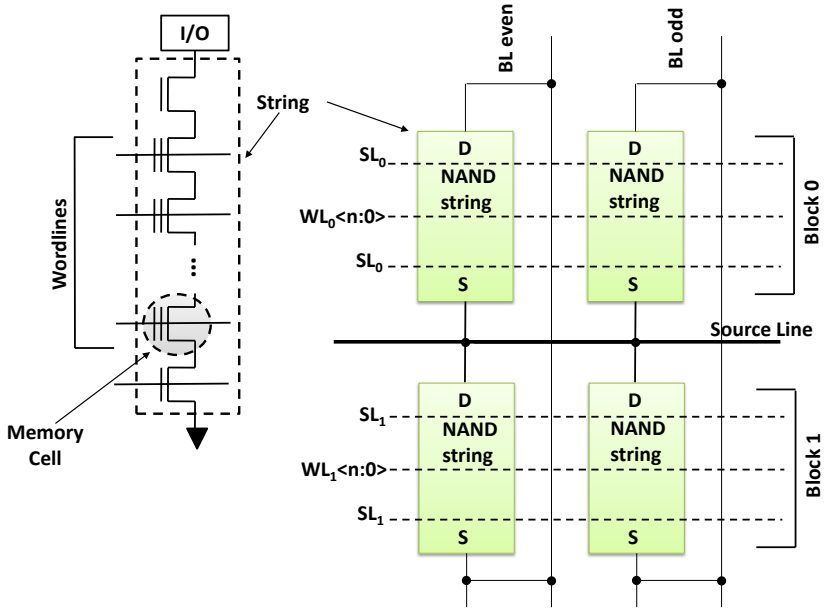


Figure 2.5: NAND string (left), NAND array (right)

Logical *pages* are composed of cells belonging to the same wordline. The page is the smallest storage unit when performing read and programming operations. The storage capabilities are strictly related to both the number of pages per wordline and the number of bits that can be stored into a single cell. Considering for example a SLC device with 4kB page, it has a wordline of 32,768 cells.

A flash *block* is made up by wordlines shared by a group of NAND strings. A block is the smallest unit on which performing the erase operation. In Fig. 2.5 two blocks are shown, composed of $WL_0 < n : 0 >$ and $WL_1 < n : 0 >$ respectively, where n is the total number of wordlines per block.

Even if the memory array is the main component of a NAND flash memory, other components are needed to perform read, program, and erase operations. The memory array is commonly split into two *planes* to better organize the entire layout. The additional circuits (see Fig 2.6) are:

- **Row Decoder**, located between the planes. This circuit aims at properly biasing all the wordlines belonging to the selected NAND string;
- **Sense Amplifier** has the task of converting the current sunk by the memory cell to a digital value. All the bitlines are linked to sense amplifiers;
- **Peripheral circuits** are composed of charge pumps, voltage regulators, logic circuits, and redundancy structures. They are in charge to generate the voltage waveforms for cells read/program/erase and for address decoding;
- **PADs** are employed to communicate with the external word.

2.3 Basic NAND functionalities

This section shortly introduces the basic NAND operations, which are program, erase and read. Pages are the smallest storage unit when performing read and programming operations. Pages already written with data must be erased prior to write new values. The erase operation is performed on a block basis.

2.3.1 Program

The programming operation logically corresponds to the 1 -> 0 transition and physically implies to move some charges (electrons) into the floating gate. For such reasons, the

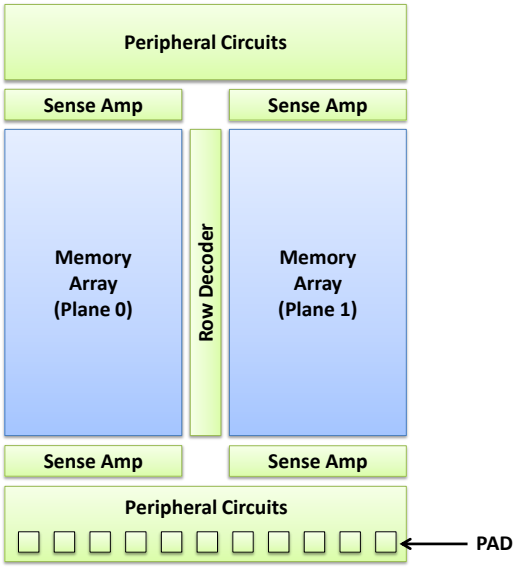


Figure 2.6: NAND Flash Memory Layout

programming operation of a NAND flash memory exploits the quantum-effect of electron tunneling (referred to as Fowler-Nordheim tunneling [61]) whenever a electric field is applied. The higher the intensity of the electric field, the higher the probability to inject a sufficient number of electrons. This implies that higher electric field and consequently higher voltages enable a more efficient program operation. This strong requirement strongly impacts the memory endurance, as the oxide degradation is impacted by these voltages.

The main benefit is the low current required (i.e., nA per cell). This is the main reason that make the Fowler-Nordheim effect mainly suitable for programming many cells in one shot, as required by NAND page sizes.

Commonly, the algorithm used to program the cells of a NAND memory page is a *Program & Verify* algorithm, which checks whether the cell has reached the target voltage threshold or not.

Due to the organization of the memory array whose cells are grouped in a matrix, all the cells along the wordline are biased at the same voltage even if they are not intended to be programmed. As shown in the Fig. 2.7, this fact introduces an additional reliability concern known as *Program Disturb*, discussed in chapter 3.

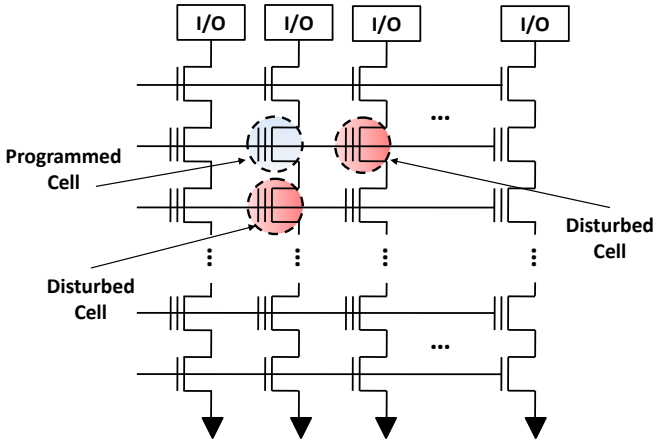


Figure 2.7: Program disturb

2.3.2 Read

The reading operation is designed to address a memory cell and extrapolate the information stored therein. The stored information is associated with the cell's threshold voltage V_{TH}. In Fig. 2.8 the logic levels of a SLC associated with threshold voltage distributions of cells are shown. If the cell has a V_{TH} belonging to the erased distribution, it contains a logic "1", otherwise, it contains a logic "0". Generally, cells containing n bit of information have 2^n different levels of V_{TH}.

Considering a fixed gate voltage, the cell current is a function of its threshold voltage. So, by resorting to a current measure, it is possible to understand to which V_{TH} distribution the memory cell belongs. Nevertheless, even this operation suffers from disturbs, since a memory cell belongs to a string including other cells. In order to avoid that during the read operation the cell's logical value be influenced by the neighbored cells, the unselected memory cells must be operated so that their threshold voltages do not affect the current of the addressed cell. As a result, their gate must be driven to a voltage (commonly known as V_{PASS}) higher than the maximum possible V_{TH}.

2.3.3 Erase

The erase operation logically corresponds to the 0 -> 1 transition and physically implies to remove the charges from the floating gate. The erase operation resets the information of all the cells in one block simultaneously. Applying a high electric field across the

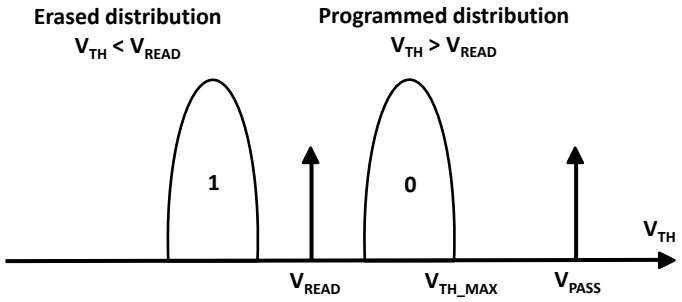


Figure 2.8: Threshold voltage distributions

tunneling dielectric of the cells in the selected block, the stored electrons are tunneled out of the floating gate to the substrate, lowering the cells' V_{TH} (see Fig. 2.9). Since flash memories wear out after a certain number of erasure cycles (*endurance*), if the erasure cycles of a block exceed this number, the block cannot be considered anymore reliable for storing data. A typical value for the endurance of an SLC flash memory is about 10^6 erasure cycles.

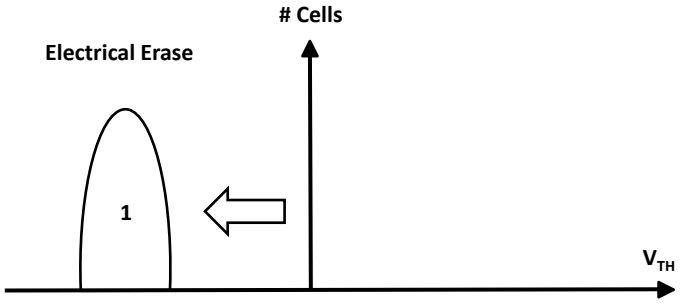


Figure 2.9: Erase state

2.4 MLC principle and characteristics

So far, both the general principle of operations and the array structure of a NAND flash memory have been presented avoiding a fully description of physical details, considered out of the scope of this thesis. For the sake of simplicity, the aforementioned concepts have been presented resorting to the SLC approach, but it is worth mentioning the MLC technology as well, as it is widely applied in most of today NAND flash memories. More-

over, the conceived adaptable NAND flash memory sub-system, presented in this thesis, exploits the MLC technology.

In principle, the MLC technology does not imply any modification of the cell itself but it uses its storing capability in a more powerful way. Storing charges in the floating gate changes the threshold voltage level of the flash cell. Up to now we have assumed that only two situations were possible with two well defined meanings:

- Presence of charges in the floating gate: corresponding to "0".
- Absence of charges in the floating gate: corresponding to "1".

This kind of usage of the flash memory cell is commonly called SLC. This means that every single cell can store one and only one bit of information, experiencing only two values 'high' and 'low' corresponding to the presence and absence of charges in the floating gate.

The threshold of the cell is function of the number of electrons stored in the floating gate and it increases with the number of stored electrons. This principle is exploited in the so called MLC flash memory technology. The idea is to have more that two levels of valid thresholds, thus allowing to store more than one bit in the cell. Each threshold level corresponds to a specific state of the cell and each state is assigned a specific binary code. Having a certain number of possible states for a cell S, the number of bits that can be stored N is equals to $\lceil log_2(S) \rceil$. For example, with four possible states we can store two bits of information assigning two bits at each of the four threshold levels of the cell.

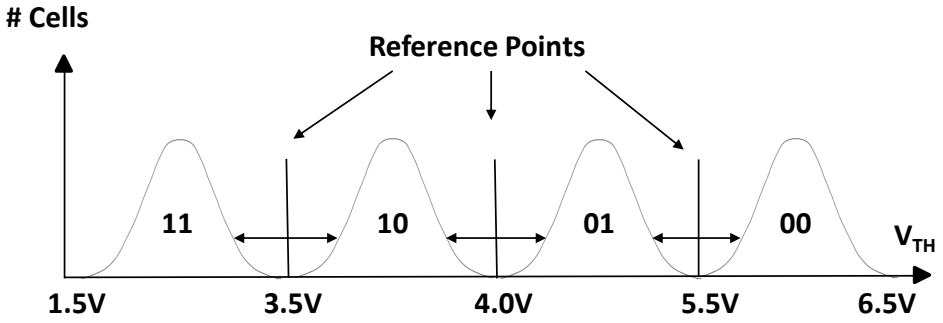


Figure 2.10: MLC-threshold distribution

This technique increases the equivalent density of bits in the memory, thus reducing the overall cost per bit. Unfortunately, this improvement in the capacity of the device

has a set of penalties in terms of complexity of the needed circuitry, reliability and performance.

The main consequence of using MLC programming technique is the reduction of the delta between each level of threshold (Fig. 2.10). As a consequence of this, a more accurate and controlled programming mechanism is needed. Typically, the correct level of threshold is achieved by a controlled series of pulses on the control gate of the selected cell employing the ISPP technique. This also leads to performance degradation in terms of program speed compared to SLC [82]. For the same reasons, the read operation in MLC is different from the normal read operation in SLC. It typically requires a more complex circuitry, while the read speeds between SLC and MLC are comparable.

This growth in complexity of the read and program operations has direct effect on the needed control logic and circuitry of the device, that become bigger than in SLC .

The reliability degradation is a direct consequence of reduction of the delta between the threshold levels. Having a general disturb on the threshold of a MLC easily can move the threshold level out of its range and therefore change the logic content of the cell. Moreover, the MLC device are more sensitive to high temperatures that cause more leakage in the cells. Combined with the increased sensitivity required to differentiate between the levels, this leakage may lead the sensors to read the wrong level. As a result, the operating temperature of MLC spans only the commercial range [2].

The endurance of SLC Flash is 10x more than MLC Flash. The endurance of MLC Flash decreases due to enhanced degradation of Si. This is the main reason why an SLC Flash is considered industrial grade Flash and while MLC Flash is a consumer grade Flash [2].

Table 2.1 provides some additional figures about these two technologies. Although some entries of the table may not be familiar to the reader, they will be addressed shortly in the sequel of this chapter.

The correct choice of the Flash memory type SLC vs MLC depends on the target system needs. If performance and durability are essential, SLC Flash are the most suitable choice. If low cost and high density are essential, MLC Flash is the right choice [2].

2.5 Logic structures

This section aims at summarizing the logic structure of a NAND flash device. Even if in the previous sections many details have already been given, hereinafter the logic organi-

	Features			*Architecture*			*Reliability*			*Array Operations*		
	Bits /cell	**Voltage**	**Bus width**	**Planes**	**Page size**	**Pages /block**	NOP	ECC-**512B**	**Endurance**	t_{READ} **(max)**	t_{PROG} **(avg)**	t_{ERASE} **(avg)**
SLC	1	3.3V, 1.8V	x8, x16	1 or 2	2,112B	64+	1	1	$<10^5$	25us	200 - 300us	1.5 – 2ms
MLC	2+	3.3V	x8	2+	4,314B+	128+	4+	4+	$<10^4$	50us	600 - 900us	2ms

Table 2.1: NAND SLC Vs MLC

zation is introduced by resorting to a systematic approach.

As outlined in Fig. 2.11, a memory is divided in pages and blocks. A block is the smallest erasable unit. Generally, there are 2^k blocks within any device. Each block contains several pages. The number of pages within a block is typically a multiple of 16 (e.g., 128). A page is the smallest addressable unit for reading and writing. Each page is composed of *main area* and *spare area*. Main area can range from 4 to 8 kB or even 16 kB while spare area is typically used for system level management (i.e., file system, ECC) and it is in the order of a couple of hundreds bytes every 4 kB of main area. Pages are usually grouped in different planes, since a flash memory with N planes can read/write and erase N pages/blocks at the same time [48].

2.6 NAND-based systems

Currently there are a plethora of flash-based devices. SSDs and flash cards are definitely the most known examples of electronic systems based on NAND Flash [65], [3], [4]. Several types are available, featuring different user interfaces, and form factors, depending on the needs of target application.

It is worth introducing here that, whenever a flash memory is integrated in a complex device, a set of activities related to the management of the flash memory itself are required. As discussed in the previous sections, memories wear out after a certain number of erasure cycles and therefore specific functions are required to prolong device lifetime. The main functions are: *Wear leveling Management* 2.6.3, *Garbage Collection* 2.6.4, *Bad Block Management* 2.6.5 which are usually implemented in form of firmware inside the memory controller or in a dedicated software layer called Flash File System, whereas ECC approaches are usually integrated as IP-Core inside the memory sub-system.

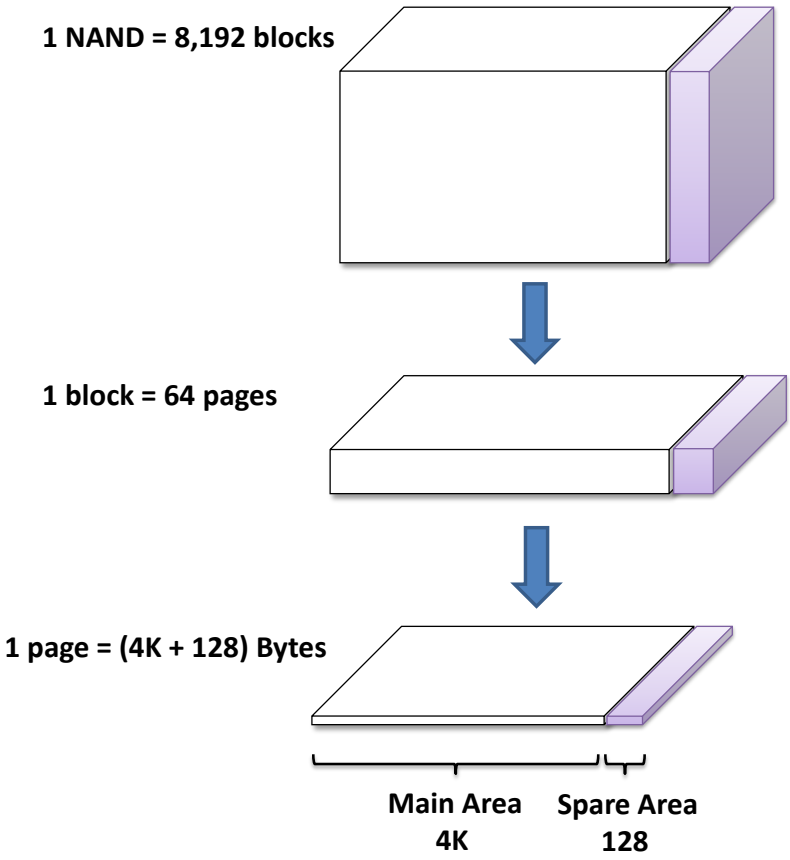


Figure 2.11: NAND logic organization

An other relevant issue is that Operating Systems (OSs) usually write data in *sectors* (e.g., 512 bytes in size) when HD is employed. Whereas NAND flash memories are written on page basis whose size is actually different from HD's sector (e.g., nowadays page size is 4 or 8KB). In addition, flash memories do not support the so-called *update-in-place* feature as each page requires to be erased before written, and the erase operation is performed on a block basis. For these reasons, a driver that works in conjunction with the Operating System is required in order to hide technological differences between flash memory and HD. to the system. The Flash Translation Layer (FTL) is in charge to accomplish this task.

This section provides first a short overview about memory interfaces currently available for NAND flash memories and their impact on the overall system performances. Then, the description of the aforementioned relevant management functions is provided

focusing on their essential role. Then an overview about the ways through which NAND flash memory can be embedded in complex system and the role of the sophisticated memory controller in the modern SSDs are pointed out.

2.6.1 Memory interface

Nowadays NAND Flash memories are pervading every type of electronic application. The availability of cheap storage memory, in combination with high densities, makes up the choice in favor of NAND Flash on board of applications traditionally linked to other types of memories (such as EEPROM and NOR) or technologies (such as Hard Disk Drives). But this is not enough when high performance applications are targeted. For these reasons NAND flash memories are expected to be further improved in terms of I/O bandwidth, as well. The I/O throughput is strictly dependent on the kind of storage interface flash memories is integrated with.

Currently there are two types of NAND Flash interfaces. The asynchronous one is similar to the regular SRAM interface, the other one is the synchronous DDR interface, which offers much higher performances than the asynchronous interface but it requires dedicated controller.

Generally, memory I/O bandwidth is determined by the memory access time, which is split into the *array access time* and the *transfer time*. The former is the time necessary to transfer the data from the NAND cells into the page buffers, while the latter is the time required to move all the data in the page buffers out of the chip through the legacy interface.

In Fig. 2.12, the flash memories access time is plotted, considering synchronous and asynchronous interfaces and different page sizes. It is possible to notice that in 1KB page devices the transfer time is in balance versus the array read time. This give the possibility of interleaving read operations from two planes, hiding, de facto, the read array time, since, during the transfer time, data are moved from the array to the page buffer. If asynchronous interface is considered, in 4KB page device this balance is completely lost. Therefore, the memory interface is an important design choice to improve performances and make NAND flash memories a key component as storage in high-end applications.

To restore the balance between read array time and transfer time it is necessary to increase the I/O throughput. On the right hand side of Fig. 2.12, it is shown the situation of 4KB devices with a DDR interface operating at 66 MHz (DDR133): the desired balance

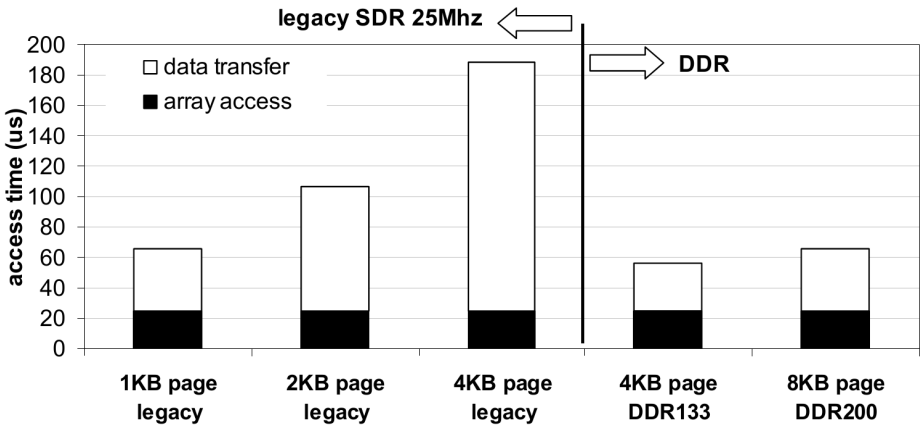


Figure 2.12: NAND memory interface: asynchronous VS synchronous [98]

is restored.

The DDR200 (200 MB/s) speed grade keeps the balance between the array and data transfer times even with 8KB page size. The picture becomes evident in Fig. 2.13 where the internal throughput capability (i.e., page size/internal access time) is plotted versus the page size. It can be shown that the legacy interface throughput was in line with the internal throughput of 1KB page NAND Flash devices.

SLC NAND with 4KB pages have an internal throughput aligned with DDR133 interface. Moving to MLC 2 bit/cell with 8KB page, the throughput still remains aligned with DDR133 because of the larger MLC internal access time. DDR400 interface roadmap is aligned with the next generation having, page size greater than 8KB and greater access time. This is in line with system interfaces roadmap evolving towards 300 MB/s rates.

For these reasons, it becomes evident that a bottleneck in the system performance is hidden in the NAND interface because data transfer was limited to 40 MB/s by the asynchronous interface. In the last years, thus, major vendors are pushing Double Data Rate (DDR) interfaces to outperform actual data transfer limits. For these reasons, our research work is focused just on synchronous NAND flash memories.

High speed NAND have seen the introduction of DDR interface in year 2008. The challenge in the following years has been the standardization of the interface among the vendors. Two solutions have been proposed as first generation of DDR NAND:

1. ONFI [119] organization has proposed the *Source Synchronous Interface* (SSI) which

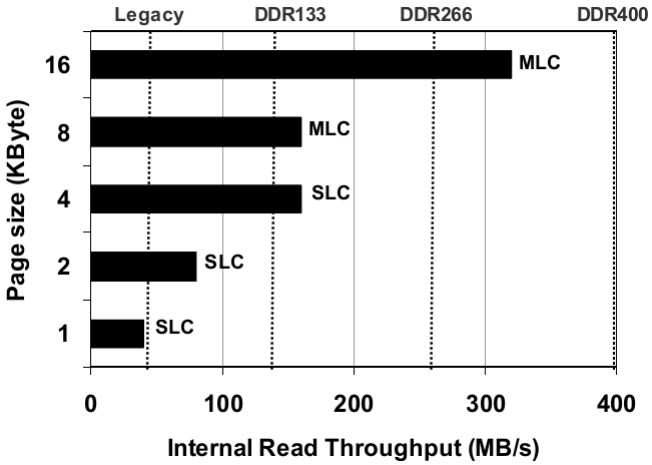


Figure 2.13: Read throughput vs page size [98]

introduces a clock and data strobe w.r.t. the asynchronous interface;

2. Samsung has proposed the *Toggle mode* NAND where only data strobe has been introduced compared to the asynchronous interface.

For the sake of simplicity, Fig. 2.14 shows the NAND pinout for SSI, only. The pins description is listed below:

- CE#: it is the Chip Enable signal. This input signal is "1" when the device is in stand-by mode, otherwise it is always "0";
- R/B#: it is the Ready/Busy signal. This output signal is used to indicate the target status. When low, the target has an operation in progress;
- W/R#: it is the Write/Read direction pin;
- CLE: it is the Command Latch Enable. This input is used by the host to indicate that the bus cycle is used to input the command;
- ALE: it is the Address Latch Enable. This input is used by the host to indicate that the bus cycle is used to input the addresses;
- CLK: it is the Clock signal;
- WP#: it is the Write Protect. This input signal is used to disable Flash array program and erase operations;

- DQ<7:0>: these input/output signals represent the data bus (i.e., 8-bit wide);
- DQS: it is an additional pin acting as the datastrobe, i.e. it indicates the data valid window.

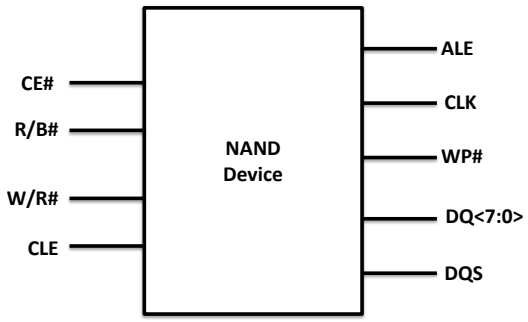


Figure 2.14: NAND Flash supporting synchronous interface

2.6.2 Flash translation layer

The flash translation layer (FTL) refers to the development of a hardware/software layer able to emulate the behavior of a traditional block device such as a hard-disk, allowing the OS to communicate with the flash using the same primitives exploited to communicate with magnetic-disks. The FTL is in charge of "translating" the typical System Calls (e.g., open, read, write) of the OS into the proper sequence of commands for the specific flash-memory chip. The OS can then write the NAND memory on a page basis, without worrying about the details of its physical implementation.

Each page of a flash is identified by both a logical and a physical address. Logical addresses are provided to the user to identify a given data with a single address, regardless if the actual information is moved to different physical locations to optimize the use of the device. The *address translation* mechanism, that maps logical addresses to the corresponding physical addresses, must be efficient to generate a minor impact on the performance of the memory. The address translation information must be stored in the non-volatile memory to guarantee the integrity of the system. However, since frequent updates are performed, a translation lookup table is usually stored in a (battery-backed) RAM, while the flash memory stores the metadata to build this table. The size of the table is a trade-off between the high cost of the RAM and the performance of the storage

system. Memories with a large page size require less RAM, but they inefficiently handle small writes. In fact, since an entire page must be written into the flash with every flush, larger pages cause more unmodified data to be written for every (small) change. Small page sizes efficiently handles small writes, but the resulting RAM requirements can be unaffordable. At the FTL, the translation table can be implemented at the level of either pages or blocks thus allowing to trade-off between the size and the granularity of the table [55].

However, traditional file systems do not take into account the specific peculiarities of flash memories, and the emulation layer alone may be not enough to guarantee maximum performance. The alternative to the block-device emulation is to exploit the hardware features of the flash device in the development of a native Flash File System (FFS) (Fig. 2.15). For efficiency reasons, this approach is becoming the preferred solution whenever embedded NAND flash memories are massively exploited [15, 135]. The literature is rich of strategies involving block-device emulation [33, 34, 73, 77, 90]. [55] offers a comprehensive comparison of available native FFS.

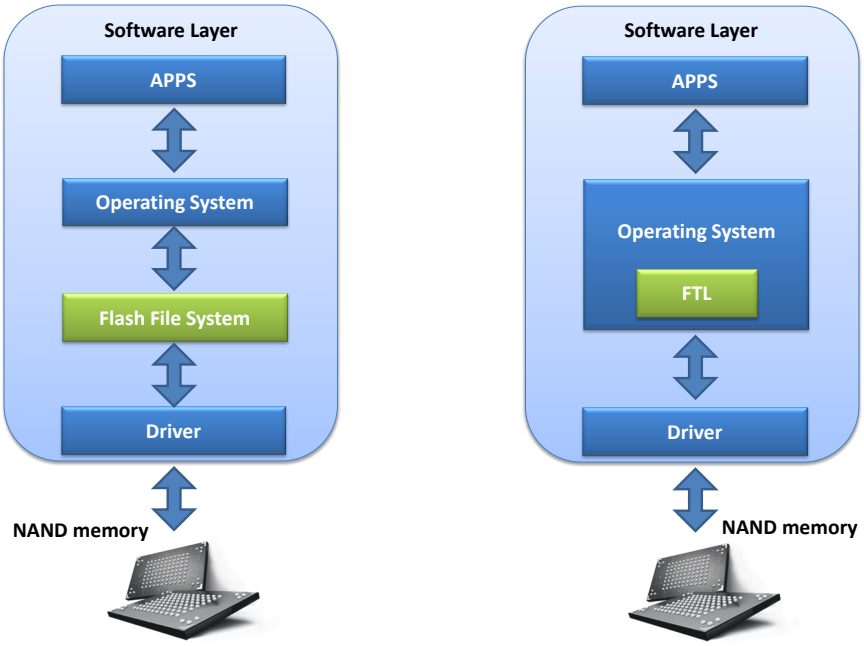


Figure 2.15: Flash File System vs Flash Tranlsation Layer

2.6.3 Wear leveling

As previously introduced, flash memories wear out after a certain number of erasure cycles (usually between 10^4 and 10^5 cycles). If the number of erasures of a block exceeds this number, the block is marked as a *bad block* since it cannot be anymore considered reliable for storing data. The overall life time of a flash memory depends on the number of performed erasure cycles. *Wear leveling* techniques [32, 34, 41, 49, 125] are used to distribute data evenly across the blocks of the entire flash memory, trying to level and to minimize the number of erasure cycles of each block. The alternative is to consider higher capacity flash-memory devices, taking care of the resulting drawbacks in terms of weight and volume [30].

There are two main wear leveling strategies: *dynamic* and *static* wear leveling. The *dynamic* wear leveling only involves those data blocks that are going to be written, while the *static* wear leveling works on all data blocks, including those that are not involved in a write operation. Active data blocks are, in general, wear-leveled dynamically, while static blocks (i.e., blocks where data are written and remain unchanged for long periods of time) are wear-leveled statically. Dynamic and static blocks are usually referred as *hot* and *cold* data, respectively. In Multi Level Cell memories it is important to move cold data to optimize the wear leveling. If cold data are not moved then the related pages are seldom written and the wear is heavily skewed to other pages. Moreover, every read to a page has the potential to disturb data on other pages in the same block. Thus continuous read-only access to an area can cause corruption, and cold data should be periodically rewritten [91].

Wear leveling techniques must be strongly coupled with garbage collection algorithms at the FTL and at the FFS level. In fact, the two tasks have in general conflicting objectives and a good trade-off must be found to guarantee both performance and endurance. Interested readers may refer to [32] for a comparative analysis of the most widely used wear leveling algorithms.

2.6.4 Garbage collection

Data stored in a page of a flash memory cannot be overwritten unless an erasure of the full block is performed. To overcome this problem, when the content of a page must be updated, the new data are usually saved in a new free page. The new page is marked as *valid* while the old page is marked as *invalid*. The address translation table is then

updated to allow the user to access the new data with the same logical address. This process introduces several challenges both at the FTL and at the FFS level.

At a certain point, the free space is going to run out. When the amount of free blocks is less than a given threshold, invalidated pages must be erased in order to free some space. The only way to erase a page is to erase the whole block it belongs to. However, a block selected for erasure may contain both valid and invalid pages. As a consequence, the valid pages of the block must be copied into other free pages. The old pages can be then marked as invalid and the selected block can be erased and made available for storage.

This cleaning activity is referred to as *garbage collection*. Garbage collection decreases the flash memory performance and therefore represents a critical aspect of the design of a native FFS. Moreover, as previously described, it may impact on the endurance of the device. The key objective of an efficient garbage collection strategy is to reduce garbage collection costs and evenly erase all the blocks.

Flexible cleaning algorithms [136], greedy policies, aging functions [41] or periodical collection approaches [131] can be adopted to minimize the cleaning cost.

2.6.5 Bad block management

As discussed in the previous sections, when a block exceeds the maximum number of erasure cycles, it is marked as a *bad block*. Bad blocks can be detected also in new devices as a result of blocks identified as faulty during the end-of-production test.

Bad blocks must be detected and excluded from the active memory space. In general, simple techniques to handle bad blocks are commonly implemented. An example is provided by the Samsung's XSR (Flash Driver) and its Bad Block Management scheme [124]. The flash memory is initially split into a reserved and a user area. The reserved blocks in the reserved area represent a *Reserve Block Pool* that can be used to replace bad blocks. Samsung's XSR basically remaps a bad block to one of the reserved blocks so that the data contained in a bad block is not lost and the bad block is not longer used.

2.6.6 Error correcting codes

Fault tolerance mechanisms and in particular Error Correcting Code (ECC) are systematically applied to NAND flash devices to improve their level of reliability. ECCs are cost-efficient and allow detecting or even correcting a certain number of errors.

ECCs have to be fast and efficient at the same time. Several ECC schema have been proposed, based on linear codes (like Hamming codes [104]) or Reed-Solomon (RS) codes [126, 127]. Among the possible solutions, Bose-Chaudhuri-Hocquenhem (BCH) codes are linear codes widely adopted with flash memories [45, 53, 79, 102]. They are less complex than other ECCs, providing also a higher code efficiency. Moreover, manufacturers' and independent studies [52, 57, 137] have shown that flash memories tend to manifest non-correlated bit errors. BCH are particularly efficient when errors are randomly distributed, thus representing a suitable solution for flash memories.

The choice of the characteristics of the ECC is a trade-off between reliability requirements and code complexity, and strongly depends on the target application (i.e., consumer electronics vs mission-critical applications) [27].

ECC can be implemented both at the software-level or resorting to hardware facilities. Software implemented ECC allow to decouple the error correction mechanisms from the specific hardware device. However, the price to pay for a software-based ECC solution is a drastic performance reduction. For this reason, available file systems tend to delegate the code computation tasks to a dedicate hardware limiting the amount of operations performed in software, at the cost of additional resources (e.g., hardware, power consumption, etc.) and reduced flexibility.

The interested reader may refer to Appendix B for more details about ECCs and BCH. Furthermore, Chapter 5 addresses the design and the practical implementation of an adaptable BCHs for NAND flash-memory.

2.6.7 How to embed NAND flash memories

In the previous sections, several technological aspects of NAND flash memories have been presented, analyzing the strategies and main solutions currently applied by vendors, from the circuit to the architectural level. The purpose of this section is to shortly summarize, from a system-level perspective, how NAND flash memories can be integrated in embedded systems.

NAND Flash is employed for code/data storage or data storage in a variety of portable and mobile applications such as cellular phones, MP3 players, digital video camcorders and personal navigation devices. NAND Flash, thus, may be embedded in applications in several ways, ranging from fully managed to raw solutions. The former are all-in-one solutions, in the sense that the flash memory is soldered together with a memory controller

in charge to manage the device wear-out hiding the system from all technological details related to the memory (e.g., internal architecture, page size). The latter, on the contrary, requires that un updated firmware into the system be able to opportunely manage the embedded raw flash. The most common strategies are as follows:

- *Raw NAND*: NAND flash chips are soldered into the Printed Circuit Board (PCB) of the device, letting the host managing wear leveling, garbage collection, FTL and so on. For instance, MP3 players employ a dedicated Flash controller to manage the raw NAND;
- *NAND with on-chip controller*: NAND flash devices are employed for both code and data storage. Contrary to raw NAND, the host is in charge to just operate the FTL. These devices (such as ONENAND from Samsung, Toshiba and Numonyx) are quite sophisticated to be exploited as main memories;
- *Multi-chip Package (MCP)*: NAND memory can be combined with faster volatile memories such as DRAM or NOR flash in order to be used both for code and data storage. MCPs are mainly used in mobile devices;
- *Managed NAND solutions*: a huge number of NAND flash memories are available, each one with different physical characteristics even considering new process technology (e.g., number of pages, block size, reliability). Therefore, the host software need to be continually updated to efficiently exploit the memory. It is evident that in many cases this approach is not feasible. On the contrary, a more efficient approach requires to combine, in a single chip, a NAND flash memory and a powerful memory controller. Memory controller is in charge to handle all wear leveling routines, ECC and FTL, allowing the host to perform only read/write operations to the device. In Fig. 2.16 a comparison between the raw and managed solutions is shown.

Since the memory controller is becoming nowadays ever more powerful, the managed NAND solutions are basically preferred in many contexts. Therefore, it is worth here an insightful description of the legacy memory controller architecture actually used by the main companies and then introducing some novel features the memory controllers will experience in the next future to meet the demanding requirements in terms of reconfigurability that applications will require.

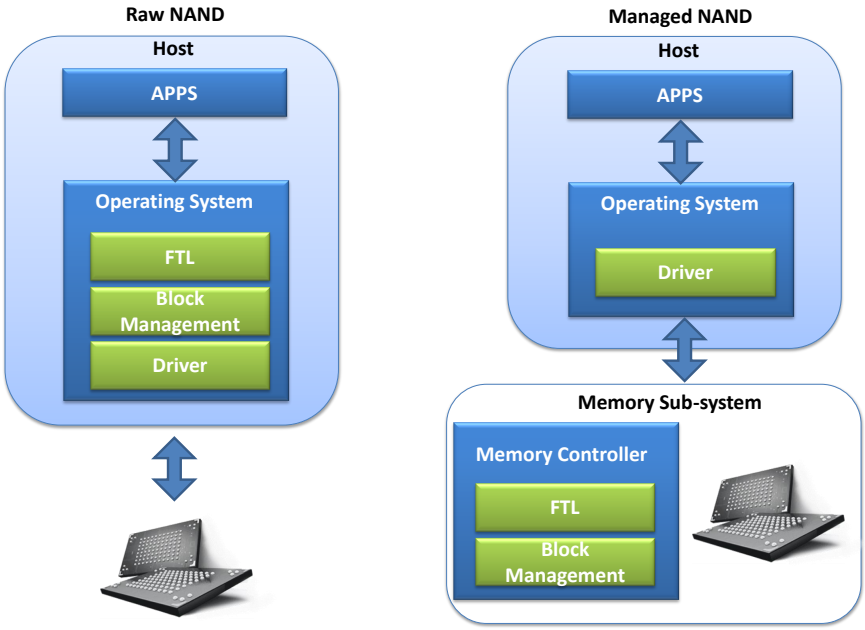


Figure 2.16: Raw NAND vs Managed NAND

2.6.8 NAND flash controller

The flash memory controller is a key component for determining the characteristic and the performance of a whole NVM sub-system.

The typical architecture of a state-of-the-art NAND flash controller in Fig. 2.17 . It is taken from the embedded computing domain [6]. The controller features an interface to the MultiProcessor System-on-Chip (MPSoC) via a state-of-the-art communication protocol, like for instance the AMBA AHB. As the NAND flash memory device is typically slower than the data bus, all data transfers are processed through an internal or external buffer. Typically, the size of the buffer is equal to the size of one or few NAND flash pages (e.g., 1, 2, 4 or 8 KB). Another typical feature consists of booting on code located in the NAND flash by automatically fetching the code. The boot configuration is, in this case, defined by static input pins.

The capability for error detection and correction is nowadays a must for NVM controllers. BCH, Reed-Solomon or Hamming codes are usually adopted. Controller providers typically deliver a specific error correcting code as a synthesis time parameter. The customer can then configure the code depending on the application requirements to better

trade-off between performance and area.

Controllers usually provide a limited set of boot-time parameters (e.g., enable/disable ECC) that can be configured through a lower-performance configuration bus driven by the MPSoC. An AMBA APB port is a typical example thereof. Through the same port, the MPSoC can also inquiry the internal state of the controller.

Finally, the NAND flash interface is Open NAND Flash Interface (ONFI) [9] compliant. It features a degree of synthesis-time parametrization in terms of address bytes, data width, chip select signals and page transfer size.

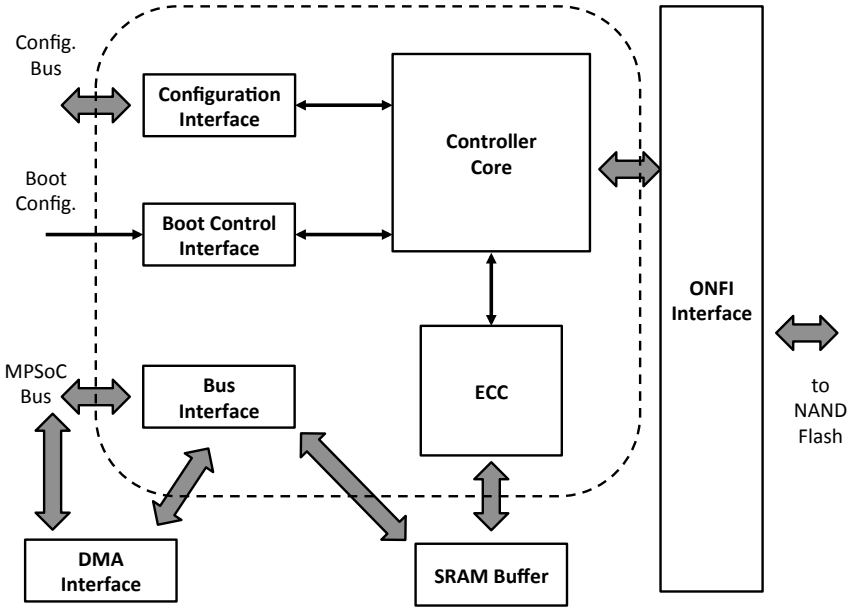


Figure 2.17: State of the art memory controller architecture for a NAND flash device

Future NAND flash controllers will feature both incremental as well as revolutionary characteristics. For instance, the MPSoC interface will change as an effect of the evolution of state-of-the-art shared and multi-layer busses into on-chip interconnection networks (NoC). In this case, the controller will be connected to the network via standard socket interfaces like the Advanced eXtensible Interface (AXI) or the Open Core Protocol (OCP) (Fig. 2.18). Specific packets heading to memory mapped configuration registers will take care of programming controller features. To the limit, packets carrying addresses, data and commands for the flash memory device will be able to set also configuration parameters on a memory transaction basis. At the same time, NVM controllers

are undergoing more radical modifications in an attempt to provide an enhanced degree of adaptation to the workload under execution or of self-adaptation to current operating conditions:

- Fixing critical design decisions at design time may lead to embedded processors that can hardly react to an often non-predictive behavior of today's complex applications. This does not only result in reduced efficiency. It also leads to unsatisfactory behavior when it comes to design criteria like performance and power. Memory transactions are pivotal for the performance of the whole embedded system, and NVM ones also for its reliability. Therefore, run-time adaptive computing (like in [66]) will have to be complemented by the availability of differentiated memory access modes with a similar degree of run-time adaptivity. As a consequence, users should be able to configure the memory access mode at run-time to meet the specific requirements of the data set they are going to process.

- Partial reconfiguration of the controller could also be achieved in a self-adaptive way. It is in fact possible to envision an integrated reliability manager collecting and elaborating results of a test unit and feedback from the ECC sub-system, to set the proper correction capability to pages. In-situ adaptation to actual operating conditions is another clear trend for future MPSoC design [93].

The resulting memory controller architecture is conceptually shown in Fig. 2.18.

In this work thesis, we prove the unprecedented trade-offs and operating points stemming from the concurrent configuration of an ECC sub-system able to provide programmable correction capability and a flash physical management sub-system providing different programming algorithms for the target NAND flash device. Overall, the tuning knobs of the controller will enable to affect directly parameters such as UBER, Read Throughput (RT) and Write Throughput (WT), and indirectly power. This results either into differentiated access modes exposed to the user or into a set of self-adaptive operating points. The flexibility to tune operating characteristics of the controller in a multidimensional optimization space is out-of-reach of current NVM controllers, where a limited set of parameters can be fixed at synthesis time or, in the best case, at boot-time[5].

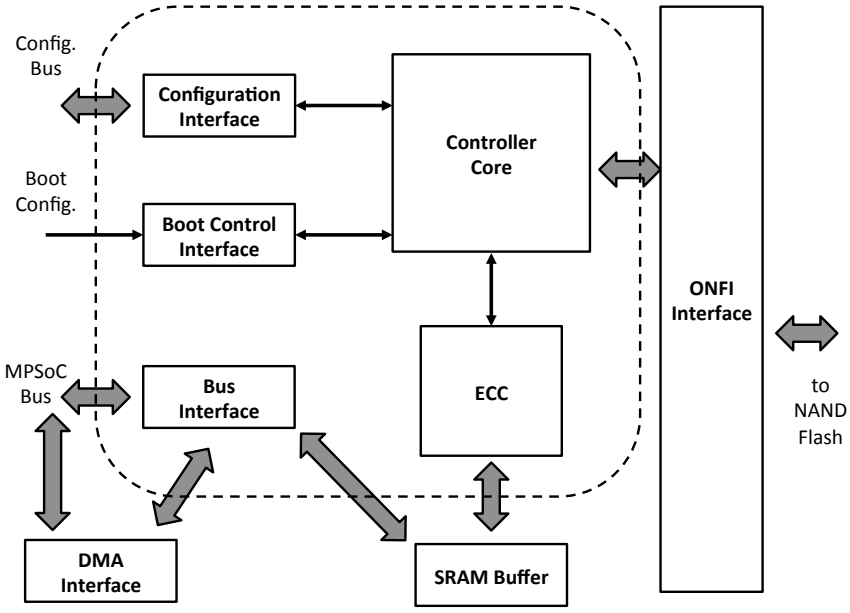


Figure 2.18: Future memory controller architecture with enhanced reconfiguration capabilities

2.6.9 Solid state drive

In the last decade, the outstanding results in terms of performance and power consumption have significantly legitimate SSD as the best storage solution. Moreover, SSDs price trend is decreasing, enabling even the market sector related to notebooks to move toward SDD-based drivers.

SSDs are able to outperform older HD in terms of reliability and latency, as they no contains mechanical parts and are basically composed of few major components: NAND flash memories, memory controller, host interface, DRAM, PCB and passives. A SSD block diagram is shown in the Fig. 2.19.

In order to maximize the I/O bandwidth, SSD architecture is designed to access more flash memories concurrently. The concurrency is achieved by resorting to different channels, each one addressing a different set of flash memories. Of course, the number of channels employed and how many flash memories have to be linked to the channel itself are strategic design decisions that can drastically impact the overall SSD performances. Interested readers may refer to [100] for a comprehensive discussion about all the main topics related to SSD.

SUMMARY

This chapter introduced in a bottom-up approach the main concepts related to the NAND flash memories and the common applications in which they are employed. First of all, the NAND memory cell has been presented. Then the NAND memory architecture has been introduced focusing on both physical and logical layers. NAND flash technology experiences good performances in terms of access time and power consumption making it the primary solution to embedded systems. However, flash memories rapidly wear-out, so sophisticated strategies need to be employed to prolong device life time. These strategies can be implemented either at software or at hardware level, depending on the target application. In this scenario a key component to fully exploit the potentiality of NAND flash memory is the memory controller. It is in charge to handle the wear-out of the memory, hiding, at the same time, to the upper layers of the system (e.g., OS, applications) details related to NAND flash memory access. Such an approach allows to simplify the interaction between the system and the flash memory, as applications have just to write and read data. The memory controller is commonly used in modern NAND-based SSDs, which are persistent storage solutions replacing older HDs.

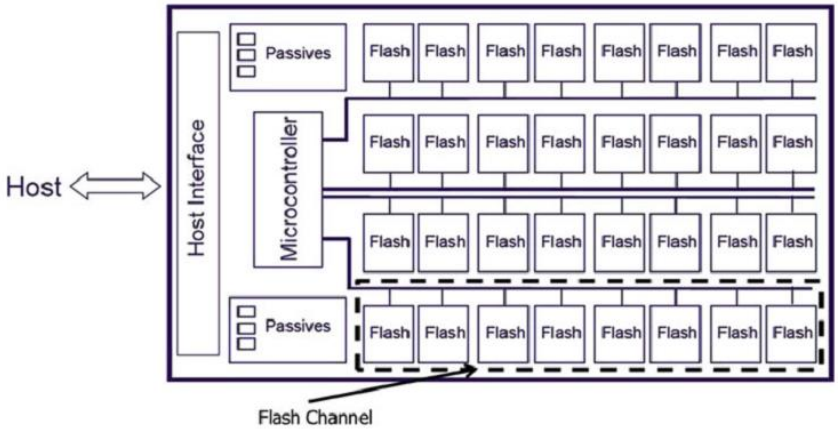


Figure 2.19: SSD architecture [100]

CHAPTER 3

NAND FLASH MEMORY RELIABILITY ISSUES

Contents of this chapter

NAND flash memories are currently expected to be more and more powerful in terms of performances, providing at the same time improved storage capabilities. These motivations push designers to look for new solutions in both architectural and device level. Nevertheless, the main concern along the innovation steep path is the reliability, since the delivered product must ensure to work properly not only in its early stage but even its overall life cycle.

In other words, NAND flash memories are expected to experience an improved endurance, together with the capability to retain unaltered stored data for years.

In other words, the capability to keep unaltered stored information for years together with the capability of supporting a minimum number of programming operations must be assured.

The reliability problem in Flash memory ICs poses more difficult and interesting challenges than do most of other IC devices.

In Flash memory IC the storage functionality is achieved employing the floating gate

technology (see chapter 2) this makes the system very different from common analog or digital ICs.

The main characteristic that distinguishes common ICs and Flash memory is the need of *high electric fields* used to program and erase the cells.

High electric fields are needed to charge and discharge the floating gate and are directly applied to the thin oxide that surrounds the FG. Therefore, charge losses and fluctuations problems are unavoidable in the flash memory technology. For the sake of clarity, the effects related to the the cycling induced degradation are shortly discussed in section 3.1.

As a consequence of the floating gate technology, we have:

- Strict relation between the reliability of a cell and the number of program/erase cycles that have been performed on it (referred to as "aging" of the cell);
- Charge losses and pseudorandom fluctuation effects due to oxide degradation and holes/electrons trapping/detrapping phenomena;
- Specific Disturbances that can be activated mainly during program operation in the cell matrix, even if erase and read operations have to be considered, as well;
- Cell to cell Interferences.

In the following, we will distinguish between *transient faults* 3.2 and *permanent faults* 3.3. Transient faults results in not well defined excitation mechanism and bring to *random errors*. Many of them are mainly related to charge losses and fluctuations phenomena. In Permanent faults the excitation mechanism is well defined and the errors, named *hard errors*, are generated whenever proper excitation occur. These faults are typically related to physical defects of the structure and are usually modeled as *disturbances*, *circuit level faults*, and *interferences*.

It will also pointed out as these effects increase dramatically their impact when Multi Level Cells (MLC) are considered.

Section 3.1 proposes a detailed description of the mechanism influencing the aging of the cell. Section3.2 is a complete analysis of the possible transient faults of NAND flash-memory. Finally, Section 3.3 addresses the modeling of permanent faults.

3.1 Cycling induced degradation

As it will be widely discussed in this chapter, the "aging" strongly impacts several reliability issues affecting NAND Flash memory technology. Of course, the aging is strictly related to the number of performed program and erase operations during the device life cycle.

Programming a cell consists in injecting some electrons in the floating gate. The program and erase operations in NAND Flash memories are usually performed through the Fowler-Nordheim (FN) tunneling mechanism.

In general, when using the FN tunneling effect, the charges are moved inward and outward the FG applying an extremely high voltage between the floating gate and the substrate (positive or negative).

Both the generated relatively high tunneling current and the related high electric field are causes of oxide degradation. Trapping of electrons is the main factor that causes degradation of the oxide, but other specific and particular phenomenons can take place as well. For instance, hot carrier generation can occur both in the oxide and in the anode from the impact ionization that occurs when energetic electrons enter the anode.

This results in a limited number of cycles per cell, beyond which the reliability of the cell is not guaranteed by manufacturers.

The main effect of the cycling induced degradation is to reduce the gap between the program and erase states thresholds window, making the low and high threshold states indistinguishable. A general way to improve the reliability of both program and erase operations is to apply *program-verify* and *erase-verify* algorithms (i.e., ISPP), respectively [23] [98]. This basically means that the operation is repeated until it succeeds. If, after a program or erase pulse, correct margins are not achieved, additional program or erase pulses are applied until a sufficient margin is obtained.

As a consequence, the degradation of the cell is also reflected in the increased number of pulses needed to perform the operations (erase/program) [23].

These defects can contribute to the degradation of the trans-conductance of the cell and to some other defects related to data retention [23].

3.2 Charge Losses and Fluctuations: *Transient Faults*

In this section we briefly present the major causes of non-deterministic errors in Flash memories. These are due to *Charge Losses and fluctuations*, and can be all modeled as *Transient Faults*.

The discussed transient faults are:

- erratic erase 3.2.1;
- charge trapping and detrapping 3.2.2;
- anomalous Stress Induced Leakage Current 3.2.3;
- random telegraph noise 3.2.4;
- few-electrons issue 3.2.5.

All these transient faults are typically tackled by error correcting techniques. In the following each phenomenon is briefly explained.

3.2.1 Erratic erase

With the aging of the cells (cycling) and in particular with tunneling, some holes can be trapped in the oxide and, in some unlucky cases, a small number of holes can be trapped in optimal position to enhance tunneling current.

This unwanted and rare situation brings the flash cell in a particular condition in which a phenomenon named *Erratic erase* can occur. The erase procedure is complicated by several concerns. First, erasing (removal of electrons) shifts the threshold voltage negatively from a positive value toward a value nearer to zero. If continued too far, the threshold voltage can become negative. This is referred to as *over-erase*. An erratic cell, thus, oscillates between the state of over-erased and the normal erased state, in an unstable way. Therefore, the threshold of the cell fluctuates randomly. The position of the trapped holes in the oxide is fundamental to activate this particular and non deterministic phenomenon and this makes it very rare. The cells that are affected by this problem are typically very few, about one per page.

A fundamental aspect of this problem is that an erratic cell typically keeps on being erratic also with cycling, thus even if you program and erase it successfully many times.

Unfortunately, the holes that are involved in this mechanisms are not those generated by band to band tunneling and thus they cannot be avoided using uniform channel erase techniques. These holes are likely those generated either by the high electric field in the oxide or by the impact ionization of the tunneling electrons in the oxide [23].

3.2.2 Charge trapping and detrapping

The charge trapping related phenomenons is quite critical [88] [121] [96]. The motivations are based on the continuous memory scaling down and on the need to apply high voltages on a very thin oxide to perform program and erase operations. The application of high fields inevitably causes defect generation and charge trapping, with significant reliability impact.

The trapped charge accumulates cycle after cycle, and eventually results in a shift of the V_{th} levels for the programmed and erased states. In general, both program and erase processes become slower with cycling, as a result of negative trapped charges in the tunnel oxide that decrease the effective field for tunneling across the floating gate. NAND cells most typically display an increased V_{th} for both programmed and erased states due to interface traps and bulk negative charge.

This degradation is generally compensated by careful program/verify algorithms, where the V_{th} is increased step by step, each step being verified by a read pulse and compared with the target value.

The obtained V_{th} can, in reality, depend also on the trapped charges in the oxide and not only by the charges that are present in the floating gate.

In the programmed state (high V_{th}), the negative charge trapped in the dielectric tend to tunnel out, causing a V_{th} decrease with time.

The detrapped charge depends logarithmically on time, as a result of the discharging current having a $1/t$ dependence due to the dispersion of trapping depths in the dielectric. Detrapping depends on temperature according to the Arrhenius law:

$$t_{det} = t_0 e^{(E_A/kT)} \tag{3.1}$$

where t_{det} is the time needed for detrapping the same fraction of charge, t_0 a constant, E_A the activation energy, k the Boltzmann constant and T the bake temperature[69].

Since detrapping is the result of many individual trapped carriers, detrapping generally occurs for all cells in the array. Due to the generalized impact of detrapping on

the array, ECCs are not effective in its correction. Rather, detrapping effects can be minimized by a careful wear leveling, taking advantage of the large available size of NAND arrays to distribute cycling-induced damage over a large number of cells [69].

3.2.3 **Anomalous** SILC

The Leakage of charges from the floating gate to either the bulk or the gate is an anomalous phenomenon that can take place in particular conditions, only [133]. The extensive program/erase cycling generally causes generation of oxide defects and traps, which can then act as stepping stones for carrier tunneling: for instance, electrons can tunnel from the inverted substrate to the floating gate in the case of positive gate stress [70]. This is generally referred to as *Trap-assisted tunneling* and can occur only when two or more defects sitting very close one to the other in the tunnel dielectric cooperate in the trap-assisted tunneling mechanism where two cooperating traps assist electrons tunneling from the cathode to the anode [69] [72] [83].

Typically only few cells in the array are in this particular and rare situation and the activation mechanism can induce a gate stress or even nothing, depending on the oxide defects. Anyway, since anomalous SILC generally affects a small minority of cells in the array, it can be corrected by an ECC. Thus, a careful evaluation of SILC effects vs program/erase cycles is mandatory for an accurate selection of the most suitable type of ECC [69] [71].

The anomalous SILC problem can both charge (gate stress) and discharge (steady condition) the FG, depending on the positions and the number of the oxide defects. The oxide traps and defects that generate this anomalous SILC get worsen during the life time of the device.

3.2.4 Random telegraph noise

Traps in the tunnel oxide not only impact reliability by the charging and discharging phenomenons, but can also cause drain current fluctuations which affect the readout operation. Fig. 3.1 shows the measured drain current in a Flash memory under read conditions.

The on-state current is affected by *Random telegraph noise* (RTN) [17] [78] [117], with statistically-distributed times τ_c and τ_e for the high and low current states. These times can be explained as the time for capturing a channel electron in an oxide trap and for re-

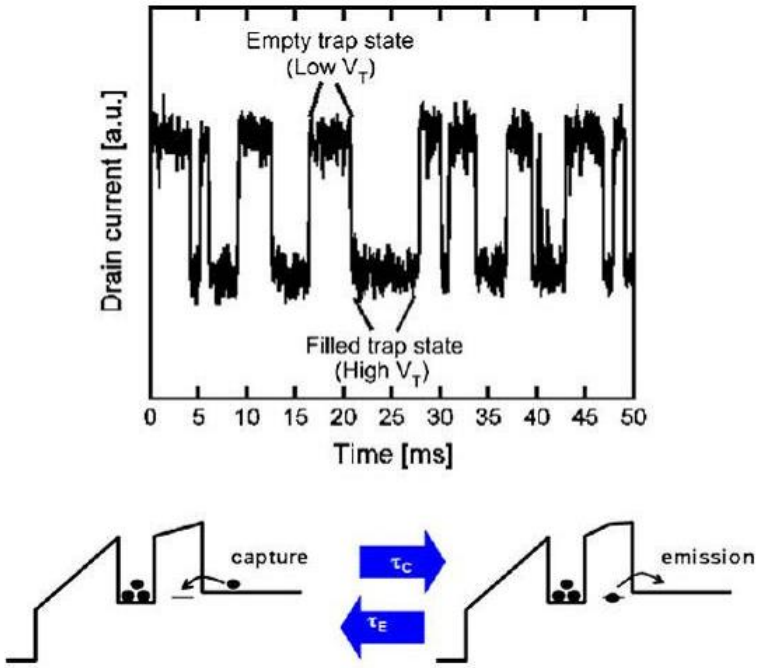


Figure 3.1: Drain current fluctuations [69]

leasing it back to the channel, respectively, as shown in Fig. 3.1. The local modification of the vertical field modulates the channel charge, thus causing a change in both the measured current and V_{th}. The unstable V_{th} is detrimental for the program-verify algorithm, thus for accurate placing of the programmed V_{th}. In fact, one cell V_{th} can unpredictably jump from the high level, during the verification, to a lower level, or viceversa. This can cause the cell to exit the target V_{th} window just after the verification, resulting in a fake success of the verify operation. V_{th} fluctuation is particularly severe for Flash memories, as compared to MOSFETs for logic applications, because of the relatively thick oxide stack[69].

3.2.5 Few-electrons issue

Few electron phenomenon [114] [115] are becoming an increasing reliability concern. Charge discretization emphasizes the effects related to the random nature of carrier tunneling. For instance, data retention time cannot be purposely engineered in a Flash memory with as few as 10 electrons in the programmed state, because of the extreme

broadening of the Poisson distribution of tunneling times, resulting in a significant probability for fast data loss in a high-density array. Even in the most optimistic perspective, these trap-unrelated reliability features will provide hard obstacles against Flash scaling beyond the 20 nm node [69].

3.3 Permanent faults

In this section we shortly present the deterministic error causes in Flash memories that are modeled as *permanent faults*. They are typically due to disturbances or physical defects.

The discussed permanent faults are:

- *memory disturbances* 3.3.1;
- *circuit level faults* 3.3.2.

3.3.1 Memory disturbances

Disturbances are faulty behaviors resulting from the FG technology [68]. As a consequence, they do belong to flash memories, but not to the other memories. The most significant ones include [68]:

- *program disturbance faults*;
- *Read Disturbance* (RD) faults;
- *Over-Erase Disturbance* (OED);
- *Over-Program Disturbance* (OPD) faults.

All these disturbances are able to modify the original value stored inside a cell.

3.3.1.1 Program disturbances

The state of an erased cell is logically "1". Programming a single NAND flash-memory cell consists in logically writing a "0". Erasing a cell means logically writing a "1" again.

Unlike RAMs, which are *random* access memories, NAND flash memories are referred as *sequential* access memory. This means that, in order to access (to read and write) a cell, are need to "pass" through the others, stressing them.

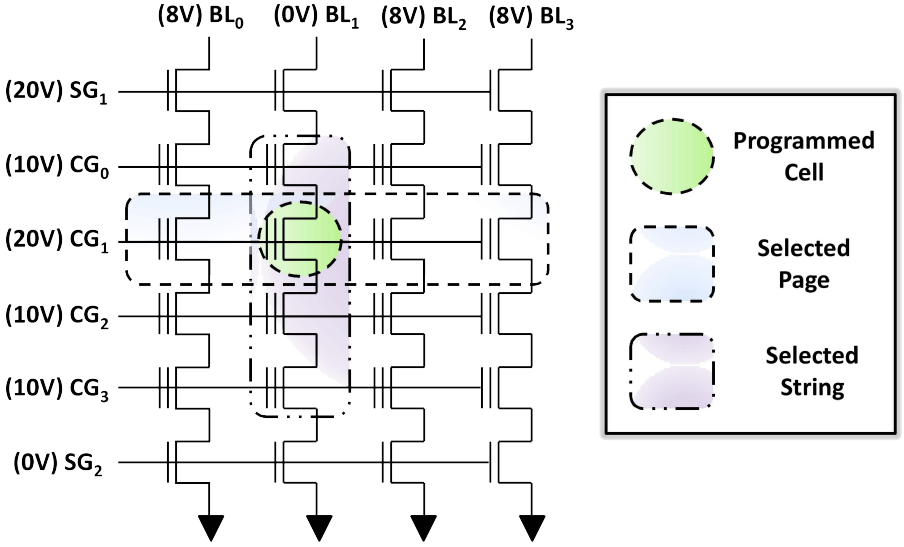


Figure 3.2: NAND Flash memories Program Disturbances

Fig. 3.2 shows how programming a single cell C11.

When a single NAND flash memory cell is being programmed (i.e., $1 \rightarrow 0$ transition), all the cells in the row (i.e., Word-Line or WL) are subject to a high control gate voltage and all the cells in the same column (i.e., Bit-Line or BL) are biased to be in the pass-transistor state. This situation can produce unintentional transitions in any of the cells in the Word-Line (WL) and/or in the Bit-Line (BL) of the one being programmed.

WL_1 is subject to high-voltage (e.g., 20 V). All the cells of BL_1 become pass-transistors. In this case, program disturbance faults can occur in: (i) a cell sharing the common WL_1; (ii) a cell sharing the common BL_1. Fig. 3.2 refers to them as *Selected Page* and *Selected String*, respectively. Program disturbances can occur only within the block the page under program belongs to [48].

According to its initial content, the faulty cell can be programmed or erased. Literature commonly refers to the transition $1 \rightarrow 0$ (i.e., unintentional programming) as:

- *Word-line Program Disturbance* (WPD) [43, 76, 109, 110, 140] or *DC-Programming* (DC-P) [111]: the selected cell under program causes an unselected unprogrammed cell on the same WL to be programmed; in Fig. 3.2, each unprogrammed cell of the *Selected Page* can be unintentionally programmed;

- *Bit-line Program Disturbance* (BPD) [43, 76, 109, 110, 140]: the selected cell under

program causes an unselected unprogrammed cell on the same BL to be programmed; in Fig. 3.2, each unprogrammed cell of the *Selected String* can be unintentionally programmed;

Literature commonly refers to the transition $0 \rightarrow 1$ (i.e., unintentional erasure) as:

- *Word-line Erase Disturbance* (WED) [43, 76, 109, 110, 140] or *DC-Erase* (DC-E) [111]: the selected cell under program causes an unselected programmed cell on the same WL to be erased; in Fig. 3.2, each unprogrammed cell of the *Selected Page* can be unintentionally erased;

- *Bit-line Erase Disturbance* (BED) [43, 76, 109, 110, 140] or *Drain Disturbance* (DD) [111]: the selected cell under program causes an unselected unprogrammed cell on the same BL to be erased; in Fig. 3.2, each unprogrammed cell of the *Selected String* can be unintentionally erased;

Fig. 3.3 provides a more generic example of program disturbances for NAND flash.

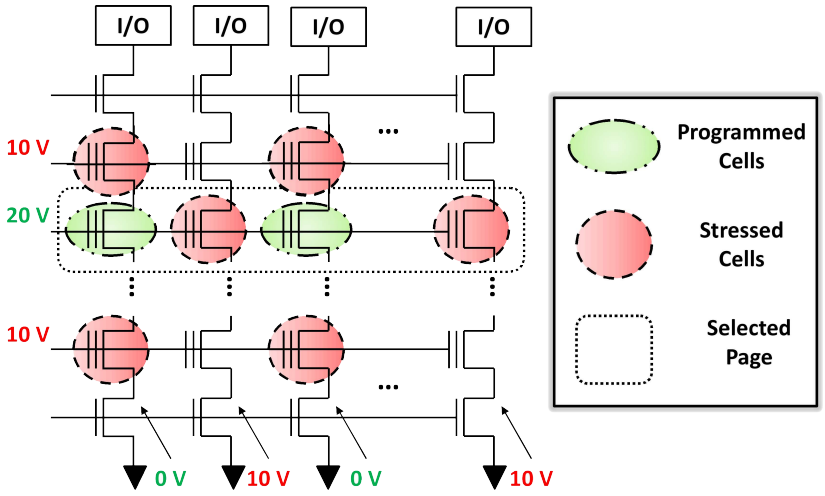


Figure 3.3: Program Disturbances in NAND Flash

Fig. 3.3 shows the programming of two cells (i.e., *Programmed Cells*) which may disturb the other cells on the same WL or BL of the programmed ones (i.e., *Stressed Cells*).

However, as the name suggests, this phenomenon is only a disturbance. As such, it does not damage cells but simply interferes with their content [48].

Finally, some NAND devices are allowing the so called *Partial Page Programming* (PPP), i.e., the ability of programming part of a page, only. This ability enables higher flexibility, but increases the chance of program disturbances.

Several mechanisms are typically employed to mitigate effects caused by program disturbances. These mechanisms are shortly described in the next paragraph.

Reducing program disturbance In order to leverage the program disturbance phenomenon, it is advisable to:

- program in a sequential way the pages belonging to a same block (e.g., from 0 to 63 for SLC, from 0 to 127 of MLC);
- limit PPP as much as possible;
- program in "one-shot" MLC-based pages;
- adopt ECC strategies to recover from disturbances; e.g., 512B-ECC1[1] per SLC page or at least 512B-ECC16 per MLC page;

3.3.1.2 Read disturbances

Being a sequential access memory, NAND flash are stressing (many) unselected pages for reading just one page. Fig. 3.4 shows the read operation of a NAND flash page.

The selected page is biased with a defined control gate voltage (e.g., 0 V), whereas all the other unselected pages are turned into pass-transistors with a higher control gate voltage (e.g., 5 V). A sufficient number of read operations[2] performed on the same page is able to produce unintentional transitions in the page being read. Furthermore, the other unselected pages may be disturbed as well.

If after consecutive reads the selected page may changes its state, then a Read Disturbance (RD) occurred. Read disturbances can occur: (i) only within the block to which the page being read belongs to; (ii) only in the unselected pages [48].

The RDs for NAND flash memories are well known in literature [76, 112, 140] as:

- RDA(E): the selected programmed cell is read and its content is erased;
- RDA(P): the selected erased cell is read and is programmed;

[1] it means 1-bit correctable (i.e., 1 error tolerated) each 512Bytes
[2] this figure is strictly linked with technology

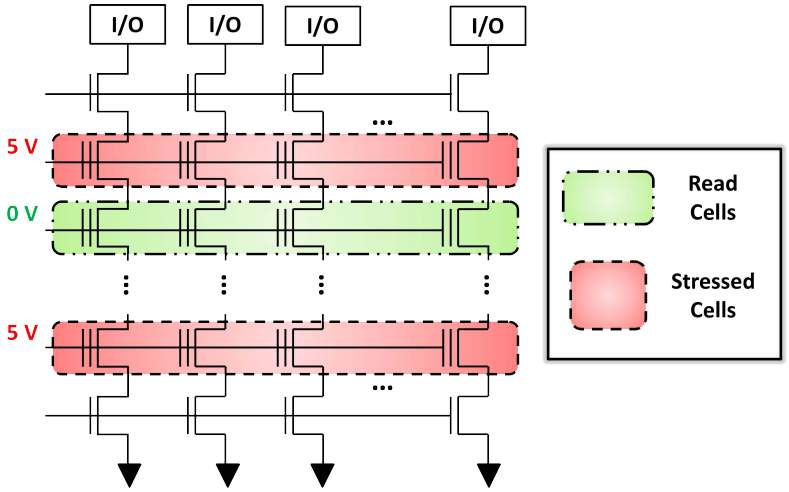


Figure 3.4: Read Disturbance in NAND Flash

- RDU(E): the selected cell is read and another unselected programmed cell is erased;
- RDU(P): the selected cell is read and another unselected erased cell is programmed;

However, as the name suggests, this phenomenon is only a disturbance. As such, it does not damage cells but simply interferes with their content [48].

Reducing read disturbance To leverage the read disturbance phenomenon, it is advisable to [48]:

- adopt a RDs counter for each block;
- as a "rule of thumb", limit to 10^6 (SLC) and 10^5(MLC) the maximum #reads of each block;
- erasing a block "reset" its RD count; when either the ECC threshold or the "rule of thumb" is exceeded, move valid data to a free block and erase the old one;
- adopt ECC strategies to recover from disturbances; e.g., 512B-ECC1[3] per SLC page or at least 512B-ECC16 per MLC page;

[3] it means 1-bit correctable (i.e., 1 error tolerated) each 512Bytes

3.3.1.3 Over-Erase Disturbance (OED)

When a flash memory block is erased, all the electrons trapped in the FGs of the cells of a block are simultaneously removed. However, it is important to remark that:

"...not all the cells have equal yield or identical physical conditions..."[48]

Therefore, there may be some cells that are already erased before (i.e., "faster" than) the others. These cells will have a net positive charge in the FG, resulting in a very low threshold [68]. This phenomenon is referred as Over-Erase Disturbance (OED). It is difficult to program cells affected by OED, because they need more program cycles than usual. The result is that automatic *Program&Verify* operations are slowed-down.

3.3.1.4 Over-Program Disturbance (OPD)

At the opposite, when a page is programmed, there may be some cells that are programmed before (i.e., "faster" than) the others. These cells will have excessive negative charge in the FG, resulting in a very high threshold [68]. This phenomenon is referred as *Over-Program Disturbance* (OPD). Erasing cells affected by OPD, requires more erase cycles than usual.

Let us point out another important aspect. A cell affected by OPD can prevent the correct reading of other cells. As an example, let us consider Fig. 3.4 and assume that there is an over-programmed cell (red ones in Fig. 3.4) on a particular BL. Note that all the cells on the same BL are connected in series and that the over-programmed cell causes an open defect on the bit-line (i.e., absence of current detected). Therefore, the result of each read operation on a cell on the same BL will produce always a logic zero even if the expected value is a logical one.

3.3.2 Circuit level faults

In this subsection Flash memory faults (physical defects) that can be modeled at the circuit level as resistors and capacitors are presented. There are three main contributions to all the possible defects of NAND flash memory [67, 74, 108]:

- intra-cell faults;
- inter-cells faults;

- cell to cell interferences.

3.3.2.1 Intra-cell faults

Fig. 3.5 shows the shorts within a Floating Gate transistor cell, i.e., Control Gate (CG), Floating Gate (FG) , Drain (D), Source (S) and Bulk (B). In particular, the possible shorts are between CG-FG, FG-D, FG-S, FG-B, CG-D, CG-S and D-S.

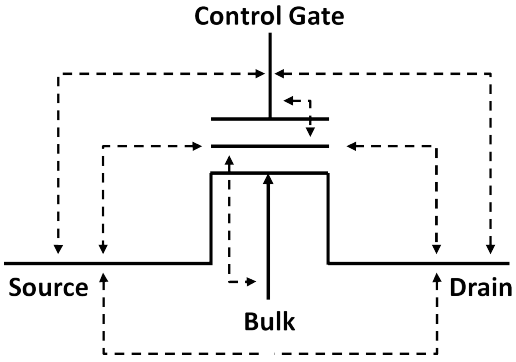


Figure 3.5: NAND Flash memory Intra-cell Faults

It can be shown that all the bridging faults of Fig. 3.5 are equivalent to a Stuck-At Fault behavior. Therefore, they are reported in Table 3.2 as SAF.

3.3.2.2 Inter-cells faults

Fig. 3.6 shows the possible faults between different cells of a NAND flash memory.

Resistive shorts between adjacent cells in the same column/row Fig. 3.6.(a) shows the shorts between adjacent cells in the same column. Also the Select Gate (SG) is considered. In particular, the following shorts can occur FG-FG, FG-SG1, FG-SG2, CG-CG, CG-SG1, CG-SG2.

Fig. 3.6.(b) shows the shorts between adjacent cells in a same row. In particular, there can be shorts between FG-FG, D-D, S-S and BL-BL [67].

We will refer the aforementioned shorts faults of Fig. 3.6 (a) and Fig. 3.6 (b) as Coupling Fault between Adjacent Cells (CFAC). In Table 3.2 the CFAC are split in CFAC$_{row}$ and CFAC$_{col}$.

Resistive shorts in the selected transistors Fig. 3.6 (c) shows the resistive short faults in a single cell. The following shorts can occur CG-FG, D-FG, S-FG, CG-FG, D-CG, D-S. It can

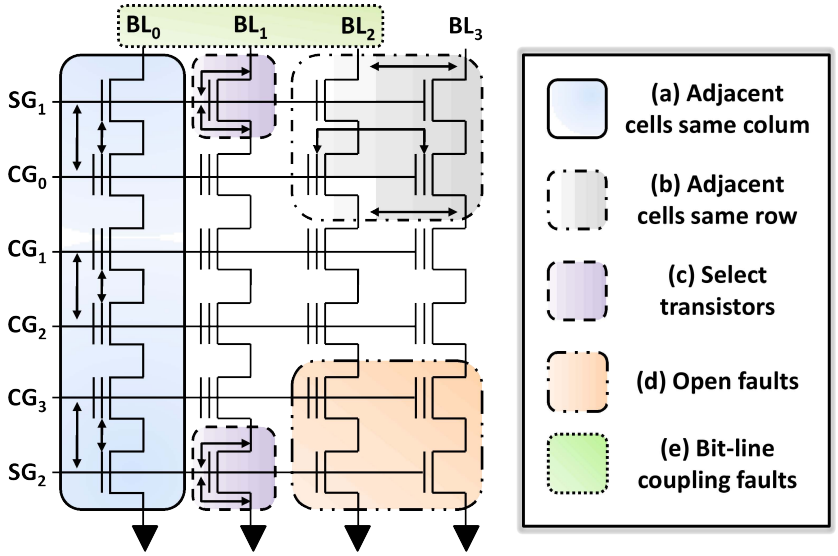


Figure 3.6: NAND Flash memory Inter-cells Faults

be shown that the faults of Fig. 3.6.(c) are equivalent to a Stuck-At Fault (SAF) behavior. Therefore, they are reported in Table 3.2 as SAF.

Open faults Fig. 3.6.(d) shows the open faults. They are shorts between SG1-D, SG1-S, SG2-D, SG2-CG. It can be shown that the faults of Fig. 3.6.(d) are equivalent to a Stuck-At Fault (SAF) behavior. Therefore, they are reported in Table 3.2 as SAF.

Bit-Line coupling faults The parasitic capacitances connecting the BLs can produce errors; Fig. 3.6.(e) shows how two hypothetical parasitic capacitors connecting BL1 with BL2 and BL2 with BL3 could lead to possible errors when reading cells belonging to the centering column BL2. Table 3.2 refers to this phenomenon as BL Coupling (BC) among three adjacent BLs.

3.3.2.3 Cell to cell interferences

In this section we present a particular type of cell to cell interference that occurs mainly in high density memory devices, that result in *permanent faults*. This kind of coupling effects are strictly related to the physical distance between adjacent cells.

Coupling effects are considered critical in MLC flash devices because they directly act on cell's thresholds, and since the threshold margins in MLC flash memory are reduced,

this can easily lead to a cell's state modification.

As cell transistor sizes are scaled down below 50 nm, a selected cell transistor gets nearer to neighboring cell transistors, so that they influence each other directly and indirectly. The indirect effect is due to parasitic capacitance-coupling effect, while the direct effect indicates the intrinsic V_{th} shift caused by a neighboring cell transistor.

In the following, we briefly describe the two main types of cell to cell interferences in flash memories.

Capacitive coupling The Capacitive Coupling (CC) considers the parasitic capacitors connected to the floating gate of cell. These capacitors connect the floating gate of a cell with:

1. the floating gates of the adjacent cells in both X and Y directions;
2. the control gates of the adjacent cells in both X and Y directions. Parasitic capacitors on the diagonal are neglected. This brings to a total of eight parasitic capacitors connected to the floating gate of cell.

Therefore, the floating-gate voltage is determined not only by the corresponding control-gate voltage, but also by the voltages of the surrounding floating gates and control gates.

Programming a cell affects the floating gate voltages, and thus the thresholds, of all the adjacent cells. The entity of the threshold shift is typically not sufficient to affect SLC technology, but it becomes a serious issue in MLC, where the different threshold distribution widens and the reliability of the memory is compromised [75].

The CC faults of [74] are functionally identical to CFAC_{row}.

Direct field effects As the cell size reduces to below 50 nm, the electric field of the adjacent cell transistor directly influences the shallow-trench isolation corner of a selected cell transistor, provoking a significant cell V_{th} shift. While conventional parasitic capacitance-coupling effect alters only the floating gate voltage, the Direct Coupling or Direct field effects (DC) intrinsically changes the cell V_{th} and provokes an intense V_{th} shift, particularly in word-line direction.

It is important to notice that this voltage shift is not still enough to produce an error in a SLC flash device.

In the sub-50 nm regime, the distance between the channel edge of a cell transistor and the floating gate of a neighboring cell transistor is so close that the floating gate

voltage of the neighboring cell transistor directly influences the channel edge, changing the electric field distribution on the channel edge.

Then, V_{th} shift is produced by the direct field effect of the neighboring cell transistor. Since about 70% of the cell current flows on the channel edge, the V_{th} of the cell transistor is determined mostly by the condition of electric field crowding and by the doping concentration of the channel edge. Therefore, the cell transistor suffers an intense V_{th} shift, particularly in the word line direction, where the floating gate faces the whole surface of the channel edge.

The DC faults of [108] are functionally identical to $CFAC_{col}$.

3.3.3 A comprehensive view about persistent faults

Table 3.1 and 3.2 sum up a comprehensive set of fault models for NAND flash memory disturbances and for NAND flash circuit level faults, respectively. In particular, activation mechanisms (Faults excitations) and the resulting errors for both memory disturbances and circuit level faults are shown. For sake of generalization, we here not perform any simplification or reduction based on specific technology information. Therefore, the presented fault models are technology independent.

Disturbance	**Initial state of faulty cell**	**Fault Excitation**	**Resulting Error**
WPD	C_{ix}='1'	Program any C_{ij} with $j \neq x$	C_{ix}='0'
WED	C_{ix}='0'	Program any C_{ij} with $j \neq x$	C_{ix}='1'
BPD	C_{xj}='0'	Program any C_{ij} with $i \neq x$	C_{xj}='1'
BED	C_{xj}='1'	Program any C_{ij} with $i \neq x$	C_{xj}='0'
RDA(P)	C_{ij}='1'	Read C_{ij} N times	C_{ij}='0'
RDA(E)	C_{ij}='0'	Read C_{ij} N times	C_{ij}='1'
RDU(P)	C_{xj}='1'	Read C_{ij} with $i \neq x$	C_{xj}='0'
RDU(E)	C_{xj}='1'	Read C_{ij} with $i \neq x$	C_{xj}='1'
OED	C_{ij}='1'	Program C_{ij}	C_{ij}='1'
OEP	C_{xj}='0'	Erase any C_{ij} with $i \neq x$	C_{ij}='0'

Table 3.1: NAND Flash Memory Disturbances

Table 3.2 sums up the NAND flash circuit level faults.

Fault	Initial state of faulty cell	Fault Excitation	Resulting Error
SAF0	C_{ij}='0'	Erase C_{ij}	C_{ij}='0'
SAF1	C_{ij}='1'	Program C_{ij}	C_{ij}='1'
$CFAC_{row}$	C_{ij}='1', $C_{i+1,j}$='1'	Program C_{ij}	C_{ij}='1', $C_{i+1,j}$='1' or C_{ij}='0', $C_{i+1,j}$='0'
$CFAC_{col}$	C_{ij}='1', $C_{i,j+1}$='1'	Program C_{ij}	C_{ij}='1', $C_{i,j+1}$='1' or C_{ij}='0', $C_{i,j+1}$='0'
BC	C_{ij}='1', $C_{i,j+1}$='1' $C_{i,j+2}$='1'	Program $C_{i,j+1}$	$C_{i,j+1}$='1'

Table 3.2: NAND Flash Memories Circuit Level Faults

SUMMARY

This chapter introduced the main concepts related to reliability issues of the NAND flash memory.

Flash-memory relies on the Floating Gate (FG) technology. However, FG is intrinsically not highly reliable and, combined with the rapid technology scaling down, it may lead to problems in terms of *data retention* and *endurance*.

Firstly, we targeted transient faults. The analyzed faulty behaviors are peculiar of flash-memories. They do not damage the cells but simply interfere with their content. Each operation (i.e., read, program/write and erase) implies a related possible disturbances (i.e., Read Disturbance, Program Disturbance, Over-Program Disturbance and Over-Erase Disturbance). To complete this approach, we modeled the NAND flash in terms of resistor and capacitors. After this step, we were able to set up a comprehensive fault model which is technology independent.

CHAPTER 4

ADAPTABLE FLASH PHYSICAL MANAGEMENT SUB-SYSTEM (PHYSICAL-LEVEL ADAPTIVITY)

Contents of this chapter

The goal of this thesis is to enhance the degree of run-time reconfigurability of an MLC NAND Flash controller through the provision of user-selectable differentiated memory access modes based on an adaptive framework, composed of physical layer adaptivity combined with architectural layer adaptivity.

This Chapter presents the *adaptable flash physical management sub-system* (physical layer adaptivity) that is in charge to manage the high-voltage sub-system of the flash memory device. The choice of the proper high-voltage sub-system highly impacts reliability and performances. It is responsible of generating the voltage waveforms for flash cell read, program and erase operations, and for address decoding. Its operation is reg-

ulated by a microcontroller embedded in the flash device itself. The physical layer considered in this work refers to state of the art 2-MLC memories [107], which store two bits per cell.

In order to accurately fulfill the aforementioned operation, a standard algorithm is usually exploited in MLC NAND Flash memories: the ISPP [98]. In fact, the program algorithm and voltage waveforms that are applied for memory writing are typically defined at fabrication time by the memory vendor and hardwired in memory operation. However, a number of variants does exist to counter the dispersion of programmed cell threshold distributions in nano-scaled flash devices, such as the *double verify* (DV) and *reduced verify* (RV) algorithms [97], [98] which differently impact reliability and performances of the flash memory.

The proposed device layer is designed to be able to switch, on-demand, among the three ISPP versions. For this purpose, an extensive modeling, simulation and implementation framework has been set up for the analog part to capture how different program algorithms impact the RBER, the power consumption, and the write throughput of the memory. It is composed of two distinct modules: (a) the high-voltage subsystem of the memory, including the charge pumps, and the voltage regulators exploited for the generation of the voltages required for the programming algorithm (including the verify stage), and (b) a compact model for NAND Flash memories with array simulation capability.

Section 4.1 presents an overview of the standard ISPP algorithm and the related physical mechanism that allows writing desired logic levels. Section 4.2 focuses on MLC's programming mechanism. Then in Section 4.3 the two variants ISPP-DV and ISPP-RV are described. Section 4.4 and Section 4.5 propose an accurate description of the simulation environment and the result of the characterization of the flash physical management sub-system, respectively. Last but not the least, Section 4.6 proposes different alternatives to physically realize the proposed approach in actual memory controllers.

4.1 The incremental step pulse programming algorithm

The logical value of stored information is associated with the cell's threshold voltage. Therefore, a memory cell is considered programmed when it reaches the desired V_{th} level. In order to control the programmed V_{th} of a NAND flash memory cell, a bit-by-bit program verify algorithm is used [116]. The program operation is split into several program pulse steps, with a V_{th} verification (sensing) operation in between. If the V_{th} of

a cell is detected above a certain program verify level, further programming of this single cell will be stopped by setting it in a program inhibit state (see Fig 4.1).

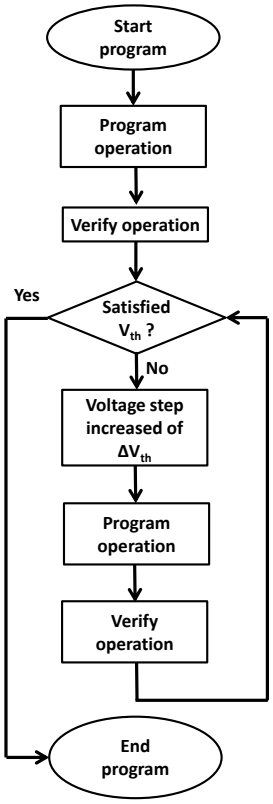


Figure 4.1: Program and verify algorithm

Commonly, the programming pulse is increased by a constant value ΔV_{th} after each program step [129]. Therefore, this programming scheme is called *Incremental Step Pulse Programming with standard verify* (ISPP-SV).

4.2 Programming MLC NAND Memories

In this thesis, MLC technology is targeted as nowadays is the preferred choice when designing embedded systems thanks to its lower cost per chip then the SLC one. Since the adaptable flash physical management sub-system considered in this thesis refers to state of the art 2-MLC memories (i.e., store two bits per cell), it is worth explaining here how the MLC's programming mechanism works and how the ISPP algorithm is commonly applied.

In 2-MLC memories each cell stores one *Least Significant Bit* (LSB) and one *Most Significant Bit* (MSB). 2-MLC memories store two bits per cell by placing four V_{th} levels identified by the statistical distributions *L*0-*L*3 of Fig. 4.2. An erase operation places all cells of a block on the L0 level. L0 is the starting point for each program operation that will place then threshold voltages of the selected cells on levels L1-L3.

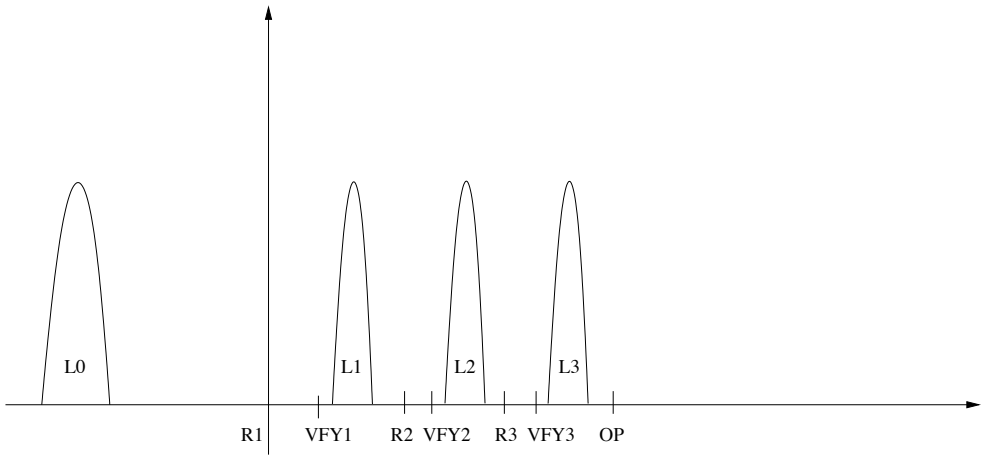


Figure 4.2: Threshold voltage distributions in a MLC NAND flash. Read levels (R1, R2, and R3), Verify levels (VFY1, VFY2, VFY3), and over-programming level (OP) are pointed out

A 2-MLC page can be programmed by applying the ISPP algorithm in *two rounds* or in a *full-sequence* approach. In the following sections both are presented.

4.2.1 Two rounds programming

Figure 4.3 shows an example of how 2 bits are associated with the four read threshold distributions stored in the cell, and how the set of programmed distributions is built starting from the erased state "E" when the ISPP algorithm with two rounds programming is employed [86], [24].

In the first round, the so-called lower-page (associated to LSBs) is programmed. If the bit is "1", the voltage threshold of the cell V_{th} does not change and, therefore, the cell remains in the erased state, E. If the bit is "0", V_{th} is increased until it reaches the D1 value.

V_{th} is modified by means of the ISPP algorithm: a voltage step is applied to the gate of the cell. Afterwards, a verify operation is performed in order to check whether V_{th} has exceeded a predefined voltage value (in this case V_{VFY1}). If the verify operation is

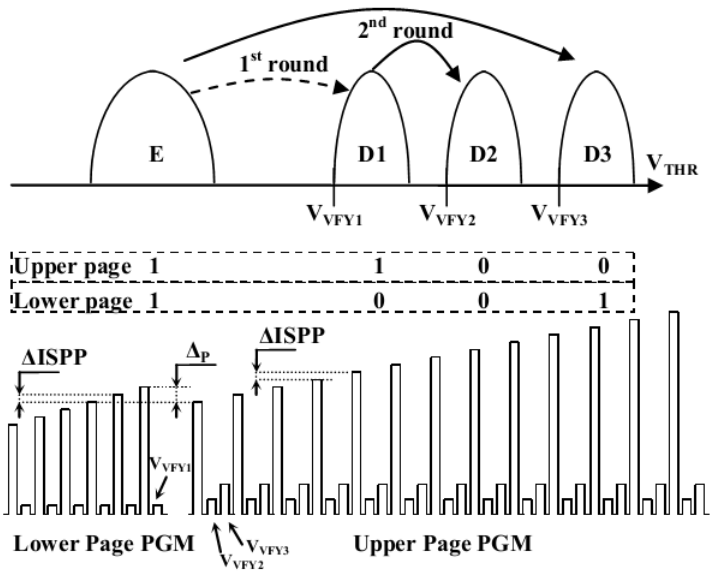


Figure 4.3: Two round program operations [98]

successful, the cell has reached the desired state and it is excluded from the following program pulses. Otherwise another cycle of ISPP is applied to the cell, where the program voltage is incremented by ΔISPP.

In the second round, the upper-page (associated to the MSBs) is programmed. If the bit is "1", V_{th} does not change and, therefore, the cell remains either in the erased state, E, or in the D1 state, depending on the value of the lower-page. When MSB is "0", V_{th} is programmed as follows:

- If, during the first round, the cell remained in E state, then V_{th} is incremented to D3.

- If, during the first round, the cell was programmed to D1, then, in the second round, V_{th} reaches D2.

Even in this case, the program operation uses ISPP, and the verify voltages are V_{VFY2} and V_{VFY3}. Lower-page programming only needs the information related to LSB, while for the upper-page it is necessary to know both the starting distribution (LSB) and the MSB.

4.2.2 Full-sequence programming

ISPP full-sequence programming [31], [98] is shown in Fig. 4.4; in this case, LSB and MSB of the same cell are programmed at the same time, and there is no need to apply the same program voltage twice. After each program pulse, three different verify operations are needed (at V_{VFY1}, V_{VFY2} and V_{VFY3}).

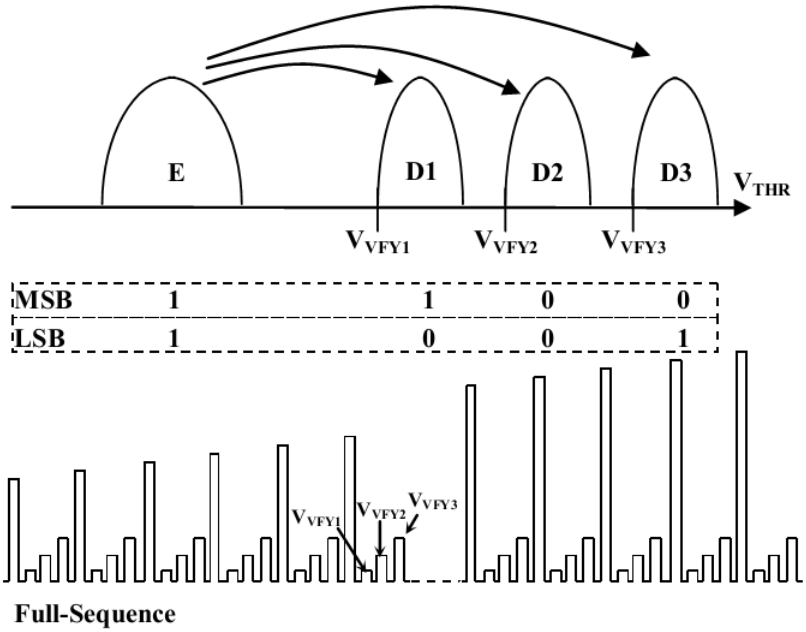


Figure 4.4: Full-sequence program operations [98]

Without loss of generality, we chose to investigate and explore the ISPP full sequence strategy instead of the two-rounds one since it reduces the simulation time and provides faster post-processing of the experimental results.

4.3 Proposed ISPP variants

The main concern related to the ISPP-SV programming algorithm is that cells to be programmed may require a variable number of ISPP steps to reach the desired V_{th}.

Due to the technological variations, V_{th} is not perfectly related to the amplitude of the ISPP pulse. There are "fast" cells that reach the verify level with few program pulses and "slow" cells that require more pulses. Both behaviors represent a threat for the reliability of the program operation. In fact, the threshold voltage distributions of the L1-L3 levels

significantly deviate from an ideal Gaussian shape. They often cross the distribution read levels and cause bit errors.

Different technological approaches for achieving distribution compactness are commonly pursued, although they share the same underlying principle: acting directly on the ISPP pulse characteristics by decreasing the ΔISPP per step or by increasing the total number of pulses per Program operation. However, although these methodologies could effectively increase the accuracy of the ISPP algorithm in terms of threshold voltage placement, a substantial penalty both in power consumption and write throughput is paid.

An alternative solution for increasing ISPP Programming accuracy with minimal burden on the programming time and complexity has been presented in [98] and [97]. This algorithm exploits a Double Verify (DV) approach, where the bitline voltage of the selected cells is modulated in order to partially decrease the ISPP step using a prior Verify level with slightly lower voltage than the original Verify level, hence compacting the final desired threshold voltage distribution.

Another concern of MLC architectures is to decrease the write throughput performance mismatch against SLC memories. Both the ISPP-SV and, to a larger extent, the ISPP-DV feature a large number of verify operations per single ISPP step even if the memory cell is far from the V_{FY} level. An interesting solution to avoid unnecessary verify operations is to use the Reduced Verify (RV) approach [98]. The number of verify operations is automatically increased as soon as the memory cells to be programmed cross a predetermined verification level. The reliability is now traded for increased programming speed as this write methodology may be less robust against page-errors.

Next sections illustrate the modeling effort of the high-voltage memory sub-system required to capture how different programming algorithms impact the raw bit error rate (RBER) and the power consumption of the memory.

4.4 Compact and accurate NAND flash Model

The case study, presented in this thesis, targets a 2-bit per cell NAND flash memory featuring a 45 nm manufacturing process designed for low-power applications. The simulation environment includes two modules: (1) the *high-voltage (HV) sub-system* exploited to generate the voltages required for the programming algorithms (including the verify

stage), and (2) a *compact model of the NAND flash memory* with array simulation capability.

The HV module is the analog core of a NAND flash memory. Modifying or reading the number of electrons stored into the floating gate requires the generation of a set of bias voltages with a desired precision, timing, and granularity. Moreover, since many voltages have a value larger than the NAND power supply, several charge pumps are required. To obtain highly accurate estimations of the energy consumption of each ISPP algorithm considered in this work, we simulated the program charge pump, the inhibit charge pump, the verify charge pump and the regulators/limiting systems according to the guidelines proposed in [80]. All the blocks have been implemented in HSPICE using the STM-45nm technology library [46]. The power consumption of each pump extracted from the SPICE simulation during the various stages of the ISPP algorithms has been then fed into a NAND flash power modeling framework based on the equation set provided by [113]. As input parameters of the model, we assumed a low-power NAND flash supplied with V_{DD} = 1.8V using an ISPP algorithm starting from 14V to 19V and ΔISPP steps of 250mV. The same settings hold for all considered programming algorithms.

The simulated HV sub-system has been designed to work with all algorithms. In fact, in a NAND flash device, the timing and sequence of the analog circuitry operations are driven by the embedded microcontroller/FSM by means of a set of interface registers required to generate the enable signals for the charge pumps. Switching from one ISPP algorithm to another does not require a modification of the HV subsystem. It rather implies a different sequence of enable signals notified through the same register interface.

An additional modeling effort was devoted to model the NAND flash cells. A compact model partially based on [128] has been developed, which includes variability effects typical of nanoscaled memories. This allowed to simulate array functionalities during a page-wide programming operation. The considered variability effects include:

- width and length geometrical variations of FG-MOS transistors;
- non-homogeneity of tunnel oxide and substrate doping;
- tunneling caused by the electron injection granularity process into the cells floating gate;
- cell-to-cell interference caused by cross-talk between adjacent floating gates;

- aging effects due to repeated program/erase cycling which typically degrades the RBER.

All these effects contribute to significantly broaden the gaussian distributions related to the programmed threshold voltage levels within the array, negatively impacting the RBER. A comprehensive description of the adopted model is provided in Appendix A.

For the sake of model validation, we were able to fit experimental data collected from [128] as showed in Fig. 4.5, where cell voltage threshold is plotted during an ISPP operation for a 41nm NAND flash technology.

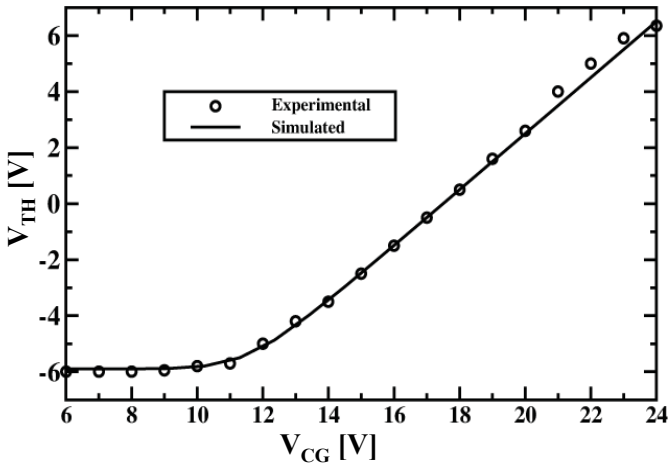
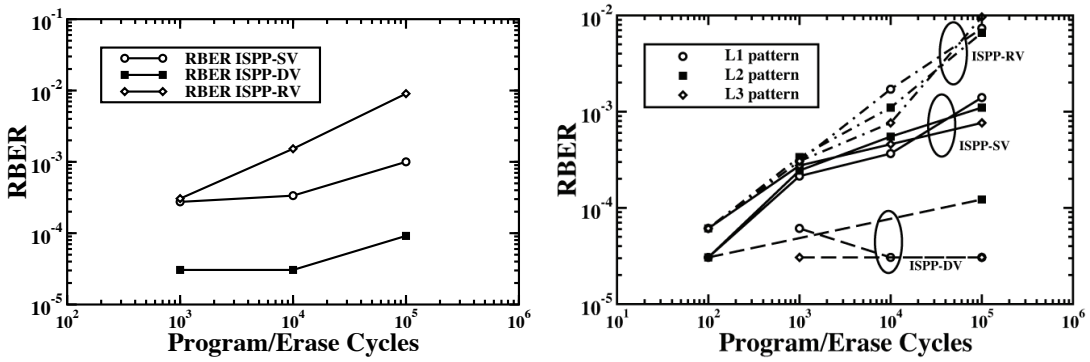


Figure 4.5: Fitting results of the NAND flash compact model with experimental data during an ISPP-SV operation featuring 7μs pulses, 1V ΔISPP

4.5 Characterization of the programming algorithms

Power consumption, RBER and the average page write time of the flash when using the ISPP-SV, the ISPP-DV, and the ISPP-RV algorithms have been characterized by means of the developed simulation framework. For each measured parameter, both the pattern dependent (L1, L2 and L3) and the pattern independent (average over the three patterns) characterization is reported. Such parameters are derived as a function of the program/erase cycles of the memory, thus enabling lifetime-wide assessment of memory features.



(a) pattern independent

(b) pattern dependent

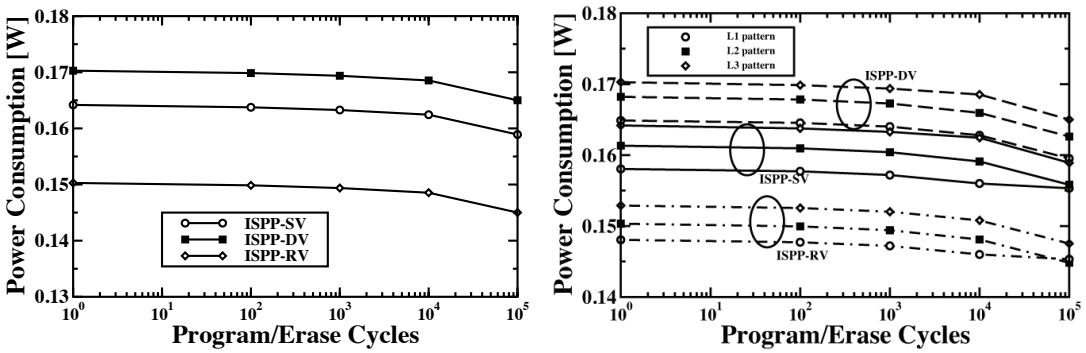
Figure 4.6: RBER characterization

Fig. 4.6 (a) and 4.6 (b) show RBER results for a simulated 4 KB page program of a NAND flash. The choice of a particular programming algorithm turns into a significant modification of the RBER figures up to one order of magnitude.

The power consumption of the memory device during a program operation with different programming algorithms has been measured and reported in Fig. 4.7 (a) and 4.7 (b). Power measures do not include I/O pins and digital portions of the flash, which are irrelevant in the comparative analysis. The most power demanding write strategy is the ISPP-DV, showing a 5% power consumption increment with respect to the baseline ISPP-SV algorithm.This is due to the increased usage of the read charge pump circuitry in the HV sub-system. Nevertheless, it does not represent a major source of power drain in the overall system consumption context. The less power demanding write strategy is straightforwardly the ISPP-RV, as the HV circuitry is enabled for a shorter lapse of time due to the increased speed features of the algorithm.

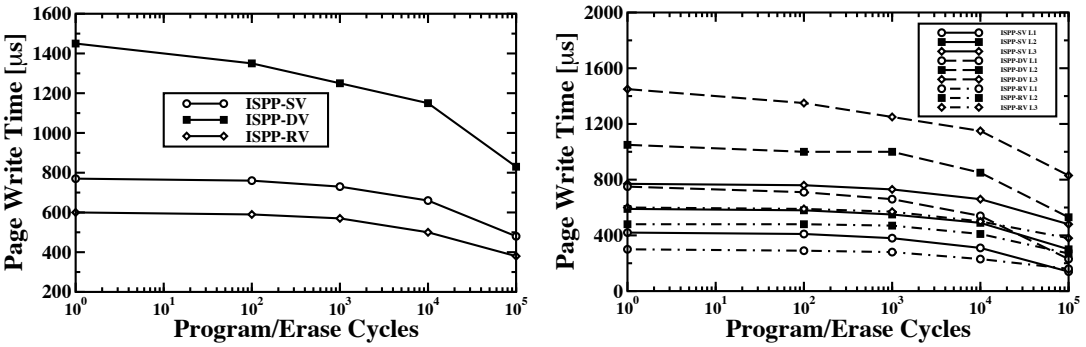
The average page write time has been calculated by simulating a random write pattern on a memory page and taking into account a fixed cell verify time (i.e., page read operation) of 30μs. Results reported in Fig. 4.8 (a) and Fig. 4.8 (b) show that the fastest algorithm is the ISPP-RV due to the reduced number of verify operations that generally cause a slow down of the writing process. What is worth to point out is that the average page write time decreases as the aging increases, due to the fastest programming behavior of the memory cells [98]. This effect is tightly coupled with a reduction of the overall memory reliability since bit errors tend to be more frequent [106].

Table 4.1 summarizes the main results obtained from the characterization of the se-



(a) pattern independent (b) pattern dependent

Figure 4.7: Power consumption characterization



(a) pattern independent (b) pattern dependent

Figure 4.8: Average page write time characterization

lected device.

Page write time (AVG) @ cycle 1	$600\mu s$ (RV) $800\mu s$ (SV) $1400\mu s$ (DV)
Page read time	$75\mu s$
Maximum considered P/E cycles	100000
Page Size	4 KB + Parity

Table 4.1: NAND Flash simulation parameters (Programming timings are provided at cycle 1)

From the presented plots, the potentials of an adaptive memory physical layer for future reconfigurable memory sub-systems become evident. By selecting a programming

algorithm among ISPP-SV, ISPP-DV, and ISPP-RV one can set the desired trade-off point between RBER, power, and write throughput, with only incremental complexity of the memory controller architecture.

After having characterized the proposed flash physical management sub-system, in the next Section different alternatives are proposed to physically implement it in actual memory controllers.

4.6 How implementing the physical-level adaptability in memory controllers

In current flash device controllers, the programming algorithm is set at fabrication time, thus preventing run-time adaptation. It is usually stored in a code-ROM integrated in the same memory die, and executed by an embedded microcontroller. Our physical layer optimization approach moves from the assumption that more than one algorithm can be stored in the code-ROM by slightly increasing its capacity. In addition, a mechanism is needed to select the desire algorithm for a transaction or a set of transactions.

The ONFI 3.0 standard for NAND Flash memories [9] envisions the possibility of implementing both new vendor-specific commands and special commands in case of development of innovative writing methodologies. It could therefore be exploited to implement the write algorithm selection through three dedicated commands such as: 0x80 = Program with ISPP-SV, 0x81 = Program with ISPP-DV, and 0x82 = Program with ISPP-RV.

The choice of the programming algorithm can be also implement through dedicated configuration registers. This approach is consistent with the methodologies exploited in today's NAND flash and memory controllers to expose a reconfigurability of the product (e.g., changing the DDR protocol timings [6], or the storage paradigm [10]).

Another possible solution, which would increase the reconfigurability of the entire NAND memory, is to replace the code-ROM with a SRAM. The SRAM is written by the memory controller with the most suitable algorithm for the memory transaction(s) at hand. Integrating more than one RAM core in the embedded microcontroller is not unusual [98], especially for debug and test purposes. In this case the ONFI command set should be enhanced only to enable writing the SRAM content: 0xFA = Write embedded microcontroller SRAM, 0x80 = Standard program ONFI command.

An alternative to the use of different programming algorithms is represented by the application of different techniques (e.g., virtual step amplitude reduction through bit-

line biasing [98], or look-up-table approaches) that may allow to vary the step characteristics during the ISPP in order to provide a multi-level cells threshold voltage tuning with a finer granularity. In terms of implementation complexity, this approach is comparable with the one selected in this thesis (i.e., multiple programming algorithms), since both require to communicate with the firmware of the NAND Flash internal controller (i.e., the Finite State Machine) which dictates the voltages and the timings requested by the ISPP.

SUMMARY

This chapter introduced the proposed adaptable flash physical management sub-system. It is in charge to manage the high-voltage sub-system of the flash memory device generating the voltage waveforms to operate write, read, and erase operations. Usually the ISPP with single verify algorithm is employed for writing the desired logical level in MLC memories. However, its inherent drawbacks show that more accurate variants called ISSP with reduced verify and ISPP with double verify can maximize write throughput performances and increasing programming accuracy, respectively. The proposed physical layer optimization approach allows, thus, to switch among these three variants at run-time. The designed simulation environment includes two modules: (1) the high-voltage (HV) sub-system, (2) a compact model of the NAND flash memory with array simulation capability. By resorting to this simulation framework, power consumption, RBER and the average page write time of the flash when using the ISPP-SV, the ISPP-DV, and the ISPP-RV algorithms have been accurately characterized.

CHAPTER 5

ADAPTABLE ECC ENCODING/DECODING STRUCTURE (ARCHITECTURAL-LEVEL ADAPTIVITY)

Contents of this chapter

This Chapter proposes the adaptable ECC sub-system which introduces additional adaptation acting at the architectural level.

The ECC sub-system proposed in this chapter implements the adaptable Bose-Chaudhuri-Hocquenghem (BCH) ECC architecture presented in [60]. BCH codes belong to the larger class of cyclic codes which have efficient decoding algorithms due to their strict algebraic architecture [22]. BCH codes perform correction over single-bit symbols and better perform when bit errors are not correlated, or randomly distributed. Several studies have reported that NAND flash memories manifest non-correlated or randomly distributed bit errors over a page [138]. BCH codes are therefore a perfect choice for their protec-

tion.

The architecture of the programmable BCH encoder and decoder, introduced in this chapter, provides a fine tuning of the ECC correction capability whose benefits are pointed-out by accurately characterizing the ECC sub-system, showing the different trade-offs offered by its programmability.

The implementation is supported by the novel ADaptive ECC Automatic GEnerator (ADAGE) design environment . This tool is able to automatically generate, in a parametric way, the whole code for each possible architecture. ADAGE concepts will be shortly introduced, when required, in the next sections.

The chapter is organized as follows: Section 5.1 shortly introduces basic notions and related works. Sections 5.2 and 5.3 present a solution to reduce resources overhead, while Section 5.4 and 5.5 overview the proposed adaptable architecture. Section 5.6 provides experimental results and concludes the chapter.

5.1 Background and related works

Several hard- and soft-decision error correction codes have been proposed in the literature, including Hamming-based block codes [64, 104], Reed-Solomon codes [123], Bose-Chaudhuri-Hocquenghem (BCH) codes [22], Goppa codes [18], and Golay codes [62].

Even though selected classes of codes such as Goppa codes have been demonstrated to provide high correction efficiency [18], when considering the specific application domain of flash memories, the need to trade-off code efficiency, hardware complexity and performances have moved both the scientific and the industrial community toward a set of codes that enable very efficient and optimized hardware implementations [45, 85].

Old SLC flash designs used very simple Hamming-based block codes. Hamming codes are relatively straightforward and simple to implement in both software and hardware, but they offer very limited correction capability [64, 104]. As the error rate increased with successive generations of both SLC and MLC NAND flash memories, designers moved to more complex and powerful codes including Reed-Solomon (RS) codes [123].

An exhaustive analysis of the mathematics governing BCH code is out of the scope of this chapter. Only those concepts required to understand the proposed hardware implementation will be shortly discussed. It is worth to mention here that, since several publications proposed very efficient hardware implementations of Galois fields polyno-

mial manipulations, such manipulations will be used in both encoding and decoding operations [87, 102, 123].

Given a finite Galois field $GF(2^m)$ (with $m \geq 3$), a t-error-correcting BCH code, denoted as $BCH[n,k,t]$, encodes a k-bit message $b_{k-1}b_{k-2}\dots b_0$ ($b_i \in GF(2)$) to a n-bit codeword $b_{k-1}b_{k-2}\dots b_0\, p_{r-1}p_{r-2}\dots p_0$ ($b_i, p_i \in GF(2)$) by adding r parity bits to the original message. The number r of parity bits required to correct t errors in the n-bit codeword is computed by finding the minimum m that solves the inequality $k + r \leq 2^m - 1$, where $r = m \cdot t$. Whenever $n = k + r < 2^m - 1$, the BCH code is called *shortened* or *polynomial*. In a shortened BCH code the codeword includes less binary symbols than the ones the selected Galois field would allow. The missing information symbols are imagined to be at the beginning of the codeword and are considered to be 0. Let α be a primitive element of $GF(2^m)$ and $\psi_1(x)$ a primitive polynomial with α as a root. Starting from $\psi_1(x)$, a set of minimal polynomials $\psi_i(x)$ having α^i as root can be always constructed [118]. For the same $GF(2^m)$, different valid $\psi_1(x)$ may exist [132]. The generator polynomial $g(x)$ of a t-error-correcting BCH code is computed as the Least Common Multiple (LCM) among $2t$ minimal polynomials $\psi_i(x)$ ($1 \leq i \leq 2t$). Given that $\psi_i(x) = \psi_{2i}(x)$ ($\forall\, i \in [1,t]$) [14], only t minimal polynomials must be considered and $g(x)$ can therefore be computed as:

$$g(x) = LCM\left[\psi_1(x), \psi_3(x) \dots, \psi_{2t-1}(x)\right] \tag{5.1}$$

When working with BCH codes, the message and the codeword can be represented as two polynomials: (1) $b(x)$ of degree $k-1$ and (2) $c(x)$ of degree $n-1$. Given this representation, both the encoding and the decoding process can be defined by algebraic operations among polynomials in $GF(2^m)$. The encoding process can be expressed as:

$$c(x) = m(x) \cdot x^r + Rem\left(m(x) \cdot x^r\right)_{g(x)} \tag{5.2}$$

where $Rem(m(x) \cdot x^r)_{g(x)}$ denotes the remainder of the division between the message left shifted of r positions and the generator polynomial $g(x)$. This remainder represents the r parity bits to append to the original message.

The BCH decoding process searches for the position of erroneous bits in the codeword. This operation requires three main computational steps: 1) syndrome computation, 2) error locator polynomial computation, and 3) error position computation.

Given the selected correction capability t, the decoding process requires first the computation of $2t$ syndromes of the codeword $c(x)$, each associated with one of the $2t$ minimal polynomials $\psi_i(x)$ generating the code. Syndromes are calculated by first computing the remainders $R_i(x)$ of the division between $c(x)$ and each minimal polynomial $\psi_i(x)$. If all remainders are null, $c(x)$ does not contain any error and the decoding stops. Otherwise, the $2t$ syndromes are computed by evaluating each remainder $R_i(x)$ in α^i: $S_i = R_i\left(\alpha^i\right)$. Practically, according to (5.1), given that $\psi_i(x) = \psi_{2i}(x)$, only t remainders must be computed and evaluated in $2t$ elements of $GF(2^m)$.

The most used algebraic method to compute the coefficients of the error locator polynomial from the syndromes is the Berlekamp-Massey algorithm [19]. Since the complexity of this algorithm grows linearly with the correction capability of the code, it enables efficient hardware implementations. The equations that link syndromes and error locator polynomial can be expressed as:

$$\begin{pmatrix} S_{t+1} \\ S_{t+2} \\ \vdots \\ S_{2t} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 & \dots & S_t \\ S_2 & S_3 & \dots & S_{t+1} \\ \vdots & \vdots & & \vdots \\ S_t & S_{t+1} & \dots & S_{2t-1} \end{pmatrix} \cdot \begin{pmatrix} \lambda_t \\ \lambda_{t-2} \\ \vdots \\ \lambda_0 \end{pmatrix} \tag{5.3}$$

The Berlekamp-Massey algorithm iteratively solves the system of equations defined in (5.3) using consecutive approximations.

Finally, the Chien Machine searches for the roots of the error locator polynomial $\lambda(x)$ computed by the Berlekamp-Massey algorithm [42]. It basically evaluates the polynomial $\lambda(x)$ in each element α^i of $GF(2^m)$. If α^i satisfies the equation $1+\lambda_1\alpha^i+\lambda_2\alpha^{2i}+\ldots+\lambda_t\left(\alpha^i\right)^t = 0$, α^i is a root of the error locator polynomial $\lambda(x)$, and its reciprocal 2^m-1-i reveals the error position. In practice, this computation is performed exploiting the iterative relation:

$$\lambda\left(\alpha^{j+1}\right) = \lambda_0 + \sum_{k=1}^{t-1}\left[\lambda_k\left(\alpha^j\right)^k\right]\alpha^k \tag{5.4}$$

Several publications proposed optimized hardware implementations of BCH codecs with fixed correction capability [45, 63, 85, 103, 122, 127]. However, to the best of our knowledge, only Chen et al. proposed a solution allowing limited adaptation by extending a standard BCH codec implementation [37]. One of the main contributions of Chen et al. is a Programmable Parallel Linear Feedback Shift Register (PPLFSR), whose generic