ТҮЙІН

Мақалада болашақ туризм бакалаврлары мен магистрлерін мамандандырылған бағдарламалық өнімдерін, ақпараттық-ізденіс жүйелерін және ғаламтор ресурстарын кәсіби қызметінде пайдалану қажеттілігі дәлелденген.

RESUME

In the article there was founded the necessity of preparation of students - the future of bachelors and masters on tourism in the area of integrated use of specialized software, information retrieval systems and resource allocation Internet in their professional activities. The content of the main components of information activity of bachelors and masters on tourism were proved. It was grounded the principles of the integrated use of ICT in professional activity of bachelors and masters on tourism.

УДК 004.087.2

Rozorinov Heorhiy, grozoryn@gmail.com
Bryagin Oleg, bryagin_oleg@ukr.net
Trush Alexander Trush-viti@mail.ru
State University of Telecommunications, Kiev, Ukraine

LOW-LEVEL DATA RECOVERY FROM FLASH MEMORY

Abstract

This article suggests a low level approach for the examination of flash memories and describes low-level data acquisition methods for making full memory copies of flash memory devices. Artifacts, caused by flash specific operations likeblock erasing and wear leveling, are discussed and directions are given for enhanced data recovery and analysis on data originating from flash memory.

Keywords: *NAND flash memory; bad-block management; flasher tools; level algorithm.*

Introduction

Over recent years, corporate end-users have increasingly needed to be fully mobile and connected, taking work home or out of the office to keep up their productivity. Staff needs to be able to synchronise files between a computer and the drive to allow key data to be backed up and available for use on the road or on other computers. Thus, the use of mobile devices such as laptops, notebooks, universal serial bus (USB) flash drives, personal digital assistants, advanced mobile phones and other mobile devices have proliferated in recent years [1,2]. Flash memory is currently the most dominant non-volatile solid-state storage technology in consumer electronic products. An increasing number of embedded systems use high level file systems comparable to the file systems used on personal computers.

Using of different types of devices (modules) of flashmemory for the electronic engineering requires the detailed acquaintance with processes which take a place at a recording and reproducing of information in such devices.

At the same time, for manufacturers of memory devices, some parts, such as, the wear levelling algorithm can be very sensitive intellectual property, so any inquiries that look like questions about the wear levelling algorithm will often be left unanswered [3].

Flash technology

Flash memory is a type of non-volatile memory that can be electrically erased and reprogrammed. Flash memory comes in two flavors, NOR flash and NAND flash, named after the basic logical structures of these chips. Contrary to NAND flash, NOR flash can be read byte by byte in constant time which is the reason why it is often used when the primary goal of the flash memory is to hold and execute firmware, while parts of NOR flash that are not occupied by firmware can be used for user data storage. Most mobile media, like USB flash disks, or multimedia centred devices like digital camera's and camera phones, use NAND flash memory to create compact mobile data storage.

Physical Characteristics

The physical mechanism to store data in flash memory is based on storing electrical charge into a floating gate of a transistor. This charge can be stored for extended periods of time without using an external power supply but gradually it will leak away caused by physical effects. Data retention specifications for current flash memory are between 10 and 100 years.

Flash memory can be written byte for byte, like EEPROM, but it has to be erased in blocks at a time before it can be re-written. Erasing results in a memory block that is filled completely with 1's. In NAND flash, erase blocks are divided further into pages, for example 32 or 64 per erase block [4]. A page is usually a multiple of 512 bytes in size, to emulate 512 byte sector size commonly found in file systems on magnetic media. Additionally, a page has a number of so called "spare area" bytes, generally used for storing meta data. Some flash disk drivers use the concept of zones. A zone is a group of blocks, usually 256 to 1024. Contrary to blocks and pages, a zone is just a logical concept, there is no physical representation. A dissection of NAND flash memory is shown in Figure 1.

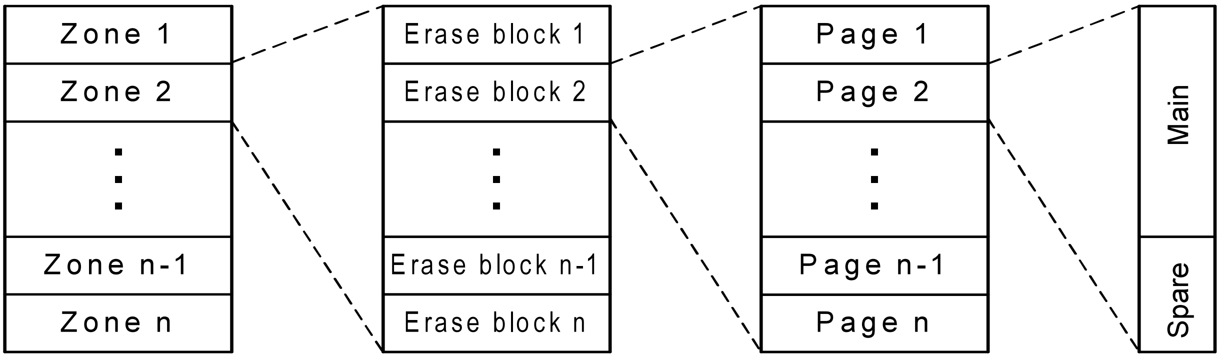

Figure 1 – Dissection of NAND flash memory

Each page has an area of bytes, often referred to as the redundant area or spare area. Table I shows spare area sizes for different page sizes. The spare area can contain information on the status of the block or the page.

Table 1 – Example spare area sizes for different page sizes

Page size, bytes	Size, bytes		
	Spare area size	Total page size	Block size
256	8	264	8448
512	16	528	16896
2048	64	2112	135168

For instance when a block turns bad, it will be marked here. The spare area can also contain "Error Checking and Correcting" (ECC data). ECC data is used to detect errors in a page. With the ECC data an error of one bit can be corrected, after which the block will be marked bad. Finally the spare area can contain information necessary for the physical to logical address mapping.

Erasing a block causes a block to deteriorate. Blocks can be erased between 10^4 and 10^6 times before bits in this block start to become inerasable. Such block is then called a "bad block". NAND flash usually already has bad blocks when leaving the factory. In datasheets of NAND flash chips, the guaranteed minimal number of good blocks when first shipped is specified. Typically at least 98% of the blocks are guaranteed to be in working order. Initial bad blocks are marked as such in the spare area.

In order to spread the erasing of blocks as evenly as possible over the full range of physical blocks, flash memory vendors have developed so called "wear levelling" algorithms [5]. The idea is that spreading the wear, caused by erasing a block, as much as possible over the whole capacity of the flash memory will increase the overall lifetime of the memory. Wear leveling can be seen as a dynamic process that rearranges pages and/or blocks continuously in order to extend flash lifetime.

The electrical interface of NAND flash differs from that of RAM. NAND flash has a multiplexed address/data bus, generally referred to as the I/O (Input/Output) lines. This bus can be either 8 or 16 bits wide. Data in the NAND flash chip is accessed by first applying the address of the required data on the I/O lines. As the highest address is generally higher than can bereached with 8 or 16 I/O line bits, the address is latched into the chip in three to five address cycles. After the address is latched into the chip, the data can be clocked out over the same I/O lines. A typical sequence to get access to data in a NAND flash chip is shown in table 2.

Table 2 – Addressing cycles for a NAND flash memory

Cycle	Input/Output lines								
	I/O_0	I/O_1	I/O_2	I/O_3	I/O_4	I/O_5	I/O_6	I/O_7	
1	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Column Address
2	A_8	A_9	A_{10}	A_{11}	Low	Low	Low	Low	Column Address
3	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	Row Address
4	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	Row Address

Logical Characteristics

There are several ways in which flash memory can be used as file storage in embedded systems [3]. Three of them are explained below. A simplified diagram of components involved in host Operating System (OS) access to a flash file system is shown in Figure 2. As a reference, the situation for a hard disk is shown on the left hand side. In case of a hard disk, the host OS accesses the hard disk through the file system driver (FSD). The FSD issues commands to the hard disk, for instance the ATA command "Read Sector" to read data from a Logical Block Address (LBA) - the address of data by the linear mapping of storage units.

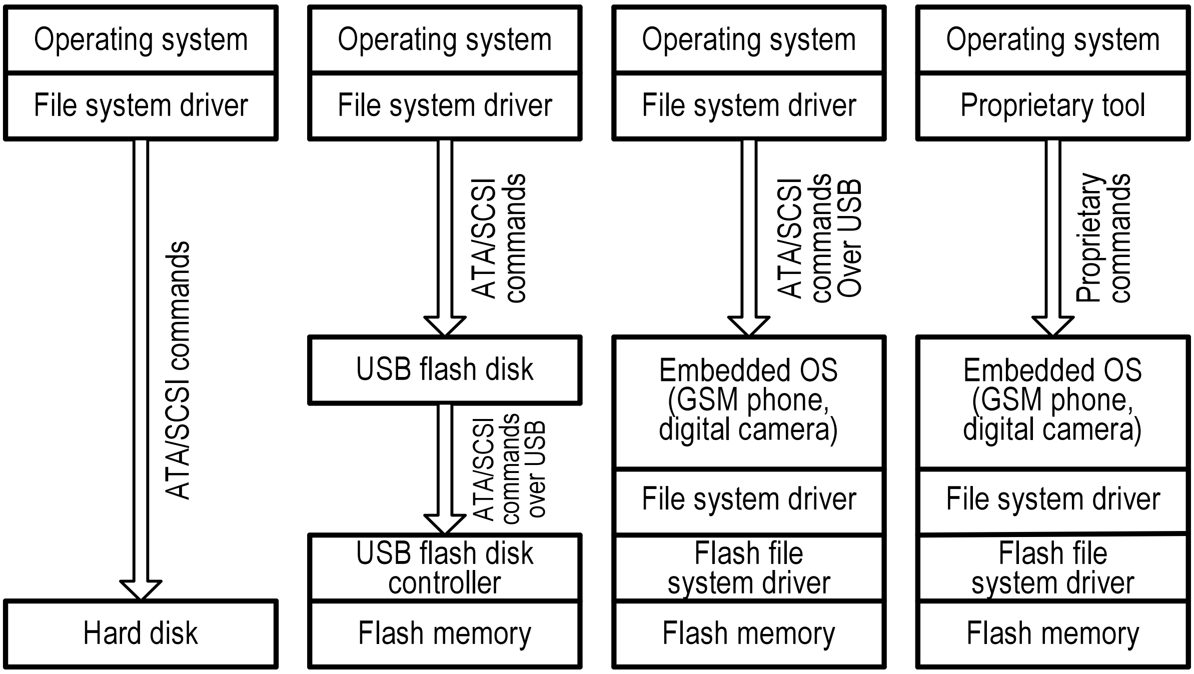

Figure 2 – Components involved in hard disk and flash memory access

Flash Translation Layer

NAND flash is typically used with a flash translation layer implementing a disk-like interfaceof addressable, re-writable 512-byte blocks, e.g. over an interface such as SATA or SCSI-over-USB.The FTL maps logical addresses received over this interface (Logical Page Numbers or

LPNs) to physical addresses in the flash chip (Physical Page Numbers, PPNs) and manages the details of erasure, wear-leveling, and garbage collection.

A flash translation layer could in theory maintain a map with an entry for each 512-byte logical page containing its corresponding location; the overhead of doing so would be high, however, as the map for a 1GB device would then require 2M entries, consuming about 8MB; maps for larger drives would scale proportionally. FTL resource requirements are typically reduced by two methods: zoning and larger-granularity mapping.

Zoning refers to the division of the logical addressspace into regions or zones, each of which is assigned its own region of physical pages. In other words, rather than using a single translation layer across the entire device, multiple instances of the FTL are used, one per zone. The map for the current zone is maintained in memory, and when an operation refers to a different zone, the map for that zone must be loaded from the flash device. This approach performs well when there is a high degree of locality in access patterns; however it results in high overhead for random operation. Nonetheless it is widely used in small devices (e.g. USB drives) due to its reduced memory requirements.

By mapping larger units, and in particular entire erase blocks, it is possible to reduce the size of the mapping tables even further [6]. On a typical flash device (64-page erase blocks, 2KB pages) this reduces the map for a 1GB chip to 8K entries, or even fewer if divided into zones. This reduction carries a cost in performance: to modify a single 512-byte logical block, this block-mapped FTL would need to copy an entire 128K block, for an overhead of 256×.

Whenever units smaller than an erase block are mapped, there can be stale data: data which has been replaced by writes to the same logical address (and stored in a different physical location) but which has not yet been erased. In the general case recovering these pages efficiently is a difficult problem. However in the limited case of hybrid FTLs, this process consists of merging log blocks with blocks containing stale data, and programming the result into one or more free blocks. These operations are of the following types: switch merges, partial merges, and full merge. A switch merge occurs during sequential writing; the log block contains a sequence of pages exactly replacing an existing data block, and may replace it without any further operation; the old block may then be erased. A partial merge copies valid pages from a data block to the log block, after which the two may be switched. A full merge is needed when data in the log block is out of order; valid pages from the log block and the associated data block are copied together into a new free block, after which the old data block and log block are both erased.

Many applications concentrate their writes on a small region of storage, such as the file allocation table (FAT) in MSDOS-derived file systems. Naive mechanisms might map these logical regions to similarsized regions of physical storage, resulting in premature device failure. To prevent this, wear-leveling algorithms are used to ensure that writes are spread across the entire device, regardless of application write behavior; these algorithms are classified as either dynamic or static. Dynamic wear-leveling operates only on overwritten blocks, rotating writes between blocks on a free list; thus if there are m blocks on the free list, repeated writes to the same logical address will cause $m + 1$ physical blocks to be repeatedly programmed and erased. Static wear-leveling spreads the wear over both static and dynamic memory regions, by periodically swapping active blocks from the free list with static randomly-chosen blocks.

Bad Block Management

Bad blocks are blocks that contain one or more invalid bits whose reliability is not guaranteed [7]. Bad blocks may be present when the device is shipped, or may develop during the lifetime of the device. Devices with bad blocks have the same quality level and the same AC and DC characteristics as devices where all the blocks are valid. A bad block does not affect the performance of valid blocks because it is isolated from the bit line and common source line by a select transistor. Bad block management, Low Level Driver (LLD), and the ECC software are necessary to manage the error bits in NAND Flash devices. Figure 3 shows how the software is used in an embedded system which uses NAND Flash for data storage.

NAND Flash devices are supplied with all the locations inside valid blocks erased (FFh). The bad block information is written prior to shipping. For single-level cell (SLC) small page (528-byte/256-word page) devices, any block where the sixth byte (x8 device)/first word (x16 device), in the spare area of the first page does not contain FFh is a bad block. For SLC large page (2112-byte/1056-word page) devices, any block where the first and sixth bytes (x8 device)/first word (x16 device) in the spare area of the first page does not contain FFh is a bad block. For SLC very large page (4224-byte page) devices, any block where the first and sixth bytes in the spare area of the first page does not contain FFh is a bad block. For multilevel cell (MLC) devices, any block where the first byte in the spare area of the last page does not contain FFh is a bad block. The bad block information must be read before any erase is attempted because the bad block information is erasable and cannot be recovered once erased. It is highly recommended to not erase the original bad block information. To allow the system to recognize the bad blocks based on the original information, it is recommended to implement the bad block management algorithm.

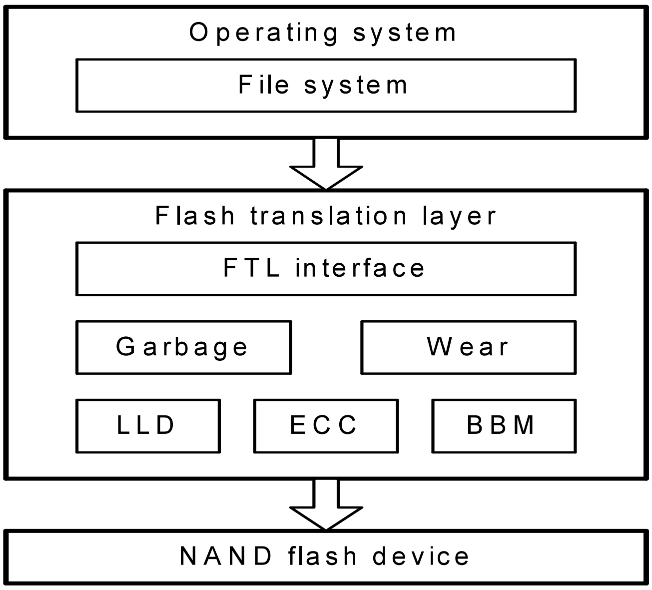

Figure 3 – Software for an embedded system using a NAND flash memory

The failures that affect invalid blocks may not all be recognized if methods different from those implemented in the factory are used. Once created, the bad block table is saved to a good block so that on rebooting the NAND Flash memory the bad block table is loaded into RAM. The blocks contained in the bad block table are not addressable. So, if the flash translation layer (FTL) addresses one of the bad blocks, the bad block management software redirects it to a good block.

Low-level data recovery

The main condition for low-level data recovery - to keep data held on a storage medium unchanged [8]. For embedded systems this principle is more challenging than it looks at first sight. Issues like network connections are similar to the open systems world although it might be more difficult to detect that an embedded system is connected to other systems. For flash memory wear leveling might cause unpredictable data changes. For example, switching mobile phones off and/or on has shown data changes probably caused by wear leveling and/or garbage collection algorithms. There are three possible data acquisition approaches are presented for obtaining a full copy of flash memory data: flasher tools, method using the JTAG (Joint Test Action Group) test access port of an embedded device and method is described in which the flash chip is physically removed and read with an external reader.

Advantages / disadvantages of physical extraction:

- It can be guaranteed that no data is written in flash memory because the embedded system stays powered down.
- Data from broken or damaged embedded systems can be recovered.
- A complete physical image can be produced (all data, inclusive spare area, bad blocks etc).
- A disadvantage is that there is a risk of damaging the flash memory chip due to the heat for de-soldering.
- The embedded system has to be opened to reach and desolder flash memory chips.

We describe a method of low-level data recovery from flash chip is physically removed.

Flash memory chip reader

A flash memory chip can be read with a commercially available memory chip programmer. A disadvantage is that a driver is needed for each type of memory chip. If a driver for a certain type of chip is not available, the manufacturer of the programmer has to program this driver. This can take some time and is not always possible when a datasheet is not available for example. Another solution is to use a universal flash chip reader. This custom made design is called "NFI memory toolkit". A schematic is drawn in Figure 4.

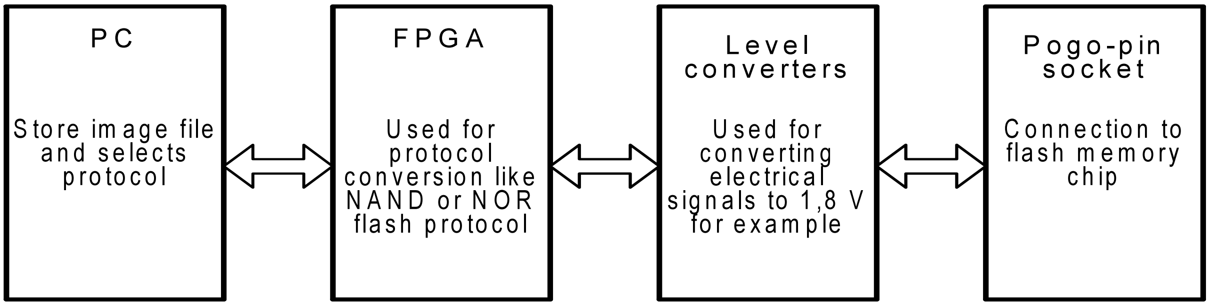

Figure 4 – Schematic of NFI memory toolkit

An FPGA is used for communicating with a flash memory chip where configurations are available for a NAND and NOR flash protocol (with multiplexed and demultiplexed address bus). All parameters, like address bus size and data bus size are fully customizable by the PC software. In case of a NOR flash memory a data structure is read from the NOR flash memory (Common Flash Interface - CFI data structure). This data structure contains all parameters needed for reading that particular flash memory (like protocol, memory size, etc). The command to read this data structure is compatible with all protocols and the toolkit software automatically uses the parameters to read a NOR flash chip without any configuration from the user. NAND flash chips can also be read automatically because the number of protocols used by NAND flash chip is very limited. The toolkit software automatically scans all protocols until a correct response is received from the NAND flash chip. Due to the automatic configuration properties of the software it is sometimes possible to read flash chips even if a datasheet is not available.

There are several third-party software offerings on the market today. Many of these packages provide multiple features, including automatic power failure-recovery, PC-file compatibility, ECC, bad-block management, directory support, and wear-leveling.

File system analysis

Making an exact copy of the flash chip(s): When the chip is extracted from the PCB, it can be read with a device programmer. When reading the content of flash chips one needs to be aware of the fact that some programmers have a special way of handling NAND flash. When programming a NAND flash in a production environment, the programmer obviously wants to skip bad blocks. Further more, when a file is loaded in the programmer, one wants to be sure that the file will fit in the flash chip, so the programmer will only accept files smaller than the guaranteed minimal number of good blocks. These two properties often also play a role when reading the device. Bad blocks are not read, and only the guaranteed minimal number of good blocks is read. Skipping of bad blocks can lead to the following problem when reconstructing the high level file system: suppose a USB memory controller divides the memory into zones of 256 blocks. Each block (belonging to the high level file system) within a zone has to have a unique number, stored in the spare area of each page in that block. Then one bad block in a zone arises. After this, the memory is imaged with a programmer that skips bad blocks. The resulting image is also split up into zones. Now, the zone with the bad block will contain one block of the next zone because all blocks after the bad block will be shifted one block in the image. Now it will be very likely that we have two blocks with the same "unique" number in one zone.

Reading up to only the guaranteed minimum number of good blocks can result in blocks at the high end of the memory chip not being read. These blocks might very well contain parts of the high level file system so not reading them might hinder reconstruction of the high level file system. There are several solutions to these problems. One is to request the manufacturer of the device programmer to make a special version of the algorithm for the specific chip which reads all blocks, good and bad.

Another is to develop an "in house" solution. For the memory toolkit an algorithm for reading NAND flash was developed. Furthermore, an adapter socket was made to make contact to TSOP 48 housings. With this system acomplete binary copy of a NAND flash memory chip can be made. The rest of this paragraph is based on complete binary copies of flash chips, made with the NFI memory toolkit.

Converting the copy to the high level file system: in order to convert the exact copy of the NAND flash memory back to the file system as seen by the host OS, the meta data in the NAND flash memory needs to be interpreted.

When we want to reconstruct the file system from a physical copy, the mapping from physical to logical, we has to explore the meta data. Meta data can be stored in the spare areas of the flash memory. In case there is a page size granularity, all spare areas within each block will contain different information. In case there is a block size granularity, spare areas within one erase block may contain at least a few identical bytes: the ones that indicate the logical block number. Meta data can also be stored in the normal pages/blocks. No generic method has yet been developed for USB memory to analyze this type of meta data storage.

Conclusion

Some techniques have been described for making low level byte-by-byte copies of flash memory chips. More research needs to be done on the flash read mechanisms used by flasher tools in order to adapt these mechanisms for usage in the next generation of data acquisitions tools. Steps have been illustrated for translating acquired flash data to a level that can be understood by existing tools targeted towards common used file systems. More research is also needed on the relation between flash specific operations like block erasing and wear leveling on one side and the resulting artifacts and potentials for data recovery and analysis on the other side.

REFERENCES

1 Розоринов Г.Н., Брягин О.В., Неня Е.В. Современные магнитные накопители для систем обработки акустической информации // Реєстрація, зберігання та обробка даних. – 2003. – Т. 5, № 3. – С. 91–105.

2 Secure USB Flash Drivers ISBN: 978-92-9204-011-6 Catalogue number: TP-30-08-571-EN-C.

3 Marcel B., Martien de J, Coert K, Ronald van der K and Mark R., Forensic Data Recovery from Flash Memory. Small Scale Digital Device Forensics Journal, Vol. 1, No. 1. – 2007.

4 Баишев А. “Твердотельные диски — надёжное решение для ответственных применений”. Ч.2 // Современные технологии автоматизации (СТА). – М. : СТА-Пресс. – 2007. – № 4. – С.68-71.

5 Boboila S., Desnoyers P. Write Endurance in Flash Drives: Measurements and Analysis / 8-th USENIX Conference on File and Storage Technologies (FAST '10), San Jose, California. February 2010.

6 Axelson J. USB Mass Storage: Designing and Programming Devices and Embedded Hosts. – Lakeview research LLC, Madison W1, P. 13-22, 2006. – P.287.

7 TN-29-59: “Bad Block Management in NAND Flash Memory”,Micron Technology, Inc.

8 Сенкевич Г.Е. Искусство восстановления данных. Аппаратные средства. – Санкт-Петербург, БХВ-Петербург, 2011. – 304 с.

ТҮЙІН

Мақалада флеш-зердені талдау тұрғылары көрсетілген, олардың негізінде операциялық жүйемен төмен деңгейде өзара әрекеттесу. Толық флеш-зерде көшірмесін жасаудың кейбір тәсілдері, флеш-зерделен алынған деректерді кеңінен қалпына келтіру жолдары көрсетілген. Флеш-зердеге тән операциялардан пайда болған артефактылар талқыланған.

РЕЗЮМЕ

В статье предлагаются подходы для анализа флеш-памяти, которые основаны на низком уровне взаимодействия с операционной системой. Показаны некоторые способы создания полных копий флеш-памяти, обозначены пути расширенного восстановления данных и анализа данных, извлеченных из флеш-памяти. Обсуждены артефакты, вызванные операциями характерными для флеш-памяти, а именно: стирание блоков или «выравнивание изнашивания».