Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Microelectronics

Design of FLASH Memory Reliability Test System

Master thesis

H. Ece Aykol

Master programme: Electronics and Communications
Specialisation: Electronics

Supervisor: Doc. Ing. Adam Bouřa, Ph.D.

Prague, August 2020

Thesis Supervisor:
Doc. Ing. Adam Bouřa, Ph.D.
Department of Microelectronics
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic

Copyright © August 2020 H. Ece Aykol

Declaration

I hereby declare I have written this master thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, August 2020

....
H. Ece Aykol

MASTER'S THESIS ASSIGNMENT

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

I. Personal and study details

Student's name: **Aykol Hanife Ece** Personal ID number: **472419**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Microelectronics**

Study program: **Electronics and Communications**

Specialisation: **Electronics**

II. Master's thesis details

Master's thesis title in English:

Design of FLASH Memory Reliability Test System

Master's thesis title in Czech:

Návrh systému pro testování spolehlivosti pamětí typu FLASH

Guidelines:

1. Learn the principle of FLASH memory operation and its failure rate depending on the load and the way the data is written [1]. Learn about testing methods [2] and statistical analysis to assess failure rates [3].
2. Implement the test bench and design a methodology for determining the reliability of selected FLASH memories, depending on storage algorithms.
3. Test the functionality of the system you design on a flash memory set, evaluate the test results statistically and suggest possible improvements.

Bibliography / sources:

[1] Aritome, S., Shirota, R., Hemink, G., Endoh, T., Masuoka, F.: Reliability issues of flash memory cells, Proceedings of the IEEE, Vol. 81 , Issue 5 , May 1993, URL: https://ieeexplore.ieee.org/abstract/document/220908/

[2] National Instruments: Introduction to MODBUS, Tutorial, 2009, URL: https://www.micronor.com/products/files/AN112/AN112_NIModbusTutorial.pdf

[3] Gingrich, P: Chi Square Tests, URL: http://uregina.ca/~gingrich/ch10.pdf

Name and workplace of master's thesis supervisor:

Ing. Adam Bouřa, Ph.D., Department of Microelectronics, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **19.02.2020** Deadline for master's thesis submission: **14.08.2020**

Assignment valid until: **19.02.2022**

Ing. Adam Bouřa, Ph.D. Supervisor's signature	prof. Ing. Pavel Hazdra, CSc. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt — Student's signature

© VUT v Praze, Design: VUT v Praze, VIC

Abstrakt

Flash paměť je typem elektricky vymazatelné a programovatelné volatilní paměti (EEPROM). Stala se jednou z nejvýznamnějších částí trhu s polovodičovými pamětmi. Výhodou flash pamětí je, že jsou elektricky přepisovatelné jako EEPROM s jedinou tranzistorovou strukturou jako dřívější EPROM. Tato kombinace umožňuje vyrábět flash paměti s vyšší hustotou a s nižšími náklady. Zatímco tržní podíl pamětí flash vzrůstá, technologie paměti flash posouvá své fyzické limity spolehlivosti paměti kvůli škálování a neustálému namáhání. Omezená životnost programu a množství přepisů (mazání) je nejvýznamnějším problémem z hlediska spolehlivosti technologie flash. V této studii je demonstrován a testován mechanismus selhání vyvolaný stresem způsobený konstantním cyklováním programování / mazání. Následně je poskytnuto řešení pro správu dat pro produkty Eaton využívající ASIC2 čip.

Klíčová slova: Flash paměť, spolehlivost, program/erase endurance, ASIC2

Abstract

Flash Memory, a type of electrically erasable and programmable read-only memory (EEPROM), became one of the most significant portions in the semiconductor memory market. Flash memories have the advantage of being electrically erasable like EEPROMs with a single transistor structure like early EPROMs. This combination allows Flash to be manufactured in great densities with lower cost and providing electrically erasable functionality. While the market share of Flash memory rises, Flash memory technology is pushing its physical limits in memory reliability due to scaling and constant stress. Program/Erase endurance is the most prominent issue in terms of reliability in Flash technology. Here in this study, the stress-induced failure mechanism caused by the constant program/erase cycling is demonstrated, tested and a data management solution for Eaton products, which are utilizing ASIC2, is provided.

Keywords: Flash memory, reliability, program/erase endurance, ASIC2

Acknowledgements

This thesis would not have been possible without the inspiration and support of a number of amazing individuals — my thanks and gratitude to all of them for being part of this journey.

I thankfully acknowledge the contributions of my manager, Ondřej Ficner, with his guidance, and valuable feedback in shaping many of the concepts presented in this thesis.

My gratitude extends to my supervisor Adam Bouřa for he was always ready to help whenever I ran into a trouble spot or had a question about my research or writing.

I am thankful for all my colleagues at Eaton European Innovation Center. I would like to thank all the nice people of the electronics lab for providing the most friendly work environment which I feel lucky to be working in.

I would like to single out Tomaš Vítek and Pavel Dědourek to express my most heartfelt gratitude for putting up with me the whole time and for being the best people ever! Without their continuous support, and amazing technical skills, this work would hardly have been completed.

I am forever thankful for all the kind people whom I am lucky to be friends with for their support and for always being there for me. I would like to thank particularly Jan Cabicar, Aslıhan Köksal, and Burim Shakjiri for their precious insight and support.

Finally, my deep and sincere gratitude is to my family for giving me their endless love, help, and support. I am forever thankful to my parents for giving me the opportunities and experiences that have made me who I am.

Contents

Abstrakt **v**

Introduction **1**

1 Theoretical Background **5**
1.1 Semiconductor Memories 5
1.2 Flash Memory 11
1.3 Flash Memory Reliability 16

2 Design and Implementation **25**
2.1 ASIC2 Internal Flash Memory 25
2.2 Endurance Test Algorithm 26
2.3 Endurance Test Hardware 31

3 Evaluation of Test Results **39**
3.1 Initial Endurance Test on Evalboard 40
3.2 Endurance Test on Testbed 42
3.3 Endurance Test Validation 47

4 Proposed Improvements for Endurance **51**
4.1 Application-Specific Optimization Modules 51
4.2 EEPROM Emulation on ASIC2 Flash Memory . . . 55

Conclusion **73**

APPENDICES **74**

A Testbed PCB Layout **75**

B Dataflash Dataflow Charts **77**

C Dataflash: Simple Use Case Example **81**

D Dataflash Module Description **85**

D.1 Key features 85
D.2 Excel Calculator Tutorial 85
D.3 User Defines 87
D.4 Dataflash Module Files 88
D.5 User Function Definitions 88

Bibliography **97**

List of Tables

1.1 Characteristics Comparison of Commercial Memories 7
1.2 Bias Configurations of Floating Gate Cell 15

2.1 Recommended Endurance Evaluation Flow 31

3.1 Failure/Success Percentage 46
3.2 Modified Failure/Success Percentage 50

4.1 Page Headers in HEX 62
4.2 Variable Element Structure 63
4.3 Data Bit Coverage in Hamming 69

C.1 Emulated EEPROM Usage Example Variables . . . 81

List of Figures

1 ASIC2 Layout 2

1.1 Memory Array 6
1.2 Semiconductor Memory Tree 7
1.3 DRAM Memory Cell 8
1.4 Floating Gate MOSFET 9
1.5 Floating Gate MOSFET with Selecting Transistor . 10
1.6 Classification of Flash Memories 11
1.7 NOR Flash Memory Array 13
1.8 I-V Characteristics of Floating Gate Cell 14
1.9 Lifetime of a System 17
1.10 Threshold Voltage (V_t) Distribution 18
1.11 Program Disturbs on Flash Memory Array 19
1.12 Window Closure of V_t 22
1.13 Program/Erase Timings 24

2.1 ASIC2 Packaging 26
2.2 ASIC2 Evaluation Board 30
2.3 ASIC2 Testbed Powering Schematic 33
2.4 ASIC2 Testbed Connections Schematic 33
2.5 Testbed ModBus Connections 34
2.6 Testbed JTAG Connections 34
2.7 Testbed Powering Schematic 34
2.8 Testbed single ASIC2 Test Cell 35
2.9 Testbed 3D Render 35
2.10 Testbed PCB Populated 36
2.11 DLL Testbed Populated 37

3.1 Endurance Cycle Distribution for ASIC2 No:12 . . 43
3.2 Endurance Cycle Distribution for ASIC2 No:17 . . 44
3.3 Standard Distribution 44
3.4 Endurance Cycle Distribution for ASIC2 No:12 (Less than 30,000 cycles) 48
3.5 Endurance Cycle Distribution for ASIC2 No:17 (Less than 30,000 cycles) 49

4.1 Maximum storage module example : Maximum = 10 52
4.2 Maximum storage module example : Maximum = 18 53
4.3 Counter storage module - Page A 55
4.4 Counter storage module - Page B 55
4.5 Simplified Header Transitions Between Sets 58
4.6 Header Flow 60
4.7 Variable Element in Flash Page 64
4.8 Header Transitions during Garbage-Collection . . . 67

A.1 ASIC2 Testbed PCB Layout 76

B.1 Write Function Data Flow - No corruption 78
B.2 Write Function Data Flow - Corruption 79
B.3 Get Function Data Flow 80

C.1 Emulated EEPROM Data Transition 83

D.1 Excel Calculator Inputs 86
D.2 Excel Calculator Information 86
D.3 Header and Linker Definitions 87
D.4 Project Header File 87
D.5 Project Linker File 87

Introduction

Reliability of Flash Memories

Flash memory is a type of electrically erasable and programmable (EEPROM) memory, which supports basic operations: read, write (referred as *program*), and erase. The isolated floating gate achieves the non-volatility by being able to preserve the charge for an extended period of time. Program and erase operations both lead to a shift in the threshold voltage of the floating gate transistor. Reading operation is done by sensing the difference in the threshold voltage [Bez+03; Pav+97].

Programming and erasing the Flash block, also called a sector but it will be referred to as a *page* throughout the study. The terminology will be explained in the chapter 1 - *Theoretical Background*, induces stress on the transistor structure causing a detrimental impact on the reliability of the Flash memory [Moh+10]. The reliability of the Flash cell greatly affects the endurance and retention of memory. The typical data retention for Flash memory is around 10-20 years [LL14; Moh+10]. Repeated programming and erasing will lead to the oxide layer wearing down and cause the layer to be ineffective. As a result of this, the Stress-Induced Leakage Current (SILC) in the memory cell will increase, which will highly affect data retention and the endurance cycle [Moh+10].

Endurance of the memory is measured with the program/erase cycles (*P/E cycle*) that a Flash memory cell can handle while maintaining the data integrity stored in the cell. Endurance of Flash memory cell is a function of the charge trapping characteristics of the oxide. The more the memory cell is stressed by frequent programming and erasing, the more charge will get trapped in the oxide and the reliability of the cell will degrade over time [Moh+10; Jae+03; BD10]. Endurance issues of Flash memory will be explained in detail later on.

The reliability of Flash memories is of a great importance due to the continuous technology scaling and Flash memories becoming the leading technology in the semiconductor market for the non-volatile data storage needs such as hard-drives for mass data storage or application code storage as well as the application parameters [Bez+03; Pav+97; SL00; Cri+96].

Problem Statement

This study contributes to quantifying the reliability of the internal Flash memory of an application-specific integrated circuit (ASIC) named ASIC2 by Eaton and offering an optimization solution for data management and storage for the ASIC2 internal Flash memory [Boe18].

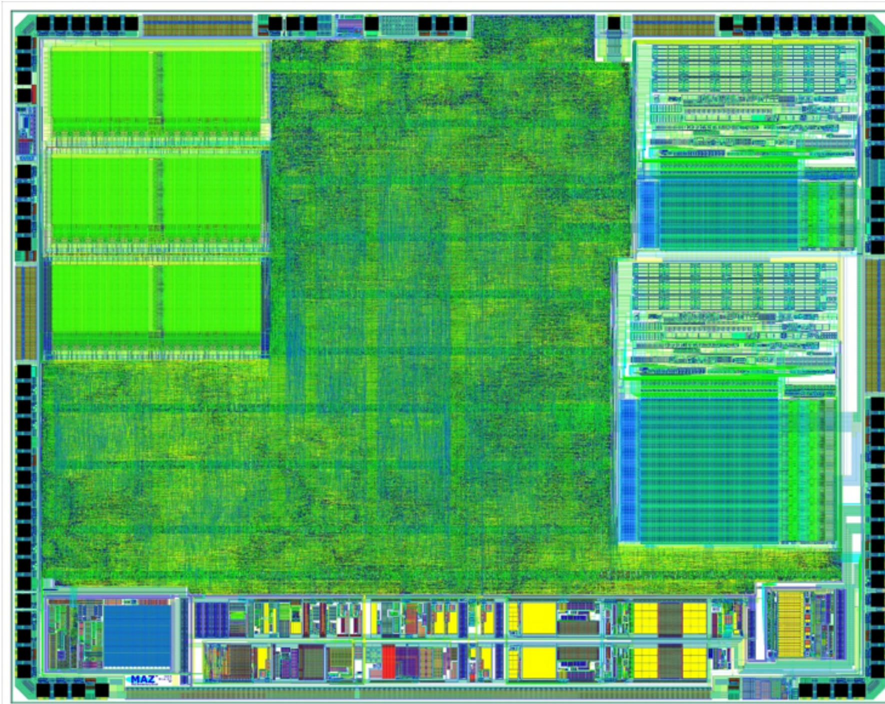

Figure 1: ASIC2 Layout. Courtesy of Eaton Internal Engineering.

ASIC2 is the second generation of Application Specific Integrated Circuit specifically designed to support SmartWire-DT (SWD) [Eat], an intelligent machine wiring solution designed by Eaton. The ASIC2 is based on dual-core 32-bit ARM technology with extended functionalities [Boe18].

ASIC2, as the successor of ASIC1, is in process of being integrated into all Eaton products, which are based on ASIC1. Some applications require persistent data storage with the possibility of being updatable many times. It was decided that the internal Flash memory of ASIC2, which is designed for program store and execution, will be utilized for data storage as well. As it will be explained in detail in the following chapters, Flash memories are not able to program the same physical location without an erase operation and constant program/erase cycle raises reliability issues [Bez+03].

The study aims to test the endurance reliability of the ASIC2 Flash memory from the firmware side to validate the endurance cycle provided by the vendor. The confidence interval for the probability of failure for a certain confidence level is calculated against the datasheet value. The next step is to offer a foundation for the decision on how to store persistent application data in the ASIC2 Flash memory, i.e. emulating EEPROM on ASIC2 Flash memory. The data storage should be persistent even in case of a power loss situation at any moment. An unexpected power-loss situation should lead to a defined state and the data storage solution should recover from that. The data management/storage should be easily scalable in storage rate and dataset size depending on the application requirements. The problem statement was proposed by the initial study provided by Eaton Industries GmbH [BB17].

Methodology

The method adapted to quantify the reliability and offer a solution for optimization is decided as:

1. Learning about Flash memory reliability testing methods
2. Coming up with the suitable endurance test method to run as a part of the firmware
3. Test-bed PCB design
4. Analysing test results and validating the data-sheet endurance cycle
5. Offering a data management algorithm to optimize the internal Flash to be used as a virtual EEPROM

Organization of Thesis

Following the steps of the *Methodology* section, the chapter 1 - *Theoretical Background* gives an in-depth explanation of the physical structure and the operational characteristics of non-volatile memory cells, with a focus on NOR Flash memory, possible and most common failure mechanisms in Flash memory cell.

Next, chapter 2 - *Design and Implementation* introduces the ASIC2 internal Flash memory, the idea of the test algorithm on how to test the ASIC2 internal Flash memory from the higher level of firmware-side, which is loosely based on the methods used during chip-level tests and lastly the test-board designed for the endurance testing.

Later on, the results from the testbed including the initial test run on the Eaton ASIC2 evaluation-board (evalboard), statistical analysis of the results, and the validation test are presented in the chapter 3 - *Evaluation of Test Results*.

Afterwards, the chapter 4 - *Proposed Improvements for Endurance* explains various optimization algorithms implemented, in both application-specific and a generic fashion, for data management in ASIC2 Flash memory.

Finally, the study is summed up and concluded in the *Conclusion*, furthermore the future work is discussed.

Chapter 1

Theoretical Background: The Physical Structure and Reliability of Flash Memories

In this chapter, the section section 1.1 - *Classification and Characterisation of Semiconductor Memories* classifies the industry standard memories in terms of the difference between volatile and non-volatile as well as their cell structures and distinctive operating characteristics. Later on, the section 1.2 - *Flash Memory* and section 1.3 - *Flash Memory Reliability* give more in-depth detail on the physical structure and characteristics of Flash memories and the reliability issues in Flash memory cells as well as the most common memory cell failure mechanisms.

1.1 Classification and Characterisation of Semiconductor Memories

Semiconductor memories are used to store and execute program code, as well as to store user data and other information, which play a vital role in computing systems and embedded systems. Memory can store millions of words, where each word contains many bits of data.

The core of semiconductor memory is the memory array, and the memory array is composed of storage units known as cells as shown in Figure 1.1. One memory cell can store one bit of data, in this case, it is called single-level cell (SLC) [HC12]. Some memory cells

can store two bits of data in a single cell then they are called a multi-level cell (MLC) [HC12; Con+98]. In addition to the memory array, address decoders, sense amplifiers, control logic circuits are also required for a memory system to function properly [HC12].

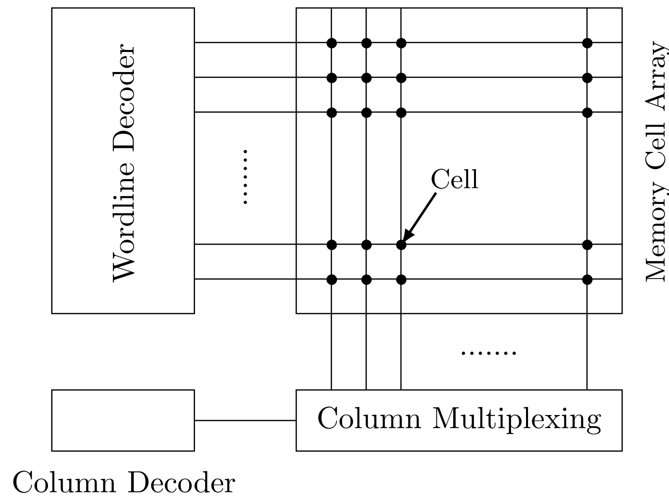

Figure 1.1: Basic memory structure. Adapted from Hai and Chen [HC12].

Information storage currently depends on the use of two different types of memories, which are divided into two categories:

1. Volatile memories
2. Non-volatile Memories

Figure 1.2 demonstrates the classification of commercial semiconductor volatile and non-volatile memories. Volatile memories have short execution times to access during run time and do logic operations, but data retention either requires constant refreshing (DRAMs) or a constant power supply (SRAMs). Both approaches are highly demanding in terms of energy required to perform data retention. Non-volatile memories have the retention time that is advantageous for long term information storage requirements. At rest, they do not need a power supply or constant refreshing in order to retain the data stored. The downside is that read, write and especially erase times are much longer compared to logic operations. Thus, they are classified as Read-Only Memories (ROMs) since it is more convenient to use these memories for storing persistent data.

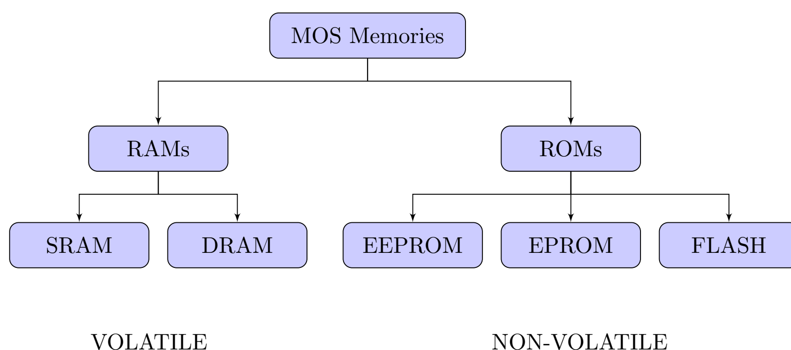

Figure 1.2: Semiconductor memory tree. Adapted from Bez et al. [Bez+03].

Table 1.1 shows the comparative study of the operational characteristics of the commercial volatile and non-volatile memories. Currently, the most common semiconductor memories are Static RAM (SRAM), dynamic RAM (DRAM) for logic operations, and non-volatile memories for data storage as well as program code storage and execution, that will be explained later on in detail.

	SRAM	DRAM	FLASH(NAND)
Read time	0.1-0.3 ns	10 ns	100,000 ns
Write time	0.1-0.3 ns	10 ns	100,000 ns
Retention	as long as V applied	<<seconds	years
Endurance	$>10^{16}$ cycles	$>10^{16}$ cycles	$>10^{4}$ cycles

Table 1.1: Comparative study of operational characteristics of commercial volatile and non-volatile memories. Adapted from Lacaze and Lacroix [LL14].

Volatile Memories

Here in this section, only DRAMs are explained as an example of a volatile memory due to their simplicity in functionality. SRAMs are working in a similar but more complex fashion compared to DRAMs.

Dynamic Random Access Memories (DRAMs) have a very short retention time. They can retain the data over a very short period of time, as in milliseconds or less [LL14]. They also require to be refreshed frequently in order to converse the information.

DRAMs consist of a metal-oxide semiconductor field-effect transistor (MOSFET) and a capacitor. Schematic of a single cell of DRAM is shown in Figure 1.3, where the MOSFET and the capacitor are linked to two conducting lines: the wordline and the bitline.

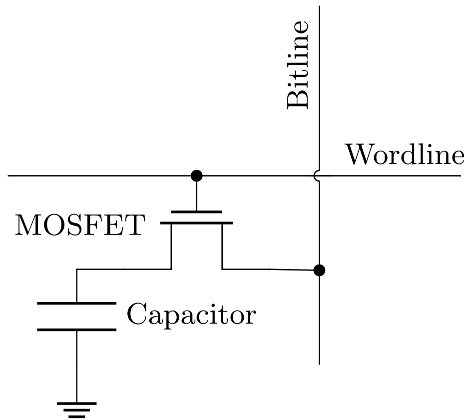

Figure 1.3: Schematic of DRAM memory cell made of an n-MOS and a capacitor. Adapted from Lacaze and Lacroix [LL14].

Corresponding to the capacitor charge, DRAM can store a '1' when the capacitor is in the charged state or '0' when the capacitor is discharged.

Programming and erasing are carried out by the transistor as in charge and discharge the capacitor. Reading is done by reading the charge of the capacitor through the bit line. The disadvantage of the process is that reading the capacitor charge literally means destroying the charge and hence the need to refresh the DRAM frequently in order to retain the data in the capacitor.

The advantage of DRAMs is their simplicity and high endurance cycling (which is greater than 10^6 cycles), which allows them to handle a great amount of program/erase cycles as well as their high speed of erasing. That makes DRAMs suitable for run-time logic operations.

The disadvantage is, as mentioned, the need for constant refreshing as the charge stored in the capacitor deteriorates over time, or after a read operation, the charge is completely destroyed and needs refreshing again. This means the capacitor requires refreshing approximately every millisecond [LL14; HC12].

Non-volatile Memories

Kahng and Sze have introduced the MOSFET with a metallic floating gate separated from the control gate by insulating layers in the 1960s at the Bells Labs [KS67].

When the floating gate is not charged (i.e. neutral state), the operation of the MOSFET device will be as normal. Given the positive charge on the control gate will create a channel in p-substrate and carry the current from source to drain.

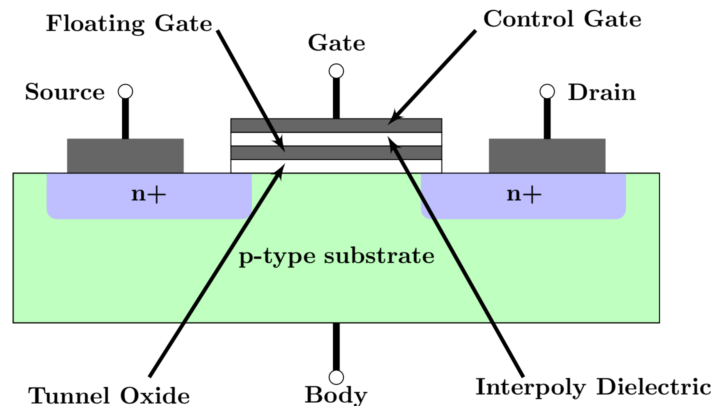

Figure 1.4: MOSFET with a floating gate. Metal sections are represented with dark grey, and insulating metal-oxide layers are represented in white. Adapted from [LL14; Bez+03]

On the other hand, when the floating gate is charged negatively, the charge shields the channel region from the control gate and restrains the channel forming between source and drain to form, thus induces a shift in threshold voltage. Given the positive charge applied on the control gate to turn the transistor conductive, the presence or absence of a charge in the floating-gate will result in a more negative or more positive threshold voltage. Hence, creating a window between 1 and 0 states corresponding to the existence or a charge on the floating gate [KS67; LL14].

The floating gate is surrounded by isolation layers which cause the charge not to be able to escape thus giving the transistor non-volatile characteristics [LL14; Cri+96]. This characteristic also is an advantage in terms of data retention (approximately 10 years [LL14; Mod99]) as well as the read operation. Read operation will not be a destructive process as it is in DRAM cells [LL14].

Erasable programmable ROM (EPROM) is a non-volatile as well as a programmable device. EPROM uses a special cell to store data compared to volatile devices such a DRAM as explained in Section 1.1 [LL14; HC12]. To erase data, EPROM has to be exposed to strong ultraviolet (UV) light for a certain amount of time, which also means that the chip has to be removed from the system when an erase operation is needed. The erase operation can take between 10 to 30 minutes [HC12; Ari+93].

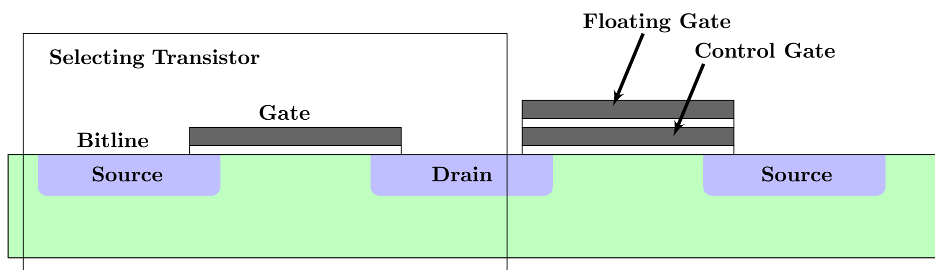

Figure 1.5: Cell structure of EEPROM with the additional selecting transistor. Adapted from Hai and Chen [HC12].

Removing the memory from the system when an erase is needed is not feasible for many applications. Thus electrically erasable and programmable ROM (EEPROM), which can be electrically programmed and erased in the system, has solved this problem. The basic memory cell of EEPROM, shown in Figure 1.5, is similar to as in EPROM in Figure 1.4. There are both the control gate and the floating gate. However, the oxide layer between the floating gate and the control gate is a lot thinner in the EEPROM cell compared to the EPROM cell which allows the EEPROM cell to be electrically erasable by making electron transportation easier. In addition to that, in some designs, a selecting transistor is used to avoid disturbance between cells in the memory array, which causes the EEPROM cell to be larger than an EPROM cell [HC12].

Programming is done by applying a high voltage pulse to the control gate and connecting the drain to the ground by letting the electrons flow through the drain to the floating gate. As opposed to the programming operation, when the control gate is grounded and a high voltage pulse is applied to the drain, the electrons flow from the floating gate to the drain thus the stored data in charge

is erased [HC12; LL14]. Here both programming and erasing operations are done electrically in systems. A charge pump circuit is often used to generate the needed high voltage pulse [HC12].

Flash memory, which is a type of EEPROM, is another electrically erasable and programmable non-volatile memory. Flash memories are explained more in detail in the next sections.

1.2 Flash Memory

Flash memories had explosive growth in the semiconductor memory market with the advantage of combining the nonvolatile storage capability and the access time comparable to DRAMs [Bez+03; Pav+97; Mas+85]. Based on the market requirements, a common way to classify Flash memory products according to their application segments is given as:

- Code storage, direct in-place execution
- Data/mass storage

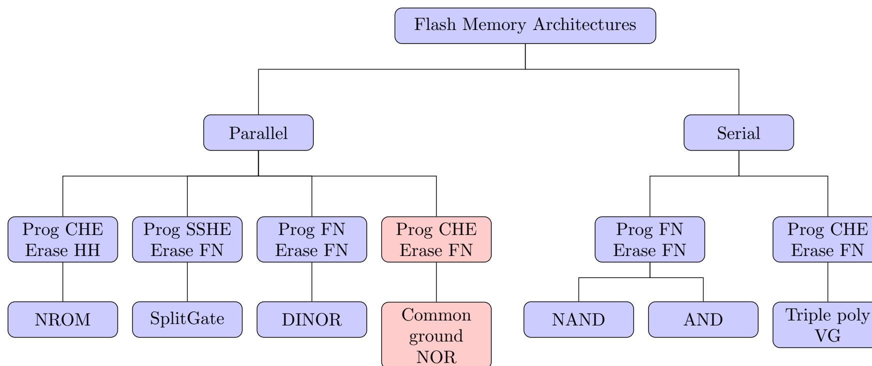

Figure 1.6: The classification tree of Flash memory architecture according to their access type and program/erase method. Adapted from [Bez+03; Pav+97]

Flash memory architecture that has been proposed over time can be classified in terms of their access type: parallel or serial. From there, they can be further divided according to their program and erase mechanisms [Bez+03; Pav+97; Ari+93]:

- Fowler-Nordheim Tunneling (FN)

- Channel Hot Electron (CHE)
- Hot-Holes (HH)
- Source-Side Hot Electron (SSHE)

Figure 1.6 demonstrates the classification of Flash memory architecture. NOR Flash and NAND Flash became the industry standards among others shown in Figure. NOR Flash with its versatility in addressing is used for program storage and execution in place as well as data storage, while NAND Flash is mainly used in the data storage market [Bez+03; Pav+97; Lai98; PB99; Mas+87; Ari00].

Throughout the study, a *page* refers to the smallest block that can be erased in a NOR Flash memory since erase operation is carried out block-wise (*page-wise*). A Page consists of a number of bytes, i.e. 512 bytes for ASIC2 internal Flash [Boe18], 1024 bytes for the microcontroller PIC32MX150F128B by Microchip [Mic19]. The terminology employed should not be confused with the commonly used terminology in NAND type Flash memories, where a page refers to the write-width and a sector/block refers to the smallest erasable area.

The rest of the thesis will focus on the concept of operating characteristics and reliability issues of NOR Flash cell since the memory under study is a NOR type Flash memory.

NOR Flash Cell

A Flash cell is a floating-gate MOSFET as presented previously in Figure 1.4. Electrically isolated floating-gate behaves as the storing electrode for the cell. A Flash memory cell is a synthesis of EPROM and EEPROM memories described previously, in the sense that a Flash cell can be programmed and erased electrically as EEPROM but a Flash cell is a single floating gate transistor as EPROM without an additional selecting transistor [Bez+03; Pav+97; PB99].

The name of Flash is given to represent the fact that the whole memory array can be mass-erased at the same time [Bez+03; Ari+93]. The name NOR Flash, on the other hand, comes from the memory cell arrangement since the rows and columns are in a NOR-like structure as demonstrated in Figure 1.7. Word line consists of Flash cells sharing the same gate, while the cells that share

the same drain forms the Bit line. Since the source is common to all of the Flash cells the structure is called a common ground NOR Flash as seen in the Figure 1.6 [Bez+03].

The thickness of the dielectric layer guarantees the non-volatility while allowing the possibility to program and erase electrically. The dielectric layer separating the floating gate and the substrate is called the tunnel oxide, since it is where the tunnelling of electrons happens, and it is in range of 9-10 nm [Bez+03]. The dielectric layer between the floating gate and the control gate is a triple layer of oxide/nitride/oxide (ONO) interpoly dielectric. The ONO thickness of the Flash cell is in range of 15-20 nm and the thickness of the interpoly dielectric is the major factor that affects the program/erase speed [Bez+03; Pav+97; Mor+96].

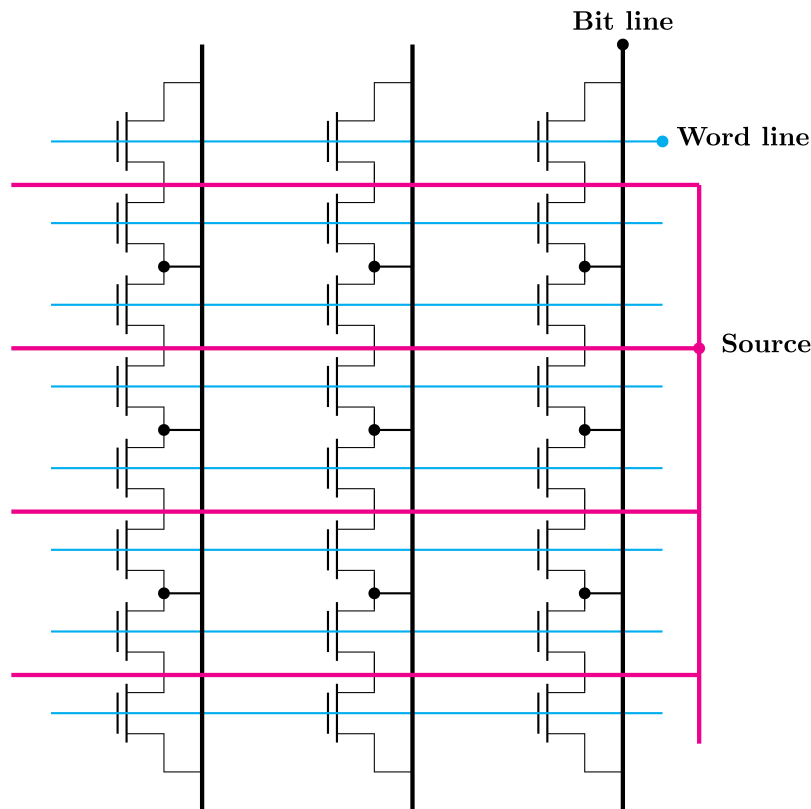

Figure 1.7: NOR Flash memory array equivalent circuit. Adapted from Bez et al. [Bez+03].

Operating Characteristics

As mentioned in previous sections, both the logical states "1" and "0" are associated with the electrons stored in the floating gate or the lack of them depending on the state. The positively charged state represents the logical state "1", while the negatively charged state of the floating gate represents the logical state "0" [Bez+03].

Reading operation is carried out by measuring the threshold voltage of the floating gate MOS transistor. The most efficient way to achieve this is to read the current driven by the cell. Figure 1.8 demonstrates the logical states "1" and "0" on the current/voltage bias plane and how they are shifted by the quantity of the threshold voltage shift, ΔV_t.

The quantity of the threshold voltage shift is proportional to the stored electron charge, Q and inversely proportional to the internal capacitance between the floating gate and the control gate, C_{FC}. As seen in Figure 1.8, the logical state "0" is defined by zero reading current and electron charge stored in the floating gate. As opposed to that, the logical state "1" is represented with no electron charge on the floating gate and a large reading current [Bez+03; Pav+97].

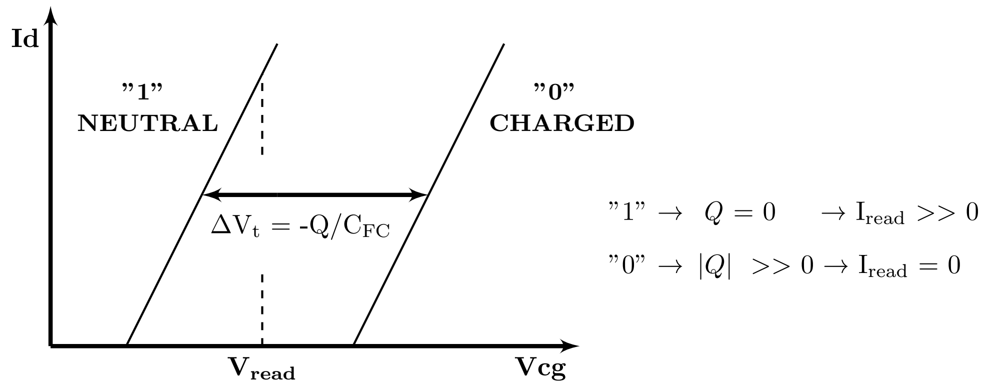

Figure 1.8: Floating-gate MOS reading operation, where Q is stored in the floating gate and C_{FC} is the capacitance between the floating gate and the control gate. Adapted from Bez et al. and Pavan et al. [Bez+03; Pav+97].

Programming operation is done by applying pulses to the control gate and the drain concurrently while the source is grounded [Pav+97]. Programming the floating gate cell leads to the physical issue of forcing an electron above or across an energy barrier. There are several physical mechanisms, as previously demonstrated in Figure 1.6, proposed to work around this problem that exploit various physical effects [Bez+03; SF99].

NOR-type Flash memory cell is programmed by the channel hot electron injection (CHE) mechanism in the floating gate at the drain side [Bez+03]. CHE mechanism originates from energizing the carriers through the source to the drain as a result of the lateral electric field that is produced by the drain-source voltage (V_{DS}). Here the electrons gain sufficient energy to be able to penetrate through the oxide layer energy barrier thus programming the floating gate [Bez+03; SPC84].

Read, program, and erase bias configurations of the floating gate Flash cell are shown in Table 1.2.

	SOURCE	CONTROL GATE	DRAIN
READ	GND	V_{CC}	V_{READ}
PROGRAM	GND	V_{PP}	V_{DD}
ERASE	V_{PP}	GND	FLOAT

Table 1.2: Source, Control Gate, and Drain Biases during typical Flash memory operations. Reference values are $V_{CC} = 5V$, $V_{PP}=12V$, $V_{DD}=5\text{-}7V$, and $V_{READ}=1V$. Adapted from Pavan et al. [Pav+97].

Erase operation requires a high voltage pulse on the source while the control gates in the word line are grounded, and at the same time, the drains on the bit line are floating [Pav+97]. Flash cell is erased with the Fowler-Nordheim electron tunneling mechanism. Fowler-Nordheim electron tunneling is a quantum mechanical tunnel induced by an electric field. A strong electric field is applied across the tunnel oxide layer from the floating gate to the silicon surface, which allows a large electron tunneling current to flow without destroying the dielectric properties of the oxide layer [Bez+03].

1.3 Flash Memory Reliability

Vast growth in the demand for high capacity and the application and the drastic increase in technology scaling, as Moore's Law defines, raises an essential need for reliability studies. The scaling of the devices alone leads to more problems during manufacturing since devices become more prone to process drift and defects as they get smaller and smaller [God08].

The probability of a system maintaining its expected functionality under the given situations for a defined period of time is characterized by the concept of reliability. Failure is the point where the system does not perform its required function [God08]. The concept of reliability of a system is quantified by:

- Probability of failure
- Required function of the system, which defines the set of tasks the system is expected to achieve
- Operating conditions
- Operating time

While the concept of failure is classified as:

- Failure mode
- Failure cause
- Failure effect

The mode defines the symptoms, the cause is the origin of the problem, which can be external or internal, while the effect is the result of the failure, which can be critical or non-relevant [God08]. In the end, most importantly, the life-time quantifies the service time provided by the system, where the system maintains the expected functionality without failure. The curve shown in Figure 1.9 demonstrates the life cycle of a system. The first region represents the early lifetime failures (Infant mortality), where the failure rate deteriorates to the second region which is the useful operating time (Useful lifetime) of the system, and with time and the aging failure rate increases back, where the system becomes worn out.

This section aims to explain the floating gate reliability and why Flash memory has a limited lifetime in terms of program/erase

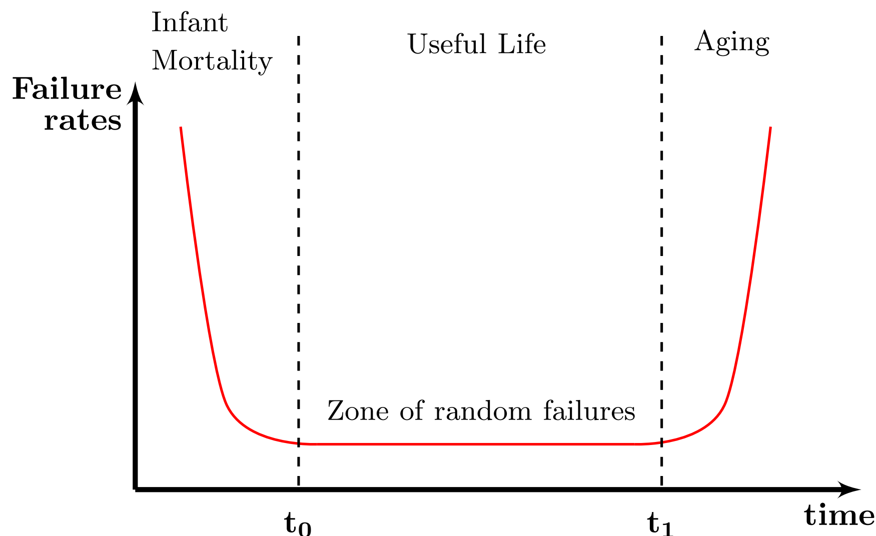

Figure 1.9: Failure rate distribution of a system during its lifetime. Adapted from Godard [God08].

cycles and data retention. The most common failure mechanisms such as threshold voltage distribution, program disturbances, over-erasing, data retention, and most importantly program/erase endurance will be explained in detail.

Program/Erase Threshold Voltage Distribution

As it was explained in the chapter 1 *Theoretical Background*, when programming and erasing the floating gate device, a voltage shift occurs in the threshold voltage seen from the control gate [Bez+03; Pav+97; Bre07]. From the voltage shift in a single memory cell, the type of distribution of the threshold voltage becomes critical when it comes to big-scale memory arrays. The scientific paper, which introduces every aspect of Flash memories by Bez et al. [Bez+03], demonstrates in Figure 1.10 how threshold voltage distribution differs between UV erase, CHE programming, and FN erase operations.

As is seen in Figure 1.10, the UV erasure operation has a quite symmetrical distribution, together with CHE programming. The programming distribution is a bit wider compared to the UV erase, it is explained in as the parameters which affect the shift in UV also influences the threshold shift of programmed cells [Bez+03].

When it comes to erasing with the FN method, Figure shows that the distribution of threshold voltages is asymmetrical and wider compared to CHE and UV erase. This phenomenon is explained with the "*tail*" cells since except the exponential tail the voltage distribution would have been symmetrical. *Tail cells* are the cells which erase faster than the average cells with the same applied voltage [Bez+03]. Tail cells can not be affiliated as an extrinsic defect since they represent a large population in the memory array [Bez+03; Pav+97; Mod99].

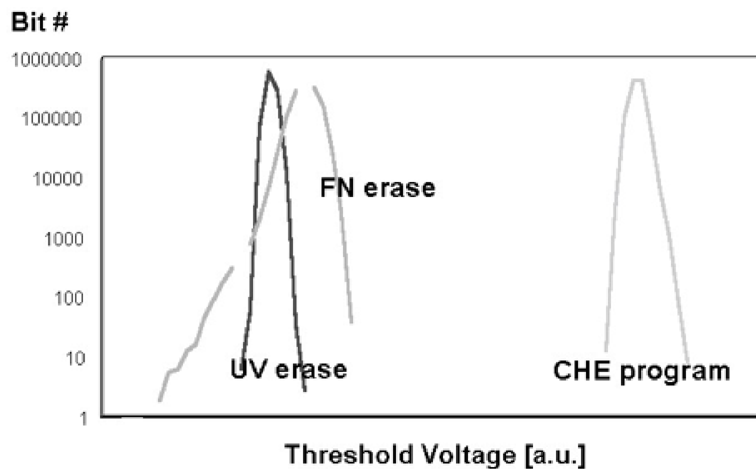

Figure 1.10: Threshold voltage *window closing* distribution over program/erase cycles on a single cell. Retrieved from Bez et al. [Bez+03].

Verification of erase operation is done by checking the shift in threshold voltage in the memory array. Tail cells, erasing faster than typical cells, cause a big issue in terms of verification of erase operation in the Flash memory page, which is one of the most significant issues in Flash technology [Bez+03; Pav+97]. Proper design considerations against these tails must be taken while designing Flash structures.

Program Disturb Mechanisms

The Flash memory array can experience two types of disturb mechanisms during programming [Ari+93; Bez+03; Pav+97]:

- Drain disturb (Column disturb or program disturb)
- Gate disturb (Row disturb or DC program disturb)

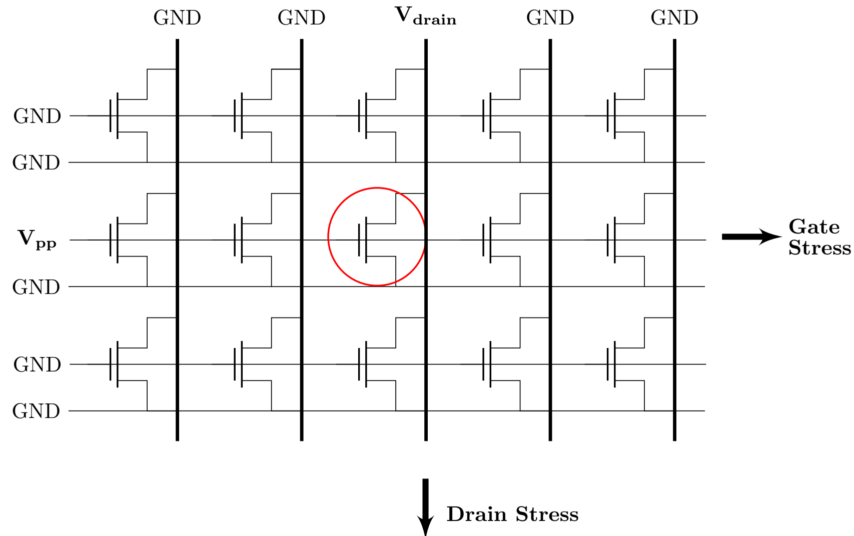

Figure 1.11: Flash memory array showing row and column disturbs (gate and drain disturbs accordingly) mechanism when the cycled cell is programmed. Adapted from Bez et al. [Bez+03].

During programming other cells in the memory array, already written cells can get disturbed due to the electrical stress applied. Two mechanisms are represented in the Figure 1.11 showing the memory array [Bez+03; Ari+93], where memory cells with common word-line (WL) or a common bit-line (BL) are concerned.

Drain Disturb

Column (or drain) disturbs occur because of the drain stress on the cells sharing same the bit-line with other cells, which are being programmed at the time. The high electric field between the floating gate and the drain may cause electrons to tunnel from the floating gate to drain. Hence cell loses charge due to FN tunneling, which causes a reduction in the threshold voltage (soft erasing) [Bez+03; Ari+93; Cap+94].

The drain disturbance mechanism depends on the number of the cells on the bit-line as well as the word-line and their organizational structure. Since the disturbance depends on the Flash memory page organization, the most efficient way to handle the drain disturb is to have a selecting transistor to separate the Flash pages and isolate them from each other. Drain disturbance may lead to critical issues in the Flash memory cell, and it is essential to mini-

mize the occurrence of the disturbance [Bez+03; Cap+94; Ari+93].

Consecutive program/erase cycling influences the drain disturb. Although Aritome et al. [Ari+93] mention that the article by Verma et al. [VM88] has shown that up to several thousand cycles the drain characteristics showed no measurable variation at all.

Gate Disturb

The gate stress causes the row disturb on the unprogrammed or erased cells sharing same the word-line with the cell, which is being programmed. Unprogrammed/erased cells have low threshold voltage as the Figure 1.8 in the chapter 1 *Theoretical Background* presented before. A high voltage is applied to the row during programming and those cells must be able to handle the stress and maintain the data integrity. Electrons may tunnel from the substrate to the floating gate when the electric field across tunnel oxide rises. Either the leakage in gate oxide, or the leakage in the interpoly dielectric may cause the loss of data. The increase in threshold voltage may lead to the point where the cells have a chance to be programmed incidentally [Ari+93; Bez+03].

As previously explained, up to a certain program/erase cycle, drain disturbance characteristics are excellent and the cell functions properly as intended [VM88; Ari+93]. Yet, program/erase cycling still affects the gate disturb characteristics by reducing the gate disturb time drastically and the main reason for that is the hole trapping during the erase operation [VM88; Ari+93].

The leakage current on the bit line can influence the reading operation. The reason for the leakage is that the cell transistor can switch into depletion mode because of the positive charge on the floating gate left by the erase operation, which is called *over-erasing* [Ari+93; Pav+97; Mod99; SL00].

Data Retention

Flash memories being a non-volatile memory by definition should be able to retain the integrity of the stored data over a defined certain period.

Therefore, the floating gate must be able to retain the charge as long as possible with the minimum amount of loss. The stored charge can leak away from the floating gate either through the

tunnel oxide or through the interpoly dielectric. There are various reasons for leakage to happen [SCE80; Mie83; Ari+93; Bez+03; Lee+01]:

1. Tunnel oxide defects
2. Interpoly dielectric defects
3. Electron detrapping
4. Mobile ion contamination
5. Thermionic emission

Firstly, oxide defects can lead to either charge loss or charge gain [Ari+93]. The defects can stem from either the physical structure of the memory cell or the structure of the device thus the defect can originate from both internal or external effects respectively [Bez+03].

Insulating layers around the floating gate may end up trapping electrons during the manufacturing process or the wafer [Ari+93]. The trapped electrons will eventually detrap mostly with the effect of high temperature causing a shift in the floating gate potential. Even without a leakage happening, there will be a charge loss due to the detrapped electrons around the floating gate [Ari+93; Bez+03; Cal+13].

Next, the charge loss due to mobile ions is the result of the positive ions penetrating the cell. When the cell is in the programmed state, thus the negative charge, incoming positive ions will offset a portion of the stored negative charge thus leading to a charge loss in the cell [Ari+93]. The solution to this issue in EPROM technology is already existing by localizing the mobile ions in a predetermined region in the silicon wafer by using high phosphorus content, a gettering element, in the intermediate dielectric [Bez+03; Cri+90; Mod99].

Thermionic emission is the phenomenon of freeing of an electron due to the temperature. This charge loss mechanism is not the most dominating, among the others mentioned, due to the fact that the activation energies for the rest of the failure mechanisms are considerably lower than the image force lowered barrier of thermionic emission [Kiz+08; Ari+93]. Thermionic emission becomes rather prominent in terms of charge loss in high-temperature

conditions [Pav+97].

The data retention in Flash memories is quantified by accelerated stress tests in harsh conditions such as high temperatures. [Ari+93]

Write/Erase Endurance

The capability to handle the stress of consecutive programming and erasing cycling is defined by the term endurance. Endurance is quantified by the minimum number of program/erase (P/E) cycles the memory can handle without a single-bit failure [Bre07; Cap+94; Bez+03; Mod99].

One of the most significant limiting factors in Flash memory reliability is the deterioration in cell performance caused by constant cycling. P/E endurance originates from mainly two factors [Cap+94; Mod99]:

1. The shifting in the program and erase threshold voltages due to oxide aging (also called window closing)

2. Tunnel oxide related single bit failures

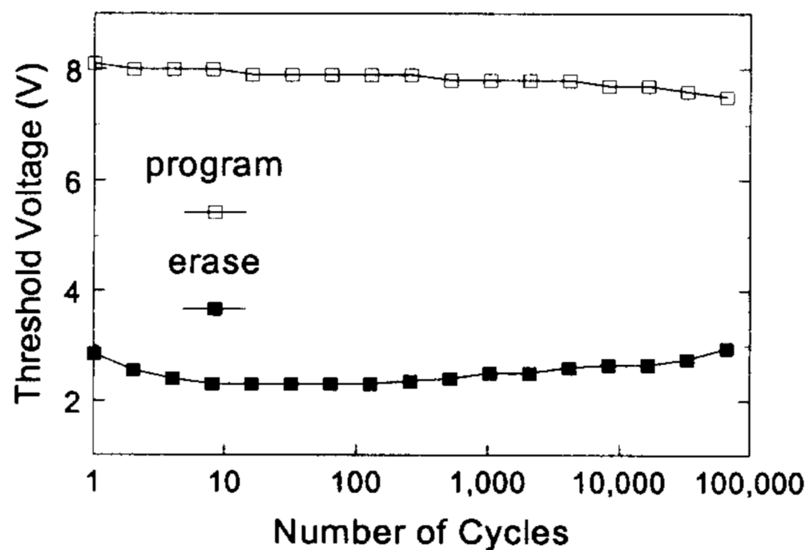

Figure 1.12: Threshold Voltage (V_t) of Programming and Erasing Window with Constant Program/Erase Cycle. Retrieved from Modelli [Mod99]

The shift in the threshold voltage of programming and erasing operations is caused by the degradation in the oxide layer and this failure mechanism is pretty consistent and easy to recreate [Mod99; CAO08; Had+89]. The oxide layer starts to wear out due to the constant stress of P/E cycling. A very typical result of a stress test is demonstrated in Figure 1.12.

As Figure 1.12 presents, the threshold voltage of the programmed state diminishes as the P/E cycle grows, this phenomenon is due to the electron trapping in the oxide. On the other hand, the threshold voltage of the erased state after an initial reduction starts to rise as Figure 1.12 shows. The piled-up positive charge intensifies the tunneling efficiency and causes the initial reduction in the threshold, yet the rising trend is induced by electron trappings [Mod99; Bez+03; Cap+94].

The stress test is conducted in a way that pulses are applied continually to the memory cell to stress it out, and the threshold voltages of programmed and erased states are obtained. The window is the difference between the threshold voltages of erased and programmed states. The threshold voltages shifting to each other is the reason the failure mechanism is called window closing. The important issue with window closing is that during reading the sense amplifier must be able to precisely distinguish the difference between programmed and erased states.

The program and erase timings getting longer as shown in Figure 1.13 is an outcome of the threshold window closing [Pav+97; Had+89; Bre07].

The current density increases with the constant high field stress applied to the thin oxide layer, which ends up affecting the voltage-current curve in FN characteristics. Stress-induced leakage current (SILC) is a failure mechanism associated with the stress-induced defects on the oxide. Stress field and the oxide thickness are the main parameters that affect SILC [Bez+03].

Endurance failures often present themselves as a single-bit failure in a Flash memory page after a certain amount of P/E cycles. During chip-level testing, frequent pulses are applied to the memory cells, hence it is important to design a firmware-level endurance test similar to the chip-level test [Bez+03].

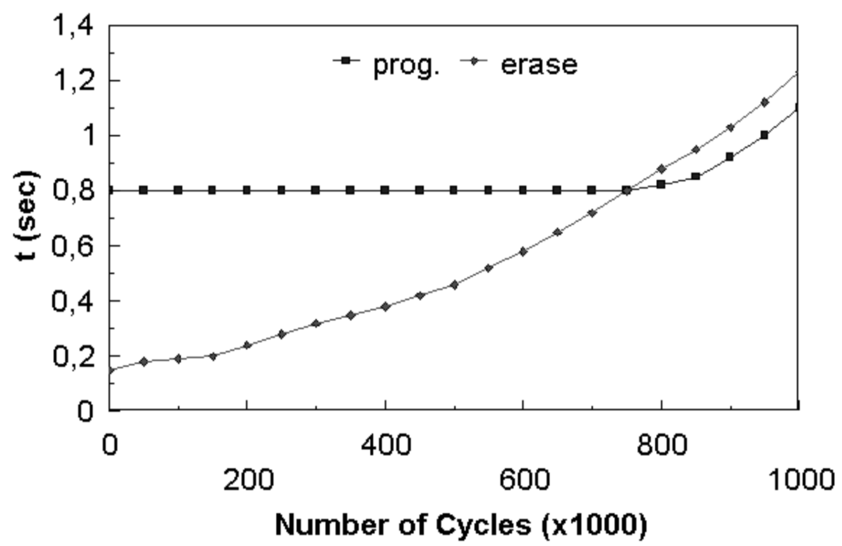

Figure 1.13: Program and Erase Timings. Retrieved from Bez et al. [Bez+03]

Chapter 2

Design and Implementation: ASIC2 Endurance Test Algorithm & Setup

2.1 ASIC2 Internal Flash Memory

ASIC2 is the second generation of Application Specific Integrated Circuit specifically designed to support SmartWire-DT (SWD) [Eat]. The ASIC2 is based on dual core 32bit ARM technology (ARM Cortex – M0+) with extended functionalities. ASIC2 comes in two variations of packages as shown in Figure 2.1, where the bigger version has additional features. The features of ASIC2 do not only support the SWD communication but can be also used with the integrated RS485 PHY for Modbus (RTU/ASCII) instead. Since ASIC2 already has a support for Modbus interface, Modbus RTU is used to log the test data through a simple Python script.

ASIC2 internal NOR Flash, in which the Dataflash module will run, is a form of electrically erasable and programmable read-only device, specially developed for program store and execution. Flash memory has a total size of 88kB. ASIC2 internal Flash is organized in blocks, (*pages*), made of 512 bytes. The erase operation is either page-wise or mass (hence the name *flash*). The write operation is limited to 32-bits (also can referred as *double-word*, or *write-width*) due to the design technology, while the read operation can be used to obtain data in 8-bit, 16-bit or 32-bit data width from the memory.

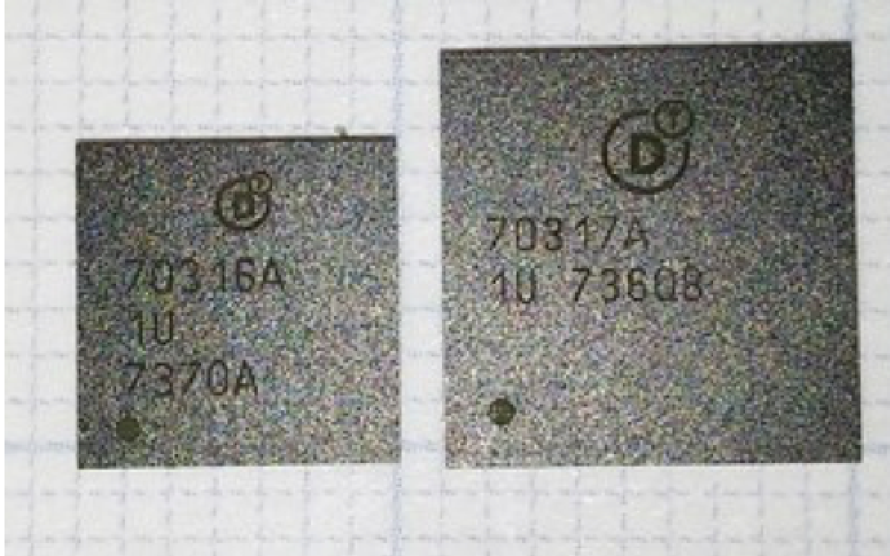

Figure 2.1: ASIC2 in two different packages

The ASIC2 internal Flash memory is claimed to have endurance cycle of 20,000 in the datasheet [Boe18]. The endurance cycle and retention tests are done by a third party company and the testing method or the details are not provided. Thus the aim of the study as explained in the *Problem Statement* is to quantify the confidence interval of the failure rate in Flash memory pages for the given endurance cycle and come up with a proposal to improve the datasheet endurance from the application-side.

2.2 Endurance Test Algorithm

Endurance Test Algorithm on Testbed

The floating gate due to its physical nature is the cause of the reliability issues in Flash memory cells [Mar+; Bez+03]. The reliability problems of the Flash memory array are data retention and program/erase endurance.

Data retention is the ability of the Flash cell to retain the integrity of the stored data for an adequate period of time. The main focus of this study is the endurance issue, which defines the number of times the Flash cell can be programmed and erased successfully while preserving data integrity without disruption. Each program and erase cycle introduces defects to the cells [Pav+97; Had+89].

A Flash page is assumed to be destroyed even if a single bit failure occurs. Testing the endurance cycle demands stressing the Flash page with consecutive programming and erasing [Gin+06].

March algorithms are well-known for memory testing, mainly testing embedded RAMs, due to the extensive fault coverage and simplicity [Mar+; M P12; Che+04; Kuo+02; Yeh+07]. The chapter 2 - *Design and Implementation* previously explained that Flash memories can not perform erase operation in random access fashion as RAMs do, but instead Flash memories erase page-wise or as a whole. Hence March test algorithms designed for RAM will not be feasible for Flash memory testing [Goo98; Mar+]. Although it is still feasible to design modify March-like test algorithms to test Flash memories from the application-side. March-FT algorithm introduced in papers [Mar+; Kuo+02; Yeh+07] can cover a large scale of disturbances except disturbances that requires consecutive read operation.

The operations used in the March-FT test are [Mar+]:

- READ: Read operation
- PROGRAM: Program operaton
- CHIP_ERASE: Mass erase the entire chip

The extended version of March-FT test is proposed by as:

CHIP_ERASE;
$\Uparrow$(READ($\sim$D)^n, PROGRAM(D), READ(D)^n);
$\Uparrow$(READ(D));
CHIP_ERASE;
$\Downarrow$(READ($\sim$D)^n, PROGRAM(D), READ(D)^n);
$\Downarrow$READ(D).

In which:

- D is the test pattern (i.e. solid 0s or all 1s)
- $\sim$D is the D pattern inverted
- n is how many times the operation is repeated
- $\Uparrow$ Address increment
- $\Downarrow$ Address decrement

March-FT algorithm explained above performs chip erase, and programs and reads consecutively. Since the point of the thesis is to only find the program/erase cycle where a particular Flash page in test encounters the first bit failure, the extended March-FT test presented is simplified by modifying the test operation as to be similar to the chip-level stress tests [Pav+97]:

- PROGRAM_PAGE: Program 512 byte page by programming 32-bit into the page 128 times consecutively to fill the entire page.
- READ_PAGE: Read 32-bits 128 consecutively.
- PAGE_ERASE : Erase the page with given physical address.

The simplified version goes as follows:

$\Uparrow$PAGE_ERASE;
$\Uparrow$READ_PAGE(∼D);
$\Uparrow$PROGRAM_PAGE(D);
$\Uparrow$READ_PAGE(D);

In which D pattern is chosen as solid all 0s and $\Uparrow$ represents physical page address increment. Every read page operation is followed by a comparison function to be able to identify the bit failure. In addition after every erase operation the page is read and compared with a solid all 1s pattern to ensure erasing was successful. In case a bit error is recognized in the page, physical page address is incremented until a certain amount of pages in the internal Flash memory is tested.

At the time a page is recognized as failed, the data get logged via Modbus RTU and received from the computer via a Python logger script. The log data include:

- ASIC2 index
- Failed page index
- Double-word address associated with the failed data
- Failed data
- Write/Erase cycle
- Time stamp

The log files are saved in a separate text file for each ASIC on the test-bed, which will be presented in the section 2.3 - *Endurance Test Hardware* in detail. The log text file example, for the ASIC2 with the Modbus address of 9, can be seen here:

```
9,64,0x2a0d0,0xbfffffff,0x86ad8,2020-02-11 14:11:28
9,65,0x2a38c,0xbfffffff,0x87e7b,2020-02-11 14:11:28
9,66,0x2a484,0xffefffff,0x73a68,2020-02-11 14:11:29
```

Later on, after obtaining all the data, the text logs are sorted and parsed with a Python script to an Excel sheet to be analyzed.

Endurance Test Algorithm on Evalboard

Before the testbed was designed and manufactured, initial testing of the endurance is done on the evaluation board for ASIC2 (referred to as *evalboard*). ASIC 2 evalboard is demonstrated in Figure 2.2. Here, the evalboard has access to almost all functionalities of ASIC chip.

The endurance test run on evalboard is essentially the same as the test algorithm method presented earlier. Minor differences are made in order to explore the program/erase endurance failure mechanism better. UART communication interface is selected to log the test data purely for the ease of implementation compared to Modbus RTU. The other difference from the testbed endurance test algorithm is that the stressing of programming and erasing did not stop after the first bit-failure occurred, the stressing of the same Flash page is rather kept going on around 1,5M program/erase cycles to have a big log data to understand how the bit-failure behave.

The log data contains:

- Program/erase cycle (there referred as write/erase (W/E) cycles)
- Number of errors has caught so far
- Type of error such as program (write) error or erase error
- Double-word address of the fail
- Failed data versus the previously read value of the failed data (referred as the *expected*)

The initial endurance test logs look like in the text file:

```
W/E Cycle 1, Page address 00022000, errors 0
W/E Cycle 2, Page address 00022000, errors 0
W/E Cycle 3, Page address 00022000, errors 0
```

In case of an error, the logs look like:

```
W/E Cycle 168926, Page start address 00022000, errors 0
ERROR:E dWord address 00022150 read FFFFFFFD expected FFFFFFFF.
W/E Cycle 168927, Page start address 00022000, errors 1
```

Here *ERROR:E* means that this error happened during *erase* operation on the *W/E cycle* 168,926, which means that the page did not erase properly. The read data and the expected data shows that a single bit got stuck at logical 0 and became D in hexadecimal instead of F in hexadecimal as expected from erase operation. Bit failure during programming (referred as *write*) would be indicated by *ERROR:W* in the log file.

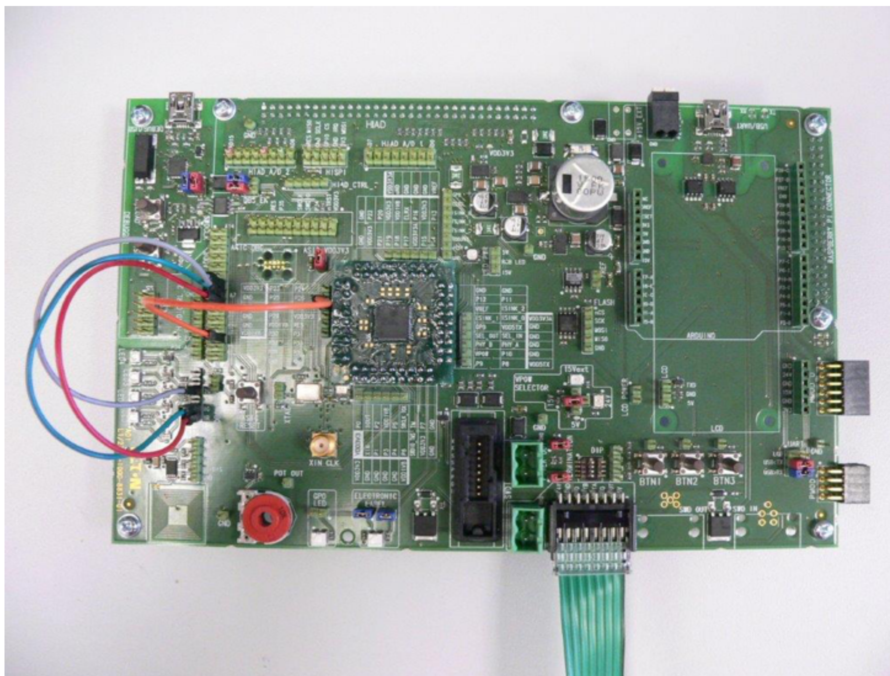

Figure 2.2: ASIC2 Evaluation board (also referred as *evalboard)*

Findings after the program/erase stress test on evalboard and how they affected some of the design choices of data management algorithm will be presented in forthcoming chapters.

2.3 Endurance Test Hardware

A test-board PCB (will be referred as the *testbed*) is designed, manufactured, and populated to run a firmware side endurance test to statistically quantify the confidence interval ASIC2 internal Flash memory reliability.

Later on, after running the firmware endurance test on the first testbed, another two more, but modified, testbeds are populated to validate the firmware test results by testing from the lower level, by using dynamic link library functions, compared to the higher level firmware testing.

Window Closing	**Oxide Breakdown**
Small sample of devices (20)	Large sample of devices (>100)
Cycle 10K times	Cycle 10K times
Test for program/erase	Test for program/erase
Repeat to >100K cycles	Program to non-equilibrium state

Table 2.1: Recommended endurance evaluation flow. Adapted from [Sha13].

The book *Semiconductor Memories: Technology, Testing, and Reliability* [Sha13] recommends an endurance evaluation flow shown in Table 2.1 in order to test for the two most common failure mechanisms in non-volatile memories: oxide breakdown and the trap-up failure (also referred as window closing). According to [Sha13], the ideal testbed would have 100 or more pieces of ASIC2 to test on, however the number of available ASIC2s was limited and the size of the testbed was required to be as compact as possible.

Pavan et al. [Pav+97] demonstrates their results of program/erase endurance stress test on memory array with the size of 1-Mb. Taking the data from Pavan et al. into consideration, the testbed was roughly estimated to have 30 pieces of ASIC2, since a Flash page is made of 512 bytes and each ASIC2 will have at least 60 Flash pages available to be tested.

It was important to be able to power the each chip separately as well as flash and debug the firmware. Another requirement was to be able to address each ASIC2 individually to log the raw test data through Modbus communication protocol. At last some indicator LEDs were needed to indicate communication and to be used during debugging or maybe to check some other functionality.

MODBUS is a data communication protocol on application-layer, which is based on a client/server architecture. Modbus has became the recognized standard of communication protocols in industrial applications and it was published by Modicon [Ins09]. Modbus communication preferred as the data communication protocol to log the test data since ASIC2 has the capability of supporting Modbus communication as the ease of addressing each ASIC2 to collect data on the testbed from a logger script written in Python.

To sum up the testboard for ASIC2 Flash memory endurance test is designed with the specifications:

- Enough number of ASIC2 to do statistical calculations
- To be able to log test data via a standard communication protocol
- To be able to address each ASIC2 to log test data in parallel
- Power each ASIC2 individually
- To be able to flash firmware to ASIC2s
- Indicator LEDs

Testbed Schematics

Testbed consists of 30 ASIC2 chips in a 6x5 layout. Each cell has their own jumper to power on and off each ASIC2 individually, a connector to program, and as well as two indicator LEDs. Each cell is also addressed individually from address 1 to address 30 for Modbus communication. This section will demonstrate each section of schematics and their PCB layout that forms an ASIC2 cell on the board, the complete KidCad layout design is presented in Appendix A.

Figure 2.3 shows the powering circuit for an ASIC2, where *JP1* is used to cut the power from each cell if needed.

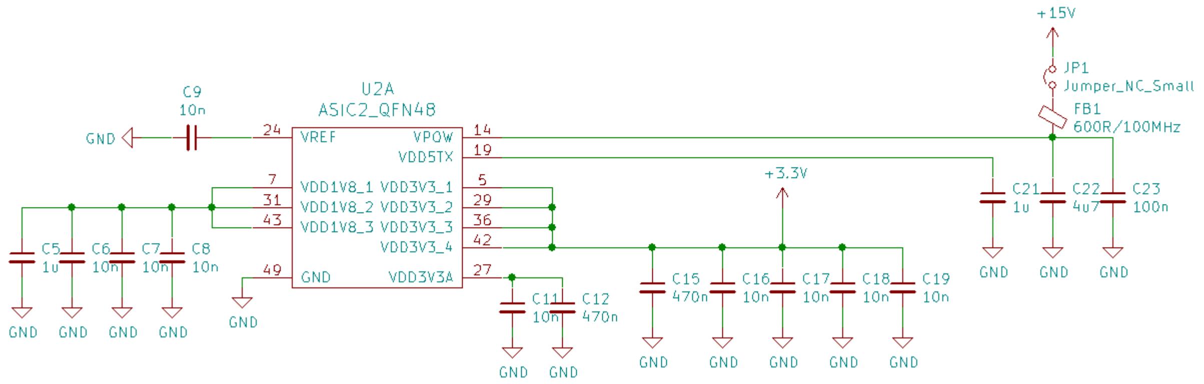

Figure 2.3: ASIC2 powering schematic

Figure 2.4 shows the ASIC2 connections, such as Modbus addressing pins, transmit/receive pins necessary for Modbus communication as well as the differential physical A and physical B signal pins. Two indicator LEDs are also, where green *D5* indicates the communication, *D3* is used for both debugging and to indicate that the endurance test on that particular cell is finished. *SWCLK_TCK, SWDIO_TMS, TDO_P35, TDI_P34* and $\overline{RES}$ are used to flash the firmware into the ASIC through JTAG hardware interface, where the connector is shown in Figure 2.6. Figure 2.5 shows the jumpers used to connect Modbus addressing to each cell. Each cell is soldered a hardcoded address from 1 to 30.

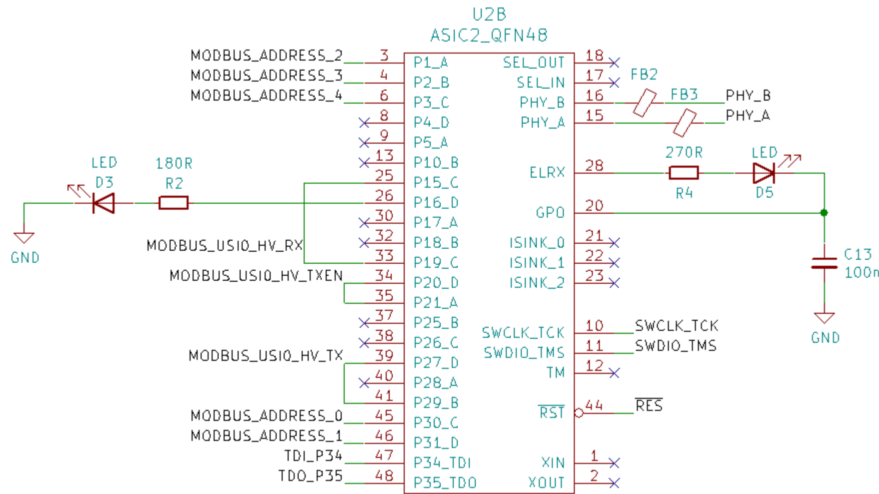

Figure 2.4: ASIC2 schematic showing the LEDs and necessary pins for Modbus communication and programming of ASIC

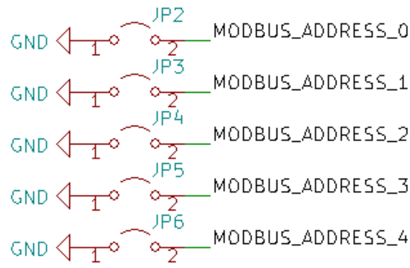

Figure 2.5: Modbus addressing connections of ASIC2

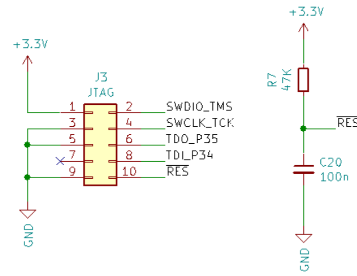

Figure 2.6: JTAG Connector and powering jumper

Lastly, the powering circuit schematic is presented in Figure 2.7, which is used to route power line to entire grid of ASIC2s. The physical A and B signal lines are also routed to the Modbus master with the same connector, *J1*.

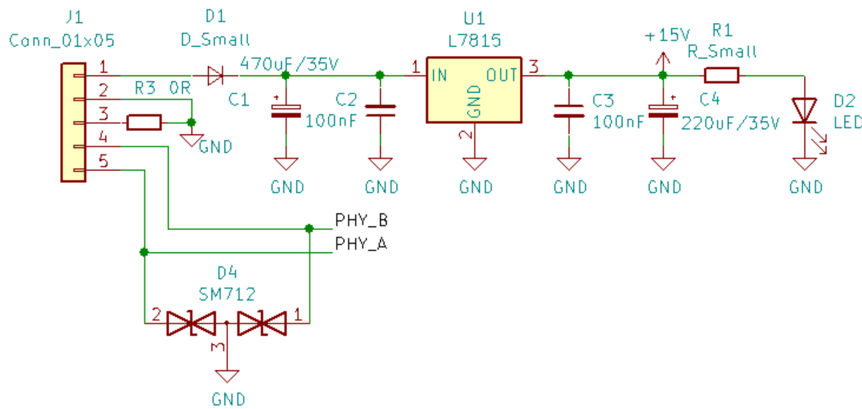

Figure 2.7: Testboard powering schematic

Complete testbed PCB layout can be found in Appendix A, while Figure 2.8 shows the layout of a single ASIC2 cell in the testbed. After a single cell is placed and routed, it was duplicated in 6x5 layout and the powering circuit is placed and routed on the right side of the PCB with the Modbus physical A and physical B lines, which are wired in a twisted pair fashion through all ASICs cells.

Figure 2.9 demonstrates the 3D view of the completed PCB layout where all components are picked according on their package area size and values. Finally Figure 2.10 demonstrates the manufactured and populated final version of the testbed board.

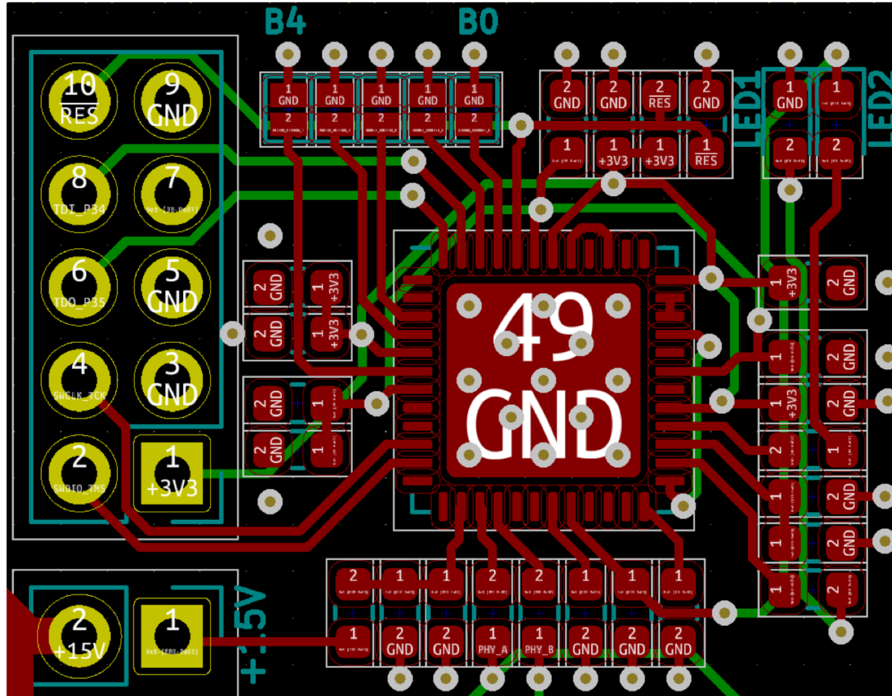

Figure 2.8: Layout drawing of a single ASIC2 cell

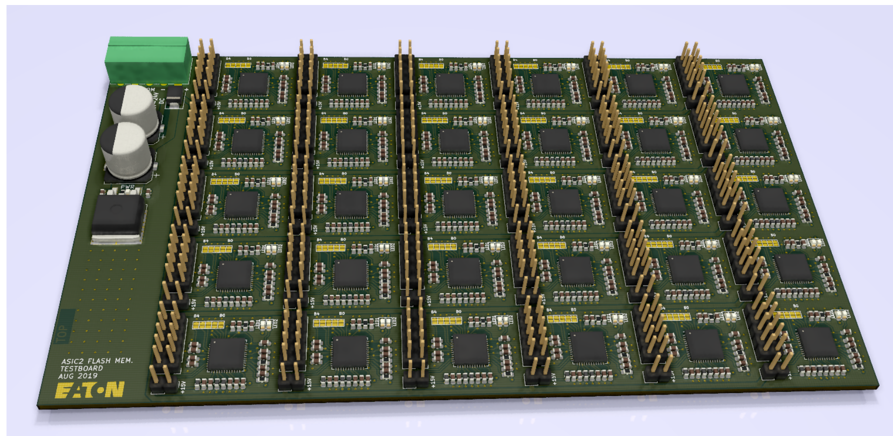

Figure 2.9: ASIC2 Flash memory testbed 3D view

As mentioned in the beginning of the section, the need for a validation test occurred after the first program/erase endurance test. In order to test on lower level using dynamic link library (DLL) functions, Modbus route through the entire PCB needed to be separated since the test was supposed to run when the ASICs are in bootloader mode to be able to use DLL functions and due to that it was not possible to address individual ASIC2s on the path at the same time.

Figure 2.11 shows the individual grey cables are connected to each ASIC2s Modbus physical A and physical B pins to be able to test each cell individually in bootloader mode using DLL functions. In total 6 ASIC2s on the first DLL testbed and later on 12 ASIC2 on the second DLL testbed were tested in order to validate the firmware endurance test data. Results and the validation methodology will be presented in the chapter 3 - *Evaluation of Test Results.*

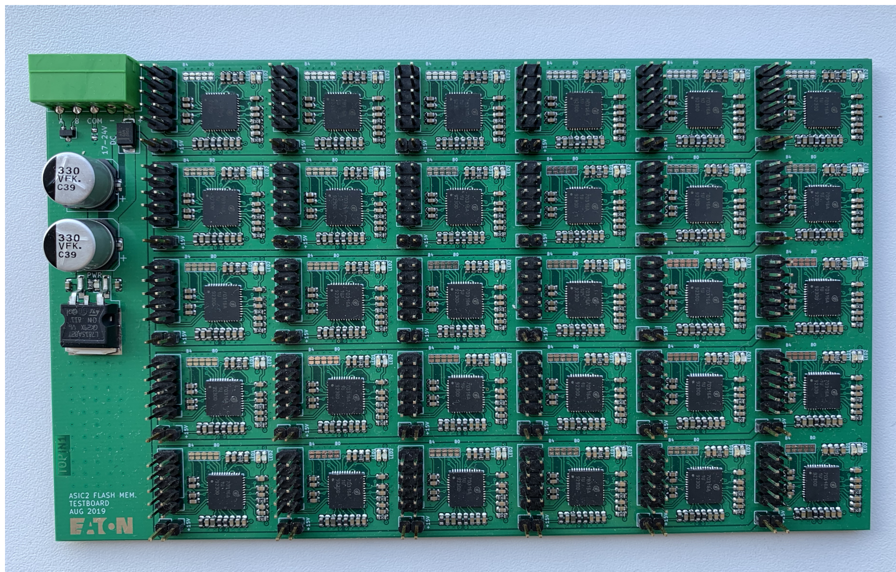

Figure 2.10: ASIC2 Flash memory testbed PCB manufactured and populated for Flash program/erase endurance testing

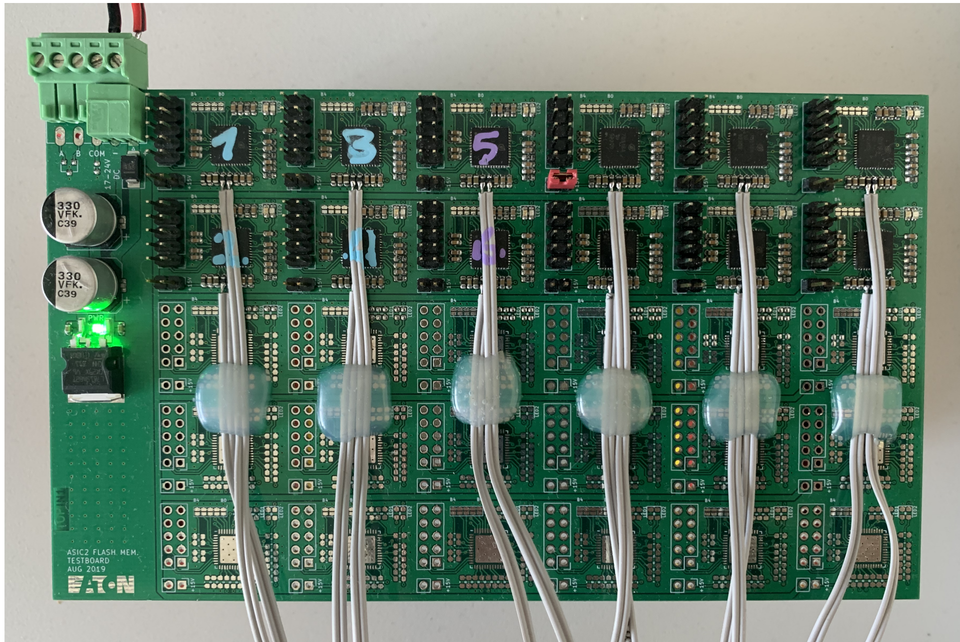

Figure 2.11: The testbed for DLL test to validate firmware endurance test

Chapter 3

Evaluation of Test Results: Endurance Test of ASIC2 Flash Memory

Previously in the chapter 1 - *Theoretical Background*, overall physical structure and failure mechanisms of Flash memory are explained. The device that will be tested, ASIC2 Flash memory, the test algorithm, and the test hardware designed for the endurance test of the ASIC2 are demonstrated in the chapter 2 - *Design and Implementation*.

In the beginning, before testbed PCB was manufactured and populated, some initial runs of the test algorithm have been done on the evaluation board (evalboard) of ASIC2. The test run on evalboard was slightly different than the actual test algorithm in terms of logging interface and the stressing of the memory. The analysis of the results from evalboard are interpreted in the section 3.1 - *Initial Endurance Test on Evalboard*.

Next, the section 3.2 - *Endurance Test on Testbed* will remind the data log format and explain the assumptions taken to sort the test results. Later on, the statistical analysis choice and the way the analysis is carried out are presented with the calculated confidence intervals for the choice of the confidence level of 80%. In the end, the validation test is introduced, which was required unexpectedly during endurance testing.

3.1 Initial Endurance Test on Evalboard

Evalboard test was run on a 3 of pages. The pages have been stressed approximately more than 1M cycles (~1-1.5M) and the information during each cycle is logged on UART. A simple Python script kept track of the logs and after the test is completed, all raw data from UART are parsed to a text file. The size of the raw data exceeds 1GB of text for each page. The format of the endurance test on evalboard logs was presented earlier in the chapter 2 - *Design and Implementation*, where the number of total errors and the double-word location of each error as well as the program/erase cycles were documented.

From the logged data it was observed that the bits, which failed previously, have a chance to recover back to behaving normally and fail again or keep functioning properly after a certain amount of program/erase stress. This behavior can be explained with electron de-trapping in the oxide layer, which also further proves that the entire Flash page becomes completely unreliable after the first bit-failure for it is not possible to estimate if a failed bit will stay in the failed state or not. Below a small portion from the log file demonstrates the erase error occurred in the double-word address 0x00022150:

```
W/E Cycle 168926, Page start address 00022000, errors 0
ERROR:E dWord address 00022150 read FFFFFFFD expected FFFFFFFF.
W/E Cycle 168927, Page start address 00022000, errors 1
W/E Cycle 168928, Page start address 00022000, errors 1
ERROR:E dWord address 00022150 read FFFFFFFF expected FFFFFFFD.
W/E Cycle 168929, Page start address 00022000, errors 2
ERROR:E dWord address 00022150 read FFFFFFFD expected FFFFFFFF.
W/E Cycle 168930, Page start address 00022000, errors 3
```

Here the logs show that the first failure happened on the cycle 168,926 to the least significant byte of the double-word saved in the address 0x00022150. 2 more W/E cycles later it is seen that the bit was erased properly, yet to fail once again on the next cycle. The same behavior is observed on almost every failed bit in all pages. So the conclusion was that to omit the Flash page at the first bit failure while implementing a data storage module for the ASIC2 Flash memory even if there is an error correction algorithm.

Later on, data processing on Python showed that each program/erase cycle introduces exclusively a single-bit failure at a time in a given byte. Hence, a byte never experiences more than one-bit failure in

a W/E cycle, even though a byte can have multiple bit failures at the end under an extreme frequent stress.

```
ERROR:E dWord address 000220f0 read DF4FBFFF expected DF6FBFFF.
ERROR:E dWord address 000220f0 read DF6FBFFF expected DF4FBFFF.
ERROR:E dWord address 000220f0 read DF4FBFFF required DF6FBFFF.
ERROR:E dWord address 000220f0 read DD6FBFFF expected DF4FBFFF.
ERROR:E dWord address 000220f0 read DF6FBFFF expected DD6FBFFF.
ERROR:E dWord address 000220f0 read DF4FBFFF expected DF6FBFFF.
```

Here above, the log file demonstrates a portion of the error message, which were logged through consecutive W/E cycles, for the double-word with the address 0x000220f0. On the first line, the read data is 0xDF4FBFFF while the previous value of it was 0xDF6FBFFF. The 3rd byte goes from 0b00101111 to 0b01101111, which means that out of failed 3 bits one of them managed to get erased as it was intended while the other 2 bits were still stuck on logical 0. The same bit (7th bit in the byte) kept toggling. On the 4th line, there were 2-bit failures happened in the double-word but in separate bytes of the 3rd and the 4th. The same behaviour was observed in all 3 pages. Hence, it was affirmed that only single-bit failure appears in a certain byte at a given W/E cycle.

Another observation was that *write error* never occurred during $\sim$1M W/E cycles. Programming of logical 0 state to the floating gate was always successful. Since error detection is done by read operation, it can be concluded that in order to read operation to sense a wrong value on the floating gate of a cell requires more W/E cycles for ASIC2 Flash memory. Since the particular failure is related to the *window closure* mechanism which was explained in detail earlier. This particular finding was also useful while implementing the data storage module. For example, the page header used for failed pages are selected as all 0s because even if the page itself showed a bit-failure, writing logical 0 on a cell is not likely to fail, details will be presented in the chapter 4 - *Proposed Improvements for Endurance.*

The initial test was run on randomly selected pages in the ASIC2 which was used for the development of the test code as well as the development of the data storage module. That alone means that the pages under test were already stressed before. So the initial test on evalboard is not reliable enough to analyze the endurance cycle failures and it does not provide healthy test data to be able to run statistical analysis. Yet, valuable information is gathered

during the extreme stress test on evalboard. Both the fact that each program/erase cycle only introduces a single-bit failure and bit failures being unstable are acknowledged while designing the data storage module.

The results of the initial was not provided neither in the appendices or as a separate file since the data file is over 3GB.

3.2 Endurance Test on Testbed

The log file format follows the order of; ASIC2 Modbus address, page on test index, failed double-word address, failed double-word data, timestamp as explained earlier.

```
9,64,0x2a0d0,0xbfffffff,0x86ad8,2020-02-11 14:11:28
9,65,0x2a38c,0xbfffffff,0x87e7b,2020-02-11 14:11:28
9,66,0x2a484,0xffefffff,0x73a68,2020-02-11 14:11:29
```

The logged data from each ASIC2 on the test-bed are saved in separate text files during the test run time. Later on, all the text files are parsed and sorted.

The location of the bit-failure was not analyzed in the scope of this study, due to the fact that the location does not inherently affect the analysis of the endurance cycle since the entire page is assumed to be "failed" independent from the location of the bit failure. The analysis of the endurance is exclusively concerning the number of how many pages have stressed in the specific ASIC2, and the number of how many cycles they have handled before a bit failure occurs.

A couple of ASIC2s, 4 pieces, was not responding after an initial run of the test algorithm, later on due to that issue the firmware of the ASIC2 is switched from dual-core application to single-core application in order to avoid the any possible problems that may occur due to the communication between cores. 26 pieces are left from the 30 ASIC2 piece test-bed and in total 60 pages with the same physical address in each chip were stressed successfully, which leaves us with 1560 pages to be statistically analyzed. In order to have an independent outcome to quantify the reliability, Flash pages which cycled less than 20,000 cycles are considered to be "*failed*", and those which cycled equal to or more than 20,000 cycles are considered to be "*succeeded*". Figure 3.1 and Figure

3.2 demonstrate the distribution histogram of *failed* and *succeeded* pages in the 12th ASIC2 and 17th ASIC2 respectively, where blue sections represent the number of pages which cycled more than or equal to 20,000 cycles and the red sections represent the number of pages which cycled less than 20,000 cycles. Table 3.1 presents the number of failed and succeeded pages in each ASIC2 and their failure over success percentage probabilities.

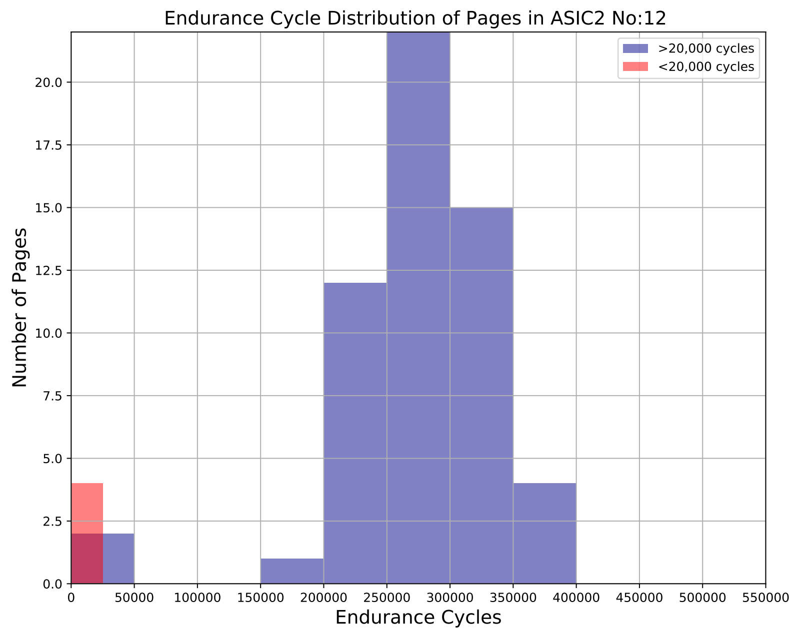

Figure 3.1: Endurance cycle distribution in ASIC2 no12

The cells in Flash memory page do affect each other during programming and erasing as it is explained in the chapter 1 - *Theoretical Background* previously. However, the pages forming the Flash memory in a particular ASIC2 are assumed to be totally independent of each other for the sake of simplicity in the statistical analysis. Each ASIC2 is analyzed individually and later on the confidence interval is calculated from combining the individual failure probabilities calculate from each ASIC2.

Failed to success percentages presented in Table 3.1 are later used to calculate a confidence interval for the population mean, based on a simple random sample (SRS) [EM97]. The standard error, s is calculated to estimate the standard deviation of the population.

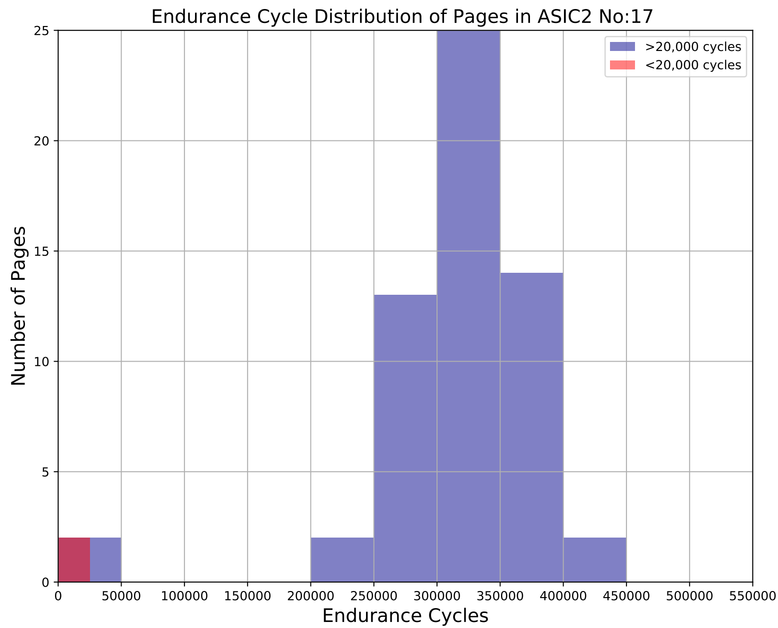

Figure 3.2: Endurance cycle distribution in ASIC2 no17

The sample mean $\overline{X}$ follows the *t distribution*, which is described by degrees of freedom. A sample size of n will have an *(n-1)* degree of freedom [EM97].

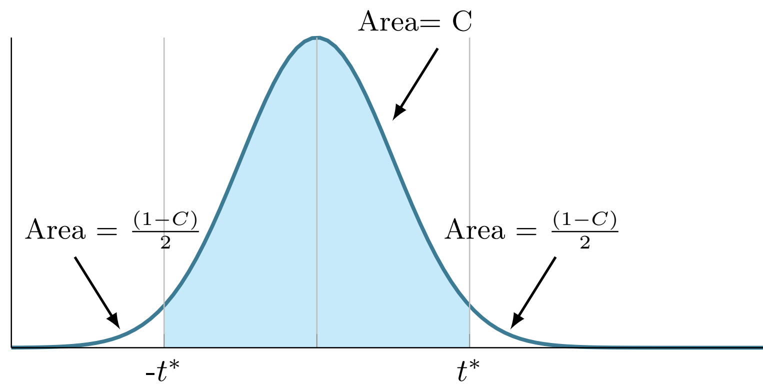

Figure 3.3: The standard normal distribution where t^* is the upper (1-C)/2 critical value [EM97; Sta97].

In the end, the confidence interval for the population mean, based on SRS, is calculated as:

$$\overline{X} \pm t^{*} \frac{s}{\sqrt{n}}$$

Where, $\overline{X}$ is the population mean, n is the population size, s is the estimated standard deviation, and t^{*} is the upper portion of (1-C)/2 in the Gaussian distribution in the Figure 3.3, where C is the confidence level, critical value for the t distribution with n-1 degrees of freedom,*t(n-1)*. The value of t(n-1) is obtained from the t distribution table [EM97]. From the values presented in Table 3.1, the population mean turns out to be 1,797, the estimated standard deviation is 1,914, and lastly the t value of 80% confidence level for 26-1=25 degrees of freedom is obtained from the t distribution table as 1,316 [EM97; Sta97].

In the end, the confidence interval of failure to success percentage with confidence level of 80% is calculated as:

$$1,797 \pm 0,494$$

$$1,303\% \leq failure/success[\%] \leq 2,291\%$$

In the end, it can be concluded that, with an 80% confidence level, the probability of the pages in ASIC2 Flash memory cycling under 20,000 cycles is between ∼1.3% and ∼2.3%.

There are couple of pages in randomly distributed ASIC2s behaving strangely, where bit fails occurs on really low program/erase cycles. In order to make sure those data are, in fact, due to the firmware, where the test algorithm is run and not infant mortality of the Flash memory another test with fewer pieces of ASIC2s is run. The next section will cover how the validation test is run and the results.

The sorted test data from the testbed is attached as a separate Excel file due to its size.

ASIC2 No.	**Failed (F)**	**Succeeded (S)**	**F/S [%]**
1	0	60	0
2	1	59	1,695
3	2	58	3,448
4	1	59	1,695
5	0	60	0
6	1	59	1,695
7	0	60	0
8	1	59	1,695
9	0	60	0
10	4	56	7,143
11	0	60	0
12	0	60	0
13	2	58	3,448
14	0	60	0
15	2	58	3,448
16	1	59	1,695
17	3	57	5,263
18	2	58	3,448
19	2	58	3,448
20	0	60	0
21	2	58	3,448
22	0	60	0
23	2	58	3,448
24	0	60	0
26	0	60	0
26	1	59	1,695

Table 3.1: The number of *failed* and *succeeded* pages with the probability of failure over success calculated individually for each ASIC2

3.3 Endurance Test Validation

Figure 3.4 demonstrates the lower portion taken from the histogram of the ASIC2 with the Modbus address 17, which is presented previously in Figure 3.1. Here in Figure 3.4, it is visible that even though there are pages, which have cycled less than the datasheet endurance cycle, still their endurance cycles are fairly close to the guaranteed cycle of 20,000. To be exact, the 4 pages, represented in red histogram bar, have cycled until 18214, 18695, 19367, 19501. If the locations of these pages are concerned, the page indices for the given failed endurance cycles are 102, 103, 96, ad 81 respectively. It is highly possible that one of the Flash pages have experienced infant mortality and affected the rest due to disturbance while programming and erasing. Since the physical organization of the memory array is unknown it is hard to explain the phenomenon.

However, it is evident that there is a strange behavior when the endurance distribution histogram of the 17th ASIC2 is analyzed for example. Figure 3.5 demonstrates the lower portion of the histogram, which was presented before in Figure 3.2. Figure 3.5 shows that there exists a page in the Flash memory, which cycled through less than 5,000 program/erase cycles. To be exact, the endurance cycle of the particular page is 1,731. 1,731 cycles of program/erase is drastically far from the datasheet guaranteed value, which raised the question if the firmware where the test algorithm is run has an influence in this unusual behavior. In order to validate that this is not an infant mortality issue but an external disturbance, another set of ASIC2s are tested on a modified testbed presented in the section 2.3 - *Endurance Test Hardware*.

For the validation test, ASIC2s are put into the bootloader mode, and the programming and erasing operations of a particular Flash page is done through the dynamic-link library (DLL) functions via a Python script. In this case, the firmware which was used previously to test the ASICs is bypassed and the programming and erasing are done at a lower level. DLL functions can directly address and operate on Flash memory without the firmware when the device is in bootloader mode. The validation test was run up to 3,000 cycles to catch early failures. 100 pages have been tested in each ASIC2 and no page has experienced a bit failure up to 3,000 cycles. Hence out of 1,200 pages there was no early failure.

In the end, the validation test has proven that the strange behavior in a couple of pages spread around the testbed in a random manner was influenced by external disturbances rather than being early-stage failures of the Flash memory itself. Taking the validation test into account, the pages that have cycled through drastically less than the datasheet value can be removed from the failure percentage calculation in the Table 3.1, to create the Table 3.2.

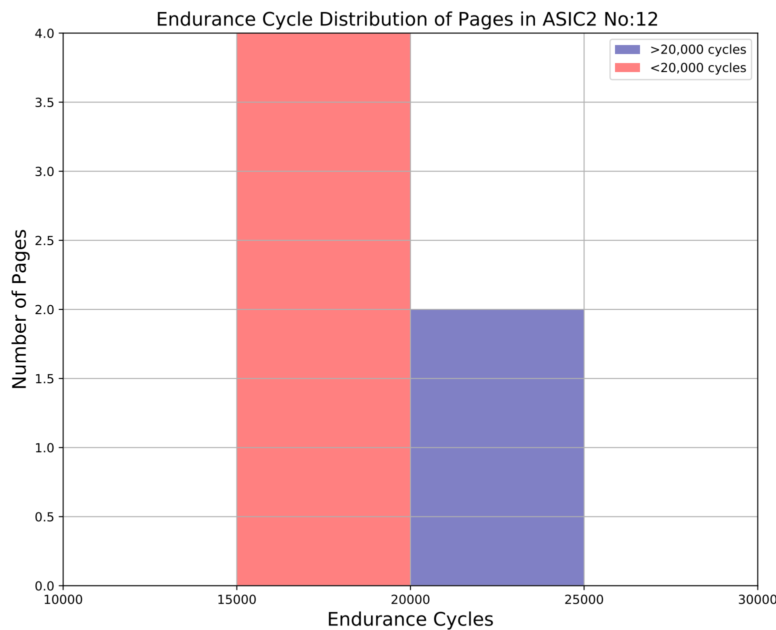

Figure 3.4: Endurance cycle distribution in ASIC2 no12 cropped up to the cycle number 30,000

Once again, the confidence interval is calculated for the confidence level of 80 %. Removing the pages with the odd behavior has changed the estimated standard deviation from 1,914 % to 1,877 % and the mean of the population has decreased from 1,797 % to 1,336 %. The t^* value is still the same since it does only concern the desired confidence level and the degree of freedom, where both did not get affected by the removal of the odd pages, hence the t^* value is 1,316.

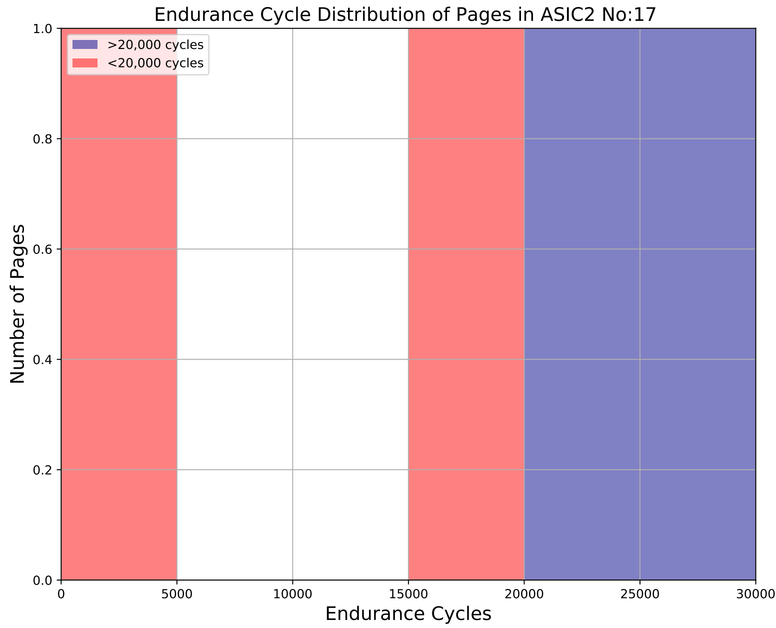

Figure 3.5: Endurance cycle distribution in ASIC2 no17 cropped up to the cycle number 30,000

$$\overline{X} \pm t^{*} \ \frac{s}{\sqrt{n}}$$

$$1,336 \pm 0,484$$

$$0,851\% \leq failure/success[\%] \leq 1,820\%$$

The result validation test using DLL functions has improved the endurance test results presented in the prior section, and with 80% confidence level, the probability of the failure rate over the success rate, assuming *failure state* represents Flash memory pages cycling under 20,000, is less than ∼2 %.

The sorted test data with the eliminated pages are also attached as a separate Excel file due to its size.

ASIC2 No.	Failed (F)	Succeeded (S)	F/S [%]
1	0	60	0
2	0	59	0
3	2	58	3,448
4	0	59	0
5	0	60	0
6	1	59	1,695
7	0	60	0
8	1	59	1,695
9	0	60	0
10	4	56	7,143
11	0	60	0
12	0	60	0
13	2	58	3,448
14	0	60	0
15	1	58	1,724
16	0	59	0
17	2	57	3,509
18	1	58	1,724
19	2	58	3,448
20	0	60	0
21	2	58	3,448
22	0	60	0
23	2	58	3,448
24	0	60	0
26	0	60	0
26	0	59	0

Table 3.2: The number of *failed* and *succeeded* pages with the probability of failure over success calculated individually for each ASIC2 where the odd behaving pages are disregarded

Chapter 4

Proposed Improvements for Endurance: How to Utilize Flash Memory for Data Storage

Two types of improvements are proposed to virtually increase the program/erase endurance for the application developer. The first two modules are designed for a specific application on ASIC2, and later on, a generic data management module is proposed to be integrated into any kind of application that utilizes ASIC2. The aim is to increase the number of write cycles, which becomes the new number of endurance cycles rather than the number of times the page gets fully programmed and erased.

4.1 Application-Specific Modules: Maximum and Counter Algorithms

A specific application, which runs on ASIC2, is required to utilize the internal Flash memory to store and mainly to update a maximum value and counter values. Two separate modules were proposed before deciding on a generic data management method for ASIC2 Flash memory. The maximum value and the counter values had to be persistent and easily updatable. So it was more feasible to come up with a specific solution to store those values in Flash memory instead of a generic data management. In the end, the counter algorithm formed the base of the write endurance counter used for the final data storage module that will be explained in section 4.2 - *EEPROM Emulation on ASIC2 Flash Memory*.

Maximum Value Storage Algorithm

A specific application running on ASIC2 required to save and update a maximum value during run time. The storing and updating of the value must avoid constant erasing of Flash memory since Flash memory has a limited program/erase endurance cycle as mentioned in chapter 1 - *Theoretical Background* [Zha+17]. On top of that, the maximum value saved in the Flash must be robust against any unexpected corruption or a disturbance in the memory cell that may happen over time.

Considering the limitations mentioned, the approach taken in order to store the maximum value was based on taking advantage of the Flash memory properties. Default state of Flash memory is its erased state, essentially when the floating gate is positively charged, being in logical "1" state, as it was explained before in chapter 1 section 1.2,. The strategy to save the maximum value was to program as many individual bits in the Flash page to be in logical "0" state that represents the value of the maximum. Since a Flash memory page in ASIC2 internal Flash memory consists of 512 bytes, a page on its own is able to store a maximum value up to 4096 (512 bytes = 4096 bits), which is more than enough for the application that needs the maximum storage solution. Thus, only one Flash memory page was enough to save the needed maximum value for the application.

Figure 4.1 presents a portion of Flash memory page, where the logical "1", erased state, bits are represented in greyscale and logical "0", programmed state, bits are represented in red color. Here in Figure 4.1 there are 3×5 bytes as in row × column configuration. Given addresses from 0x22000 to 0x22004 are the physical byte addresses. In the example we see 10 bits marked in red thus the saved maximum value is 10 at that moment.

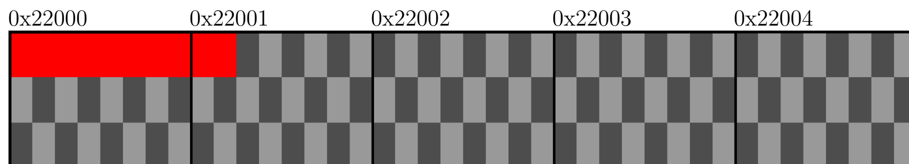

Figure 4.1: Visual representation of maximum value 10 saved in Flash page where red bits are logical "0", greyscale bits are logical "1".

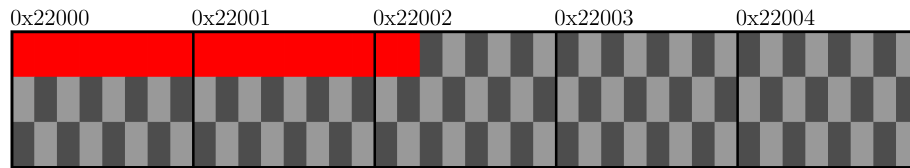

Figure 4.2: Visual representation of maximum value 18 saved in Flash page where red bits are logical "0", greyscale bits are logical "1".

Updating the maximum value here means to appending another programmed bit in the Flash memory page, as it is represented in Figure 4.1 and Figure 4.2 where the maximum value was 10, and then updated to 18. If the incoming new maximum is higher than the stored one, the number of "0" that represents the maximum value is increased in the same amount of the difference between the old maximum and the new maximum. In case the incoming new maximum value was less than the stored one, nothing happens as expected.

Reading the maximum memory from the application sides is achieved by counting how many "0" state bits saved in the particular Flash memory page.

One big advantage of the maximum storage module is the robustness it provides against saving the maximum value as a numerical value instead of the number logical "0" bits representing the value. In case of a bit failure in the Flash memory page allocated saving the maximum value, the maximum storage algorithm will only deviate by 1.

Assuming the maximum value saved from the application is 240 in decimal, 0b11110000 in binary. If there is a bit corruption on the most significant bit, the value will end up as 0b**0**1110000 in binary which will translate to 112 in decimal. That corruption will yield a percentage error of $\sim$53%. On the other hand, in case of a bit corruption, with the maximum storage algorithm, the number of the maximum value will only deviate by $\pm$ 1, which is not as drastic as the first case.

User functions for maximum storage module are:

- dataflash_maximum_get() : Read the maximum number stored in Flash

- dataflash_maximum_set(): Store a new maximum value in Flash

- dataflash_maximum_clear() : Clear the Flash page allocated for maximum value storage

Counter Storage Algorithm

As it was mentioned, a specific application running on ASIC2 also required a storage system to be able to save persistent and updatable counter data. Again, it was important to avoid constantly erasing the Flash page due to the lifetime limitation of Flash memory [Zha+17], also to be as robust as possible to any unexpected corruption in Flash page.

The approach taken to store the counter value was to have two separate Flash pages, one of the pages (Page A) is used to increment every time a counter increment is called from the application side, while the second page is used as a multiplier (Page B), which gets incremented when the first page is full. Then the first page gets erased and the same procedure repeats again.

According to the application needs, only the first double-word (32 bits) of the Page B (second page) is used as a multiplier, and the entire page of the Page A (first page) is used as a counter. So that the counter value is obtained as:

$$Count = Page\ A + (Page\ B \times Flash\ Page\ Size)$$

Where the maximum value of the counter would be:

$$Count = 4094 + (32 \times 4096) = 135168$$

When the counter reaches the maximum value, it will not start over but instead will stay at the maximum value till it is reseted manually from the application side. It is an important detail for the user to keep in mind.

To visually demonstrate how the Flash pages look like while counting, as an example, the counter value can be assumed as 28700, which leads to:

$$\begin{aligned} Count &= Page\ A + (Page\ B \times 4096) \\ 32800 &= 28 + (7 \times 4096) \end{aligned}$$

Here in Figure 4.3 and Figure 4.4, Page A and Page B are represented but the middle portions of the Flash pages are cut for simplicity. According to the example value, Page A contains 28 bits in logical "0" state, while the multiplier, Page B, contains 7 bits in logical "0" state. The figures represent the logical "1" bits in greyscale, while logical "0" bits in red.

Figure 4.3: Page A = 28

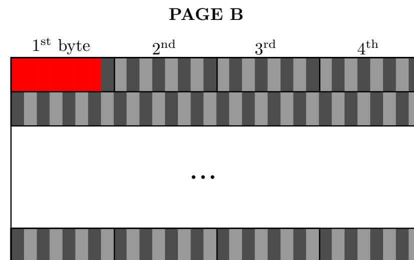

Figure 4.4: Page B = 7

User functions for counter storage module are:

- dataflash_counter_get() : Read the counter value saved in Flash
- dataflash_counter_inc(): Increment the counter
- dataflash_counter_clear() : Clear the Flash page allocated for the counters

Later on, a modified version of the counter algorithm is used for the generic data storage algorithm, dataflash module, which will be explained in section 4.2 - *EEPROM Emulation on ASIC2 Flash*.

4.2 EEPROM Emulation on ASIC2 Flash: Dataflash Module

The basic idea of using Flash memory to store application variables, with no storage algorithm implemented, is that writing the value on a Flash page and when the update is required, erasing the Flash page and writing the new value of the variable. Due to the limited lifetime of Flash memory, this approach is not feasible as data management.

The most basic approach to solving this issue would be to fill the Flash pages with the new incoming values and read the variable with the highest physical address. In this case, the endurance cycle

would be improved a considerable amount yet this approach would support only one variable element since there is no addressing of multiple application variables. Based on this idea, there are a couple of already existing modules, which include a virtual addressing section to be able to address multiple application variables, on top of that some have error detection methods such as cyclic redundancy check (CRC) [STM18; Che+04; Lab10; Tec19].

The proposed optimization module is based on the fundamentals of available algorithms in the market, which is to save in increasing orders of physical address in a Flash page [STM18; Che+04]. The approach taken is to have the basic idea of storage from the existing algorithms and to modify the features according to the needs of ASIC2 applications. Furthermore, a more sophisticated error detection and correction algorithm is implemented compared to a simple CRC. Overall, the goal of the proposed optimization algorithm is to improve both the endurance cycle and the robustness in terms of the integrity of the stored data.

As mentioned in *Problem statement*, some applications using ASIC2 require a constant and persistent data storage preferably without the additional cost of external EEPROM. Internal NOR type Flash memory of ASIC2 is used to store and execute the firmware but still there is a considerable amount of free space left in internal Flash, it was decided to optimize that section to be used as an internal data management which emulates an external EEPROM. This section will describe *the Dataflash Module* which is developed to substitute a standalone EEPROM by emulating the EEPROM mechanisms using ASIC2 on-chip Flash memory. Emulation is achieved by utilizing a number of pages, depending on the application requirements, in the Flash memory for data management.

Dataflash module has the following features:

- Simple functions: initialize, write, read, and diagnostics.
- Additional functions: reset the endurance counter and format a single page to save from erase time.
- Hamming error detection and correction is used to ensure robustness.
- Simple and scalable model according to the desired endurance cycle and the number of application variables.

Dataflash module has the following limitations:

- At least six Flash memory pages (3072 bytes) need to be allocated to be used as internal data management.
- Write function may occasionally take from 20ms to 40ms (depending on how the erase time is configured in application). This limitation must be considered in time critical applications.
- The emulated EEPROM size is flexible and only limited by the Flash memory size allocated to that purpose by the user.

Working Principle

The data storage approach is based on having two sets of Flash memory pages allocated. Sets can be made of one or several pages depending on the endurance cycle and number of application variables, which will be explained in detail in later sections. In order to ensure a certain amount of endurance cycle with high confidence level and to avoid garbage-collection during run time as much as possible, the minimum number of pages is decided as 2 pages per set (in total 4 pages for storage algorithm + 2 pages for the endurance counter comes to minimum 6 pages in total). An Excel sheet calculator, explained in Section D.2, provided with the module lets the end-user know, and gives the proper error while calculating the space to be allocated.

The first set of pages is initially erased and first set is used to store the new data and writing operation happens sequentially in increasing order of Flash memory addresses. That ensures the data with the highest physical Flash address is the most current one. Once the first set is full, with the next write call from application, garbage collection is done from the first set to the second set.

The second set collects only the latest/valid data from the first set. After garbage collection, the remaining area in second set is used to write the new coming data. When the page(s) in second set is full, garbage collection is done from second set to first set. Algorithm continues to toggle the data between two sets as described.

For each page, a header field that occupies the first 64-bits indicates the current status of the page. Each page can have six possible states:

- **ERASED**: the page is empty.
- **ACTIVE**: the page is currently being used to store new data.
- **VALID**: the page is full. This state does not change until all valid data has been garbage collected to the receiving page.
- **ERASING**: the valid data in the page is being garbage collected or has already been garbage collected. This page is ready to be erased when it is needed.
- **RECEIVE**: the page is used during garbage collection to receive the valid data from previous set. This state will either become active or valid depending on how much data previous set holds.
- **INVALID**: a corruption has been detected in this page. This page will be exempted from the algorithm.

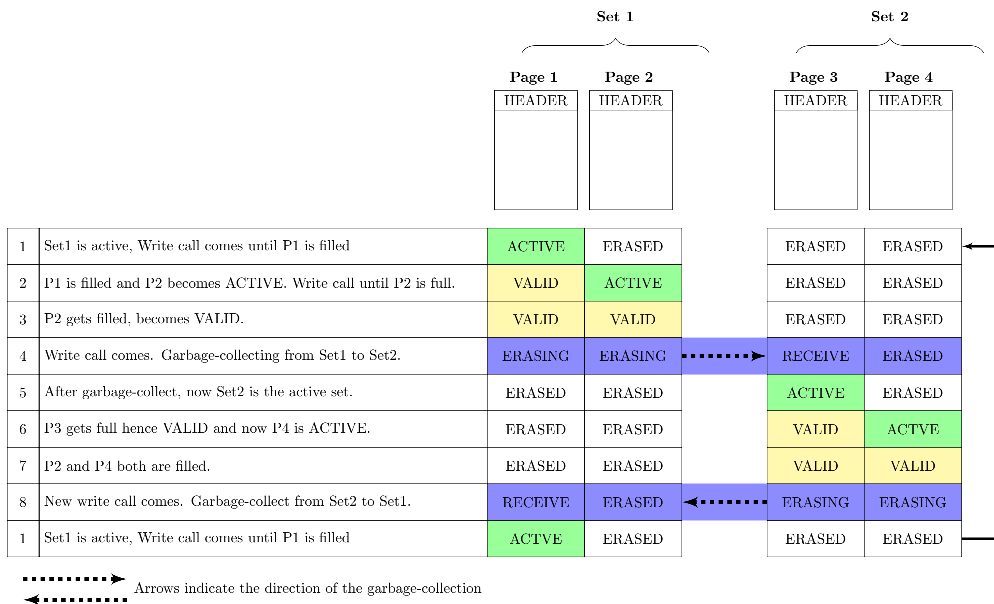

Figure 4.5: Header transition of pages [Simplified]

Figure 4.5 shows the status evolution of set headers (assuming no corruption happened) in the case where each set is made of 2 pages. Figure demonstrates a very simplified version (Garbage collecting is represented as one step here for simplicity) to give the idea of how header statuses change, detailed transitions are explained in *Writing Data*.

On top of pages allocated for the sets described above, two extra pages are allocated for a counter algorithm to keep track of the endurance cycle for diagnostics purposes. The counter algorithm used in dataflash module is a modified version of the counter algorithm which was introduced in previous section 4.1 - *Counter Storage Algorithm*. Here in the dataflash module, the multiplier page, Page B, is extended to use the entire page, 512 Bytes, instead of only the first 32-bits section as in the original algorithm. So the counter is able to count up to:

$$\begin{aligned} Count &= Page\ A + (Page\ B \times Flash\ Page\ Size) \\ &= 4094 + (4096 \times 4096) \\ &= 16,781,312\ cycles \end{aligned}$$

Counter algorithm that is used to keep track of write endurance cycle is limited to the maximum number ∼16.7 million cycles as calculated above. For applications that require higher *write cycle* (refers to the number of times write function is called instead of program/erase cycle of a page) should be aware of that and make sure to reset the counter before since after the counting algorithm reaches the maximum it will stay as it is.

The initial study done by Eaton to use the internal Flash memory for a data storage algorithm presents a couple of application that may require a such module for data management [BB17]. Among those presented applications, the highest write endurance (used interchangeably with *write cycle*) needed was ∼1M cycles [BB17], hence the modified counter algorithm is more than enough to support all applications given as an example in the study. That's why the counter algorithm was not further modified to support more than ∼1.6M cycles.

Page Header Transitions

Figure 4.6 demonstrates how header statuses change with the called action from application side as well as the internal operations, i.e. garbage-collection operation or finding a corruption.

Flash memory starts as all erased (all 1s) in the first programming, thus the default header of a page is ERASED header. When the write function is called from application, the appropriate page becomes ACTIVE. After enough write function calls (63 to be exact

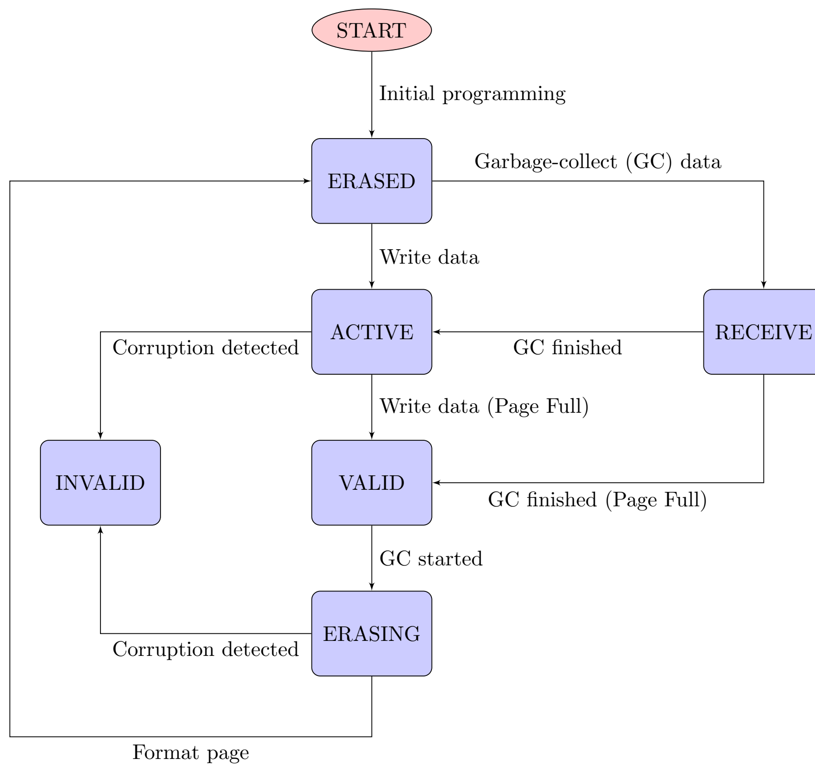

Figure 4.6: Header transitions

to fill a page considering the garbage-collection has not happened previously) the page gets full and with the next write call from application (64th call), the current page becomes VALID, and the next page in set becomes the new ACTIVE.

After all the pages in the set becomes VALID, garbage collection starts with the incoming write function call. VALID pages that needs to be garbage-collected becomes ERASING, which header indicates these pages are going in garbage collection process, and the page that is receiving the collected data is indicated with RECEIVE header. In case where there is more than a page worth of valid variable elements are collected RECEIVE page may become VALID, if no RECEIVE page will become ACTIVE, and the same process will keep toggling between two sets.

If a corruption is found during garbage collecting, page becomes INVALID and will never be used again, and is omitted from the algorithm all together.

ERASING header indicates both that the garbage-collecting has started and ended. There is no distinction between those two states in terms of header. In case a power-loss happens during run time while the previous set was being garbage-collected, when the system is powered, initialization function is responsible to handle completing the garbage-collection.

In case a corruption is found in the ACTIVE page during a get function call, the next write function call will take care of it in that instant and make the particular page's header INVALID. That only happens to ACTIVE page, other pages with VALID or ERASING headers will always get handled by garbage collecting since it is not an urgent problem as in ACTIVE page case.

Once again, if a power-loss situation occurs when there is a corrupted data caught by the get function in the ACTIVE page and there was no time for write function to handle the corrupted page, initialization function is supposed to handle the situation as it is in interruption during the garbage-collection situation. That issue is recognized recently and it will be handled in the next releases of the Dataflash module.

Page and Variable Element Format

Header Format

For ASIC2 internal Flash memory, each page is comprised of 512 bytes (245 words), where a word is made of 16 bits. The minimal write width in ASIC2 Flash memory was designed as 32-bits (also called *double-word*, where 16 bits form a *word*).

Flash memory by principle allows us to program only 0 to the already programmed space without erasing due to the physical structure of the floating gate MOS transistor as explained in chapter 1. It is not possible to switch from 0 to 1 without erase operation and erase operation is allowed either as mass erase of entire Flash memory or page-wise, again, as a result of the physical structure of the Flash memory.

The header for the algorithm occupies the first 64 bits (4 words), and each variable element is made of 64 bits (4 words) as well. Thus, a single Flash page can store up to 63 variable elements as calculated below:

$$\frac{256\ words}{4\ words} - \ 1\ (Every\ page\ has\ a\ header) = 63\ Variable\ Elements$$

During initial start-up every page will be on default state which is erased state so that the initially all Flash memory pages will be all 1s. Being able to program individual 0 bits allow us to change the status of the headers without an erase operation. Hence the headers are decided by taking this advantage into consideration as shown in Table 4.1.

The header state transitions are demonstrated in detail in the *Page Header Transitions*.

Header	1st word	2nd word	3rd word	4th word
ERASED	0xFFFF	0xFFFF	0xFFFF	0xFFFF
RECEIVE	0xAAAA	0xFFFF	0xFFFF	0xFFFF
ACTIVE	0xAAAA	0xAAAA	0xFFFF	0xFFFF
VALID	0xAAAA	0xAAAA	0xAAAA	0xFFFF
ERASING	0xAAAA	0xAAAA	0xAAAA	0xAAAA
INVALID	0x0000	0x0000	0x0000	0x0000

Table 4.1: Page headers used in Dataflash module [HEX]

Variable Element Format

Each variable element is defined by a virtual address and a data value to be store in Flash memory for subsequent retrieval and update.

The virtual address, *VA*, is 8-bit long and the data value is 32-bit long, which means that any data up-to 32-bit can be stored. It would be recommended to combine smaller size application parameters into a single 32-bit parameter to reduce the number of variable elements. For example; to have a single 32-bit variable element instead of four 8-bit elements can save from Flash memory space and/or help achieving higher endurance with the same size of memory space.

Both virtual address and data contains error detection and correction redundancy portion to ensure their integrity. The error de-

tection and correction algorithm used in Dataflash module, Hamming[12,8], allows the module to detect and correct 1 bit error in a byte, for more detailed explanation refer to Error Detection and Correction Code: Hamming[12,8], thus the data portion has a 16-bit long redundancy while the virtual address portion has a 4-bit of redundancy to ensure the integrity and as shown in Table 4.2. There is a 4-bit section left unused for now, it can be utilized in future releases, for example to have an extra CRC check or maybe to extend the already integrated Hamming algorithm to detect 2-bit error.

VA	**VA Parity**	**N/A**	**Data Parity**	**Data**
8 bits [LSB]	4 bits	4 bits	16 bits	32 bits [MSB]

Table 4.2: Variable element structure

8-bit portion allocated for virtual addressing is the limiting factor for why it is only possible to use 256 different variable elements. It is important for the end-user to keep in mind if there are more parameters that needs to be saved. In that case, the parameters need to be utilized by the application programmer to fit into the limited number of parameters the module allows. Figure 4.7 shows how header and variable elements are physically stored in a Flash memory page.

Each 4-bit redundancy can achieve 1-bit error detection and correction in 8-bit via Hamming[12,8]. So that implemented Hamming[12,8] is able to detect 4-bit error in 32-bit data and a 1-bit error in 8-bit virtual address. In case of more than 1-bit error happening, Hamming ECC may recognize the error depending on the Hamming distance of the error but the error cannot be recovered and returned value will not be correct. In this case the get() function returns the appropriate status to let the user know that a corruption was detected and attempted to be corrected.

Next internal working procedures of write, and get functions as well as the additional features in Dataflash Module will be explained in detail. An example usage case is also presented in Appendix C by visualizing the Flash memory pages to demonstrate the data transitions.

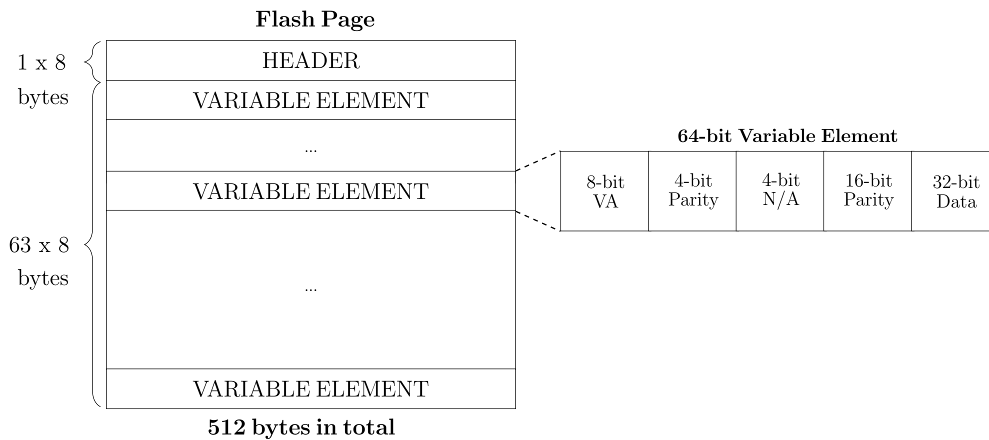

Figure 4.7: Variable element in a Flash page

Writing Data

When the write function is called, first of all, the algorithm gets checked if there is still enough space left in both sets to support the desired number of application variable elements. On the assumption that there is not sufficient space left in both sets, means that physical Flash memory pages left uncorrupted are not enough to handle the given number of application elements, the write function will return the error *DATAFLASH_WRITE_OPERATION_FAIL*. This is an unrecoverable error, it is the point where the algorithm goes expired.

If the algorithm still has an adequate amount of physical space to support the application variables, the boolean array, which keeps the corruption status of the allocated Flash memory space in the beginning, is read. If the active page hasn't corrupted yet then its header goes through Hamming check to make sure the entire Flash memory page is corruption-free of any bit failures.

Later on, the active page page is read through to see if there is still an empty slot left to save the incoming data. If the active page still has a slot, then the incoming data and its corresponding virtual address go through Hamming redundancy calculation. The variable element structure is formed with the data, virtual address, and calculated redundancies as explained previously. Next, the created variable element is stored in the appropriate free slot in the active page.

If the current active page is already filled, but there is still at least one more page left in the active set, the new page converts to the ACTIVE header while the previously ACTIVE page gets the header VALID. The saving procedure as mentioned before is applied to the new active page.

In such a case that the currently active set does not have any more pages available, the garbage-collecting procedure starts between sets. During garbage-collection, only the valid data associated with each virtual address is collected from one set to another. After garbage-collection is done, the same saving procedure is applied. Garbage-collection procedure is explained in detail in *Garbage-Collection*.

An important decision point is, which is marked in the Figure, when corruption is found on the currently active page. The data flow describing this particular case is redacted from Figure B.1, and visualized in Figure B.2 for simplicity. Assuming a corruption has caught, the active page is marked as INVALID. The active set is checked whether there is any other page available or not. If so, the new page becomes ACTIVE. If not, the broken page, as well as the rest of the set, is garbage-collected. At the end, the aforementioned saving procedure is applied and the write function returns *DATAFLASH_NO_ERROR* status.

Important on user side is that write function returns one of two statuses as above-mentioned:

- DATAFLASH_NO_ERROR : means write operation is done.
- DATAFLASH_WRITE_OPERATION_FAIL : means that the usable memory space left in Flash memory is not enough to support the required number of variable elements. It basically means that it is the point where the algorithm expires. No write operation will happen after this error is seen, it is an unrecoverable error.

More details about the inputs and return types of the user functions are described in Appendix D.5.

Reading Data

Get command reads through pages from the highest physical address to lowest to get the latest data associated with the given

virtual address. Figure B.3 shows the data flow of how the data is read from Flash memory with the given virtual address.

Data is considered as "healthy" if the Hamming redundancy checks out and *get function* returns the appropriate status to indicate no corruption is found. If Hamming algorithm has detected a corruption and error correction is performed, returned status also indicates the situation to let the end user know that a corruption is detected and corrected. If nothing turns out with the requested virtual address, it can be a case of either the requested virtual address was never written by the user, or it got corrupted to the point of no recovery. Get function also indicates the situation with the appropriate status.

Due to limitation of Hamming(12,8) algorithm only one bit error detection and correction is possible for each byte. That means in case of more than one bit failure in a byte, Hamming ECC algorithm will try to fix it as if it is a single bit error. So it becomes important to let the user know that if the data has been through error correction or not.

Read function return statuses can be summarized as:

- DATAFLASH_NO_ERROR : No error found either in the data or the virtual address saved in Flash.
- DATAFLASH_RECOVERED : An error/corruption is detected and corrected.
- DATAFLASH_NOT_FOUND_IN_FILESYSTEM : The requested virtual address doesn't exist in the Flash memory space. It can be that virtual address was never written in, or had an unrecoverable corruption.

More details about the inputs and return types of the user functions are described in Appendix D.5.

Garbage-collection

Garbage-collection is triggered when all the allocated Flash memory pages in a set are filled and there is a new write call from the application side.

First, the last page with the highest physical address' header changes from VALID to ERASING, and the available page in the other set's

header becomes RECEIVE. Only the valid data is collected from the ERASING page to the RECEIVE page. During the collection process, each variable element goes through Hamming check, and in case corruption is found that specific page is tagged as INVALID. Next, each VALID page goes through the same process: header changes to ERASING and scanned through out the page for valid data and corruption.

		P1	P2	P3	P4	P5	P6
1	P3 is full but hasn't recognized yet.	VALID	VALID	ACTIVE	ERASED	ERASED	ERASED
2	New write call realizes P3 is full.	VALID	VALID	VALID	ERASED	ERASED	ERASED
3	GC starts on P3	VALID	VALID	ERASING	RECEIVE	ERASED	ERASED
4	P2 is being collected	VALID	ERASING	ERASING	RECEIVE	ERASED	ERASED
5	P2 is done. P1 being collected.	ERASING	ERASING	ERASING	RECEIVE	ERASED	ERASED
6	GC finished.	ERASING	ERASING	ERASING	ACTIVE	ERASED	ERASED

Figure 4.8: Header transition during garbage-collection

The headers of VALID pages change to ERASING header one by one while they are going through the collecting process. The reason is that if a power failure happens during garbage-collecting, initialization function can realize that since there is no special header that indicates whether the garbage-collection ended or not. Figure 4.8 demonstrates an example of how the header transitions happen during the garbage-collection process where each set has three pages allocated.

It is important to note that the reason why pages are left in ERASING state and are not physically erased is that the erase operation in ASIC2 internal Flash memory takes between 20-40ms and the erase function itself is a blocking function, which can be a significant problem in time-critical applications. Assuming the application allocated 100 pages for the data management portion of Dataflash module, which means that each set will have 50 pages and if they were to erased during garbage-collecting, the erase function will block the ASIC2 for 100ms.

Error Detection and Correction Code: Hamming[12,8]

Hamming code was developed in 1950 by Richard W. Hamming, an American mathematician and computer scientist. Richard was irritated by the inability of punch card readers to correct errors, so

he spent several years developing error-correction algorithms. The result was a family of algorithms called Hamming code. To this day, Hamming code is still in use in DRAM, RAID storage, and other mediums that require simple error correction [Ham50; Coo; Rah17].

Below formula is used to calculate the needed redundancy bits (r) for given length of data bit (m) for single-bit error detection and correction using Hamming code [Rah17; Fie04].

$$2^r \geq m + r + 1$$
$$r = number\ of\ redundancy\ bits$$
$$m = data\ bits$$

For the dataflash module, the result of +1M cycle endurance test run on eval-board showed that each byte only had single-bit error. So based on the test results, Hamming[12,8] is selected for dataflash module.

Hamming[12,8] is a type of Hamming code where the message length is 12 bits and out of those 8-bit is the data (m=8) and 4-bit is the redundancy (r=4).

Parities are calculated as [Rah17]:

$P1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7$
$P2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$
$P3 = D2 \oplus D3 \oplus D4 \oplus D8$
$P4 = D5 \oplus D6 \oplus D7 \oplus D8$

Table 4.3 visualizes how each parity bit covers for each data bit, where green cells represent that the data bit is covered by parity bit and red cells represent the data bit is not covered by given parity.

As seen in the table, each bit is covered by at least two parity bits to detect and correct a single bit change.

The Hamming[12,8] implemented in dataflash module is a single-error-detecting and single-error-correcting (SEC) algorithm as mentioned that has an overhead of 4-bit for each byte. That is the

	D1	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**
P1								
P2								
P3								
P4								

Table 4.3: Data bit coverage of each parity bit

reason of the number of the redundancy bit sections shown in the variable element table in Section 4.2.

Since the data can be up-to 32 bits, for each byte of it there is 4-bit redundancy which results with 16-bit redundancy in total. This also allows us to detect and correct 4 bits of error in the entire 32-bit of data. For the 8-bit virtual address, there is simply 4-bit redundancy. When writing a variable element, both its virtual address and the data itself are stored with their corresponding redundancy.

When reading a variable element, the Hamming parity is again calculated in RAM and checked against the stored value in Flash memory to see if any error occurred or not. If the parities match, means that there is no error, the variable element is considered as valid. If parities don't match, means that the variable element has corrupted and it is both indicated in RAM and in Flash memory to be taken care of in the next steps of algorithm, which is the garbage-collection process that handles all corrupted data and pages.

Hamming128 implemented in dataflash returns detection and correction statuses as [Coo]:

- HAMMING_NO_ERROR
- HAMMING_RECOVERED
- HAMMING_UNCORRECTABLE

Later on during garbage-collecting, each variable element is corrected during collection process and saved to the proper page in Flash and the pages that contains at least one corrupted variable element are tagged as INVALID header in flash. The pages with INVALID headers are omitted from the algorithm.This situation is expected to happen after the guaranteed life cycle (LT) is exceeded and the user should be aware of the write cycle as well

as how many running pages left in algorithm through diagnostics function.

Write Endurance Estimation

The equation to estimate the write endurance of an application variable is set up as below.

$$LT = \left[\frac{(Pages\ in\ Set\ \times\ 63) - (EEPROM\ Variables - 1)}{EEPROM\ Variables}\right] \times 2\ \times\ P/E\ Cycle$$

$$Where\ LT\ stands for\ Life - Time\ or\ write\ endurance\ cycle$$

Pages in set represents the number of pages allocated to each set., *EEPROM Variables* is the number of application variables. *P/E Cycle* is the datasheet guaranteed endurance cycle.

The formula is decomposed into the following steps:

1. A Flash page is limited to 63 number of total variable elements that it can store. Multiplying that with the number of Flash pages allocated for a set gives us how many variable elements a set is able to store.

2. Subtracting the number of application variables minus one from the total number of application elements in the set gives the number of write operation times that it takes to fill the set before garbage-collecting. Minus one is there due to the fact that the garbage-collection operation does not collect the data associated with the new incoming virtual address. Thus, it leaves one spot open and gains the algorithm an additional write cycle before the next garbage-collection.

3. Dividing the number of total write cycles obtained in the numerator by the number of application variable elements estimates the number of write times of the entire application variables. Here it is assumed that all variables are constantly written in consecutive order for the sake of simplicity.

4. The Dataflash algorithm consists of 2 sets, hence the multiplication by 2 to take both sets into account.

5. Finally, multiplying by the tested endurance (P/E) cycle gives the final estimated write endurance.

Tested endurance cycle is obtained from the endurance test run on the specially designed testbed as it is presented in detail in section 2.3 - *Endurance Test Hardware*. The calculated endurance cycle from the test is as a matter of fact the expected erase cycle of a particular Flash page, yet we are using it to formulate the write endurance cycle. The reason for that is based on the test algorithm. The endurance cycle for a Flash page is obtained the moment when there is a single bit error occurred. Hence the estimated endurance of a Flash page is driven by the weakest bit in the 512-byte page, hence to come up with a conservative estimation for the write cycle the endurance cycle from the first bit fail is used.

The provided Excel sheet to calculate needed space for the given amount of application variables and the endurance cycle the application requires is based on the life time formula. The life time formula given is rearranged to calculate the number of pages per set.

The Dataflash algorithm does not inherently increase the Flash page life time per se. Assuming there is only one 32-bit application variable and one Flash page is used and but no data management algorithm. This situation will end up resulting with the given endurance cycle as the write endurance of application variable. Now, if there is the simplest data management approach mentioned is implemented, where the highest physical address points to the latest data. For the sake of example, the data-sheet guaranteed endurance cycle, 20,000, can be used for calculation. The simplest algorithm will yield:

$$LT = \frac{512\ bytes}{32\ bits} \times 20,000 = 2,560,000\ cycles$$

Meanwhile the Dataflash with only one page each set will yield:

$$LT = \left[\frac{(1 \times 63) - (1 - 1)}{1}\right] \times 2 \times 20,000 = 2,520,000\ cycles$$

Which is slightly less than the simplest algorithm for the given very specific situation. Yet, the applications require more than one variable as well as the stored data to keep its integrity. A

small portion of endurance cycle was sacrificed for some specific cases, i.e. having only one application variable element, in order to keep the data coherent while improving the write endurance cycle.

In addition to Dataflash module, as previously explained, two more application specific modules are proposed to be able to get the most out of the Flash memory page in term of write endurance.

Conclusion

ASIC2 is the second generation of Application Specific Integrated Circuit specifically designed by Eaton. ASIC2 internal Flash memory, which was designed to store and execute the firmware, was tested by the third-party vendor on chip-level. The program/erase endurance cycle, which is the key issue in Flash memory reliability, is provided in the datasheet. Yet no detail about the test was provided such as the testing methodology, the population or the confidence level percentage the given result falls into. This master thesis contributes to the evaluation of the confidence interval of the program/erase endurance provided by the vendor of ASIC2 and attempts to improve the endurance cycle virtually by implementing a data management module for ASIC2 Flash memory.

The endurance of ASIC2 Flash is tested from the firmware-side, running the test code in the application program, on a testbed that is specially designed for the purpose. Design requirements and details on test algorithms are presented earlier in chapter 2 - *Design and Implementation*. The test algorithm is a simplified combination of the actual chip-level stress testing of floating gate devices and the March tests [Yeh+07] which are mainly utilized to test RAMs.

At last, in the chapter 4 - *Proposed Improvements for Endurance*, two types of endurance improvements were proposed: two modules for a specific application, and a generic data management module. The first two modules are implemented for a specific application that runs on ASIC2. Both algorithms were developed according to the requirements of the application itself. The idea of the generic data management algorithm presented, called *Dataflash Module*, is based on the EEPROM emulation algorithm by STMicroelectronics (STM) [STM18]. STM's EEPROM emulation is further modified removing the CRC and implementing a Hamming[12,8] error detection and correction code inspired by the Hamming[12,8] presented in [Coo]. The width of virtual addressing and the data

portions are selected in order to address as many application variables with the data portion kept as the write-width of the ASIC2 Flash memory. Virtual address and data portion widths are also selected considering the capability of the Hamming[12,8] implementation.

Future work includes analysis of the location of the bit failures in a double-word from the already collected test data from the evalboard and the testbed with a more intricate data processing method. Lastly, a detailed testing of the proposed improvement of the endurance cycle of ASIC2 internal Flash memory is also planned on the same presented testbed design.

Appendix A

Testbed PCB Layout

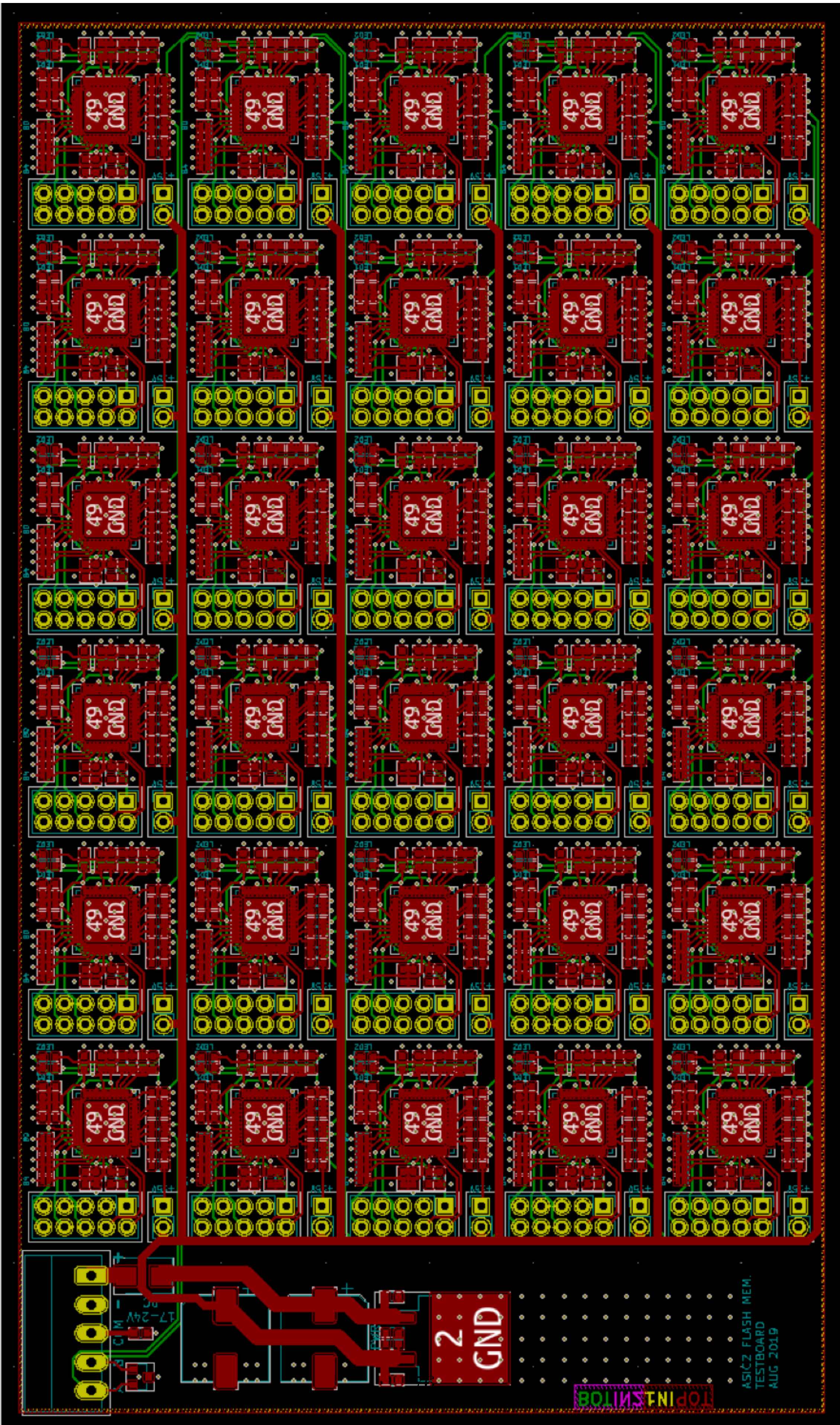

Figure A.1: ASIC2 Testbed PCB Layout

Appendix B

Dataflash Dataflow Charts

Write flow chart is separated into two parts: the part with no corruption (Figure B.1) and the part with corruption detected on the ACTIVE page (Figure B.2).

Next, the flow of Get is presented on Figure B.3.

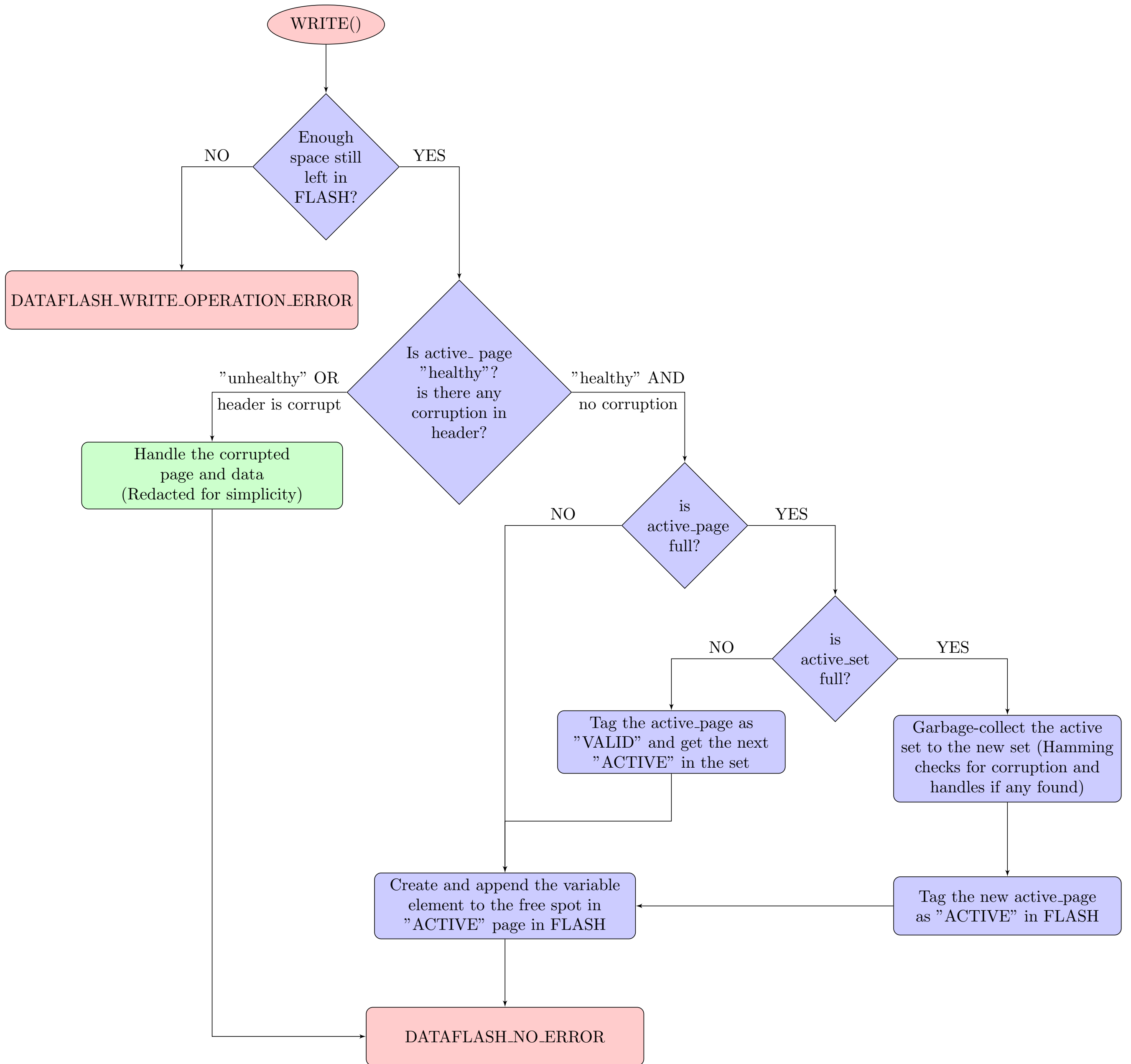

Figure B.1: Write function data flow [Simplified with no corruption]

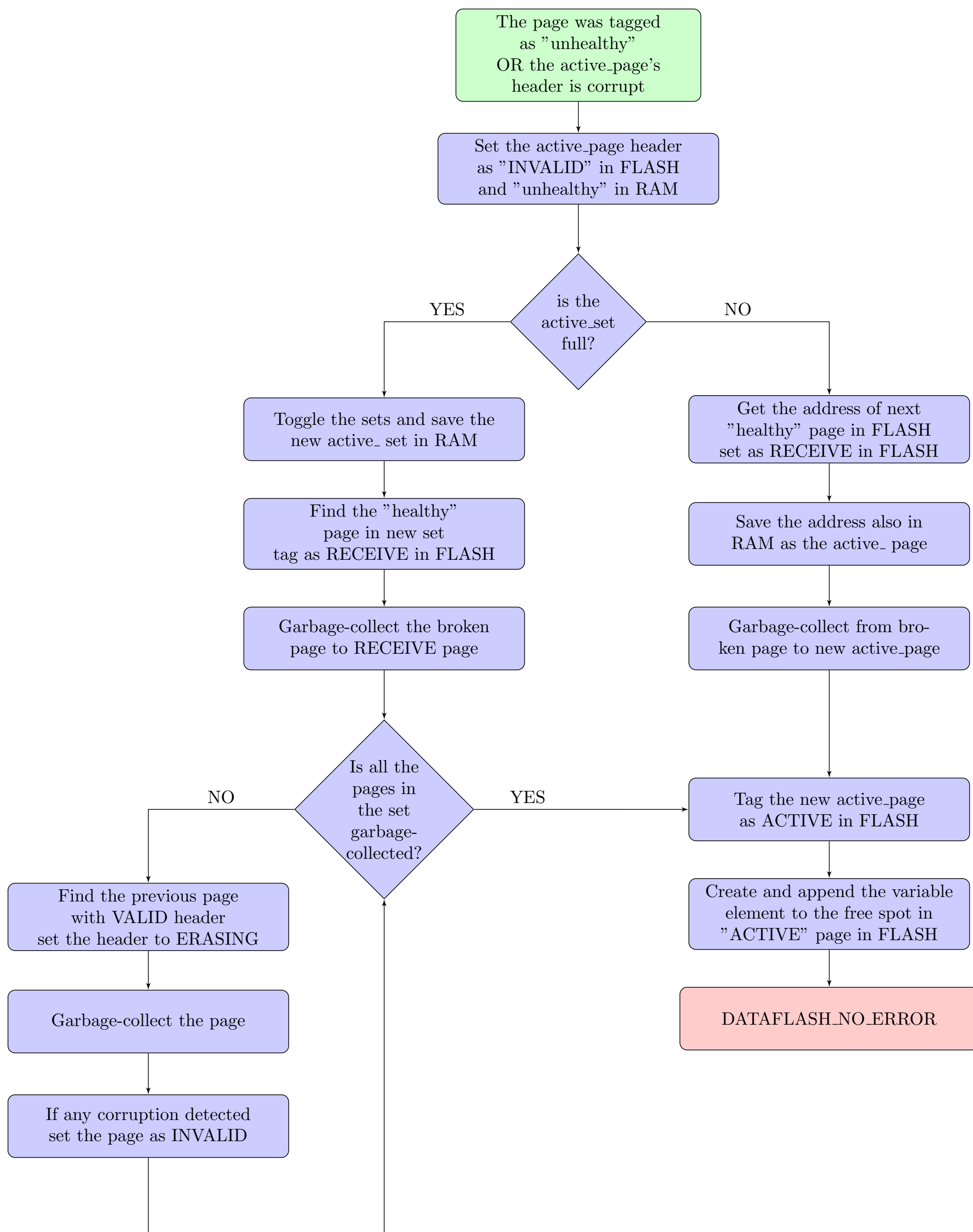

Figure B.2: Write function data flow - corruption in the active page

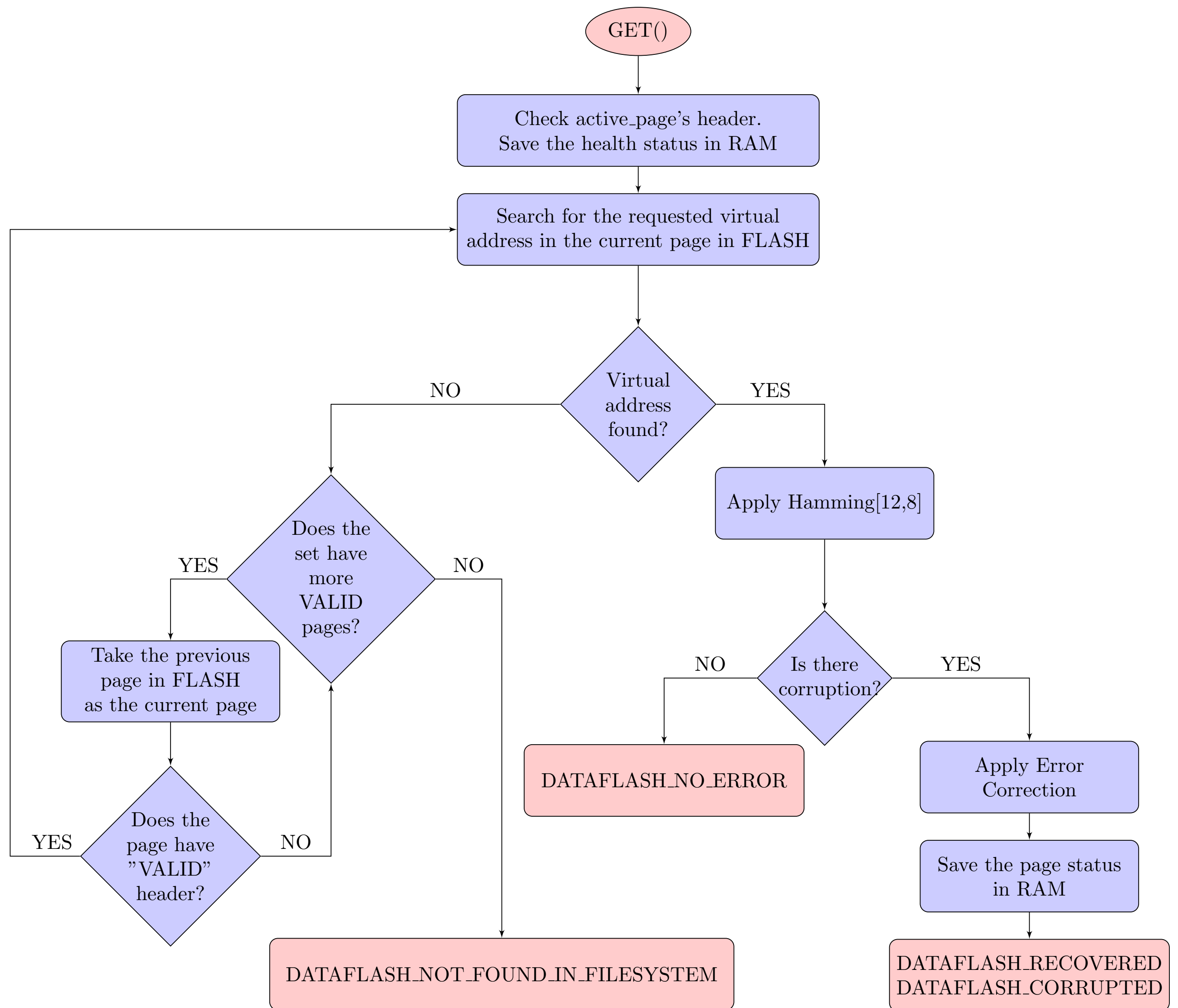

Figure B.3: Get function data flow

Appendix C

Dataflash: Simple Use Case Example

The following example shows how the dataflash module works as data management for three variables with associated virtual addresses as shown in Table C.1.

For the given amount of number variable elements which is 3, only single page for each set is enough. It is important to remind that, dataflash module has the limitation of minimum 2 pages for each set (which ends up as 6 pages in total considering the counter pages for endurance cycle) to have a high level of confidence. For the sake of example, the storage algorithm is visualized by using only one page for each set. Set1 and Set2 has the corresponding pages Page1 and Page2. To simplify the example, Hamming redundancies and the the unused 4-bit portion are not actually calculated but instead represented as "H" for each nibble (4-bit).

Variable	**Virtual address [HEX]**	**Data width**
Var1	0x01	32-bit
Var2	0x10	16-bit
Var3	0xFF	8-bit

Table C.1: Emulated EEPROM usage example variables

Figure C.1 shows only the write function calls which are done sequentially with increasing order of physical Flash memory addresses and how valid data toggle between sets to achieve high endurance and data reliability.

First variable 1 is written, and variable 2, and then variable 3. Later on variable 1 is written in a loop 60 times (to fill the page) and that point the first page and the first set is full. Next, write call is for variable 3 with data 0xFD and that particular write function call triggers the garbage collecting since the currently active set is already full.

Red lined section indicates garbage-collecting process. First ACTIVE set becomes VALID and later on it becomes ERASING as it was seen in the header status transition flow in Figure 4.6 and at the time VALID header becomes ERASING we have the RECEIVE header on the next set that is collecting the valid data. After garbage collecting is finished RECEIVE header becomes ACTIVE and same process goes on and on.

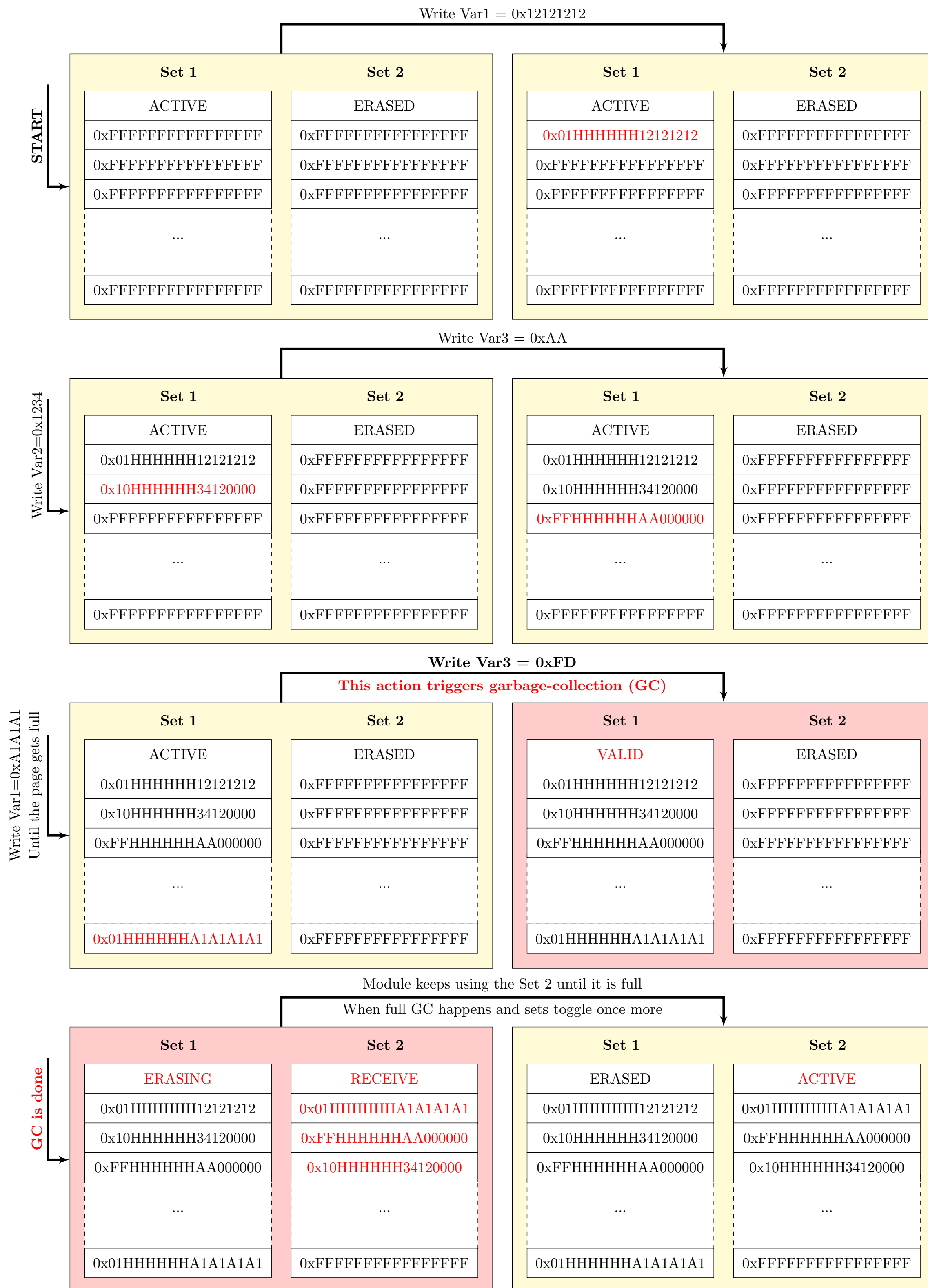

Figure C.1: Example data transitions

Appendix D

Dataflash Module Description

This section describes the dataflash module functions available for the user.

D.1 Key features

- User-configured EEPROM size
- Supports different size (up-to 32-bit) data variables
- Increased memory endurance compared to the raw Flash memory endurance
- Error detection and correction algorithm increases data reliability
- Robust against power-loss interrupts

D.2 Excel Calculator Tutorial

Excel sheet supplied is used to calculate user defines which are explained in detail in Section D.3. Sheet can be found in the module directory.

Sheet requires user to enter three inputs as seen in Figure D.1:

- Number of application variables [32-bits] : How many variables will be written during the application
- Desired number of write endurance cycles : How many times the variables will be updated during run time

- FLASH1 memory space start address [HEX] : The start address of the page right after the firmware ends which is available for dataflash module to start using as data management and counting endurance

INPUTS:	
Number of application variables [32-bits]	3
Desired number of write endurance cycles	1000000
FLASH1 memory space start address [HEX]	0x22000

Figure D.1: Input section in Excel calculator sheet

According to the given inputs, there is an info section to help the user as seen in Figure D.2. Info section is lets user know about:

- Memory space needed : How many pages/bytes will be needed to store given variables for given endurance cycle

- Variable element info : Whether the module can support given number of variable elements as there is a limitation of 256 variable elements due to virtual addressing section being only 8-bit.

- Memory space info : This is to let user know if the given start address in input is in FLASH1 and/or if it is indeed a valid start page address.

INFO:		
Memory space needed	Bytes	3072
	Pages	6
Var. El. Info	Number of variable elements is OK.	
Memory Space Info	Start address is OK.	

Figure D.2: Information about inputs on Excel calculator sheet

Lastly if given inputs are all okay, Definition section shown in Figure D.3 gives the outputs to be saved in both header file and linker file as shown in Section D.3.

DEFINITIONS:	
DATAFLASH_EPROM_SETTINGS.H	
DATAFLASH_APPLICATION_VARIABLE_ELEMENTS	3
DATAFLASH_ADDRESS_MIN	0x22000
DATAFLASH_ADDRESS_MAX	0x22BFF
LINKER :	
DATAFLASH_ADDRESS_MIN	0x22000
DATAFLASH_ADDRESS_MAX	0x22BFF

Figure D.3: Definitions to be saved in header and linker files

D.3 User Defines

The dataflash EEPROM emulation algorithm parameters need to be configured according to the application needs. Parameters must be calculated by using the Excel sheet (refer to Section D.2).

dataflash_eeprom_settings.h

```
1  #ifndef DATAFLASH_EEPROM_SETTINGS_H
2  #define DATAFLASH_EEPROM_SETTINGS_H
3
4  /*define application requiements for dataflash module*/
5  #define DATAFLASH_APPLICATION_VARIABLE_ELEMENTS         1
6  #define DATAFLASH_ADDRESS_MIN                           0x00029000U
7  #define DATAFLASH_ADDRESS_MAX                           0x00029BFFU
8  /*dataflash eeprom emulation definitions*/
9  #endif
```

Figure D.4: Header file for dataflash settings

```
34
35
36 define symbol __region_DATAFLASH_start__ = 0x00029000;
37 define symbol __region_DATAFLASH_end__ = 0x00029BFF;
38
39 define memory mem with size = 4G;
40 define region USER_DEFINED_MEMORY_region = mem:[from 0x0002FA00 to 0x0002FBFF];
41 define region CHECKSUM_region    = mem:[from 0x0002FDFC to 0x0002FDFF];
42
43
```

Figure D.5: Linker definition for dataflash module

D.4 Dataflash Module Files

Dataflash module comes with the following source files:

- **dataflash_count.c** : contains the counter functions to keep track of the write endurance, which is used by the dataflash algorithm internally.
- **dataflash_eeprom_emulation.c** : contains the EEPROM emulation functions those can be called from the user program:
  - dataflash_eeprom_init
  - dataflash_eeprom_write
  - dataflash_eeprom_get
  - dataflash_eeprom_diag
  - dataflash_eeprom_format_page
  - dataflash_eeprom_reset_counter
- **dataflash_utils.c** : contains necessary internal functions for dataflash module.
- **dataflash_hamming.c** : contains Hamming parity calculation function and ECC for the dataflash module.

D.5 User Function Definitions

The set of functions that can be called by the application program are described. They are defined in *dataflash_eeprom_emulation.c* module.

dataflash_eeprom_init()

Input: None
Return type: None

Configures the EEPROM emulation variables and memory space to their initial state as well as the endurance counter used by diagnostics.

In case of power loss during run time, it restores the Flash pages to a known state back again

dataflash_eeprom_write(uint8_t virtual_address, uint32_t data)

Input: 8-bit virtual address and up-to 32-bit data
Return type: dataflash_data_status_t

Takes the virtual address and the data associated with it and saves to the appropriate place in emulated EEPROM.

Data value can be up-to 32 bits, which means that less than 32 bits can be also used but it would be more favourable to combine smaller data into 32 bit of data to achieve higher endurance with the same amount of memory space allocated (See Excel sheet for calculations).

Returns status can be either:

- DATAFLASH_WRITE_OK: if the write operation was successful
- DATAFLASH_WRITE_ERROR: in case the majority of memory space is corrupted and not enough space is left to save data. Means that algorithm is expired and it won't keep writing.

dataflash_eeprom_get(uint8_t virtual_address, uint32_t *ptr)

Input: 8-bit virtual address and 32-bit pointer for data
Return type: dataflash_data_status_t

Gets the latest valid and healthy data from emulated EEPROM space that is associated with the given 8-bit virtual address. Return statuses may be:

- DATAFLASH_NO_ERROR: the data is returned with no corruption.
- DATAFLASH_RECOVERED: the data has been corrupted and error corrected.
- DATAFLASH_CORRUPTED: the data has been corrupted but it is recognized as uncorrectable.
- DATAFLASH_ADDRESS_DOESNT_EXIST: the virtual address doesn't exist in emulated space: either virtual address wasn't given by application at all OR it got corrupted and couldn't corrected properly.

In case of error recovered state, it is up to user to handle the information accordingly since the ECC implemented is able to detect and correct 1-bit error [Hamming128].

dataflash_eeprom_diagnostic()

Input: None
Return: dataflash_diagnostic

Returns the information about running EEPROM emulation. Returned diagnostic structure contains:

- uint8_t active_running_pages: Number of healthy running pages still in use. In the beginning it is defined by user how many pages in total is allocated for the emulation, so the difference will be the pages that has corrupted and omitted.
- uint32_t write_cycle: The endurance cycle. It is the number of write happened. So it is up to user to do comparison with the estimated number of endurance cycle calculated by Excel sheet.

dataflash_eeprom_format_page()

Input: None
Return: None

Formats a page with "ERASING" header beforehand that will be used by emulation algorithm later on to save from page erase time (20ms-40ms depending on configuration). Make sure to have enough time available (20-40ms) since erasing a flash page is a blocking function.

dataflash_eeprom_reset_counter()

Input: None
Return type: None

The counter algorithm that is used to count the write endurance cycle is limited to 16.7M cycles. Thus, if the application requires higher endurance count, in order to make sure the correct endurance count, the user is required to reset the endurance counter after it passes a certain amount i.e. 16M to start counting from 0 again.

Bibliography

[Ham50] R. W. Hamming. "Error Detecting and Error Correcting Codes". In: *Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1950.tb00463.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1950.tb00463.x.

[KS67] D. Kahng and S. M. Sze. "A floating gate and its application to memory devices". In: *The Bell System Technical Journal* 46.6 (1967), pp. 1288–1295.

[Wan79] S. T. Wang. "On the I-V characteristics of floating-gate MOS transistors". In: *IEEE Transactions on Electron Devices* 26.9 (1979), pp. 1292–1294.

[SCE80] R. E. Shiner, J. M. Caywood, and B. L. Euzent. "Data Retention in EPROMS". In: *18th International Reliability Physics Symposium*. 1980, pp. 238–243.

[Mie83] N. R. Mielke. "New EPROM Data-Loss Mechanisms". In: *21st International Reliability Physics Symposium*. 1983, pp. 106–113.

[SPC84] Simon Tam, Ping-Keung Ko, and Chenming Hu. "Lucky-electron model of channel hot-electron injection in MOSFET'S". In: *IEEE Transactions on Electron Devices* 31.9 (1984), pp. 1116–1125.

[Mas+85] F. Masuoka et al. "A 256K flash EEPROM using triple polysilicon technology". In: *1985 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*. Vol. XXVIII. 1985, pp. 168–169.

[Mas+87] F. Masuoka et al. "New ultra high density EPROM and flash EEPROM with NAND structure cell". In: *1987 International Electron Devices Meeting*. 1987, pp. 552–555.

[VM88] G. Verma and N. Mielke. “Reliability performance of ETOX based flash memories”. In: *26th Annual Proceedings Reliability Physics Symposium 1988*. 1988, pp. 158–166.

[Had+89] S. Haddad et al. “Degradations due to hole trapping in flash memory cells”. In: *IEEE Electron Device Letters* 10.3 (1989), pp. 117–119.

[Suz+89] E. Suzuki et al. “Hole and electron current transport in metal-oxide-nitride-oxide-silicon memory structures”. In: *IEEE Transactions on Electron Devices* 36.6 (1989), pp. 1145–1149.

[Cri+90] G. Crisenza et al. “Charge loss in EPROM due to ion generation and transport in interlevel dielectric”. In: *International Technical Digest on Electron Devices*. 1990, pp. 107–110.

[Cri+92] G. Crisenza et al. “Floating gate memories reliability”. In: *Quality and Reliability Engineering International* 8.3 (1992), pp. 177–188. DOI: `10.1002/qre.4680080305`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.4680080305`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.4680080305`.

[Ari+93] S. Aritome et al. “Reliability issues of flash memory cells”. In: *Proceedings of the IEEE* 81.5 (1993), pp. 776–788.

[Cap+94] P. Cappelletti et al. “Failure mechanisms of flash cell in program/erase cycling”. In: *Proceedings of 1994 IEEE International Electron Devices Meeting*. 1994, pp. 291–294.

[Dun+94] C. Dunn et al. “Flash EPROM disturb mechanisms”. In: *Proceedings of 1994 IEEE International Reliability Physics Symposium*. 1994, pp. 299–308.

[Cri+96] G. Crisenza et al. “Non Volatile Memories: Issues, Challenges and Trends for the 2000’s Scenario”. In: *ESSDERC ’96: Proceedings of the 26th European Solid State Device Research Conference*. 1996, pp. 121–130.

[Mor+96] S. Mori et al. “Thickness scaling limitation factors of ONO interpoly dielectric for nonvolatile memory devices”. In: *IEEE Transactions on Electron Devices* 43.1 (1996), pp. 47–53.

[EM97] Valerie J. Easton and John H. McColl. *Statistics Glossary v1.1.* 1997. URL: http://eprints.gla.ac.uk/120164/.

[Pav+97] P. Pavan et al. “Flash memory cells-an overview”. In: *Proceedings of the IEEE* 85.8 (1997), pp. 1248–1271.

[Sta97] Yale University - Department of Statistics. *Confidence Intervals.* 1997. URL: http://www.stat.yale.edu/Courses/1997-98/101/confint.htm (visited on 1997).

[Con+98] A. Concannon et al. “A novel CMOS compatible multi-level flash EEPROM for embedded applications”. In: *56th Annual Device Research Conference Digest (Cat. No.98TH8373).* 1998, pp. 78–79.

[Goo98] Ad J. van de Goor. “Testing Semiconductor Memories: Theory and Practice”. In: 1998.

[Lai98] S. Lai. “Flash memories: where we were and where we are going”. In: *International Electron Devices Meeting 1998. Technical Digest (Cat. No.98CH36217).* 1998, pp. 971–973.

[Mod99] A. Modelli. “Reliability of thin dielectric for non-volatile applications”. In: *Microelectronic Engineering* 48.1 (1999), pp. 403–410. DOI: https://doi.org/10.1016/S0167-9317(99)00414-1.

[PB99] Paolo Pavan and Roberto Bez. “The Industry Standard Flash Memory Cell”. In: *Flash Memories.* Boston, MA: Springer US, 1999, pp. 37–90.

[SF99] Luca Selmi and C. Fiegna. “Physical Aspects of Cell Operation and Reliability”. In: Jan. 1999, pp. 153–239. ISBN: 978-0-7923-8487-8. DOI: 10.1007/978-1-4615-5015-0_4.

[Ari00] S. Aritome. “Advanced flash memory technology and trends for file storage application”. In: *International Electron Devices Meeting 2000. Technical Digest. IEDM (Cat. No.00CH37138).* 2000, pp. 763–766.

[SL00] Yun-Gueon Shin and Jong-Hwa Lee. “A study of electrical characteristics and reliability on flash EEPROM cell”. In: vol. 2. Feb. 2000, 228–233 vol. 2. ISBN: 0-7803-6486-4. DOI: 10.1109/KORUS.2000.866031.

[Iel+01] D. Ielmini et al. “Statistical modeling of reliability and scaling projections for flash memories”. In: *International Electron Devices Meeting. Technical Digest (Cat. No.01CH37224)*. 2001, pp. 32.2.1–32.2.4.

[Lee+01] W.H. Lee et al. “Data retention failure in NOR flash memory cells”. In: Feb. 2001, pp. 57–60. ISBN: 0-7803-6587-9. DOI: `10.1109/RELPHY.2001.922882`.

[Hoe+02] A. Hoefler et al. “Statistical modeling of the program/erase cycling acceleration of low temperature data retention in floating gate nonvolatile memories”. In: *2002 IEEE International Reliability Physics Symposium. Proceedings. 40th Annual (Cat. No.02CH37320)*. 2002, pp. 21–25.

[Iel+02] D. Ielmini et al. “Localization of SILC in flash memories after program/erase cycling”. In: *2002 IEEE International Reliability Physics Symposium. Proceedings. 40th Annual (Cat. No.02CH37320)*. 2002, pp. 1–6.

[Kuo+02] Kuo-Liang Cheng et al. “RAMSES-FT: a fault simulator for flash memory testing and diagnostics”. In: *Proceedings 20th IEEE VLSI Test Symposium (VTS 2002)*. 2002, pp. 281–286.

[Bez+03] R. Bez et al. “Introduction to flash memory”. In: *Proceedings of the IEEE* 91.4 (2003), pp. 489–502.

[Jae+03] Jae-Duk Lee et al. “Data retention characteristics of sub-100 nm NAND flash memory cells”. In: *IEEE Electron Device Letters* 24.12 (2003), pp. 748–750.

[Che+04] Chen He et al. “Reliability model and implementation for EEPROM emulation using flash memory”. In: *2004 IEEE International Reliability Physics Symposium. Proceedings*. 2004, pp. 657–658.

[Fie04] James Fiedler. *Hamming Codes*. 2004. URL: `https://orion.math.iastate.edu/linglong/Math690F04/HammingCodes.pdf`.

[Gin+06] Olivier Ginez et al. “An overview of failure mechanisms in embedded flash memories”. In: *24th IEEE VLSI Test Symposium* (2006), 6 pp.–113.

[Bre07] Joe E. Brewer. *Introduction to Nonvolatile Memory.* John Wiley & Sons, Ltd, 2007. ISBN: 9780470181355. DOI: 10.1002/9780470181355.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470181355.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470181355.ch1.

[Gay07] Georgi Nedeltchev Gaydadjiev. "Testing of modern semiconductor memory structures". In: (2007).

[Yeh+07] J. Yeh et al. "Flash Memory Testing and Built-In Self-Diagnosis With March-Like Test Algorithms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.6 (2007), pp. 1101–1113.

[CAO08] Andrea Chimenton, Massimo Atti, and Piero Olivo. "Reliability of floating gate memories". In: *Error Correction Codes for Non-Volatile Memories.* Dordrecht: Springer Netherlands, 2008, pp. 103–130. ISBN: 978-1-4020-8391-4. DOI: 10.1007/978-1-4020-8391-4_5. URL: https://doi.org/10.1007/978-1-4020-8391-4_5.

[God08] Benoît Godard. "Design for Reliability Techniques in Embedded Flash Memories". PhD thesis. Université Montpellier II - Sciences et Techniques du Languedoc, Oct. 2008.

[Kiz+08] M.E. Kiziroglou et al. "Thermionic field emission at electrodeposited Ni–Si Schottky barriers". In: *Solid-State Electronics* 52.7 (2008), pp. 1032–1038. ISSN: 0038-1101. DOI: https://doi.org/10.1016/j.sse.2008.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0038110108000713.

[Ins09] National Instruments. *Introduction to MODBUS.* National Instruments, NI. 2009. URL: https://www.micronor.com/products/files/AN112/AN112_NIModbusTutorial.pdf.

[BD10] Simona Boboila and Peter Desnoyers. "Write Endurance in Flash Drives: Measurements and Analysis". In: *FAST'10.* San Jose, CA, 2010.

[Lab10] Silicon Laboratories. *EEPROM EMULATION FOR FLASH MICROCONTROLLERS.* 1st ed. Silicon Laboratories Inc. Austin, Texas, 2010.

[Moh+10] Vidyabhushan Mohan et al. “How i Learned to Stop Worrying and Love Flash Endurance”. In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Storage and File Systems*. HotStorage'10. Boston, MA: USENIX Association, 2010, p. 3.

[Azi+11] H. Aziza et al. “Non volatile memory reliability prediction based on oxide defect generation rate during stress and retention tests”. In: *2011 International Semiconductor Device Research Symposium (ISDRS)*. 2011, pp. 1–2.

[HC12] Li Hai and Yiran Chen. “Nonvolatile Memory Design”. In: CRC Press, 2012. Chap. 1-2. DOI: https://doi.org/10.1201/b11354.

[M P12] Dr M Parvathi. “Modified March C-With Concurrency in Testing for Embedded Memory Applications”. In: *International Journal of VLSI Design & Communication Systems* 3 (Oct. 2012), pp. 43–51. DOI: 10.5121/vlsic.2012.3504.

[Cal+13] M. Calabrese et al. “Accelerated reliability testing of flash memory: Accuracy and issues on a 45nm NOR technology”. In: *Proceedings of 2013 International Conference on IC Design Technology (ICICDT)*. 2013, pp. 37–40.

[Sha13] Ashok K. Sharma. *Semiconductor Memories: Technology, Testing, and Reliability*. 1st. Wiley-IEEE Press, 2013. ISBN: 0780310004.

[LL14] Pierre Camille Lacaze and Jean-Christophe Lacroix. “Non-Volatile Memories”. In: John Wiley & Sons, Ltd, 2014. Chap. 1-2.

[Lom14] Chris Lomont. *Flash Endurance Testing*. 2014. URL: https://hypnocube.com/2014/11/flash-endurance-testing/ (visited on 11/25/2014).

[Mac14] Macronix. *Program/Erase Cycling Endurance and Data Retention of Macronix SLC NAND Flash Memories*. 1st ed. Macronix International Co., Ltd. Oct. 2014.

[BB17] Olaf Boecker and Boris Buchheimer. *Studie Dataflash*. Tech. rep. 1. Eaton Industries GmbH, 2017.

[Mic17] Micron. *NOR Flash Cycling Endurance and Data Retention*. C. Micron Technology Inc. Nov. 2017.

[Rah17] Robbi Rahim. *BIT ERROR DETECTION AND CORRECTION WITH HAMMING CODE ALGORITHM.* Sept. 2017. DOI: 10.31227/osf.io/j3w5z. URL: osf.io/preprints/inarxiv/j3w5z.

[Zha+17] Tao Zhang et al. "Flash Drive Lifespan *is* a Problem". In: *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. HotOS '17. Whistler, BC, Canada: Association for Computing Machinery, 2017, pp. 42–49. ISBN: 9781450350686. DOI: 10.1145/3102980.3102988. URL: https://doi.org/10.1145/3102980.3102988.

[Boe18] Olaf Boecker. *ASIC2 Datasheet*. Eaton Industries GmbH ICPD-E-S&S-PD. 2018.

[STM18] STMicroelectronics. *EEPROM emulation techniques and software for STM32 microcontrollers*. 2nd ed. STMicroelectronics. Geneva, Switzerland, 2018.

[Mic19] Microchip. *32-bit Microcontrollers (up to 256 KB Flash and 64 KB SRAM) with Audio and Graphics Interfaces, USB, and Advanced Analog*. PIC32MX150F128B. Microchip Technology Inc. May 2019.

[Tec19] Microchip Technology. *EEPROM Emulation for Flash-Only Devices*. Microchip Technology Inc. 2019.

[Coo] David Cook. *Hamming Error Correcting Code (ECC)*. URL: https://www.robotroom.com/Hamming-Error-Correcting-Code-1.html.

[Eat] Eaton. *SmartWire-DT intelligent wiring system*. URL: https://www.eaton.com/us/en-us/catalog/machinery-controls/smartwire-dt-intelligent-wiring-system.html.

[Mar+] S Martirosyan et al. "An efficient testing methodology for embedded flash memories". In: *2017 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, pp. 1–4.