EXPLOITING PHYSICAL PROPERTIES OF FLASH MEMORIES FOR ENHANCING SECURITY AND ENERGY EFFICIENCY OF EMBEDDED SYSTEMS

by

PRAWAR POUDEL

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
The Department of Electrical & Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2021

In presenting this dissertation in partial fulfillment of the requirements for a Doctor of Philosophy degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this dissertation.

______________________________ ____________
(student signature) (date)

DISSERTATION APPROVAL FORM

Submitted by Prawar Poudel in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering and accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering.

Committee Chair
(Dr. Aleksandar Milenkovic) (date)

Committee Member
(Dr. Biswajit Ray) (date)

Committee Member
(Dr. David Coe) (date)

Committee Member
(Dr. Jeffrey H Kulick) (date)

Committee Member
(Dr. Mohammad Haider) (date)

Department Chair
(Dr. Ravi Gorur) (date)

College Dean
(Dr. Shankar Mahalingam) (date)

Graduate Dean
(Dr. Sane Lane) (date)

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Engineering/Electrical & Computer Engineering

Name of Candidate Prawar Poudel
Title Exploiting Physical Properties of Non-Volatile Memories for Enhancing Security and Energy Efficiency of Embedded Systems

Counterfeit electronics has become a significant concern in the globalized semiconductor industry. Recycled, over-produced, out-of-spec, defective, cloned, or tampered-with chips can enter the supply chains. These chips end up in a variety of products, from low-end consumer gadgets to mission-critical systems that are used in transportation, finance, military, or healthcare applications. Non-volatile flash memory has been one of the primary targets of counterfeiters due to its proliferation in storage solutions, e.g., SSD drives, USB drives, and SD cards, as well as in embedded and Internet-of-Things platforms. Thus, finding new ways to ensure that only genuine non-volatile flash memory is used in electronic products is very important for chip manufacturers, industry, governments, and consumers alike. Unfortunately, the existing approaches for tracking origins of flash-memory chips or chips with embedded flash memory modules can easily be circumvented by motivated and resolute counterfeiters. On the other side, more complex technical solutions suggested by other researchers in academia often incur significant costs due to additional on-chip resources, required changes in physical masks, and required changes in common practices and product flows used in the industry.

This dissertation introduces two techniques to help secure chips containing flash memory by exploiting their physical properties. First, we introduce a technique for fingerprinting of microcontrollers with embedded NOR flash memory. This technique leverages partial erase operations of flash memory segments to expose semiconductor process variations and defects that are unique to each memory segment. Second, we introduce Flashmark, a technique for watermarking NOR flash memory through repeated stressing that irreversibly changes physical properties of flash memory cells. A corresponding watermark extraction technique is developed to retrieve imprinted watermark through digital interfaces. Finally, this dissertation introduces a technique that reduces time and energy consumed by critical flash memory operations in ultra-low-power microcontrollers. The proposed technique utilizes partial NOR flash memory erase and program operations that proved to have no negative impacts on accuracy and longevity of information stored in the flash memory. The experimental evaluation utilizing a family of commercial microcontrollers demonstrates that the proposed techniques are cost-effective, robust, and resilient to changes in voltage and temperature as well as to aging effects.

Abstract Approval: Committee Chair ______________________

Department Chair ______________________

Graduate Dean ______________________

ACKNOWLEDGMENTS

My utmost and sincere thanks go to Dr. Aleksandar Milenkovic for his continued support and trust that he had put on me. His continuous help, guidance, and inspiration has helped me complete this dissertation and the research presented herein. The experience that I have gained working with him, technical and otherwise, will always hold dear to me.

I would like to thank Dr. Biswajit Ray, Dr. David Coe, Dr. Jefferey Kulick and Dr. Mohammad Haider for serving in my committee. I would also like to thank Dr. Ravi Gorur, the Chair of the Electrical and Computer Engineering Department, for providing me financial support and an opportunity to work as a Graduate Teaching Assistant during my time at the University of Alabama in Huntsville.

I am grateful to my lab mates Dr. Mounika Ponugoti, Mr. Ranjan Hebbar, Mr. Igor Semenov, and Mr. Amir Ramezani whom I could reach out any time with the problems that I faced.

Special thanks go to Ms. Jacqueline Siniard and Ms. Danniele Worsham who have always helped me promptly regarding administrative matters. Similarly, I would also like to thank Mr. Chris Hardy for his technical assistance in lab during my time at UAH.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Pages

LIST OF CODES

Code Pages

CHAPTER 1

INTRODUCTION

Non-volatile memory (NVM) is a type of computer memory that retains its content after the removal of power supply. NVMs play an important role in providing persistent storage for code and data in a range of computing platforms, including low-end embedded and Internet-of-Things (IoTs) platforms, smartphones, personal computers, high-end workstations, and warehouse-scale computers.

The earliest NMVs such as EPROMs (Erasable Programmable Read Only Memory) and EEPROMs (Electrical Erasable Programmable Read Only Memory) have been replaced by new technologies that overcome some of their shortcomings: EPROMs erase require exposure to ultra-violet (UV) light, whereas EEPROMs are relatively slow. Toshiba introduced flash memory technology in 1987 [1]. Flash memory gets its name from having fast erase times. In addition to being fast, flash memories have a high bit density [2], [3] and today they represent the dominant type of NVMs. More recently, newer types of NVMs are emerging, including Magneto-Resistive Random Access Memories (MRAM), Ferro-electric Random Access Memories (FeRAM), Resistive Random Access Memories (RRAM) and 3D XPoint memory. This dissertation focuses on flash memories.

A basic building block of flash memories is a flash memory cell that typically holds one bit of information stored in the form of charge on the floating gate of a Floating Gate Metal Oxide Semiconductor Field Effect Transistor (FG-MOSFET). Based on the organization of the flash memory cell arrays, flash memories are classified into two groups: (a) NOR flash memory and (b) NAND flash memory. NOR flash memories are typically used for storing firmware, boot codes, and keeping user data, whereas NAND flash memories are mostly used for mass storage purposes. Of all the NVMs, flash memories are the most widely used. The annual revenue from sales of NAND flash memories alone is currently US$ 60 billion and is forecast to reach US$ 80.3 billion by 2025 [4]. High density NAND flash memories are used in portable storage media such as USB drives and SD cards, automobiles, as well as in gadgets like MP3 players, cameras, smartphones, and computers. Colloquially referred to as Solid State Drives (SSDs), flash memory based mass storage media significantly outperform traditional Hard Disk Drives (HDDs) [5]. Thus, flash memories are also extensively used in high end computing platforms, data centers, and in the cloud [6], [7].

It is projected that 50 billion IoT devices will be connected to the Internet by 2030 [8]. Mobile and embedded devices including wireless sensor networks, industrial systems, transportation systems, as well as wearable and implanted electronics are ubiquitous. The increase in the number of connected devices that perform important and mission-critical tasks brings forth new design challenges related to their security and energy efficiency. Verifying authenticity of connected devices is one design issue that is critical for ensuring security and integrity. Recent reports about counterfeit electronic components being used in commercial products have caused significant concern for semiconductor industry and consumers alike [9]–[12]. Authentication of electronic components can be used to ensure integration of genuine components during system design.

The problem of authenticating a device is solved by assigning an identifier to each device. Such identifiers are saved in non-volatile memories or battery backed RAM. However, storing identifiers in memory opens possibility of adversaries cloning the identifiers. Physical Unclonable Functions (PUFs) extract identifiers from physical properties of a device to reflect inherent manufacturing and process variations [13], [14] from functionally similar devices. These identifiers act as fingerprints and remove the need for identifiers to be stored in NVMs or battery backed RAM. Since the fingerprints are derived from the die-to-die manufacturing variations, it is difficult to recreate or clone them. There are many proposals of custom circuits for PUF based fingerprint generation [13], [15]–[17]. However, adding extra components increase the on-chip area as well as manufacturing cost. Thus, recent proposals focus on using existing resources for fingerprint extraction [18]–[20]. This dissertation introduces a new technique that extracts fingerprints from unique properties of partially erased NOR flash memories embedded in modern microcontrollers (Section 1.1).

The use of PUFs for detection of counterfeits require lengthy PUF extraction as well as maintenance of large databases with entries for every manufactured chip. In addition, it requires a method for contacting the chip manufacturer to verify the authenticity of each chip, which may place an additional burden on system integrators, as the time and costs of verification are increased. To address this concern, this dissertation introduces a watermarking technique to permanently imprint the manufacturer information into the flash memory. This watermarking can be used to distinguish between counterfeit and genuine electronic devices (Section 1.2.)

Although flash memory chips offer a more energy-efficient storage alternative to storage solution based on hard disks, flash memory operations require significant amount of energy. Flash program and erase operations are power hungry as they rely on internal charge pumps to generate high voltages needed to move charges to/from floating gates within flash memories. Thus, finding a way to minimize energy consumed by these operations is very beneficial for systems that are frequently updated or use internal flash memory for storing application critical data. Energy efficient flash memory operations pave way for reduction in operating costs, especially in case of implanted electronics. To address this concern, this dissertation introduces a technique that utilizes partial or aborted flash memory erase and program operations that proved to have no negative impacts on accuracy of information stored in the flash memory, but provide significant savings in time and energy (Section 1.3)

Newer flash memory technologies are based on 3D structure. Further scaling of planar (2D) flash memory is not possible beyond 14 nm because of significant concerns in structural and mechanical integrity, as well as due to concerns in reliability. Characterization of the newer flash memory offers insight into the properties of the flash memory that can be used to enhance performance and reliability. This dissertation characterizes a state-of-the-art family of 3D flash memory (Section 1.4).

1.1 Microcontroller Fingerprints

Recent proposals for generation of fingerprints offer solutions to verify authenticity of an electronic device by providing a unique identifier that is derived from manufacturing variations [18], [19], [21]. However, these techniques often rely on complicated algorithms for PUF generation. These algorithms require computationally cost-prohibitive operations for low-end microcontrollers and/or privileged operations; for example, lowering supply voltages and power-up cycle [18], [19], [22].

Use of embedded flash memory for generating fingerprint in low-end microcontrollers for authentication purpose will create a widely adaptable solution. Embedded NOR flash memories in microcontrollers are typically used for storing code and data and they are treated as read-only memories. However, modern microcontrollers support in-system programmable flash memories that can be erased and programmed internally by running programs. This dissertation uses early termination of flash erase operation to expose manufacturing variation of the flash memory for fingerprint generation. The extracted fingerprint can serve as a device fingerprint, is highly reliable, requires lightweight compute and modest storage resources. These features make the proposed technique highly suitable for low-end microcontrollers.

1.2 Watermarking

Several research efforts that are proposed to detect counterfeit flash memory chips [23]–[25], exploit the fact that the physical properties of a used flash memory is significantly different than a new flash memory chip. Unfortunately, these techniques can only be applied to detect recycled flash memory chips.

A technique to permanently imprint a manufacturer information into the flash memory will allow a customer or a system integrator to ascertain that the flash memory chip or a system-on-a-chip (SoC) with embedded flash memory is genuine. For example: a manufacturer can imprint "accept" or "reject" information on every die they produce. This will prevent the counterfeiters from entering the out-of-spec or fall-out chips into the supply chain. System integrators and end users will be able to detect counterfeit chips before integrating them in their products.

This dissertation presents a watermarking technique that exploits a known property of flash memory that flash cell oxides degrade when exposed to stress (repeated program-erase operation). The degradation of oxide is permanent and cannot be reversed. Selective stressing of the flash memory cells can be exploited to imprint manufacturer id, speed grade, chip test status and other manufacturer related information into the flash memory permanently. Thus, the watermarking technique is robust and can be used to verify that the flash memory chip or the SoC is genuine. The proposed watermarking technique is demonstrated using embedded NOR flash memory in a family of low-end microcontroller.

1.3 Energy Efficient Flash Memory Operations

Since writing and updating data into the embedded flash memory is possible through the software code running in the processor core, it is also used as a non-volatile storage of application data in sensor applications and IoT to prevent loss of data in case of power loss. However, the energy consumed by flash memory operations (both program and erase) are significantly higher.

This dissertation characterizes flash memory operations in a family of low-end microcontrollers and makes an observation that early termination of erase operation successfully erases flash memory. Similarly, early termination of program operation can program the flash memory without loss of any information. Experimental evaluation performed on NOR flash memory of low-end microcontroller reveals that significant energy is saved by such early termination of flash memory operations. Thus, this dissertation proposes replacing nominal flash memory operations with partial flash memory operations to save energy consumed.

1.4 3D NAND Flash Memory Characterization

Characterizing flash memory offers insight into its reliability and performance metrics, such as bit error rates, program times, erase times and performance degradation with ageing. These metrics are useful for creation of better algorithm for Flash Translation Layer (FTL) design. FTL is a system software that interfaces flash memory chips to the host computer. It sits between the file system and the flash memory and hides the details of interfacing.

This dissertation performs timing characterization of a family of state-of-the-art 3D NAND flash memory using an FPGA based set up that is developed in-house. This characterization reveals that significant timing differences exists between programming units i.e. between different pages while programming.

1.5 Major Contributions

The major contributions of this work are as follows:

1. It characterizes the behavior of embedded NOR flash memories based on early termination of erase operations, i.e. partial erase operations

and early termination of program operations, i.e. partial program operations.

2. It introduces a technique for extracting fingerprints from the partially erased flash memory and proposes a fingerprint-based authentication system. It explores the robustness of the proposed technique as a function of the environmental conditions and usage history of the chips. The results demonstrate that proposed technique is cost-effective, robust and resilient to changes in voltages and temperature as well as ageing effects.
3. It introduces a watermarking technique for securing global supply chains of NOR flash chips or chips containing embedded NOR flash memory. The watermark imprinting is permanent and can be used to identify genuine chips from counterfeit ones.
4. It introduces an energy efficient flash memory technique in ultra-low-power microcontrollers based on partial flash memory operations. The experimental evaluation performed on a family of microcontrollers shows that the proposed technique can save up to 98% of the energy consumed for flash erase operations and up to 75% for flash program operations.
5. It characterizes a family state-of-the-art 3D NAND flash memory that reveals significant timing variations in the program operations between different pages.

1.6 Outline

The remaining of this dissertation is organized as follows. CHAPTER 2 gives a brief background discussing principles of NOR and NAND flash memory organization and their operation. CHAPTER 3 describes experimental platforms used in this research. CHAPTER 4 presents the microcontroller fingerprint based authentication technique. It first characterizes the flash memory cells. Then, it outlines the algorithm for enrollment of a fingerprint and authentication of a fingerprint. It also presents the results of experimental evaluation and discusses the robustness of the proposed fingerprint technique. CHAPTER 5 presents the watermarking technique. It first characterizes the flash memory for stress induced properties and presents the technique to imprint the manufacturer information into the flash memory. Then, it presents the experimental evaluation. CHAPTER 6 first characterizes the partial flash memory operations and presents the energy saving techniques using the results of the characterization. The experimental evaluation and the robustness of the proposed energy saving technique is also presented in the chapter. CHAPTER 7 presents the characterization of a family of 3D NAND flash memory and suggests possible ways the results can be used to improve the design of the flash translation layer. CHAPTER 8 concludes the dissertation.

CHAPTER 2

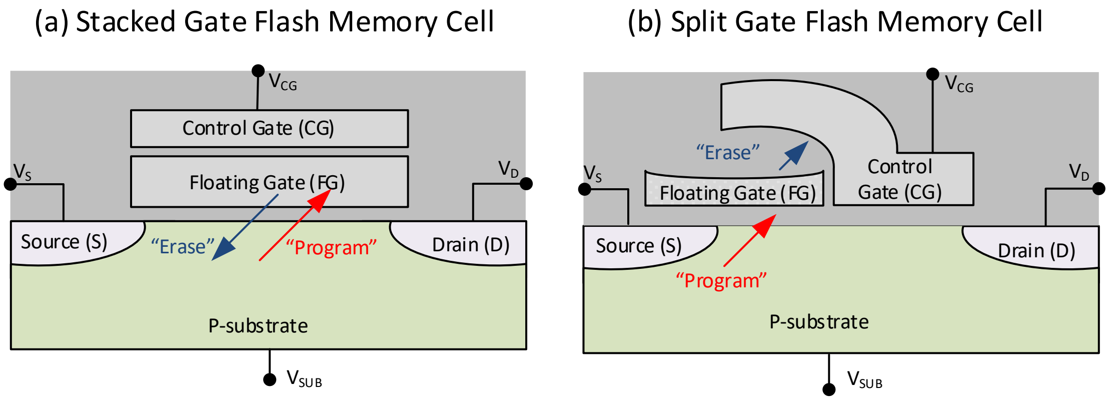
BACKGROUND

Flash memory is composed of an array of memory cells. These flash memory cells are similar in structure to MOSFETs (Metal-Oxide-Semiconductor Field-Electric Transistor) with an important distinction: flash memory cells have an extra gate in addition to the Control Gate (CG). This additional gate is called Floating Gate (FG). Thus, flash memory cells are also called FG-MOSFETs. The floating gate sits between the CG and substrate and is surrounded by insulating oxide layers on all sides. Charges can be trapped on the FG by application of appropriate electric fields. These charges cannot escape the floating gate even after the removal of external electric fields because of the insulating oxides. This makes FG-MOSFETs a building block for non-volatile memories.

In this chapter, we dive a bit deeper into the basics of flash memory cells, discuss different types of flash memories, and describe how they work. In Section 2.1, we look into the structure of the FG-MOSFET. We discuss basic flash memory cell operations in Section 2.1.1 and Section 2.1.2. In Section 2.2, we introduce flash memory cells that store more than one bit of information per cell. In Section 2.3, we present the organization of flash memory arrays. In Section 2.4, we present a brief introduction of a 3D NAND flash memory that utilizes charge-trap (CT Flash) technology.

2.1 Structure of Flash Memory Cells

The cross-section of an FG MOSFET is shown in Figure 2.1(a). The FG-MOSFET consists of the Control Gate (CG) and the Floating Gate (FG) stacked on top of each other, instead of a single gate terminal as in an MOSFET. Because of this stacked structure where CG is placed on the top of FG, FG-MOSFETs are also called *Stacked Gate Flash Memory Cells*. *Split Gate Flash Memory Cells* are slightly different types of FG MOSFETs where the CG covers a portion above the substrate and also covers a portion over the FG, whereas the FG only covers a part above the substrate [26], [27]. Cross-section of a split-gate FG MOSFET is shown in Figure 2.1(b).


(a) Stacked Gate Flash Memory Cell

(b) Split Gate Flash Memory Cell

Figure 2.1 (a) Stacked Gate FG MOSFET cross-section; (b) Split-Gate FG MOSFET cross-section.

The source (S) and drain (D) terminals in the substrate correspond to the terminals in a conventional MOSFET. Similar to the traditional MOSFETs, the control gate in the FG-MOSFET is used to control switching operations. If the applied voltage at the control gate is greater than a certain threshold voltage (V_{TH}), the current flows from source to drain.

The FG traps charge carriers or electrons. The presence of electrons in the FG increases the threshold voltage, relative to the threshold voltage when the FG is devoid of electrons. Based on the presence or absence of electrons, an FG-MOSFET can be in one of two states: *Programmed* and *Erased*. The presence of electrons in the FG corresponds to the programmed state that is characterized by the threshold voltage V_{THP}. Absence of electrons in the FG corresponds to the erased state characterized by the threshold voltage V_{THE}.

The symbol for an FG MOSFET is shown in Figure 2.2(a). Figure 2.2(b) shows the current-voltage characteristic of FG-MOSFETs. To determine the state of a flash memory cell, a read reference voltage (V_{REF}) is applied to the control gate. This voltage is placed in between the threshold voltages of the erased and programmed states, i.e. $V_{THE}<V_{REF}<V_{THP}$. The state of the cell is determined based on whether there is a current flow (I_{DS}) between the source and drain or there is no current flow. A sensing circuitry is designed to detect the current; if there is a current flow, the flash memory cell is determined to be in the erased state ($V_{REF} > V_{THE}$), whereas absence of the current flow indicates the flash memory cell is in programmed state (the transistor is on, $V_{REF} < V_{THP}$).

Due to manufacturing differences and difficulty in controlling electric charge on the floating gate, the threshold voltages of the erased (V_{THE}) and the programed (V_{THP}) states are not constant values, but rather are modeled as a Gaussian distribution. Figure 2.2(c) shows the probability density function (PDF) of V_{THE} and V_{THP}. The read reference voltage V_{REF} is placed in the middle relative to the peaks with enough margin to determine that a cell state is read correctly. The erased state is represented as a logic ‘1’ and the programmed state is represented as a logic ‘0’ as shown in Figure 2.2(b) and Figure 2.2(c).

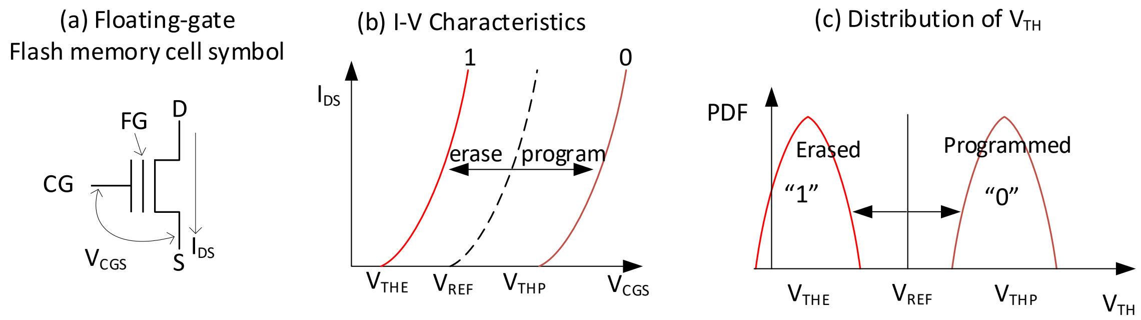


Figure 2.2 (a) FG MOSFET symbol; (b) I-V characteristic of FG MOSFET; (c) Threshold voltage (V_{TH}) distribution of FG MOSFET.

The major operations that can be performed on a flash memory cell are (i) Erase; (ii) Program and (iii) Read. Since the memory cells store information in the form of charge on the floating gate, these operations deal with removing, injecting, or sensing the charge (electrons) on the floating gate, respectively. In the following section, we discuss these operations in flash memory cells and required biasing in their terminals based on types of the flash memory cells. Section 2.1.1 discusses stacked gate cells, whereas Section 2.1.2 discusses split-gate flash memory cells.

2.1.1 Stacked Gate Flash Memory Cell Operations

Erase. For the electrons to be removed from the FG of *Stacked Gate* cell, the control gate CG is grounded. A high positive voltage (~20 V) is applied to the substrate while source and drain terminals are kept floating. The electrons that are trapped in the FG are ejected from the FG to the substrate through Fouler Nordheim (FN) tunneling as shown by blue arrow in Figure 2.1(a).

Program. A program operation involves injecting electrons into the FG of the flash memory cell as shown by a red arrow in Figure 2.1(a). The mechanism that is utilized in this process is also FN tunneling of electrons from the channel formed into the FG. Electrically, for the flash memory cell to be programmed, a high program voltage is applied to the control gate (~15-20 V), the source terminal is grounded and the drain is provided with a high positive voltage (Vcc) [28]. Typically, a program operation relies on an iterative approach called incremental step pulse programming (ISPP) – it consists of multiple short program pulses followed by a corresponding verify steps.

Read. For a read operation, a zero voltage is applied to the CG [28]. Since the threshold voltage of an erased flash memory cell (V_{THE}) is less than 0 V [28], [29], an erased flash memory cell will conduct current with source terminal grounded and drain terminal biased at a positive voltage. Thus, it will be read as logic '1'. A programmed flash memory cell will be read as logic '0' because it will not be able to conduct any current.

2.1.2 Split-Gate Flash Memory Cell Operations

Erase. An erase operation requires a high voltage on the CG (V_{CG} ~ 12 V), whereas the source and drain terminals are grounded. The electrons in the FG are removed through the FN tunneling as shown by the blue arrow that is labeled "Erase" in Figure 2.1(b) above. The removal of electrons from the FG reduces the threshold voltage, i.e. V_{TH}=V_{THE}.

Program. During a program operation electrons are injected into the FG by the source-side hot-carrier injection (SSI) method as shown in by the red "Program" arrow in Figure 2.1(b). A large voltage is applied to the source terminal (V_S ~ 10 V), whereas a small positive voltage is applied to the CG (V_{CG} ~ 2 V). The drain terminal is kept at a very small positive voltage (V_D ~ 0.5 V).

Read. A read operation for sensing the threshold voltage of the split gate flash memory involves applying a small positive read voltage to the CG. Here, $V_{CG} = V_{READ}$ ~ 3V, whereas a sensing voltage is applied to the drain terminal ($V_D = V_{SENSE}$ ~ 2 V). An erased split gate flash memory cell will conduct the current (logic '1'), whereas a programmed cell will not conduct the current (logic '0').

2.2 Multi-Level Cell Flash Memory

The threshold voltage distributions shown in Figure 2.2(c) assume that one flash memory cell stores a single bit of information. These flash memory cells are called Single-Level Cells (SLCs), and the memory composed of SLC flash cells are called SLC flash memory. However, due to continued scaling and technological improvements, multiple bits of information can be stored in a single flash memory cell. Such flash memories are consequently called Multi-Level Cell flash memories. The multiple bits of information stored in the flash memory cell can be distinguished by multiple levels of the threshold voltage, V_{TH}. For example, for a n-bit per cell n-MLC flash memory, the threshold voltage can assume any one of 2^n possible levels.

Conventionally, MLC refers to 2-bit flash memory cell where the threshold voltage can assume any of 2^2 or 4 voltage levels. Higher density arrangements are also available today in commercial products, such as Triple-Level Cell (TLC, 3-bits per cell, 8 V_{TH} levels), Quad-Level Cell (QLC, 4-bits per cell, 16 V_{TH} levels), and Penta-Level Cell (PLC, 5-bit per cell, 32 V_{TH} levels). Figure 2.3 shows V_{TH} distributions for different types of multi-level flash memory cells. The bits that each V_{TH} distribution corresponds to are labeled in each of the plots. With an increase in the number of V_{TH} levels, multiple read reference voltages are also needed to differentiate between the V_{TH} levels. For a n-bit multi-level cell, 2^n-1 read reference voltages are needed. These multiple read reference voltages are appropriately shown in Figure 2.3.

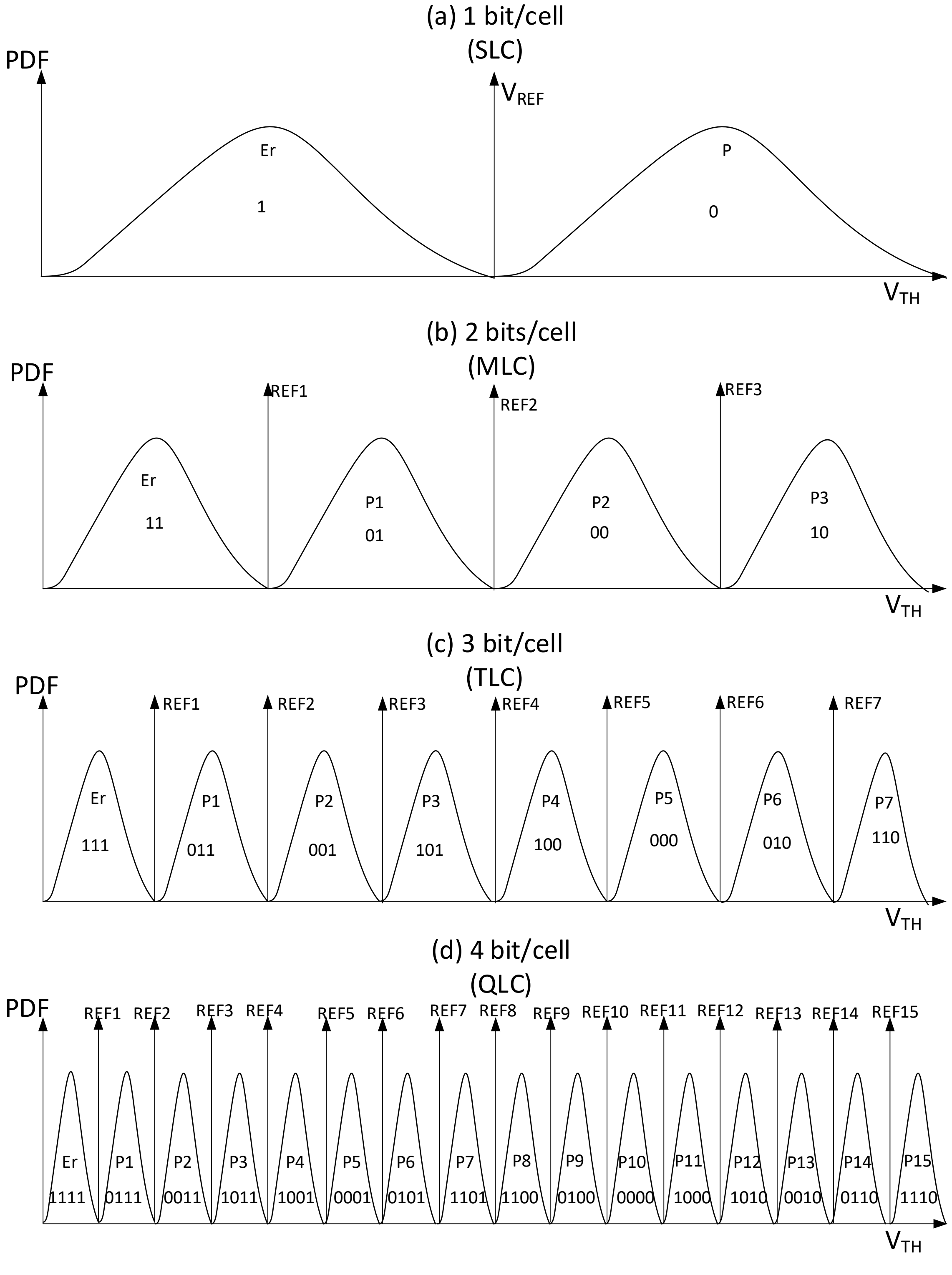


Figure 2.3. Threshold voltage (V_{TH}) states in multi-level cell flash memory.

The program and read operations in multi-level cell flash memory involve a sequence of steps. To keep the discussion simple, let us consider the case of a 2-bit MLC flash memory as an example. The two bits that are stored in each cell of MLC flash memory are encoded as (*MSB*, *LSB*). The *MSB* indicates the most significant bit value while *LSB* indicates least significant bit value. The encoding of the V_{TH} to represent one of four states is shown in Figure 2.4(b) where *Er* is the erased state. *Er* corresponds to the logical value of (1,1) whereas other three programmed states are indicated by *P1*, *P2* and *P3*. The encoding of the adjacent programmed states differs by a Hamming Distance of 1 bit where *P1* = (0,1), *P2* = (0,0) and *P3* = (1,0).

Figure 2.4 shows the sequence of program stages in an MLC flash memory cell. Of the two bits to be programmed into the MLC, the first stage of program operation injects charge corresponding to the LSB bit. For example, if the LSB is '1', there is no significant change in V_{TH} due to this program operation. But in the case where LSB is '0', V_{TH} assumes an intermediary state as shown by the middle plot in Figure 2.4. Then, the second stage of program operation injects charge into FG based on MSB. This causes the flash memory cell to attain its final V_{TH}.

A read operation involves detecting the V_{TH} state by performing multiple comparisons. This involves using multiple read reference voltages (REF1, REF2 and REF3) as shown in Figure 2.4. For an MLC flash memory cell, the following sequence of steps occurs when reading its LSB or MSB state:

1. To read the LSB bit of a flash memory cell, V_{TH} is compared with REF2 voltage.
    - If the V_{TH} detected is greater than REF2, the value of LSB is read as 0.
    - Otherwise, LSB is 1.

2. To read the MSB bit, two reference voltage (REF1 and REF3) comparisons should be made.
    - ○ If V_{TH} detected is greater than REF1 and less than REF3, then MSB is 0.
    - ○ Otherwise, MSB is 1.

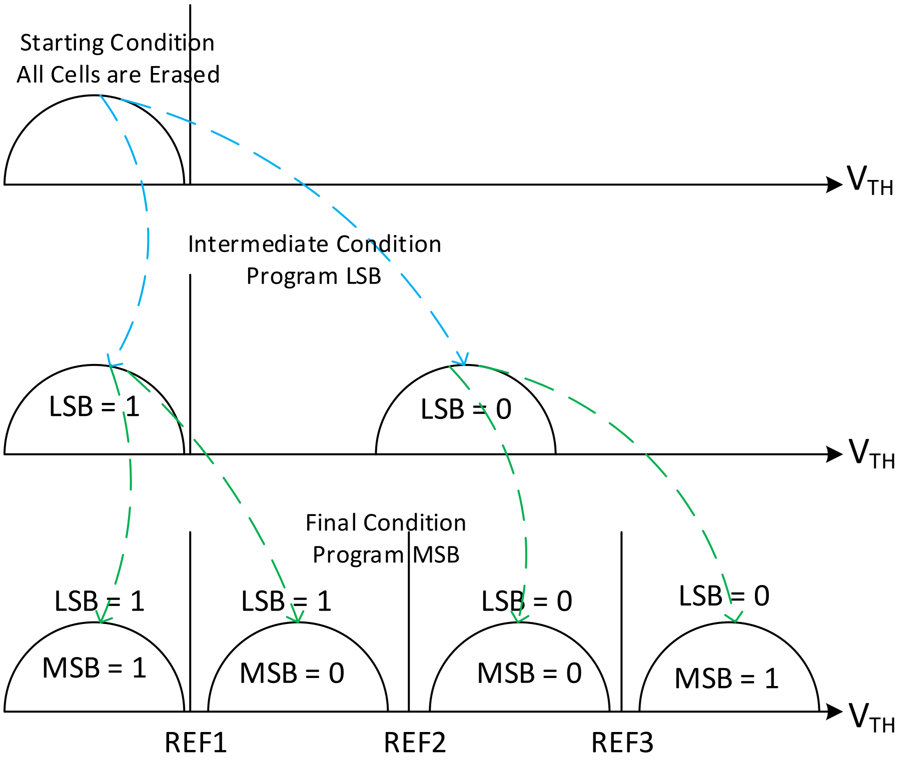


Figure 2.4. An example of program operation in 2-bit MLC flash memory.

2.3 Flash Memory Organization

Flash memories are generally organized as a two-dimensional matrix of flash memory cells. Based on the organization of the matrix structure, flash memories can be broadly classified into NAND and NOR flash memories. In Section 2.3.1, we discuss the details of NAND flash memories and in Section 2.3.2, we discuss the details of NOR flash memories.

2.3.1 NAND Flash Memories

A NAND flash memory consists of multiple *NAND Strings*, where a *NAND String* consists of a series of flash memory cells connected in a way similar to a series of NMOS transistors in a NAND logic gate. Figure 2.5(a) shows a NAND string − a serial connection of NAND flash memory cells where the source terminal of one flash memory cell is connected to the drain of another flash memory cell. We can have 32-64 flash memory cells in a string.

On either end of these strings are regular transistors called select gates. The select gates are used for proper biasing of the NAND string for different flash memory operations. The select gate on the drain end of a NAND string serves to connect or disconnect the string to the Bit Line. The select gate at the source end connects the string to the Source Line.

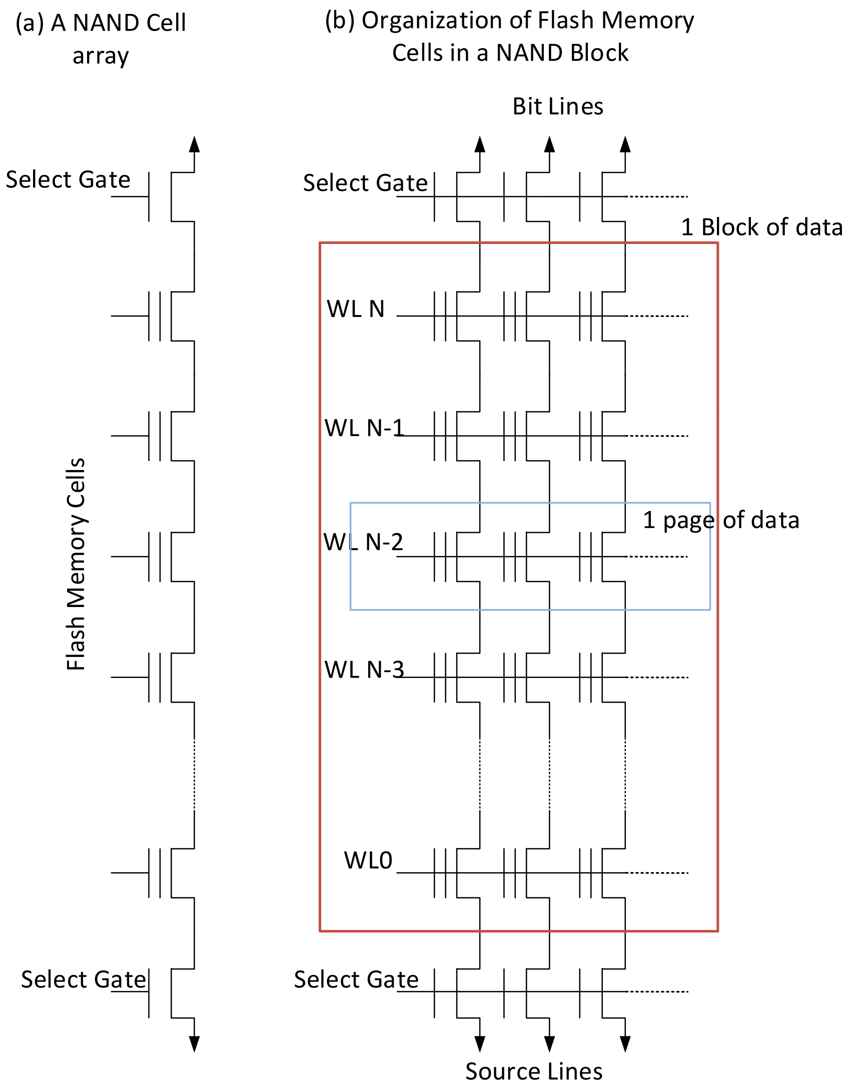


Figure 2.5 NAND flash memory array structure. (a) Arrangement of flash memory cells in an array forming NAND string (b) A complete storage element forming NAND flash memory.

Multiple NAND strings are connected in parallel to form a *Block* of NAND memory as highlighted by the red rectangle in Figure 2.5(b). Flash memory cells in the same row of NAND strings are connected via their CGs to a common line called *Wordline* (WL). The cells in the same row form a *Page* (highlighted by blue rectangle).

One page typically contains many cells, e.g., between 32 Kilobits to 128 Kilobits. Consequently, multiple such pages (e.g., 32 to 64) form a flash memory block, and a NAND array includes multiple such blocks.

Major operations performed in a flash memory array are erase, program, and read operations. The electrical details of each operation are discussed in Section 2.1.1. However, it is worthwhile to mention that the erase operations in a NAND memory are performed at a granularity of a single flash block, whereas a program or a read operation is performed at a granularity of a single page.

NAND flash memory allows for a compact organization of the flash memory cells. Because of this compact organization, high density storage products can be designed out of NAND flash memory. Thus, NAND flash memory is extensively used for bulk storage devices, like Solid State Drives (SSDs), Thumb Drives, Embedded Multimedia Cards (eMMCs), and Secure Digital (SD) Cards.

2.3.2 NOR Flash Memories

In a NOR flash memory, memory blocks consist of a two-dimensional matrix of flash memory cells. Flash memory cells in a vertical column share a common bitline (BL). The source terminals of all the flash memory cells in a block are connected together as shown in Figure 2.6(a). Control gates of all flash memory cells in the same row are connected to a shared wordline (WL). Figure 2.6(a) shows an example block of NOR flash memory that is composed of 64 16-bit words.

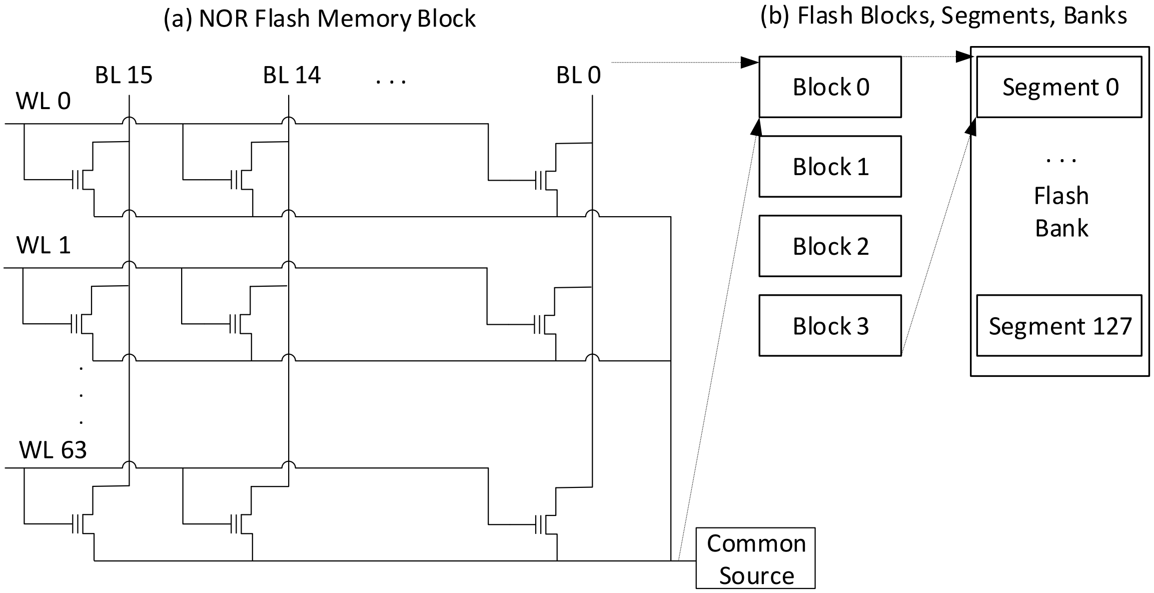


Figure 2.6 NOR flash memory architecture. (a) A block of NOR based flash memory (b) Organization of flash memory blocks, segments, and a bank.

Multiple such blocks are combined to form a segment and multiple segments are combined to form a bank of NOR flash memory as shown in Figure 2.6(b). In the example NOR flash memory presented in Figure 2.6, 4 blocks form a segment of a flash memory and 128 such segments form a bank. Multiple such banks can be present in a NOR flash memory.

An erase operation in a NOR flash memory spans a complete segment, whereas program and read operations are performed at a granularity of a single byte or a word. This is enabled by the parallel connection of the flash memory cells as discussed earlier. Since NOR flash memory allows for program and read operations at the word/byte level, they are used for storing software, boot codes, BIOS codes, etc. However, NOR flash cells are larger than their NAND counterparts because their source terminals are all connected together, Consequently, NOR flash memories have lower bit density and thus a higher cost than their NAND flash counterparts.

2.4 3D NAND Flash Memories

Discussion presented so far assumes two-dimensional (2D) flash memory. 2D NAND or planar flash memories reached scaling limits around 2015 with cells manufactured at 14 nm technology needs. Further scaling of flash memory cells is not possible due to significant reliability and endurance issues, caused by charge leakage or electrical charge migration from one cells into an adjacent cell, and cell-to-cell program interference [29]–[37]. 3D NAND flash memories offer a solution to these limitations. However, 3D flash memories are fundamentally different from planar flash memories in their cell structure and overall organization [38], [39]. In this section, we discuss a cell structure of a 3D NAND flash memory and its organization.

2.4.1 3D NAND Flash Memory Cell

Contrary to the FG-MOSFETs used in planar flash memory cells, 3D flash memory cells use floating gate as well as charge-trap memory cells. Moreover, instead of stacked structure of planar flash memory, 3D NAND flash memory cells are cylindrical in a gate-all-around structure. Figure 2.7 (a) shows a charge-trap (CT) flash memory cell. The CT layer sits radially inside the Control Gate (CG) and between the oxide layers. CG forms an outer cylinder for each flash memory cell. Charges are stored in the CT layer that is insulating in nature as opposed to conducting nature of floating gates in FG-MOSFETs. Source, Drain and Substrate are cylindrically inside the CT layer forming the center of the flash memory cell.

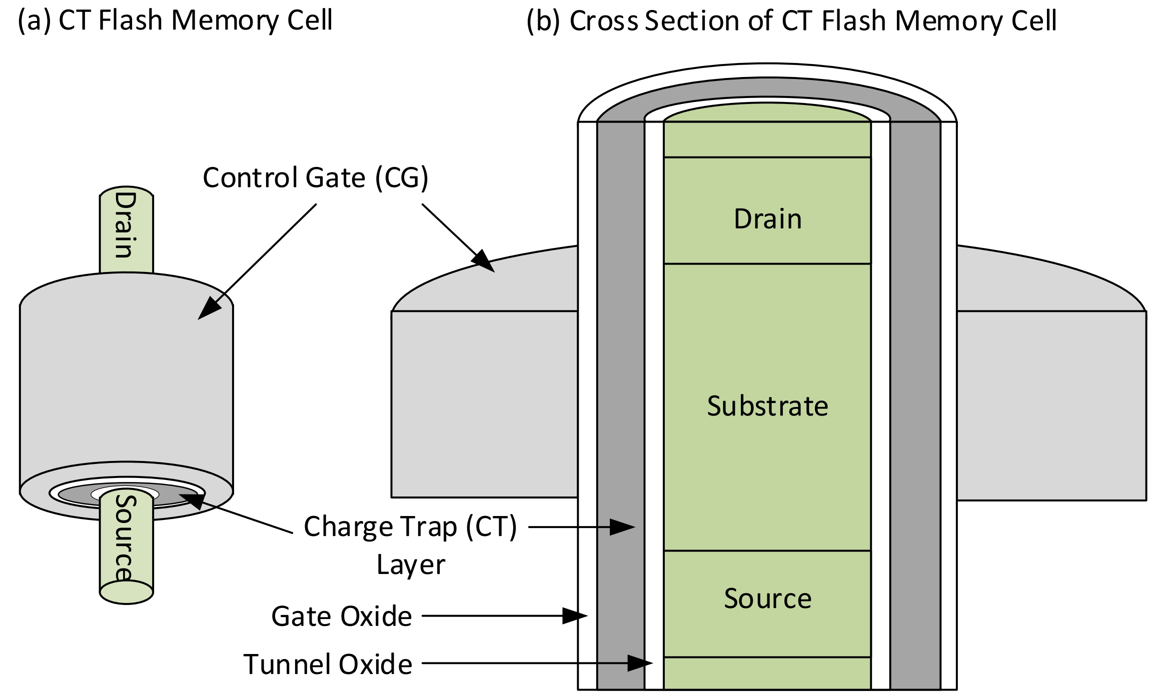


Figure 2.7. Charge Trap (CT) flash memory cell structure in 3d flash memory.

2.4.2 Organization of 3D NAND Flash Memory

Since 3D NAND flash memory requires vertical stacking, simple stacking of multiple layers of planar flash memory cells would require many critical masks to be replicated for each vertical layer during manufacturing process. This would increase the cost of fabrication [28].

Instead, 3D NAND flash memories stack multiple rectangular planes on top of each other. Each plane forms a Control Gate plane or CG plane as shown in Figure 2.8. Multiple holes are drilled into these planes vertically and are filled with oxide, charge trap, and polysilicon materials [39].

At each intersection of the pillar and the CG plane, a NAND memory cell is formed. A vertical 3D NAND string formed as a result is shown in Figure 2.8(a). The select transistors are then connected on top and bottom of these vertical structures which are then connected to Bit Line and Source Line, respectively.

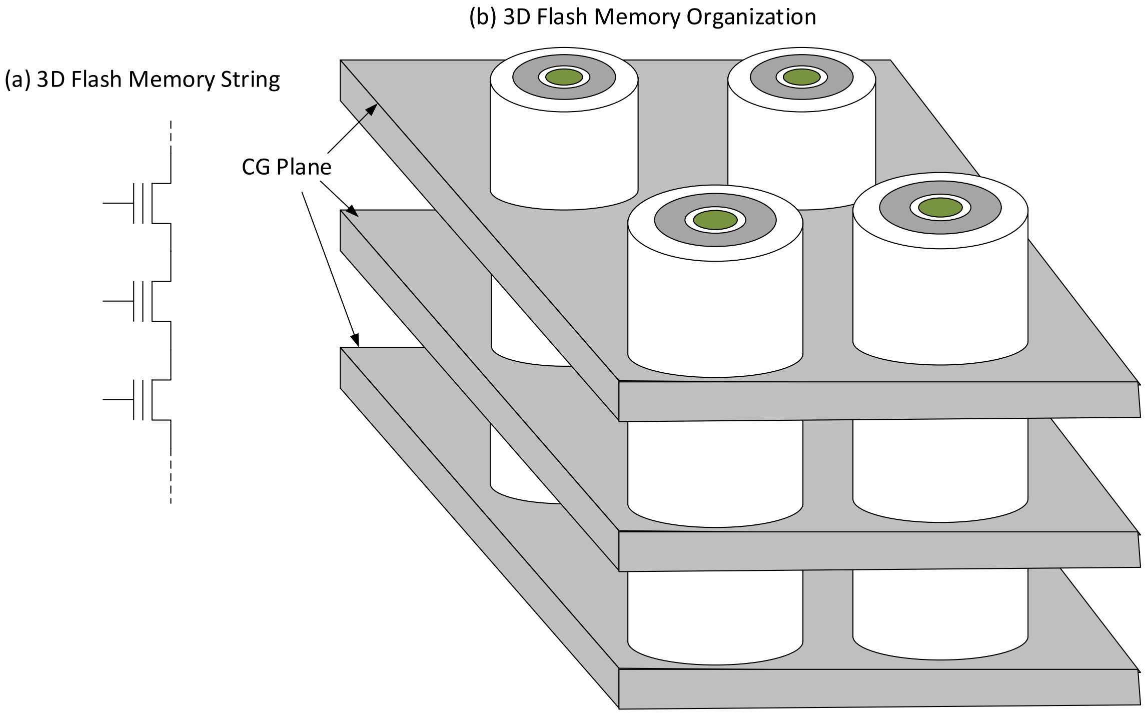


Figure 2.8. 3D NAND flash memory organization.

CHAPTER 3

EXPERIMENTAL ENVIRONMENT

Research presented in this dissertation uses split-gate NOR flash memory that is embedded into the MSP430 microcontrollers. We describe the MSP430's NOR flash memory in Section 3.1. Section 3.2 describes a framework and setup that is developed as a part of dissertation research and used for characterization of 3D NAND flash memories.

3.1 MSP430's Embedded NOR Flash Memory

Modern microcontrollers are targeting different low-power and low-cost applications. They are designed as Systems-on-Chip (SoCs), integrating a central processor core, flash memory, Random Access Memory (RAM), clock subsystem, and a variety of input/output peripherals on a single die.

NOR flash memories are used for storing programs and data in modern SoCs, including microcontrollers. Data from NOR flash memory can be read at the smallest granularity of a single byte. This allows for programs stored in NOR flash memory to be executed without copying them to RAM memory. This method of execution is called Execute-In-Place (XIP). The benefit of XIP along with higher error resilience of NOR flash memory has found a niche in low power microcontrollers and for storing BIOS codes and firmware in high-end computer platforms.

The internal cell structure of the MSP430's embedded NOR flash memory corresponds to the split-gate flash memory cell discussed in Section 2.1.2. The internal organization of the MSP430's NOR flash memory corresponds to the one presented in Section 2.3.2. In this section, we dive into programmer's view of the MSP430's NOR flash memory.

3.1.1 Flash Memory Module

The MSP430's embedded NOR flash memory module includes a flash controller and a NOR flash array. The array is composed of multiple banks of flash memory and can be accessed through address bus and data bus. In normal operating mode MSP430's flash memory behaves as a Read Only Memory (ROM) allowing programs to read instructions and data. However, the flash memory controller allows for in-system programming (ISP) of the flash memory using a timing generator and a voltage generator as shown in Figure 3.1(a) [40]. Since programming and erasing the flash memory involves bringing up high voltages required for biasing different terminals of the flash memory cell, the voltage generator and timing generator play a crucial role in ensuring proper flash memory operations. The voltage generator is responsible for binging up the required voltage levels for program and erase operations while the timing generator is responsible for controlling the duration of the operations.

Figure 3.1(b) and Figure 3.1(c) show the timing diagrams for program and erase operations, respectively. Both the program and erase operations have an initial phase in which the voltage generator brings up the voltages to the required levels that are maintained during the flash program and erase operations. Upon completion of the flash operation, the voltage generators are turned off and the flash array is back in the read-only mode.

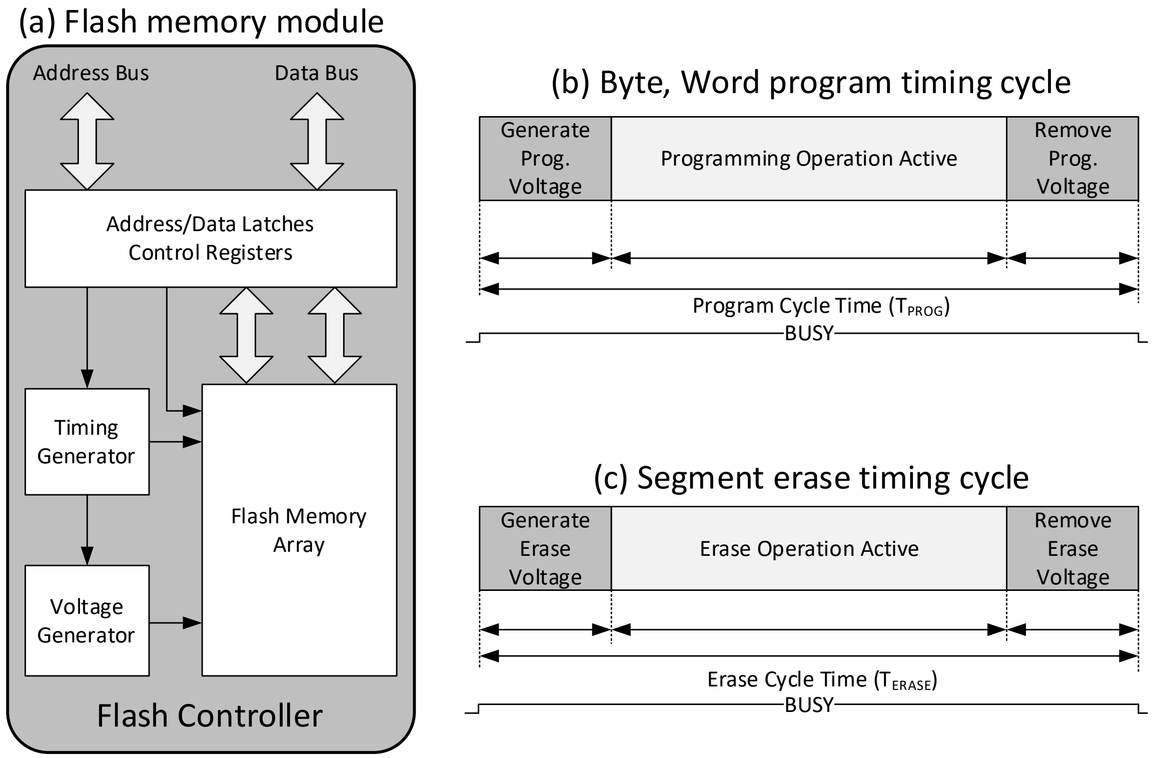


Figure 3.1 (a) Block diagram of flash memory module. (b) Program cycle timing diagram. (c) Erase cycle timing diagram

The controller supports flash program operations at the byte level, word level (2 bytes), double-word level (4 bytes), or a block level (64 bytes). Program operations in a segment must be preceded by a segment erase operation. Thus, the granularity of erase operations is a single segment of 512 bytes. Bank erase and mass erase operations are also supported. A bank erase operation erases an entire flash memory bank, whereas a mass erase operation erases the entire flash memory (all banks, if multiple banks are present). During flash erase and program operations the flash controller halts the processor as both programs and data reside in the flash memory.

3.1.2 Erase and Program Operations

The erase and program operations involve configuring the flash controller. The flash controller is interfaced through a set of control registers (FCTLx). To configure the flash module for erase and write operations, certain bits in the control registers FCTLx are set. For an erase operation, after the control registers are configured, a dummy write to any address in a segment will initiate an erase operation on the selected segment as shown in Figure 3.2(a). However, a program operation is performed for a single word as shown in Figure 3.2(b) [41].

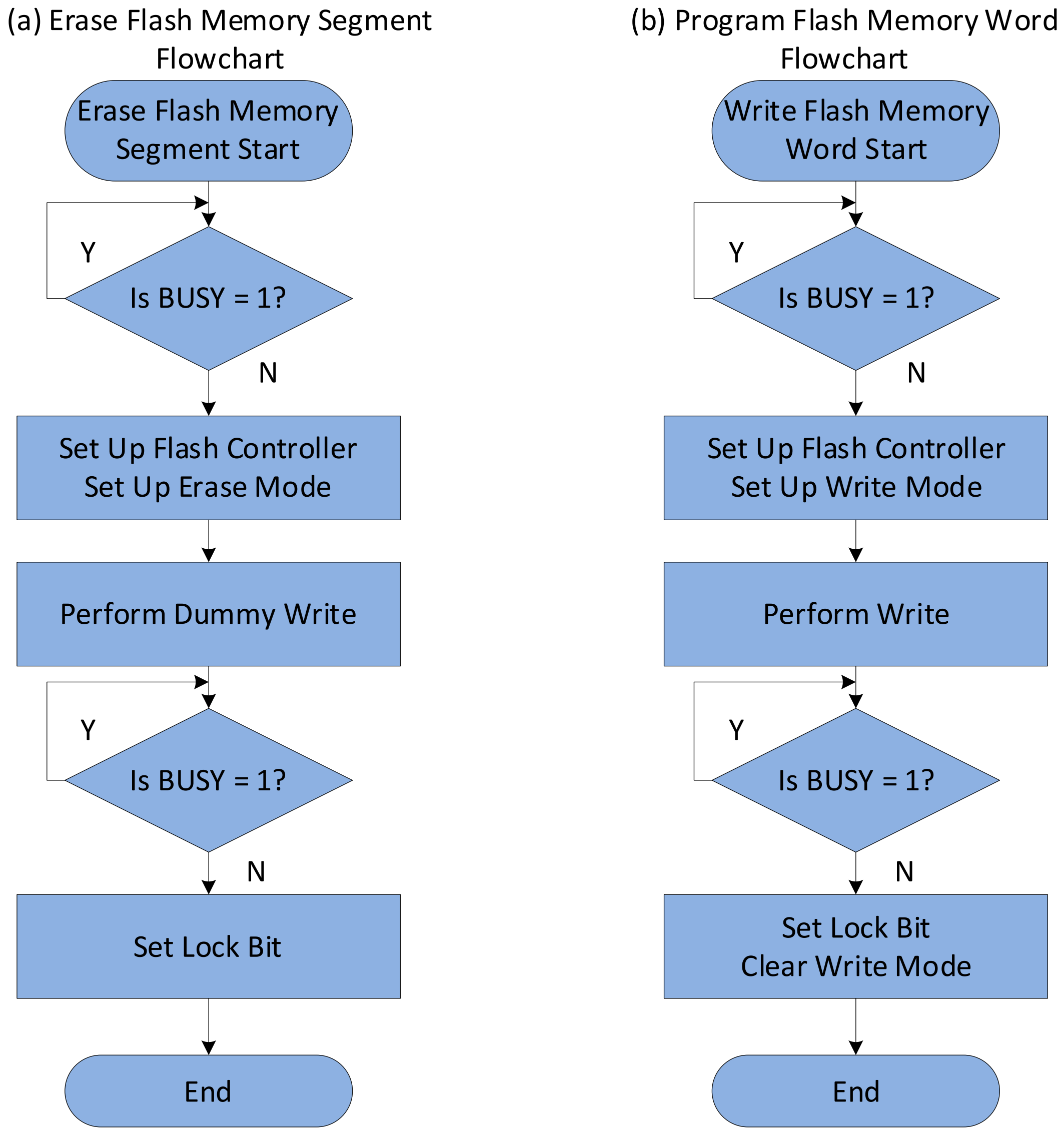


Figure 3.2. (a) Flowchart for flash memory segment erase. (b) Flowchart for flash memory word program.

Code 3.1 outlines a subroutine that is used to program a flash memory word. To program an entire segment, the word program operation must be performed repeatedly in a loop (Figure 3.2(b)). The completion of the flash memory operation can be confirmed by checking the BUSY flag. Once the BUSY flag is reset, LOCK bit must be set to ensure no accidental changes to flash memory are made.

```
1. void program_word(SegmentAddress, WordIdx, WordData){
2.   FCTL3 = FWPW;                            // flash password
3.   FCTL1 = FWPW+WRT;                        // configure word write
4.   *(SegmentAddress +WordIdx) = WordData; // write data to flash word
5.   while(FCTL3&BUSY);                       // wait until busy
6.   FCTL1 = FWPW;                            // clear erase operation
7.   FCTL3 = FWPW+LOCK;                       // lock the flash memory
8. }
```

Code 3.1. Subroutine for program operation of a flash memory word.

Flash memory controllers support long word write and block write operations to accelerate the program operation. Long word write is similar to word write operation (Code 3.1). Instead of writing a single word into flash memory (line 4 of Code 3.1), successive lines of codes can be used to write into two consecutive words of flash memory for a long word write. Code 3.2 outlines a subroutine that programs a block (128-bytes or 64-words) of flash memory where *BlockIdx* is the index of block to be programmed inside the flash memory segment *SegmentAddress*. *BlockData* is an array of data to be written into the block.

```
void program_block(SegmentAddress, BlockIdx, BlockData){
  while(FCTL3&BUSY);                          // wait until busy
  FCTL3 = FWPW;                               // flash password
  FCTL1 = FWPW+BLKWRT+WRT;                    // configure block write
  FlashAddress = SegmentAddress+BlockIdx*64;  // get starting address
  for(uint8_t i=0;i<32;i++){
    *(FlashAddress+2*i) = BlockData[2*i]      // write long word into
    *(FlashAddress+2*i+1) = BlockData[2*i+1]; // ..flash memory
    while(!(FCTL3&WAIT));                     // test WAIT bit
  }
  while(FCTL3&BUSY);                          // wait until busy
  FCTL1 = FWPW;                               // clear erase operation
  FCTL3 = FWPW+LOCK;                          // lock the flash memory
}
```

Code 3.2. Subroutine for program operation of a flash memory block.

3.1.3 Partial Flash Memory Operations

Instructions of a program fetched and executed by the CPU generally reside in the flash memory. We call such flash memories Execute-In-Place (XIP) memories.

Temporary variables held in the heap and program stacks are however created in the RAM memory.

When the processor initiates a flash segment erase or a word program operation (by performing a write operation into the specified address), the flash controller takes over and stalls the CPU. The CPU remains stalled until the flash memory operation is complete. However, flash erase and program routines can be copied into the RAM memory, and the program counter can be redirected to continue execution from the RAM memory. This way, the processor can continue to execute instructions while the flash controller is still busy performing a flash operation.

The MSP430 flash memory controller provides an emergency exit (EMEX) control bit through one of its control registers. Setting this bit programmatically initiates termination of an ongoing flash memory operation. Consequently, the CPU can initiate a flash erase or program operation from the RAM memory, wait for a certain period of time, and then set the EMEX bit to terminate the ongoing operation. This mechanism is used to perform partial flash memory operations. Figure 3.3 illustrates the timing diagram in case of a partial flash operation. Every flash memory erase or program operation includes three distinct parts: generating high voltage needed for the operation, operation itself, and lowering the voltages and transitioning to a read-only mode. By setting the emergency exit, the current operation is aborted abruptly, voltages lowered, and the flash memory transitions to normal read-only mode.

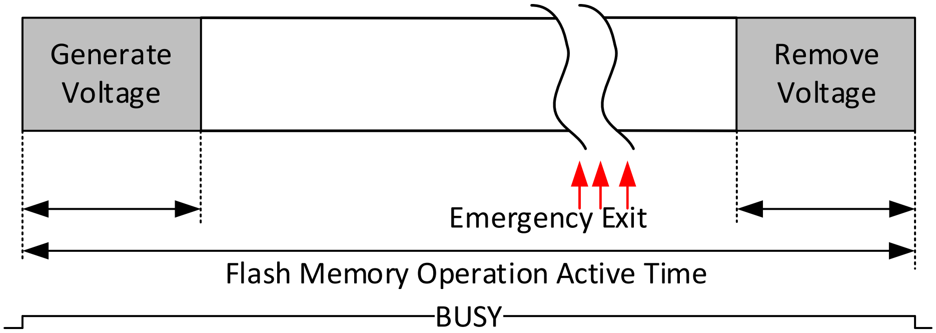


Figure 3.3. Emergency exit of flash memory operation.

Code 3.3 outlines a subroutine for partial flash erase operation. This subroutine is run from RAM, ensuring that the CPU remains active while the erase operation is being performed. The subroutine starts by checking a BUSY bit to see if the flash memory controller is busy (line 2). If it is not busy, the flash memory controller is configured for a segment erase operation using ERASE bit (line 4). A dummy write operation is performed at a flash memory address belonging to a segment which needs to be erased (line 5). This write initiates the segment erase operation. Since this subroutine is being run from RAM, the CPU is still active. The CPU can be made to voluntarily stall for PE_TIME clock cycles (line 6). EMEX can be issued after the wait time to prematurely terminate the flash memory operation (line 7). A similar sequence of steps is carried out for flash memory program operations to implement a partial flash program operation.

```
1.  void partial_erase_segment(FlashAddress, PE_TIME){
2.    while(FCTL3&BUSY);        // wait until busy
3.    FCTL3 = FWPW;             // flash password
4.    FCTL1 = FWPW+ERASE;       // configure segment erase
5.    *FlashAddress = 0;        // dummy write to initiate erase operation
6.    __delay_cycles(PE_TIME);  // wait for PE_TIME cycles
7.    FCTL3 = FWPW+EMEX;        // emergency exit to terminate erase operation
8.    while(FCTL3&BUSY);        // wait until busy
9.    FCTL1 = FWPW;             // clear erase operation
10.   FCTL3 = FWPW+LOCK;        // lock the flash memory
11. }
```

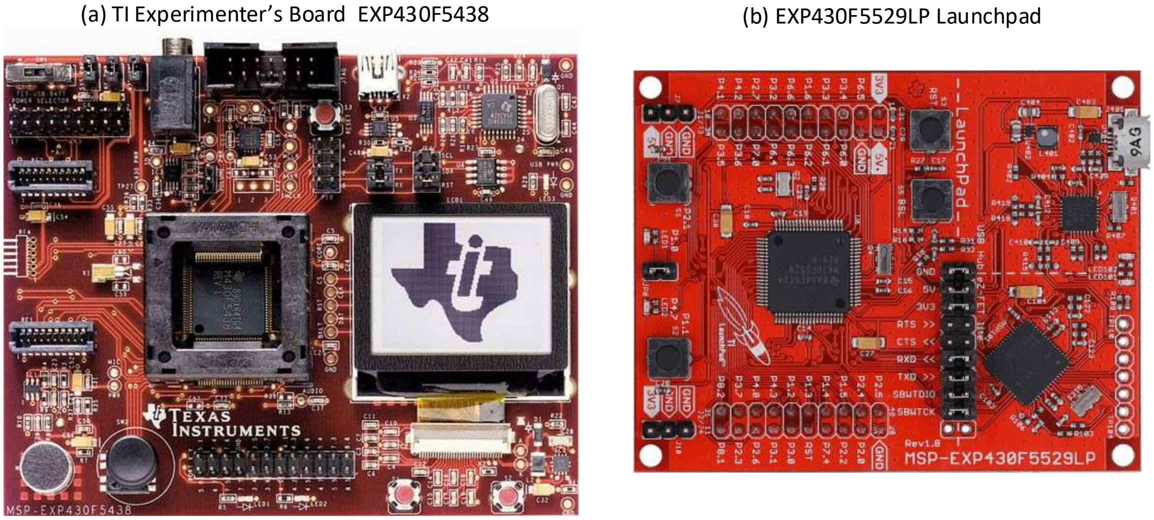
Code 3.3. Subroutine for partial erase operation of a flash memory segment.

Partial flash memory operations are a useful tool that allow us to gain insights into transitions of flash cells from programmed to erased or erased to programmed states as a function of time. By modulating the PE_TIME (or PP_TIME) and reading data after partial flash operations, we can extract physical properties of flash memory cells, e.g., voltage threshold distributions and others. We will exploit these partial flash memory operations to create fingerprints of the flash memories.

3.1.4 Experimental Setup

The NOR flash memory based experiments are performed on two families of MSP430 microcontrollers, namely, MSP430F5438 and MSP430F5529. Research-specific setups and experiment flows are described in later sections.

The TI EXP430F5438 Experimenter's board, shown in Figure 3.4(a), is used in experiments utilizing MSP430F5438 microcontroller [42]. This evaluation board features a 100-pin drop-in socket that can hold one MSP430F5438 chip. The TI EXP430F5529LP Launchpad platform, shown in Figure 3.4(b), is used in experiments utilizing MSP430F5529 microcontrollers [43]. The Code Composer Studio (CCS) Integrated Development Environment (IDE) on aWindows 10 machine is used to interface the microcontrollers in both the cases.


(a) TI Experimenter's Board EXP430F5438

(b) EXP430F5529LP Launchpad

Figure 3.4. Experimental setup for MSP430 microcontrollers. (a). TI Experimenter's Board MSP-EXP430F5438. (b). TI Launchpad Platform MSP-EXP430F5529LP.

3.2 NAND Flash Memory Setup

3.2.1 ONFI Specification

NAND flash memory chips use a standardized low-level interface that is developed by a working group of flash memory manufacturers. This standard for interfacing Common Off-the-Shelf (COTS) flash memory chips, named Open NAND Flash Interface (ONFI) specifies physical interfaces; chip identification mechanisms; command set for reading, erasing and programming NAND flash; timing requirements; and data integrity features. Typically, a host, a main processor of a computer system, a specialized controller, or a specialized storage processor, is responsible to implement this interface.

The flash memory chips may include one or more dies packed into a single package. A die includes an on-die controller and a NAND array as shown in Figure 3.5. The on-die controller consists of a set of control, address, and data registers that are used to latch commands, addresses, and data. The on-die controller is responsible to accept commands, addresses, and data, and generate a sequence of control signals to carry out ONFI commands.

The physical interface of NAND flash memory chips defined by the ONFI specification includes TSOP-48, WSOP-48, LGA-52, BGA-63, BGA-100, BGA-152, BGA-132, BGA-272, BGA-252 and BGA-316 [44]. Regardless of the physical interface of the chip, the specification defines a set of control signals and data signals a host device uses to interface NAND flash memory chips. For example, an 8-bit asynchronous data interface assumes 8 data lines that are denoted as DQ0 to DQ7. The control signals include Chip Enable (CE#), Command Latch Enable (CLE), Address Latch Enable (ALE), Read Enable (RE#), Write Enable (WE#), Ready/Busy (R/B#) and Write Protect (WP#). The suffix "#" indicates that a control signal is active low.

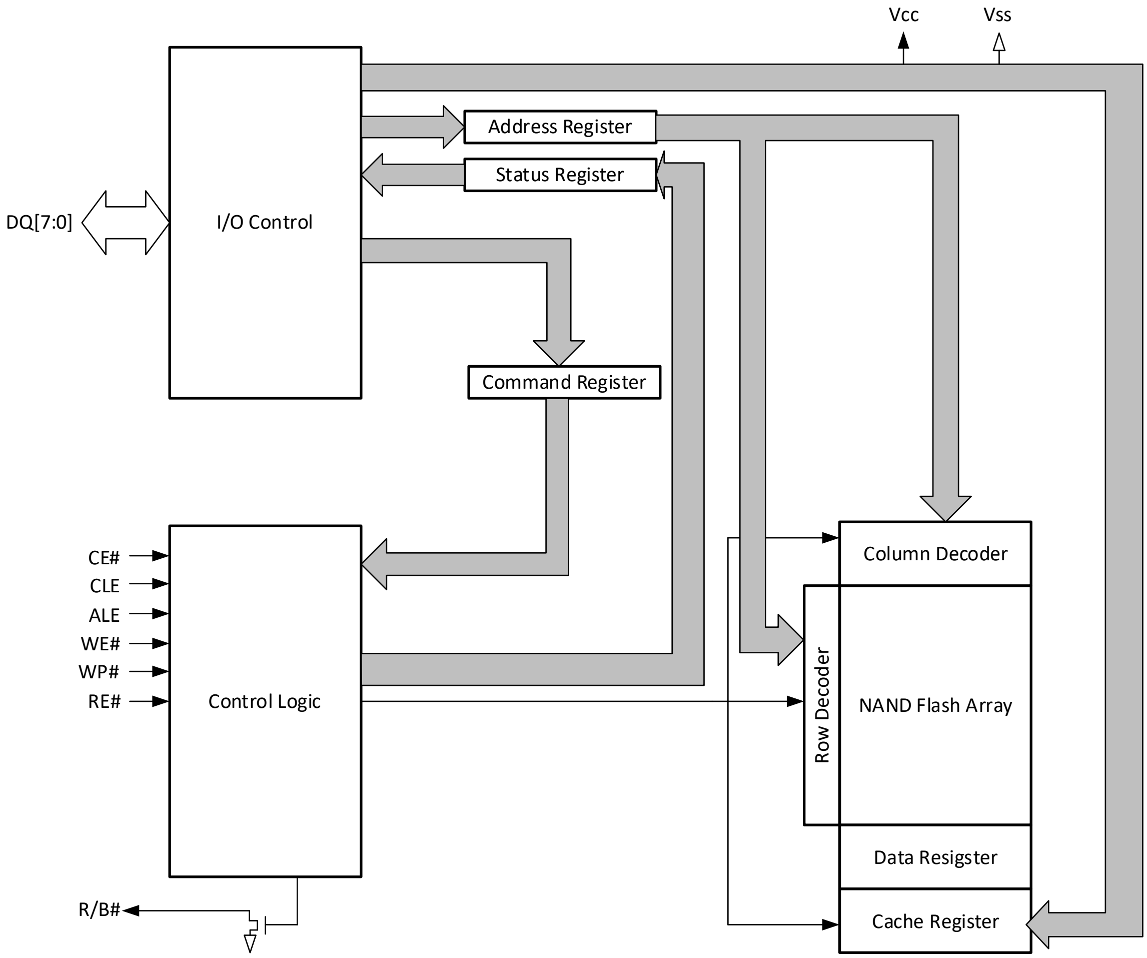
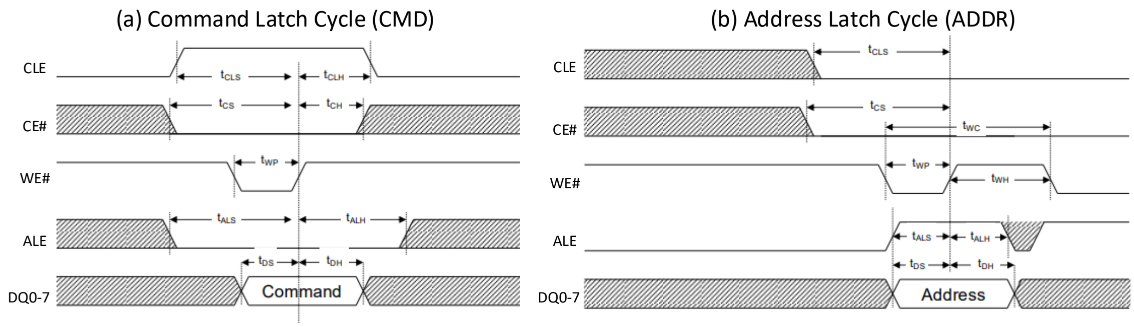


Figure 3.5. Block diagram of a NAND flash memory die.

The data lines DQ0-DQ7 are used to transfer commands, memory addresses, and data to the NAND chip, as well as to read out the data from the NAND chip. When the ALE signal is set at logic ‘1’, the host indicates that the data lines are carrying a portion of the address inside the NAND array. Similarly, a logic ‘1’ on the control line CLE, indicates that the data lines are carrying an ONFI command that will be executed by the NAND flash memory.

The ONFI specification defines different modes of interfacing the NAND flash memory chips. Asynchronous mode is the simplest mode. while faster modes of interfacing that offers double data rates (DDR) are also available. To keep our discussion simple, let us discuss asynchronous mode of interfacing. Figure 3.6 shows a command latch cycle (Figure 3.6(a)) and an address latch cycle (Figure 3.6(b)) for asynchronous mode. The value placed on data lines DQ0-DQ7 is written into the command register of the device on the rising edge of WE# when CLE is high, ALE is low and RE# is high (Figure 3.6(a)). Similarly, the value placed on DQ0-DQ7 is written into the address register of the device on the rising edge of WE# when ALE is high, CLE is low and CE# is low (Figure 3.6(b)).


(a) Command Latch Cycle (CMD)

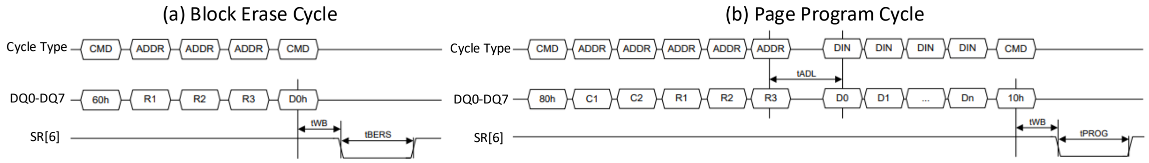
(b) Address Latch Cycle (ADDR)

Figure 3.6. Asynchronous mode timing cycles. (a). Command latch cycle. (b). Address latch cycle.

A data cycle is indicated by a low value on both ALE and CLE lines. Depending on the value of write enable (WE#) line and read enable (RE#) line, data is either input (DIN) to the internal data register of the NAND flash memory device from data lines DQ0-DQ7 or data is output (DOUT) from the internal data register to the data lines DQ0-DQ7. The value placed on DQ0-DQ7 is read into the data register of the NAND device at every rising edge of WE# when both ALE and CLE are low while RE# is high. Similarly, the value from the data register of the NAND device is placed into the data lines DQ0-DQ7 at every falling edge of RE# when both ALE and CLE are low while WE# is high.

Let us see an example of an erase operation defined by the ONFI specification. For an erase operation, the host should send a command corresponding to the erase operation and the address of the block to be erased. Figure 3.7(a) shows the sequence of command and address cycles required for a block erase operation. The host initiates a command cycle (CMD) with value 0x60 in the data lines followed by three address cycles (ADDR) that send the row address or the address of the block to be erased. This is followed by the command 0xD0, which initiates the erase operation. During the erase operation, the NAND device goes into a busy state. This is indicated by the R/B# pin being low. Alternately, this is also indicated in status register (SR) of the NAND flash device. The block erase time can be determined by polling the state of R/B# line or reading the status register.

The sequence of commands and address cycles for a page program operation is shown in Figure 3.7(b). In addition to the command and address cycles, multiple data input cycles (DIN) are also needed in page program operation. Sending a command 0x80 indicates a program operation. This should be followed by 5 address cycles. The address indicates the row and column addresses of the page to be programmed. The five address cycles are followed by the data to be written into the flash memory page. This data is placed in the data register of the flash memory device. A command 0x10 will initiate the write operation of the data into the address specified. The flash device goes into the busy state until the program operation is active.



(a) Block Erase Cycle

(b) Page Program Cycle

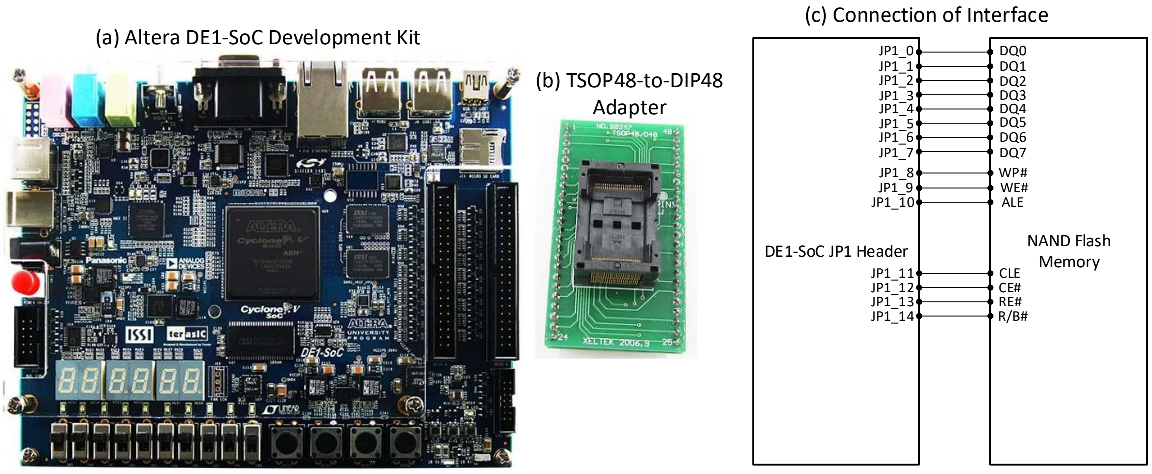
Figure 3.7. Flash memory operation cycle. (a) Block erase cycle. (b). Page program cycle.

3.2.2 FPGA Based ONFI Interface

To interface the COTS NAND flash memory chips, we developed an FPGA based ONFI interface. This interface implements asynchronous mode of interfacing NAND flash memory chips. An ARM based computer that is implemented in Altera DE1-SoC Development Kit (Figure 3.8(a)) acts as the host machine. The ONFI commands to interface the NAND flash memories are implemented in C/C++ [45].

Altera DE1-SoC integrates an ARM based Hard Processor System (HPS) that interfaces input/output device such as push buttons, slider switches, seven segment displays, LEDs, Audio CODEC, VGA control, multiple HPS Timers, and others. It also consists of two jumper extensions that allows interfacing any external peripherals.

Our design makes use of the jumper extension (JP1) in DE1-SoC to connect to external NAND adapter. The NAND flash memory adapter can vary depending upon the physical interface of the chip. An example of adapter to interface a TSOP-48 NAND flash chip is shown in Figure 3.8(b). This adapter provides DIP-48 connection for a TSOP-48 chip, which is then interfaced to the JP1 extension using jumper cables. The connection of different data lines (DQ0-DQ7) and control lines to the different pins of the JP1 extension is shown in Figure 3.8(c).



(a) Altera DE1-SoC Development Kit

(b) TSOP48-to-DIP48 Adapter

(c) Connection of Interface

Figure 3.8. NAND flash memory interfacing setup. (a). Altera DE1-SoC development kit running an ARM computer. (b). TSOP48-to-DIP48 adapter to house ONFI TSOP48 nand flash memory chips. (c). Connection of interface between the DE1-SoC platform and NAND memory.

CHAPTER 4

MICROCONTROLLER FINGERPRINTS

A device fingerprint is a unique bit vector extracted from the physical properties of the electronic device. Such a fingerprint can be used for identification purposes through differentiation between functionally identical devices. In our work, we extract fingerprints from an embedded NOR flash memory for identification or authentication purposes. We leverage partial erase operation in flash memory to expose the inherent manufacturer variations among the flash memory cells and export them though digital interface as a bit vector [46]. The exposed physical property, i.e., differences in threshold voltage distributions, is inherent to each specific device and specific memory cells and cannot be cloned [47].

4.1 Fingerprint-based Device Authentication

Let us consider a concept of fingerprint-based device authentication. We distinguish two actors: a *Manufacturer* and a *Customer*. The manufacturer is a producer of electronic components or a vendor relying on device-fingerprint based authentication to distinguish between legitimate or illegitimate users. The customer can be any user or a component that is requesting a service from the manufacturer. A fundamental service is verification of device authenticity, triggered by customers in various scenarios – e.g., to ensure the use of genuine chips before they are integrated into a larger system or to access firmware updates, or to report sensor readings.

Figure 4.1 shows a system view of the fingerprint-based device authentication. It consists of two phases, namely *Enrollment Phase* and *Authentication Phase*. The enrollment phase is conducted entirely at the manufacturer's premise before shipping the device. In this phase, the manufacturer characterizes every device and extracts and enrolls a single or multiple fingerprints. These fingerprints are unique bit-vectors derived from a chosen physical property. The fingerprints that are generated during the enrollment phase are called *Enrollment Fingerprints* (EFs). The manufacturer stores these fingerprints in a secure database in the manufacturer's premise.

The authentication phase is performed partly at the manufacturer's site and partly customer's site. It starts by a customer placing an authentication request to the manufacturer. The manufacturer sends a challenge to the customer, e.g., the address of a memory segment where a fingerprint is extracted from during enrollment. Customer, then, undergoes the steps to extract a fingerprint. The fingerprint that is extracted during this authentication phase is called *Authentication Fingerprint* (AF). Once an AF is extracted, the customer sends it to the manufacturer as a response. The manufacturer compares the received AF with the EFs in the database. Based on the matching criteria, the manufacturer generates a response to the customer and decides whether to confirm or deny the authenticity of the device. The metrics that are generally used for comparing the fingerprints are Hamming Distance [15], [18], [48]–[52], Jaccard Index [53], or Correlation [47], [54], [55].

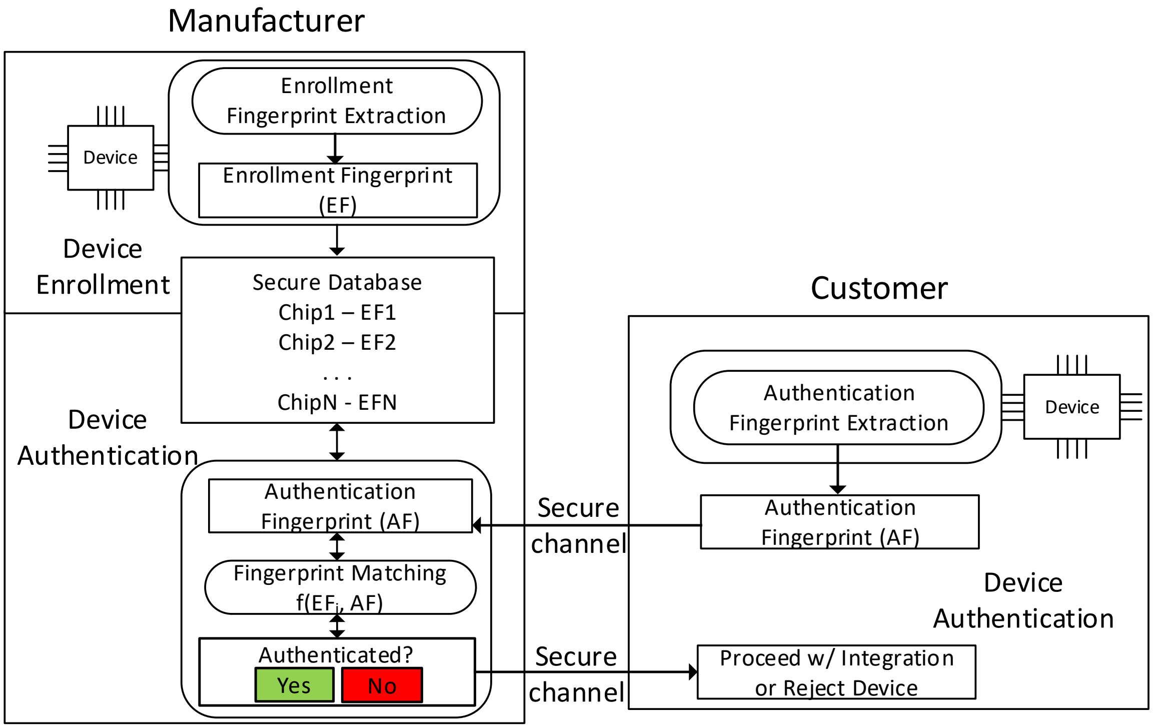


Figure 4.1. System view of a fingerprint-based authentication system.

4.2 Related Work and Motivation

The problem of identifying and authenticating an electronic device requires each device to have an identifier. This identifier must be unique to each device. Thus, one of the straightforward approaches to this problem is to store a unique identifier in a dedicated memory location. This identifier is assigned by the manufacturer and stored in dedicated non-volatile memory, like EPROM, EEPROM, flash memory, programmable fuse or in battery backed RAM.

The approach of assigning an identifier and storing the identifier in a secure memory location is reliable and is stable over a long period of time. However, such identifiers can easily be cloned by adversaries. In addition, when identifiers are stored in dedicated non-volatile memories, extra mask layers are required during manufacturing. Storing identifiers in battery backed RAM creates a need for battery that powers the RAM at all times. All these approaches increase the manufacturing cost.

Physical Unclonable Functions (PUFs) and Physical Fingerprinting are the solutions that do not require storage of the identifiers. In these solutions, the identifiers are extracted from a known physical property of the device. The physical property is usually caused by inherent manufacturing and process variations. Identifiers derived in this way have a property that no two physical devices share the same identifier, while staying fairly stable for a single device. Since the identifiers are based on the inherent variations, they cannot be cloned by an adversary even with physical access to devices.

PUFs use the inherent process variation as a key in mapping input challenges to responses (CR- pairs) [13]. The concept of PUF implementation in conventional integrated circuits (ICs) was first introduced by Gassend et al. in 2002 [16] for building secure smartcards. Their proposal tests several implementations based on self-oscillating loop with a non-monotonic delay circuit on FPGAs. Since then, several proposals have been made using different physical properties to derive PUFs from a circuit. Lee et al. [17] fabricated an arbiter based PUF that exploits inherent delay characteristics of wires and transistors in ICs. Suh and Devadas [15] demonstrate ring oscillator based PUFs for device authentication and secret key generation.

The proposals discussed above mitigate the problem of clonability that existed in the stored identifier-based approach. However, all these proposals make use of custom circuitry for PUF generation. Addition of a dedicated circuitry for generating

PUFs increases on-chip area and manufacturing cost. Thus, several alternative approaches to extract PUFs or fingerprints from a component that already exists in the IC are proposed.

Holcomb et al. [18] demonstrate extraction of identifying fingerprints from power-up state of embedded SRAM cells in microcontrollers and discrete SRAM modules. Liu et al. [56] propose data remanence based PUF extraction from SRAM memory cells by momentarily shutting down the power. Bacha et al. [57] use on-chip error correction logic built into higher-end processor cache for generation of PUF. Similarly, several PUFs or fingerprint extraction techniques based on DRAM memory are also proposed. These proposals are based on power-up states of DRAM [54], latency variations [53], write failures [48], refresh pausing [58] and decay [59].

Multiple recent proposals target flash memory to use manufacturer variation among flash memory cells for generation of fingerprints. Wang et al. [47], [55] explore commercial-off-the-shelf NAND flash memory for generating unique fingerprints. Their proposal makes use of repeated partial program operation to expose the latency variation among flash memory cells to attain programmed state. Jia et al. [60] propose partial erase and program disturbed based technique to generate keys from NAND flash memories in addition to partial program based technique. Sakib et al. [51] present an aging resistant PUF extraction technique from NAND flash memory. In case of NOR flash memory, Clark et al. [61] utilizes partial erase operation at a reduced supply voltage to expose erase speed variability of 1.5-transistor flash memory cells. Nguyen et al. [62] use repeated erase suspend operation for true random number generation and fingerprint extraction from NOR flash memory that is similar to that of

Clark et al. Mandadi [49] relies on repeated partial program operation for generation of PUF based on NOR flash memory.

All these proposals indicate a strong interest for and represent an advancement in the field of device identification/authentication. However, most of the proposals for extraction of fingerprints or PUFs from memory rely on privileged operations like powering cycles, reset operations, lowering voltages, or changing controller parameters. In addition, proposals for flash memory require cost prohibitive computational or storage resources that exceed the capacity of low-end microcontrollers that are widely used in IoT applications.

Our proposal of microcontroller fingerprinting extracts fingerprints from an embedded NOR flash memory in low-power microcontrollers. The proposed technique utilizes a partial erase operation, where a segment erase operation is prematurely aborted to expose threshold voltage variations among flash memory cells. The NOR flash memories in modern microcontrollers are often in-system programmable that can be erased and programmed from software. This makes our approach applicable through software without any privileged operations. The proposed technique does not rely on any error detection or correction algorithms, thus minimizing compute requirements.

4.3 Partial Erase Operation in MSP430 Flash Memory

Since our fingerprint extraction algorithm is based on a partial erase operation, this section is dedicated to characterizing flash memory cells based on partial erase operation. A partial erase operation implementation is described in previous section, Code 3.3. A selected segment of the NOR flash memory is preconditioned by programming all flash memory cells, i.e., all bits in a segment are cleared (logic ‘0’). Then, the flash memory controller is configured for an erase operation through FCTLx control registers. Next, the erase operation is initiated by performing a dummy write to a memory location in the flash segment that is to be erased. Finally, after a certain period of time, the EMEX signal is issued so that the flash memory operation is terminated.

Figure 4.2 shows the state of partially erased flash memory segment. The content of the segment is determined by the duration of the partial erase, TPERASE. A too short TPERASE results in no changes in the state of flash memory cells. A too long TPERASE results in all flash cells being fully erased (read as a logic ‘1’). However, TPERASE can be chosen in such a way that some flash cells transition to the erased state (fast to be erased) and others remain in the programmed state (slow to be erased), as illustrated in Figure 4.2 (the partially erased segment state).

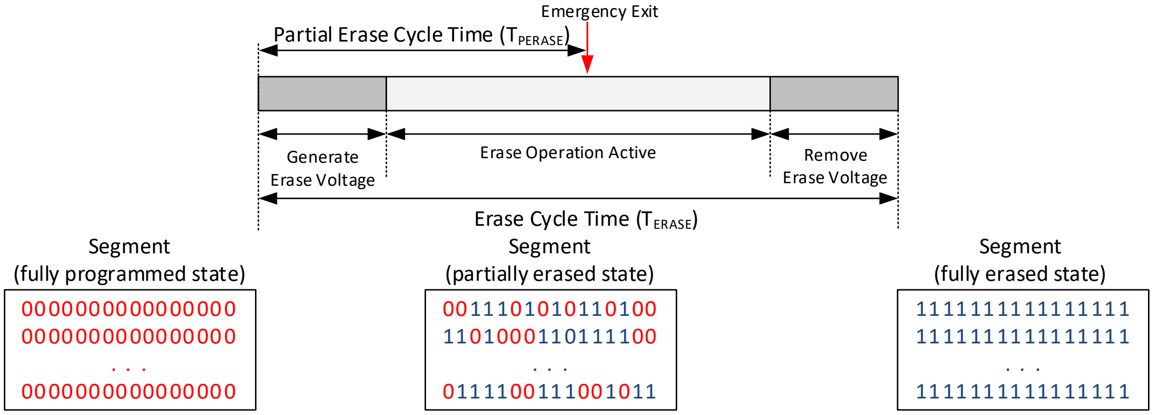


Figure 4.2. NOR flash segment state as a function of erase time.

To characterize flash memory cells in a segment through partial erase operations, a set of experiments as illustrated in Code 4.1 is performed. First, the segment is fully erased (line 4), then the segment is programmed (line 5). Before performing each partial erase operation, the flash segment is programmed so that the following partial erase operation is performed on flash segment with all cells in the programmed state, i.e., every byte reads as 0x00. Then, a partial erase operation is performed (line 6) with a partial erase time of T_{PERASE}. After the partial erase operation, the flash cells in the segment are read for a certain number of times (N_R times). Read operation of the flash segment is performed for a parameterized number N_R times to determine if the flash cells are stable or unstable.

After the read operation is performed (line 8 to 13), all the flash memory cells are characterized into three categories: *Stabel1s*, *Stable0s* and *UnstableBits*. The *Stable1s* are the flash memory cells that are read as a logic 1 (erased) N_R times. Similarly, *Stable0s* are the flash memory cells that are read as a logic 0 (programmed) N_R times. *UnstableBits* are those that change their state at least once for N_R reads.

After the characterization is performed, the partial erase time T_{PERASE} is increased by a certain period (Δt), and the process of erasing, programming, partial programming, reading, and characterizing is repeated. This process is continued until the partial erase time is equal to the nominal erase time of the segment (T_{ERASE}).

```
void partial_erase_characterize(flash_address){
  TPERASE = 0;
  do{
    fully_erase_segment(flash_address);            // Write 1s
    fully_program_segment(flash_address);          // Write 0s
    partial_erase_segment(flash_address,TPERASE);  // Partial erase
    FSAND=11..1b; FSOR=00..0b;
    for(i=0; i<NR; i++){                           // NR segment reads
      FR = ReadEntireSegment(flash_address);
      FSAND &= FR;
      FSOR  |= FR;
      SWDelay();
    }
    Stable1s = FSAND & FSOR;
    Stable0s = FSAND | FSOR;
    UnstableBits = FSAND ^ FSOR;
    Ratio = Count(Stable1s)/Count(Stable0s);
    TPERASE += Δt;}                                 // Increase TPERASE
  while (TPERASE <= TERASE);
}
```

Code 4.1. Characterization of a NOR flash segment using partial erase operations.

The results obtained from the characterization are shown in Figure 4.3. The experiment is performed for 12 flash memory segments, collected from 3 MSP430F5438 sample chips. The red lines show the percentage of programmed flash memory cells, blue lines show the percentage of erased flash memory cells, and green lines show the percentage of unstable flash memory cells. We can identify three distinct regions in Figure 4.3. The first region is where the T_{PERASE} is very small and all the flash cells are in programmed state (illustrated by red lines being at 100% and blue lines at 0%). In the second region, a small change in the partial erase time results in a significant change in the states of flash memory cells; consequently, the slopes of red and blue lines are very steep. In this region, the number of unstable flash cells reaches its peak value. However, this peak never exceeds 2% as shown in Figure 4.3(b). In the third region, the percentage of erased cells slowly approaches 100%.

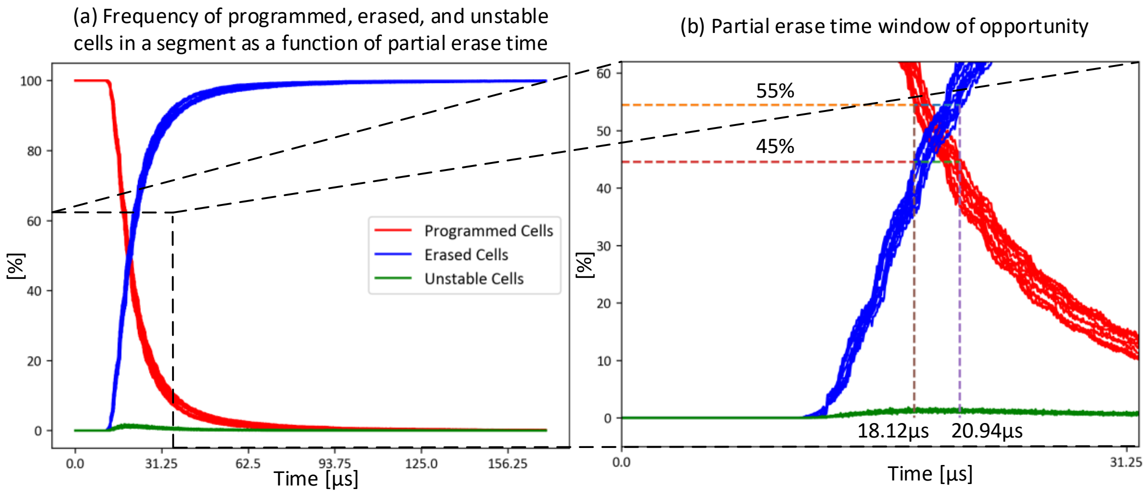


Figure 4.3. Percentage of the erased, programmed, and unstable flash memory cells as a function of the partial erase time.

4.4 Partial Flash Erase for Fingerprint Generation

Partial flash memory operations are often used to explore properties of flash memory cells when transitioning from one logical state to another. For example, an erase operation involves changing the state of flash memory segment from logic state ‘x’ to logic state ‘1’. Partial erase operations with different erase times help to reveal the speed of transitions from programmed to erased state. This speed correlates to the threshold voltage variations of the flash memory cells inside a segment. These variations in physical property of the flash cells are a result of manufacturing process variations rather than a result of deliberate action. This exposed physical property expressed in terms of logical state of flash memory cells can be used as a fingerprint.

A partial program operation is another operation that can be used to reveal flash cells’ physical properties. However, the method used on a partial program operation is not optimal as it requires a greater precision and time resolution in controlling the duration of the operation [63]. A flash program operation in NOR flash memory is performed on bytes or words (16- bits). To generate longer bit vectors for fingerprint, a partial program operation needs to be repeated on multiple words. An erase operation, on the other hand, is carried out on an entire segment. In addition, an erase operation takes more time than a program operation, making it easier to control its duration.

Since a fingerprint is a binary vector, the number of bits of the vector in state ‘1’ must be close to the bits in state ‘0’. This means that the fingerprint must have a hamming weight close to 50% to ensure that the fingerprint is not biased towards either ‘1’ or ‘0’. Thus, while performing an erase operation, we must be able to identify a vicinity of partial erase time where the number of flash cells in the erased state is equal or close to the number of flash cells in the programmed state. A similar partial erase time is shown in Figure 4.3(b) as the window of opportunity. This shows that by carefully tuning the partial erase time, the flash memory cells can be brought to the state where approximately half of the cells are in the stable ‘1’ state and the other half are in the stable ‘0’ state.

Let us now describe the step-by-step generation of a flash memory fingerprint from a NOR flash segment using a partial erase operation and the classification procedure that we discussed in Section 4.3 above. For this purpose, we take a flash memory segment S1. This flash memory segment is erased and programmed so that all the cells are at logic ‘0’ state. Then a partial erase operation is performed such that the number of flash cells in erased state is approximately equal to the number of flash cells in programmed state.

The state of the flash memory segment S1, organized in 256 words (vertical columns) with 16 bits each (horizontal rows), after a partial erase operation is presented in Figure 4.4(a). The programmed cells are shown in red while the erased cells are shown in blue. Unstable cells are shown in green.

The process of extracting a fingerprint from the segment S1 is repeated and the state of the segment (denoted as S1' here) is illustrated in Figure 4.4(b). An XOR operation between two states S1 and S1' is shown in Figure 4.4(c). A vast majority of the flash cells marked in blue in Figure 4.4(c) indicates that the proposed fingerprint extraction is a repeatable procedure.

A partial erase operation to yield ~50% distribution of the erased state and the programmed state is performed in another flash memory segment S2. The state of the segment S1 is shown in Figure 4.4(d). An XOR operation between the state of the segment S1 and the state of the segment S2 is shown in Figure 4.4(e). A random pattern indicates that the fingerprints extracted from two segments of the same NOR flash memory are not correlated.

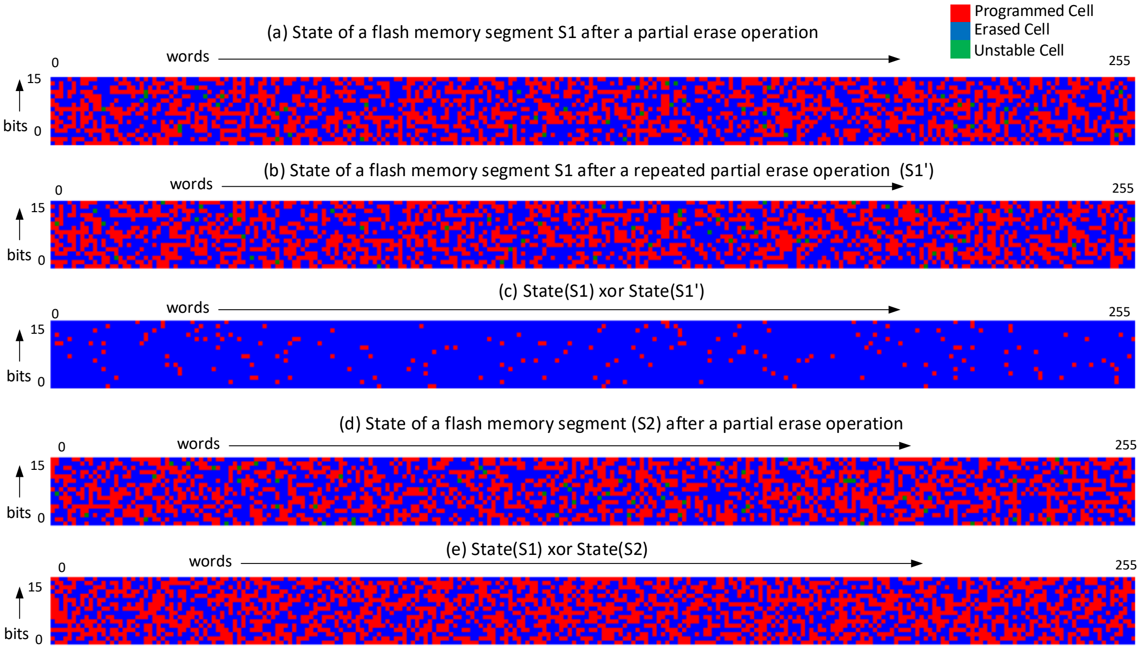


Figure 4.4. State of flash memory cells after a partial erase operation in memory segments S1 and S2.

The results from Figure 4.4 provide a preliminary evidence that the characterization of the flash memory cells after a partial erase operation can be used for fingerprint extraction, repeatedly producing fingerprints unique for each flash memory segment.

However, to analyze how the state of flash memory cells changes as a function of the partial erase time, we design another set of experiments where the partial erase time is varied. In every iteration of the experiment, the flash memory segment is erased, then fully programmed, and then a partial erase operation is performed, followed by the characterization step. Next, T<span>PERASE</span> is increased by a small period of time and the experiment is repeated.

Figure 4.5 shows the results of the experiment for 32 flash memory cells, numbered from 0 to 31, while TPERASE is varied from 13.5 µs to 19.5 µs in small but nonuniform time steps. A red square indicates the programed state, a blue square indicates the erased state, and a green square indicates the unstable state. Each flash cell has a particular time when it changes its state. For example, the flash cell at bit position 2 changes its state at T_{PERASE} ≈ 17.75 μs. The transitions exhibit monotonicity, i.e. the memory cells typically remain in the erased state once they change the state. Exceptions are possible, but they are rare. For example, bit 13 at T_{PERASE} = 15.938 μs transitions to programmed state.

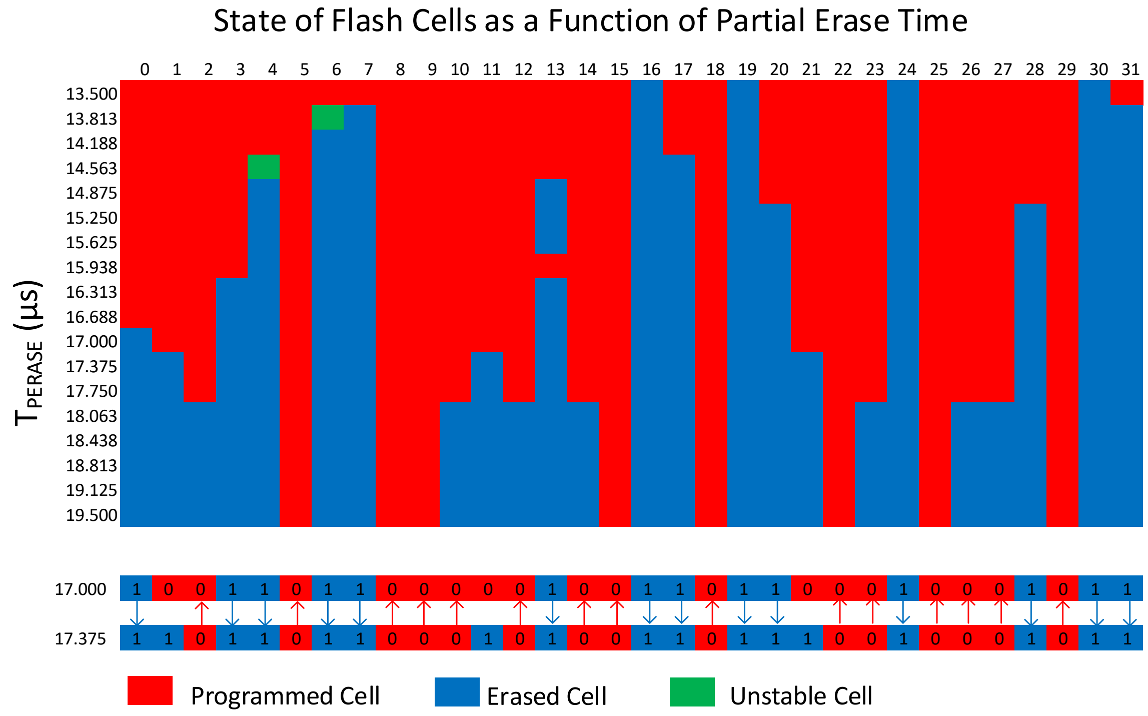


Figure 4.5. States of flash memory cells as a function of T_{PERASE}.

Let us select the flash memory state in Figure 4.5 at T_{PERASE} = 17.375 μs as a part of our fingerprint during enrollment phase. The number of erased cells at this partial erase time is 17 (53.125%) and the number of programmed cells is 15, with no unstable cells out of 32 bits under consideration. Based on the observations presented above, any future procedure that performs similar characterization of the flash memory segment with T_{PERASE} = 17.375 µs should yield similar fingerprint. Similarly, the fingerprint generated from another flash segment or another flash memory chip should produce an entirely different fingerprint.

To compare the fingerprint generated during the enrollment stage, i.e. enrollment fingerprint (EF), and any future fingerprint generated for authentication purpose, i.e. Authentication Fingerprint (AF), we develop a metric. Here, we assume that the T_{PERASE} during authentication, i.e. $T_{PERASE.AUTH}$ is less than or equal to the T_{PERASE} during enrollment phase, i.e. $T_{PERASE.ENROL}$. This should ensure that any cell that is programmed in the enrollment fingerprint, should remain in the programed state in the authentication fingerprint. Similarly, any cell that is erased in the authentication fingerprint, should be erased in the enrollment fingerprint. We develop two metrics to capture these properties that are described below:

(a) **Matching0s**: Matching programmed cells (logic '0') in EF and AF should be a subset of the programmed cells in EF, i.e. the ratio of the number of matching 0s in EF and AF to the number of 0s in EF should be close to 1 in the ideal case.

(b) **Matching1s**: Matching erased cells (logic '1') in EF and AF should be a subset of erased bits in AF, i.e. the ratio of the number of matching 1s in EF and AF to the number of 1s in AF should be close to 1 in the ideal case.

Thus, combining the two parameters described above, we define the Similarity Index (SI) shown in Equation 1.

$$SI = \frac{\left(\frac{\text{\#Matching 0s in EF and AF}}{\text{\#0s in EF}} + \frac{\text{\#Matching 1s in EF and AF}}{\text{\#1s in AF}}\right)}{2} \qquad \text{Equation 1}$$

For the example presented in Figure 4.5, if we take authentication fingerprint at T_{PERASE} = 17.0 µs, the authentication fingerprint has 14 1s and 18 0s. The number of matching 1s is 14 and the number of matching 0s is 15. Thus, the similarity metric is (14/14 + 15/15)/2 = 1. Thus, these two fingerprints are considered matching.

4.5 Device Identification using Flash Memory Fingerprints

The protocol to verify the authenticity of devices using flash memory fingerprint involves two stages: (a) device enrollment and (b) device authentication. The device enrollment phase is carried out by the device manufacturer (chip maker) or the manufacturer of the product (e.g. IoT platform vendor). The result is the enrollment fingerprint that is stored in a database. The algorithm for enrollment phase involves partial erase operation and characterization similar to what we have seen in Code 4.1 above.

The algorithm for enrollment phase is shown in Code 4.2. Here, instead of walking through all the partial erase times, the algorithm searches for suitable T_{PERASE} within a time window defined by $T_{PERASE.MIN}$ and $T_{PERASE.MAX}$ that is most likely to result in evenly spilt number of programmed and erased bits. The flash segment is fully erased and fully programed (lines 7 and 8) and partially erased (line 9). It is then repeatedly read N_R times to characterize flash memory cells (lines 12 to 15). The characterization operation in this case adds the values read from each bit position and performs majority voting. This means the bit whose sum is greater than $N_R/2$ is characterized as 1 while the bit whose sum is less than or equal to $N_R/2$ is characterized as 0 (lines 17 to 20). This step ensures that each flash bit position in characterized as either 0 or 1 (lines 16–20). For every flash bit that is characterized as 1, the variable *count_erased* is increased to keep track of the number of erased flash bits.

A ratio of the number of flash bits that are classified as erased flash bits to the total number of bits in a segment is also calculated (line 23). If the ratio variable *Ratio* is in the range of 0.50 to 0.55, the current characterization of flash segment bits is recorded as the enrollment fingerprint. In the case where there is a chance of multiple memory segments to be used for fingerprint generation, the address of the memory segment can also be added to the fingerprint descriptor.

If the number of erased bits is outside the desired range of [0.50, 0.55], the partial erase time TPERASE is either increased or decreased (lines 25 and 26) and the entire operation is repeated. The microcontroller used in this experiment could be tuned to control flash memory operation at a single processor clock cycle, thus providing opportunity for fine-grained control of TPERASE. However, coarse-grained control would also suffice.

```
void generate_enrollment_fingerprint(segment_address){
  done = false;
  $T_{PERASE}$ = ($T_{PERASE}$.min + $T_{PERASE}$.max)/2;
  do {
    fingerprint = 0000..00b;                     // Initialize fingerprint
    count_erased = 0;                            // Clear erased bits counter
    fully_erase_segment(segment_address);        // Write 1s
    fully_program_segment(segment_address);      // Write 0s
    partial_erase_segment(segment_address,$T_{PERASE}$);// Partial erase
    for(i=0; i<$N_{Words}$; i++){                // For each word in a segment
      $F_{SSUM}$[16]= {};                        // Initialize bit sum vector
      for(j=0; j<$N_R$; j++){                    // For NR times
        $F_R$ = ReadWord(i);                     // Read word i
        $F_{SSUM}$ += $F_R$;                     // Sum individual bit values
      }
      for(b_index=0; b_index<16; b_index++){     // For each bit
        if($F_{SSUM}$>$N_R$/2) {                 // Use majority voting
          fingerprint |= (1<<(b_index*i));       // to determine its value
          count_erased++;
        }
      }
    }
    Ratio = (count_erased)/(4096);
    if (Ratio > 0.50 && Ratio <= 0.55) done = true;
    else if (Ratio < 0.50) $T_{PERASE}$ = $T_{PERASE}$ + $\Delta$t;
    else $T_{PERASE}$ = $T_{PERASE}$ – $\Delta$t;
  } while (!done);
  EnrollmentFingerprint = {fingerprint, SegmentAddress};
}
```

Code 4.2. Algorithm for generation of enrollment fingerprint.

The authentication procedure is similar to the fingerprint enrollment procedure with one difference. The authentication is initiated by a customer. We assume that the typical partial erase time used for enrollment operation ($T_{PERASE.ENROL}$) is publicly disclosed for a family of devices, as well as the lower boundary of the Ratio value that is used in creating the enrollment fingerprint (i.e., 0.5 in Code 4.2). Alternately, the customer may place an authentication request for the given type of device and the manufacturer responds by providing these parameters. In the case multiple enrollment fingerprints exist for a single device, the manufacturer may also send the starting address of the flash memory segment to be used for generating fingerprint.

The customer side extracts the authentication fingerprint using the steps shown in Code 4.3. The flash memory segment is fully erased and programmed (lines 6 and 7). Then, a partial erase operation is performed with $T_{PERASE} = T_{PERASE.ENROL}$-dT. The flash memory segment is repeatedly read and characterized to obtain the fingerprint (lines 9 - 21). The fingerprint thus generated is considered the authentication fingerprint if the ratio of number of erased cells to the total number of cells in the memory segment is in the range [0.45, 0.50] (lines 23 - 25). Otherwise, the partial erase time T_{PERASE} is adjusted (lines 26 and 27) and the steps are repeated until the condition is satisfied.

```
1.  void generate_authentication_fingerprint(segment_address){
2.    $T_{PERASE}$ = $T_{PERASE.ENROL}$-dT;
3.    do{
4.      fingerprint = 0000..00b;                     // Initialize fingerprint
5.      count_erased = 0;                            // Clear erased bits counter
6.      fully_erase_segment(segment_address);        // Write 1s
7.      fully_program_segment(segment_address);      // Write 0s
8.      partial_erase_segment(segment_address,$T_{PERASE}$);// Partial erase
9.      for(i=0; i<$N_{Words}$; i++){                        // For each word in a segment
10.       $F_{SSUM}$[16]= {};                                 // Initialize bit sum vector
11.       for(j=0; j<$N_R$; j++){                             // For NR times
12.         $F_R$ = ReadWord(i);                              // Read word i
13.         $F_{SSUM}$ += $F_R$;                               // Sum individual bit values
14.       }
15.       for(b_index=0; b_index<16; b_index++){     // For each bit
16.         if($F_{SSUM}$>$N_R$/2) {                          // Use majority voting
17.           fingerprint |= (1<<(b_index*i));       // to determine its value
18.           count_erased++;
19.         }
20.       }
21.     }
22.   $Ratio_{AUTHENTICATE}$ = (count_erased)/4096;
23.   if (0.45 <= $Ratio_{AUTHENTICATE}$ <= 0.50) {          // termination condition
24.     AuthenticateFingerprint = fingerprint;
25.     break;
26.   }else if ($Ratio_{AUTHENTICATE}$ < 0.45) $T_{PERASE}$ = $T_{PERASE}$ + Δt;
27.     else $T_{PERASE}$ = $T_{PERASE}$ - Δt;
28. }while(1);
```

Code 4.3. Algorithm for generation of authentication fingerprint.

The authentication fingerprint is sent to the manufacturer who then compares the two fingerprints using the Similarity Index (SI) defined in Section 4.4. A lower bound of the similarity index can be required to consider the authentication successful (e.g., SI>0.85). Otherwise, the device fails authentication.

4.6 Experimental Evaluation and Results

The experimental set up based on TI MSP430 family of microcontrollers is used to perform enrollment and initial authentication at room temperature. For the experimental evaluation purpose, we take three physical chips of MSP430F5438. 50 flash memory segments from each of the three chips are used for a total of 150 logical devices. Section 4.6.1 discusses the results from evaluation of the enrollment phase. Section 4.6.2 discusses the evaluation of the authentication phase, specifically fingerprint uniqueness and reliability. To evaluate robustness of the proposed technique, we evaluate its effectiveness in presence of different environmental conditions. Section 4.6.3 discusses the effects of changes in the power supply voltage and environmental temperature. Section 4.6.4 explores feasibility of sub-segment fingerprints. Flash memories are prone to wear-out effects as they age. Section 4.6.5 evaluates the wear-out effects on the proposed fingerprint based authentication. Section 4.6.6 concludes this section with a comparative performance analysis of the proposed technique relative to the existing techniques.

4.6.1 Enrollment Phase

Figure 4.6 shows the results of enrollment characterization. The distribution of the ratio of erased cells (logic ‘1’) in enrollment fingerprints to the total number of cells in a segment is shown in Figure 4.6(a). The target ratio is between 50% and 55%

as discussed in Code 4.2. The logical devices used for our evaluation have similar characteristics to those shown in Figure 4.3, thus a successful enrollment can be found in a single try.

A trade-off exists between minimum change of partial erase time (Δt in Code 4.2) and the ability to achieve the target ratio. Our setup allows us to control partial erase times with a single clock cycle resolution (~ 16 MHz in our setup), and thus we can achieve a near perfect ratio of 50%. However, a relatively coarse grained Δt is used to minimize the number of steps in the enrollment phase. Since the characteristics of logical devices (segments) are similar, enrollment typically takes place on a first try.

For each of the enrollment fingerprint, we ensure that the number of unstable bits is below a threshold (< 2%). Figure 4.6(b) shows the distribution of stable bits in a segment. We can observe that more than 98% of all the bits in fingerprints are characterized to be stable bits. This experiment confirms that the number of unstable bits during the partial erase process does not exceed the given threshold.

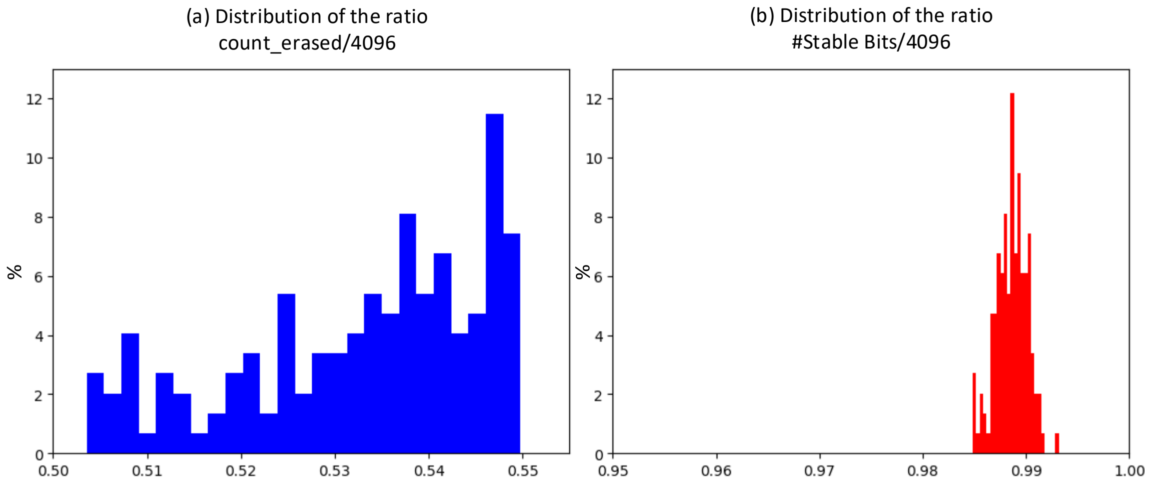


Figure 4.6. Fingerprint enrollment characterization.

4.6.2 Uniqueness and Reliability

For a fingerprint generated from a device, the two properties of interests are uniqueness and reliability. Uniqueness indicates that a fingerprint that is generated from a flash memory segment on address X on sample device D1 is completely different from a fingerprint created on the same address X on sample device D2. Furthermore, each segment within one physical device yields a unique fingerprint. Reliability means that fingerprints repeatedly extracted from a single memory segment of a device are almost identical to each other.

We make use of the Similarity Index (SI) presented in Section 4.4 to analyze both uniqueness and reliability. Figure 4.7(a) presents the distribution of SI values obtained by comparing the enrollment fingerprint of each of 150 logical devices with the enrollment fingerprints obtained from other logical devices. We call this metric *Inter Similarity Index*, which represents the uniqueness of the fingerprint generated. The distribution peaks at 0.5 with tails extending from 0.47 to 0.53.

Figure 4.7(b) shows the distribution of SI determined by pairing the corresponding authentication and enrollment fingerprints of all logical devices. We call this metric *Self Similarity Index*, which represents the reliability of the fingerprint generated. The SI distribution ranges from 0.89 to 0.97. Consequently, we can define a threshold for the authentication phase – if the SI value of an authentication fingerprint is at least 0.89 when compared with one of the enrollment fingerprints in the database, the device passes authentication. Otherwise, the authentication fails.

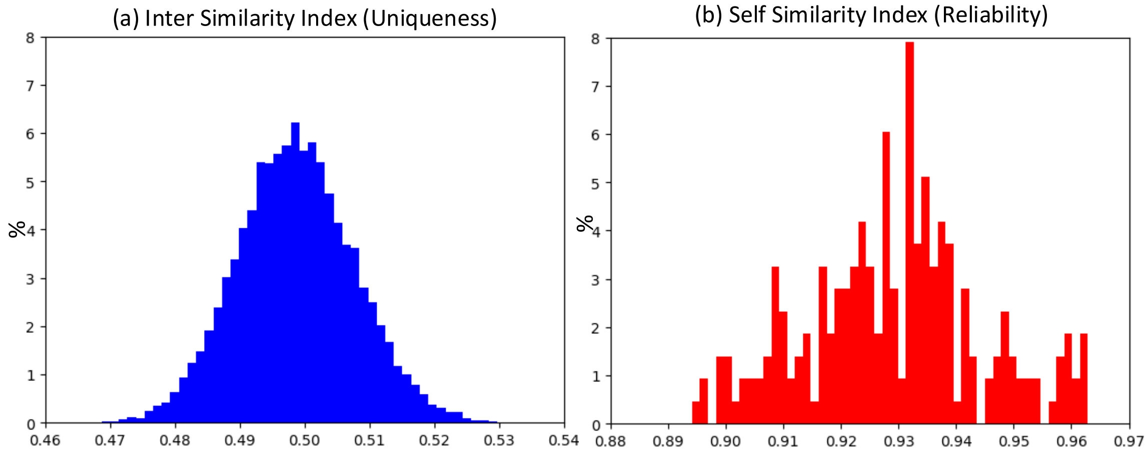


Figure 4.7. Evaluation of similarity index.

4.6.3 Effects of Environmental Factors

The proposed fingerprint based authentication must be resilient to changes in environmental conditions to be a feasible solution for device authentication. Thus, the effects of power supply and temperature effects are analyzed. For experimental purposes, the authentication fingerprints are collected while the microcontroller is running at different supply voltages and at different temperatures. The enrollment fingerprints are created under the default power supply at room temperature (Supply Voltage = 3.3 Volts, T = 25°C).

Figure 4.8(a) shows the similarity indices as a function of the power supply used during the authentication process. Similarity indices are shown as box-and-whisker plots. The orange horizontal line represents the median value, while the two horizontal ends of each box represent Q1 and Q3 (0.25 and 0.75 quantiles). The minimum and maximum of each set of observation is represented by the two extremities. The top plots represent the self-similarity indices while the bottom plots represent the inter similarity indices. The power supply voltage is varied from 3.0 V to 2.0 V (minimum required for proper flash operation) with a decrement of 0.2 volts.

The results in Figure 4.8(a) show that there is a very little impact of supply voltage on the inter-similarity index. The self-similarity index changes with the power supply voltage. However, this change is not significant to impact the feasibility of the proposed mechanism. There is no observable trend in self-similarity index as a function of the supply voltage. The likely explanation for this relative stability is that the internal charge pumps in the flash controller provide relatively stable voltage to the flash memory, irrespective of the external supply voltage.

Similarly, the effect of ambient temperature on the similarity index are explored by varying environment conditions. Figure 4.8(b) shows the inter- (bottom) and self- similarity index (top) distributions using similar box-and-whisker plots. The enrollment fingerprints obtained at 25°C are compared to the authentication fingerprints generated at 25°C, 0°C and 50°C. The impact of the temperature on the inter-similarity index is minimal. It does reduce the median of the self-similarity index and the distribution. But, the difference between the self-similarity index and inter-similarity index is still healthy. This exhibits the feasibility of the proposed mechanism over a range of temperature. Our experimental set-up was not conducive to a wide range of temperature-based testing; however, the feasibility of the proposed method is not expected to change over the prescribed range of operating temperature as set by the device manufacturer.

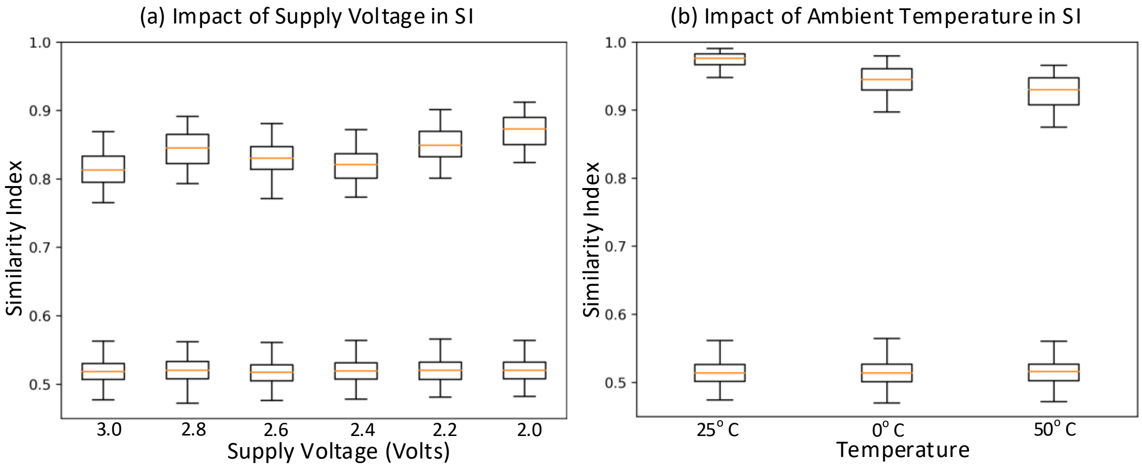


Figure 4.8. Impact of environmental factors on Similarity Indices (inter and self) (SI).

4.6.4 Sub-segment Fingerprints

Analyses presented so far considers entire 4096-bit segment as a fingerprint. However, we discuss feasibility of sub-segment fingerprints. The results presented in Figure 4.4 indicate that the distribution of programmed and erased bits is uniformly random throughout the segment, without any spatial anomalies. Consequently, we can consider using *n* consecutive bits within a flash segment as a fingerprint.

A series of experiments is performed considering sub-segment fingerprints, where n = 4096, 2048, 1024, 512 and 256. For the experiment, a 4096-bit long enrollment fingerprint is divided into 4096/n (indexed from 0 to 4096/n-1) separate enrollment sub-fingerprints each of n-bit length. Similarly, the authentication fingerprint is also divided into 4096/n separate authentication sub-fingerprints each of n-bit length.

Figure 4.9 shows the inter- and self- similarity indices as a function of the fingerprint length. The results show that even a 256-bit fingerprint provide a healthy margin between the self- and inter- similarity indices without use of any encoding or error correction techniques which often tend to be computationally cost prohibitive for low-power IoT systems.

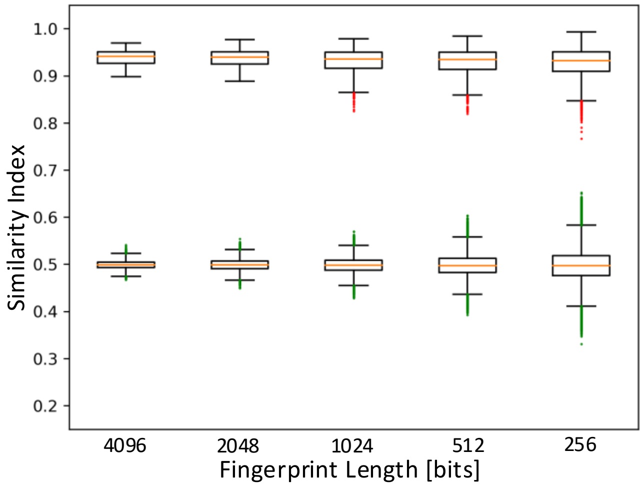


Figure 4.9. Evaluation of Similarity Index using sub-segment fingerprints.

If a 256-bit fingerprint is used to uniquely identify logical devices, the number of logical devices that we can identify is 2400. This is because 150 logical devices (segments) are considered in our experiments, each with 16 (i.e. 4096/256) sub-segment fingerprints. Thus, the number of bits required for a unique fingerprint is $\lceil \log_2 2400 \rceil$=11.22 bits. This means that 11.22/256 = 0.048 bits of the fingerprint per single flash bit. Using this metric, a 512-bit long fingerprint can uniquely identify ~5.6 million devices, whereas a 1024-bit fingerprint can uniquely identify 31.37 trillion devices.

4.6.5 Wear-out Considerations

Flash memories are susceptible to errors caused by wear-outs or stresses in the flash cell oxides (or ageing) that arise from a repeated erase and program operations. Thus, we study the effects of ageing by generating authentication fingerprints under different wear-out levels. NOR flash memory typically can sustain 100,000 program-erase cycles, before failing permanently. Thus, we can assume that the proposed method can be used to create at least 50,000 authentication fingerprints before permanently failing. This is based on the assumption that a single full PE cycle and a single partial erase operation should be sufficient to generate an authentication fingerprint during the authentication phase.

Figure 4.10(a) shows the self-similarity and inter-similarity indices for two sample chips as a function of the number of PE cycles. Here the enrollment fingerprint is generated while the flash memory segment is relatively new, whereas the authentication fingerprint is obtained at different wear-out levels. Each of the wear-out level is achieved by manually performing repeated program-erase operations (PE cycles). The results show that there is negligible change in the inter-similarity index at all stress levels. The self-similarity index decreases sharply after the initial 500 PE cycles; after this initial drop, the self-similarity index enters a region with mild degradation as the number of PE cycles increases. It remains close to 0.8, even after 30,000 PE cycles.

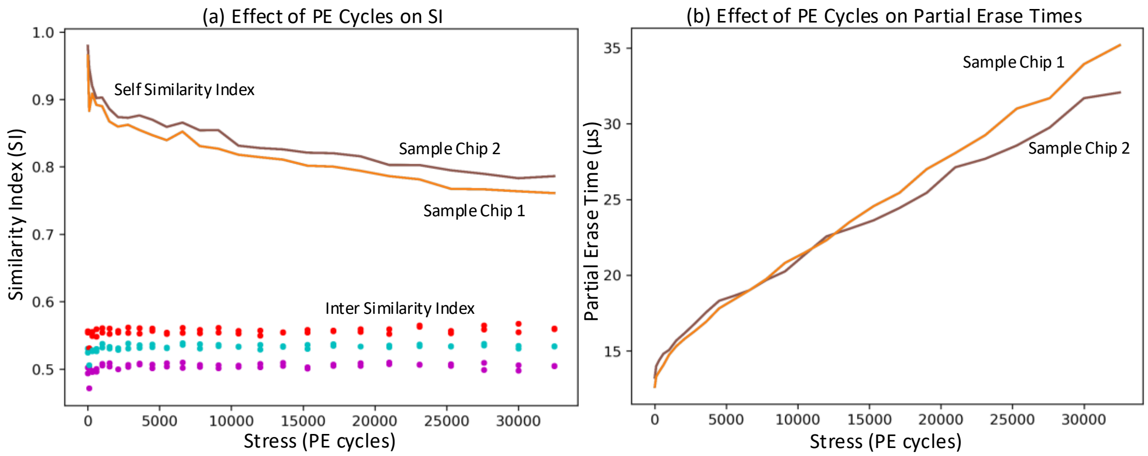


Figure 4.10. Impact of PE cycles on (a). SI and (b). Partial erase times needed to a generate qualified fingerprint.

For the two sample chips presented in Figure 4.10(a), we use a linear fit to model the self-similarity index as a function of the number PE cycles. The modeled changes of the self-similarity index are described in Equation 2.

$$Self\_SI_{Stress} = Self\_SI_0 - (k * 10^{-6} * Stress) \qquad \text{Equation 2}$$

The parameter *Stress* is the number of PE cycles and *k* is the parameter that captures the degradation of similarity index relative to the index of the fresh memory block, *Self_SI₀*. The value of *k* varies between 4 and 5 for sample chips used in this study.

The impact of stressing on the inter-similarity index is modest. The upper tail of the inter-similarity index distribution is below 0.57, thus leaving enough margin between the self-similarity indices and inter-similarity indices (Figure 4.10(a)). This confirms the robustness of the proposed technique in presence of ageing.

The partial erase time (T_{PERASE}) required for generation of a qualified authentication fingerprint also changes with PE cycles. Figure 4.10(b) shows how T_{PERASE} changes as a function of the stress level. The partial erase time steadily increases with an increase in the stress level. Consequently, the partial erase time when generating authentication fingerprint may need to be adjusted with an increased number of PE cycles. The algorithm presented in Code 4.3 (lines 26 and 27) already accommodates for change in the partial erase time.

4.6.6 Performance Considerations

Ideally, generating fingerprints should take as little time as possible. The proposed algorithm achieves a very good performance and can be implemented on a low-end microcontroller requiring as little as 4 KB of RAM memory. The full erase operation of a memory segment accounts for about 25 ms. Full segment programming takes ~3 ms which sets all the bits in the flash memory segment to 0. The partial erase operation takes less than 50 µs, whereas reading a flash memory segment takes about 50-150 µs, depending upon the number of reads (parameter N_R in Code 4.3). In the case when multiple retries are required for generation of a qualifying fingerprint, the time required is in multiples of 30 ms. In our experiments, we are able to generate a qualified fingerprint in a single try, but even with multiple retries the achieved performance compares favorably to other proposed techniques that require several seconds or even minutes [55] to derive a fingerprint. Our proposed method does not require any helper data [49] or helper functions [61]. Since the proposed fingerprint extraction generates a fingerprint of 4096 bits in 30*n ms, the throughput of our proposal is 136/n Kbits/s. Here, n is the number of retries required to generate the qualifying fingerprint. This shows the best-case throughput is 136 Kbits/s.

Table 4.1 presents a summary of fingerprint techniques in flash memories. It reports the type of flash memory used, throughput achieved in generating fingerprints/PUFs, size of fingerprint in bits and the memory overhead required by the extraction algorithm. The reported overhead is the number of bits needs to extract one fingerprint bit. Our extraction algorithm requires one bit of memory per one bit of generated fingerprint, thus significantly outperforming other algorithms. The size of the fingerprint, memory overhead and the throughput achieved make the proposed method suitable for resource-constrained low-end microcontrollers.

Method	**Flash Type**	**Throughput**	**Evaluation: fingerprint size [bits], memory overhead per bit of fingerprint, latency, and implementation cost**
Applies multiple partial programs to extract threshold variations in flash memory cells [55]	NAND Flash Memory	Not reported	*Size:* 16 Kbits *Overhead:* 1+sizeof(uint) *Latency:* High, requires repeated partial program operations
Applies multiple partial erases, partial program, and program disturbances pulses to induce disturbances [60]	NAND Flash Memory	1364.1 bits/s	*Size:* 128 bits, *Overhead:* ~ 142 (Bit-map method), ~16 (Position-map method) *Latency:* High, requires repeated flash operations
Lowers supply voltage to suspend erase operation and expose threshold voltage variation [64]	1.5T Superflash Memory	Not reported	*Size:* 512 Kbits *Overhead:* 2 *Latency*: Small *Special:* Requires lowering power supply

Repeatedly applies program operations with selected data to achieve program disturb [51]	NAND Flash Memory	9.09 Kbits/s	*Size:* 20 Kbits *Overhead:* ~1 *Latency:* High, requires repeated program operations
Applies repeated partial program operations to induce disturbances [49]	NOR Flash Memory	Not reported	*Size:* 3n bits *Overhead:* 8 *Latency:* High, requires repeated program operations *Special:* Requires BCH encoding
Applies repeated erase suspend operation for fingerprint generation [62]	Superflash Memory	266 bits/s	*Size:* 16 bits *Overhead*: 256 *Latency*: High, requires repeated partial erase operations
This work: applies simple partial erase operation to expose threshold voltage variation	Embedded NOR Flash Memory	136/n Kbits/s	*Size*: 4 Kbits *Overhead*: 1 *Latency*: Small

Table 4.1. Summary of fingerprinting techniques in flash memories, their throughputs and memory overheads.

4.7 Concluding Remarks

A robust and reliable fingerprint can be generated from partially erased NOR flash memory. The generated fingerprint can uniquely identify NOR flash memories. In-system programmable NOR flash memory is a standard part in all modern microcontrollers. Thus, this technique of fingerprint generation for identification purpose can be readily implemented in modern microcontrollers. The proposed technique is especially suited for low-power embedded systems and IoT as it offers following advantages over existing approaches: (a) it requires no additional hardware support, (b) it is solely implemented in software and can easily be implemented in modern microcontrollers, (c) it produces robust fingerprints that are resilient to changes in operating conditions, and (e) the algorithm self-adapts to the changes in flash memory ageing effects.

CHAPTER 5

FLASHMARK

Electronics counterfeiting is a longstanding problem that is growing in scope and magnitude [65] in modern globalized semiconductor supply chains. This chapter presents a technique for detection of counterfeiting in NOR flash memories. Section 5.1 discusses the problem of counterfeiting in semiconductor industry. Then, Section 5.2 discusses prior related efforts tackling the problem of counterfeiting. Our technique for counterfeit detection is based on flash memory watermarks. This watermarking technique, *Flashmark* [66], imprints manufacturer information into a dedicated section of NOR flash memory. Section 5.3 describes the proposed watermarking technique. Section 5.4 demonstrates the experimental evaluation and performance metrics on embedded NOR flash memories in a family of low-cost microcontrollers. Finally, Section 5.5 concludes the chapter.

5.1 Counterfeiting in Semiconductor Industry

Counterfeiting in semiconductor industry represents a significant concern because recycled, remarked, overproduced, out of specification, cloned, fall-out and tampered with chips can enter globalized supply chains. Recycled or remarked chips include old chips that are forged with new information and sold as new. 80% of all the counterfeited electronics are recycled or remarked chips [65]. Overproduction is another major issue where an Intellectual Property (IP) designer requests a foundry to manufacture a certain number of chips. However, the foundry manufactures more chips than requested, and sells the surplus for profit. Similarly, rejected dies and fall-out chips can also enter the supply-chain through bad actors. Some of the manufactured chips fail a few corner cases during testing phase and are normally rejected. Such factory rejected or out-of-specification chips are also injected into supply chain through counterfeiters with access to the chip packaging sites or rogue employees. Reverse engineering of an IP for production of fake, cloned chips is yet another source of counterfeiting.

Counterfeiting in semiconductors poses a grave threat to consumers as well as to the semiconductor industry. The sales of counterfeit parts have been a cause for substantial losses to semiconductor industry [12]. It is estimated that U.S. based chip companies alone lose as high as $7.5 billion annually [67]. On the consumer side, counterfeit products provide sub-standard products and create problems in many application domains ranging from consumer electronics to mission critical applications such as transportation, health care, and military [11]. It is estimated that nearly 15% of electronic parts purchased by the U.S. military were made of counterfeit parts in 2011 [9], [67], [68].

Despite many proposals to cope with integrated circuit (IC) counterfeiting, portion of counterfeit chips continues to increase. Reports of counterfeit ICs in the supply chain has quadrupled from 2009 to 2011 [10]. A report published in 2010 indicates that 50% of Original Component Manufacturer (OCM) and 50% of distributors have encountered counterfeit parts [69]. It was also reported that a Senate Armed Services Committee uncovered more that 1 million “bogus parts” in the supply chain of the Department of Defense [67].

Analog ICs, microprocessor ICs, memory ICs, programmable logic ICs and transistors make up around 68% of counterfeited semiconductors [65]. Memory ICs alone accounted for 13.1% of all reported counterfeited semiconductors [70]. Flash memory ICs is a major target of counterfeiters because of its ubiquitous presence in electronic systems [23], [25], [71]–[74]. There can be multiple pathways for counterfeit flash memory chips to enter the supply chain. First, flash memories are used as storage media in many electronic products, e.g., smartphones, MP3 players, solid state drives, and USB drives. These electronic components have a limited lifetime; however, the flash memories remain functional even after the end of the product lifecycle. This provides an opportunity for adversaries to retrieve the flash memories from the printed circuit boards and recycle them as new. Second, the rejected dies or fall-out chips that fail post-fabrication tests can also enter the supply chain through counterfeiters with access to packaging sites, which are located in various countries. Third, counterfeiters may buy inferior flash chips from less reputed manufacturers and sell them for a higher price by rebranding the chip.

The use of inferior or defective counterfeit flash memories results not only in economic losses for the original chip manufacturer, but may also result in failures in end user applications, ranging from a loss of data and premature end-of-life to more serious catastrophic events.

5.2 Related Works and Motivation

A number of recent research efforts tackles the problem of counterfeit IC detection. Most of the proposals are based on physical and electrical inspection methods to detect the defects or damages that might be present in the component or IC under test. The physical inspection methods rely on inspecting the ICs thoroughly. In some of the proposals, low-power microscope (with less than 10x magnification) and X-ray imagery is used for careful analysis of the component under test which is then compared with a golden model. The golden model is an established genuine IC. Other complicated inspection methods involve use of Scanning Electron Microscopy, Terahertz Imaging and Scanning Acoustic Microscopy. X-Ray Fluorescence, Fourier Transform Infrared Spectroscopy, Ion Chromatography, and Energy-Dispersive X-Ray Spectroscopy are also used to analyze the chemical composition of ICs [65], [75]–[77].

Another class of techniques involves performing electrical analysis. This class of techniques perform different tests to identify electrical defects in ICs. These tests reveal AC and DC parameter shifts of the ICs under test from those equivalents of genuine or new ICs [65].

All the proposals above demand costly equipment and procedures and require rigorous analysis by experts. Moreover, most of the proposals are invasive in nature and suffer from limited effectiveness. Tracking ICs throughout the supply chain is another method that can be used to avoid counterfeiting. Tracking an IC throughout the supply chain can be performed by providing each ICs with a unique id. Electronic Chip Identifiers (ECIDs) is one of such solutions that programs a unique ID into the antifuse one-time programmable memory [78]. However, ECIDs are not common in flash memory chips. ECIDs require changes in physical masks and dedicated on-chip resources. Physical Unclonable Functions (PUFs) are another class of techniques that are used for chip identification, authentication, and on-chip key generation. They can also be used for tracking ICs or flash chips throughout the supply chain. However, use of PUFs require lengthy extraction process as well as maintenance of a large database for Challenge-Response (CR) pairs. In addition, it requires a method for contacting the chip manufacturer to verify the authenticity of each chip, which may place an additional burden on system integrators, as the time and costs of verification are increased.

Some of the recent proposals involve detecting changes in the physical properties of flash memories [23]–[25] to address the concerns of counterfeit flash memories. Used or recycled flash memory chips have significantly different timing characteristics. For example, program, erase and read times of a used flash memory chip differs significantly from new flash memory chips [24]. Sakib et al. [25] suggest that a change in the erase time is the best metric to distinguish between a fresh and a used or recycled flash memory chip. Sweeping partial program based technique is used by Guo et al. [23] to estimate age of flash memory to identify fake or recycled flash memory chip.

The proposals discussed above illustrate the growing interest in counterfeit detection and prevention. Whereas the existing proposals offer solutions to detect recycled flash memory chips, they cannot be readily applied to address other kinds of counterfeiting.

Our proposed technique, *Flashmark,* can be used to secure global supply chain of NOR flash memory chips or chips with embedded NOR flash memories by using digital imprints or watermarks. The proposed method relies on (a) imprinting watermarks into the physical properties of flash memory cells; and (b) extracting physical properties of flash memory chips through standard digital interface. Watermarks are imprinted by degrading the oxides of flash memory cells using repeated program erase operations. This degradation of oxides changes the physical properties of the flash memory cells and cannot be reversed, i.e. the imprinting is permanent [79], [80] and cannot be reversed. For example, if flash memory manufacturers imprint "accept" or "reject" information on every die they produce, they will prevent the counterfeiters from entering the out-of-spec or fall-out chips into the supply chain undetected. Even if the counterfeiter gets physical access to the fall-out chips, they will fail to convert the "reject" watermark into "accept."

5.3 Proposed Watermarking Technique: Flashmark

A flash memory chip manufacturer can imprint a known watermark into a section of flash memory during the die-sort testing phase. The imprinted information can include manufacturer identifier, die identifier, chip speed grade, chip testing status (e.g. "accept" or "reject") and other manufacturer related information as shown in Figure 5.1.

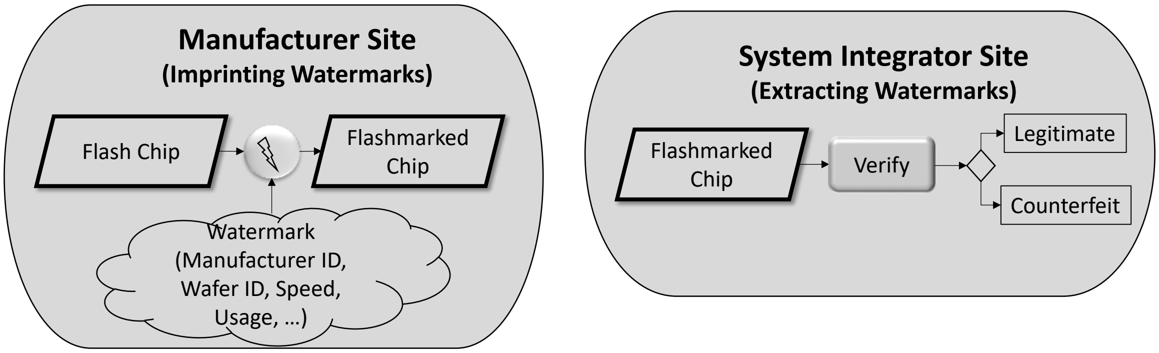


Figure 5.1. System view for the proposed watermarking technique.

Flashmark relies on a more elaborate process of imprinting watermark that is resilient to counterfeiters' tampering efforts. The imprinting of Flashmark that is performed at the manufacturer site involves repeated program-erase (PE) cycles. The program operation is performed by writing the data to imprint (flashmark), while the erase operation is a normal erase operation. These repeated PE cycles stress the flash memory cells, namely induce charge traps in the cell's oxide layers. This stress reduces the capability of flash cells to hold the charge and change their threshold voltages.

Flash memory continues to behave as expected as long as the level of stress, i.e. number of PE cycles, is below the cell's endurance limit (100,000 PE cycles for NOR flash memory) before the flash memory cells permanently fail. This is because the distance between the threshold voltages (V_{THE} and V_{THP}) is sufficiently large to ensure correct behavior. Therefore, we devise a partial erase based method to extract this change in V_{TH} (an 'analog' physical property) of the flash memory cells through a standard digital interface. Thus, the watermarking technique is based on changing the properties of flash memory cells by inducing changes in their threshold voltages, and using partial erase based method to extract the information that reflects the change in V_{TH}.

In the following subsections, we characterize behavior of the embedded NOR flash memory in presence of stress and formulate procedures for imprinting and extraction of the watermark. Section 5.3.1 presents the results of the NOR flash memory cells' characterizations that enable Flashmark. Section 5.3.2 describes the elaborate processes of imprinting and extraction of the watermark.

5.3.1 Stress-Induced Properties of Flash Memory Cells

Flash memory cells subjected to repeated program/erase operations (P/E) will change their physical properties, namely their threshold voltages (V_{TH}) may deviate from the 'normal' levels. To analyze this change in V_{TH}, we perform a characterization experiment on multiple flash memory segments. These flash memory segments are preconditioned by repeated erase and program operations starting from fresh flash memory segments (0 K PE operation) to worn-out flash memory segments (100 K PE operations). Each PE operation (or a cycle of PE stress) means that each word in a segment and each bit in a word is programmed and then the segment is erased.

At every 20,000 PE cycles, we perform the characterization of the flash memory cells in the segment. The algorithm used for characterization is presented in Code 5.1. For characterization, the segment is fully erased (line 5) and programmed (line 6). Then a partial erase operation is applied with $T_{PERASE} \approx 0$ (line 7). After the partial erase operation, we perform a majority voting-based characterization by reading the content of each word N_R times (lines 8 - 21). Each flash memory cell is characterized as erased (logic '1') if it is read as 1 for more than $N_R/2$ times. Otherwise, the cell is characterized as programmed (logic '0'). For each of the partial erase operations, the number of flash memory cells that are characterized as programmed (variable *cells_0*) and erased (variable *cells_1*) is reported for plotting. The partial erase time, T_{PERASE}, is increased and the characterization is repeated until T_{PERASE} is equal to the nominal segment erase time T_{ERASE}.

```
Void characterize_segment(segment_address){
  TPERASE = 0;
  do {
    cells_1 = 0, cells_0 = 0;
    fully_erase_segment(segment_address);          // Write 1s
    fully_program_segment(segment_address);        // Write 0s
    partial_erase_segment(segment_address,TPERASE);// Partial erase
    for(i=0;  i<NWords;  i++){                     // For each word in segment
      FSSUM[16]=  {};                             // Initialize bit sum vector
      for(j=0; j<NR; j++){                         // For NR times
        FR = ReadWord(i);                          // Read word i
        FSSUM  +=  FR;                             // Sum individual bit values
      }
      for(b_index=0; b_index<16; b_index++){       // For each bit
        if(FSSUM>NR/2) {                           // Use majority voting
          cells_1 += 1;
        }else {
          cells_0 += 1;
        }
      }
    }
    Report {cells_1, cells_0};
  TPERASE += Δt;}                                  // Increase TPERASE
  while (TPERASE <= TERASE);
}
```

Code 5.1. Algorithm for majority voting based characterization of flash memory segment.

Figure 5.2 shows the results of the characterization on MSP430F5438 microcontrollers. The vertical axis represents the number of flash cells while the horizontal axis represents the partial erase times (T_{PERASE}). The red lines show the number of flash cells that are characterized as erased cells (labeled *Cells_1*), while the blue lines show the number of programmed cells (labeled *Cells_0*). Since a single segment of the flash memory used for the experiment contains 4096 bits, the maximum value that the vertical axis can assume is 4096. The labels in each of the lines indicate the number of PE cycles applied before the characterization is performed. This means that 0 K indicates a fresh flash memory segment, 20K indicates 20,000 PE cycles applied before the characterization experiment is performed, 40K indicates 40,000 PE cycles applied, and so on. The plot only presents the data from 0 to 120 µs as the plot plateaus with $T_{PERASE}>120$ µs.

Figure 5.2 shows that all cells are in the programmed state for $0 \leq T_{PERASE} \leq 18$ µs for an unstressed flash memory segment (labeled 0 K). However, this number abruptly decreases with a slight increase in T_{PERASE} and the number of cells in the erased state increases. All 4096 flash memory cells are in the erased state for $T_{PERASE} \geq 35$ µs. It should be noted that the nominal segment erase time $T_{ERASE} = 24$ ms. Hence, the changes in the state of flash memory cells occur very early in the erase operation and they are sudden.

The shape of the curves obtained from segment subjected to PE cycles differ significantly from that obtained at 0 K. As the number of PE cycles increases, the transitions from the programed state to the erased state start later in the erase operation, they are more gradual, and it takes more time to erase all the cells in the segment. As per our observation, the partial erase times required to erase all cells in a segment subjected to 20 K stresses is 115 µs. The minimum T_{PERASE} when all the cells in a segment read as erased increases with stress levels; these time are 203 µs, 226 µs, 687 µs and 811 µs for 40 K, 60 K, 80 K and 100 K PE cycles, respectively.

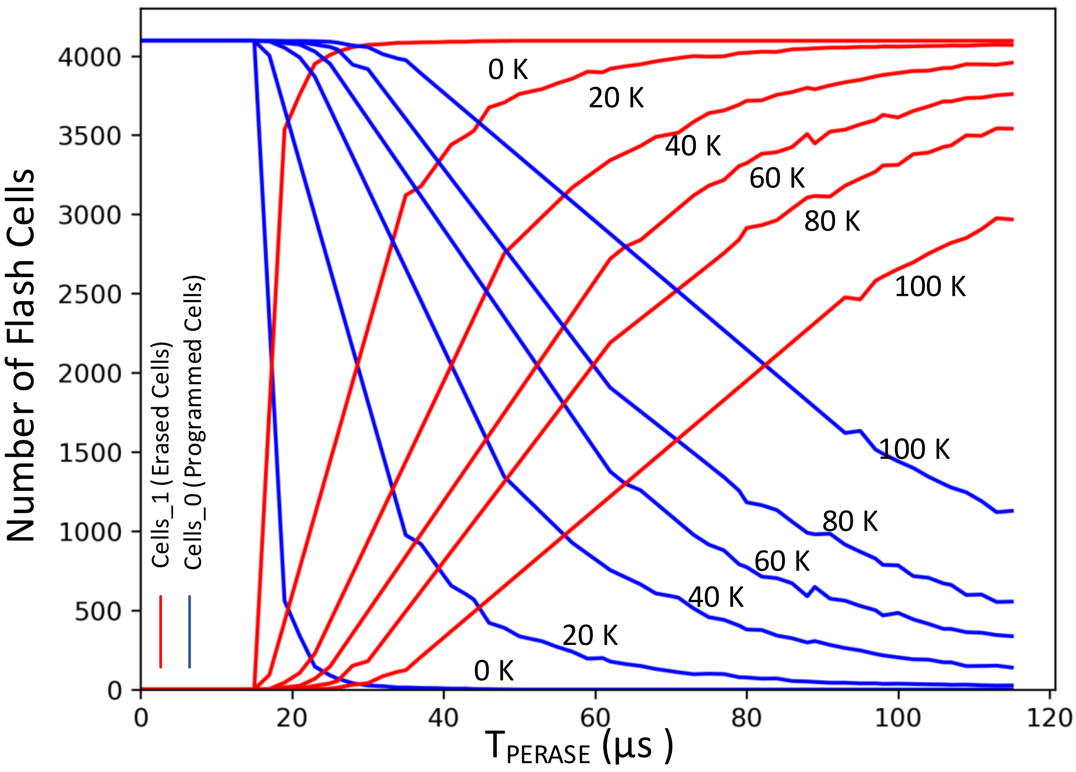


Figure 5.2. State of flash memory cells in a segment as a function of the partial erase times.

Figure 5.2 shows that analyzing the states of flash memory cells after a partial erase operation could be used to determine the stress levels or analog properties of flash memory cells. Figure 5.3 shows the number of programmed cells obtained from the characterization experiment for a stressed (50 K) and an unstressed flash memory segment (0 K) as a function of the partial erase times. By choosing an appropriate partial erase time, characterization can be performed to determine the states of individual flash memory cells. We call this partial erase time t_{PEW} to indicate that it is the partial erase time used for watermarking. If the majority of flash cells are programmed, this means the flash cells are worn out and resist the erase operation. If the majority of cells transition to the erased state, they are considered fresh. The idea is illustrated in Figure 5.3 by the dotted vertical line in the plot at t_{PEW} = 23 µs. At partial erase time of 23 µs, almost all the flash memory cells (3,833 out of 4,096) that are stressed 50 K are still at programmed state. This is indicated by the value of 50 K line being close to maximum value. On the contrary, almost all the flash memory cells on plot 0 K are in the erased state. This is indicated by the value of 0 K line being close to 0 at t_{PEW} = 23 µs.

This property of stressing and identifying stressed flash memory cells from the unstressed flash memory cells is the basis of our watermarking scheme, *Flashmark*.

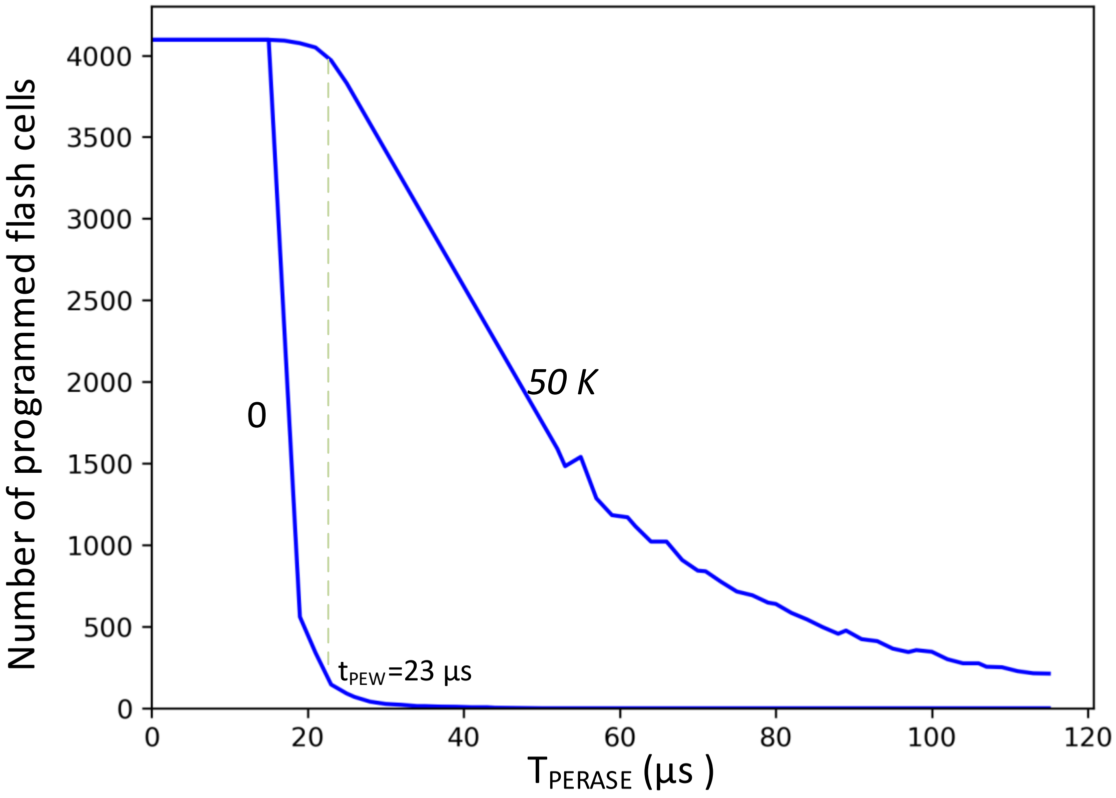


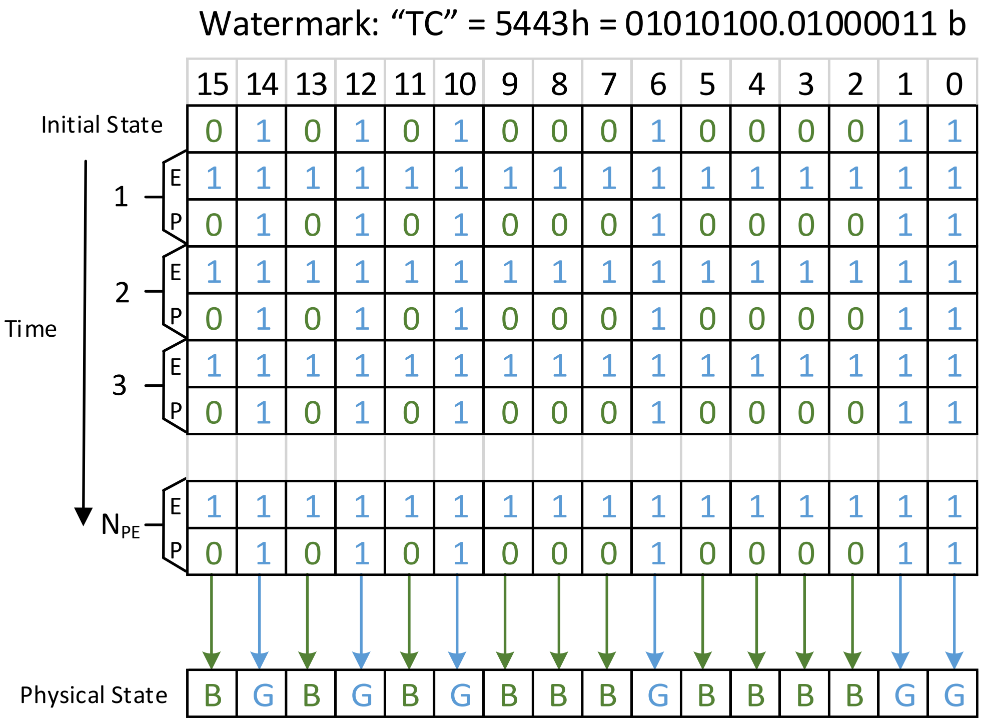
Figure 5.3. Detecting changes in physical properties caused by stressing.

5.3.2 Watermarking Procedure

Flashmark proposes an elaborate method where a manufacturer can imprint the information in a way that is more resilient to counterfeiter's tampering efforts. As an example of imprinting the watermark, let us assume that a word of flash memory segment is reserved for watermarking. Let us also assume that a manufacturer wants to imprint ASCII value of a string "TC" that stands for a virtual *Trusted Chipmaker*.

A word reserved for storing the watermark contains initially only "fresh" or "good" flash memory cells. At this stage, the physical properties of these flash memory cells are close to perfect ones. For imprinting the watermark, repeated erase and program operations are performed. Figure 5.4 illustrates the imprinting process. Here *E* indicates a single round of erase operation, while *P* indicates a single round of program operation. The numbers on the left indicate erase and program round number. The data that is used for program operation corresponds to the content of the watermark. In our case, this information is ASCII value for "TC" which is "01010100.01000011" in binary.

The erase operation brings all the flash memory cells to logical state of '1' (i.e. removes charges from the floating gates), whereas the program operation brings the logical state of selected flash memory cells to '0' (i.e., charges the floating gates). However, a program operation affects only the flash memory cells that need to be programmed to a logical '0'. Thus, repeated erase and program operations degrade the flash memory cells selectively. After a number of repeated program and erase cycles, sufficient differences exist in physical properties between the flash memory cells that do not change their state between successive erase and program operations and the flash memory cells that change their state repeatedly between erase and program operations. We call the flash memory cells that do not change their state as “good” or ‘G’ as labeled in Figure 5.4 and we call the flash memory cells that undergo charge/discharge cycle repeatedly as “bad” or ‘B’ as labeled in Figure 5.4. The bad flash memory cells accumulate defects in the oxides as they are program-erased a number of times. These defects or the degradations in the oxide layers of “bad” flash memory cells are permanent.


Watermark: “TC” = 5443h = 01010100.01000011 b

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Initial State		0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1
1	E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	P	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1
2	E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	P	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1
3	E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	P	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1
N_{PE}	E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	P	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1
Physical State		B	G	B	G	B	G	B	B	B	G	B	B	B	B	G	G

Figure 5.4. An Illustration of watermark imprinting into a flash memory word.

Code 5.2 outlines the steps taken for imprinting and reading the watermark into and from the NOR flash memory. Reading of watermark is performed by the system designer to verify that the chips received are genuine before incorporating them into a product. This reading of watermark is not equivalent to common flash memory read operation. To read the watermark, the physical property of the memory cells needs to be extracted. And, we need to differentiate between “good” and “bad” flash memory cells.

It is a challenging task to distinguish between the “good” and “bad” flash memory cells using digital interface only. Additional challenges arise from semiconductor manufacturing processes that induces significant cell-to-cell variations. These variations need to be distinguished from those induced by the watermark imprinting procedure.

We use a partial erase operation to distinguish between the “good” and “bad” memory cells using digital interface only as discussed in 5.3.1. Code 5.2, bottom outlines the steps taken in extracting the watermark. Here, the flash segment is first erased and programmed so that all the cells are at logical ‘0’ state. Then, a partial erase operation is performed. A read is performed after the partial erase operation.

```
void imprint_flashmark(segment_address,NPE,Watermark){
  for(stress = 1; stress<=NPE; stress++) {
    fully_erase_segment(segment_address);        // Write 1s
    for(i= 1; i<=num_words; i++) {
      Program Word[i] in SegAddr with Watermark[i];
    }
  }
}
```

```
void extract_flashmark(segment_address, TPERASE){
  fully_erase_segment(segment_address);        // Write 1s
  fully_program_segment(segment_address);      // Write 0s
  partial_erase_segment(segment_address,TPERASE);  // Partial erase
  for(i=1; i<num_words; i++) {
    Read Word[i] in SegAddr;
  }
}
```

Code 5.2. Algorithm for imprinting and extracting watermark.

For the partial erase operation during the extraction phase, the partial erase time, or a window of partial erase times is determined for a family of flash memories based on its characterization results. It is assumed that this partial erase time which brings the flash segment to the state that maximizes the likelihood of extracting the watermark is determined by the manufacturer and publicly communicated to the system integrators.

5.4 Experimental Evaluation and Results

Experimental evaluation of Flashmark is performed on NOR flash memory of a family of low power microcontrollers from Texas Instruments (TI). The devices used are MSP430F5438 and MSP430F5529 microcontrollers. In the following subsections, we will focus on analysis of key parameters and design metrics such as: (a) bit error rate − the percentage of watermark bits that can be correctly retrieved using *extract_flashmark()* procedure; (b) imprint time – the time needed to imprint the watermark into a flash memory segment using *imprint_flashmark()* procedure; (c) extract time – the time needed to extract watermark using *extract_flashmark()* procedure, and (d) flash memory overhead. Section 5.4.1 presents the analysis of the bit error rate during extraction. Section 5.4.2 presents a replication based watermarking that improves the bit error rate. Finally, Section 5.4.3 discusses design metrics such as the imprint time, extract time, and overhead.

5.4.1 Bit Error Rate Analysis

For the bit error analysis of the watermark extraction procedure, a watermark that consists of upper-case ASCII characters is created – upper-case letters are replicated to create a 512-byte long watermark to fill a flash memory segment completely. Thus, the watermark data consists of 1689 set bits ('1') out of total 4096 bits.

This watermark is imprinted into a segment using the *imprint_flashmark()* procedure Code 5.2, while varying the number of PE cycles, N_{PE}, from 0 to 100 K (the endurance limit of the flash memory). At every 20 K PE cycles, the watermark is extracted using the *extract_flashmark()* procedure Code 5.2 (with a single read), but with varying T_{PERASE}. The values extracted from the flash memory segment are then compared with the original watermark to compute the bit error rate.

Figure 5.5 shows the bit error rate of the extracted watermark as a function of partial erase times, T_{PERASE}, for different stress levels. It should be noted that the extraction procedure involves full erase and full program of a flash memory segment. Thus, below certain partial erase times (~18 μs), the entire flash memory segment is expected to be read in the programmed state, i.e., '0', resulting in the error rate of 41.2%. This corresponds to the percentage of set bits ('1') in the watermark.

For unstressed segment (labeled 0 K in Figure 5.5), the bit error rate reaches the percentage of bits set to logic '0' for larger T_{PERASE} (~ 22 μs). However, for stressed flash memory segment, the values of bit error rates drop. For example, for a segment subjected to N_{PE} = 20 K, the error rate drops to as low as 19.9% for a particular partial erase time. This shows that by increasing the parameter N_{PE}, the physical difference between unstressed and stressed cells increase. This increases our ability to recover original watermark. The minimum error rates observed are 11.8 % for N_{PE} = 40 K, 7.6 % for N_{PE} = 60 K, 2.3 % for N_{PE} = 80 K and for 2.0 % for N_{PE} = 100 K.

Thus, we can conclude that higher the number of stresses is, the lower is the bit error rate when extracting the watermark. We can also observe that there is a particular window of the partial erase time that minimizes the bit error rate in the extraction procedure. This window slightly shifts to right as the number of stresses increases.

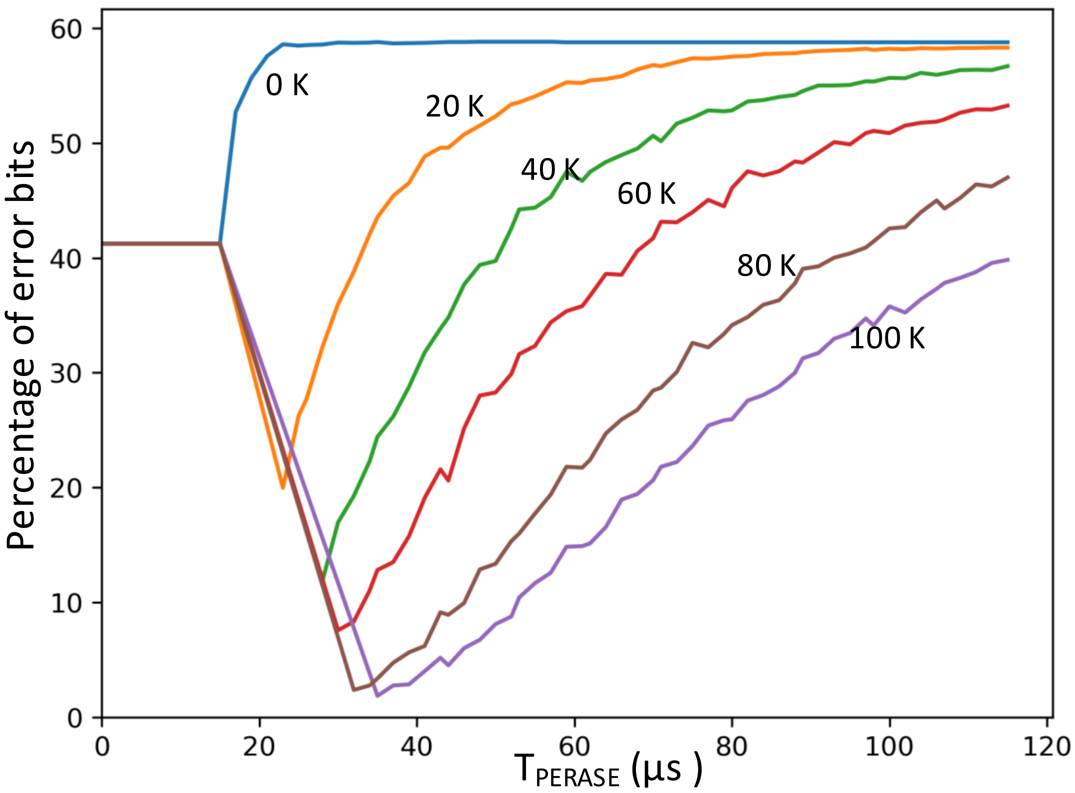


Figure 5.5. Bit Error Rate (BER) for a single-read 512-byte watermark extraction as a function of partial erase times.

5.4.2 Replication Based Watermark

Figure 5.5 shows that higher number of PE cycles (N_{PE}) improves the bit error rate during extraction. However, we would also like to have minimum value of N_{PE} and reduce the imprint time. Since the flash memory cells are affected by manufacturing variations that result in different physical characteristics of flash memory cells, random noise, charge retention effects and other physical processes, it is not feasible to achieve zero-bit error rate. Thus, as an extension to the baseline technique, we analyze replication-based imprinting.

Since watermarks require modest memory footprint, watermark data can be replicated into multiple memory locations without significant memory overhead. Figure 5.6 shows a 7-way replication of a 30-bit vector, a part of the watermark. The flash memory segment has been stressed 50 K times and the watermark is extracted from each replica using partial erase time t_{PEW} = 28 μs. The top row shows the actual watermark where the black squares represent bits at logic '1' and white squares represent bits at logic '0'. The extracted watermarks are shown in rows labeled 1 to 7. The recovered watermark constructed using majority voting is shown in the last row. This recovered watermark fully matches the imprinted watermark.

Figure 5.6 shows a closer look on bit errors from each extracted replica of watermark. For example, bits 1 and 29 in replica number 1 are errors whereas bits 1, 5, 21 and 27 are errors in replica number 3. Figure 5.6 also shows that bit errors occur more frequently on stressed bits – i.e., 'bad' bits have a higher probability of mischaracterizing than "good" bits. For example, bit number 5 is mischaracterized on replica numbers 3, 4 and 5. This observation can be utilized for further tuning of watermark extraction procedures.

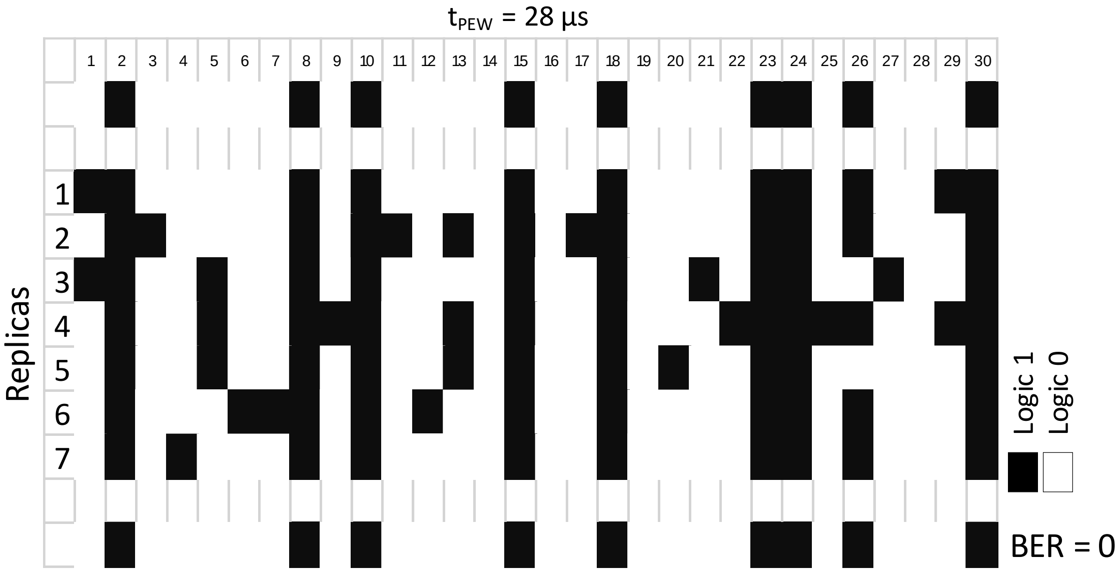


Figure 5.6. Extraction of watermark from replicated copies using majority voting.

Replication based technique helps to address the issue of inherent manufacturing variations that results in different physical characteristics among flash memory cells. These are the variations that induce cell to cell differences in seemingly similar flash memory cells.

Replicated watermarks also widens the range of T_{PERASE} for extraction as compared to the cases when no replication is used. Figure 5.7 shows the result of extracting watermarks using 3- ,5- and 7- replicas. The number of PE cycles analyzed are 40 K, 50 K, 60 K and 70 K. For the plot with 40 K PE cycles, the minimum bit error rates of 5.2%, 2.4%and 0.96% are obtained with 3, 5, and 7 replicas respectively. This is a significant improvement when compared to the bit error rate of 11.8% observed with no replication. For the case with N_{PE} = 70 K PE cycles, the watermark is fully recovered using 3-way replication (i.e. 0 bit error rate).

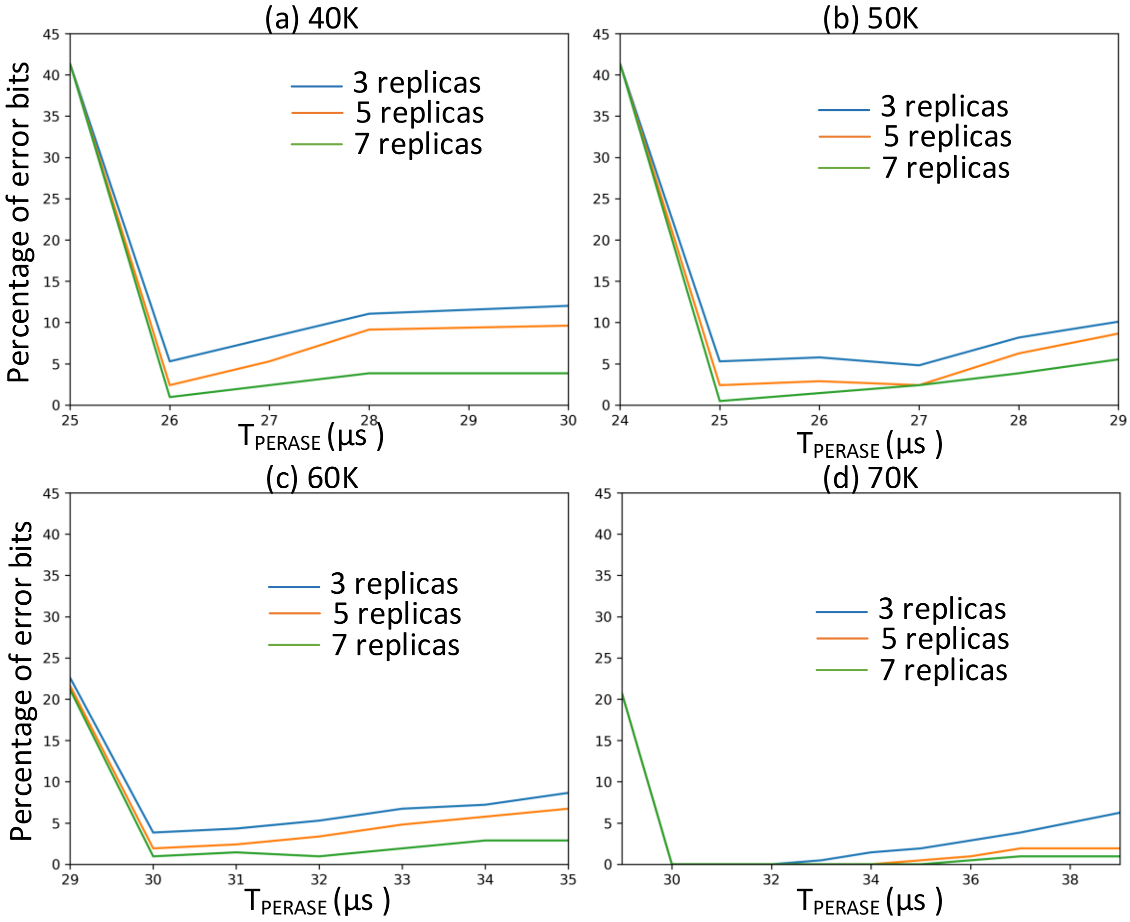


Figure 5.7. Analysis of replication based watermark extraction.

5.4.3 Overhead Analysis

The imprinting time for the watermark is directly proportional to number PE cycles, N_{PE}. A complete segment erase time is 25 ms, while time taken by program operation depends on the approach of programming. Our baseline implementation employing a full segment erase and full segment program takes ~1380 s for imprinting watermark when N_{PE} = 40 K. This time increases to 2415 s when N_{PE} = 70 K. A faster erase and program for imprinting using partial flash memory operations will significantly reduce the time. This accelerated procedure will reduce the imprint time to 387 s for N_{PE} = 40 K while the same for N_{PE} = 70 K is 678 s. However, it should be noted that the erase and program time is a function of the NOR flash memory type being used. A number of newer stand-alone NOR flash memories have significantly faster erase and program times. In addition, NOR flash memories are more resilient to stresses than NAND flash memories, thus requiring more N_{PE}.

Extraction of watermark is a faster operation. The baseline implementation using seven replicas take about 170 ms for extraction of watermark.

Since a complete segment of memory has to be allocated for watermark, a complete segment ought to be sufficient for even the cases with multiple replicas.

5.5 Concluding Remarks

Watermarking helps to differentiate between genuine products and counterfeit products. The proposed watermarking technique, *Flashmark*, imprints manufacturer information permanently into a section of flash memory by changing the physical properties of flash memory cells. By retrieving and verifying integrity of flash watermarks, system integrators and end users will be able to detect counterfeit chips before integrating them into their products. Flashmark offers following advantages over the existing counterfeit detection methods. First, imprinting and extracting watermark is done using standard system commands, thus it can be automated and implemented as a part of chip testing procedure. Next, it does not require inclusion of any extra hardware (e.g., antifuse memory or aging sensing circuitry) nor modification of chip design/fabrication process. It is universally applicable to all flash memory chips irrespective of manufacturer of type of memory. Finally, unlike PUF based systems, *Flashmark* does not require maintenance of chip specific large databases nor the system integrators need to contact the original chip manufacturer to verify authenticity of the chip.

CHAPTER 6

ENERGY SAVINGS

Modern embedded systems are resource constrained where energy-efficiency is a key design requirement. Flash memories play a central role in many such systems. Though flash memories offer a more energy-efficient alternative to hard disks, they still require a substantial amount of energy. This chapter presents a technique that reduces time and energy consumed by critical flash memory operations in ultra-low power microcontrollers by employing partial flash operations. Section 6.1 discusses the need for energy efficient flash operations in embedded systems. Section 6.2 discusses prior related efforts. The proposed partial erase and partial program based flash memory operations are discussed in Section 6.3. Section 6.4 presents the results of the experimental evaluation. It shows that the proposed technique saves 98% energy for erase operations and up to 75% energy for program operations [81]. Finally, Section 6.5 concludes the chapter.

6.1 Motivation

Energy efficiency is a key design requirement for many resource-constrained embedded systems that are commonly used in wireless sensor networks, wearable or implanted electronics and IoT. Many of such systems are battery powered. Energy efficient designs enable a smaller form factor, longer operating times and reduce the operating costs by eliminating need for frequent battery changes. This might especially be important in cases of implanted devices that may require surgical procedure for battery replacement. Modern ultra-low-power microcontrollers are typically used in these systems to perform sensing, processing, communication, and actuation tasks

Ultra-low-power microcontrollers typically integrate flash memory into their system on a single chip. Flash memory serves as a non-volatile storage that contains system firmware and constants and behaves as read only memory during normal operation. However, to support frequent firmware updates or to prevent loss of critical application data in case of power loss, flash memory in modern microcontrollers support in-system programming (ISP), rather than through external JTAG interfaces. As discussed in Section 3.1, flash memory controller in modern microcontrollers enable the erase and program operations of flash memory from within the system.

Flash memory program operation involve moving electrons into the floating gate and erase operation involve removing electrons from the floating gate. These operations are power hungry and rely on internal charge pumps to generate high voltages needed for the movement of electrons. Thus, finding ways to minimize energy consumed by flash memory operations is very beneficial for the systems that frequently update and use flash memory for storing critical data.

6.2 Related Works

In ultra-low-power microcontrollers, the minimum voltage required for proper operation of a processor core is lower than the minimum voltage required for proper flash memory operations. An approach to lower the supply voltage to save energy consumption was proposed by Salajegheh et al. [82], [83]. The supply voltage is lowered such that the minimum voltage requirement of the processor is met, but the supply voltage is lower than that is required for reliable operations of flash memory. To remedy the possible loss of information, they employ one of the following techniques: (a) repeated in-place write operation, (b) multiple places write operation and (c) a RS-Berger code of data. They report energy savings of up to 34% for in-place write operations.

Tseng et al. [84] propose the use of power supply voltage scaling in flash memory chips to save energy. They explore trade-offs between energy savings achieved by reducing power supply and the increased bit error rate and find an optimal design point. Underpowering of flash memory can save up to 45% of the energy consumed at the cost of increased bit error rate using a dynamic voltage scaling mechanism. In MLC NOR flash memory, the latency and energy consumed by flash write operations are dependent on the bit pattern of the data written. Thus, design of codewords to reduce the frequency of certain power hungry bit patterns ('01' and '10') and increase the frequency of other bit pattern is proposed by Papirla et al. [85]. They report reduction of energy consumption by 33% and latency of flash memory operations by 24%.

A lazy amnesic compression based data storage technique was proposed by Nath for storage of data in flash memory [86]. The energy required for a flash memory write is reduced by using lossy compression, where the compression rate is adjusted based on the age of data i.e. fidelity of "old" data is lower than the fidelity of new data. Mathur et al. introduce Capsule [87], a log structured object storage system for flash memories that supports fine-grained allocation of space for storage objects, such as, streams, files, arrays, and queues.

All the proposals discussed above demonstrate significant potential in reducing the total energy consumed by flash memory operations. However, they introduce significant overhead in time, compute resources, and storage space. Our proposal for saving energy and time employs partial flash memory operations, i.e. the flash memory erase and program operations are replaced by partial flash erase and program operations, respectively. It does not incur any additional latency or memory overhead.

6.3 Proposed Energy Efficient Flash Memory Operation

The energy efficient flash memory operations proposed in this chapter are based on the partial flash memory operations. Thus, an extensive analysis is performed to characterize partial flash erase and partial program operations. Section 6.3.1 presents the characterization results. These results are used to develop partial erase and partial program operations that are described in Section 6.3.2.

6.3.1 Characterization of Partial Flash Operations

The steps carried out to characterize partial flash erase operations are presented in Code 4.1. First, a flash memory segment is fully erased (all cells are set at logic '1'). This is followed by a full segment program operation (all cells at set at logic '0'). Then an erase operation is initiated. This erase operation is prematurely aborted after time T_{PERASE} by setting the emergency exit bit (EMEX, Code 3.3) to achieve partial erase operation. After the partial erase operation, each words in the flash memory segment are read N_R times and each bits are characterized as either programmed, erased or unstable (Code 4.1). The same process is repeated by varying the partial erase time, T_{PERASE}, from 0 to T_{ERASE} (the nominal erase time of a segment).

A similar experiment is performed to characterize the partial program operation (program a word). In this case, first a selected segment is fully erased. Then, for each word in the segment, the program operation is initiated and subsequently aborted after a partial program time, T_{PPROG}. After the partial program operation, each word in the flash memory segment is read N_R times and each bit is characterized as either programmed, erased or unstable (Code 4.1). The experiment is repeated by varying T_{PPROG} from 0 to T_{PROG} (the nominal program time for a word).

Figure 6.1(a) and Figure 6.1(b) show the results of the partial erase characterization and the partial program characterization, respectively. The x-axes show the partial erase times and the partial program times. Figure 6.1(a) is similar to the partial erase characterization results shown in Figure 4.3. For a small T_{PERASE} (T_{PERASE} ~ 0 μs), the erase operation is aborted promptly. Expectedly, all 4,096 cells in a segment are characterized as stable programmed cells (indicated by label Stable 0s). As T_{PERASE} is increased, the number of stable programmed cells starts decreasing and the number of stable erased cells starts increasing. The plot shows that the majority of cells change their state in a narrow time window. We can observe that by T_{PERASE} ~ 50 μs, all 4,096 cells in a segment are in the erased state (indicated by label Stable 1s), although the nominal erase time per specification ranges between 23-32 ms. Thus, the partial erase time of T_{PERASE} = 50 μs appears to be sufficient to completely erase the flash segment.

One may argue that although flash cells appear to be erased, their threshold voltage may not quite correspond to the nominal V_{THE}, that is, they may be in a weak erased state. Fortunately, the flash controller supports so-called marginal read operations that allow us to identify whether erased cells are in the weak or strong erased or programmed states. Despite T_{PERASE} being significantly shorter than the nominal erase time, the marginal reads do not report weakly erased bits.

Similarly, Figure 6.1(b) shows that all the flash memory cells in a segment remain in erased state (Stable 1s) for small partial program times, $T_{PPROG} \sim 0$ µs. However, with an increase in T_{PPROG}, the number of flash cells in erased state starts decreasing and the number of flash cells in programmed state (Stable 0s) increases. All the flash memory cells are in the programmed state at $T_{PPROG} \geq 27$ µs although the nominal word program time is between 64-85 µs [88]. To verify that transition of flash cells is complete, marginal reads are used to identify that no cells are weakly programmed cells.

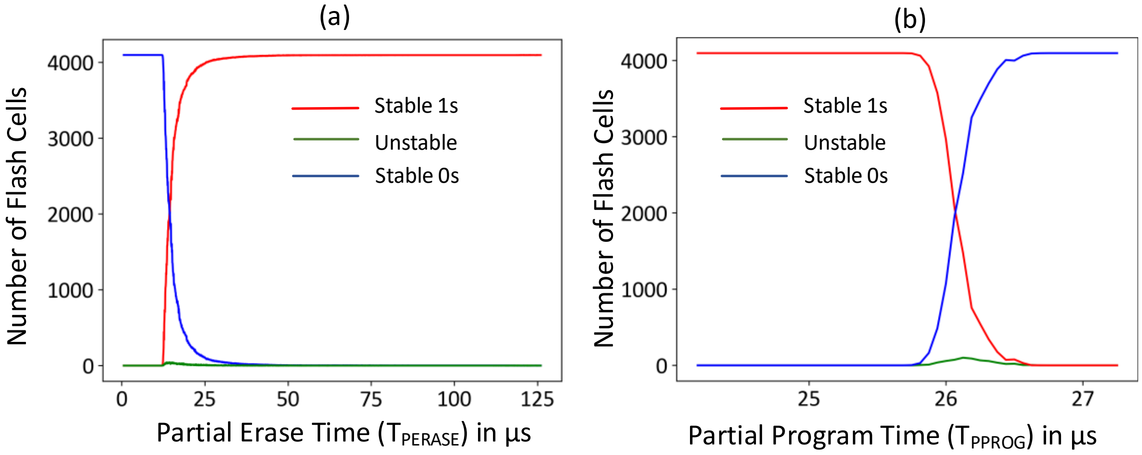


Figure 6.1. State of flash segment cells as a function of (a). Partial erase times; and (b). Partial program times.

Based on these observations, we propose to use the partial erase and the partial program flash operations instead of nominal erase and program operations. This approach reduces the execution time of programs that require frequent in-system flash operations as well as reduce the energy consumed by these tasks.

6.3.2 Implementation Programs

Code 6.1 shows a subroutine that implements the proposed partial erase operation of a segment and Code 6.2 shows a subroutine that implements the proposed partial program operation of a word. Both are tailored for the MSP430 flash-based microcontrollers. These subroutines are later used for time and energy profiling in Section 6.4.2.2.

The segment partial erase subroutine configures the flash memory controller registers (line 2, 3 and 4 in Code 6.1). After the dummy write to initiate erase operation (line 5), the code waits for a duration defined by optimal segment erase time T_{PE}. Then, emergency exit (EMEX) bit is set to abort the erase operation. The parameter T_{PE} is obtained from the characterization described in Section 6.3.1.

```
1. void partial_erase_segment(FlashAddress, T_PE){
2.   while(FCTL3&BUSY);        // wait until busy
3.   FCTL3 = FWPW;             // flash password
4.   FCTL1 = FWPW+ERASE;       // configure segment erase
5.   *FlashAddress = 0;        // dummy write to initiate erase operation
6.   __delay_cycles(T_PE);     // wait for T_PE
7.   FCTL3 = FWPW+EMEX;        // emergency exit to terminate erase operation
8.   while(FCTL3&BUSY);        // wait until busy
9.   FCTL1 = FWPW;             // clear erase operation
10.  FCTL3 = FWPW+LOCK;        // lock the flash memory
11.}
```

Code 6.1. Subroutine for partial erase of a segment.

For a partial program of a word, first the flash controller registers are configured for write operation (line 2 and 3 of Code 6.2). Then, the data *WordData* is written to the address of the word (line 4 of Code 6.2). This initiates a word program operation. However, unlike the nominal program operation, the code presented in Code 6.2 waits for a time defined by optimal word program time T_{PP} and sets EMEX bit to abort the program operation (line 6 of Code 6.2). The parameter T_{PP} is obtained from the characterization result discussed in Section 6.3.1. A complete segment can be programmed using the subroutine presented in Code 6.2 for each word in the segment.

```
1.  void partial_program_word(SegmentAddress, WordIdx, TPP, WordData){
2.    FCTL3 = FWPW;                         // flash password
3.    FCTL1 = FWPW+WRT;                     // configure word write
4.    *(FlashAddress+WordIdx) = WordData;  //  dummy write to initiate word
                                             program operation
5.    __delay_cycles(TPP);                  // wait for TPP
6.    FCTL3 = FWPW+EMEX;                    // emergency exit to terminate erase
                                             operation
7.    while(FCTL3&BUSY);                    // wait until busy
8.    FCTL1 = FWPW;                         // clear erase operation
9.    FCTL3 = FWPW+LOCK;                    // lock the flash memory
10. }
```

Code 6.2. Subroutine for partial program of a word.

6.4 Experimental Evaluation and Results

The experimental environment is based on MSP430 family of microcontrollers from Texas Instruments, namely MSP430F5438 and MSP430F5529 at room temperature and under nominal power supply of $V_S = 3.3$ V. Section 6.4.1 presents the evaluation of optimal flash erase time (T_{PE}) and optimal flash program time (T_{PP}) using the characterization discussed in Section 6.3.1 on multiple segments from different chips. Section 6.4.2 discusses the energy profiling set-up used (Section 6.4.2.1) and quantifies the energy savings achieved by the proposed technique (Section 6.4.2.2).

6.4.1 Evaluation of Optimal Flash Operation Parameters (T_{PP} and T_{PE})

We extend the process described in Section 6.3.1 to characterize flash memory behavior in presence of partial erase and program operations. Table 6.1 shows the results of the characterization performed on five segments (Seg 0 – Seg 4) from four different samples of MSP430F5438 chips (Chip 0 – Chip 3). Each segment in each chip is profiled to determine the minimum partial program time (t_{PPROG}) when all the bits within a word that needs to be programmed are indeed programmed. Similarly, each segment is profiled to find the minimum partial erase time (t_{PERASE}) when all the bits in the segment are indeed erased.

The minimum partial program times (t_{PPROG}) are fairly uniform across different segments and chips. They range between 26 μs and 27 μs, compared to 64-85 μs nominal program time per reference manual [88] or to ~65 μs measured in our experiments. Thus, these results indicate that slightly over 1/3rd of the nominal program time is sufficient to complete a program operation. We take a conservative approach and set an optimal partial program time T_{PP} = 28 μs in further experiments.

The partial erase times (t_{PERASE}) are significantly lower than the nominal erase times. However, partial erase time vary across different segments within a chip and across different chips. For example, for Chip 2 the observed partial erase times varied from 57 μs to 113 μs while that for Chip 3 varied from 34 μs to 46 μs. We take a conservative approach and choose an optimal erase time T_{PE} = 115 μs for further experiments.

MSP430F5438		t_{PPROG}	Optimal	Nominal	MSP430F5438		t_{PERASE}	Optimal	Nominal
Chip	Seg	(µs)	(T_{PP})	Time	Chip	Seg	(µs)	(T_{PE})	Time
0	0	27	28 µs	64-85 µs	2	0	113	115 µs	23-32 ms
0	1	27			2	1	57		
0	2	26			2	2	57		
0	3	27			2	3	80		
0	4	27			2	4	57		
1	0	26			3	0	46		
1	1	26			3	1	34		
1	2	26			3	2	34		
1	3	26			3	3	35		
1	4	26			3	4	35		

Table 6.1. Result of partial flash operation characterization for MSP430F5438 microcontrollers.

The result of similar characterization performed for MSP430F5529 microcontrollers is shown in Table 6.2. Here, we find that the optimal programming time for a word, T_{PP}, to be 16 µs (the nominal programming time T_{PROG} is 65-85 µs) while the optimal erase time of a segment, T_{PE}, is determined to be 73 µs (the nominal erase time is 23 - 32 ms.

MSP430F5529		t_{PPROG}	Optimal	Nominal	MSP430F5529		t_{PERASE}	Optimal	Nominal
Chip	Seg	(µs)	(T_{PP})	Time	Chip	Seg	(µs)	(T_{PE})	Time
0	0	15	16 µs	64-85 µs	2	0	50	73 µs	23-32 ms
0	1	15			2	1	72		
0	2	15			2	2	18		
0	3	15			2	3	15		
0	4	15			2	4	18		
1	0	15			3	0	15		
1	1	15			3	1	15		
1	2	15			3	2	19		
1	3	15			3	3	18		
1	4	15			3	4	15		

Table 6.2. Result of partial flash operation characterization for MSP430F5529 microcontrollers.

6.4.2 Energy Profiling

The test programs developed in Section 6.3.2 is used to analyze the energy consumption by nominal flash memory operations as well as partial flash memory operations using optimal parameters from Section 6.4.1. Section 6.4.2.1 describes the energy profiling setup used. Section 6.4.2.2 presents the compares the energy consumed by nominal partial flash operations and quantifies the energy savings.

6.4.2.1 Energy Profiling Setup

The development board with the MSP430 microcontroller (Section 3.1.4) is connected to a setup for runtime energy profiling that consists of a National Instruments (NI) PXIe-4154 battery simulator, an NI PXIe-6361 data acquisition card and a workstation. This is a set up for automated measurement of energy consumed by embedded computing platform [89].

The battery simulator supplies power to the development board and samples the current drawn at a rate of $F_S = 100{,}000$ samples per seconds. The energy consumed is computed using Equation 3. Here, M is the number of samples collected during an operation, V is the power supply, and Δt is the sampling period, i.e., $\Delta t = \frac{1}{F_S}$.

$$E = \sum\nolimits_{i=0}^{M} V \cdot I_S \cdot \Delta t \qquad \text{Equation 3}$$

6.4.2.2 Energy Profiling Results

Figure 6.2 shows the current drawn by a nominal flash memory operation (indicated by the blue lines) and the current drawn by partial flash memory operations (indicated by the orange lines). The baseline current drawn by the microcontroller when idle is ~1 mA. When a flash memory operation is initiated, a steep increase in current is observed. This current remains high for the duration of the operation. For example, for Figure 6.2(a), the nominal time for erase operation T_{ERASE} ~27 ms. The current drawn is ~4 mA for the duration of nominal erase operation. For a partial erase operation, the orange line shows a sudden spike of T_{PE} ~115 µs. Although the duration of the two erase operations are vastly different, the amplitude reaches ~4 mA in both cases.

The total energy consumed by the nominal flash segment erase operation, on average, is ~258.6 µJ. The partial erase operation that achieves the same result, however, consumes only 3.3 µJ of energy. Thus, the proposed partial erase operation saves over 98% of energy consumed by the nominal erase operation.

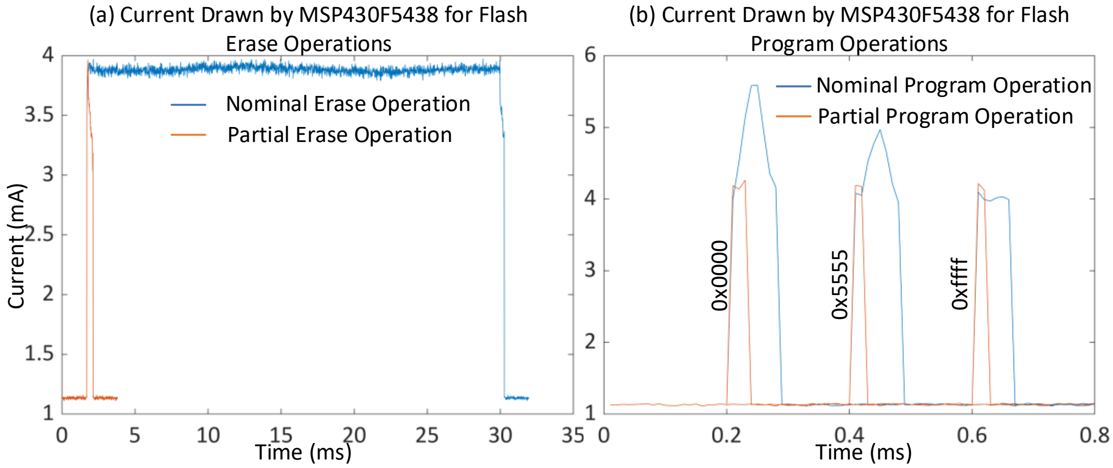


Figure 6.2. Current drawn by a family of MSP430 microcontroller for flash memory operations.

The current drawn by the program operation is shown in Figure 6.2(b). Here the current drawn during programming of a 16-bit word is shown. We used a sequence of three data patterns, namely, a data pattern with all 0s (0x0000), a data pattern with all 1s (0xffff), and a data pattern with alternate 1s and 0s (0x5555). Since a flash memory segment is erased before the program operations are carried out, the data pattern with all 0s have to program all the bits within each word, whereas in case of the data pattern with all 1s, no bit is actually programmed. These three sequences of data patterns are labeled as 0x0000, 0x5555 and 0xffff in Figure 6.2(b). The current profiles indicate that the energy consumed for both nominal and partial flash memory operations are data dependent.

The comparative analysis of the energy consumed by the program operations with different data patterns is shown in Table 6.3. Since the program operation of a segment in NOR flash memory is achieved by programing each word in the segment, we make a comparison of the energy consumed by word program operation as well as the segment program operation. The nominal program operation measurement presented, however, uses a block-wide program approach, which is a more optimized programming operation (Section 3.1.2). Block-wide program operation cannot be used in conjunction with EMEX signal, so our proposed technique uses the word-by-word programming approach. The energy savings made are still quite significant. This value ranges from 24% when a complete segment is programed using 0xffff pattern to 64.2% when writing all zeros data pattern.

Word Programming	Energy Consumed (μJ)		Savings (%)	Segment Programming	Energy Consumed (μJ)		Savings (%)
	Nominal	Partial			Nominal	Partial	
0x0000	9	3	66.7	0x0000	127	45.5	64.2
0x5555	8	2	75	0x5555	92.3	43.6	52.8
0xffff	5	2	60	0xffff	53.9	40.8	24.3

Table 6.3. Energy consumed for word and segment program operations.

6.4.3 Stress Analysis

NOR flash memory can sustain 100,000 program-erase (PE) cycles before it permanently fails. To test the robustness of the parameters (T_{PP} and T_{PE}) found through the characterization in Section 6.4.1, we stress the flash memory segments by performing repeated program-erase (PE) operations. At every 10,000 PE cycles we characterize the stressed flash memory segments to determine the optimal erase and program times. This process is repeated until the maximum endurance (100,000 PE cycles) of the flash memory segment is reached.

Figure 6.3(a) and Figure 6.3(b) show the optimal partial erase times as a function of PE cycles applied to flash memory segments. These plots present multiple segments from two sample chips of the MSP430F5438 family. The results indicate that the optimal partial erase time increases with an increase in the stress level. The optimal partial erase time for a fresh flash memory segment is below 100 μs, whereas the optimal partial erase time is ~ 925 μs for a flash memory segment that is exposed to 100K PE cycles. Figure 6.3c and Figure 6.3d show the optimal partial program times collected across multiple segments from two sample chips. The optimal partial program times for MSP430F5438 family of microcontrollers is not significantly affected by the number of PE cycles. Thus, a conservative approach of using an optimal partial program time of 28 μs would work irrespective of the stress level.

Similar analysis is repeated for MSP430F5529 family of microcontrollers. The results of the analysis are presented in Figure 6.4. Figure 6.4a and Figure 6.4b show the change in the partial erase times as a function of PE cycles applied to the flash memory segments. The change follows a similar trend as the change of optimal partial erase times in MSP430F5438. However, some irregularities in the trend can be seen. Thus, a conservative estimate based on characterization results can be used for a fully stressed flash memory segment. The optimal partial program times, however, are not affected.

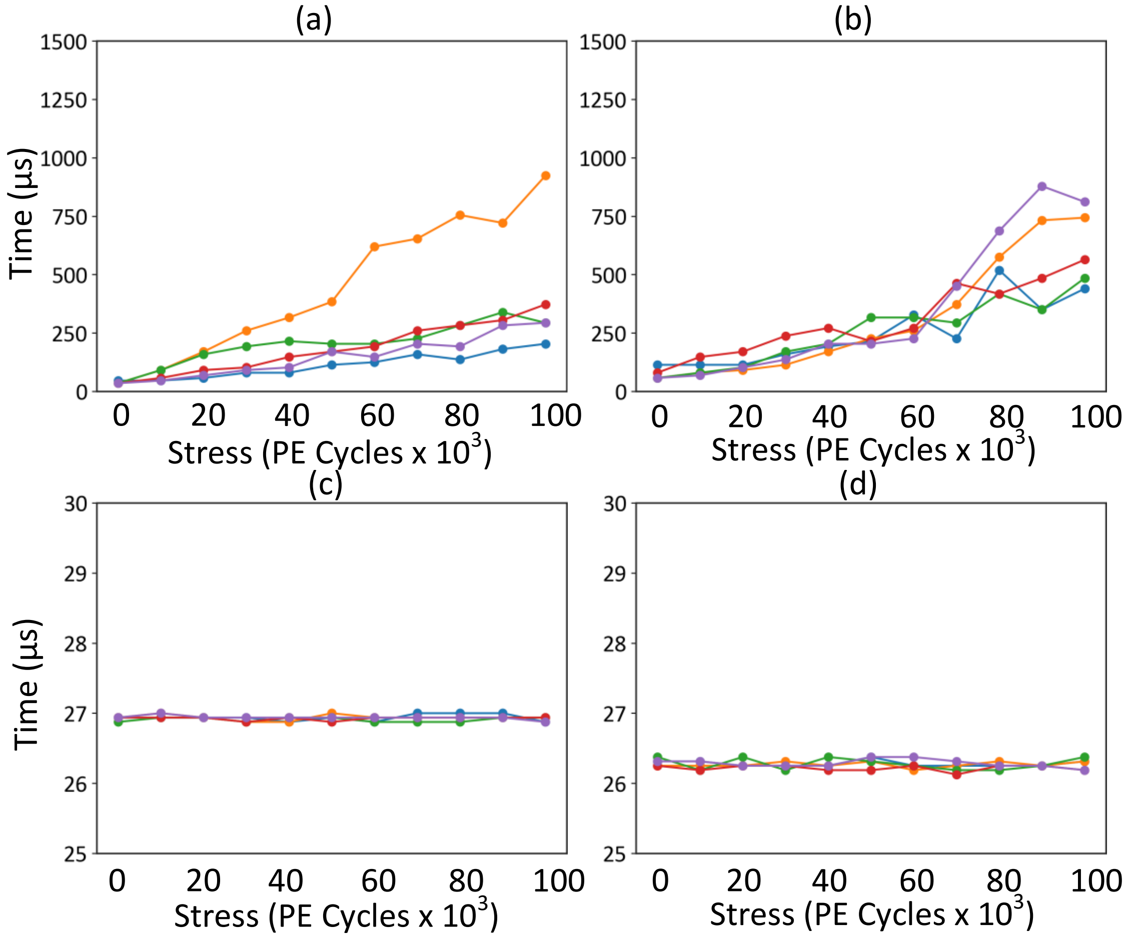


Figure 6.3. Partial erase (a,b) and partial program (c,d) as a function of PE cycles for MSP430F5438.

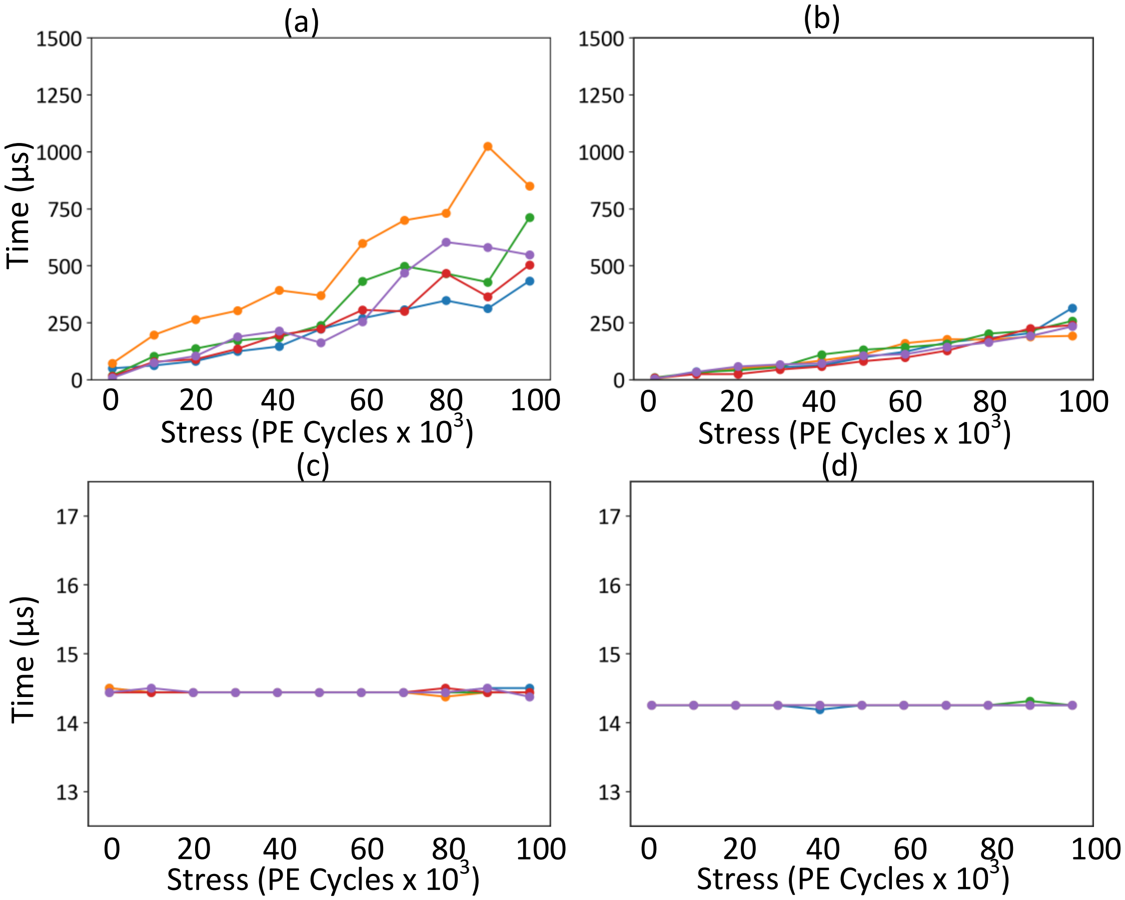


Figure 6.4. Partial erase (a,b) and partial program (c,d) times as a function of PE cycles for MSP430F5529.

6.5 Concluding Remarks

Partial flash erase and partial flash program operations can be used to replace nominal erase and program operation without any loss of information. This technique is implemented and evaluated on a family of commercial microcontrollers to demonstrate savings in energy of over 98% for segment erase operation and from 24-64% of savings for segment program operations. The proposed technique does not require any hardware changes and can be solely implemented in firmware.

CHAPTER 7

3D MLC NAND FLASH MEMORY CHARACTERIZATION

Continued scaling of technology node coupled with integration of multiple bits per flash memory cell has caused a significant concern in reliability and endurance of 2D NAND flash memories. Further scaling of 2D memories below 30 nm causes structural and mechanical integrity issues in addition to issues related to charge leakage, charge migration, and cell to cell interference [90]. 3D NAND flash memories allow scaling in the vertical direction. This leads to improvements in storage density, while employing larger flash memory cells [91], [92]. Consequently, 3D NAND flash memories storing multiple bits per cell are more robust than equivalent 2D NAND flash memories; for example, the endurance limit, i.e., the maximum number PE cycles before permanently failing, for 2D TLC NAND flash memory is 1,000 PE cycles, whereas the endurance limit of 3D TLC NAND flash memory is 10,000 PE cycles.

Recent research efforts have investigated the properties of 3D NAND flash memories to further enhance their performance and reliability. The reliability of 3D NAND flash memory pages, measured in bit error rates, varies across different layers [38], [91], [93], [94]. Likewise, the pages in the same physical layer have similar properties [95]. Similarly, the performance, measured in page program time, varies among the pages in a single block [91], [94]. These characterizations suggest ways to improve the reliability and performance by exploiting physical properties of the chip. Other unique properties of 3D NAND flash memories are also explored in a quest to improve their reliability. For example, oxide charge traps in 3D NAND flash memories are mended on its own during the idle time between program and erase operations. This recovery can be accelerated by exposing chips to high temperatures [96].

The properties discussed above are important for designing Flash Translation Layer (FTL). FTL is a system software that sits between the file-system and the NAND flash chips [97]. NAND flash memory chips, that are used for mass storage purposes in solid-state drives (SSDs) and other electronic devices, are interfaced to the host computer through the FTL. FTL hides the details of interfacing, and is also responsible for address translation, error management, garbage collection, bad block management, wear leveling, and concurrency management.

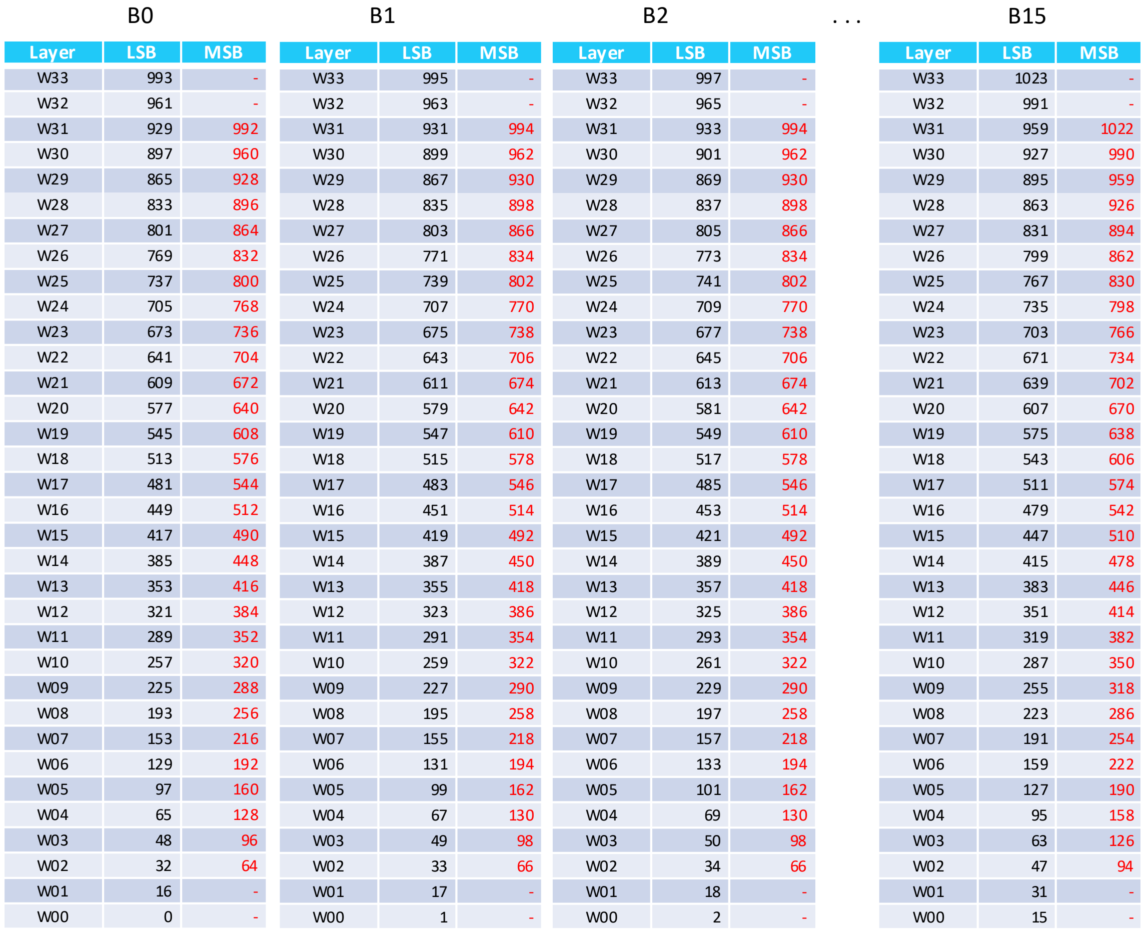
In this chapter, we characterize a family of 3D NAND flash memory chips. The characterization reveals the internal details of the flash memory chip. The characterization is performed using the FPGA based setup discussed in Section 3.2. Section 7.1 presents the details of 3D MLC NAND flash chips used. Section 7.2 presents the results of timing characterizations performed. Finally, Section 7.3 discusses the implication of the characterization results and concludes the chapter.

7.1 3D MLC NAND Flash Memory

The manufacturer part number of the NAND flash memory used for characterization is MT29F256G08CBCBBWP. This family of 3D NAND flash memory chips are produced by Micron Technology Inc. This flash memory chip has 38 physical layers (tiers) and is composed of floating gate cells [92], [94]. Of the 38 layers, 32 layers are active layers while 6 layers are dummy layers. The 6 dummy layers are organized into two groups of 3 layers each. The dummy layers are placed at the top and bottom of the

32 active layers. All 32 active layers and 2 closest dummy layers (1 from top and 1 from bottom of 32 active layers) are used for storing data. The 2 dummy layers as well as one active layer from top and one active layer from bottom are reserved for storing data in the SLC mode only. The remaining 30 active layers are made of MLC cells.

A flash block consists of 1,024 pages that are organized into 16 sub-blocks, marked as B0, B1, up to B15 as shown in Figure 7.1. The 34 layers that are used for storing data are labeled as W01-W33. Layers W00, W01, W32, and W33 can hold data only in the SLC mode (W00 and W33 are so called dummy layers), creating unshared or LSB-only pages. The layers W02 to W31 can hold data in the MLC mode, holding data that is logically organized into two shared pages: an LSB (Least Significant Bit) and an MSB (Most Significant Bit) page. In Figure 7.1, LSB page indices are given in the left column in black and MSB page indices are written in the right column in red for each layer and sub-block.



B0			B1			B2			. . .	B15		
Layer	LSB	MSB	Layer	LSB	MSB	Layer	LSB	MSB		Layer	LSB	MSB
W33	993	-	W33	995	-	W33	997	-		W33	1023	-
W32	961	-	W32	963	-	W32	965	-		W32	991	-
W31	929	992	W31	931	994	W31	933	994		W31	959	1022
W30	897	960	W30	899	962	W30	901	962		W30	927	990
W29	865	928	W29	867	930	W29	869	930		W29	895	959
W28	833	896	W28	835	898	W28	837	898		W28	863	926
W27	801	864	W27	803	866	W27	805	866		W27	831	894
W26	769	832	W26	771	834	W26	773	834		W26	799	862
W25	737	800	W25	739	802	W25	741	802		W25	767	830
W24	705	768	W24	707	770	W24	709	770		W24	735	798
W23	673	736	W23	675	738	W23	677	738		W23	703	766
W22	641	704	W22	643	706	W22	645	706		W22	671	734
W21	609	672	W21	611	674	W21	613	674		W21	639	702
W20	577	640	W20	579	642	W20	581	642		W20	607	670
W19	545	608	W19	547	610	W19	549	610		W19	575	638
W18	513	576	W18	515	578	W18	517	578		W18	543	606
W17	481	544	W17	483	546	W17	485	546		W17	511	574
W16	449	512	W16	451	514	W16	453	514		W16	479	542
W15	417	490	W15	419	492	W15	421	492		W15	447	510
W14	385	448	W14	387	450	W14	389	450		W14	415	478
W13	353	416	W13	355	418	W13	357	418		W13	383	446
W12	321	384	W12	323	386	W12	325	386		W12	351	414
W11	289	352	W11	291	354	W11	293	354		W11	319	382
W10	257	320	W10	259	322	W10	261	322		W10	287	350
W09	225	288	W09	227	290	W09	229	290		W09	255	318
W08	193	256	W08	195	258	W08	197	258		W08	223	286
W07	153	216	W07	155	218	W07	157	218		W07	191	254
W06	129	192	W06	131	194	W06	133	194		W06	159	222
W05	97	160	W05	99	162	W05	101	162		W05	127	190
W04	65	128	W04	67	130	W04	69	130		W04	95	158
W03	48	96	W03	49	98	W03	50	98		W03	63	126
W02	32	64	W02	33	66	W02	34	66		W02	47	94
W01	16	-	W01	17	-	W01	18	-		W01	31	-
W00	0	-	W00	1	-	W00	2	-		W00	15	-

Figure 7.1. Page indices within a flash memory block in MLC mode.

An erase operation of the 3D NAND flash memory affects an entire flash block of 1,024 pages. For program operation, the pages must be programmed in an order starting from lowest page index to the highest page index. LSB pages must be programmed before the corresponding MSB pages are programmed. A premature termination of a program operation in any of the LSB or MSB pages may corrupt data in the corresponding shared page.

The flash chip can also be configured to work in the SLC mode only by using an ONFI command SET FEATURE. In the default MLC mode, 10-bits are available to address each of the 1024 pages inside a block. In the SLC mode, only 9-bits are made available to address 512 pages inside a block.

7.2 Timing Characterization of 3D NAND Flash Memory

Timing analysis offers insights into the latency of flash memory operations. Flash memory program and read operations are performed on granularity of a page. Thus, we characterize page program and read operations. These characterization results can be used by the FTL to employ data placement algorithms that will increase effective throughput. Section 7.2.1 and Section 7.2.2 present the results of timing characterization of page program operations in the MLC and SLC modes, respectively. Section 7.2.3 characterizes page read operation times.

7.2.1 Page Program Times in MLC Mode

The program operation of each page, regardless of the type, follows a sequence of operations according to the ONFI standard. To program a complete page, the host first issues a command for a page write, then sends the page address, and writes the data into a data register inside the flash memory chip. Once the entire page data is loaded into the data register, a program confirm command (0x10) is issued to initiate the page program operation. During this operation the content of the data register is programmed into the NAND flash array. The flash memory chip indicates the completion of the program operation by asserting the Ready/Busy (R/B#) signal. The host can check the status of the program operation (success or failure) by using an ONFI command to read the status register of the flash memory chip.

We perform a set of experiments to measure the time taken by the program operations. For this purpose, first a fresh block is erased (all data bytes read as 0xff) and then each page in the block is programmed (all data bytes read as 0x00). We record the time taken to program the data from the data register into the NAND array. This time corresponds to the period between issuing the command 0x10 and the completion of program operation indicated by setting the R/B# pin. We refer to this time as the *page program time*.

Figure 7.2 shows page program times for all 1024 pages in a representative block. The horizontal axis represents the page indices (indexed 0 to 1023) in a block. The vertical axis represents the program time in microseconds. Each dot plotted represents the program time for the page whose index is represented in horizontal axis.

Three groups of pages can be distinguished based on the program times of the pages in a block: (a) Unshared and LSB Pages whose page program times fall in the range from 490 to 600 μs; (b) MSB Pages whose page program times fall in the range from 1280 to 1602 μs; and (c) Unshared and LSB Pages whose page program times fall in the range from 1140 to 1350 μs.

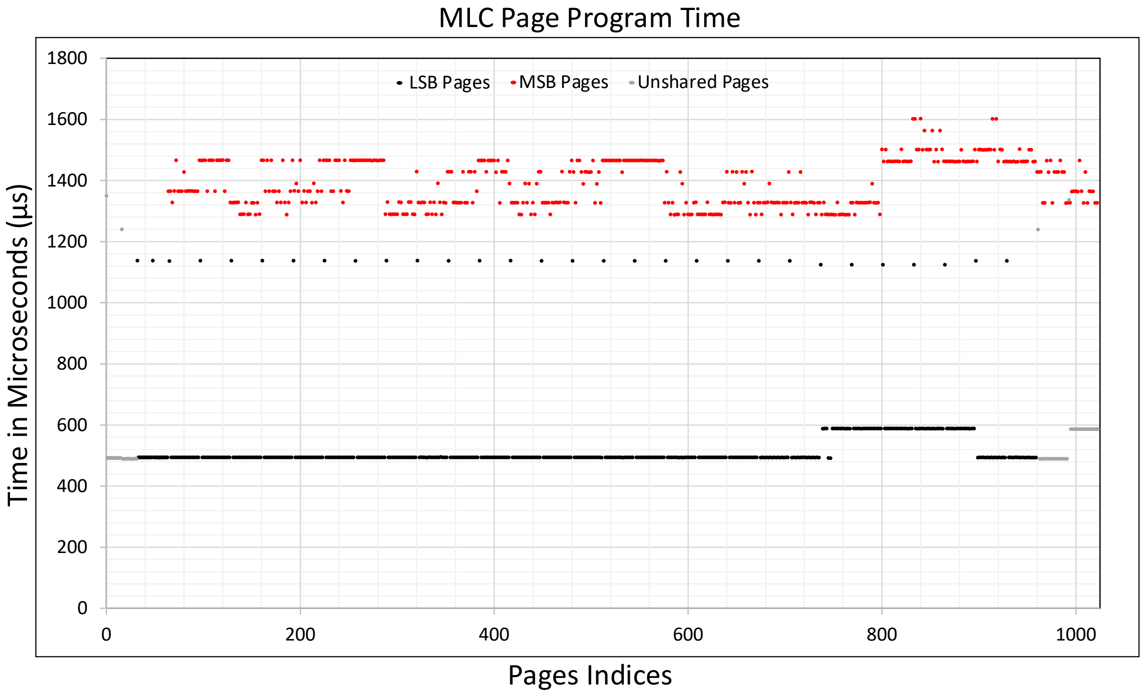


Figure 7.2. Page program times as a function of page indices for all 1024 pages in a block.

Unshared Pages. Figure 7.3 shows the program times for the unshared pages only. From these results we can draw the following observations. The first page to be programmed in a group of 16 pages that share the same wordline (belong to a single layer), generally takes more time than other pages after an erase operation. This is due to the need to charge parasitic capacitances presented by the control gates of all cells on the shared wordline (and the same physical layer). The subsequent pages in the given layer do not see this additional latency because parasitic capacitances remain charged. Thus, pages 0 and 993 in layers W00 and W33 take ~1350 µs, whereas pages 16 and 963 in layers W01 and W32 take ~1240 µs. The subsequent pages within a sub-block take ~490 µs.

The pages in the topmost layers, W33, require more time to be programmed, ~587 µs. One possible explanation could be as follows. The flash cells in this layer are typically exposed to higher voltages during the erase cycles – they are closest to the bit line transistors, so they see the highest voltage on their terminals. There is a voltage drop across the NAND array, so the cells at the edges of the NAND array see higher voltages and their oxide layers are exposed to higher levels of stress. Consequently, the distribution of the V_{TH} in the erased state of the cells in this layer is shifted far to the left, thus requiring longer program times to transition the cells to the programmed state.

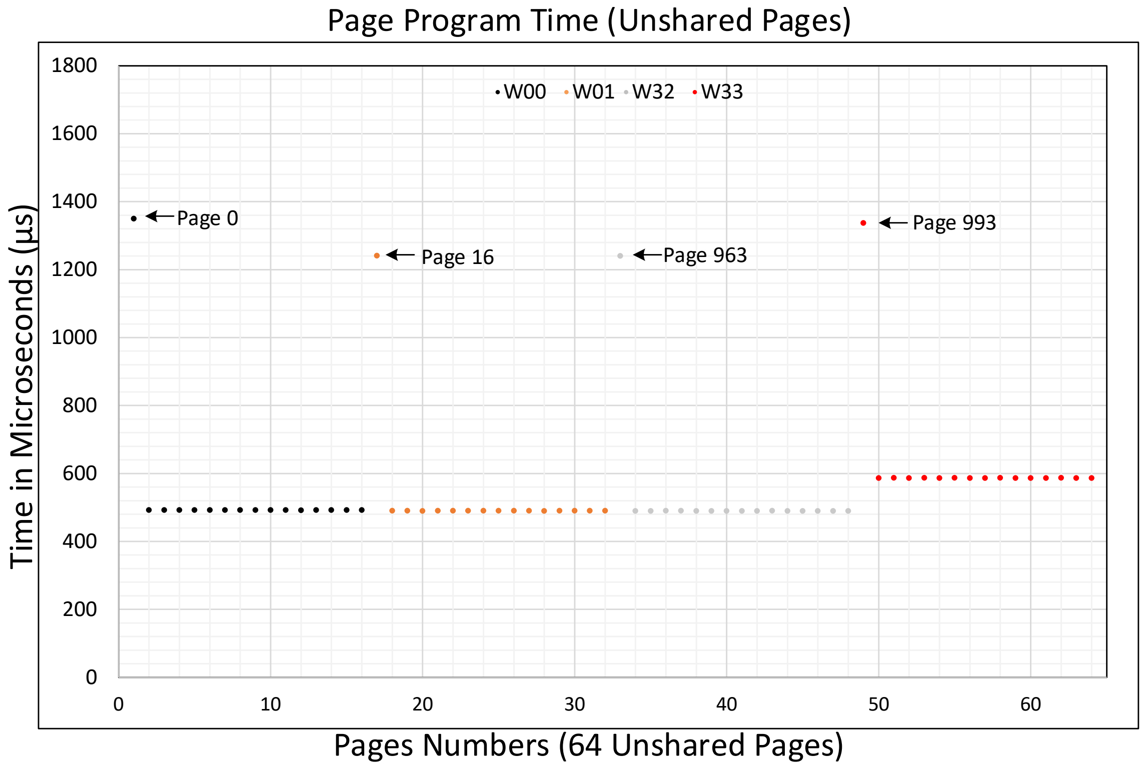


Figure 7.3. Page program times of 64 unshared pages in a block.

Shared LSB Pages. Figure 7.4 shows the program times for shared LSB pages. Similarly to the unshared pages, the first page in a given layer takes ~1138 µs, whereas the remaining pages on the given layer take ~494 µs. The times appear to be uniform across the inner layers. There is a notable exception in one area where the programming of the remaining pages (not first page) is close to ~588 μs. Based on our analysis these LSB pages are located in layers W25, W26, W27, W28, and W29 (5 layers in total). The remaining two layers with shared pages W30 and W31 require ~494 μs. This anomaly could be caused by specific properties of the manufacturing process.

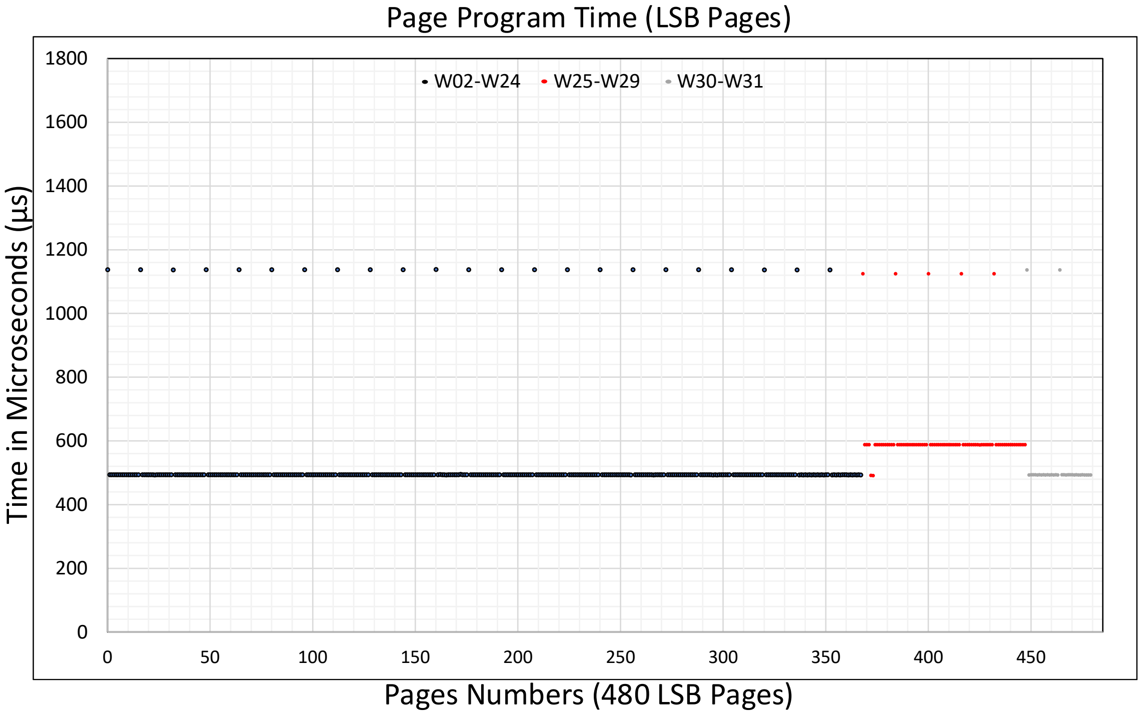


Figure 7.4. Page program times of 480 LSB pages in a block.

Shared MSB Pages. Figure 7.5 shows the page program times for shared MSB pages. The program times range between ~1280 μs and ~1602 μs. It appears that there are several distinct discrete programming times in this range.

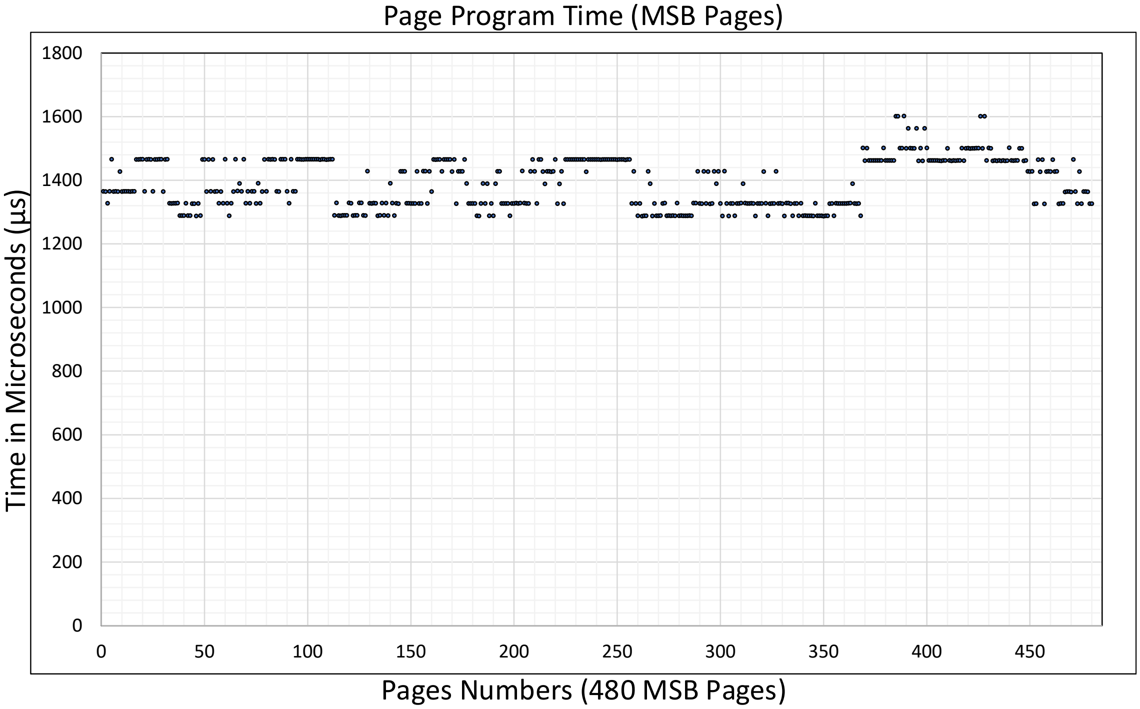


Figure 7.5. Page program times of 480 msb pages in a block.

7.2.2 Page Program Times in SLC Mode

We perform a set of experiments to analyze the program times of NAND flash memory in the SLC mode. First a fresh block is converted into the SLC mode, the block is first erased. Then, all 512 pages in the block are programmed with all zeros.

Figure 7.6 shows the program times of all 512 pages in a block. The program times for the majority of SLC pages is ~367 µs. 32 pages (every 16th), i.e. first pages to be programmed in a given layer, take ~582 µs (32x16=512). The exceptions to this rule are the first pages in the first layer W00 and the last layer W33 that require ~692 µs. This is consistent with what we observed previously in the MLC mode, except that the program times are significantly lower than the ones observed in the MLC mode. In addition, several groups of pages (not including the first page in a layer) require significantly lower program times – the pages with indices 65-79 (layer W04), 113-127 (layer W07), 241-255 (W15), 257-281 (W16), 385-399 (W24) require only ~271 µs. We do not have a plausible explanation for this phenomenon.

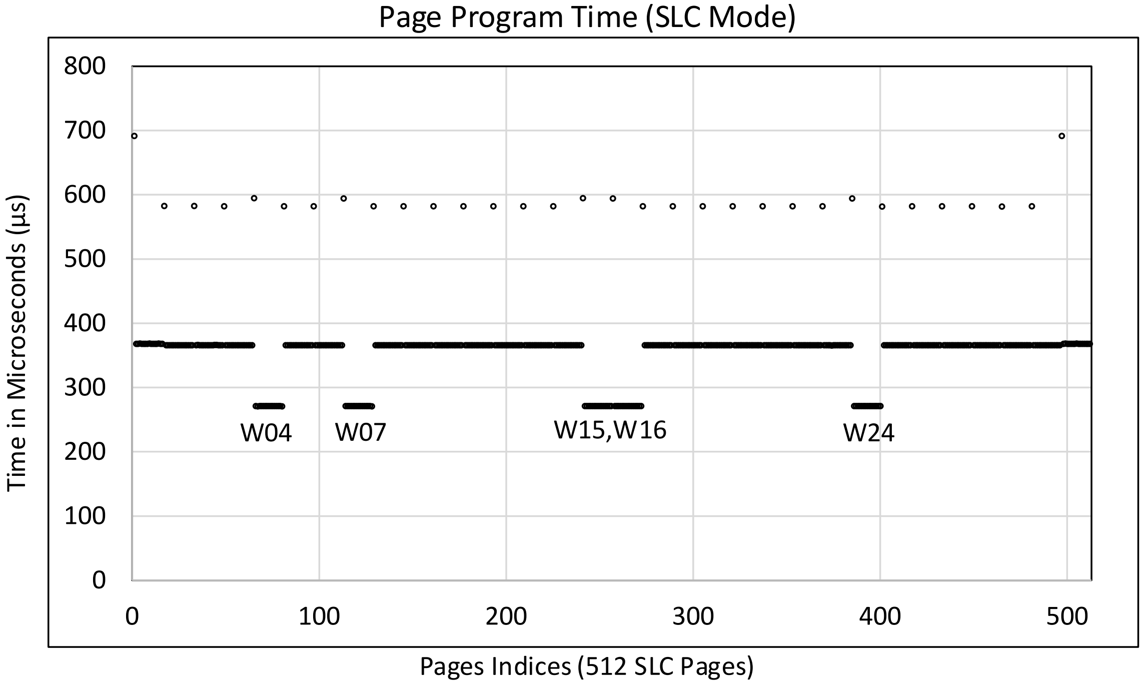


Figure 7.6. Page program times as a function of page indices in SLC Mode (512 Pages)

7.2.3 Page Read Times in MLC Mode

The read operation in NAND flash involves two distinct stages: first, the data from the NAND array is brought to the internal data register after issuing a read command; second, the content of data register is read using data-out (DOUT) cycles. For read time analysis, we measure the time taken to transfer data from the NAND array to the data register.

Figure 7.7 shows the page read times from 100 pages when the memory is operating in the MLC mode. The read times of unshared pages are ~91 µs, which is identical to when reading erased unshared pages. Similarly, the read times of MSB pages are ~120 µs, which identical to when reading erased MSB pages. Expectedly, the read times of LSB pages are less than read times of MSB pages and are ~93.5 µs.

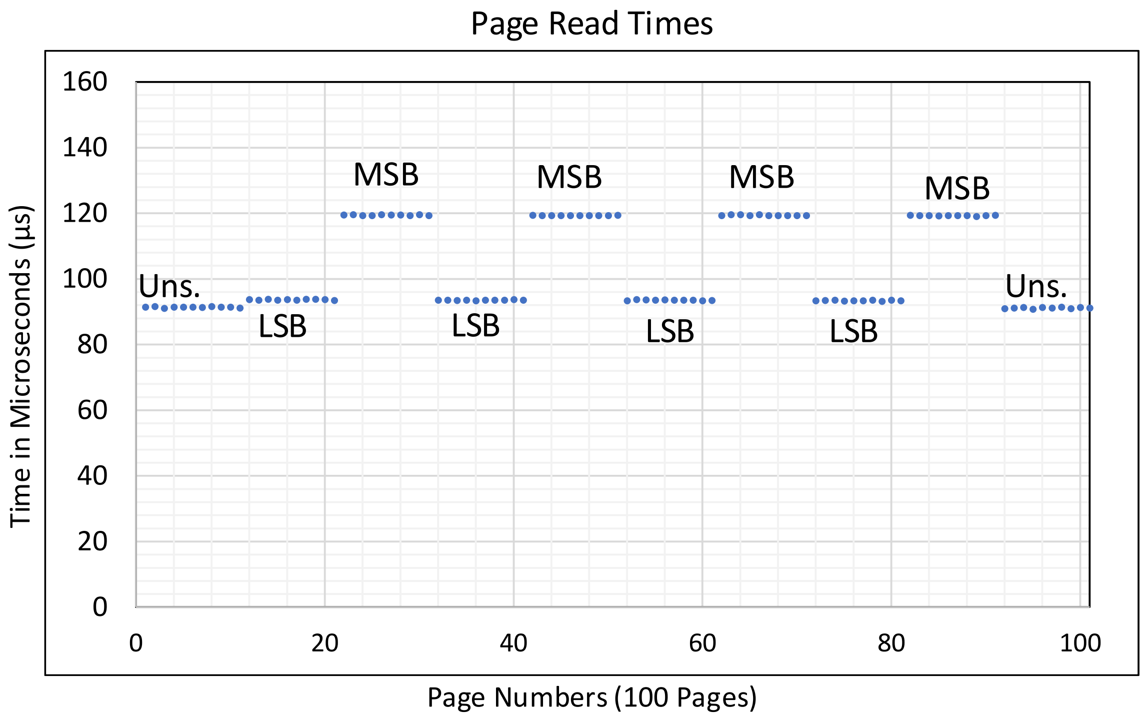


Figure 7.7. Page read times for 100 selected pages.

7.3 Concluding Remarks

The page program and page read timing characterizations reveal the following insights:

- Pages in the first subblock (B0) are slower to program,
- Unshared and shared LSB pages in subblocks B1 to B15 are faster to program than the shared MSB pages,

- Programming unshared pages in topmost dummy layer, W33, is slower to program than unshared pages in other layers, and
- Reading Unshared and Shared LSB pages is faster than reading shared MSB pages.

These characterization results can be used to develop better algorithms during an FTL design. For example, the faster program times offered by unshared pages and shared LSB pages can be exploited to enhance the write performance of the FTL. Based on the write hotness of the data, the placement algorithm can place the hot data into either unshared or shared LSB page. Similarly, the faster read times of the unshared pages and shared LSB pages can be exploited to design the placement algorithm such that read hot data is written into unshared pages or shared LSB pages.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This dissertation describes two techniques, namely (a) Microcontroller Fingerprinting and (b) *Flashmark*, that can be used to secure chips that contain flash memories by exploiting their physical properties. This dissertation also presents a technique to reduce time and energy consumed by critical flash memory operations.

Microcontroller fingerprinting uses partial erase operations of flash memory segments that bring them into the state that exposes physical properties of the flash memory cells through a digital interface. These properties reflect semiconductor process variations and defects that are unique to each microcontroller or a flash memory segment within a microcontroller. This dissertation explores threshold voltage variation in NOR flash memory cells for generating fingerprints and describes an algorithm for extracting fingerprints. This dissertation also presents the algorithms for generating the enrollment and authentication fingerprints. The experimental evaluation utilizing a family of commercial microcontrollers demonstrates that the fingerprints are unique and repeatable, and the proposed technique is cost-effective, robust, and resilient to changes in voltage and temperature as well as to aging effects. Performance evaluation of the proposed technique shows that it significantly outperforms other related proposed techniques.

Flashmark uses repeated stressing to irreversibly change the physical properties of flash memory cells to permanently imprint manufacturer information into dedicated blocks of flash memory chips. The imprinted watermark is read out by sensing the changes in physical properties of flash cells through standard digital interface using partial erase operations. The proposed technique is successfully demonstrated on embedded NOR flash memories in low-power microcontrollers, but the proposed method is applicable broadly to NOR and NAND flash memories. The dissertation also explores the metrics of interest such as the bit error rates, the watermark imprint time, and the watermark extraction time.

The technique to reduce the time and energy utilizes partial flash memory operations, i.e. partial flash erase and partial flash program operations. The partial erase and program operations abort nominal erase and program operations at an optimal time that guarantees no loss of information. The proposed flash operations are implemented and evaluated on NOR flash memory of commercial microcontrollers. This dissertation demonstrates that they can provide significant saving in energy of over 98% for segment erase operations and from 24-64% for segment program operations. This dissertation also shows how flash stress level impacts the parameters of interest, i.e. optimal partial erase times and partial program times, for the proposed technique.

Finally, this dissertation presents a timing characterization of operations in a family of 3D NAND flash memory chips. The timing characterization results show that significant timing variations exist among program operations of different pages.

Although this dissertation presents the microcontroller fingerprints generated from an embedded NOR flash memory for authentication purpose, future works can extend this technique for secure key generation. Secure key paves way for encryption as well as authentication. Similarly, *Flashmark* can be expanded to newer flash memories that have significantly faster program and erase times. Since NAND flash memories perform program and erase operations in multiple successive pulses, future works can expand the technique of early aborting flash memory operations to save time and energy to NAND flash memories.

Future works can use the results from 3D NAND characterization to develop better data placement and management algorithms in the FTL. For example, the faster program times offered by unshared pages and shared LSB pages can be exploited to enhance the write performance of the FTL. Based on the write hotness of the data, the placement algorithm can place the hot data into either unshared or shared LSB page. Similarly, the faster read times of the unshared pages and shared LSB pages can be exploited to design the placement algorithm such that read hot data is written into unshared pages or shared LSB pages.

REFERENCES

[1] "1987: Toshiba Launches NAND Flash," *eWEEK*, Apr. 11, 2012. https://www.eweek.com/storage/1987-toshiba-launches-nand-flash/ (accessed Mar. 05, 2021).

[2] F. Masuoka, M. Momodomi, Y. Iwata, and R. Shirota, "New ultra high density EPROM and flash EEPROM with NAND structure cell," in *1987 International Electron Devices Meeting*, Dec. 1987, pp. 552–555, doi: 10.1109/IEDM.1987.191485.

[3] R. D. Pashley and S. K. Lai, "Flash memories: the best of two worlds," *IEEE Spectrum*, vol. 26, no. 12, pp. 30–33, Dec. 1989, doi: 10.1109/6.45032.

[4] "Non-Volatile Memory Market | Growth, Trends, Forecasts (2020 - 2025)." https://www.mordorintelligence.com/industry-reports/non-volatile-memory-market (accessed Mar. 02, 2021).

[5] S. S. Rizvi and T. Chung, "Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems," in *2010 2nd International Conference on Computer Engineering and Technology*, Apr. 2010, vol. 7, pp. V7-297-V7-299, doi: 10.1109/ICCET.2010.5485421.

[6] "What is Flash Storage? - Benefits of SSD vs HDD | NetApp." https://www.netapp.com/data-storage/what-is-flash-storage/ (accessed Mar. 02, 2021).

[7] "When Does Cloud Computing Need Flash?" https://www.micron.com/about/blog/2018/july/when-does-cloud-computing-need-flash (accessed Mar. 02, 2021).

[8] "Number of connected devices worldwide 2030," *Statista*. https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/ (accessed Mar. 02, 2021).

[9] EETimes, "Chip counterfeiting case exposes defense supply chain flaw," *EETimes*, Oct. 24, 2011. https://tinyurl.com/u7wvlkh (accessed Oct. 27, 2019).

[10] "Reports of Counterfeit Parts Quadruple Since 2009, Challenging US Defense Industry and National Security - Omdia." https://tinyurl.com/spusztx (accessed Aug. 21, 2019).

[11] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014, doi: 10.1109/JPROC.2014.2332291.

[12] “Counterfeits Costing Semiconductor Industry Billions - EE Times Asia.” https://tinyurl.com/wcuzooz (accessed Oct. 23, 2019).

[13] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, “Physical Unclonable Functions and Applications: A Tutorial,” *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014, doi: 10.1109/JPROC.2014.2320516.

[14] R. Pappu, “Physical One-Way Functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, Sep. 2002, doi: 10.1126/science.1074376.

[15] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Proceedings of the 44th annual conference on Design automation - DAC ’07*, San Diego, California, 2007, p. 9, doi: 10.1145/1278480.1278484.

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, “Silicon physical random functions,” in *Proceedings of the 9th ACM conference on Computer and communications security*, Washington, DC, USA, Nov. 2002, pp. 148–160, doi: 10.1145/586110.586132.

[17] J. W. Lee, Daihyun Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas, “A technique to build a secret key in integrated circuits for identification and authentication applications,” in *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, Jun. 2004, pp. 176–179, doi: 10.1109/VLSIC.2004.1346548.

[18] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers,” *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, Sep. 2009, doi: 10.1109/TC.2008.212.

[19] C. Böhm, M. Hofer, and W. Pribyl, “A microcontroller SRAM-PUF,” in *2011 5th International Conference on Network and System Security*, Sep. 2011, pp. 269–273, doi: 10.1109/ICNSS.2011.6060013.

[20] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum, “A Proof of Concept SRAM-based Physically Unclonable Function (PUF) Key Generation Mechanism for IoT Devices,” in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2019, pp. 1–8, doi: 10.1109/SAHCN.2019.8824887.

[21] A. Aysu, S. Gaddam, H. Mandadi, C. Pinto, L. Wegryn, and P. Schaumont, “A Design Method for Remote Integrity Checking of Complex PCBs,” in *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 1517–1522, doi: 10.3850/9783981537079_1007.

[22] M. Laban and M. Drutarovsky, “Leakage free helper data storage in microcontroller based PUF implementation,” *Microprocessors and Microsystems*, p. 103369, Nov. 2020, doi: 10.1016/j.micpro.2020.103369.

[23] Z. Guo, X. Xu, M. M. Tehranipoor, and D. Forte, “FFD: A Framework for Fake Flash Detection,” in *Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC ’17*, Austin, TX, USA, 2017, pp. 1–6, doi: 10.1145/3061639.3062249.

[24] P. Kumari, B. M. S. B. Talukder, S. Sakib, B. Ray, and M. T. Rahman, “Independent detection of recycled flash memory: Challenges and solutions,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Apr. 2018, pp. 89–95, doi: 10.1109/HST.2018.8383895.

[25] S. Sakib, P. Kumari, B. M. S. B. Talukder, M. T. Rahman, and B. Ray, “Non-Invasive Detection Method for Recycled Flash Memory Using Timing Characteristics,” *Cryptography*, vol. 2, no. 3, p. 17, Sep. 2018, doi: 10.3390/cryptography2030017.

[26] A. R. Duncan, M. J. Gadlage, A. H. Roach, and M. J. Kay, “Characterizing Radiation and Stress-Induced Degradation in an Embedded Split-Gate NOR Flash Memory,” *IEEE Transactions on Nuclear Science*, vol. 63, no. 2, pp. 1276–1283, Apr. 2016, doi: 10.1109/TNS.2016.2540803.

[27] T. Instruments, “MSP430 Flash Memory Characteristics.” Texas Instruments Incorporated, 2008, [Online]. Available: http://www.ti.com/lit/an/slaa334b/slaa334b.pdf.

[28] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*. Springer Netherlands, 2010.

[29] R. Micheloni, A. Marelli, and S. Commodaro, “NAND overview: from memory to systems,” in *Inside NAND Flash Memories*, R. Micheloni, L. Crippa, and A. Marelli, Eds. Dordrecht: Springer Netherlands, 2010, pp. 19–53.

[30] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, “Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2017, pp. 49–60, doi: 10.1109/HPCA.2017.61.

[31] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling,” in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2013, pp. 1285–1290, doi: 10.7873/DATE.2013.266.

[32] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, “Data retention in MLC NAND flash memory: Characterization, optimization, and recovery,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 551–563, doi: 10.1109/HPCA.2015.7056062.

[33] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, “Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct. 2013, pp. 123–130, doi: 10.1109/ICCD.2013.6657034.

[34] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery,” in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Rio de Janeiro, Brazil, Jun. 2015, pp. 438–449, doi: 10.1109/DSN.2015.49.

[35] Y. Cai, G. Yalcin, O. Mutlu, and E. F. Haratsch, “Error Analysis and Retention-Aware Error Management for NAND Flash Memory,” vol. 17, no. 1, p. 25, 2013, [Online]. Available: https://users.ece.cmu.edu/~omutlu/pub/flash-error-analysis-and-management_itj13.pdf.

[36] C. M. Compagnoni, A. S. Spinelli, S. Beltrami, M. Bonanomi, and A. Visconti, “Cycling Effect on the Random Telegraph Noise Instabilities of nor and nand Flash Arrays,” *IEEE Electron Device Letters*, vol. 29, no. 8, pp. 941–943, Aug. 2008, doi: 10.1109/LED.2008.2000964.

[37] N. Mielke *et al.*, “Bit error rate in NAND Flash memories,” in *2008 IEEE International Reliability Physics Symposium*, Apr. 2008, pp. 9–19, doi: 10.1109/RELPHY.2008.4558857.

[38] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, “Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation,” in *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, Jun. 2018, p. 106, doi: 10.1145/3219617.3219659.

[39] R. Micheloni, L. Crippa, C. Zambelli, and P. Olivo, “Architectural and Integration Options for 3D NAND Flash Memories,” *Computers*, vol. 6, no. 3, Art. no. 3, Sep. 2017, doi: 10.3390/computers6030027.

[40] “MSP430x5xx and MSP430x6xx Family User’s Guide.” Texas Instruments Incorporated, Mar. 2018, [Online]. Available: https://www.ti.com/lit/ug/slau208q/slau208q.pdf.

[41] P. Poudel, “Using NOR Flash Memory in Microcontrollers for Generating True Random Numbers,” 2018, [Online]. Available: http://www.ece.uah.edu/~milenka/docs/prawar.poudel.thesis.pdf.

[42] “MSP-EXP430F5438 Evaluation board | TI.com.” https://www.ti.com/tool/MSP-EXP430F5438 (accessed Jan. 27, 2021).

[43] “MSP-EXP430F5529LP Development kit | TI.com.” https://www.ti.com/tool/MSP-EXP430F5529LP (accessed Jan. 27, 2021).

[44] "Open NAND Flash Interface Specification." ONFI Workgroup, Feb. 12, 2020, [Online]. Available: https://bit.ly/2Nl2qw2.

[45] sickRanchez-c137, "sickRanchez-c137/onfi_plus," Nov. 28, 2020. https://github.com/sickRanchez-c137/onfi_plus (accessed Mar. 05, 2021).

[46] P. Poudel, B. Ray, and A. Milenkovic, "Microcontroller Fingerprinting Using Partially Erased NOR Flash Memory Cells," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 3, p. 23, Mar. 2021, doi: doi.org/10.1145/3448271.

[47] Y. Wang, "Flash Memory For Ubiquitous Hardware Security Functions," Jan. 2014, Accessed: Mar. 07, 2021. [Online]. Available: https://ecommons.cornell.edu/handle/1813/36037.

[48] M. S. Hashemian, B. Singh, F. Wolff, D. Weyer, S. Clay, and C. Papachristou, "A robust authentication methodology using physically unclonable functions in DRAM arrays," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2015, pp. 647–652, doi: 10.7873/DATE.2015.0308.

[49] H. Mandadi, "Remote Integrity Checking using Multiple PUF based Component Identifiers," 2017, [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/78200.

[50] S. Rosenblatt, S. Chellappa, A. Cestero, N. Robson, T. Kirihata, and S. S. Iyer, "A Self-Authenticating Chip Architecture Using an Intrinsic Fingerprint of Embedded DRAM," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 11, pp. 2934–2943, Nov. 2013, doi: 10.1109/JSSC.2013.2282114.

[51] S. Sakib, A. Milenković, M. T. Rahman, and B. Ray, "An Aging-Resistant NAND Flash Memory Physical Unclonable Function," *IEEE Transactions on Electron Devices*, vol. 67, no. 3, pp. 937–943, Mar. 2020, doi: 10.1109/TED.2020.2968272.

[52] B. M. S. B. Talukder, B. Ray, M. Tehranipoor, D. Forte, and M. T. Rahman, "LDPUF: Exploiting DRAM Latency Variations to Generate Robust Device Signatures," *arXiv:1808.02584 [cs]*, Aug. 2018, Accessed: Mar. 28, 2019. [Online]. Available: http://arxiv.org/abs/1808.02584.

[53] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2018, pp. 194–207, doi: 10.1109/HPCA.2018.00026.

[54] F. Tehranipoor, N. Karimina, K. Xiao, and J. Chandy, "DRAM based Intrinsic Physical Unclonable Functions for System Level Security," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI - GLSVLSI '15*, Pittsburgh, Pennsylvania, USA, 2015, pp. 15–20, doi: 10.1145/2742060.2742069.

[55] Y. Wang, W. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, “Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints,” San Francisco, CA, May 2012, pp. 33–47, doi: 10.1109/SP.2012.12.

[56] M. Liu, C. Zhou, Q. Tang, K. K. Parhi, and C. H. Kim, “A data remanence based approach to generate 100% stable keys from an SRAM physical unclonable function,” in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2017, pp. 1–6, doi: 10.1109/ISLPED.2017.8009192.

[57] A. Bacha and R. Teodorescu, “Authenticache: harnessing cache ECC for system authentication,” in *Proceedings of the 48th International Symposium on Microarchitecture*, Waikiki, Hawaii, Dec. 2015, pp. 128–140, doi: 10.1145/2830772.2830814.

[58] S. Sutar, A. Raha, and V. Raghunathan, “D-PUF: an intrinsically reconfigurable DRAM PUF for device authentication in embedded systems,” in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, Pittsburgh, Pennsylvania, Oct. 2016, pp. 1–10, doi: 10.1145/2968455.2968519.

[59] A. Schaller *et al.*, “Decay-Based DRAM PUFs in Commodity Devices,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 462–475, May 2019, doi: 10.1109/TDSC.2018.2822298.

[60] S. Jia, L. Xia, Z. Wang, J. Lin, G. Zhang, and Y. Ji, “Extracting Robust Keys from NAND Flash Physical Unclonable Functions,” in *Information Security*, Cham, 2015, pp. 437–454, doi: 10.1007/978-3-319-23318-5_24.

[61] L. T. Clark, J. Adams, and K. E. Holbert, “Reliable techniques for integrated circuit identification and true random number generation using 1.5-transistor flash memory,” *Integration, the VLSI Journal*, Nov. 2017, doi: 10.1016/j.vlsi.2017.10.001.

[62] T.-N. Nguyen, S. Park, and D. Shin, “Extraction of Device Fingerprints Using Built-in Erase-Suspend Operation of Flash Memory Devices,” *IEEE Access*, vol. 8, pp. 98637–98646, 2020, doi: 10.1109/ACCESS.2020.2995891.

[63] P. Poudel, B. Ray, and A. Milenkovic, “Microcontroller TRNGs Using Perturbed States of NOR Flash Memory Cells,” *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 307–313, Feb. 2019, doi: 10.1109/TC.2018.2866459.

[64] L. T. Clark, J. Adams, and K. E. Holbert, “Reliable techniques for integrated circuit identification and true random number generation using 1.5-transistor flash memory,” *Integration*, vol. 65, pp. 263–272, Mar. 2019, doi: 10.1016/j.vlsi.2017.10.001.

[65] U. Guin, D. DiMase, and M. Tehranipoor, “Counterfeit Integrated Circuits: Detection, Avoidance, and the Challenges Ahead,” *J Electron Test*, vol. 30, no. 1, pp. 9–23, Feb. 2014, doi: 10.1007/s10836-013-5430-8.

[66] P. Poudel, B. Ray, and A. Milenkovic, “Flashmark: Watermarking of NOR Flash Memories for Counterfeit Detection,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2020, pp. 1–6, doi: 10.1109/DAC18072.2020.9218521.

[67] D. McGrath, “Semiconductor Counterfeiting is a Global Problem,” *EETimes*, Jul. 26, 2017. https://www.eetimes.com/semiconductor-counterfeiting-is-a-global-problem/ (accessed Feb. 10, 2021).

[68] “National Defense Authorization Act for Fiscal Year 2012.” [Online]. Available: https://tinyurl.com/rc4bpzt.

[69] M. Crawford, T. Telesco, C. Nelson, J. Bolton, K. Bagin, and B. Botwin, “DEFENSE INDUSTRIAL BASE ASSESSMENT: COUNTERFEIT ELECTRONICS.” Jan. 2010, [Online]. Available: https://bit.ly/3ryGCeK.

[70] D. McGrath, “EETimes - IHS: Counterfeit parts represent $169B annual risk -,” *EETimes*, Apr. 04, 2012. https://www.eetimes.com/ihs-counterfeit-parts-represent-169b-annual-risk/ (accessed Mar. 01, 2021).

[71] S. Byrne, “Fake and counterfeit USB flash drives spreading on Amazon,” *Myce.com*, Jul. 04, 2014. https://www.myce.com/news/fake-and-counterfeit-usb-flash-drives-spreading-on-amazon-72165/ (accessed Feb. 10, 2021).

[72] eTeknix.com, “Kingfast unknowingly sent counterfeit SSDs with mislabelled Flash NAND for review,” *eTeknix*, Feb. 17, 2013. https://www.eteknix.com/kingfast-unknowingly-sent-counterfeit-ssds-with-mislabelled-flash-nand-for-review/ (accessed Oct. 27, 2019).

[73] “Global Report – eBay Fake Memory 2008 – 2009,” *Fake Flash Memory Information - FlashChipDirector*, Jan. 10, 2010. https://flashfakecentral.wordpress.com/2010/01/10/global-report-ebay-fake-memory-2008-2009/ (accessed Feb. 10, 2021).

[74] “Fake Flash News - Internet & eBay Fraud,” *Fake Flash News - Internet & eBay Fraud*, Feb. 04, 2012. https://fakeflashnews.wordpress.com/2012/02/04/ebay-china-resumes-flash-memory-fraud-with-a-vengeance-if-you-want-a-cheap-bargain-for-usb-flash-drives-mp-players-or-memory-cards-on-ebay-expect-to-be-scammed-sandisk-memorette-samsung-transce/ (accessed Feb. 10, 2021).

[75] K. Ahi, N. Asadizanjani, S. Shahbazmohamadi, M. Tehranipoor, and M. Anwar, “Terahertz characterization of electronic components and comparison of terahertz imaging with x-ray imaging techniques,” 2015, doi: 10.1117/12.2183128.

[76] U. Guin, D. DiMase, and M. Tehranipoor, “A Comprehensive Framework for Counterfeit Defect Coverage Analysis and Detection Assessment,” *Journal of Electronic Testing*, vol. 30, no. 1, pp. 25–40, Feb. 2014, doi: 10.1007/s10836-013-5428-2.

[77] S. Shahbazmohamadi, D. Forte, and M. Tehranipoor, “Advanced Physical Inspection Methods for Counterfeit IC Detection,” Nov. 2014, pp. 55–64, doi: 10.31399/asm.cp.istfa2014p0055.

[78] N. Robson *et al.*, “Electrically Programmable Fuse (eFUSE): From Memory Redundancy to Autonomic Chips,” in *IEEE Custom Integrated Circuits Conference*, Sep. 2007, pp. 799–804, doi: 10.1109/CICC.2007.4405850.

[79] L. M. Grupp *et al.*, “Characterizing flash memory: anomalies, observations, and applications,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, New York, New York, 2009, p. 24, doi: 10.1145/1669112.1669118.

[80] Y. Wang, W. Yu, S. Q. Xu, E. Kan, and G. E. Suh, “Hiding Information in Flash Memory,” in *IEEE Symposium on Security and Privacy*, May 2013, pp. 271–285, doi: 10.1109/SP.2013.26.

[81] P. Poudel and A. Milenković, “Saving Time and Energy Using Partial Flash Memory Operations in Low-Power Microcontrollers,” in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, Mar. 2020, pp. 183–189, doi: 10.1109/ISQED48828.2020.9137034.

[82] M. Salajegheh, Y. Wang, K. Fu, A. Jiang, and E. Learned-Miller, “Exploiting Half-wits: Smarter Storage for Low-power Devices,” in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, Berkeley, CA, USA, 2011, pp. 55–68, Accessed: Sep. 28, 2019. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960479.

[83] M. Salajegheh, Y. Wang, A. (Andrew) Jiang, E. Learned-Miller, and K. Fu, “Half-Wits: Software Techniques for Low-Voltage Probabilistic Storage on Microcontrollers with NOR Flash Memory,” *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 1–25, May 2013, doi: 10.1145/2465787.2465793.

[84] H.-W. Tseng, L. M. Grupp, and S. Swanson, “Underpowering NAND flash: Profits and perils,” in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–6, [Online]. Available: https://ieeexplore.ieee.org/document/6560755.

[85] V. Papirla and C. Chakrabarti, “Energy-aware error control coding for Flash memories,” in *2009 46th ACM/IEEE Design Automation Conference*, Jul. 2009, pp. 658–663, doi: 10.1145/1629911.1630085.

[86] S. Nath, "Energy efficient sensor data logging with amnesic flash storage," in *2009 International Conference on Information Processing in Sensor Networks*, Apr. 2009, pp. 157–168, [Online]. Available: https://ieeexplore.ieee.org/document/5211934.

[87] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Capsule: an energy-optimized object storage system for memory-constrained sensor devices," in *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, Boulder, Colorado, USA, 2006, p. 195, doi: 10.1145/1182807.1182827.

[88] T. Instruments, "MSP430F543x, MSP430F541x Mixed-Signal Microcontrollers datasheet (Rev. F)." Texas Instruments Incorporated, 2009, [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f5438.pdf.

[89] A. Dzhagaryan, A. Milenković, M. Milosevic, and E. Jovanov, "An environment for automated measurement of energy consumed by mobile and embedded computing devices," *Measurement*, vol. 94, pp. 103–118, Dec. 2016, doi: 10.1016/j.measurement.2016.07.073.

[90] K. Prall, "Scaling Non-Volatile Memory Below 30nm," in *2007 22nd IEEE Non-Volatile Semiconductor Memory Workshop*, Aug. 2007, pp. 5–10, doi: 10.1109/NVSMW.2007.4290561.

[91] F. Wu *et al.*, "Characterizing 3D Charge Trap NAND Flash: Observations, Analyses and Applications," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, Oct. 2018, pp. 381–388, doi: 10.1109/ICCD.2018.00064.

[92] K. Parat and C. Dennison, "A floating gate based 3D NAND technology with CMOS under array," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec. 2015, p. 3.3.1-3.3.4, doi: 10.1109/IEDM.2015.7409618.

[93] S. Nie, Y. Zhang, W. Wu, and J. Yang, "Layer RBER Variation Aware Read Performance Optimization for 3D Flash Memories," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2020, pp. 1–6, doi: 10.1109/DAC18072.2020.9218631.

[94] Q. Xiong *et al.*, "Characterizing 3D Floating Gate NAND Flash: Observations, Analyses, and Implications," *ACM Trans. Storage*, vol. 14, no. 2, pp. 1–31, May 2018, doi: 10.1145/3162616.

[95] Y. Shim, M. Kim, M. Chun, J. Park, Y. Kim, and J. Kim, "Exploiting Process Similarity of 3D Flash Memory for High Performance SSDs," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, Oct. 2019, pp. 211–223, doi: 10.1145/3352460.3358311.

[96] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and

Temperature Awareness," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Feb. 2018, pp. 504–517, doi: 10.1109/HPCA.2018.00050.

[97] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of Flash Translation Layer," *Journal of Systems Architecture*, vol. 55, no. 5, pp. 332–343, May 2009, doi: 10.1016/j.sysarc.2009.03.005.