Analysis of flash memory wear based on cache configuration and available memory

Markus Andersson
ma6254an-s@student.lu.se

Filip Hermansson
fi2807he-s@student.lu.se

August 28, 2023

Department of Electrical And Information Technology
Lund University

Master's thesis work carried out at Beijer Electronics AB.

Supervisor: Erik Larsson, erik.larsson@eit.lth.se
Examiner: Christian Nyberg, christian.nyberg@eit.lth.se

Abstract

This thesis aimed at examining the effects of file and disk cache for certain writing patterns and file sizes for Beijer Electronic's human machine interface (HMI) panels with the focus on write amplification factor (WAF). The goal was to determine optimal configurations to reduce WAF for different use cases to prolong the lifetime of Beijer Electronic's HMI panels. A test application and logs were created to measure commands sent over the bus for chosen tests. It was found that write amplification decreased naturally as file sizes grew. It was also found that enabling disk cache without file cache had the most negative impact on the write amplification factor, but that enabling file cache together with disk cache reduced the negative effect as well as improved writing speed. Furthermore, only enabling file cache or not enabling any form of cache resulted in the lowest write amplification factor for file sizes between 10 and 10000 KB, however always resulting in slower execution times than configurations with both caches enabled. It was also found that operating the panels at close to max capacity reduced writing speed because of the internal wear-leveling required for the SD cards during tests to not wear out prematurely, but no increase in write amplification factor was measured.

Keywords: flash memory, file cache, disk cache, write amplification factor, over-provisioning, HMI panel, wear leveling

Acknowledgements

To begin with we would like to thank Erik Larsson at Lund University who, through consistent enthusiastic guidance, made the technical and administrative parts of this masters thesis a transparent and easygoing process.

Furthermore we want to thank Beijer Electronics for both the opportunity and resources to make it possible to conduct this masters thesis. At Beijer Electronics we would like to especially thank Christoffer Ohlsson and Daniel Forsberg for their role in presenting the project, supplying all technical equipment and solving all other administrative tasks that occurred during the process of this masters thesis.

Lastly we would like to send a big thank you to Stefan Lindgren, also at Beijer Electronics, for his passionate and continuous technical guidance with the possibility to discuss both practical and theoretical problems. Without you the scientific parts of this masters thesis would be significantly more difficult and tedious.

Contents

Glossary

.NET A software framework developed my Microsoft. Used to develop applications that may be executed on the Windows OS. 33

API Stands for Application Programming Interface. Is a software mediator that allows two applications to communicate. Often used to more easily extract and share data within or between different organization products. 22

ATP Company manufacturing NAND memories. 36

capacitor Device that stores electrical energy in an electric field. 14

CRC Stands for Cyclic Redundancy Check. Is an error-detecting code commonly used in storage devices to detect and fix accidental data changes to minimize any data corruption. 19, 33

die Flash die, or chip, is a segment of cells. 21

EEPROM Stands for electrically erasable programmable read-only memory. 18

embedded system A computer system where the computer processor, memory and input/output is combined. Often used in smaller mechanical or electronic systems. In shorter terms a computer where the memory, processor and input/output is placed on a single board. 22

eMMC Stands for Embedded Multimedia Card. Is an embedded storage solution which has a MMC interface, flash memory and controller. Used for high performance application such as smart phones and tablet computers. 12, 34

FAT Stands for file allocation table. Is a method of storing data commonly used in embedded systems. Variations include FAT16 and FAT32. 24

FTL Stands for flash translation layer. 21

GUI Stands for Graphical User Interface. A form or user interface that allows a user to interact with an electronic device through graphical icons such as buttons. 33

IDE Stands for Integrated Development Environment. A software application providing extensive equipment for software development. Often containing a source code editor, build automation tools and a debugger. 34

MOSFET Stands for metal–oxide–semiconductor field-effect transistor. 15

NAND NAND, or NOT-AND is a logic gate representing the logical inverse of the AND gate. 19

NOR NOR, or NOT-OR is a logic gate representing the logical inverse of the OR gate. 17

OS image A collection of files that contains the OS, the executable and all related data files. In other words the collection of files that makes the OS functional. 13

OS kernel A computer program in the absolute core of an operating system that is always present in memory. The kernel handles interactions between software and hardware controls such as memory or I/O. 22

PLC Stands for Programmable Logic Controller. An industrial computer often used in manufacturing processes especially designed to withstand harsh conditions. 12

RAM Stands for Random Access Memory. Type of computer memory where each element can be changed in any order in essentially the same amount of time. 15

ROM Stands for Read Only Memory. Memory that cannot be modified once written to. 23

SD Stands for secure digital and is a type of memory commonly used by computers. It is a non-volatile flash memory. 34

SPI The Serial Peripheral Interface is a short-distance interface used for synchronous comm unctions through serial ports. 35

TexFAT Stands for transaction safe extended file allocation table. It is an extension of FAT, is transaction safe and supports large tables. 24

tunneling Tunneling in quantum mechanics is a concept that electrons can behave as waves or particles. And can cancel the effects of an energy barrier provided that it is thin enough. This means that an electron can pass through a physically impassable medium since there is a non-zero chance that it can exist on the other side. 18

volatile memory Computer memory that only maintains its data when the device is powered. 14

Win32 A 32-bit version of Windows. 22

Windows Registry A hierarchical database that stores low-level settings for the Microsoft Windows operating system. 34

Chapter 1
Introduction

Although the term flash memories is a comprehensive term with multiple types of storage devices, all useful in specific contexts, they all have one common unavoidable flaw. Since the basic principle of all flash memories is to store and contain a specific electronic charge in each memory transistor the memory will, through continuous use, inevitable degrade. This degradation yields a memory unable to successfully store the intended information and will eventually become unusable. Due to the fact that each flash memory has a lifetime the ability to lower the degradation, and hence extend the memory lifetime, becomes a main factor.

The topic of memory degradation is something that has been well studied since the creation of flash memories. With this heavy scientific background multiple countermeasures to more efficiently write information to flash memory have been developed to lower the degradation rate. However, many of these countermeasures are conducted by privatized companies that develop and sell flash memory devices. Consequently, due to the ability to stay competitive, a majority of companies choose to keep the internal schematics of their memory efficiency algorithms hidden. [1] As a natural consequence the collaboration between certain software systems and a flash memory storage device may become more difficult.

One of the software systems where this difficulty becomes more predominant is the operating system. The reason for this is because a vast majority of operating systems closely handle the storage process of storing software program information in connected hardware storage devices. Therefore any ambiguity of internal working algorithms in the hardware storage device may cause issues in the storage process. These issues often result in memory inefficiency which will lower the lifetime of the storage device.

Due to these reasons the purpose of this masters thesis is to conduct research of the synergetic storage process between an operating system running on a hardware that has a connected flash memory storage device. By running custom designed memory storing programs in the context of a specific operating system the results of this masters thesis might hopefully

be used to better design future memory efficient programs in the operating system.

The structurally design of this report is as follows. In chapter 2 the background of the project is presented. In this chapter the project problem description, thesis goals and scope are given. Additionally the reader can also find all relevant technical information behind the research in chapter 2. In chapter 3 the methodology of all conducted tests as well as test configurations are described. Chapter 4 presents the results of the conducted tests and chapter 5 yields a discussion of the results in chapter 4. Ultimately chapter 6 presents a concise conclusion of this masters thesis.

Chapter 2
Background

2.1 Beijer Electronics AB

This masters thesis was conducted at Beijer Electronics' headquarters located in Malmö, Sweden. Beijer Electronics is a multinational company that specializes in communication, digitization and automation through web based services and hardware products. Although the company's initial business model mainly consisted of producing programmable logic controllers (PLC), micro data systems and other smaller hardware equipment it has gradually shifted towards producing more advanced hardware with a graphical user interface front-end. Today one of the main hardware products manufactured by the company is their HMI panels. [6]

2.1.1 Human-Machine Interface

Following the rapid and advanced evolution of computer based information systems used by the modern society there is no surprise that humanity have become highly dependent on different computer systems with a flawless communication working within or between them. These days a vast majority of industry processes uses some type industrial control system (ICS), often consisting of multiple subsystems, that requires sufficient communication to not halt either parts of or a whole process in its entirety. Although many parts of industry processes today have become automated, by either software or hardware, there are still parts that require human interaction. This is where HMI:s play a central role.

Simply put a HMI is a user interface that connects a person to a system of machines, often in the context of an industrial process, by displaying and sending sufficient data to connected machines or other HMIs in the system. The interface hardware itself is often in a form of a screen or dashboard (see Figure 2.1) which in many cases is even built-in to a machine but can also exist as an independent separated panel. The HMI hardware enables a two-way commu-

nication between a human operator and a machine. This is most often achieved by letting the HMI communicate directly to a PLC, or some other type of industrial controller, that exists in the system. [5]

The existing two-way communication allows a human operator to send commands to a machine or receive data from it. The operator can often easily control one or multiple machines by just a simple touch of a screen. The commands sent from the HMI to the machine vary in complexity. Simple commands are just to turn the machine on or off. More complex commands however, may allow the operator to control certain algorithms running in the machine or even perform specific urgent commands in case of an emergency. Finally since the HMI can display useful data sent from a machine it allows any authorized user to see important information such as process status, reports and historical logs. [21, p. B-7]

Figure 2.1: A Beijer Electronics HMI panel. [7]

2.1.2 Problem Description

Beijer Electronics place significance on delivering well functioning products of high quality to their customers. For Beijer Electronics, whose main product line is their HMI panels, this significance of high quality translates into developing and producing hardware of high endurance together with well structured software that may be executed on the panel. Without these important qualities the company cannot sufficiently guarantee that their products upholds the warranty that is placed on the product. Unfortunately, with hardware products consisting of multiple hardware subparts, constructed by different suppliers, this is not always an easy task.

This difficulty is of course also present in Beijer Electronic's HMI panels where the internal schematics and algorithms of the different hardware subparts are not always explicitly known. They may also differ between different production batches depending on what subsuppliers Beijer Electronic's suppliers themselves use for a specific production batch. One of the more central hardware subparts of the HMI panel, where this difficulty becomes more evident, is the eMMC flash memory. Although degradation for any flash memory is inevitable a number of Beijer Electronics customers have continuously experienced problems with flash memory degradation of a too fast rate where the warranty placed on the HMI panel cannot be successfully sustained.

As described the reasons for this fast rate of flash memory degradation may depend explicitly on faults in the hardware and the cooperation between multiple hardware subparts from different producers. However, in reality, this is only one piece of the puzzle. The producers of the hardware parts in the HMI panels are well established, making the risk of faulty hardware close to non-existent. Hence the reason for a fast memory degradation might be just as, or even more, likely to stem from the customer's explicit usage of the HMI panel and the software they execute on it. This of course introduces a new set of challenges for the company, namely retrieving and examining the software used by the customers who experience a too fast memory degradation. In practice this is often a tedious process and might not even be achievable in certain cases.

2.1.3 Thesis Goals

Producing and selling HMI panels that successfully meet the placed product warranty are obviously important features for Beijer Electronics. Having costumers experiencing a too fast rate of flash memory degradation, resulting in the entire HMI panel being considered consumed in a time span before the end of the placed warranty, may eventually harm the reputation of the company and hurt its chances of staying competitive. Hence the objective of this masters thesis is to yield more understanding about the flash memory that Beijer Electronics' HMI panels use. Particularly how different built in OS caches, namely the file and disk cache, may affect the degradation of the flash memory. Hopefully the results of this thesis may be used in the future to produce more efficient OS images that will degrade the flash memories at a slower rate which will help Beijer Electronics stay competitive.

2.1.4 Limitations and Scope

Due to the fact that this work is conducted as a masters thesis there is an implicit time limit, namely 20 weeks each consisting of 40 hours. This time limit is the central reason to why certain decisions need to be made to restrict the scope in such a way that it can be realistically completed within the 20 weeks. As discussed in the previous section the reasons for a fast memory degradation might depend on both hardware faults and certain memory heavy software usage. Because of the limited time span this masters thesis will focus on handling software usage and in what way software running on the HMI panel may degrade the flash memory. The reason for this decision is that examining faulty hardware is harder and requires multiple HMI panels which is harder and costlier. The risk that faulty hardware is the reason for fast memory degradation also significantly lower than the reason being memory heavy software usage.

This masters thesis will solely conduct research on how the HMI panel's operating system, Windows Compact Embedded, can be used to slow down the flash memory degradation. Since all customer software running on the HMI panel will use Windows Compact Embedded focusing on the operating system is a more specific and better suited approach. However many of the results found in this paper might still be applicable to other operating systems.

Furthermore, due to the reason that the flash eMMC memory is soldered directly to the

embedded board in the HMI panel it cannot be exchanged when degraded. Hence the entire HMI panel is considered consumed whenever the eMMC flash memory is worn out. Because of this reason flash memory degrading tests executed in this paper will not use the internal eMMC flash memory but will instead use external SD cards inserted into the HMI panel. These cards are of course exchangeable when degraded and will not consume the entire HMI panel. Since degradation of both eMMC and SD card flash is a time consuming process, often taking well over a month of constant usage before significant degradation becomes visible, the specific SD card used in the tests will be a (industrial) card that degrades as fast as possible.

Lastly, without the possibility of collecting user metadata such as file sizes, results acquired from tests conducted may differ from costumer experiences. Attempts will be made to emulate customers' memory read/write patterns, but will not be identical.

2.1.5 Research questions

Since the thesis, analysis and research all have been conducted in close correlation with a large multinational company the research questions have been formulated in a way to put the main focus on solving the specific company based problems. However the question formulation has been carefully chosen to create generalization and hence find a good balance between cooperate interests and academic research.

- How do file cache and disk cache effect flash wear in embedded systems?
- How do different levels of free memory effect IO performance and flash wear?
- What are the optimal panel configurations for Beijer Electronics?

2.2 Flash Memory

As long as computers have existed the ability to store data has played a central role for the development of the modern small, fast and efficient computers that exist today. Although notorious for being of enormous size in the past one can today, through decades of revolutionary innovation, hold a physical memory chip available to store multiple gigabytes of data in the palm of the hand. By using any kind of computational electrical device one will unconsciously be dependent on some kind of memory storage. In fact, without any memory a computer processor would not be able to compute anything of value since, just for basic calculations, a computer reads and writes to memory multiple times a second. [2, p. 5-7]

For a majority of computer devices the ability to use memory both temporarily, for current ongoing calculations, and more statically, to store long lasting information (even through loss of power), is elementary. To achieve this computers generally consist of at least two different types of memory, both with different traits depending on their specific purpose. Memory used as storage space for ongoing program instructions currently executed in the computer processor needs to be fast but continuously updated, often multiple times a second. This kind of volatile memory memory will not need to store long lasting data and therefore generally consists of an electrical network of capacitors that store binary data as long as an electric

voltage is applied across the network. Although this kind of memory is essential for the performance of the computer it will not be discussed further since this thesis only handles flash memory and its immediate surroundings. [24, p. 657, 664-667]

Flash memory is a read-write non-volatile RAM computer memory. To achieve the non-volatile characteristic the flash memory uses a special kind of transistor, the floating gate transistor, to physically trap electrons in a insulate layer consisting of silicon using a high voltage pulse. When electrons are trapped in the transistor they will remain even when there is no electrical voltage present and hence the non-volatile feature has been successfully achieved. However, due to the non-volatile feature, flash memories must have an erase-before-write architecture. Therefore, when a memory block is to be reprogrammed, an erase cycle must be executed to reset all affected memory cells before they can be written to again. Although all flash memories achieve the non-volatile feature there are differences between certain types of flash. Common differences are the number of bits stored in each floating gate transistor and how memory blocks are reprogrammed. [24, p. 647]

The architecture of floating gate transistors allows, as stated above, the memory to be non-volatile. But it is also the reason why flash memory gradually wears out. The nature of erase operations are such that when many cycles are performed the voltage threshold for programming and erasing are reduced and increased respectively, which is a result of oxide aging. The results of this aging is slower program/erase times which introduces an end of life expectancy for the memory. [17]

Even though the term flash memory might, to many readers, not be entirely unfamiliar it can still be difficult to grasp the practical use for this type of memory. However, average people, regardless of technical background, likely to encounter some kind of flash memory in their everyday life. Due to the importance of flash memory it is today a widespread technology used in many different branches where different memory size, shape and speed are relevant. A common flash memory device that is used in both a professional and personal context is the Universal Serial Bus (USB) stick. Thanks to its robustness and relative small size it has become a popular item to move or temporarily store data. An even smaller flash memory device is the SD memory card which is a popular choice to store data in the contexts of smaller electric devices where data retrieval needs to be fast and conducted often. A common flash memory choice for embedded system is the eMMC flash. It is soldered directly to the embedded system and cannot be easily replaced. Lastly, a common memory device for computers is the SSD flash. Although a bit bigger in size and not as easily exchangeable as the USB and SD card flash it can often handle faster transfer speeds.

2.2.1 The Floating Gate Transistor

Flash memories consist of a special kind of transistor called the floating gate transistor, see Figure 2.2. This transistor, just as for most non-volatile read write memories, is the essential element why flash memory gains the feature of non-volatility. The design resembles a regular MOSFET transistor but it has one additional gate, called the floating gate, placed between

the control gate and the channel. In contradiction to the control gate, the floating gate is not connected to anything which is why the gate has received its name. By making a certain threshold of electrons move from the channel to the floating gate the floating gate transistor achieves the possibility to store binary information in a non-volatile fashion. By using the ability to store electrons in the floating gate a specific amount of binary information, depending on the specific cell programming (see Section 2.2.5), can be written to and read from the transistor. However, due to the chemical and physical compounds of the floating gate transistor any information currently stored in the floating gate must be erased before it can be reprogrammed and store new information. [24, p. 647-649]

Avalanche-injection to the Floating Gate

In the floating gate transistor the drain is connected to ground whilst the source is connected to the load, that is, the power system. This will effectively create an electrical current between source and drain when the power system connected to the source is turned on. Hence by applying a sufficiently strong positive load to the control gate a portion of the electrons will gain enough energy to pass through the oxide insulate, through an effect called avalanche injection, and enter the floating gate. This is of course due to the electrical field that exists in the channel between source and drain. However, because of the negative charge of the electron the more electrons that enter the floating gate the less the effective voltage over the channel will become. Eventually a certain threshold will be reached and no more electrons can avalanche inject. When the positive load of the control gate is removed any electrons placed in the floating gate will become trapped and remain there until an erase event is executed on the transistor. In this way the floating gate contains its specific charge even when all power in the system is removed which is the core concept behind the non-volatility of the flash memory. [24, p. 647-648][8, p. 493]

Reading Binary information from the Floating Gate

Depending on the amount of electrons that are trapped in the floating gate it will contain a specific charge. By now applying a reference voltage across the source and drain a specific current can be measured and translated into specific binary values. If more electrons are trapped in the floating gate less current will pass through it and vice versa. Now the binary values are be mapped to the specific measured voltage of the transistor which directly depends on the charge of the floating gate. [8, 493-494] How binary values are mapped to measured voltages depends on the cell coding of the transistor which can be read in Section 2.2.5. Additionally a more detailed explanation of retrieving binary information from flash memory cells is located in Section 2.2.4.

Erasing the charge and insulate degradation

Since the reprogramming of the floating gate requires the charge to be reset the trapped electrons must somehow leave the floating gate. This is done by applying a high voltage charge across source and drain while simultaneously applying a sufficiently strong negative charge on the control gate. This forces the trapped electrons through the oxide insulate and into the channel which will neutralize the charge of the floating gate. Each cycle of write

and erase commands will require high voltages to force electrons through the oxide insulate. This will inevitably lead to the degradation of the oxide insulate layer and make it thinner. A thinner insulate layer will lose the capability to uphold the charge when the power is off and hence the non-volatile functionality of the transistor will degrade until an end-of-life threshold is reached. The erasing architecture of flash memory cells will be more thoroughly explained in Section 2.2.3. [8, p. 492, 495]

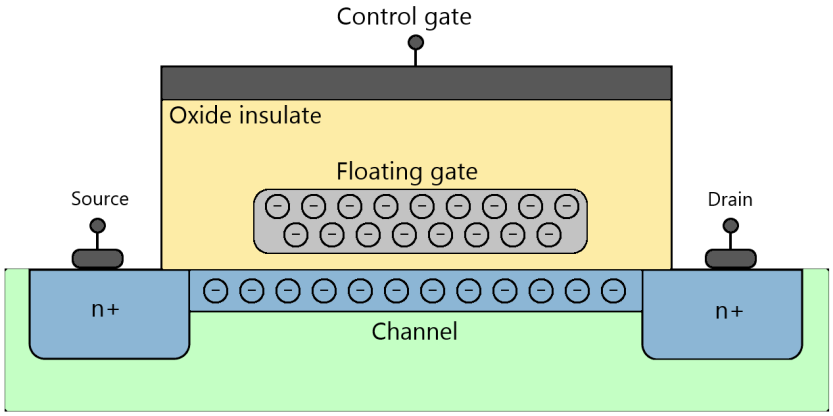


Figure 2.2: A floating gate transistor.

2.2.2 NAND & NOR flash

Flash memories are used for several different purposes. As such, they require a certain amount of tailoring to the specific system they are used in. The two main categories of flash memories are NAND and NOR flash, and they differ in how the floating gate transistors, hereby denoted as cells, are arranged.

NOR flash is organized on a byte or word basis, while NAND is organized on whole pages or blocks. NOR having a parallel architecture is the reason that random access is so fast for NOR flash. NOR can be further classified into several subcategories, but are not covered more in this thesis.

NAND is unlike NOR organized serially. See Figure 2.3. In NAND memory arrays several cells can be found between two special transistors, which are not used to store information, and two contacts. This makes random access slow due to the fact that no contacts are directly accessing memory cells. However, since there are fewer transistors per cell the effective cell size is significantly smaller than that of NOR, making NAND a more size efficient choice and thereby costing less per bit. NAND also has the advantage on a per-page basis, where writing to whole pages is significantly faster than NOR (0,4 μs/byte compared to 10 μs/byte). All things considered NOR is a good choice for storage requiring performance code storage and code execution, while NAND is better suited for larger and cheaper storage. [16, 4]

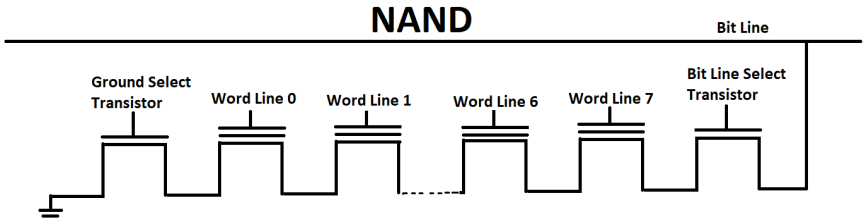


Figure 2.3: Rough schematic of a NAND array consisting of a number of floating gate transistors.

2.2.3 Erase-Before-Write Architecture for NAND memory arrays

Just as any other type of memory the flash EEPROMs have their own specific set of restrictions. Due to the structure of NAND memory arrays, see Section 2.2.2 and Figure 2.3, a single floating gate transistor cannot be solely reprogrammed. The specific size of each memory block is unique for each individual memory but usually stores hundreds of bytes of information. Because of this memory array design the whole memory block must be erased whenever a single memory cell needs to be reprogrammed.

This feature in NAND arrays eventually generates one of the main restrictions in flash memories. Namely the erase-before-write problem. As a consequence of needing to erase an entire block before being able to write to it a lifetime consisting of a number of erasure cycles are placed on the memory block. This is due to the fact that an erasure process damages the insulate in the floating gate transistor. Eventually this leads to the loss of the non-volatile capability of the memory block. The exact number of erase cycles a memory block can handle through a lifetime varies greatly depending on what cell coding the memory uses, see Section 2.2.5, but usually stays in the interval of a couple of thousands cycles to a hundred thousand cycles. [13]

The erasure process of a memory block is not as straight forward as one might initially think. Firstly a 0 V voltage is applied to the drain whilst a high voltage charge is placed on the source, see Figure 2.4. Any potential electrons in the floating gate are thereafter ejected via tunneling through the source gate. This process should erase all memory cells simultaneously. However, to make sure that no cells stay programmed, a read cycle is applied. If any cells are still detected as programmed a new erase cycle begins. This algorithm continuous until all cells in the memory block are erased. This algorithm is potentially very slow, at least compared to the write and read cycles, and typical erase times are between 100 ms to 1 s.

After the erase cycle is completed a write operation cycle may begin, see Figure 2.5. To program a memory cell the block source and drain are grounded whilst high voltage pulse is applied to the control gate of the memory cell to program. The memory cells with a positive voltage applied to their control gate will receive electrons from the channel to the floating gate through avalanche-injection. The remaining memory cells with no voltage applied to the control gate will remain empty. Depending on the NAND memory coding the specific charge in the floating gate whilst programming the cell may alter. [24, p. 651-652]

Through continuous ongoing research there are of course several techniques developed to halter the NAND flash degradation and hence extending the lifetime of the memory. A well designed Flash Translation Layer (FTL), see Section 2.2.6, will perform balanced wear leveling and garbage collecting algorithms that spread the load more equally to all memory blocks. Most memory producers also choose to include a technique called over-provisioning. In this technique a certain threshold of additional memory blocks are added to the memory. The extra blocks are said to become over-provisionized. An over-provisioned memory block cannot be reached by the user and can only be used by the internal algorithms of the memory controller, such as the FTL. In turn this technique extends the memory lifetime since the intensive writing of internal memory algorithms may now use these extra memory blocks. [12] Finally most memory producers include a Cyclic Redundancy Checksum (CRC) to counteract data corruption. This check becomes more important in the later stages in memory lifetime since worn out memory blocks generate bit errors at a higher rate. Eventually, even though the countermeasures described are present, the floating gate transistors in a memory block will degrade enough to yield bit errors too big for even a CRC algorithm to correct. When this happens a memory block is often denoted as a bad block in memory terminology and becomes unusable. [28, p. 1]

2.2.4 Reading from a NAND array

When reading from an individual cell in a NAND array a high voltage is applied to all the other cells in the array, also known as applying voltage to the word lines, see Figure 2.3. This makes them conduct electrons through their channel whether they have electrons trapped in their floating gate or not. Then, a different threshold voltage is applied to the specific cell that will be read. This voltage makes the cell conductive only if there are no or very few electrons in its floating gate. Since all cells are connected serially, if electrons are not able to flow through the cell, meaning it has electrons trapped, then no current will be measured at the bit line. This is interpreted as a logical 0 in single level cells. If however there is a current at the bitline, meaning there are no electrons at the floating gate, it is interpreted as a logical 1. [24, p. 651-654]

2.2.5 Single-Level Cell (SLC) & Multi-Level Cell (MLC)

A 0 or a 1 is read from a NAND flash cell by reading the output voltage of the floating gate. By increasing the voltage levels more than one bit can be stored in a single cell. If the voltage levels are four the cell can effectively store two bits, since the levels can represent 00, 01, 10 and 11. Therefore, every doubling of the number of voltage levels adds one bit of storage to the cell. The benefits of this is that more data can be stored on the same amount of cells without increasing process complexity. It does however increase the time for programming and reading the cells. Implementing these intervals also require precise signaling and sensing inside the memory so that bit errors don't occur which would make the memory cells prone to degradation and decrease endurance. [16]

As seen in Figure 2.6 we can see that by designating voltage intervals as specific bit values, the storage capabilities of a single NAND cell will be increased.

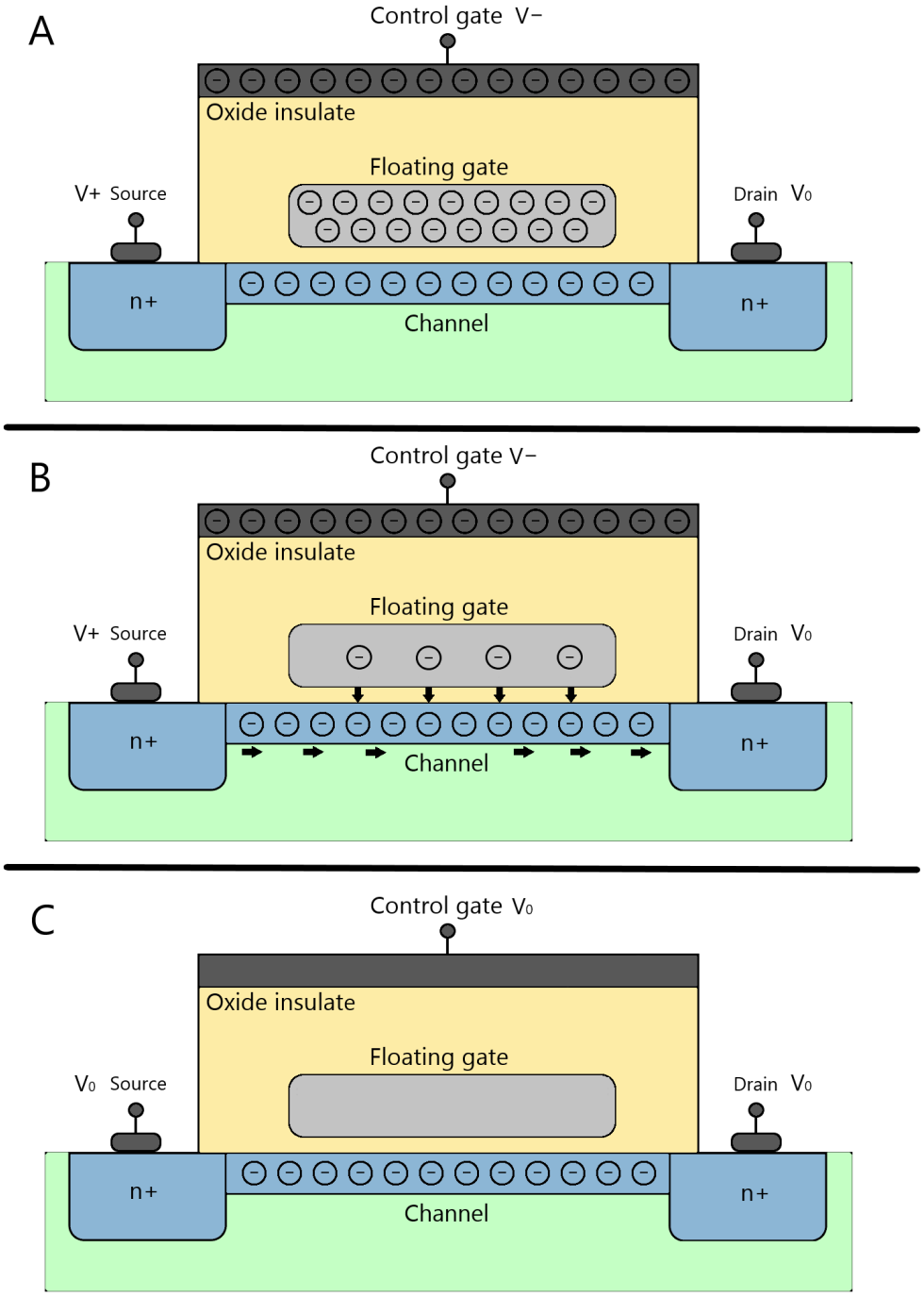


Figure 2.4: The erasure cycle of a NAND memory floating gate transistor.
A - A negative voltage is applied to the control gate, a positive voltage is applied to the source and the drain is grounded.
B - Electrons in the floating gate are ejected through tunneling.
C - The control gate and source are grounded and the floating gate remains empty.

2.2.6 Flash Translation Layer

FTL is a layer between the file system and the memory array. Since flash memories lack the capability to write to single pages inside a flash, something that is hard for the file system to manage, the responsibility of this is placed on the FTL. Doing so allows the file system to write to individual pages and the FTL handles the details of how this should be done by, for example, copying the entire block and writing it to a new place with the new page as well. The FTL also handles garbage collection, clustering of hot and cold data, and wear leveling. All essential tasks for the FTL to do efficiently.

Clustering of hot data is done for the purpose of optimizing garbage collection in flash. Since the garbage collector has to delete data blockwise it is optimal if the data contained inside all blocks is organized in a gradient from cold to hot. This way when the garbage collector needs to clear hot data that often becomes invalid, it does as few unnecessary erases of cold data as possible thereby reducing flash wear. [16]

Wear leveling

Wear leveling is performed in the FTL layer for the purpose of increasing the total lifespan of a flash memory by distributing the writes from frequently used to less used blocks. There are three main types of wear leveling, dynamic, static and global. Dynamic wear leveling makes update data being written to a block actually point to a new block, while marking the original one as invalid. This prevents repeated writes to the same logical address from wearing out the physical block connected to that address by switching the block that it points to. This kind of leveling is unable to fully wear level the memory since parts of memory is never updated, which makes static wear leveling necessary. Static wear leveling regularly transfers static blocks from one address to another, which allows most blocks to be used to their end of life.

Lastly global wear leveling ensures even wear between different chips/dies. Since both dynamic and static wear leveling are done on the local level it is possible that one die runs out of spare blocks and has to enter read only mode long before the other dies are worn out. This can be prevented by sharing spare blocks by employing a spare block manager. If a die depletes all its spare blocks and encounters an erase error, instead of entering read-only mode it can use a spare block from another die [29]. It is also possible to implement global wear leveling by reallocating zones that are frequently accessed in one die to another one. These types of wear leveling are implemented in different ways by different manufacturers. This can change wear behavior of memories with identical write patterns depending on the wear leveling implementation. For example, industrial grade SD cards and SSD memories usually have more advanced wear leveling algorithms.

Write Amplification Factor (WAF)

Due to numerous different factors the actual amount of data written to a flash memory is rarely exactly identical to the size of the information that is supposed to be saved in the memory. Metadata caching, wear leveling algorithms and garbage collection are all commonly present in a flash memory process and increase the amount of stored data during a

memory saving process. To avoid excessive memory degradation the difference between the supposed data amount and the actual data amount saved to memory needs to be small. Therefore the Write Amplification Factor (WAF) was developed as a unit to measure this important difference. In short, a WAF is a numerical value that references the difference between the size of the actual stored memory amount and the specific data amount that was to be saved to the memory.

As an example, if a memory process would have a Write Amplification Factor of 4 this would mean that the memory process stored four times as much information as the physical data that was supposed to be saved. Therefore a Write Amplification Factor as close to 1 is preferable. Then the memory process saves only the actual data to save with as little additional metadata as possible. Since the amount of data saved to a memory is directly correlated with its degradation rate a WAF value of 1 will lower any degradation compared to higher WAF values. [22]

Sequential I/O

Sequential writing and reading is important not only for its performance improvements over random writing. It is also necessary in maintaining a low WAF. This is because of the way writing is done to flash memories. As seen in Figure 2.7 certain writing patterns can increase the amount of writes disproportionately to its contents. If data from for example a file is written sequentially, when it is erased there is no need for garbage collection methods, since whole blocks are filled with data from the same file. However, if data were to be written non-sequentially, upon deleting said data it would require rewriting of large amounts of data that is contained within the same blocks, costing time and causing excess wear on the system.

2.3 Windows CE

Windows Compact Embedded is an operating system developed by Microsoft designed for small memory devices or embedded systems that require a minimum size. Although different from the everyday Windows OS it is still based on a subset of the standard Win32 API. [3, p. 4] Since Windows CE is a modular component-based operating system it provides flexible but reliable functionality needed for specific embedded system. Hence a developer may choose to include only relevant parts of the operating system for a certain type of embedded system, making it as small as possible. [11, p. 1-3]

2.3.1 Windows CE Storage Stack

The Windows CE Storage Stack is a vital part of the CE OS kernel that handles file management and I/O controls. It contains multiple different levels of collaborative modules that create and translates OS files into comprehensible I/O operations for the connected storage device. These operations will eventually be used by the storage device FTL to physically store the actual data. See Figure 2.8 for a visual implementation of the levels in the storage stack with a connected NAND memory.

File system manager

In Windows CE the file system, and all file related APIs, are managed by the ***FileSys.dll*** module. This is also the module that implements the objects store and storage manager. It will unify all file systems into a single system under one common root '\'. Every file is now identified with a unique path from the root in a hierarchical tree. [18, p.164, 187].

The ***FileSys.dll*** module contains different modules of its own. Firstly the Object Store module handles a memory heap which contains the RAM system registry together with the RAM file system. [18, p. 77, 82] There is also the ROM File System being another module handled by ***FileSys.dll*** that is connected to the unified file system under '\Windows'. All files in the ROM are read-only files. [18, p. 192] Finally there is the Storage Manager module which has four primary responsibilities. [18, p. 77, 149]

These are

- **Storage drivers** - Device drivers for physical storage mediums, also known as block drivers since they provide access to randomly addressable blocks of data storage. [18, p. 146-149]
- **Partition Drivers** - Translates the storage driver. Exposes the same interface as a storage driver and translates block addresses for a partition into the true address of the block on the storage device. It then finally passes the call to the storage driver. [18, p. 148-149, 189]
- **File system drivers** - Drivers that organize the data on a storage device as files and folders. [18, p. 138-140]
- **File system filters** - Process calls for a file system before the file system gets them. This allows for specialized handling of file access for data encryption and compression. [18, p. p. 190-191]

File cache

A file cache is, as the name implies, a cache for files. The primary purpose of this cache is not only to reduce the amount of writes being done to disk memory, but also to improve I/O performance. The file cache works as follows: any reads or writes to disk is first written to the cache, which is flushed at regular intervals determined by the cache manager, or when large I/O operations are performed. Any potential read or write requests to the same area of memory can when data is stored in cache be handled by accessing cache memory instead of disk memory. Since cache is limited in size and files larger than the cache can not be stored there, the effectiveness of the file cache is largely dependent on the size of the files being accessed and created. [26] Another function of file cache is to temporarily store files before being written to disk. Using file cache as a primary storage before writing to disk allows the file system to optimize the writing operation, usually being able to send fewer commands and reducing overhead.

File caching also acts as a regular cache, where several writes to the same region of memory can be stored in local cache, reducing the amount of data traveling over the bus and

reducing the wear of the memory cells repeated erase/write cycles would cause. This in turn can improve I/O performance, as the bus is free to handle other requests. File caching can also reduce latency usually caused by disk accesses by temporarily storing frequently accessed data in cache.

The module cachefilt.dll contains the settings for file caching implemented as a file system filter. All file caching is implemented as a file system filter (file system caching manager). It works with any file system and does not require changes to the file system. [18, p. 190-191]

File caching also improves I/O performance by rearranging certain random I/O operations to be sequential. Sequential operations can be performed quicker by the system than by random access writes and require fewer block erasures[10, p. 98-99].

Disk cache for Metadata (Metadata, logical level)

Disk cache in this context is responsible for caching metadata. That is, for example, file names, file size, access rights, memory location etc. It is also known as system cache, cache buffer, disk buffer or just cache.

One of its purposes is write acceleration. Upon data being sent from the system to disk, the disk cache may signal the system that the writing operation is complete. This is somewhat dangerous, as loss of power during this period may cause data integrity to be lost. The boon is, however, that it frees the system from waiting for the I/O operation to complete, thereby increasing I/O performance significantly. Disk cache also allows for metadata to be stored in the cache, which improves execution time for operations requiring this metadata. Lastly, since disk cache allows metadata to be stored in cache, repeated erase/write operations to the same meta data will not cause unnecessary wear on the flash cells. This is because these operations are first performed on the meta data in the cache. Then the data on disk only needs to be updated once.

Disk caching in Windows CE is implemented as an auxiliary library. The file system driver must use this library somewhere in its implementation to be able to use this service. The disk caching is used to cache file system metadata. TFAT and exFAT file system can be configured to use disk caching.

TeXFAT architecture

The TexFAT file system is a file storage system with emphasis on transaction safety and is an extension from the original FAT file system. It uses a file allocation table at the beginning of every volume. The OS can use the stored tables to traverse the cluster chains quickly and easily, since the table shows the beginning and end of every cluster in the section. TexFAT is an improvement over the original FAT file system that corrupts data when loss of power occurs during a file system operation. TexFAT, however, maintains data integrity during such occurrences. It does this by making its file system update functions atomic by utilizing two FATs, one as a primary FAT and one as a buffer. When a update is to be performed the primary FAT is not directly written to. First the buffer FAT receives the update and if loss

of power occurs during this time the main FAT is not corrupted by being interrupted in the middle of an operation. If the update is done as planned to the buffer FAT it is then copied to the main FAT, completing the transaction. [14] [27] By default only folder modifications and FAT updates are protected by using TexFAT, but by changing registers normal file data updates can also be made transaction-safe [25].
TexFAT is compatible with both Windows, Mac OS and Linux, therefore making it used in many systems including Windows C.E 6.0.

Storage Device Driver

The storage device driver is a special kind of software that handles the communication between the OS and connected hardware. In most operating systems, including Windows CE, there exists multiple different storage device drivers for different types of storage devices. These types of drivers are often the final layer in the OS and communicate directly to a connected storage device or to its FTL.

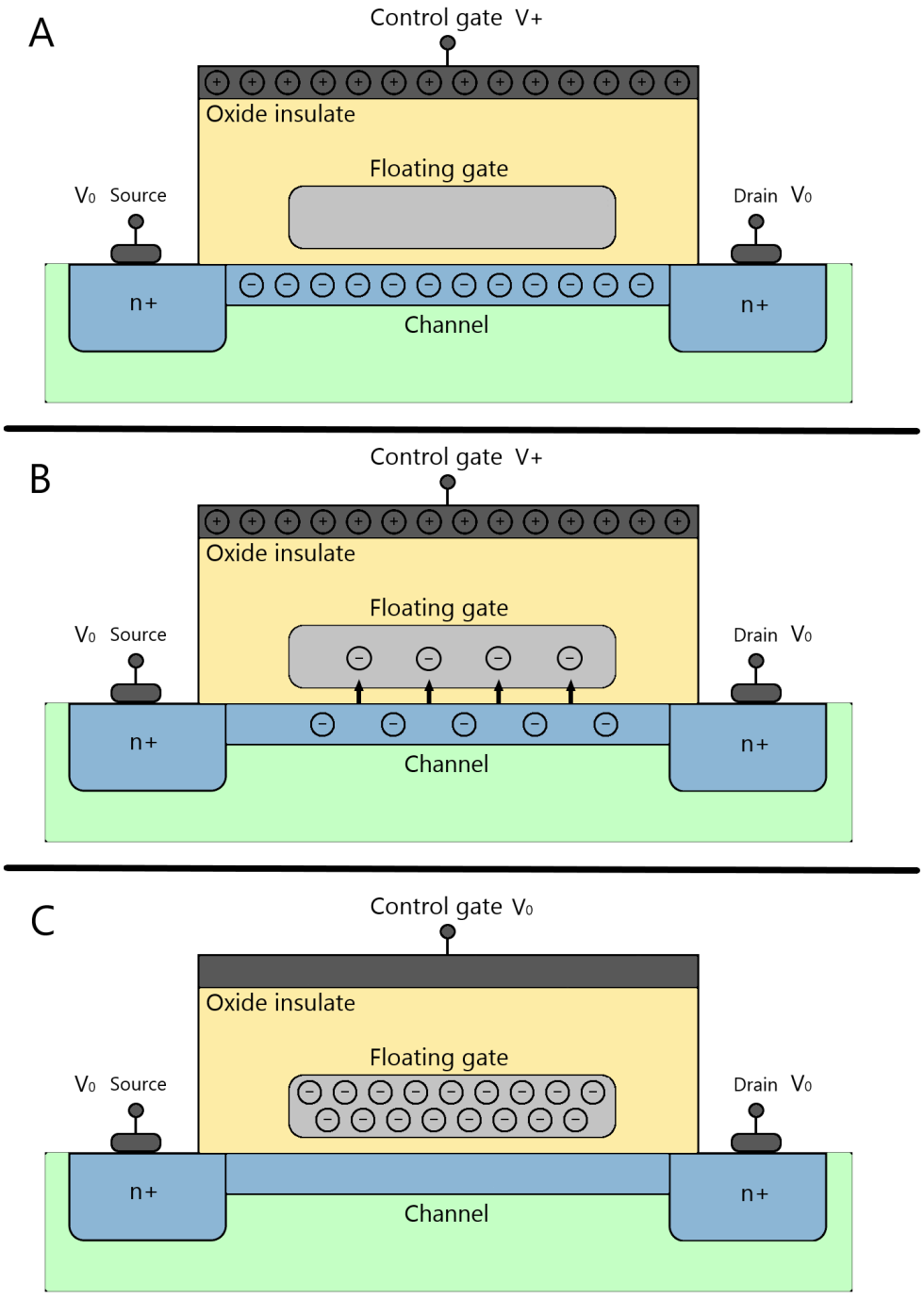


Figure 2.5: The write cycle of a NAND memory floating gate transistor.
A - A positive voltage is applied to the control gate, source and drain are grounded.
B - Electrons in the channel enter the floating gate through avalanche-injection.
C - The control gate is grounded and the electrons trapped in the floating gate remain.

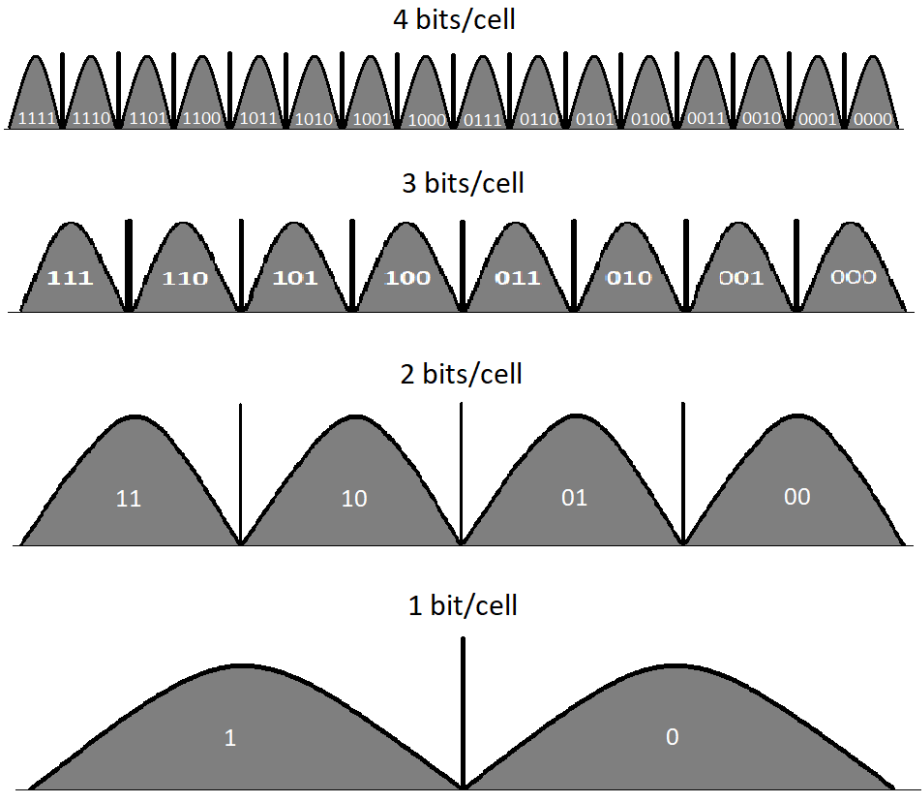


Figure 2.6: Representation of how doubling the voltage levels measures more bits are able to be stored in a cell.

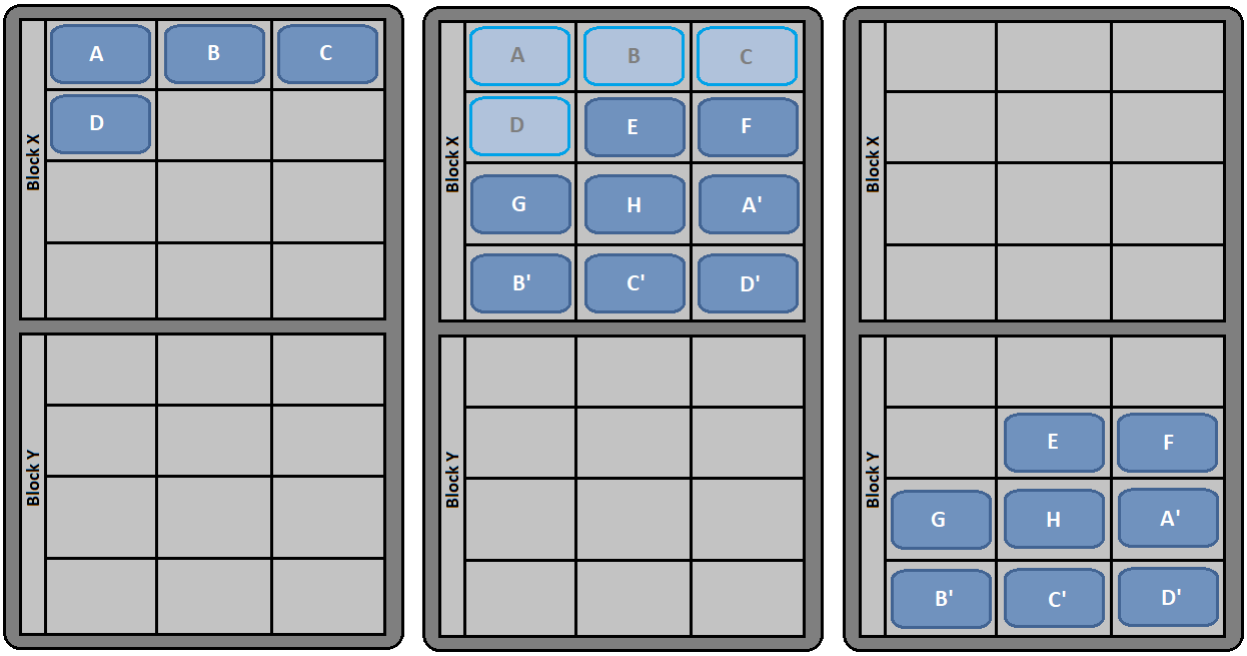


Figure 2.7: Garbage collecting causing write amplification.
Left figure: Pages A-D are written to block X,
Middle figure: Pages E-H are also written to block X, four replacement pages A'-D' are written to block X as well. The original pages A-D are stale, invalid data that cannot be overwritten until the whole block is erased.
Right figure: To remove stale data pages E-H and A'-D' are read and written to block Y, allowing block x to be erased. This is garbage collecting.

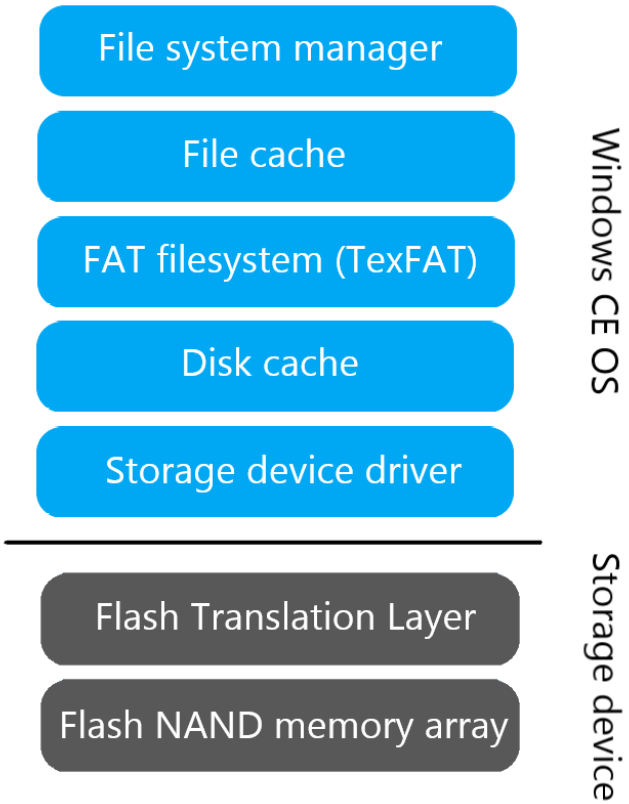


Figure 2.8: The storage stack for Windows CE with a connected NAND flash storage device.

Chapter 3
Previous research & methodology

3.1 Previous research

The topics of flash memory degradation, floating gate transistors and memories in a NAND-flash context are broad and well established subjects that all have a substantial existing research base. Although no previous work exists in the niche context of flash memory degradation in Beijer Electronic's HMI panels there certainly exists more general research relating to the concepts of this thesis. Much of current research regarding flash memories revolves around improving the flash memories themselves, more specifically topics such as cold-hot grouping, spatial locality or learned indexes [23] designed to improve performance.

According to Detlev, important factors in developing flash memory architectures are the cost per bit, scalability, power efficiency and performance values. Performance values range from random read/write latency to sequential read/write data bandwidth to random data throughput. These however are not the main focus for this thesis, but may instead be regarded as parameters that could be affected by potential changes to a flash memory system. The book also mentions concepts for improving the read durability for NAND memories.[2] One of these concepts involve increasing flash read durability by continually injecting write/erase cycles between intervals of reads. It generally succeeds in extending read durability, but at the cost of a large overhead. Another, more closely related concept, is the concept of wear leveling and the FTL as described in Section 2.2.6. Detlev described and compared two methods of wear leveling, static and dynamic, and found them both to be severely lacking, citing that this form is best fitted for small, cheap storage devices.

Research has also been conducted in replacing the traditional FTL wear leveling algorithm. In 2011, Boukhobza et al. proposed replacing the wear leveling algorithm and garbage collection mechanism completely with a two-tiered flash solution instead. They proposed that the FTL could be replaced with a full cache-based solution. [9] Although this does not appear to

have caught on yet.

In the context of garbage collection, Wang et al. produced in 2020 a hybrid mapping memory block algorithm to lower the negative WAF effect of modern garbage collectors. Together with the hybrid mapping algorithm they redeveloped a greedy garbage collector that successfully performed memory simulations which conducted an improvement in WAF values. [22] Although garbage collectors are not the main focus in this thesis, they are of course still highly relevant in discussing any results yielded from memory testing.

Meza et al. conducted in 2015 a large-scale study of flash memory degradation in SSDs using data collected from Facebook's data centers. The study took place over four years consisting of millions of operational hours. The results showed that the SSD flash went through several failure periods where that memory wore out and lifetime strictly corresponded to the total amount of data written to the SSD flash memory. Their results also concluded that the amount of written data executed by the system software or OS may, due to system-level buffering and wear reduction techniques, inflate the total amount of data actually written to the flash memory. [15] Even though this research was performed on SSDs the structural semantics are close to identical compared to SD cards and hence this research is deemed relevant enough for this thesis.

Many reports have found that the most straightforward way of improving the endurance of flash memories is by the process of over-provisioning. Over-provisioning in this context means that the memory inside the device is larger than the logical addresses supplied to the user, even though the entire memory is actually used. Hence the additional over-provisioned memory blocks are accessible only by the memory controller and can never be used directly by the user. [19] Most flash memory producers select a specific percentage of the total memory blocks to be over-provisioned. Although over-provisioning flash memories is a common technique to extend the memory lifetime it is not something that may be easily changed once the production of the memory has been finished.

Other research challenges are currently being dealt with. According to Frontiers of Quality Electronic Design current challenges for flash memories include lifetime improvements, which in turn includes write-reduction techniques. The popular techniques do not, however, deal with caching, but instead deal with flip-and-write and two-stage writes. [20, p. 137-165]

Due to the lack of existing proper research into how caching affects flash memory endurance, when collecting data for this masters thesis one may conclude that such research has not yet been done, or at least has not been widely published. As discussed in this section the research in context of flash memory endurance is vast. However most of this research seems to include FTL, over provisioning, garbage collecting or memory mapping as their main analysis but not caching, perhaps considering it to be an external issue.

3.2 Methods

To conduct the analysis of this master thesis data indicating the wear of flash memory placed in Beijer Electronic's HMI panels needed to be gathered. This was achieved by explicitly designing a test application written in `C++` running on Windows CE which could be executed on the HMI panels. The purpose of the application was to allow for a program to continuously write a number of files of specific size to the flash memory, wearing it down in the process. In the remaining parts of this thesis this memory degrading process will be denoted as a test or test execution.

Both the front-end and back-end of the test application were developed where the front-end consisted of a GUI created in the .NET framework. The GUI was used to show the current status of the test as well as presenting interactive buttons to start or stop the current execution. The back-end consisted of all file handling and the structure of the test itself meaning the number and size of the files to write. Due to the fact that flash memory, in general, takes an arbitrarily long period of time to wear the test application was designed to allow its user to conduct and design tests that could be executed indefinitely. See Figure A.1 for a visual representation of the `C++` application in question.

A third-party program, called Tera term, was used for all log files. This program could read data from a computer port and display and save the output as well as timestamp any input. It was used by connecting HMI panels running the previously mentioned test to a USB-port on a PC. Tera term then logged, timestamped and saved any output produced from any panel individually.

In addition to the test application a `C#` data analysis application was created. This application also consisted of a graphical .NET GUI where the user could load specific log data files from each individual test conducted on a HMI panel. Using the test log files the application could calculate and display exactly what logical memory addresses were used in the test. The application also used a Matlab API to execute two different Matlab scripts that were used for all data plots in this thesis. The purpose of this application was not to execute it on the HMI panels but only run it on personal computers. Hence the application was designed in the context of the Windows OS and not Windows Compact Embedded. In Figure A.2 a snapshot of the `C#` application can be seen.

Finally another third party application, Hard Disk Sentinel, was used to collect health information of the SD cards used in the tests. The health information consisted of multiple useful data points. However, only a subset of these data points were used in the analysis of this masters thesis. Namely the expected remaining lifetime percentage of the SD card, the number of spare and bad memory blocks, the total erase count and finally the CRC error count.

3.2.1 HMI Panels at Beijer Electronics

All technical testing conducted for this masters thesis was executed on four individual HMI panels of the same model. This is to assure that any variation in the results of executed tests

are solely due to the structure of the test itself and not due to any variation between HMI panel models. The specific HMI panel used is Beijer Electronics **X2 pro 7** model. The panel operates on the Windows CE Embedded 8.0 operating system and have 2GB SSD eMMC flash memory storage and a slot for an external SD memory card. [7] See Figure 2.1 for a visual picture of the panel.

3.2.2 HMI OS

The Windows CE operating system image that the panels were using could be viewed to extract details about the file system and surrounding systems. These could also be edited to change the behavior of the system, thereby affecting the memory differently. Through specific changes to the OS image the file and disk cache could be turned off or on for each individual panel used in the tests. For disk cache this was explicitly toggled by altering a flag value in the operating system code. File cache however was turned on and off by changing specific values in the Windows Registry together with explicit flag values in the file I/O code.

All data handled by the panels application was redirected to an external SD card to allow for more precise data to be recorded than with the HMI:s built-in memory. Writing to an external SD card also avoided degrading the soldered eMMC memory which is significantly more expensive. The panel OS image also included the SD card drivers of Windows CE. These were altered to allow for logging of all memory requests sent over the SD card bus. The requests consisted of four vital data points. First the specific SD command which described if the request was a read from or write to memory request. Secondly the logical memory block address to read from or write to. Thirdly the total amount of blocks the request handled starting from the address in the second data point and counting forward. The last data point consisted of the specific block size expressed in bytes. Therefore the collected log files could be used to calculate exactly what logical memory addresses that were used in a test and how many bytes were read from or written to each used address. Using this information the total data amount read from and written to the SD card could be calculated.

After any specific alterations where made to the Windows CE OS code it needed to be flashed onto the HMI panel hardware to update its software. This would eventually make the effects of any OS code changes present in the panel. All code changes were done by using the Windows Visual Studios IDE. This application was also used to build the OS code which produced the OS image files that would be flashed to the hardware. For the flashing process two in-house applications were used. Firstly an application called Trinity was used to connect a computer to the HMI panel using an Ethernet cable between the two devices. Secondly an application, named Image Loader Production, would then be used to load the image files and flash them onto the panel that was connected through the Trinity application. After a quick reboot of the HMI panel it would now run on the newly flashed OS image.

3.2.3 Synopsis

A `C++` file writing application was executed on four different HMI panels, all with specific individual configurations and ATP SD cards, to stress test the SD card memories. The file size and count was explicitly set in this application. During an ongoing test execution, due to

modifications made in the SD card drivers, all relevant SD memory bus requests were logged, describing what logical memory addresses were read from or written to along with the total byte amount this request handled. After or during a test execution the external SD memory cards were physically removed from the HMI panel and placed into a personal computer. Here a third party application was used to display relevant health information of the SD memory card. All test logs files were also gathered and analyzed using a *C#* application which calculated and displayed the total amount of data read from and written to the SD card for the specific test execution.

3.3 Test Configurations

In this masters thesis the experimental segment is divided into two different types of test executions. The first type of tests were quickly performed and focused on analyzing the difference between file sizes and cache settings. The second type of tests were long time running where the focus was to analyze the difference of long term degrading depending on cache settings and available space of the SD cards. These tests continuously executed until the SD card was considered consumed. From here on the first type of tests will be referred to as short-term tests and the second test type as long-term tests. Both types are described more thoroughly below.

3.3.1 Short-term tests

Tests monitoring bus commands to SD cards were performed. These tests could be done relatively quickly and without needing brand new SD cards to monitor health status. From these tests information such as total data written to disk, FAT table accesses, WAF and execution time could be gathered.

To monitor the writing patterns of the panels parts of the OS drivers were changed to produce output to an output port on the back of the panel. The output consisted of the SD SPI command denoting what type of request was made, a logical address, a block size showing the size of blocks read, and lastly the number of blocks. The logical address denotes where data started being read from or written to depending on what type of command was used. Four specific commands were monitored, single block read, multiple blocks read, single block written and multiple blocks written. The output ports of the panels were monitored through the duration of the test, logged and timestamped. These logs could later be analysed.

After the tests were completed the theoretical lowest amount of data needing to be written could be compared against the actual data being sent over the bus to the SD card. A WAF could then be calculated based on the ratio between the two amounts. The WAF would then indicate the efficiency of the writing pattern and the file/disk caches.

To discern the effects of different cache configurations four short term tests were performed, all with different permutations of file cache and disk cache enabled/disabled. This was done by editing the system software registers responsible for enabling these functionalities. Disk caching was also disabled in the application layer itself to prevent any additional appli-

cation buffers from acting as a local cache. This was done by setting write flags in `C++` code, namely `FILE_FLAG_WRITE_THROUGH` and `FILE_FLAG_NO_BUFFERING` as arguments in the `createfile` function. In the applications where application caching was not turned off these flags were instead set to `FILE_ATTRIBUTE_NORMAL` and `FILE_FLAG_OVERLAPPED`.

3.3.2 Long-term tests

Four long-term tests were also conducted on the HMI:s. These four tests were the results of enabling and disabling disk caching and file caching on the panels.

To show different effects of the SD card built in FTL algorithm two of the four SD cards were filled to 90% capacity. The writing patterns were such that the entire memory was filled and then deleted with a fixed file size of 75 MB. In the case of the cards being filled to 90% capacity initially only 10% of the memory was repeatedly used.

Further data was gathered by using a third party software called Hard Disk Sentinel, which for a supported SD card can display health information. This health information includes, depending on the specific card, viable data points such as the current used lifetime of the card, the number of blocks that have been worn out and remaining spare blocks. The SD cards used were chosen partly because of their compatibility with this program. Health information such as bad block count were logged before and after large writing operations to give an estimation of real wear on the SD cards.

The reason the writing operations were completed on external SD cards was to not place excessive wear on the built in eMMC flash drive. The SD card chosen was selected for its industrial rating, flash architecture (MLC), health monitoring capabilities, low storage capability (8 GB) and low total erase cycle capability. A low total erase count and storage capability will let blocks wear out relatively quickly, which could then be used as a way of measuring real wear on SD cards.

Different manufacturers employ different wear-leveling techniques, and this is usually considered an industry secret. Therefore, knowing the inner workings of SD cards is usually not feasible. The chosen SD cards were manufactured by ATP, who like most companies do not provide exact information regarding their wear-leveling techniques. However, they do advertise their cards with "advanced wear leveling algorithm", which is then explained to be similar to static wear leveling with the exception that it includes the whole flash device instead of a single die. [4] The ATP SD cards can therefore be considered to be using a form of global wear leveling.

Chapter 4
Results

4.1 Short term tests

4.1.1 Panel Configuration

The configuration settings for all panels during the short-term tests can be seen in Table 4.1.

	File Cache On	File Cache Off
Disk Cache On	Panel A	Panel B
Disk Cache Off	Panel C	Panel D

Table 4.1: Configuration settings for panels A-D on short-term tests.

4.1.2 Main Results

Results of panels running short-term tests can be seen in tables 4.2, 4.3, 4.4 and Table 4.5, please note unit differences between Data Written (GB) and Data Read (MB).

The data point of table access reads denotes percentage of total memory amount read from the FAT table. This is also the case for table access writes. Data written and read is the sum

of data sent by write and read commands over the bus to the SD card, calculated by the *C#* application described in Section 3.2 using the data in all individual log files saved by the panels during test execution. The write amplification factor was calculated by dividing the data written with the theoretical minimum amount needed to write the files to memory. Total data written from the test application in these tests was 10 GB respectively and was therefore deemed as being the minimum theoretical amount needed to write the content of each test to memory. The execution time was calculated by the `C++` application executing all tests.

Test 1 - 10 KB						
Panel ID	Data Written (GB)	Data Read (MB)	Table Access Reads (%)	Table Access Writes (%)	Write amplification factor	Execution Time (H)
Panel A	87.3	120	0	86	8.70	23.25
Panel B	146.5	40960	0	43	14.65	126.35
Panel C	21.5	5	22	43	2.15	50.10
Panel D	23.5	4610	0	37	2.35	131.15

Table 4.2: 10 KB file size test results.

Test 2 - 100 KB						
Panel ID	Data Written (GB)	Data Read (MB)	Table Access Reads (%)	Table Access Writes (%)	Write amplification factor	Execution Time (H)
Panel A	20.6	16	1	50	2.06	3.75
Panel B	26.9	6963	0	36	2.69	20.58
Panel C	11.5	1	46	11	1.15	19.28
Panel D	11.7	462	0	10	1.17	28.13

Table 4.3: 100 KB file size test results.

4.1.3 Sequentiality

As described in Section 2.2.6, sequential writing patterns are important to decrease unnecessary garbage collection and therefore premature wear on flash memory. It is also important to reduce execution time even on flash technology. Through analysis of test logs the average amount of sequential data that was written before a break appeared was calculated and logged in Figure 4.6. As can be seen in Table 4.6, sequentiality varied vastly depending both on file size and cache configuration.

Test 3 - 1000 KB						
Panel ID	Data Written (GB)	Data Read (MB)	Table Access Reads (%)	Table Access Writes (%)	Write amplification factor	Execution Time (H)
Panel A	12.2	0	0	17	1.22	1.55
Panel B	12.2	737	0	12	1.22	1.71
Panel C	10.3	1	100	3	1.03	2.38
Panel D	10.2	47	3	2	1.02	3.88

Table 4.4: 1000 KB file size test results.

Test 4 - 10000 KB						
Panel ID	Data Written (GB)	Data Read (MB)	Table Access Reads (%)	Table Access Writes (%)	Write amplification factor	Execution Time (H)
Panel A	10.79	0.2	100	7	1.079	1.28
Panel B	10.23	65	0	1	1.023	1.78
Panel C	10.10	0.6	100	1	1.010	3.33
Panel D	10.03	5	23	0	1.003	1.69

Table 4.5: 10000 KB file size test results.

Panel ID / Test File Size	10 KB	100 KB	1 MB	10 MB
Panel A	86 KB	103 KB	76 KB	71 KB
Panel B	10 KB	18 KB	68 KB	572 KB
Panel C	21 KB	57 KB	64 KB	66 KB
Panel D	1 KB	6 KB	54 KB	557 KB

Table 4.6: Sequentiality results for short-term tests.

4.2 Long-term tests

4.2.1 Configuration

The cache configurations for the panels performing the long-term tests varied from those in the short-term tests. Please see Table 4.7 for all panel configurations in the long-term tests.

4.2.2 Main Results

The results of the long-term test execution can be seen in Table 4.8. All data points were calculated after each panel had finished its execution, meaning until an average total value

	90% full	0% full
File Cache On & Disk Cache Off	Panel C	Panel A
File Cache Off & Disk Cache On	Panel D	Panel B

Table 4.7: Configuration settings for panels A-D, long-term tests.

of 3000 block erase count was achieved and the remaining SD card lifetime then was to be considered as 0 %.

Just as for the short-term tests the data points of data written and read were calculated by using the previously explained `C#` application and all individual panel log files. The amount of files written was logged by the `C++` application that executed the tests on the panel. The write amplification factor was calculated by multiplying the total number of files written with the file size (75 MB). This value was considered as the lowest possible theoretical data amount needed store all written files in memory. This value was then divided by the actual written data amount, as seen in the first column, to finally gain the WAF value. Execution time was constantly monitored by the test `C++` application. Finally the average speed was simply calculated by dividing the total written data amount by the total execution time.

Long-term tests							
Panel ID	Data Written (GB)	Data Read (MB)	Files Written	Write amplification factor	Execution Time (Days)	Average Speed (MB/S)	Files written per hour
Panel A	9477.49	1160	122710	1.0301	39.5215	2.7755	129.4
Panel B	9601.71	1847	124233	1.0305	39.3153	2.8267	131.7
Panel C	8458.73	792	109447	1.0305	41.2181	2.3752	110.7
Panel D	8058.04	1991	104281	1.0303	39.3153	2.3722	110.6

Table 4.8: Results for the long-term tests.

4.2.3 Spare Block Usage

One of the data points gathered from the health information in Hard Disk Sentinel was the spare block count. This value shows how many over-provisioned memory blocks, see Section 2.2.3, were available and not already in use by any internal mechanisms of the card. The more spare blocks in use the lower this value would be. The initial spare block count for each card was 102. See Table 4.9 for the average free spare block counts and Figure 4.1 for a plot of the spare block count throughout the long-term test executions.

Panel ID	Average Spare Block Count
Panel A	82.9130
Panel B	77.4783
Panel C	82.0435
Panel D	71.3913

Table 4.9: The average free spare block count for Panel A-D.

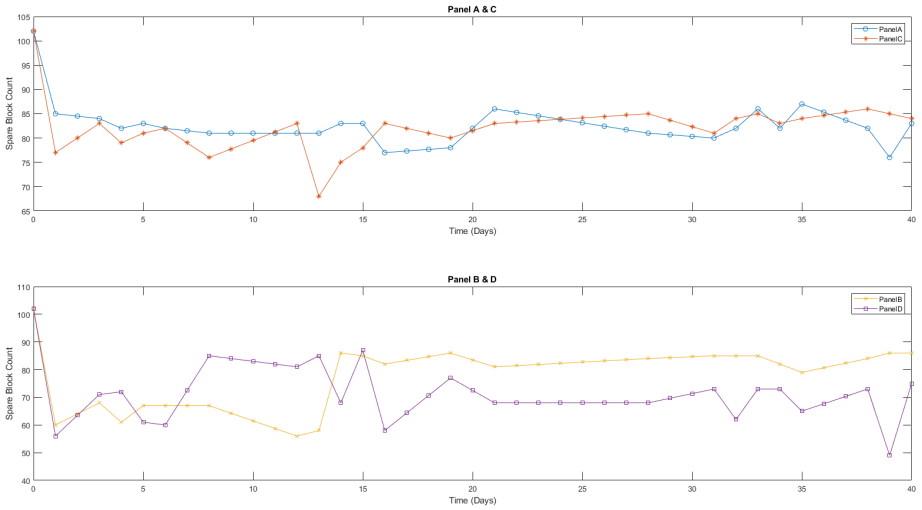


Figure 4.1: Spare block count for all panels during the long-term tests.

4.2.4 Logical address usage for the SD bus

Because of the panel OS changes previously described in Section 3.2 the panels' operating system could log all traffic sent through the SD card bus. This traffic was then monitored through individual panel log files. The `C#` application, as seen in A.2, deciphered these log files and translated them to display what amount of data that was written to and from what addresses. The translated data was then used in a `Matlab` script to create the plots seen in Figure 4.2. These plots show the addresses used by the SD card bus in the context of linear time for a 24 hours testing session. As can be seen there is a distinct difference between panels A and C, where both SD cards were 90% full; and panels B and D that had SD cards which were completely empty. The difference is clearly seen by noticing the amount of different addresses used by the bus for the different panels. This is most easily seen by looking at Y-axis values for the plots of panels A and C and then compare it to the same values in the plots of panels B and D.

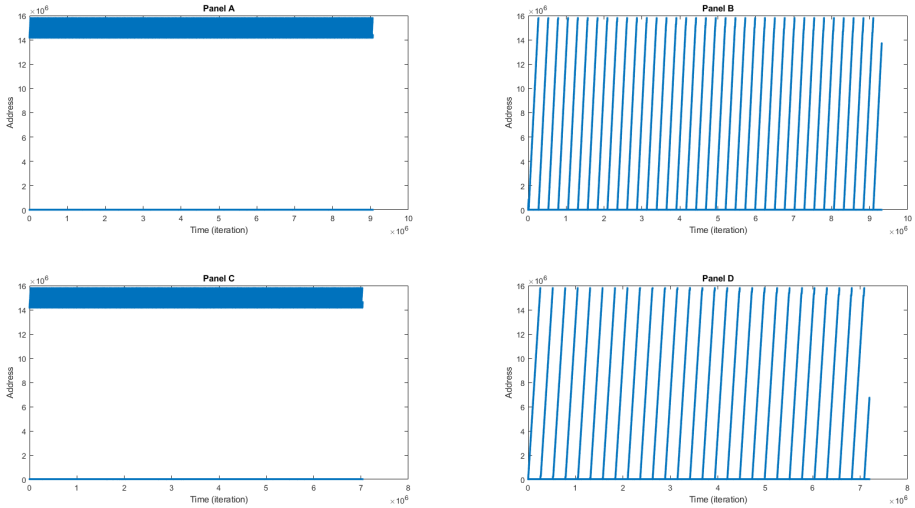


Figure 4.2: Sequential plots for Panel A-D for a 24 hour testing session.

4.2.5 Bad block counts

Throughout the long-term test executions the SD card bad block count can be used as a clear indication of flash memory wear. This data point was expected to increase in all SD cards when the expected lifetime threshold was approaching a value of zero. However after the long-term test finished only the SD card in panel C had a single bad block. The remaining cards in panel A, B and D had no bad blocks. The data retrieval of bad block count was retrieved by using the third party software Hard Disk Sentinel.

Chapter 5
Discussion

5.1 Short-term tests

5.1.1 Write amplification factor

WAF as a result of file size

First thing of note when analysing the WAF of the short-term tests is that it significantly decreases as file size grows. This can be attributed to the overhead associated with creating a new file, since creating a new file requires the OS to not only allocate an available section of memory, but also creates necessary metadata. With this information in hand it is fairly evident that even smaller files would have a larger WAF for all panel configurations and larger files would approach a WAF of 1. Therefore it is not the individual magnitude of the WAF that is important, but rather the relative size to other panel configurations.

WAF as a result of caching configurations

For file sizes below 1000 KB configuration B performed the worst in terms of WAF. It is also noteworthy that the same configuration also had the most data read from memory for all file sizes by a factor greater than nine. Panel configuration A, which also had file caching enabled and performed worst for file sizes above 1000 KB, also resulted in a higher WAF for all four tests. But it appears that enabling file caching lowered the impact of the enabled disk cache for smaller file sizes.

The panel configurations with the lowest WAF were panel C and D depending on file size, but not significantly differing except for test 1, where file size was the smallest.

Panel B had the highest WAF for test 1, which upon examination of the log files was the result of several repeated writes to the same address coupled with repeated writes to the FAT

table. The reason for the repeated writes to the same address is not known, although it was not observed in any other panel configuration.

5.1.2 Execution time

Although not the primary focus of the paper, execution time is an important aspect to consider. A cache configuration with low WAF can be deemed infeasible if the execution time is insufficient for the tasks required of the system.

For test 1 the lowest execution time was panel A, 23.25 hours. This was, however, not the most effective in terms of WAF. Although, if faster execution times are required than the ones provided by different panel configurations B or C, panel configuration A could be a reasonable option. Panel D performed consistently worst, except for test 4 where both panel B and A performed worse. This indicates that for smaller file sizes up to 1 MB both file and disk caching used individually improve I/O performance. However, this trend seems to shift for larger file sizes and it can therefore not be determined that this applies for larger file sizes as well.

Execution time was consistently faster for panel B than for panel D up until test 4, indicating that the disk cache was responsible for improving I/O performance for smaller file sizes. This is even more remarkable considering that the WAF for panel B for test 1 was 7 times larger than that of panel D. A reason for the improved execution time coupled with the high WAF for panel A could be write acceleration. As described in Section 2.3.1 disk caches can use write acceleration to improve I/O performance, which would explain the fast execution times despite a higher data amount being written. Using this configuration for small repeated writes would undeniably cause unnecessary wear on the SD card, however as seen for both larger file sizes in the short-term tests as well as in the long-term tests, the write amplification is significantly lower for larger sizes.

5.1.3 Data read

In all four tests panel B read a significant amount of data from disk which decreased as file size grew. This may be the result of the file cache needing a large amount of metadata to correctly write files from cache to disk, but the lack of disk cache meant that data needed to be read from disk every time. Panel configuration D corroborates this hypothesis, since disabling both file and disk cache causes significantly less data to be read than file cache on and disk cache off (panel B).

5.1.4 Table access reads & writes

Table access reads and writes measure how many of the reads and writes were done to the FAT table on disk as a percentage of all reads and writes. As can be seen in all four short-term tests for only panel configuration C was there a significant percentage of reads done from the file allocation table. And in all tests the data read from panel C was almost non-existent. This

may be due to the fact that the file allocation table did not need updating as frequently since a panel with file cache enabled sends fewer write commands in total.

5.1.5 Sequentiality

As seen in Table 4.6, the sequentiality of an application can vary greatly. Generally, the table shows a correlation between large file sizes and a greater sequentiality in the program, with the exception of panel A. The reason for this may be the increased writing speed and amount caused by enabling file and disk cache that facilitates the need to make frequent updates to the file allocation table.

5.1.6 Limitations & unknowns

These short-term tests analysed cache configurations for only four different file sizes. As can be seen in the execution time for panel B for the different file sizes, the disk cache seems to have an adverse effect on performance for the 10 and 100 KB tests. However, for sizes of 1000 and 10000 KB these effects diminish relative to different configurations. Without further increasing file size this trend can not be asserted for larger file sizes.

The commands logged were because of time constraints limited to commands strictly related to writing or reading data. Because of this certain nuances of the interaction between panel and SD-card may have been lost.

Utilising different writing patterns might also have revealed situations where certain unexpected configurations would perform better. However, because of time constraints only one writing pattern was examined.

5.2 Long-term tests

5.2.1 WAF

As seen in Table 4.8, the monitored WAF hardly differed between the different configurations. This is most likely due to the small amount of overhead that exists relative to the file sizes, which would result in WAF's close to 1 with only minute variations. However, due to other factors discussed below there may be an actual WAF significantly larger than the one recorded.

5.2.2 Over-provisioning usage for card space

As can be seen in Table 4.9 and in Figure 4.1 there seems to exist a correlation between a panel only having disk cache on and a higher usage of spare blocks. The average value for panel B, 77.4783, and panel D, 71.3913, shows that there exists a roughly 5% and 11% increase of spare block usage for panel B and D compared to the usage in panel A and C. This increase in spare block usage suggests that panel B and D have a harder working FTL to achieve wear

leveling.

Without the file cache the panel OS cannot use any RAM memory to store file data before being sent in bundles to the SD card. Therefore it is likely that the OS sometimes requires the FTL of the SD card to temporarily store this file data in the over-provisioned spare blocks while certain reorganizing operations are performed, which would explain the difference in spare block usage

Interestingly there also seems to be a correlation between a near capacity SD card memory and a higher usage of spare blocks, especially for panels with only the disk cache active. The fuller SD card in panel C only have an increase of 1% in spare block usage compared to panel A whilst panel D have an increase of 6% in usage compared to panel B. This change is likely caused by panel C having file cache activated and therefore the panel can use the OS RAM memory file cache to store vital data. Meanwhile panel D cannot and must use the SD card spare blocks for the same data.

Eventually one may conclude that a more filled SD card will require the internal working wear leveling algorithms in the card FTL to work harder to alter the 10% free memory blocks to even out the wear. Unfortunately this isn't visible in the SD bus requests, since the FTL may change the addresses for wear leveling and hence we see the same logical addresses continuously used as can be seen in Figure 4.2. However, in reality the FTL is likely cycling through all memory blocks, continuously shifting what 10% memory blocks are to be considered free, evening out the wear in the process. Although this alteration is not explicitly visible the higher spare block usage of panel D implies that this wear leveling process is happening. Also, if the FTL wouldn't cycle through different blocks, it is highly likely that panel C and D would break long before A or B, since the available blocks would be worn out. Since this leveling feature requires significantly more additional data to be internally transferred the OS in panel D, the FTL is required to store this data directly in the SD card spare blocks. Meanwhile the OS file cache can temporarily store the majority of this additional data which is why the difference between spare block usage is significantly smaller between panel A and C than between C and D.

5.2.3 Execution time & Average Speed

As can be seen in Table 4.8, the execution time is not a measure of time to accomplish a task, but rather a measure of how fast the panel broke. To gauge the writing speed of the individual configurations files written per hour is the important field. This field shows that there were significant differences in writing speed between panels A and B; and C and D. This can also be gauged in the "Average speed" field, since the WAF is low enough to not affect the numbers significantly. From these fields it could be determined that operating a panel near capacity reduces writing speed by circa 15 %, most likely due to the internal shuffling of data that regularly needs to be performed. It could also be determined that, for these specific file sizes, alternating between file cache on and disk cache off to file cache off and disk cache on did not meaningfully affect the writing speed of the program.

5.2.4 Correlation between theoretical and actual SD card wear

As described in Section 4.2.5 the amount of bad blocks was surprisingly low as the long-term tests were conducted. Retrieving a single bad block in only one out of four SD cards was indeed unexpected especially considering, as previously described, that the specific SD cards used in the long-term tests have a MLC cell encoding with a known default erase cycle lifetime of only 3000. Since the long-term tests executed until this threshold was hit, and then the theoretical default lifetime was reached, one could consider the very low bad block count unreliable.

Because of the very low values of bad blocks one may with high probability conclude that the SD cards have not experienced any advanced wear. One can at least conclude that the actual card wear is far from close to correlate with the achieved theoretical default lifetime of 0%. The reasons why there exists such a gap between the theoretical and actual SD card wear are multiple. Firstly, the most obvious reason is that the placed 3000 lifetime erase cycles is a default value given by the producers of the SD card. To uphold any warranties placed on the card the producers are likely to set a default erase cycle value on the lower end of the actual spectrum that exists in practicality. Meaning that to assure that the card can handle a certain placed amount of cycles a lower value is chosen accordingly.

A second reason is that the block erase cycle data point retrieved from Hard Disk Sentinel software, which determined the end point of the long-term tests, was an average value. This resulted in all long-term tests terminating when the memory blocks of the SD card had experienced 3000 erase cycles on average. Hence, in reality, many of the blocks are likely to have endured less erase cycles than the default value of 3000.

In addition to these two main reasons the relatively low transfer speed existing in the long-term tests yields a less hard working SD card. This together with the well designed physical SD card slot in the HMI panels used in all tests eventually lead to a lower average temperature of the SD cards. Since a lower SD card temperature conducts a lower strain of the isolating layers in the floating gate transistors the memory blocks can handle a higher number of erase cycles during their lifetime.

In the end, the results consisting of very low bad block counts are not as unexpected as one might initially think. Eventually running the tests just until the default threshold of erase cycles is reached is not enough to experience any advanced wear of the SD cards. In the context of a longer existing time frame one would, by letting the long-term tests continue to execute, likely experience an exponential increase of bad blocks after a certain amount of time.

5.2.5 Limitations & unknowns

Regarding the long-term tests, as a result of limited time only one file size was possible to test. As can be seen in the short term tests, file size can have a large impact on outcome. The tests were also performed on one type of platform for a specific use and operating system. Therefore it is not possible to claim that these findings necessarily hold true for different

platforms or operating systems. Because of these two factors the long term tests are limited in applicability.

Due to the limited time frame the long-term tests were only executed until each SD card would reach an average memory block erase count value of 3000, the default NAND flash endurance of the card. Because of this, the long-term tests only executed until the default life-time of the SD-card was reached and the card is to be consider consumed. However, as the results show, the number of bad memory blocks for the cards are very nearly non-existent, meaning that the cards are likely far from consumed in practice. Therefore the results of the long-term tests are not possible to use for yielding any results in deciding what test configuration would generate a non-theoretically consumed SD-card.

5.3 Optimal panel configurations for different practices

Considering the industrial context of HMI panels any small changes in performance may result in large and expensive differences. There are many different aspects to consider when choosing an optimal panel configuration and the specific context of the HMI panel may greatly alter what traits are considered optimal. The optimal values of transfer speed, amount of data written, WAF value and memory capacity are all important features to decide for an optimal panel configuration. Therefore, there unfortunately exists no single optimal panel configuration.

However, considering the results yielded from this master thesis testing one could map important panel traits with a specific panel configuration. In the context of high stability and reliability a panel configuration of having no cache enabled or only file cache enabled would be a better choice. This configuration lowers the WAF value but scarifies transfer speed as can be seen for panel C and D in the tables 4.2, 4.3, 4.4 and 4.5. If possible one should also consider only writing files of substantial size, 1000 kilobytes or bigger, since this also lowers the WAF values. However, as seen in the previously referred tables, increasing the file sizes lowers the importance of panel configuration as the difference between WAF values decreases for tests with bigger file sizes. For executions with big file sizes one might instead choose the configurations of panel A and B, sacrificing a small decrease in WAF value but increasing the data transfer speed significantly.

If, by contrast, data transfer speed would be considered more important than low WAF values then the configuration of panel A, meaning having both caches enabled, would likely be the best fit. This configuration yields a low execution time for the tests regardless of file size. However, one must be aware that this configuration increases the WAF values quite significantly for executions using small file sizes as seen in Table 4.2. For the purpose of increasing data transfer speed additionally the memory used must not be cluttered by great amounts of unnecessary data. Instead, as can be seen in Table 4.8, there is an increase in write speed for panel A compared to panel B and also panel C compared to panel D. Therefore, by al-

lowing an execution to use the whole capacity of the used memory the transfer speed will be increased further.

In the context of everyday usage, where a middle ground of low WAF values together with reasonable transfer speeds is needed, enabling both caches as in panel A is likely the best choice. As seen in the tables of Section 4.1.2 the configuration of panel A yields low WAF values whilst it simultaneously experiences good transfer speeds.

5.4 Research questions

Thanks to the results of the short-term and long-term tests all three research questions could be answered. Although the scope of this thesis is limited the results yielded enough information to also additionally discuss topics such as over-provisioning and the correlation between theoretical and actual SD card wear. Both are subjects that were not perceived as possible to answer before the thesis was finished.

How does file cache and disk cache effect flash wear in embedded systems?

The direct impact of different cache configurations could most easily be seen in the difference of WAF values and data transaction speed. These two reference points also differ depending on the explicit file size that was used in the specific test. Lower file sizes having the disk cache enabled resulted in a higher WAF value, especially when only the disk cache was enabled. When considering larger file sizes the explicit WAF value difference between the cache configurations diminished.

How does different levels of free memory effect IO performance and flash wear?

As discussed in subsection 5.2.3, IO performance is negatively impacted by operating at near capacity, likely due to internal shuffling of blocks to achieve wear leveling. Therefore it is in the users best interest, regardless of use case, to have as much free memory as possible to improve I/O performance.

What is the optimal panel configurations for Beijer Electronics?

As previously stated different use cases will have different optimal configurations. It is therefore difficult to accurately state which configuration would be best for Beijer Electronics and their customers without first examining their writing patterns. It would be possible to sell different versions of the HMI panels with different emphasis, either on speed, durability, or a middle ground.

5.5 Further Work

To expand the knowledge of how caching effects flash wear multiple different operating systems with different implementations of these tools are affected. Furthermore, examining the effects under different operating conditions might yield different results. Doing random writes, multiple sequential read to the same file, and varying file sizes might all have some effect on wear. Writing patterns closely resembling actual writing patterns will be more relevant to study since the result would be easily applicable.

Further examining additional configurations, as well as their effects when combined with each other. Closely examining these effects might yield unexpected results. A closer examination of the FTL could also give researchers a better understanding of what happens with data after it arrives at the destination memory. Conducting a further study of sequentiality patterns may be important in discovering faster and more durable flash memories. An expanded view of the patterns would provide further insight in the inner workings of how flash memories and file systems cooperate. Additionally examining a broader range of commands sent between to panels and their SD-cards may give further clues into the exact behaviour of the panels.

Finally, continuously monitoring panels performing long-term test until they were not able to function would give a deeper insight into how good the underlying FTL actually is at wear leveling. Currently we suspect that shifting blocks between dies for panels near capacity is something that the FTL is capable of, though the precise cost of this was unable to be determined given time constraints.

Chapter 6
Conclusion

In conclusion, during the experiments conducted it was gathered that file cache and disk cache effect the WAF, especially for small file sizes. It was found that enabling disk cache had the most negative impact on WAF, but that the impact lessened as file sizes grew. It was also found that enabling file cache alongside disk cache helped stave the negative effect of disk cache until file sizes grew beyond 1000 KB. The configuration resulting in the lowest WAF was the configuration with only file cache enabled or both forms of caches disabled. Disabling all forms of cache however did result in the slowest write speed for file sizes of 1000 KB and below.

Regarding sequentiality it was found that file and disk cache enabled resulted in the most sequential writing pattern, up until file sizes of 1000 KB. For larger file sizes only enabling disk cache or disabling both forms of cache resulted in the most sequential program.

It was also found that writing to panels close to full memory capacity slowed down execution time significantly. This behavior is believed to be the result of necessary shuffling being performed by the FTL in the background, which would theoretically increase the WAF over the measured amount.

We found that SD cards underestimate their flash cell durability to the extent that they may continue to operate far beyond their stated lifetime.

Lastly it is important to note that these findings are limited in their scope and platform specific research should be done before accepting these findings as true for different platforms, as different platforms have their own implementations of these systems.

References

[1] Tom. Coughlin. A timeline for flash memory history [the art of storage]. *IEEE Consumer Electronics Magazine, Consumer Electronics Magazine, IEEE, IEEE Consumer Electron. Mag*, 6(1):126 – 133, 2017.

[2] Richter. Detlev. *Flash Memories - Economic Principles of Performance, Cost and Reliability Optimization*. Springer, 2013.

[3] Boling Douglas. *Programming Microsoft Windows CE .NET*. Microsoft Press, 2003.

[4] ATP Electronics. How wear leveling increases ssd lifetime. `https://www.atpinc.com/blog/how-SSD-wear-leveling-works`, 2020. Accessed: 20.02.2023.

[5] Beijer Electronics. Hmi panels - what is hmi? `https://www.beijerelectronics.com/en/Products/Operator___panels`. Accessed: 12.02.2023.

[6] Beijer Electronics. Our company. `https://www.beijerelectronics.com/en/About___us/Our___company`. Accessed: 10.02.2023.

[7] Beijer Electronics. X2 pro 7. `https://www.beijerelectronics.com/en/Products/Operator___panels/X2___pro/X2___pro___7___with___2___Ethernet`. Accessed: 14.02.2023.

[8] Bez. Roberto et al. Introduction to flash memory. *Proceedings of the IEEE*, 91(4), 2003.

[9] Boukhobza. Jalil et al. A cache management strategy to replace wear leveling techniques for embedded flash memory. In *2011 International Symposium on Performance Evaluation of Computer & Telecommunication Systems*, pages 1–8, 2011.

[10] Boukhobza. Jalil et al. Characterization of oltp i/o workloads for dimensioning embedded write cache for flash memories: A case study. 2011.

[11] Grattan. Nick et al. *Windows CE 3.0: Application Programming*. Prentice Hall PTR, 2000.

[12] Jeremic. Nikolaus et al. Operating system support for dynamic over-provisioning of solid state drives. In *Proceedings of the ACM Symposium on Applied Computing*, number 27th Annual ACM Symposium on Applied Computing, SAC 2012, pages 1753–1758 – 1758, 2012.

[13] Jian. Hu et al. Pud-lru: An erase-efficient write buffer management algorithm for flash memory ssd. *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 69 – 78, 2010.

[14] Malueg. Michael et al. Transaction-safe fat file system, US patent, February, 2007.

[15] Meza. Justin et al. A large-scale study of flash memory failures in the field. 43(1):177–190, jun 2015.

[16] Micheloni. Rino et al. *Inside NAND Flash Memories*. Springer Netherlands, 2010.

[17] Pavan. Paolo et al. Flash memory cells- an overview. *Proceeding of the IEEE, proc. IEEE*, 85(8):1248–1271, 1997.

[18] Pavlov. Stanislav et al. *Windows® Embedded CE 6.0 Fundamentals*. Microsoft Press, 2008.

[19] R. Haas et al. The fundamental limit of flash random write performance: Understanding, analysis and performance modelling, 2010.

[20] Rai. Sadhana et al. *Frontiers of Quality Electronic Design (QED) [Elektronisk resurs] : AI, IoT and Hardware Security*. Springer International Publishing, 2023.

[21] Stouffer. Keith et al. Guide to industrial control systems (ics) security. *NIST Special Publication*, Vol 800(82), 2015.

[22] Wang. Li et al. Write amplification trade-off analysis in hybrid mapping solid state drives. *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA), Industrial Electronics and Applications (ICIEA), 2020 15th IEEE Conference on*, pages 1896 – 1901, 2020.

[23] Wang. Shengzhe et al. Learnedftl: A learning-based page-level ftl for improving random reads in flash-based ssds. 2023.

[24] Rabaey. Jan. *Digital integrated circuits : a design perspective*. Upper Saddle River, N.J, 2003.

[25] Microsoft. Texfat overview, 2008. available at `https://learn.microsoft.com/en-us/previous-versions/cc907929(v=msdn.10)`.

[26] Microsoft. File caching. 2021. Available at: `https://learn.microsoft.com/en-us/windows/win32/fileio/file-caching`, Accessed: 2023-01-26.

[27] Yadhu N. Gopalan Sachin Patel. Transaction safe fat file system improvements, US patent, September, 2011.

[28] STMicroelectronics. Crc overview, 2022. available at https://www.st.com/resource/en/application_note/an5507-cyclic-redundancy-check-in-stm32h7-series-flash-memory-interface-stmi pdf.

[29] Cactus technologies. Ctwp013: Wear leveling - static, dynamic and global, 2019. Available at: "https://www.cactus-tech.com/wp-content/uploads/2019/03/Wear-Leveling-Static-Dynamic-Global.pdf. Accessed: 31.01.2023.

Appendices

Appendix A
Figures

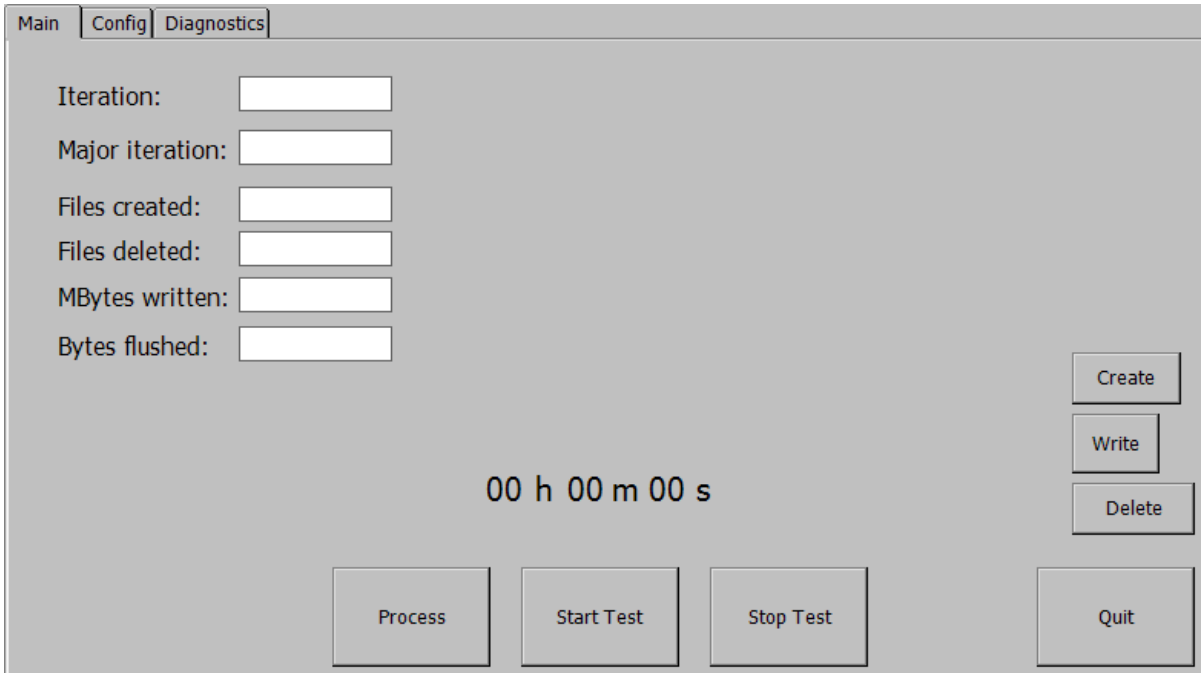
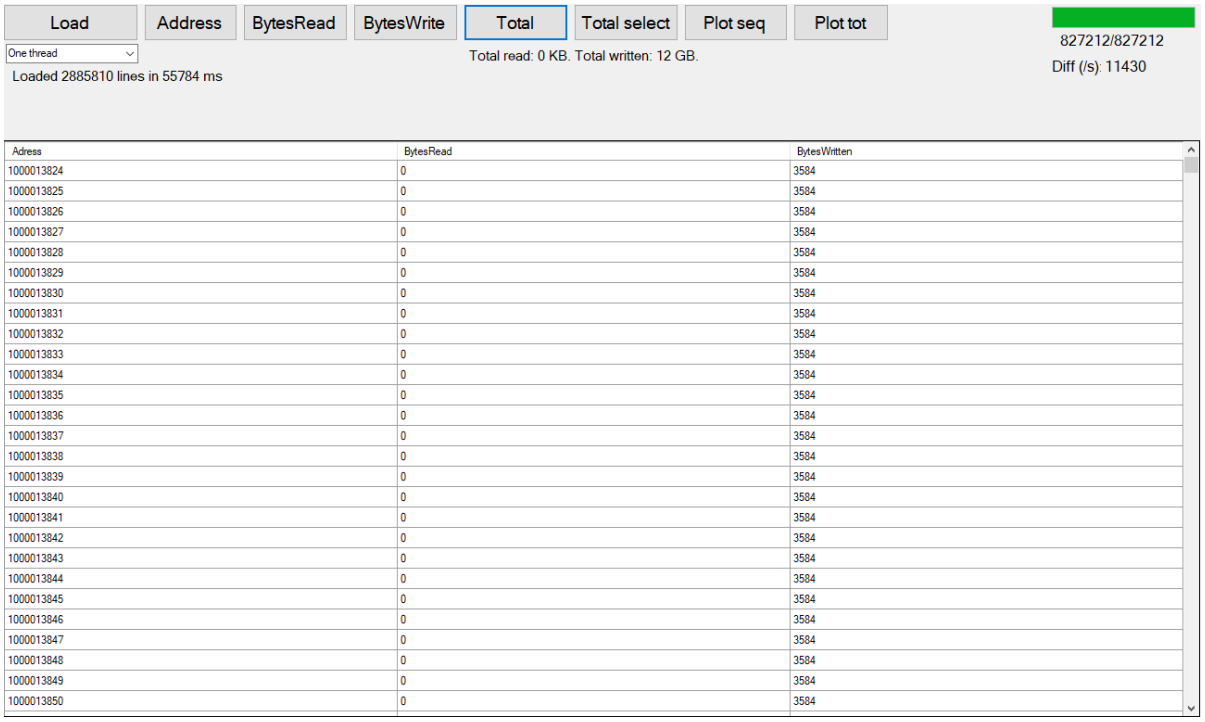


Figure A.1: A `C++` application running on a HMI panel developed to write a number of files with specific size to an inserted SD card.



Adress	BytesRead	BytesWritten
1000013824	0	3584
1000013825	0	3584
1000013826	0	3584
1000013827	0	3584
1000013828	0	3584
1000013829	0	3584
1000013830	0	3584
1000013831	0	3584
1000013832	0	3584
1000013833	0	3584
1000013834	0	3584
1000013835	0	3584
1000013836	0	3584
1000013837	0	3584
1000013838	0	3584
1000013839	0	3584
1000013840	0	3584
1000013841	0	3584
1000013842	0	3584
1000013843	0	3584
1000013844	0	3584
1000013845	0	3584
1000013846	0	3584
1000013847	0	3584
1000013848	0	3584
1000013849	0	3584
1000013850	0	3584

Figure A.2: A `C#` application developed to analyze created log files from executed HMI panel tests. Used to calculate the total number of read and written bytes. Can also display every unique memory block address with the associated total number of bytes read and written.