IGOR SESTANJ

RAID FLASH DATA RECOVERY COOKBOOK

This page is intentionally blank

Igor Sestanj

NAND Flash Data Recovery Cookbook

Part of a joint effort research and development project of My Data Recovery Lab (USA), Data Solutions d.o.o. (Serbia) and Advanced Data Recovery Analytics (USA)

1. Introduction

- 1.1. Flash memory
  - 1.1.1 NAND
  - 1.1.2 NOR
  - 1.1.3 Facts

- 1.2 Flash Data Recovery

2. Pre-Image Operations

- 2.1 Chip-Off or JTAG
  - 2.1.1 Standard Packages
    - 2.1.1.1 TSOP Signals
    - 2.1.1.2 TLGA Signals
    - 2.1.1.3 BGA Signals
  - 2.1.2. Signal Tracing
    - 2.1.2.1 The NAND Flash Interface
    - 2.1.2.2 JTAG Signals
    - 2.1.2.3 eMMC Signals
    - 2.1.2 4 USB Signals
    - 2.1.2.5 Monolith Signals
    - 2.1.2.6 Solid-state drives Techno Signals
    - 2.1.2.7 oneNAND Signals

- 2.2 Removing memory chips form the PCB (If needed)

  - 2.2.1 Soldering operations
  - 2.2.2 JTAG pads
  - 2.2.3 TSOP
  - 2.2.4 TLGA
  - 2.2.5 BGA
  - 2.2.6 Monolith
  - 2.2.7 USB Flash
  - 2.2.8 Memory Cards
  - 2.2.9 Solid-state drives
  - 2.2.10 EMMC
  - 2.2.11 oneNAND

3. Physical Image Reading

- 3.1 DUMP creation via JTAG
- 3.2 DUMP creation using Chip-Off
  - 3.2.1 Protocol parameters
  - 3.2.2 Bit error analysis (NAND direct access mode) and power adjustment for bit error minimization
  - 3.2.3 Bad columns analysis and removal
  - 3.2.4 ECC detection
  - 3.2.5 Physical images extraction (dump reading)

4. Virtual image Creation

- 4.1 Physical image structure analysis and description

- 4.2 Data transformations: Inversion and Scrambling (XOR) analysis
  - 4.2.1 Inversion
  - 4.2.2 Scrambling
  - 4.2.3 Byte combination
  - 4.2.4 Memory Modem

- 4.3 Data Organizations in NAND memory
  - 4.3.1 Crystal Geometry Parameters
    - 4.3.1.1 Blocks
    - 4.3.1.2 Pages
    - 4.3.1.3 Data Area
    - 4.3.1.4 Spare Area
    - 4.3.1.5 Logical Block Number
    - 4.3.1.6 Logical Page Number
    - 4.3.1.7 Block Header
    - 4.3.1.8 ECC
    - 4.3.1.9 Block Write Counter

- 4.4 Virtual page and block allocation analysis
  - 4.4.1 Virtual Block Allocation
    - 4.4.1.1Sequential
    - 4.4.1.2 Parallel
    - 4.4.1.3 Combined
    - 4.4.1.4 Spare area analysis

- 4.5 Flash Translation Layer
  - 4.5.1 Block Mapping
  - 4.5.2 Other Functions
  - 4.5.3 Translation table analysis and creation
  - 4.5.4 Block sorting and filtering.
  - 4.5.5 Analysis of LBN sequence integrity

1. Introduction

Welcome readers and all those interested,

Throughout most of my professional career I have been involved with data recovery. Doing mostly recovery of hard disk drives and RAIDs. From late 2011 I started my involvement with Advanced Data Recovery Analytics where we are trying to estimate possibilities for recovery with the power of both statistics and analysis. This is a very different approach to the traditional one. The traditional approach involves in-lab diagnostics, however, we discovered that numbers are saying a lot!

Sometime in January 2013 it became obvious that we had no valid data regarding the possibility of recovering data from digital storage media. It was at this point that NAND devices started flooding data recovery companies worldwide. The reason was obvious! NAND devices break and they malfunction! This was a relatively new field in the industry and most of us were simply not familiar with it but in many cases clients were not interested in recovery if the price was over $50.

Most engineers and technicians did not expect the transition from HDD to NAND to happen so fast and quite frankly, most of us didn't really know what to do. We were all keeping an eye on Ace Lab and their support, who at the time were also without a solution. In other words, in some cases … no one knew how to approach the problem!

Rusolut and to some extent Soft Center are doing a great job filling this gap. Forums like HDD Guru and others revealed that a few members were way ahead the others, including myself. So, I got busy.

For some, NAND flash memory data recovery can easily be represented as an assembly of separate, sometimes encrypted, raw data into a RAID-like structure. For others it is an enigma. The process of recovery is at times tedious. It involves techniques which are very different from the ones we traditionally use on hard disk drives. NAND flash data recovery involves no clean-room, no precise mechanics whatsoever and yet requires advanced skills like soldering, data structure, file systems and mathematics.

This guide should help you understand the concept of NAND. The rest is up to your ability to get your mind out of the box and resolve the problem efficiently.

Think of this guide as a diary, my notes from wastelands of flash negative-and data storage media. There are still question which need answering but this publication can be considered as a completion of part one in my NAND flash memory research paper.

I hope it will help you in your work as much as it was helpful in mine.

Igor Sestanj

Project manager, Advanced Data Recovery Analytics

1.1 Flash memory

Flash memory is like a mouthful of *electrically erasable programmable read-only memory (EEPROM)* but far more economical than traditional EEPROM devices. Generally EEPROMs are organized as arrays of floating-gate transistors. NAND Flash memory can be seen as a sort of an opposite of RAM (Random Access Memory), which is a type of fast, volatile memory used for storing data temporarily. Flash memory was invented by Dr. Fujio Masuoka for Toshiba in 1984.

Types of flash memory:

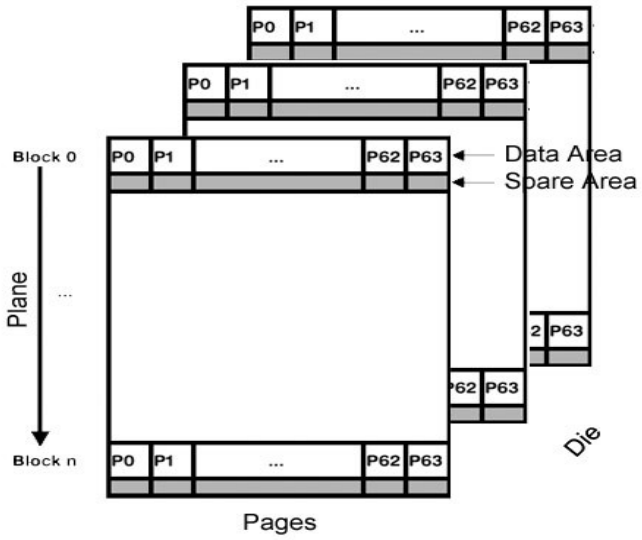
- NAND
- NOR

1.1.1. NAND

NAND gate is a logic gate which produces an output that is false only if all its inputs are true; thus its output is a complement of the AND gate. It is made using transistors. If you remember De Morgan's theorem, AB=A+B, a NAND gate is equivalent to inverters followed by an OR gate. **(1)**

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Let’s have a look at this in more detail and see how NAND actually works. First of all, we need to know our way around the different entities on a flash chip (or “package“), which are; the die (or crystal), the plane, the block and the page.



The package is the memory chip. Each package contains one or more dies (for example one, two, or four). The die is the smallest unit that can independently execute commands or report status. Each die contains one or more planes (usually one or two). Although with some restrictions, commands can be send to both planes in the same time.

Further, each plane contains a number of blocks, which are the smallest unit that can be erased. Which is important!

The last but not least, is the fact that each block contains a number of pages, which are the smallest unit that can be written to. All these entities have their place in very complex math running in the background.

In order to recover data correctly one must find the correct scheme which was applied to NAND memory. The main reason NAND became such a popular data storage medium is the price of memory itself. When memory controllers got cheap and powerful enough flash NAND memory became one of the fastest growing technologies within the data storage industry.

Today, the NAND type of memory is primarily used in:

1. USB Flash drive
2. Compact Flash Card
3. Secure Digital (SD), mini SD, microSD
4. Smart Media Card
5. SDHC Flash Card
6. Multimedia card(MMC)
7. Memory stick (Duo, Pro, HG, Micro)
8. XD-Picture Card
9. Solid-state drives
10. Mobile devices
11. eMMC
12. oneNAND
13. Embedded platforms
14. Enterprise storage
15. more...

The impressive list above is the reason why we will be dealing with NAND memory in this guide!

1.1.2. NOR

The other type of flash memory is the NOR. The NOR gate is a digital logic gate that implements logical NOR - it behaves according to the truth table to the right. A HIGH output (1) results if both the inputs to the gate are LOW (0); if one or both input is HIGH (1), a LOW output (0) results. NOR is the result of the negation of the OR operator. It can also be seen as an AND gate with all the inputs inverted.

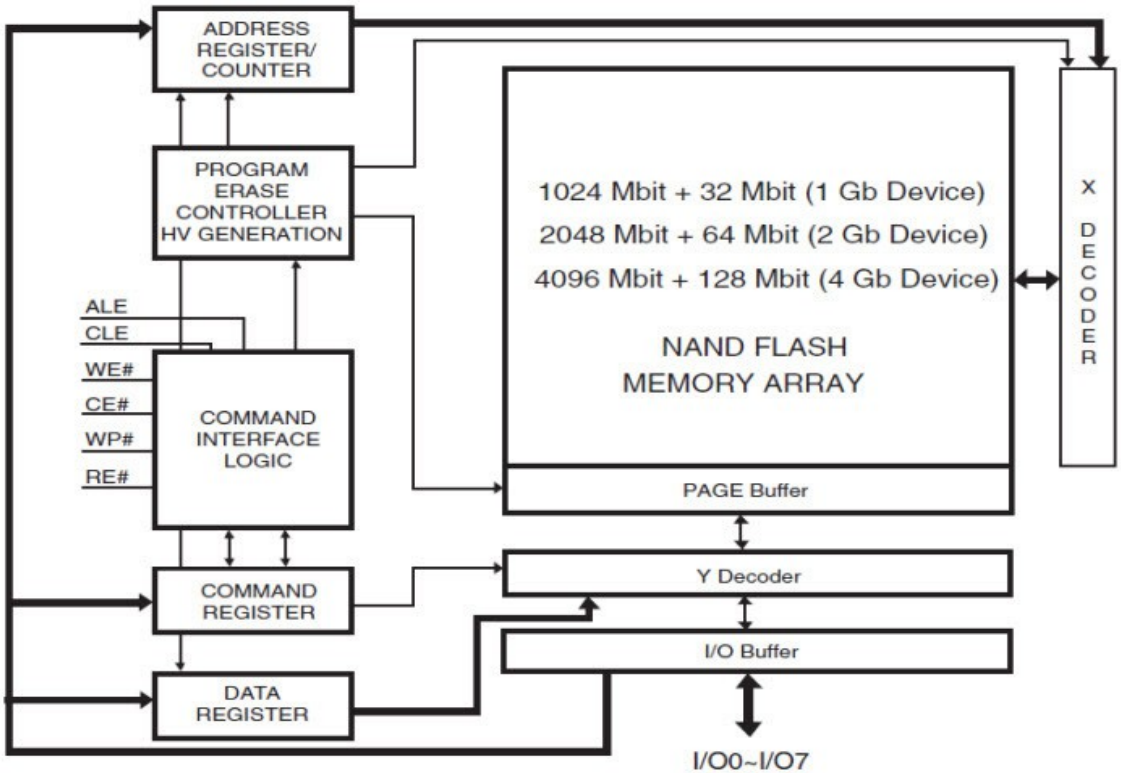
INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

NOR is a functionally complete operation—NOR gates can be combined to generate any other logical function. **(2)**

By contrast, the OR operator is monotonic as it can only change LOW to HIGH but not vice-versa. Mostly in use as a ROM chip which is not the subject of this guide.

1.1.3 Facts

NAND flash memory requires less chip area per cell, allowing greater storage density and lower cost per bit than NOR flash. NAND Flash also has up to ten times greater endurance compared to NOR flash. However, NAND flash does not provide a random-access external address bus. In NAND Flash, data must be read on a block-wise basis, with typical block sizes of hundreds to thousands of bits. This makes NAND flash unsuitable as a replacement for ROM, since most microprocessors and micro controller chips required byte-level random access.



Example of NAND

1.2. NAND Flash Data Recovery

World leading data recovery R&D establishments are listed below. All of these establishments operate as reverse engineering projects. It is not within my knowledge whether their involvement is recognized with the rest of the community such as ONFI or JEDEC?

Rusolut http://rusolut.com/
Ace Lab http://www.acelaboratory.com/
Soft Center http://www.soft-center.ru/
UFED, XRY (mobile forensics)

Rusolut and Ace Lab have blogs which are very useful and one can learn a great deal about NAND flash.

According to several R&D establishments a recovery of data for NAND flash based memory should include:

1. Pre-Image Operations
2. Reading Physical Image (DUMP)
3. Creating Virtual Image
4. Assemble of Logical Image
5. Recovery of data from the Logical Image

This is the logic or general idea we are following in this paper as well.

Apart from the general procedure it is important to distinguish which type of NAND is in use on the device you are trying to recover data from. There are three basic types we mostly find:

1. rawNAND (SSD, USB flash, some mobile devices...)
2. oneNAND (mobile devices, some USB flash drives...)
3. eMMC (mobile devices, some USB Flash, memory cards, some laptops...)

It is crucial to have detailed schematics and ONFI/JEDEC standardization diagrams on hand in order to get the best results. The main reason lies with the fact that different types of NAND have different properties. The rawNAND architecture uses a separate memory chip as a driver, translator and for error correction while eMMC have everything inside one chip making it possible for this type of architecture to act as a block device. The one in the middle is indeed in the middle as it has a separate controller or processor for driver and translation purposes as well as build-in ECC. So, let's say that the chances are one in a thousand that a flash drive you will work on has eMMC or oneNAND on it? For example, when eMMC chips are used in flash drives, they have been sourced as faulty or refurbished, and the eMMC is connected to PCB using the direct NAND pads. It is all based on what the manufacturer have in-stock.

The NAND memory chip market these days has it all, from superb quality chips to trash. It all sells, depending on who the buyer is. Like everything in this world, flash memory chips come with a price tag, as an example, more expensive NAND memory chips are not very likely to

be installed on cheap USB flash drives, however you will find an SLC chip on some enterprise SSDs.

Flash data recovery is very different from what most of us in the industry became used to. Although important, it is less about the actual precision than the ability to mostly “read and sometimes write” on the bit level. Knowing Linux becomes more relevant these days and although big players still deliver their tools for Windows platforms, Android was a big game changer for Unix like platforms. File systems are also becoming very different but we will discuss these later. For recovery purposes we have two basic conditions:

- Working state – device can be powered on and reaches the recovery state (use software)
- Inoperable – device does not power up and memory chip has no damage (use information from this guide)

It is sometimes especially useful on USB flash storage media to transplant NAND memory chips to a donor USB drive. In this case one needs to have a donor board with the exact same controller. For other devices this technique is not very useful because of the complexity of each device. The method is comparable to PCB replacement on very old hard drives, like the Quantum LCT series.
In case you want to try this approach out see 2.2 for more details. This approach is very effective because if it is done correctly you will have data right away. We are doing some research to find a possible application of this approach with SSDs, however this method is not within the scope of this paper.

Below is a list of manufacturers of flash memory controllers for various flash memory devices like SSDs, USB flash drives, SD cards, and CompactFlash cards. Some manufacturers are missing due to mergers with some of those listed below (like Fusion-io now San Disk):

- Greenliant Systems, USA
- Indilinx, South Korea
- Intel, USA
- Jmicron, Taiwan
- Marvell, USA
- Phison, Taiwan
- Samsung, South Korea
- SandForce, USA
- SanDisk, USA
- Silicon Motion, Taiwan
- STEC, USA
- Hyperstone, Germany
- Toshiba, Japan

Other good resources are Forensics Focus and TEEL technology for those who are searching for good forensic tools, forensic software, equipment, forms, community, etc. **(20)**

2. Pre-Image Operations

In the data recovery industry the practices of Chip-Off and JTAG have become topics of great interest. Primarily for devices that are damaged or locked with an encryption scheme that are beyond the abilities of software tools to bypass or crack. The chip-off and JTAG methods are among the alternative solutions for ones looking to gain access to the memory chip itself.

But before we try anything we should search the web for data sheets, pin-outs, etc.

A useful website to search for data-sheets is http://component.iiic.cc/

2.1. Chip-Off or JTAG

The "chip-off" method describes the practice of removing a memory chip(s) from a circuit board and reading them via serial to USB interface. Since memory chips are in most cases tested and debugged with the “JTAG”, this alternative method becomes more popular as NAND memory has seen greater use on mobile devices. The term JTAG (Joint Test Action Group) is the acronym and the name of an IEEE group that set the standards for what eventually became the 1149.1 Standard Test Access Port and Boundary-Scan Architecture.

What does this mean?

The JTAG group established a set of standards for testing wire-line interconnects on printed circuit boards. Today, these ports are used for testing integrated circuits, and are considered the common test and debug interface for mobile devices and digital products such as memory cards, USB flash media, solid-state drives, eMMC, etc.

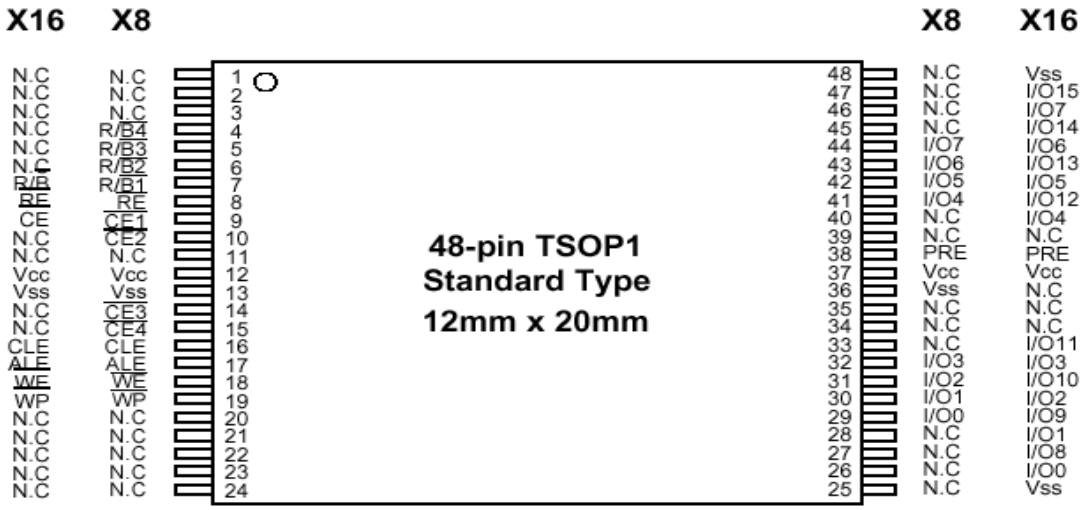
These methods have been in practice for over a decade now in the integrated circuit (IC) programming and testing fields. However, the ability of these low-level access methods to acquire raw data from the memory chip for anyone involved in data recovery or computer forensics is critical. It offers another way, a backdoor to access and acquire the raw data from the memory chip. This will also be cover in this paper.

2.1.1 Standard Packages

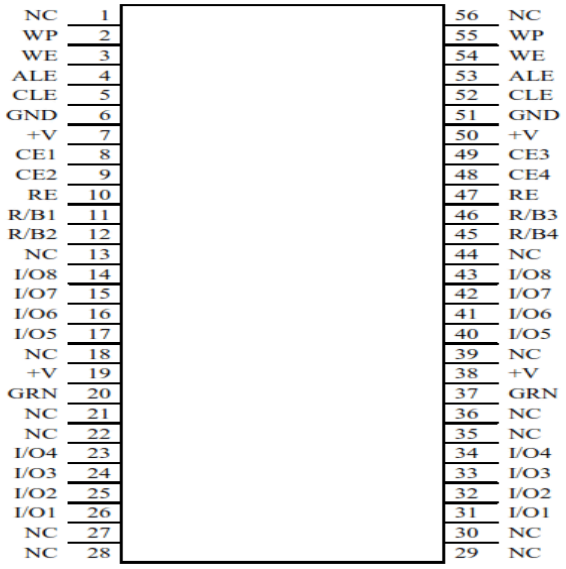
There are many standards when it comes to chip packages. I will be dealing here only with the most common few we find every day in this business.

2.1.1.1 TSOP Signals

Thin Small Outline Package, or TSOP is a type of surface mount IC package. They are very low-profile (about 1mm) and have tight lead spacing (as low as 0.5mm). TSOPs are rectangular in shape and come in two varieties: Type I and Type II. For the purpose of data recovery we will be dealing with: TSOP Type I ICP.



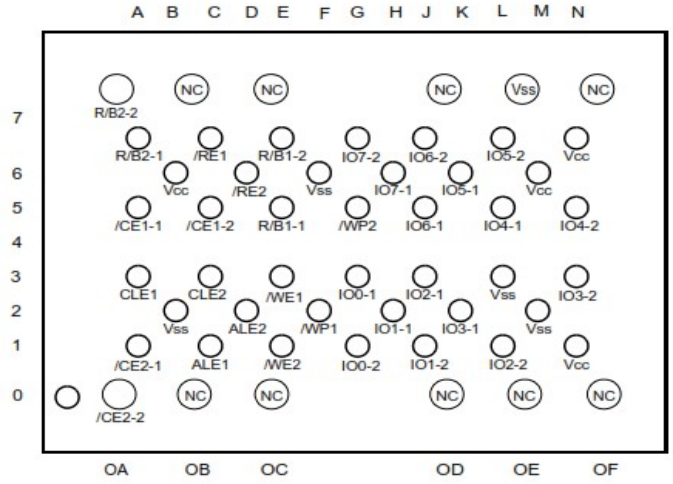
TSOP48 – Pins =48, W (Width) = 12mm L (Length) = 18,4mm P (Pitch) = 0.5mm



TSOP56 – Pins = 56, W (Width) =14mm L (Length) = 18,4mm P (Pitch) = 0,5mm

2.1.1.2 TLGA

The TLGA package is very similar to BGA with the difference that TLGA is basically an LGA with Land-pads ready to be soldered the same way as BGA.



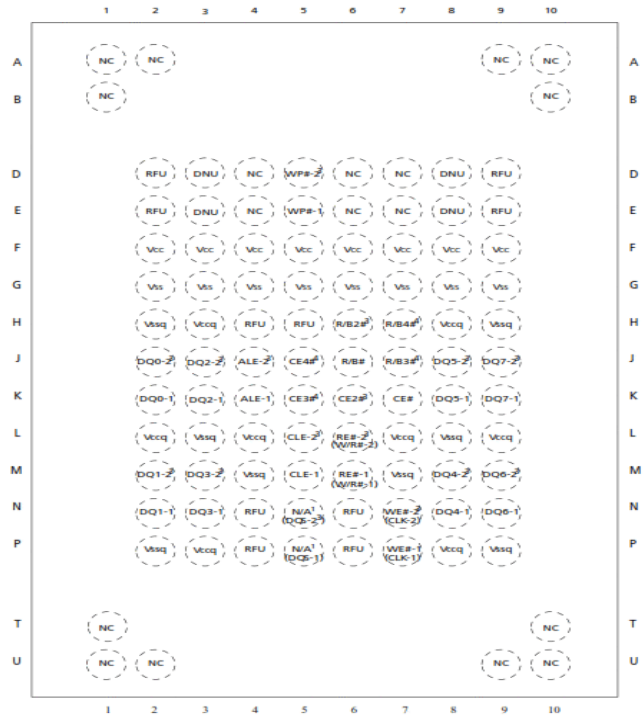
TLGA52

2.1.1.3 Ball Grid Array (BGA)

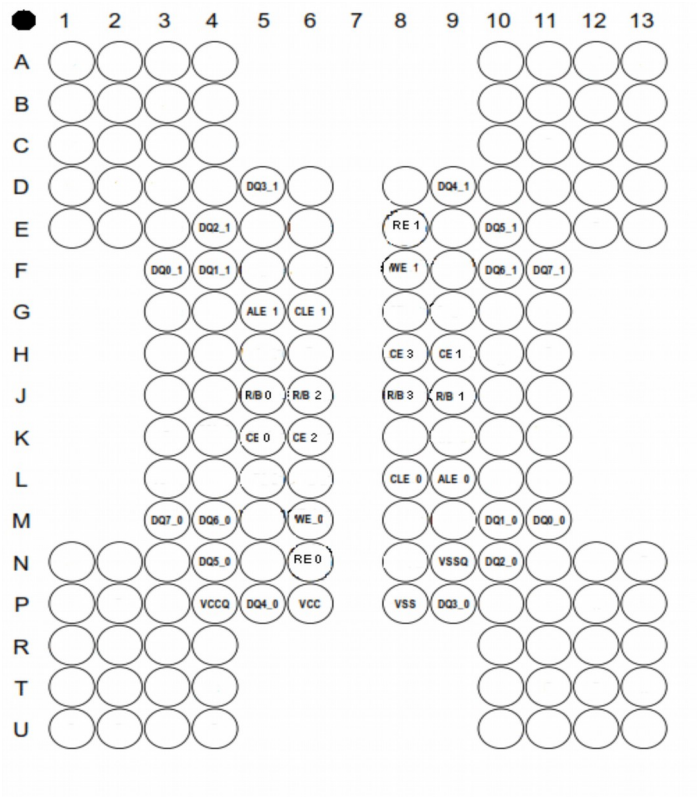
The Ball Grid Array (BGA) has been around since the 1980's but the pin pitch started out with 1.5 mm and then quickly went to 1.27 mm (50 mils) for about 15 years. Then in the late 1990's, the 1 mm pitch BGA was introduced and every couple of years a smaller pin pitch was introduced.

Today 0.4 mm pitch BGA's are in every cell phone and 0.3 mm pitch BGA's are the next generation.

In use with flash memory: BGA100, BGA152, BGA154, BGA224 and BGA 161, 165, 169, 186, 221 are widely in use for eMMC.



BGA 100

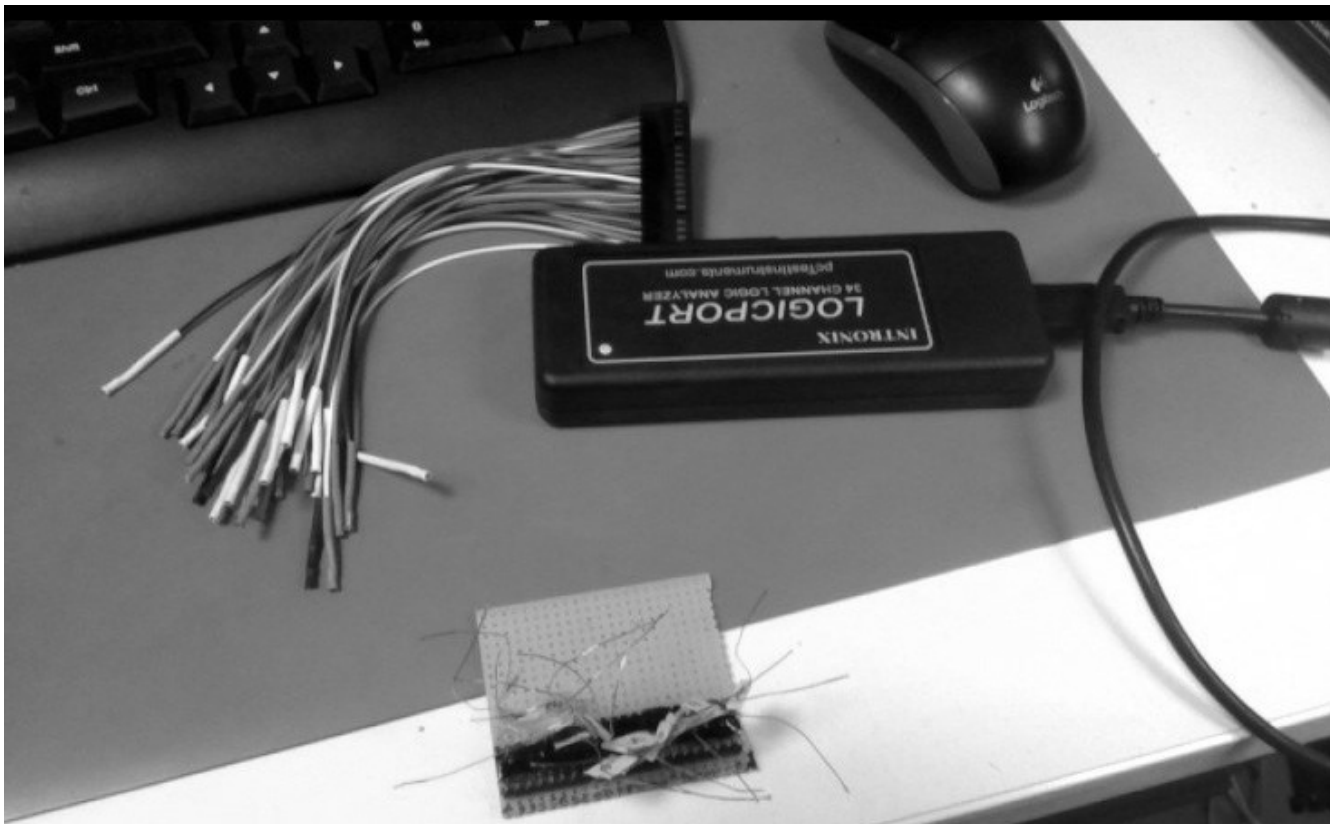


BGA 152 with dual 8-bit data access pinouts

2.1.2. Signal tracing

Signal tracing should be done with help of the logic analyzer. However, depending on which way you want to acquire data from the memory chip these places on the actual motherboard/chip/PCB pads need to be well understood and found. Meaning, that if you decide to read data from the chip itself, information about these actual pins will likely be available through a standardization and chip package (set out in more detail below).

Working with the logic / digital analyzer is especially important when you are recovering data from some monolithic flash storage devices.



Logic Analyzer and simple wire adapter

The Logic analyzer is a useful device because it can take snapshots of multiple data channels making it possible to distinguish one signal from another. The process is fairly simple when you have the exact same drive. You look for the controller name and model since each one will load its special set of hex values (called controller signatures) first. This signature can be altered or otherwise transformed (Inversion/XOR) so you need to keep this in mind. For details see 4.2. Converted to binary data you look for ones and zeros to find which signal is which. See 2.1.2.1 for details.

In case of JTAG most of the time, pads will be clearly visible on the PCB. Some storage devices have those pads disabled, some hidden and in some cases there are none and you need to use some other pads and make them available to solder wires to it. It is a fair statement to say that most of the present day flash storage devices are JTAG compatible or that there is some way to read raw data from the memory chip using a serial connection other than a traditional NAND flash interface. In many cases popular “jtagging” is easier than traditional NAND reading.

2.1.2.1 The NAND Flash Interface

The Chip-Off approach is traditionally related to NAND Flash Interface. In order to read any memory chip through the NAND reader and serial to USB connection it is necessary to trace the following signals (or have them located according to the IC package):

NAND flash

Pin	Driver (Device)
CE	CHIP ENABLE The CE input enables the device. Signal is active low. If the signal is inactive, device is in standby mode.
WE	WRITE ENABLE The WE input controls writes to the I/O port. Commands, address and data are latched on the rising edge of the WE pulse.
RE	READ ENABLE The RE input is the serial data-out control. When active (low) the device outputs data.
CLE	COMMAND LATCH ENABLE This pin should be low, when writing commands to the command register.
ALE	ADDRESS LATCH ENABLE When active, an address can be written.
WP	WRITE PROTECT Typically connected to VCC (recommended), but may also be connected to port pin.
R/B	READY/BUSY OUTPUT The R/B output indicates the status of the device operation. When low, it indicates that a program, erase or read operation is in process. It returns to high state when the operation is completed. It is an open drain output. Should be connected to a port pin with pull-up. If available a port pin which can trigger an interrupt should be used.
I/O0 - I/O7	DATA INPUTS/OUTPUTS The I/O pins are used to input command, address and data, and to output data during read operations.
I/O8 - I/O15	DATA INPUTS/OUTPUTS I/O8 - I/O15 16-bit flashes only.

DataFlash

DataFlash chips are commonly used when low pin count and easy data transfer are required. DataFlash devices use the following pins:

Pin	Meaning
CS	ChipSelect This pin selects the DataFlash device. The device is selected, when CS pin is driven low.
SCLK	Serial Clock The SCLK pin is an input-only pin and is used to control the flow of data to and from the DataFlash. Data is always clocked into the device on the rising edge of SCLK and clocked out of the device on the falling edge of SCLK.
SI	Serial Data In The SI pin is an input-only pin and is used to transfer data into the device. The SI pin is used for all data input including opcodes and address sequences.
SO	Serial Data Out This SO pin is an output pin and is used to transfer data serially out of the device.

- Data transfer width is 8 bit.
- Chip Select (CS) sets the card active at low-level and inactive at high level.
- Clock signal must be generated by the target system. The serial flash chips are always in slave mode.

Bit order requires most significant bit (MSB) to be sent out first.

- GND – **Ground -** Test Input (grounded)
- Vcc - **Positive Supply** (core)
- Vccq - **Positive Supply** (I/O)
- Vss - **Negative supply** (ground) **(3)**

The basic interface is fairly simple. The I/O is tri-stated. The Command Latch Enable (CLE) pin and the Address Latch Enable (ALE) pin act as multiplexer select pins by selecting which internal register is connected to the external I/O pins. There are only three valid states as shown in the table below:

Ale	CLE	Register Selected
0	0	Data Register
0	1	Command Register
1	0	Address register
1	1	Not defined

The key to understanding how the NAND flash operates is the realization that in the NAND flash, the read and program operation takes place on a page basis (i.e. 528 bytes at a time for most NAND devices) rather than on a byte or word basis like NOR flash. A page is the size of the data register. The erase operation takes place on a block basis (for most NAND devices, the block size is 32 pages).

There are only three basic operations in a NAND flash: read a page, program a page, and erase a block.

Data is shifted into or out of the NAND Flash register 8 or 16 bits at a time.

Data is output from the data register in a similar fashion by means of the read enable (RE#) signal, which is responsible for outputting the current data and incrementing to the next location. The WE# and RE# clocks can run as fast as 25ns per transfer. When RE# or chip enable (CE#) are not assigned LOW, the output buffers are tri-stated. This combination of CE# and RE# activates the output buffers, enabling NAND Flash to share the data bus with other types of memory, such as NOR Flash, SRAM, or DRAM. This feature is sometimes referred to as “chip enable don’t care.” The primary purpose of this reference is to differentiate very old NAND flash devices, which require CE# to be asserted for the entire cycle.

All NAND flash operations are initiated by issuing a command cycle. This is accomplished by placing the command on I/O [7:0], driving CE# LOW and CLE HIGH, then issuing a WE# clock. Commands, addresses, and data are clocked into the NAND Flash device on the rising edge of WE# (see Figure and Table below).

Most commands require a number of address cycles followed by a second command cycle. With the exception of the RESET and READ STATUS commands. New commands should not be issued when the device is busy (see Figure and Table one following page).

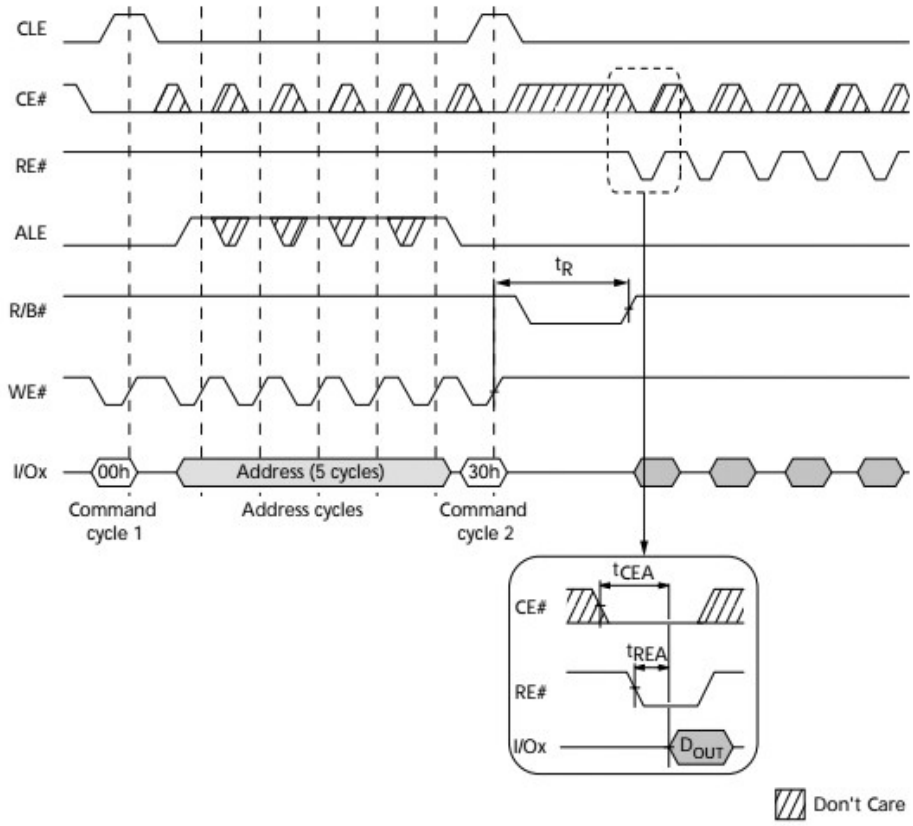
NAND Flash Commands

NAND Flash Commands – NFC are basically listed in this same table.

When any NAND flash command is issued, CE# and ALE must be LOW, CLE must be assigned, and write clocks (WE#) must be provided.

When any NAND flash address is issued, CE# and CLE must be LOW, ALE must be assigned, and write clocks (WE#) must be provided. While the device is busy, only two commands can be issued: RESET and READ STATUS.

Command Cycles for NAND Flash Operations



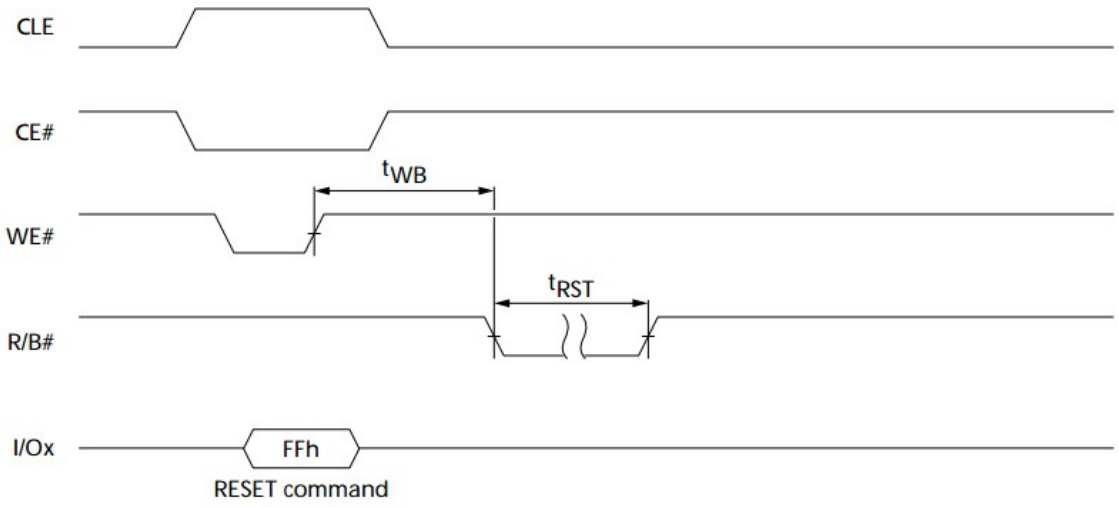
Command Cycles and Address Cycles

Command	Command Cycle 1	Number of Address Cycles	Data Cycles Required[1]	Command Cycle 2	Valid During Busy
READ PAGE	00h	5	No	30h	No
READ PAGE CACHE SEQUENTIAL	31h	-	No	-	No
READ PAGE CACHE SEQUENTIAL LAST	3Fh	-	No	-	No
READ for INTERNAL DATA MOVE	00h	5	No	35h	No
RANDOM DATA READ	05h	2	No	E0h	No
READ ID	90h	1	No	-	No
READ STATUS	70h	-	No	-	Yes
PROGRAM PAGE	80h	5	Yes	10h	No
PROGRAM PAGE CACHE	80h	5	Yes	15h	No
PROGRAM for INTERNAL DATA MOVE	85h	5	Optional	10h	No
RANDOM DATA INPUT	85h	2	Yes	-	No
ERASE BLOCK	60h	3	No	D0h	No
RESET	FFh	-	No	-	Yes

Reset

The simplest NAND Flash command is the RESET (FFh) command. The RESET command does not require an address or subsequent cycle(s). Simply assign CLE and issue a write clock with FFh on the data bus, and a RESET operation is performed. This RESET command must be issued immediately following power-up, and prior to issuing any other command.

RESET is one of two commands that can be issued while the NAND flash device is busy. If the device is busy processing a previous command, issuing a RESET command aborts the previous operation. If the previous operation was an ERASE or PROGRAM command, issuing a RESET command aborts the command prematurely, and the desired operation does not complete.



RESET COMMAND

Read ID

The READ ID (90h) command requires one dummy address cycle (00h), but it does not require a second command cycle. After the command and dummy addresses are assigned, the ID data can be read out by keeping CLE and ALE LOW and toggling the RE# signal for each byte of ID.

	Option	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0	Value[1]
Byte 0 – Manufacturer ID										
Manufacturer	Micron	0	0	1	0	1	1	0	0	2Ch
Byte 1 – Device ID										
MT29F2G08AAD	2Gb, x8, 3V	1	1	0	1	1	0	1	0	DAh
MT29F2G16AAD	2Gb, x8, 3V	1	1	0	0	1	0	1	0	CAh
MT29F2G08ABD	2Gb, x8, 1.8V	1	0	1	0	1	0	1	0	AAh
MT29F2G16ABD	2Gb, x16, 1.8V	1	0	1	1	1	0	1	0	BAh
Byte 2										
Number of die per CE#	1							0	0	00b
Cell type	SLC					0	0			00b
Number of simultaneously programmed pages	1			0	0					01b
Interleaved operations between multiple die	Not supported		0							0b
Cache programming	Supported	1								1b
Byte value	MT29F2Gxxxxx	1	0	0	0	0	0	0	0	80h
Byte 3										
Page size	2KB							0	1	01b
Spare area size (bytes)	64B						1			1b
Block size (w/o spare)	128KB			0	1					01b
Organization	x8		0							0b
	x16		1							1b
Serial access (MIN)	25ns	1				0				1xxxb
Serial access (MIN)	35ns	0				0				0xxx0b
Byte value	MT29F2G08AAD	1	0	0	1	0	1	0	1	95h
	MT29F2G16AAD	1	1	0	1	0	1	0	1	D5h
Byte value	MT29F2G08ABD	0	0	0	1	0	1	0	1	15h
	MT29F2G16ABD	0	1	0	1	0	1	0	1	55h
Byte 4										
Reserved								0	0	00b
Planes per CE#	1					0	0			00b
Plane size	2Gb		1	0	1					101b
Reserved		0								0b
Byte value	MT29F2Gxx	0	1	0	1	0	0	0	0	50h

Table shows the format of the 5-byte response on Micron Device

When it comes to reading raw pages into a DUMP, one needs to have a physical reader attached to a computer and to have software capable of issuing NAND flash Commands. Most technicians will work within applications such as PC3000 Flash, Flash Extractor, VNR or similar using their native readers and issuing NFC to a memory chip (in command like environment) will not be required because the program does it. However, if you want to have reading done some other way, using Linux for example, being familiar with what I have tried to explain here is a plus. Having said that, playing with your Logic Analyzer issuing NFC to NAND is the way to go!

We will not discuss all NAND Flash Commands here because it is usually enough to use Read ID and have all signals sorted. If not it is not within my scope of knowledge. Please share!?

There are more details in each IC package description provided further in this document.

2.1.2.2 JTAG Signals

The JTAG connector pins are:

1. TDI (Test Data In)
2. TDO (Test Data Out)
3. TCK (Test Clock)
4. TMS (Test Mode Select)
5. TRST (Test Reset) optional.
6. GND

Reduced pin count JTAG uses only two wires, a clock wire and a data wire. This is defined as part of the IEEE 1149.7 standard. The connector pins are:

1. TMSC (Test Serial Data)
2. TCK (Test Clock)

2.1.2.3 EMMC Signals

eMMC stands for embedded Multimedia Card. It's basically an MMC very much like an SD card, embedded onto the device's PCB and is a way to provide cheap internal storage. The eMMC device also has a controller that makes the eMMC bootable, so it can be used as a system drive inside mobile devices and laptops. However, it doesn't have the flash translation layer for multiple memory chips, a special high-quality hardware to handle page to block translation, wear-leveling or bad block. This is done with the flash file system. Many times eMMC is possible to recover (data) either by reading it via the NAND interface and later creating a virtual image out of it or tracing native eMMC signals (if the chip is working) and read its content like any block device. In the last case, firmware has to be able to boot up the device. Signals relevant to data recovery are given below:

1. GND,
2. CMD,
3. CLK,
4. DAT0,
5. VCC 3.3V,
6. VCCQ 1.8V

If the chip is not working that way you can try to trace its native NAND pads and attempt to read it that way. Also good to mention here are so called eMCP chip platforms which may consist of DDR and eMMC within the same chip.

2.1.2.4 USB Signals

There are a few ways these signals are useful to trace, in case the USB connector is broken on a flash drive, for JTAG connections or to have access to a mobile device.

- USB- USB+ - USB Bus Data = these pins connect to the USB bus data signals.
- LOOPFLTR – Transceiver Filter = this pin provides the ability to supplement the internal filtering of the transceiver with an external network, if required.
- RBIAS – Transceiver Bias = A precision 9.09K resistor is attached from ground to this pin to set the transceiver's internal bias currents.
- RTERM – Termination Resistor = A precision 1.5K resistor is attached to this pin from a 3.3V supply.
- FSFS+ - Full Speed USB Data = U These pins connect to the USB- and USB+ pins through 31.6 ohm series resistors.
- POWER, GROUNDS, AND NO CONNECTS (may be important for smartphone recovery)
- VDD +2.5V Core power
- VDDIO +3.3V I/O power
- VDDP +2.5 Analog power
- VSSP Analog Ground Reference
- NAND flash Chip Select Signal - CS [7:0] - OPU8; these pins can be used to chip enable the NAND flash devices when multiple NAND flash devices are used.
- RESET input signal. RESET is a low active signal used by the system to reset the chip. The active low pulse is usually at least 100ns wide?
- TEST Input - TEST [0:1] - these signals are used for testing the chip and are normally unconnected.

2.1.2.5 Monolith

This type of package basically requires chip to be read from the PCB itself. Signals useful for data recovery are:

1. Vcc,
2. GND,
3. CLE,
4. RE,
5. CE,
6. ALE,
7. WP,
8. WE,
9. RB,
10. D0-D7

Getting to these requires the use of logic analyzer or pad layouts, which can be obtained via support pages from AceLab or Rusolut. (May require credentials)

2.1.2.6 Solid-state drives

Although they use the same type of NAND flash memory chips as USB flash drives SSDs (solid-state drives) have more chips which tend to be faster, better-quality chips organized with the controller and firmware that contains more advanced features. For example, the SSD controllers read and write over all memory chips in the drive, so it’s not limited by the speed of an individual chip as much.

The controller resembles the RAID configuration as it uses multiple chips in parallel to speed things up. When you write to a solid-state drive, the drive might actually be writing to twenty different NAND Flash chips at once. The solid-state drive’s firmware also performs wear-leveling operations to ensure data you write to the drive is spread across the physical drive evenly to prevent the flash memory from wearing out. The controller presents the memory to the computer in a consistent order so the computer behaves normally, but the drive is shuffling things around in the background.

SSDs also support advanced features like TRIM to speed things up. There’s no real need for an “SSD optimization” utility because the SSD’s firmware is automatically optimizing the drive, shuffling data around for better performance. A solid-state drive is also typically connected to the computer over a SATA 3, mSATA, or SATA Express interface, which will be faster than the interfaces available to a common flash drive or memory cards.

Finding pads necessary to recover data from SSD are usually referred to as techno pads. These are useful when one is using Ace Lab's PC-3000 SSD. Although there is no universal way to obtain access to SSD via these pads they provide a proper backdoor to the drive's sectors.

2.1.2.6 oneNAND

The oneNAND is an advanced generation, high-performance NAND-based Flash memory. It integrates an on-chip NAND flash array memory with two independent data Buffers, boot RAM buffer, a page Buffer for the flash array, and a one-time-programmable block. The combination of these memory areas enables high-speed pipelining of reads from the host BufferRAM Page Buffer and NAND Flash Array memory. The OneNAND also includes a Boot RAM and boot loader. This enables the device to efficiently load boot code at device startup from the NAND array without the need for an off-chip boot device. One block of the NAND array is set aside as an OTP memory area. This area, available to the user, can be configured and locked with secured user information. On-chip controller Interfaces enable the device to operate in systems without NAND Host controllers.

2.2. Removing memory chips from the PCB

Removing memory chips from the PCB (if possible, or tracing signals from the media itself) or using a flash reader to get a dump of all data from all chips separately. The IC package includes the following: TSOP48, TLGA52, TSOP56, BGA100, BGA152, BGA154, BGA224, etc.

2.2.1 Soldering Operations

For the most part the soldering operation for flash media requires: soldering station/hot gun, solder wick, flux, tweezers, etc. Soldering a memory chip back to the PCB requires tools such as BGA re-ball soldering stencils, solder balls (solder spheres), solder paste, re-flow ovens and related equipment. In cases where data recovery is only possible with a donor flash controller this equipment and skill, are highly recommended.
For those skilful enough to perform operations; no special tools are required. **(26)**

As stated above possible IC package options are listed on the pages below:

- TSOP 48
- TSOP56
- TLGA52
- BGA100
- BGA152
- BGA224

Other common eMMC BGA IC packages:

- BGA153
- BGA169
- BGA162
- BGA186
- BGA221

For some chips, preparation is relatively easy. For example, the TSOP NAND chip that is commonly found in thumb drives, SD cards, digital voice recorders, digital answering machines, or iPhone 2 and iPhone3G, is an easier type of memory chip to remove and read. It does requires high skill and precision handling, but compared to BGA, TSOP is way easier to work with. This is largely due to the relative standardization of the TSOP memory architecture and pin layout. The TSOP chip has connectors around the outer edge of the chip, which are connected by soldering onto the PCB. Removal from the board, as well as attachment to a chip-reading adapter, is relatively easy.

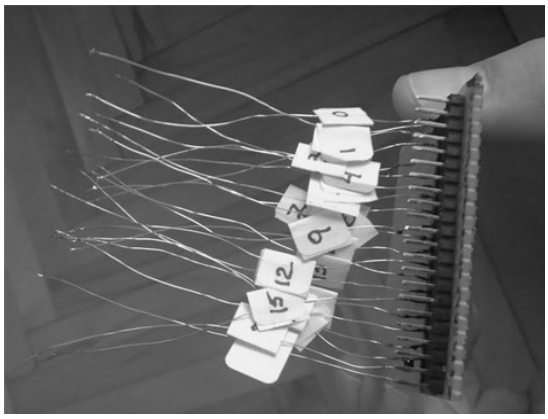
On the other hand, the BGA (Ball Grid Array) chip has multiple connectors on the underside of the chip which are soldered to the device's PCB. In most cases they are secured with epoxy, which makes the task even more challenging.

While TSOP chips are relatively manageable, the connectors on a BGA memory chip require, in many cases, rework through a process known as “re-balling”.

The BGA chips are commonly used in mobile devices and some flash devices, while TSOP can often be found on SSDs and memory cards. Fortunately, the re-balling effort has become less burdensome on newer devices as more advanced soldering techniques have become available and can be applied.
JTAG is an option in which you don't have to mess up with chips but having a device connected to a service port and although slower, one can read everything without removing chips.

2.2.2. JTAG

JTAGging is for many including myself the best of all the options. Why? First of all it is the best way to find out whether the NAND chip (mostly eMMC) is working or not. This rule involves dead devices or sometimes the one in bootloop. Once you have the layout of the JTAG pins, soldering is usually easy. Some devices require an adapter but in most cases you will have pads clearly visible on the PCB itself. In the case of pads not being available, you have to solder thin wires to the component on board. I recommend designing an adapter and using a stereo microscope to connect everything up.



Example of a basic Wire adapter

2.2.3. TSOP

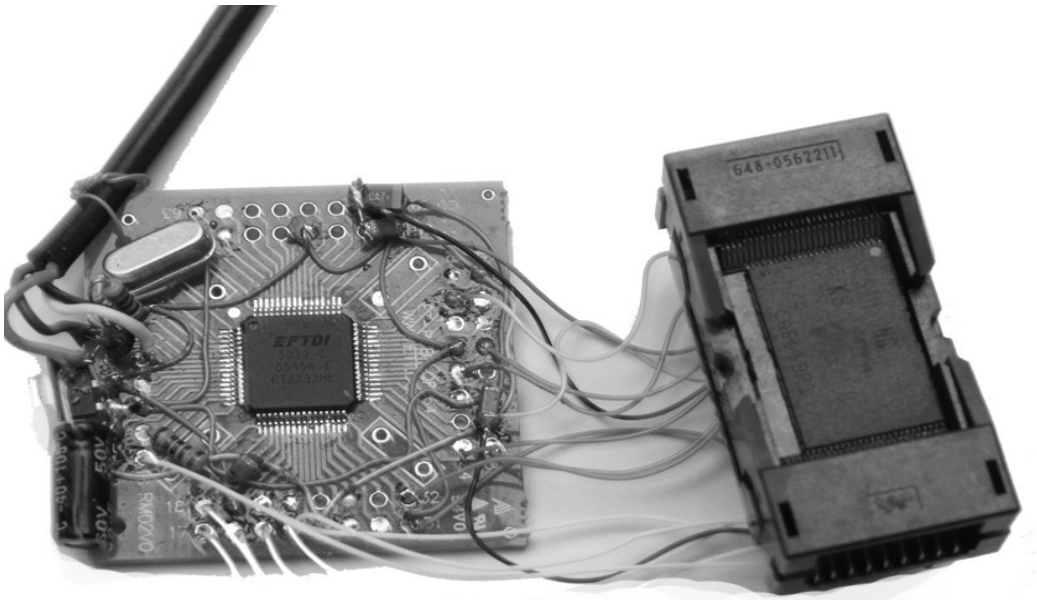
Removing TSOP packages is quite easy and tutorials are available all over the internet. I will include a video in the reference section below on how NAND Flash memory works.
A crucial tool to remove a TSOP package chip off the PCB is a solder wick (a copper braid impregnated with flux). You lay the wick on top of the joint you want to remove solder from. Then apply your solder iron on top of the wick. As the wick heats up the solder will flow from the joint into the wick.

Ideally you should use a hot plate as a pre-heater. This will speed up the heating process.

There is also info on the web for using a toaster oven. I haven't used this though.

On the next page is an example of a reader based on Linux Sprites mod and chip FT2232H along with a TSOP socket reader.

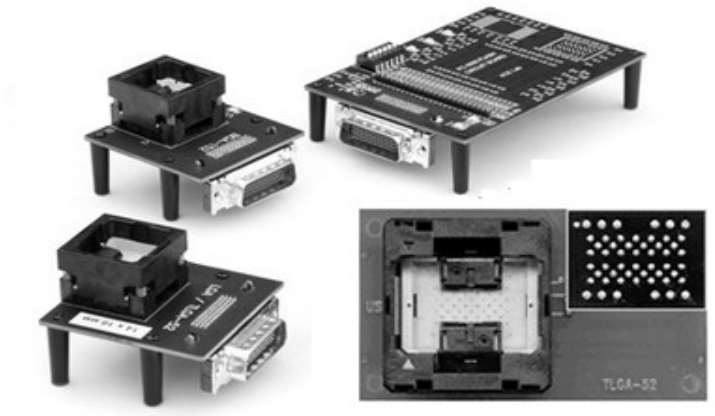
IR rework station with pre-heat plate



Sprites-FT2232H

2.2.4 TLGA

TLGA is (as stated above) very similar to BGA. It is a Grid Array of pins laid down / facing below the chip. Once removed using an IR rework station, contacts should be cleaned and mounted on the proper adapter. Examples from ACE Lab and Soft Center are shown in the image below. Note how similar they are for TLGA and BGA (top left). The one on the top right is for monolithic devices or universal if you like. This last, universal one I recommend buying or making yourself.



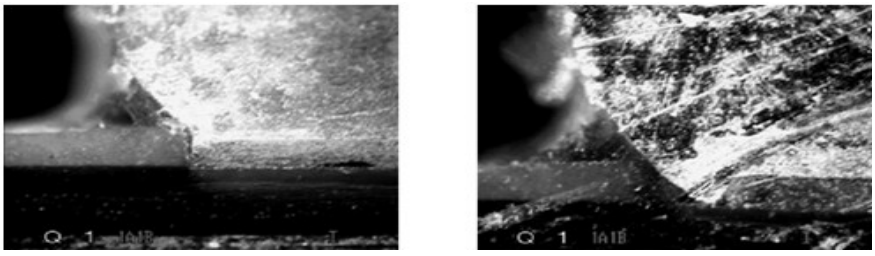
Example of LGA, BGA and monolith adapters

2.2.5 BGA

There are 2 types of BGA Ball leads: collapsing and non-collapsing.

Collapsing - BGA with 0.65 mm pitch and higher. Land-pad is smaller than the Ball size to allow the Ball to collapse around the sides of the Land-pad. This requires a non-solder mask defined Land-pad where the solder mask must be larger than the Land-pad

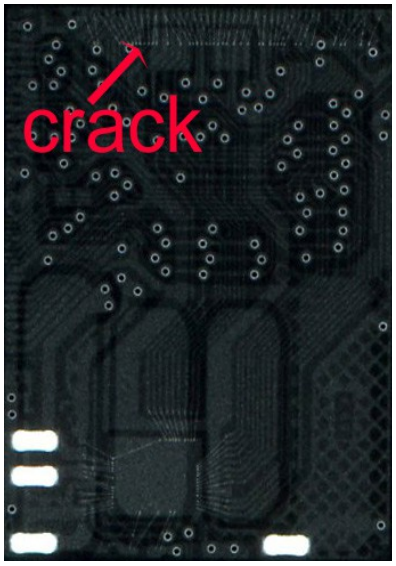
Non-collapsing – BGA with 0.5 mm pitch and smaller, where the Land-pad is larger than the ball to allow for via-in-pad technology and provide an adequate annular ring. The solder mask can be the same size as the Land-pad. In some cases the Land-pad for fine pitch BGA's is solder mask defined where the solder mask encroaches slightly over the Land-pad. This provides protection for any trace routing between the Land-pads. Widely in use with lash memories.



Collapsing and non-collapsing BGA

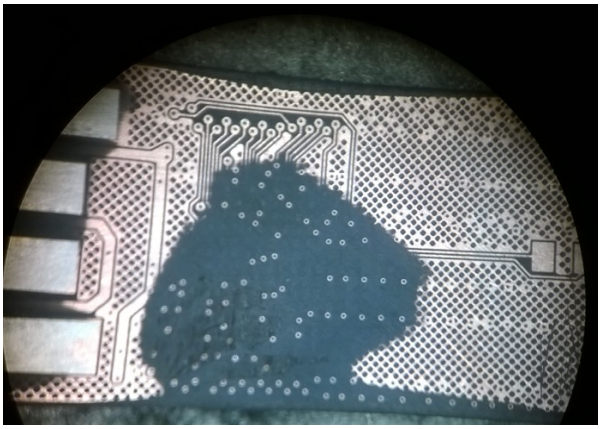
2.2.6 Monolith

1. Remove the protective layer (use sandpaper 2000+ grit)
2. Use the logic analyzer to trace the right pin-out (test-points or signals)
3. Use a flash reader connected directly to test-points and read dump (all crystals/dies).



Example of X-ray view of a cracked monolith flash drive

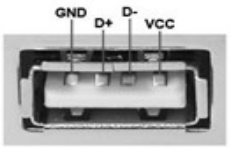
Most flash chips come in ball grid array (BGA) packages, and even the ones that do not are often mounted on a PCB next to other BGA packages. After PCB Assembly, boards with BGA packages are often x-rayed to see if the balls are making proper connections to the proper pad, or if the BGA needs rework. These x-rays can erase programmed bits in a flash chip (convert programmed "0" bits into erased "1" bits). Erased bits ("1" bits) are not affected by x-rays. I recommend using x-rays for research purposes but never on chips with user data. **(27)**
Below you can see a good example of a monolith USB stick destroyed while sanding using way too rough paper for the job (1000 grit). I recommend 2500 grit and flat surface for the best results.



Example of monolith destroyed with too much sanding

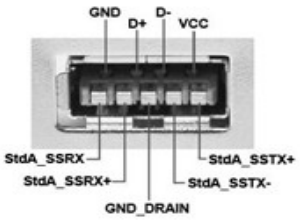
2.2.7 USB flash

High-Speed USB 2.0 A plug pinout



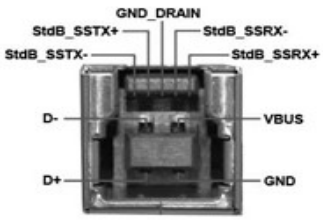
1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
Shell	Shield	Connector Shell

SuperSpeed standard A plug pinout



1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
5	StdA_SSRX-	Blue
6	StdA_SSRX+	Yellow
7	GND_DRAIN	GROUND
8	StdA_SSTX-	Purple
9	StdA_SSTX+	Orange
Shell	Shield	Connector Shell

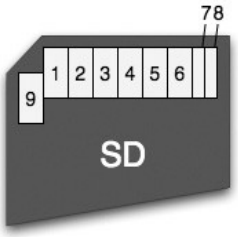
SuperSpeed standard B plug pinout



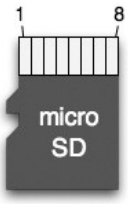
1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
5	StdA_SSTX-	Blue
6	StdA_SSTX+	Yellow
7	GND_DRAIN	GROUND
8	StdA_SSRX-	Purple
9	StdA_SSRX+	Orange
Shell	Shield	Connector Shell

Source: pinouts.net

2.2.8 Memory cards

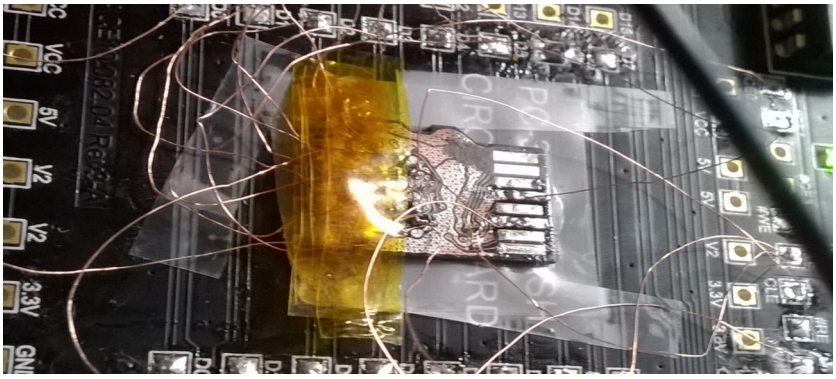


Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X



Pin	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

SD and microSD - pinouts



Example of a MicroSD card connected via NAND interface using PC3000 Flash adapter

Be extremely careful when removing the protective coating on MicroSD card as you can damage the copper wiring print. I have seen the best result using class pen instead of fine sand paper. The yellow strips shown on the example above is a heat protective tape fixed in place by super glue.

2.2.9 SSD

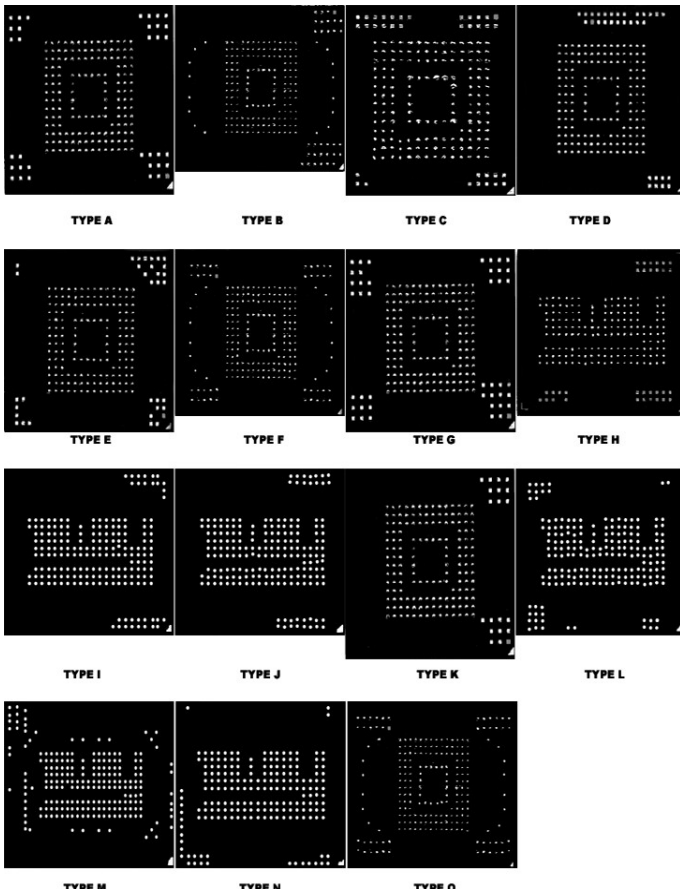
The most noteworthy items of SSD are:

1. The Flash Controller
2. The NAND chips
3. Other active Integrated Circuits (ICs)
4. Resistors, inductors, & capacitors
5. An underlying Printed Circuit Board (PCB)
6. The actual SATA header / connector.

In most cases the Chip-Off method is not very useful for SSD data recovery due to its complexity. An elegant solution to recover data from SSD is a hardware/software solution offered by ACE Laboratory as PC3000 SSD, which in most cases uses a special techno mode to access and recover data. This software may be explained in a separate chapter at some point. This paper will not deal with SSDs.

2.2.10 eMMC

eMMC comes as a BGA and on the picture below you can see various layouts.



ATF Classification - (http://www.advance-box.com/)

Generally you should have an IR rework station with a preheating plate. In most cases some form of plastic will secure the chip in place. Removing of the plastic or epoxy can be done with chemicals or with a pre-heater. Once you heat up the PCB around the eMMC chip keep the PCB in place and use something sharp; like a surgical knife or razor blade to physically remove the epoxy. Try to be as gentle as possible. Also try working with the help of a magnifying glass or at least use a cheap digital USB microscope.

So you have removed all the glue or epoxy? Adjust your IR rework to cover the entire chip run to shine two or three times and you can use tweezers to remove the eMMC chip. All you have to do now is clean the bottom of the BGA chip, remove any loose solder and prepare the chip for placement into a socket.

Another option is to re-ball the chip onto the reader pads but it is an unnecessary step.

3. Reading a Physical Image

In most cases the results of such deep access and recovery of physical memory data is a large pile of raw data that needs to be manually carved through and decoded with other tools and skills, utilizing a range of solutions. Solutions are available through support of data recovery equipment manufacturers to custom scripts to carve for data.

While the use of chip-off and JTAG can be very useful for data recovery, it is imperative that we understand the risks involved and gain a proper understanding before making our attempt at data recovery. I cannot stress the importance of proper training enough! Plenty of practice and performing a first (and second and third, if necessary) run-through with similar devices.

Chip-off and JTAG/eMMC techniques are indeed opening up new avenues for the industry to recover data, and we have heard of more than a few instances where old phones in evidence archives that were believed to be inaccessible are now being looked at again, this time successfully.

3.1 DUMP creation via JTAG/eMMC

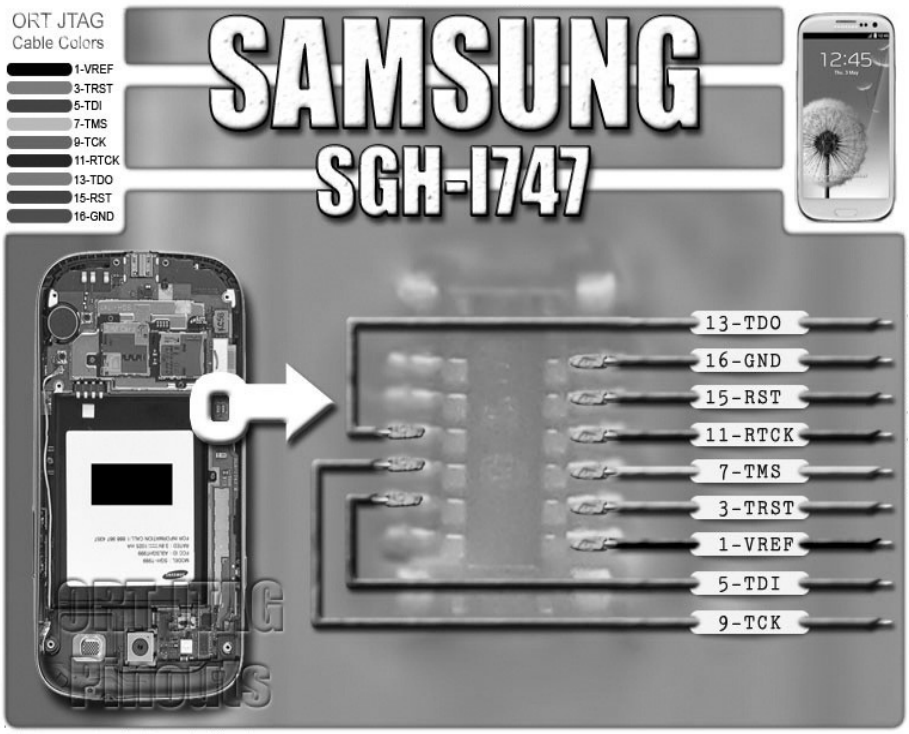
Typically, mobile devices are implementing the BGA memory chips and incorporate JTAG for test and debugging. JTAG ports recently have become a way into the memory chip to retrieve a physical image of the data without requiring the removal of the chip. This has become particularly useful with devices that are not supported by data recovery tools due to the removal of data ports for users. In many cases, the JTAG approach is the only cost-effective way of acquiring data.

The tools required to perform an examination through the JTAG ports include:

1. The equipment to disassemble a device to the point of revealing the JTAG ports
2. Wiring and solder equipment to connect the appropriate wires from the JTAG ports to the JTAG tool.
3. JTAG software and hardware.
4. USB cables
5. Power supply

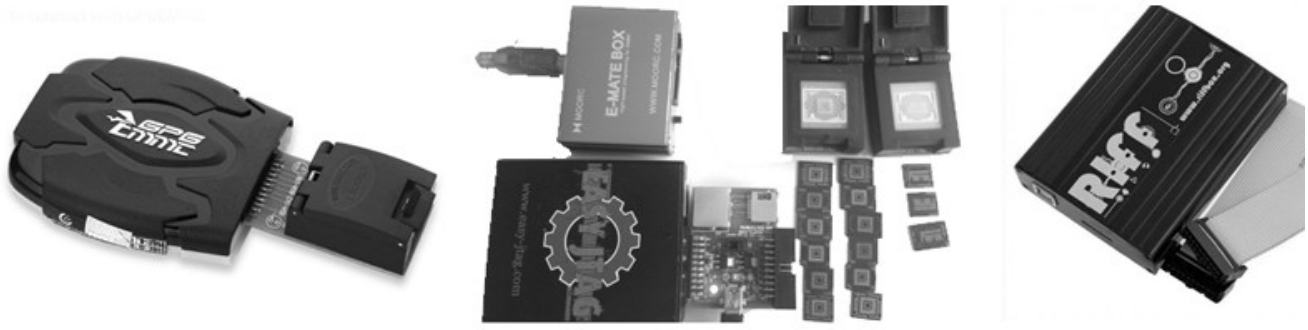
JTAG equipment varies in price from a few hundred dollars to several thousand dollars. The main purpose of this equipment is to test and debug and **NOT TO RECOVER DATA!**

So be extremely careful when operating in this mode. See the example on the next page.



Source: ort-jtag.com

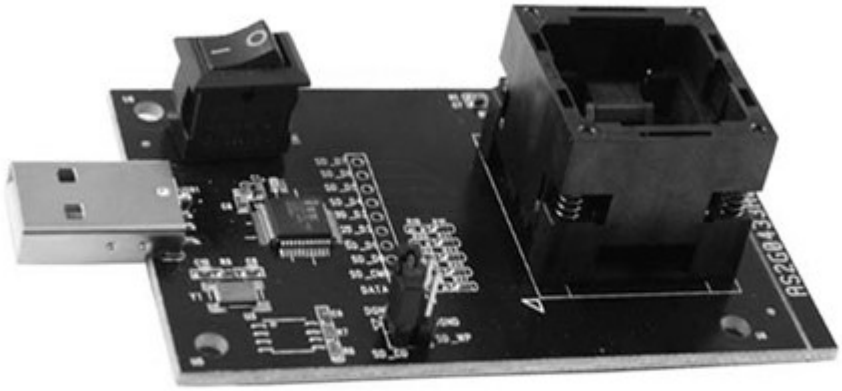
eMMC reading from a chip can be done with various USB sets such as easyJTag Z3X, GPG, ATF, eMATE etc. While superb for JTAG, RIFF, as an example, has limited options when it comes to reading eMMC chips.



GPG EASY JTAG / E-MATE RIFF

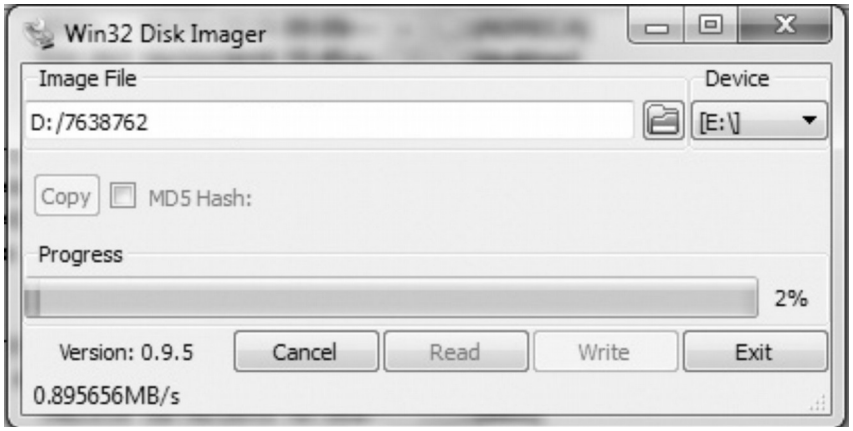
Another good option is to have a reader with a socket and direct connections to SD. See image below:

Connect the chip to the socket, the socket to the adapter, the adapter to a box and the box to a computer. Run appropriate software and you are ready to start.



Example of USB eMMC reader (BGA socket)

Reading a logical image via JTAG or directly from an eMMC chip saves time later since data does not require transformation(creation of a virtual image). Data is read as a logical image since eMMC has built in VFL and FTL both. JTAG is the safest but slowest way. Reading data via eMMC interface is faster and if the chip has no firmware errors, the most convenient way. Boxes such as e-Mate or eMMC to a USB reader will allow the creation of a logical image (clone) like any other device. Here is a simple utility to create images out of the eMMC chips.



Win32 Disk Imager

3.2 DUMP creation using Chip-Off and NAND interface

Once a chip is removed, the next step is to ensure that it can be read. Apart from the tools to remove the chips from the boards, the common configuration for chip-reading equipment typically consists of:

1. The memory-reading hardware.
2. The adapter that holds the chip (and attaches to the chip-reading tool).
3. Software running on the PC to acquire the data from the chip in the adapter that is attached to the reader.
4. Enough space on the hard drive.

Among the majority of data recovery engineers there is a strong belief that raw reading of flash memory chips requires tools like: Visual Nand Reconstructor, PC-3000 Flash or Flash Extractor. Although highly recommended for reconstruction of raw data they are not necessary for chip reading. Reading can be done with almost any serial to USB adapter and terminal access fast enough run NAND chip.

To extract a physical image from NAND memory (DUMP) it is necessary to read the chip's ID and set reading parameters. The chip may or may not support ONFI standardization. The chip parameters are divided into 4 identification parameters as follows: physical parameters, crystal geometry, identification parameters and reading protocol.

3.2.1 Protocol parameters

To extract a physical image from NAND memory (DUMP) it is necessary to read the chip's ID and set reading parameters. Chip may or may not support ONFI standardization. The chip parameters are divided into 4 identification parameters such as: physical parameters, crystal geometry, identification parameters and reading protocol.

Reading protocol

There are two address types used: the column address and the row address. The column address is used to access bytes or words within a page, i.e. the column address is the byte/word offset into the page. The row address is used to address pages, blocks, etc. **(4)**

- **COL cycles** – how many bytes are necessary for all flash memory address space. By default COL = 2
- **ROW cycles** – default ROW = 2
- **Second read command cycle** - flag to determine if it is necessary to use a second cycle read command 0x30 according to JEDEC standard.
- **Data transfer protocol –** transfer protocols can be: Asynchronous ONFI/SDR, DDR, WL triple address, WL Triple address + DDR

Identification parameters

- Model – chip model number
- Vendor – chip vendor
- ID – unique ID

Physical Parameters

- Speed – signal frequency (high = 8mb/s, medium = 4mb/s, low=1,5mb/s)
- Power – voltage level on power input Vcc (pins 12&37)
- I/O power – voltage of I/O ports VccQ (pins 34&39)
- Bus – number of data transmission lines (IO bus) and can be 8-bit/16-bit
- Pinout – signal assignment of pins
- VSP – vendor specific pin. Determines the signal state of VSP1 (pin 38), VSP2 (pin 35), VSP3 (pin 20). Default state is 0 when flag is on 1.

3.2.2 Bit error analysis and power adjustment for bit error minimization

While reading the chip a noticeable internal noise and interference will occur. Higher noise levels results in bit errors and data corruption. This problem is particularly increased in the case of TLC flash chips.

If the physical image is extracted with a high number of bit errors, the correction through Error Correction Code, storage in the spare area of the page is impossible. In this instance the data becomes damaged and recovery is also impossible.

Rusolut researchers experimentally proved that lowering the power of NAND memory reduces the internal noise of NAND and completes reading operations with less bit errors (remained errors can be corrected using ECC). For many contemporary chips a significant reduction of bit errors is notable when voltage is lowered to 2.5v or even 1.8v. The voltage level should be adjusted experimentally, starting at a standard 3.3v and going down to the limit. To get the best noise level ratio, look for errors in the spare area. In case of VNR this can be done while in Bitmap mode.

Note: When too low power is applied, the NAND chip or reader may hang (the reader may not be able to operate and must be re-plugged). In this case power must be increased in one step. To check the number of bit errors the NAND direct access mode is used. Use the default NAND chip power 3.3V and turn the reader's power ON.

Bit error patterns look like contamination, in the form of "bad pixels". In the data area of the page bit error estimation is difficult, especially if the data is scrambled. On the other hand, the data is clearly visible in the spare area (as dots within columns in the spare area). The distribution of bit errors is mostly random. A visual evaluation can help to determine if power adjustment is needed!

3.2.3 Bad columns analysis and removal

Every dye of modern NAND chips consists of several planes. Typically, it consists of 2 or 4 planes. The plane consists of an array of memory cells grouped into pages and blocks. Planes are connected so that plane 0 is composed of even numbered physical blocks (0, 2, 4, 6,), and Plane 1 comprises odd numbered blocks (1, 3, 5, 7,). Because of the relatively poor quality, TLC NAND chips have factory defects. The defects for each crystal and plane are unique. Within all the pages of each block from one plane defects are identical. These defects are called Bad Columns.

Bad Columns patterns appear as vertical columns with a width of 2 or 1 bytes. They are typically filled with FFFF, 0000, or any other value. As presented below. Each plane stores a number of pages which can be observed as an array or columns. The noise-like pattern is good while a straight pattern usually refers to dead cells. To establish whether there are bad columns scroll to the end of each plane. If they all end at the same place there are no bad columns. If you see a difference you will have to remove them. Refer to your device user manual (PC3K Flash, VNR, etc.) for more details on how to remove bad columns.

Example of a chip with two planes and bad column as it is represented in bitmap/noise view in AceLab's PC3000 Flash. Note that planes has different size.

The Bad Columns pattern and its offsets identification is crucial for a factory defects removal from a physical image.

3.2.4 ECC detection

NAND relies on ECC to compensate for bits that may spontaneously fail during normal device operation. A typical ECC will correct a one-bit error in each 2048 bits (256 bytes) using 22 bits of ECC code, or a one-bit error in each 4096 bits (512 bytes) using 24 bits of ECC code. If the ECC cannot correct the error during read, it may still detect the error. When doing erase or program operations, the device can detect blocks that fail to program or erase and mark them as bad. The data is then written to a different, good block, and the bad block map is updated.

Hamming codes are the most commonly used ECC for SLC NAND flash. Reed-Solomon codes and Bose-Chaudhuri-Hocquenghem codes are commonly used ECC for MLC NAND flash. Some MLC NAND flash chips internally generate the appropriate BCH error correction codes. We will go into details about ECC later in this paper.

3.2.5 Physical images extraction (dump reading)

As stated before NAND Flash memory can be read using Chip-Off and JTAG as a sequence of pages called DUMP. Once created a DUMP will act as a binary image file which needs to be properly converted in order to have access to user data. This is done with Virtual image creation covered in Chapter 4 of this manual (or Chapter 5 for eMMC).

4. Creating Virtual image

In case you acquired your DUMP using the JTAG/eMMC method you do not have to create a virtual image. In other words you skip this part and go straight to the creation of a logical image. Why is that? It is because you already have a built in memory controller of pages and blocks are sorted with the device acting like a block device. Virtual image creation is all about the creation of a virtual block device when data has been acquired from raw NAND Flash chip (DUMP). There are two parts in virtual image VFL and FTL, which I will describe in more details further along in this paper.

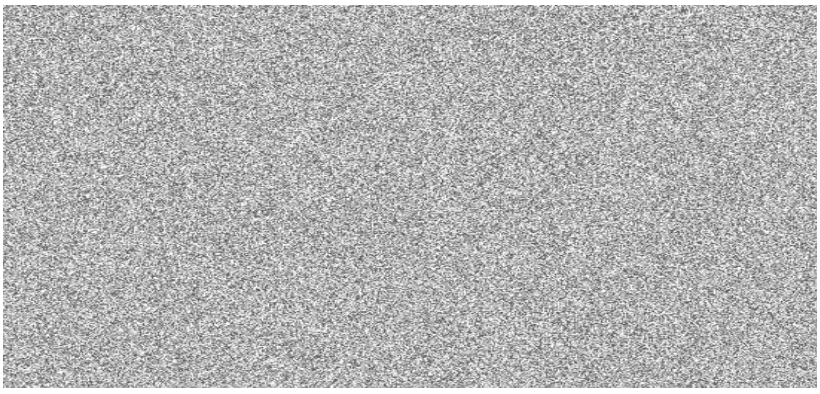
4.1 Physical image structure analysis and description

Once we have the NAND Flash memory DUMP in place it needs to be analyzed and then re-arranged (transformed) in order to have access to user data. This analysis covers: data transformation algorithm, page and block arrangement, combination of multiple DUMPs and finally discovery of translation formula. There are other steps which we will cover later in this manual but their application depends on memory type and other peculiarities of controller used in each specific case. Search for raw data is the first step. The results will tell you if there is some data transformation on the media you are working on. Long story short, if you cannot see a single file … the data has been transformed in some way.

4.2 Data transformations: Inversion and Scrambling (XOR) Analysis

Most flash controllers will transform data before writing it to the NAND memory chip. Transformation of data is done to reduce wear of memory cells. The wear leveling is especially important in TLC chips. There are three typical inversion operations:

- Inversion
- Scrambling (XOR)
- Byte combination



Scrambled (XORed) noise-like data with no typical patterns of data density (entropy of white noise) viewed in Bitmap mode.

Unlike true encryption, scrambling patterns are not designed for security reasons and are therefore easy to identify, extract and then use for data recovery. Resolving this issue starts with determining the structure of the NAND chip. The data area is likely to be XORed, but the spare area markers and the ECC code should on the other hand, be “intact”. Sometimes the spare area markers are XORed too, this is not common case.

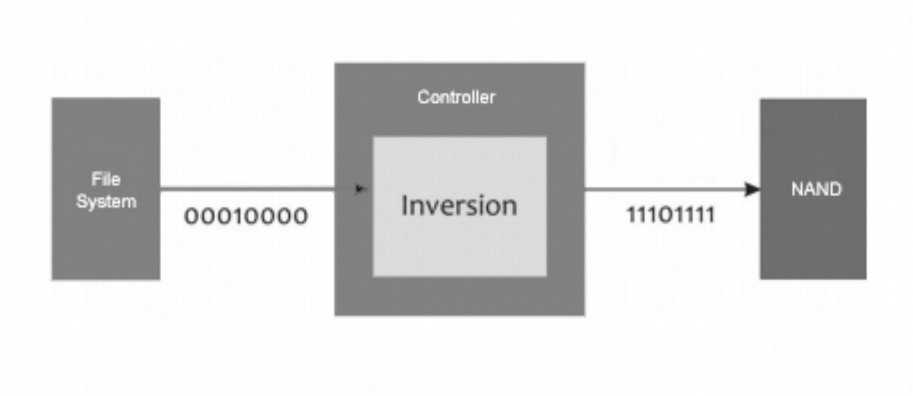
Reverse transformation algorithms are available through the support channel of your preferred data recovery equipment supplier. In case it is unknown for them you can usually ask for their additional help. In most cases combination of algorithms from AceLab, Rusolut and SoftCenter will likely lead to a solution.

However if nothing works identifying the key is usually done with the help of bitmap viewer. Typically, the physical image contains lot of fragments of the Xor key. This is because key scrambled with zeros during data recording process returns pure key fragments or even whole key [Key + 00 = Key]. The key identification relies on visual recognition of the key fragments in dump, using Bitmap viewer mode. The idea behind is simple user’s data contain a lot of zeros, therefore many key fragments are in dump.**(35)**

The solution should contain: **Pattern period** (in pages), **Page size** and full understanding of how it has been applied. Much like controller generates that number of pages of pattern and then it is transformed, shifted with block, pages, etc. This can be a very time consuming process!

4.2.1 Inversion

Inversion is used to reduce the number of writing cycles by creating mathematical NOT operation. Inversion will basically alter all user data to its inverted form. While recovering it is important to revert information to its original form.



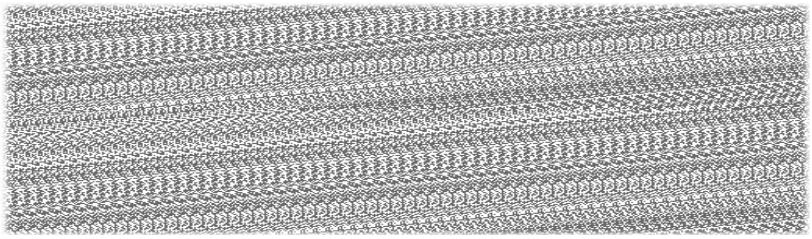
Inversion block diagram

4.2.2 Scrambling (XOR)

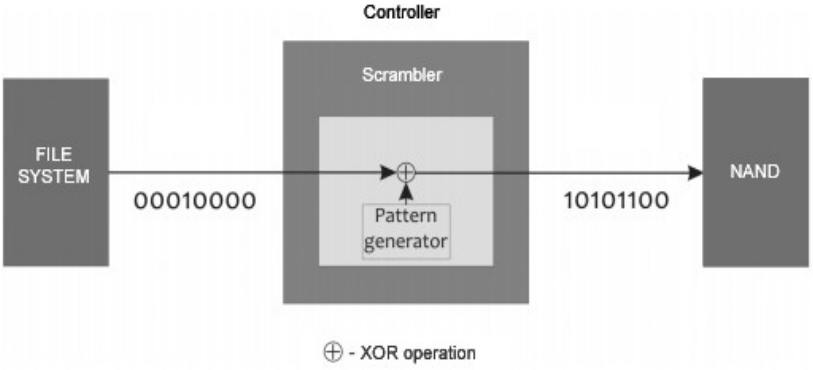
Scrambling (XOR) converts data to binary noise. This transformation is applied to almost all flash controllers. TLC chips are exposed to instability and degradation when they are used with standard user data which have patterns. In order to avoid this, user data is scrambled with a specific binary sequence (XOR key), basically creating noise from user data. The binary sequence is not stored in the flash controller. It is generated on the fly while reading and writing. This sequence is not unique, all controllers of one model use the same key. To convert noise to data it is necessary to use one of the XOR keys from your software resource base. In many cases it is possible to extract the XOR key directly from the DUMP because only the Data Area is scrambled. Other ways include getting the scrambling polynom from the controller's firmware which in most cases is very difficult (but not impossible).

Again, it is good to start in bitmap view because you can actually find different patterns. Search for blocks with a large number of scrambled zeros (fragments of XOR key) to determine the page structure and also to find a proper XOR key to decrypt data.

When scrambling is performed on a data area that contains zeros, a very specific pattern appears in the noise, which belong to fragments of the XOR key. Therefore, any pattern in the scrambled data area that doesn’t looks like a noise, is XOR key fragments.



Bitmap view XOR key fragments



Block diagram of XOR scrambler

4.2.3 Byte combination

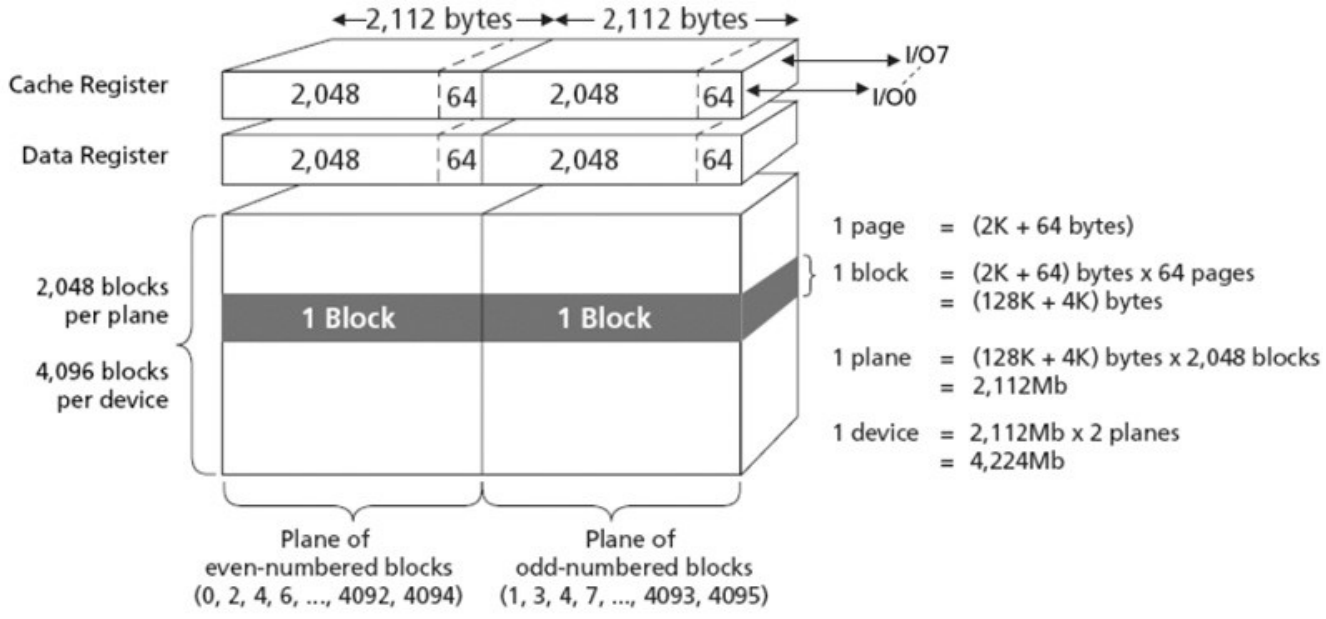
Another way to transform data before writing to NAND Flash memory probably works the same way as byte combination. I need to do more research on this.

4.2.4 Memory Modem

Another solution comes from the Israeli company DensBits. They have designed the Memory Modem™, a new technology created for low-cost, high-performance NAND flash-based storage systems. For example, their DB3610 Memory Modem eMMC controller is designed for 3-bit or TLC flash memories. Back in 2012 Seagate and DensBits announced their strategic partnership for future solid-state drives from Seagate. In principal their approach is very similar to the XOR scrambling process but I wanted to mention this approach so that you are aware that there are different ideas in this field.

Memory Modem is a blend of ECC, DSP, and flash management techniques that treat the flash chip like a noisy communications channel and some say that its ECC performs better than the industry's best ECC implementations, even LDPC.

4.3 Data Organizations in NAND memory



Simplified physical structure of NAND memory is divided into blocks and pages...Source: http://cushychicken.github.io/assets/cooke_inconvenient_truths.pdf)

To describe NAND memory interior structure, and therefore functionality, we use:

Physical block – Physical flash memory (crystal) is organised into physical memory blocks. Depending on the organization of a particular flash memory, physical blocks can be: 64, 128, 256, 512, 1024 or 2048 Kbytes (in case of TLC memory cells they are between 1,5 and 3 Mbytes-excluding spare area). The physical block is the minimal memory area that can be erased for one erase operation.

Virtual block – Actual flash organization involves virtual memory blocks. Virtual blocks have one to several physical blocks. It depends on the page allocation scheme and the storage device organisation. Virtual block is therefore related to the Logic Block Number stored in the spare area.

Physical page – Physical blocks are divided into smaller memory areas operated by the controller as a minimal data volume for reading and writing. Page size depends on the organization of concrete flash memory. Page size can be 512, 2048, 4096, 8192, 16384 bytes (excluding service area)

Virtual Page – Several physical pages related to the Logic Page Number. Virtual block is therefore a set of virtual pages.

Data area (logical sector) – minimal memory area. This information is available to the OS and flash device, logical sector is usually 512 bytes. To read one logical sector you need to read and store to buffer the whole page.

Spare area (SA) – Similar to hard drives flash devices use special service areas to store service data and ECC code. This area does not take part in the LBA. The spare area consists of bytes that define position, type and other parameters of physical blocks, ECC and others.

As previously stated the NAND Flash memory is composed of the blocks of pages, one block is usually composed of 16, 32 or 64 pages. As previously noted most NAND Flash devices uses 512 bytes / 256 words in the “Cell Array” page area (also called “data area”) and an extra 16 bytes / 8 words in the “Spare Cell Array” page area (also called “spare area”). In this case a total of 528 bytes / 264 words per page sometimes referred to as “small page”. For capacities usually greater than 1 Gbyte the “large page” is used which contains 2048 bytes or 1024 words of data area and 64 bytes or 32 words respectively for the spare area. In total 2112 bytes or 1056 words. **(31)(32)**

In a page read operation, a page of 528 bytes is transferred from memory into the data register for output. In a page write operation, a page of 528 bytes is written into the data register and then programmed into the memory array. In a block erase operation, a group of consecutive pages is erased in a single operation. In a brand new device all usable blocks are in an erased state.

4.3.1 Crystal Geometry Parameters

These are the parameters commonly in use:

- **Page Size (bytes)** – minimal data size for read operation
- **Nominal Block Size (bytes)** – in case of TLC flash memory chips the nominal block size is a multiple of 4, while real block size is a multiple of 3. In cases where for example nominal block size is equal to 256 pages, real block size equals 192. The remaining 64 pages are addressed inside flash memory but physically they do not actually exist.
- **Real Block Size (bytes)** - minimal data size for write and erase operations (PE cycles)
- **Nominal Plane size (bytes)** –for TLC chips
- **Real plane size (bytes)** – capacity of crystal
- **Number of planes in the crystal (**1/2/4**)**
- **Number of crystals in the chip (**1/2/4; CE**)**

4.3.1.1 Blocks

Typical block sizes include:

32 pages of 512+16 bytes each for a block size of 16 Kbytes
64 pages of 2,048+64 bytes each for a block size of 128 Kbytes
64 pages of 4,096+128 bytes each for a block size of 256 Kbytes
128 pages of 4,096+128 bytes each for a block size of 512 Kbytes.

4.3.1.2 Pages

Several pages make the block. The page has two areas (data and spare area). User data will be written to the data area while the spare area stores ECC and other service data.

Flash controllers use different page layouts. Controllers can place SA after the DA or in the middle of it. In most cases controllers keep them together.

In cases where data recovery is performed from flash memory with broken or missing chips, page layout is one of the critical goals. The analysis of such requires determination of sizes and offsets relative to offset “0x00” of each page. Every page which uses the same flash controller uses the same scheme.

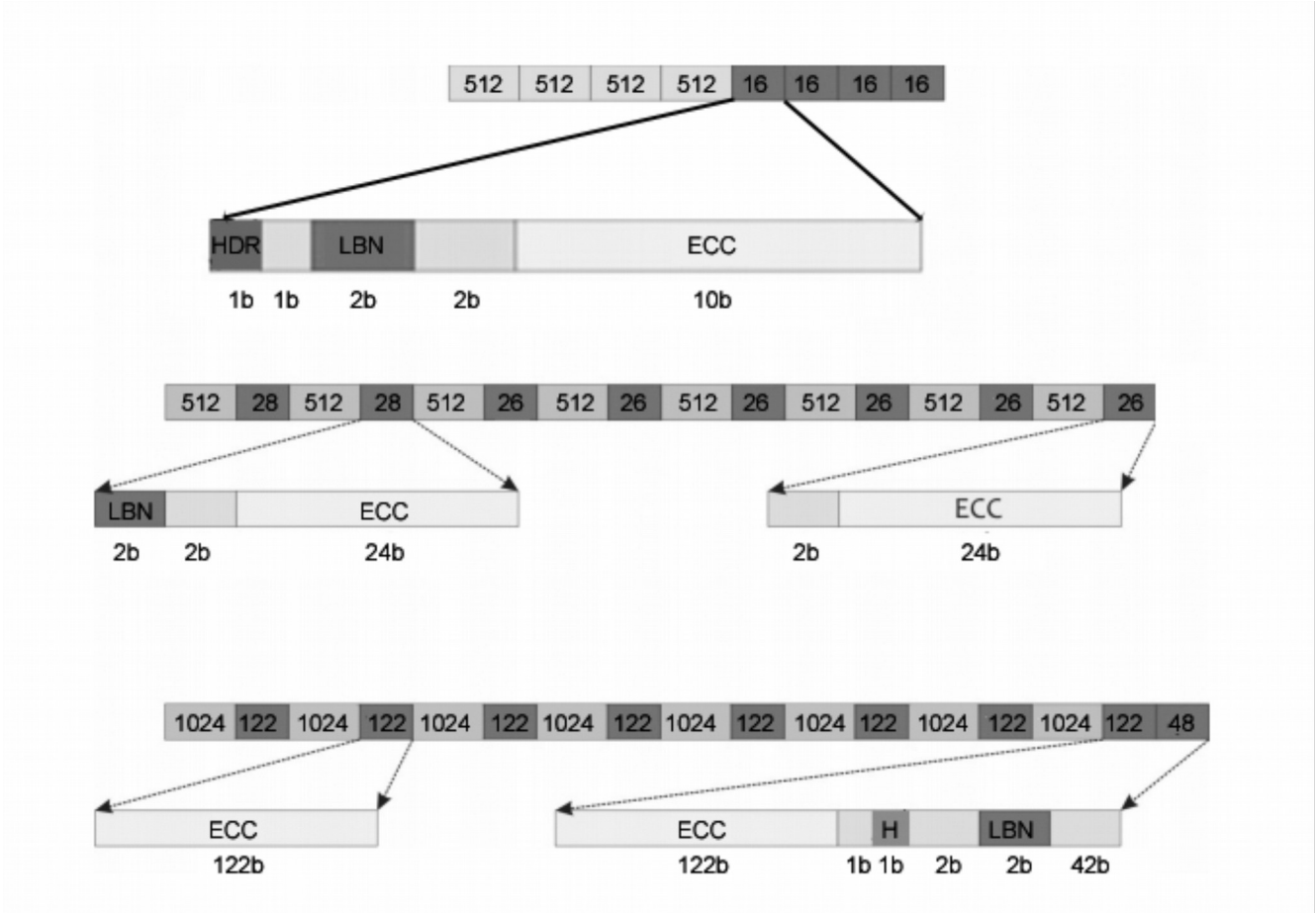
Logical sector always multiplies with 512 bytes like 512, 1024, 2048, 4096 bytes. Most of the present day flash controllers use 1kbyte sector sizes. In many cases the size of the SA can be very different (usually 1/32 of sector size).

Therefore, SA=Page Size-DA. Spare area is usually not fully used and the FFS marker can appear at the end of each!

4.3.1.3 Spare Area

There is no rule about the actual layout of the spare area. They are all different. However, they all store the same type of data, sometimes in slightly different order with data about:

- Error Correction Code (ECC)
- Block Header
- Logical Block Number (LBN)
- Logical Page Number (LPN)
- Block Write Counter
- Bank Number



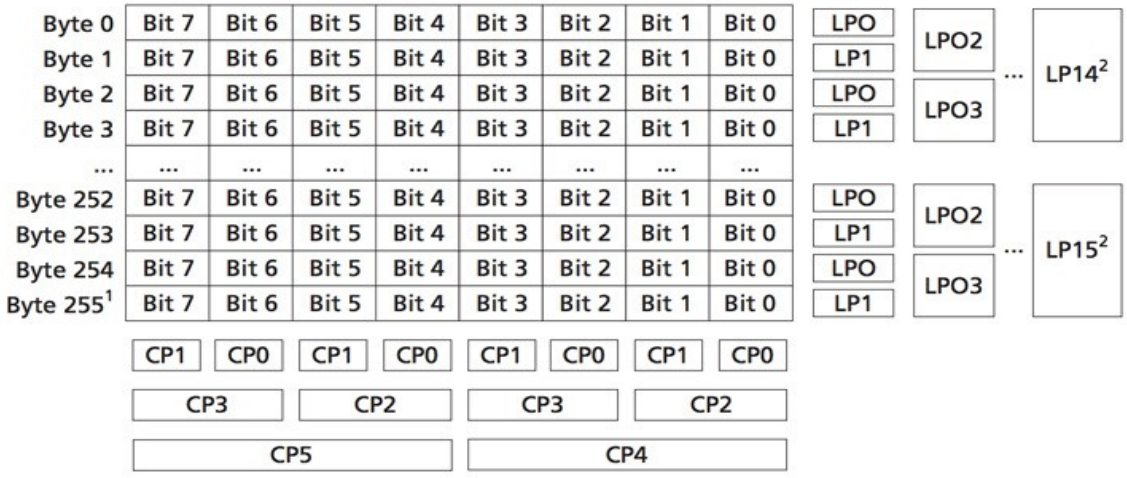
Pages and spare area details

Proper analysis of the spare area makes effective recover of data possible for almost any flash memory.

4.3.1.4 Error Correction Code [5]

The ECC functions as a check on the page data within a NAND device. It uses a fixed algorithm to generate an ECC value from a page of NAND data. During a write, this value gets stored in Flash along with the corresponding data. When that page is read back from Flash, it is run through the same algorithm to generate a second ECC value. If this second ECC value is identical to the original ECC value stored during the page write, then the data can be considered correct. Commonly in use: Hamming Algorithm, Reed-Solomon Algorithm, Bose-Chaudhuri-Hocquenghem (BCH) algorithm.

One of the most common coding schemes for guaranteeing NAND data is a Hamming code. A Hamming code depends on generating a set of parity bits from the data. A parity bit is a simple way of accounting for the number of binary ones in data. If the number of ones is even, then the parity bit is set as a one; otherwise, the parity bit is zero.



Hamming Code Generation (Source: Micron Technology)

The diagram above shows an abbreviated method of generating a Hamming code for a 256-byte page of NAND data. This example shows how the parity bits of each byte (referred to as "line parity" in the example above) and each column are generated. Higher-order bits are generated by doing bitwise XOR operations on the lower-order bits (e.g. CP2 = CP1 ^ CP0; CP5 = CP3 ^ CP2 ;). This example of a 256 byte page will produce a 22-bit ECC value, which is stored in NAND as follows:

Assignment of Data Bits With ECC Code

ECC	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Ecc0[1]	LP07	LP06	LP05	LP04	LP03	LP02	LP01	LP00
Ecc1[2]	LP15	LP14	LP13	LP12	LP11	LP10	LP09	LP08
Ecc2[3]	CP5	CP4	CP3	CP2	CP1	CP0	LP17	LP16

Notes:
1. The first byte (Ecc0) contains line parity bits LP0–LP07.
2. The second byte (Ecc1) contains line parity bits LP08–LP15.
3. The third byte (Ecc2) contains column parity bits CP0–CP5, plus LP16 and LP17 generated only for a 512-byte input.

What happens when your ECC values don't match? As seen previously, there are plenty of ways for a bit to be wrong in a NAND flash page. As you may have guessed, they're called error correction codes for a reason - they can correct errors in paged data! Part of the brilliance of the Hamming code system is how simple it is to determine the source of the error within the block. The algorithm depends on XOR-ing the two ECC values. The resulting value can be interpreted as follows:

- ECC1 ^ ECC2 == 0: the two ECC values are equal; no error is present in the paged data or ECC value.
- Half of the XORed bits equal 1: a single bit error exists in the page of data.
- Exactly one XORed bit equals 1: a single bit error exists in one of the ECC values.
- Any other value: an uncorrectable error occurred.

Error correction codes provide a reasonably good system for catching flaws in NAND data. However, ECC is not without limitations. The Hamming code, while useful, can only detect two bit errors in the page used to generate it. Its correction capabilities are even lower - it can only correct flawed pages with one bit error! Any more errors than that, and the data becomes unrecoverable. There are more advanced error correction algorithms available now, but they generally come at the expense of requiring more bits for the ECC value. More modern algorithms like Reed-Solomon or Bose-Chaudhury-Hocquenghem (BCH) are capable of correcting up to twenty four bits within a page of data. They also tend to be very software intensive operations, and generally require specialized hardware acceleration to be used at a speed acceptable within most embedded systems.

Note: The ECC code area is proportional to the number of errors that the controller can correct and usually takes 3 to 15% of logical sector size and 65 to 95% of SA. Tools like VNR and PC3000 Flash have an automatic ECC decoder for correcting reading errors using data from SA.

There are some great resources to start with basic of ECC on the web. **(33)**

If the ECC area has a very high entropy in bitmap view it looks like noise with no patterns. Sometimes the ECC code has one empty byte (FF) at the end, which looks like the column.

4.3.1.5 Logical Block Number

The Logical Block Number relates virtual block (and pages) to a logical block in the file system. In most cases the flash controller dedicates 2bytes for it. Just by sorting LBN in ascending order it is possible to reconstruct a logical image. The Logical block number pattern has 2 bytes (8 bits) size and changes from block to block.

Note: Depending on bank size LBN starts with 0000 and ends with 01FF, 03FF, 0FFF, FFFF. Sometimes high bit can store extra service information (1000, 13FF)

Virtual block allocation scheme is the most important step in reconstruction of the logical image. This is done by joining the physical chip images in specific order. To do this we need

to use file system structures (FAT tables, FAT folders, NTFS file records, MFT records and other FS metadata), LBN and other SA markers.

We can use LBN markers (they are identical for every page in one virtual block) and in case they are identical across crystals and chips with the same physical block the allocation is parallel. In cases where chips and crystals have different LBN markers at the same physical address, sequential allocation is in use.

The places where the LBN pattern changes are the boundaries of the virtual block. The LBN pattern and its offset identification is essentially for block arrangement into a logical image (Markers table element settings).

4.3.1.6 Logical Page Number

The Logical Page Number determines the position of the virtual page within the block. Most flash controllers will not use LPN in SA so their pages will be written in sequence (linear translation). When LPN is in use this means the flash controller has pages all mixed inside of a block. LPN therefore can be used to arrange logical image (ascending order). The LPN pattern and its offset identification is essentially for block arrangement into a logical image (Markers table element settings). The Logical page number pattern has 1 or 2 bytes size and changes from page to page within the virtual block.

More than 90% of controllers do not use LPN due to sequential page allocation within the block.

4.3.1.7 Bank Number

Bank Number determines where virtual blocks belong. Bank usually has 1024, 2048 or more blocks. Since LBN begins with zero segmentation using banks makes block addressing shorter. Some flash controllers dedicate 1 byte in the spare area to store the Bank Number. In cases where several blocks use the same LBN, the actual Bank Number can be found through the physical block address. The end of each bank can contain a reserved zone where physical bad blocks are allocated. More than 90% of controllers do not use bank number due to sequential bank allocation within the physical image. **(11)**

4.3.1.8 Block Header

Every controller manufacturer uses its own block header format. Block Header is a one byte marker that describes block purpose such as:

- Main Blocks - block that store user data
- Replacement blocks - (blocks with updated user data in cases where the old Main Blocks have not been overwritten)
- Log Blocks - blocks with the page updates in cases where the old data has not been overwritten
- Factory Bad Blocks

- Translation Table Blocks – blocks that contain translation tables of the controller
- Reserved Table Blocks – blocks assigned for bad block re-allocation (FFFFs)
- Firmware Blocks – blocks that contain firmware data
- System Blocks – system blocks are the ones we usually don't really care about
- Unused blocks – other types of blocks which have not been researched yet or hold no useful data

The block header pattern has 1 byte size and does not change very often mostly because 95-99% of all blocks in NAND Flash memory are used to store data (main blocks) and have the same header.

4.3.1.9 Block Write Counter

Block Write Counter contains the number of write cycles. In most cases flash controllers never use all of these areas at the same time. However, values like ECC, LBN, BH are widely in use among manufacturers.

The write counter pattern has 1 byte size and changes from block to block. It may have an interest for digital forensic purposes.

4.4 Virtual Page And Block Allocation Analysis

When assembling the logical image the actual algorithm which emulates NAND to a block device becomes important. Nowadays this is usually done with firmware. Firmware is run by the controller which is built into the storage device. The device has a special subsystem to take care of an interface which provides block I/O access. Well, the interfaces are different and they are defined by different specifications, e.g., MMC, eMMC, SD, USB mass storage, ATA, and so on. But all of them provide block-based access to the device

Until now we have dealt with (one may say), the physical components of NAND Flash memory. In other words, we have dealt with the physical organization of the memory itself.

However, this is just the beginning.

Apart from its physical properties in order to present itself as a block device every NAND Flash memory chip has to have another layer (VFL), which is responsible for remapping bad blocks and presenting an error-free NAND.

The Virtual Flash Layer (VFL) layer knows the physical geometry and translates virtual page numbers used by the FTL to physical addresses (number + physical page number).

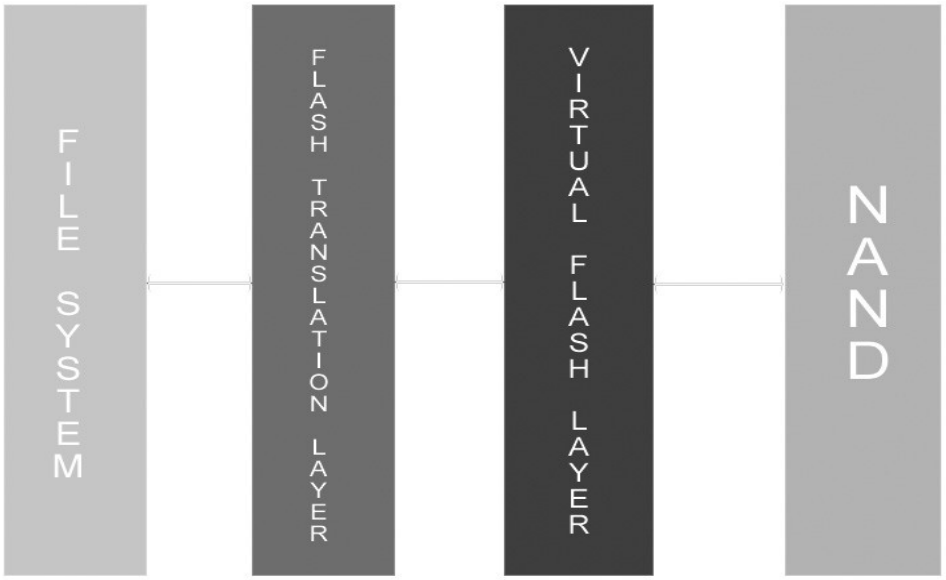


Diagram of steps for NAND to be seen as block device

The best example is eMMC memory which as a stand-alone chip acts as a block device. Although raw NAND flash has a separate controller, from the computer's operating system point of view, all the translation is done before the computer sees your NAND based memory as a block device.

FTL layer operates over VFL, and presents the block device interface to the operating system to file system. It translates block device logical page numbers (LPNs) to virtual page numbers, handles wear leveling and garbage collection of blocks containing outdated data.

Hint! Usually devices that support hardware encryption, all pages that contain data structures related to VFL and FTL are encrypted by a static metadata key. **(13)**

Hint! Linux has an abstraction of a block device. For example; hard drives are block devices. Linux has many file systems and the block I/O subsystem, which includes elevators and so on, which have been created to work with block devices (historically - hard drives). The MTD can automatically detect the width of the flash device bus and the number of devices necessary for implementing the bus width. - Wikipedia

4.4.1 Virtual Block Allocation

What makes flash memory popular is its speed and capacity. Price for the actual memory has been in constant decline over the years. Speed and capacity comes from engineering solutions which use multi-plane R/W operations within one crystal. Other solutions involve parallel distribution of data across several chips. The actual process is similar to RAID 0 in which data has been simultaneously distributed to several blocks within various crystals and different chips.

In cases where data is written, flash controllers will shape up the virtual block by the size of its buffer, physical block size and data transfer channels to allocate data across chips/crystals/blocks. This way of allocating can be divided into three groups.

- Sequential Allocation
- Parallel Allocation
- Combined Allocation

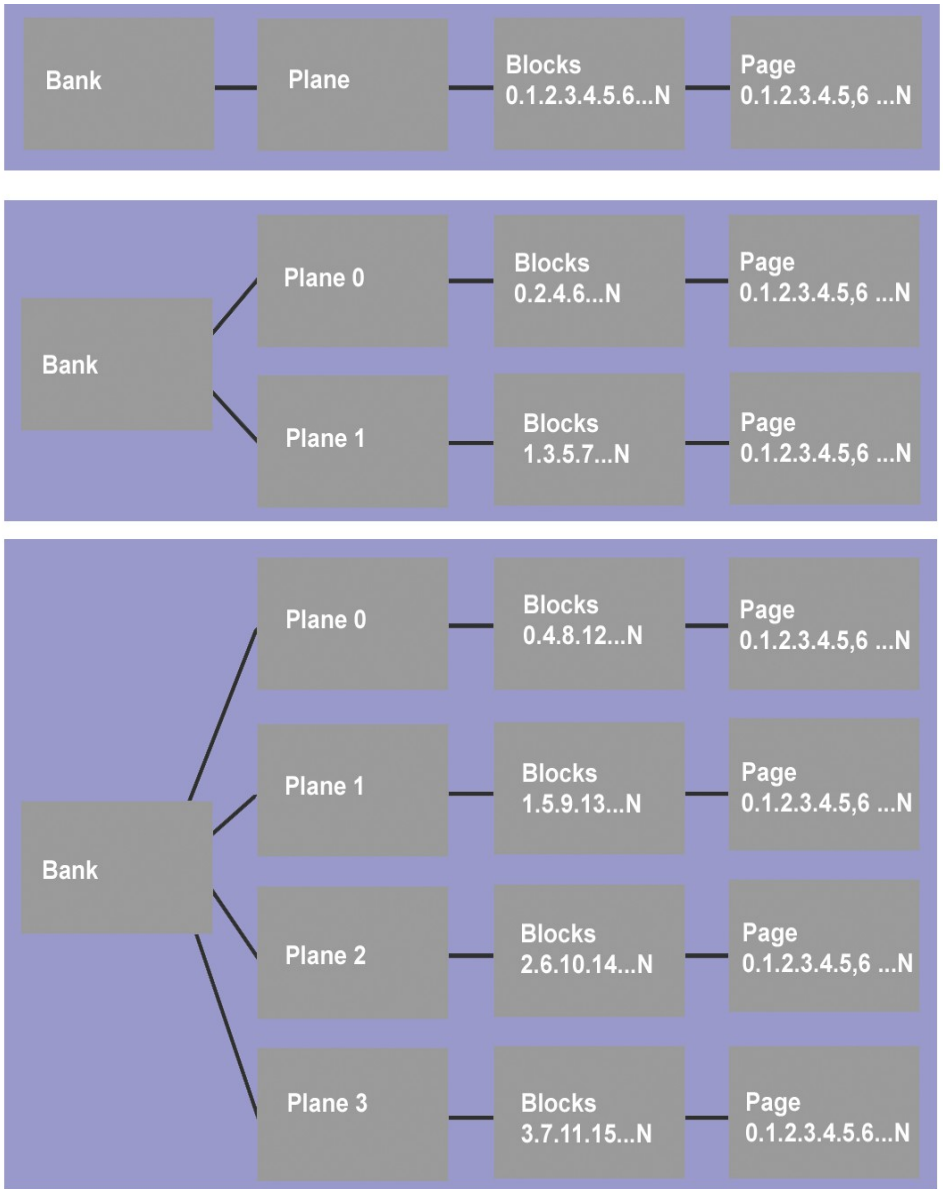
4.4.1.1 Sequential Allocation

In cases where sequential allocation is applied, the size of the flash controller's buffer determines the size of the virtual block. The controller deals with one block at a time, recording page by page switching between crystals/chips. When all of the last blocks are used the controller switches to another chip/crystal. Sequence allocation will also continue writing inside of each block, page after page. This way of allocation traces its origins to JBOD (Just Bunch Of Drives) array applied on hard disk drives. In order to combine NAND, physical image chips must be in sequential order too. The virtual block equals the physical block.

4.4.1.2 Parallel Allocation

In the case of Parallel allocation the virtual block equals several physical blocks (2, 4, 8, 16). The pages from the buffer are written synchronously to the NAND locks/planes/crystals/chips. The size of each virtual block depends on the number of planes/crystals/chips which take part in parallel allocation and the physical block size of a NAND chip. See the representation given below:

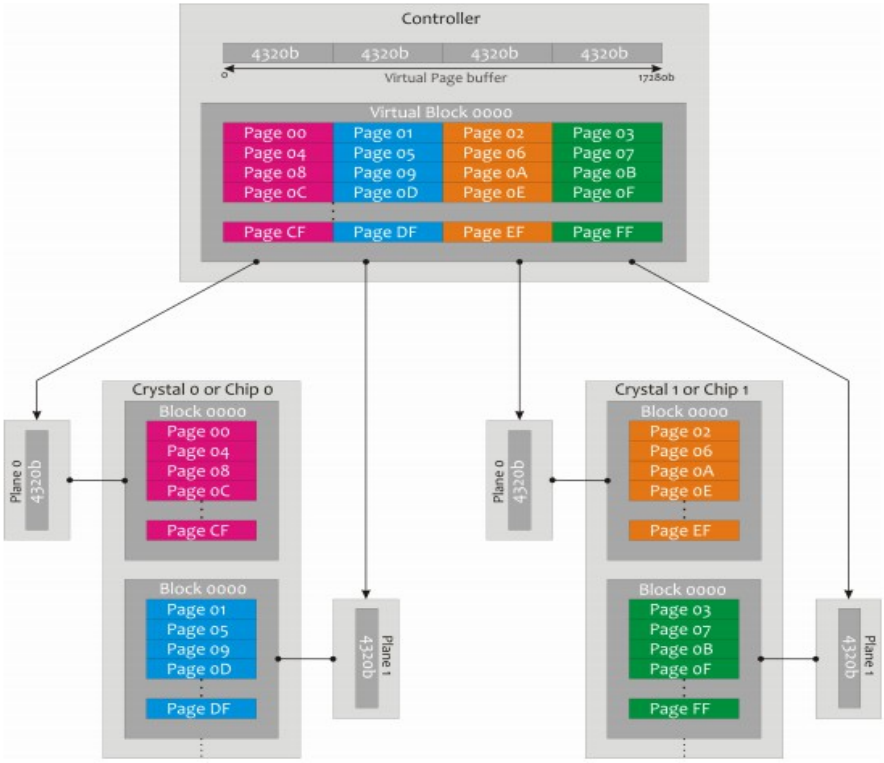
In cases where the flash controller uses multi-plane for read and write operations, the virtual block is equal to two physical ones of NAND memory. The flash controller will record two pages at the same time into two neighboring blocks belonging to different planes in one crystal. In this case each plane has its own buffer and separate memory cell arrays. Tools like VNR have options to remove multi-plane allocation influence.



Sequential Allocation, Parallel Allocation Between two planes and four planes

4.4.1.3 Combined Allocation

In case of combined allocation various serial and parallel combinations are possible. Combinations depend on the number of memory chips, crystals in each chip, number of planes, etc. To illustrate this we can have parallel between blocks in one crystal and serial between crystals and chips used. Virtual block therefore depends on the number of chips, crystals and chips but also to physical size of each block.

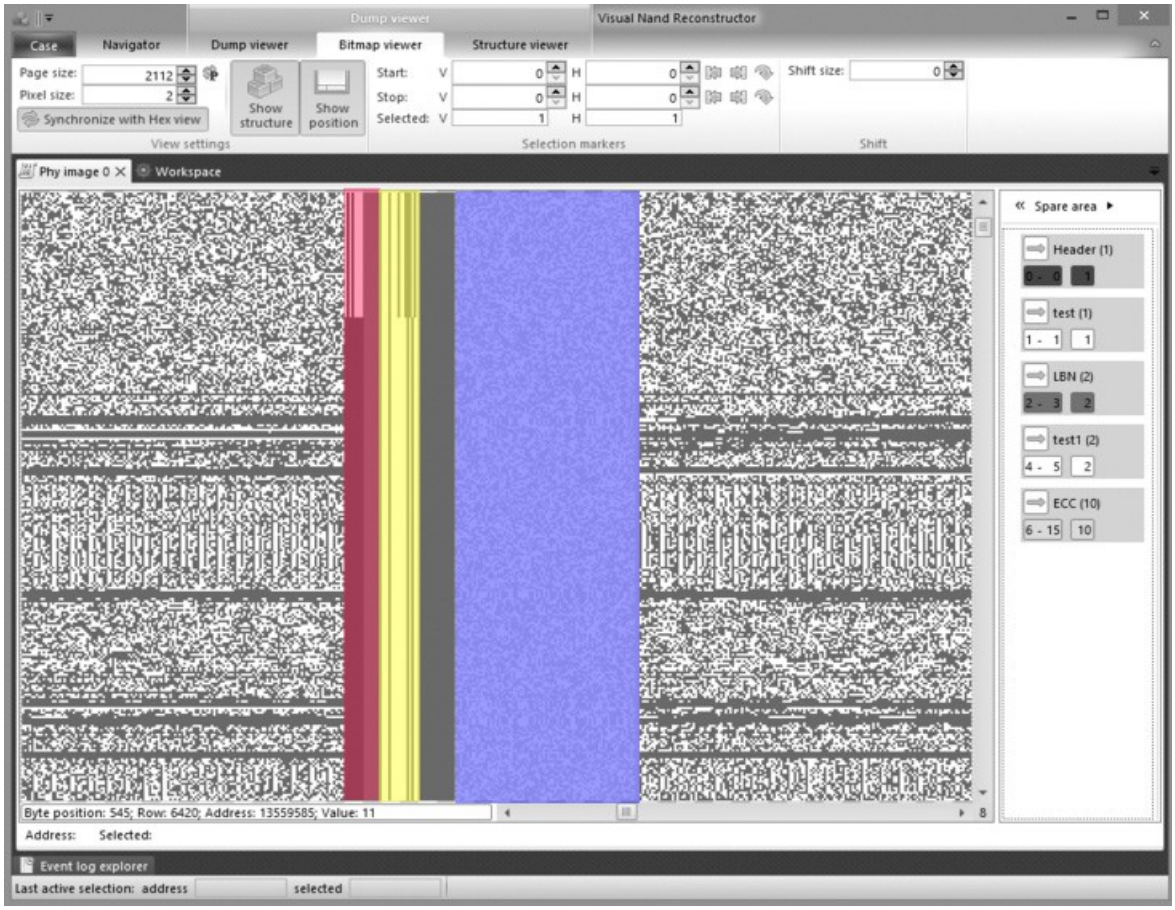


Combined virtual block allocation parallel between two blocks of one crystal
Picture: VNR Book (Rusolut)

4.4.1.4 Spare Area Analysis

Virtual image analysis can be done without ACE Lab, Rusolut, SoftCenter and others but you need to start from scratch. For Linux you can find tons of stuff. Anyone fortunate enough to have time and resources for such a research project will have my full support in any way that I can. It is a wonderful project!

However, most of us want to have the job done and we will end up with one or two or all hardware and software possible. Diving into the raw data and finding a pattern is what this job is all about. The image below is a great representation of how this stuff works. On both sides you have some data and in the middle is the spare area for the one to the left. This sequence has all these pages to blocks to virtual blocks... We see Error Correction Code (ECC), Block Header, Logical Block Number (LBN) and what is not shown; is a Block Write
Counter and Bank Number markers all inside the spare area.



Rusolut's VNR screen with marked ECC (blue), LBN (yellow) and Block Header (red)

4.5 Flash Translation Layer

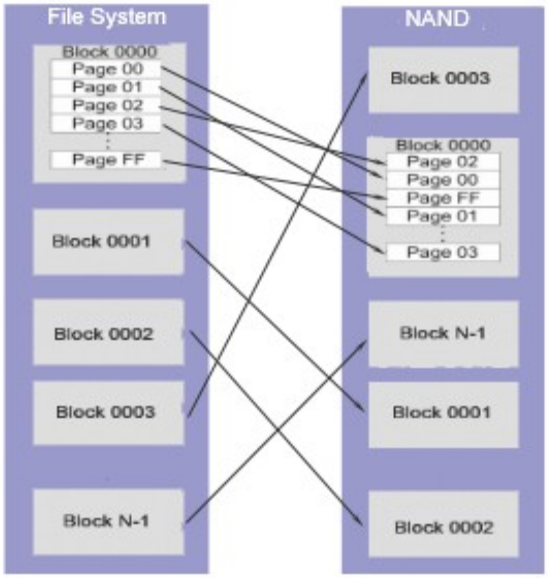
Flash Translation Layer (FTL) is a high performance NAND flash data manager to support sector-based file systems (FAT, NTFS, etc.). FTL operates with Single Level Cell (SLC) and Multi Level Cell (MLC).

As stated before, FTL makes NAND flash chips appear to the OS just like any other block storage device. FTL is an integral part of flash controller firmware.

- Block Mapping
- Wear Leveling implemented to prolong the NAND flash chips lifetime. It also monitors and spreads the number of Page Program (Write) and Block Erase cycles across the entire NAND flash chip
- Bad Block Management
- Garbage collection,
- Block Reclamation

4.5.1 Block Mapping

In the data recovery industry Block Mapping is widely accepted as the most important feature of any FTL. Block mapping is responsible for the allocation of logical sectors to physical blocks and pages. The vast majority of flash controllers use only block translation where pages are linearly translated within the block. Others use block and page translation.



Erratic translation of blocks and pages

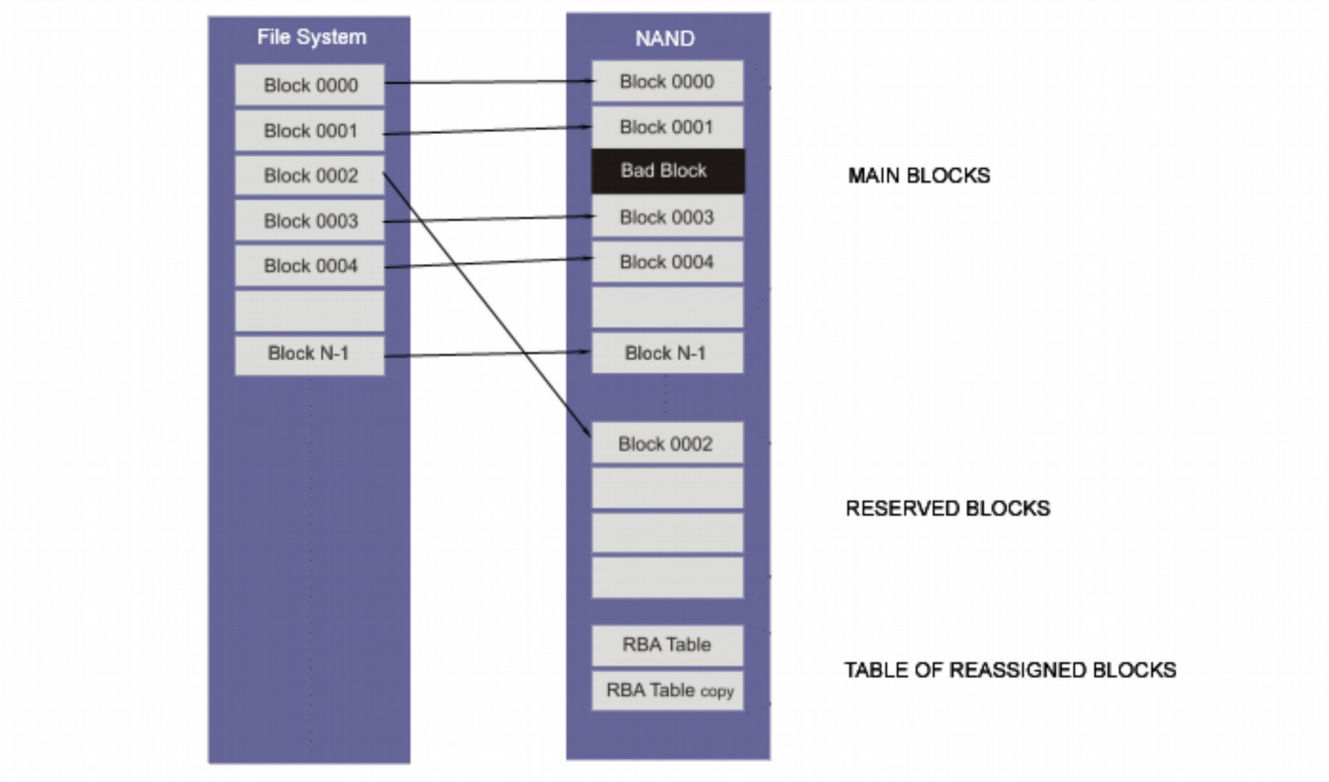
4.5.2 Other important functions of Flash Translation Layer

Wear Leveling – specific algorithm designed to extend longevity of the NAND and deals with average wear leveling of physical blocks in the chip. This is done without specific reference to the block number. The main types of wear-leveling algorithms are:

- Dynamic WL
- Static WL
- FAT filter

Bad Block Management – controls physical blocks of NAND memory and detects bad blocks. In detail, in order to measure cells' performance to store data, an ECC (Error Correction Code) is used. The ECC is stored in Spare Area (SA) of each page.

When a Bad Block is detected FTL marks to SA that the block is bad and moves data to a different block. Similar to traditional BBM applied to hard disk drives, FTL uses Bad Block exclusion and BB reallocation using spare sectors from the reserved Area.



Block are represented linearly which is never the case with NAND
Diagram: VNR Book (Rusolut)

Features like Garbage Collection and Block Reclamation are crucial for inactive blocks which have been erased.

Block Reclamation will populate these blocks back to a list of available ones. These are important in case you want to extract an older version of FS meta-data.

4.5.3 Translation Table Analysis

In previous chapters we dealt with physical parameters of the FTL or the manner in which it is done using hardware. Another option is to use the Flash File Systems (FFS).

Understanding of the FFS is crucial in order to efficiently recover data from individual chips and when the actual controller is missing or has malfunctioned. Although FFS is not closely related to USB flash drives and SSDs (as they use a translation algorithm implemented into the flash memory controller), in most cases even these are based on some type of FFS.

Most engineers and technicians involved with data recovery use solutions provided by data recovery research companies such as ACE Lab, Rusolut and others. These companies have a resource oriented databases with various solutions for translation schemes. They are in most cases closely related to the flash memory controller and will have an ID that may be found inside the spare area.
ID is a signature to a specific translator. Using solutions like these reduces time and effort, especially when dealing with a larger amount of cases.

In case you want to plan more serious research about translation schemes I recommend that you continue reading this chapter. Otherwise skip it, it is rather boring.

By definition the file system is used to control how data is stored and retrieved. With flash this is achieved with Flash File System and it has been done with the help of the flash micro controller, which has proper instructions set to deal with it. Without an FFS, information placed in flash memory pages would be one large body of data with no way of telling where one piece of information stops and the next begins.

The Flash File System should not be confused with the actual file system of the flash storage device containing data to recover. FFS works on the level of FTL and is not visible to the OS unless flash memory uses a Memory Technology Device (MTD), a type of device file in Linux.

To recover data it is crucial to make a proper analysis (Translation Table Analysis). The flash file system applied on flash memory in most cases would be proprietary to the device controller used. However, in many cases it will be some variation of log-structured file system.

A log-structured file system has data and meta-data written sequentially to a circular buffer, called a log. Couple this with what we discussed before in the spare area analysis and you should start understanding how flash memory actually works. Log-structured file systems are designed to deal with: Page Write and Block Erase Cycles, Bad Block Management, Garbage Collection and Block Reclamation.

Log-structured file systems make fewer in-place writes and thus prolong the life of the device by wear leveling.

The more common flash file systems include:

- Journaling Flash File System 2 (JFFS2)
- Unsorted Block Image File System (UBIFS)
- Log-structured File System (LogFS)
- Yet Another Flash File System (YAFFS)
- Flash-Friendly File System (F2FS)

4.5.3.1 Journaling Flash File System 2 (JFFS2)

The Journaling Flash File System 2 (JFFS2) can be found on devices up to 1GB capacity. To overcome this issue, the Erase Block Summary (EBS) was introduced in version 2.6.15 of the Linux kernel. EBS is placed at the end of each block and updated upon each write to the block, summarizing the block's content; during mounts, EBS is read instead of scanning whole blocks. The default file system for Android Open Source Project since Android 4.0.

4.5.3.2 Unsorted Block Image File System (UBIFS)

The Unsorted Block Image File System (UBIFS) is a successor to JFFS2 as a file system for use with raw flash memory media. Development began in 2007, with the first stable release made to Linux kernel 2.6.27 in October 2008. The file system was developed by Nokia engineers with the help of the University Of Szeged, Hungary.

UBIFS works on top of an unsorted block image (UBI) device, an erase block management layer, which is itself on top of a memory technology device (MTD). UBI serves two purposes, tracking NAND flash bad blocks and providing wear leveling. UBIFS tends to perform better than JFFS2 for large NAND FLASH devices. UBIFS preserves or improves upon JFFS2's on-the-fly compression, recoverability and power fail tolerance. UBIFS's on-the-fly data compression allows ZLIB (deflate algorithm) or LZO. JFFS2 stores file system indexes in memory whereas UBIFS stores indexes in flash.

4.5.3.3 Log-structured File System (LogFS)

A log-structured file system is a file system in which data and metadata are written sequentially to a circular buffer, called a log. The design was first proposed in 1988 by John K. Ousterhout and Fred Douglis and first implemented in 1992 by John K. Ousterhout and Mendel Rosenblum.

The design of log-structured file systems is based on the hypothesis that this will no longer be effective because ever-increasing memory sizes on modern computers would lead to the I/O becoming write-heavy because reads would be almost always satisfied from memory cache. A log-structured file system thus treats its storage as a circular log and writes sequentially to the head of the log.

Recovery from crashes is simpler. Upon its next mount, the file system does not need to walk all its data structures to fix any inconsistencies, but can reconstruct its state from the last consistent point in the log.

Log-structured file systems, however, must reclaim free space from the tail of the log to prevent the file system from becoming full when the head of the log wraps around to meet it. The tail can release space and move forward by skipping over data for which newer versions exist farther ahead in the log. If there are no newer versions, then the data is moved and appended to the head.

To reduce the overhead incurred by this garbage collection, most implementations avoid purely circular logs and divide up their storage into segments. The head of the log simply advances into non-adjacent segments which are already free. If space is needed, the least-full segments are reclaimed first. This decreases the I/O load of the garbage collector, but becomes increasingly ineffective as the file system fills up and nears capacity. *(Wikipedia)*

4.5.3.4 Yet Another Flash File System (YAFFS)

Yaffs (Yet Another Flash File System) was designed and written by Charles Manning, of Whitecliffs, New Zealand, for the company Aleph One.

Yaffs1 was the first version of this file system and was designed for the then-current NAND chips with 512 byte page size (+ 16 byte spare (OOB; Out-Of-Band) area). Work started in 2002, and it was first released later that year. The initial work was sponsored by Toby Churchill Ltd, and Brightstar Engineering.

These older chips also generally allow 2 or 3 write cycles per page, which YAFFS takes advantage of - i.e. dirty pages are marked by writing to a specific spare area byte. Newer NAND flash chips have larger pages, first 2K pages (+ 64 bytes OOB), later 4K, with stricter write requirements. Each page within an erase block (128 kilobytes) must be written to in sequential order, and each page must be written only once.

YAFFS2 was designed to accommodate these newer chips. It was based on the YAFFS1 source code, with the major difference being that internal structures are not fixed to assume 512 byte sizing, and a block sequence number is placed on each written page. In this way older pages can be logically overwritten without violating the "write once" rule. It was released in late 2003.

YAFFS is a robust log-structured file system that holds data integrity as a high priority. A secondary YAFFS goal is high performance. YAFFS will typically outperform most alternatives. It is also designed to be portable and has been used on Linux, WinCE, pSOS, eCos, ThreadX, and various special-purpose OSes. A variant 'YAFFS/Direct' is used in situations where there is no OS, embedded OSes or bootloaders: it has the same core file system but simpler interfacing to both the higher and lower level code and the NAND flash hardware. (Wikipedia) YAFFS2 is also the default file system for Android Open Source Project since kernel version 2.6.32 and not supported on newer versions.

4.5.3.5 Flash-Friendly File System (F2FS)

F2FS (Flash-Friendly File System) is a flash file system initially developed by Samsung Electronics for the Linux kernel.

The motive for F2FS was to build a file system that from the start, takes into account the characteristics of NAND flash memory-based storage devices (such as solid-state disks, eMMC, and SD cards), which are widely used in computer systems, ranging from mobile devices to servers.

F2FS was designed on a basis of a log-structured file system approach, which it adapted to newer forms of storage. Jaegeuk Kim, the principal F2FS author, has stated that it remedies some known issues of the older log structured file systems, such as the snowball effect of wandering trees and high cleaning overhead. In addition, since a NAND-based storage device shows different characteristics according to its internal geometry or flash memory

management scheme (such as the Flash Translation Layer or FTL), it supports various parameters not only for configuring on-disk layout, but also for selecting allocation and cleaning algorithms. Google products uses it!

So F2FS is another log-structured file system (LFS) meaning that data is always written to previously unused space and free space is managed in large regions which are written sequentially. The reason we are going into details about F2FS is because majority of producers uses this file system on their mobile devices Lets take a look into the structure of it:

- File system layer
- File name layer
- Data Unit layer
- Metadata layer

File system layer

F2FS has six components that make up its file system layer: Superblock, Checkpoint, Segment Information Table (SIT), Node Address Table (NAT), Segment Summary Area (SSA) and Main area.

1. The partition begins from Superblock. There are two copies of it to avoid file system crash. It includes basic partition information and some default parameters.
2. Checkpoint contains file system information, bitmaps for valid NAT/SIT sets, orphan inode lists, and summary entries of current active segments.
3. SIT includes segment information such as valid block count and bitmap for the validity of all the blocks stored in Main area.
4. NAT contains block address table for all the node blocks.
5. SSA includes summary entries which contain the owner information of all the data and node blocks.
6. Main area contains file and directory data.

File Name Layer

File names in F2FS are stored in dentry blocks. Each dentry block is 4 K in size and consists of 214 dentries and file names.

A dentry occupies 11 bytes and has four attributes:

- hash – file name's hash value;
- ino – inode number;
- len – file name length;
- type – file type (directory, symlink, etc).

A dentry block has the following structure: bitmap (27 bytes) – reserved (3 bytes) – 214 dentries (11 bytes each) – 214 file names (8 bytes each). Dentry blocks form buckets.

Metadata layer

F2FS, like traditional file systems, has three types of node: inode, direct node, indirect node. An inode contains lots of forensically important information, such as file size, allocated blocks, ownership (including UID and GID) and MAC times (Modified, Accessed, Changed).

- The Modified timestamp is updated when the file or directory is written.
- The Accessed timestamp is updated when the file or directory is read.
- The Changed timestamp is updated when the inode is modified.

Data Unit Layer

Like many other file systems, F2FS is comprised of blocks. Each block is 4K in size.

Blocks are collected into segments. Each segment contains 512 blocks and is 2MB in size. Each segment has segment summary block (file plus offset of each block in the segment).

Segments are collected into sections. Sections have flexible size (power of two). There are always six sections “open” for writing at any time. This allows the data to be kept separate from indexing information (nodes).

Sections are collected into zones. These zones make up the main area of the filesystem.

There are also some subsystems often called flash file systems but are more truthfully block drivers since they do not actually have a file system interface. These include:

4.5.3.6 TrueFFS

True flash file system or TrueFFS is not a file system as it does not provide a file system interface but a disk interface. TrueFFS is correctly termed a **flash translation layer**. TrueFFS is designed to run on a raw solid-state drive which is not a most common case. TrueFFS implements error correction, bad block re-mapping and wear leveling. Externally, TrueFFS presents a normal hard disk interface. It was created by M-Systems, which was later acquired by Sandisk. A derivative of TrueFFS, called TFFS or TFFS-lite, is found in the VxWorks operating system, where it functions as a flash translation layer, not as a fully functional file system. Since flash translation layer is used to adapt a fully functional file system to the constraints and restrictions imposed by flash memory devices this is a good place to look for recovery options.

4.5.3.7 ExtremeFFS

ExtremeFFS is a technology being developed by SanDisk allowing for improved random write performance in flash memory compared to traditional systems such as TrueFFS. Sandisk claims that the technology improves random access speed in Solid-state drives by a factor of 100. The company plans on using ExtremeFFS in an upcoming multi-level cell implementation of NAND flash memory.

4.5.3.8 Summary

In the real world and in case you work with VNR, PC3K Flash or Flash Extractor you will use resources provided with each of these tools. Having this information about various flash files systems may help in case you stumble upon an unsupported case. Resolving a translator is commonly based on signature. Do a search and comparison to weigh up what the probability for a correct translation of raw data would be.

There is an option to add virtual translator in PC3000 Flash and a deep understanding of this part is required.

4.5.4 Block sorting and filtering

Once we finally have a better picture on how the NAND works we can in some cases do it manually. When it comes for user data they can appear in:

- Main Blocks
- Replacement - blocks (blocks with updated user data when old Main Blocks has not been overwritten)
- Log Blocks - blocks with the page updates in case when the old data is has not been overwritten yet

4.5.5 Analysis of LBN sequence integrity

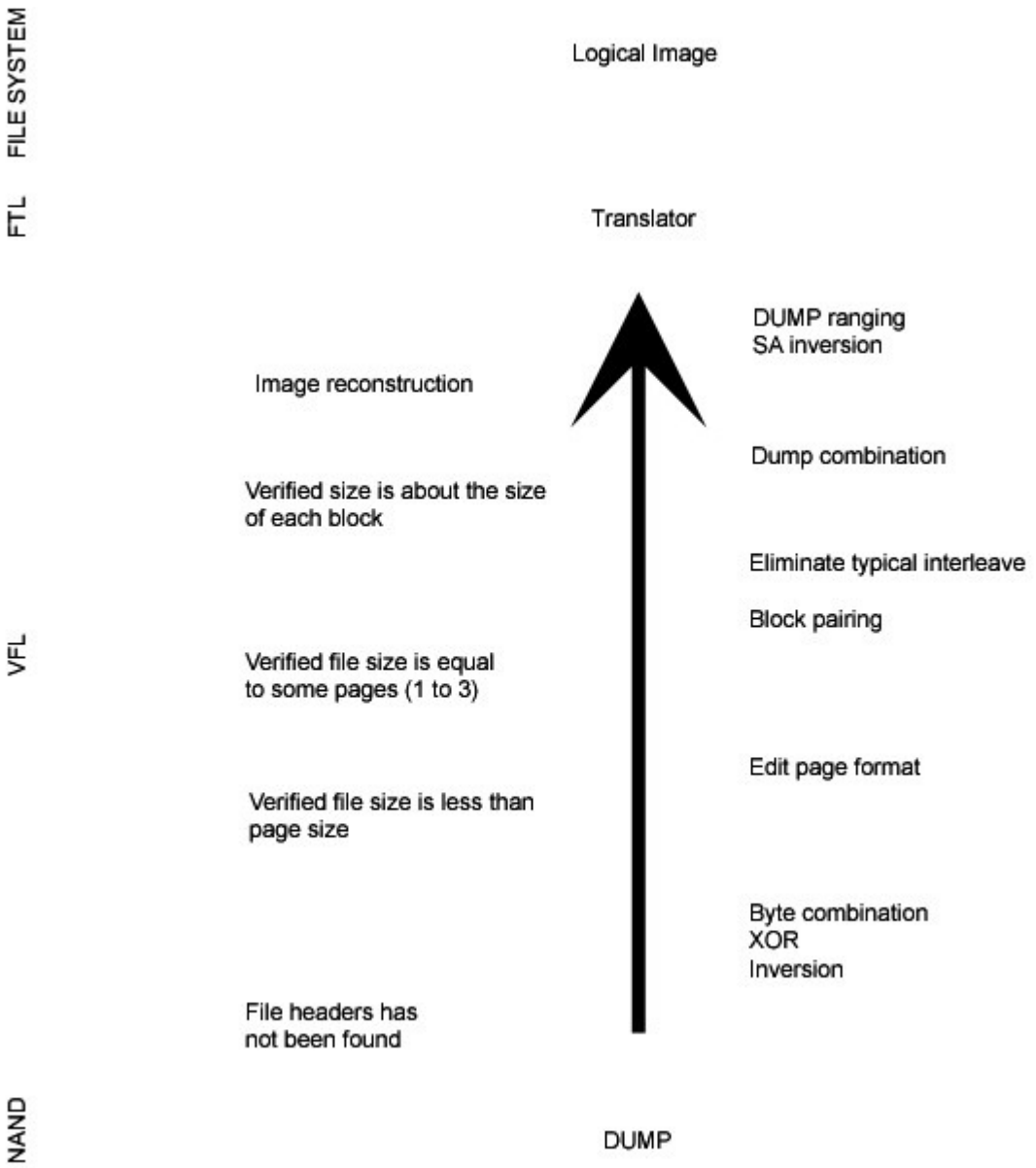
Analysis of our LBN sequence integrity involves duplicated blocks and missing blocks. As they may appear in our sequence, we want to keep them out or at least properly sort them.

4.6. Summary

Let's take a look at the entire process. We have bunch of “shuffled” raw data from which we need to re-arrange data to pages and pages to blocks, keeping their order, bad block count, etc. To do so we use the following method.

Remember the schematics at the beginning of this paper? NAND-VFL-FTL-FS.

That is the way it works. Our job follows that order … not the other way around.



Steps to successfully recover Logical image from DUMP
(Flash NAND Devices) according to AceLab – Stages on the left are author's representation

Three phases of logical reconstruction from Rusolut

First phase - Physical image

1. Bit error analysis via NAND direct access mode
2. Power adjustment for bit error minimization
3. Bad columns analysis and removal
4. ECC detection
5. Physical images extraction (dump reading)

Second phase -Virtual image

1. Physical image structure analysis and description using Bitmap and Structure viewer (Virtual bock size, Page structure). Manual or automatic Data area analysis.
2. Data transformations: Inversion and Scrambling (XOR) analysis using Bitmap viewer. Manual or automatic Data transformation analysis.
3. Virtual page and block allocation analysis: multi-plane allocation, sequential and parallel page allocation between chips/crystals. Manual or automatic Page allocation analysis.

Third phase - Logical image

1. Spare area analysis in Bitmap mode. SA markers analysis: LBN, LPN, Header, ECC, Bank number, Write counter.
2. Setting up parameters of Markers table element and translation table creation
3. Translation table analysis: Block sorting and filtering. Analysis of LBN sequence integrity: duplicated blocks and missing blocks.
4. Block arrange mode and list creation: Main blocks, Replacement blocks, LOG blocks.
5. Data extraction

Typical operation performed on raw DUMP are listed below:

- ECC search
- Inversion
- XOR (Descrambler)
- Pair
- Separate
- Rotate
- Unite
- Offsets
- Arrange blocks
- Bit verification
- Sorting
- Translation
- Other...

5. Assembling a Logical Image

You should have a block device assembled before this step (or any virtual flash and flash translation layer related issued resolved). You either acquired data through a JTAG/eMMC port or you read your data via a NAND interface and you had data transformed the way the controller managed data while the device was operational.

This step is all about the actual file system you have on this newly acquired block device. This file system is very different from the one we discussed before. This one works with the operating system on one side and the flash translation layer on the other (in most cases).

If you have done everything correctly and there are no errors on your logical image, you should be able to mount this logical image as a drive. Ideally you'll see the files listed in their respective folders with the correct sizes and other attributes.

On the contrary, if you have another pile of data which needs to be sorted in the correct way, retrieving data can be done using various software tools. For readers who use PC3000 they can use Data Extractor, VNR has its own logical image file browser, etc. For users preferring the Windows environment, tools like R-Studio, UFS explorer or even WinHex can be quite useful here. Another very useful tool is Santoku Linux (www.santoku-linux.com).

5.1 Reconstructing a file system

The file system reconstruction (if necessary), is usually done with a hardware-software product. In most cases you will be dealing with the most popular file systems such as: FAT, exFAT, NTFS, EXT2/3/4, HFS+, UFS1/2, XFS, ReiserFS, VMFS or VMDK (VMWare) images.

Logically damaged virtually transformed images can also have damaged logical structures which prevents access to the user information. This can be caused by: damaged information about partitions (MBR, GPT, APS, etc.), slight damage to file system metadata, deleted folders and files, significant damage to data and metadata caused by formatting, logical damage caused by viruses, etc.

At this point, it is preferable that you have a good knowledge of file systems and data structure in general.

5.2 Data extraction and verification

The final chapter relates to actual extraction of lost data and its verification. There are numerous ways to do copy/recover files and also many ways to verify what you have copied. I will not go into details here. View thumbnails to verify images, view random files, etc.

6. Conclusion

NAND Flash memory uses address, data and command codes all multiplexed in one same bus. Once the physical parameters are set the data is transferred byte by byte on page basis with extra space for service data. These are all basic parameters of one NAND Flash memory.

We proved “jtaging” is a great approach to save time recovering data from a NAND Flash memory. Although this type of connection can be used to communicate to virtually any electronic device you have around you JTAG is especially useful on devices such as: cell phones, tablets, embedded systems, etc. It is fascinating to me that although disabled at times it is possible to hook-up a hard drive via JTAG and read buffer for example with nothing but RIFF.

Another time-saver is reading data of eMMC memory chip (mobile devices with bad PCB) directly via USB adapter as a standard block device.

Reading raw data from a NAND chip (chip-off) can be tedious at times. During the recovery phase it is critical to obtain original information about the model, chip ID and controller ID. Transformation algorithm can be found on-line (if one had success with it and was kind enough to put it out there), with the help of technical support form Ace Lab for example, or to do it the old fashioned way. In-lab.

To have the transformation done correctly, or to create a virtual image from a raw binary file in most cases involves inversion/XOR and similar transformations, resolving the mix of pages, blocks, etc. Additional problems involves ciphering data internally to protect manufacturer's wear-leveling algorithm.

Phase 1 to this research project has been completed with success. In the next one we will be working on various hip resolving transformations. I will try to publish finding as we move along.

It will take me some time to have more information on this subject. I may publish another paper when I have more info to share. Until then, I wish you all the best in your work.

Thanks for reading.

In case you want to contact me use the following email address ***igor.sestanj (at) adreca.net***

Igor Sestanj

7.Reference

7.1 With thanks to:

Fellow researcher: Nikola Radenkovic

Editor in Chief: Len Rorke

2nd Editor: Natalie Jones

Cover: Aleksandar Mihajlovic

Associates for mobile devices: Aleksandar Mihajlovic, Marko Radenkovic

Translations (from Russian): Marija Marjanovic

In Belgrade, Serbia, 2015 – 2016

7.2 References

1 – **Electronics Engineiring's Handbook, Donald D Fink – Donald Christiansen,** Circuit Principles – Digital Circuits **3-48** (1982)
2 – **Electronics Engineiring's Handbook, Donald D Fink – Donald Christiansen,** Circuit Principles – Digital Circuits, **3-48** (1982)
3 - Micron**, NAND Flash** (http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2919_nand_101.pdf)
4 - ONFI (http://onfi.org/~/media/ONFI/specs/ONFI_1_0_Gold.pdf)
5 - Nash **Reilly's NAND Flash: How It Breaks** (http://cushychicken.github.io/nand-pt5-how-nand-breaks/)
6 - Nash **Reilly's NAND Flash: Device Architecture, Pt. 1** (http://cushychicken.github.io/nand-pt3-arrays/)
7 - Nash **Reilly's NAND Flash: Device Architecture, Pt. 2** (http://cushychicken.github.io/nand-pt4-pages-blocks/)
8 - **Working with MTD Devices** (http://opensourceforu.efytimes.com/2012/01/working-with-mtd-devices/)
9 – **Wikipedia, Joint Test Action Group** (https://en.wikipedia.org/wiki/Joint_Test_Action_Group)
10 –**Wikipedia, Flash File Systems** (https://en.wikipedia.org/wiki/Flash_file_system#FTL)
11 - **Rusolut, Binary Patterns in NAND Flash** (http://rusolut.com/binary-patterns-in-nand-flash-memory/
12 - **Reverse engineering NAND Flash for fun and profit** (http://community.hpe.com/t5/Security-Research/Reverse-engineering-NAND-Flash-for-fun-and-profit/ba-p/6418140#.VkRNYdKrQcV)
13 - (http://esec-lab.sogeti.com/posts/2012/06/28/low-level-ios-forensics.html)
14 – **Rusolut, NAND Bad Columns analysis and removal** (http://rusolut.com/nand-bad-columns-analysis-and-removal/)
15 - **The book – RUSOLUT** (http://rusolut.com/wp-content/uploads/2015/01/VNRbook2.pdf)
16 – **Ace Lab PC-3000 Flash. How to use dump of memory chip from other readers**

(http://blog.acelaboratory.com/how-to-use-dump-of-memory-chip-from-other-readers.html)
17 - **PC-3000 Flash. How to change the page size** (http://blog.acelaboratory.com/569.html)
18 – **AceLab YouTube** (https://www.youtube.com/user/PC3000datarecovery/videos)
19 – **Toshiba, NAND Flash Applications Design Guide** (Atsushi Inoue, Doug Wong, Revision 1.0, April 2003)
20 – **TEEL technologies** (www.teeltech.com)
21- **My Data Recovery Lab, Flash Translation Layer** (http://www.mydatarecoverylab.com/flash-translation-layer/)
22 - **My Data Recovery Lab, NAND Data Recovery** (http://www.mydatarecoverylab.com/nand-data-recovery/)
23 - **Gillware Data Recovery, Monolithic USB Flash Drive Data Recovery** (https://www.youtube.com/watch?v=SMwHuK9Esxo)
24 – **How Flash Memory Works** (https://youtu.be/s7JLXs5es7I)
25 - **Memory Modem TM FTL Architecture for 1Xnm / 2Xnm MLC and TLC Nand Flash** August 21, 2012 (Hanan Weingarten, CTO, DensBits Technologies)
26 – **YouTube Video** (https://www.youtube.com/watch?v=FR6LdbD1YsM)
27 – **X-Ray inspection on NAND Flash** (http://www.spansion.com/Support/Application%20Notes/X-ray_inspection_on_flash_AN.pdf)
28 - **Design Tradeoffs in a Flash Translation Layer** (Garth Goodson, Rahul Iyer)
29 - **A forensics overview and analysis of USB flash memory devices** (Krishnun Sansurooah, Edith Cowan University, 2009)
30 - **PC-3000 Flash. Block rotation in Phison PS2251-03** (http://blog.acelaboratory.com/pc-3000-flash-block-rotation-in-phison-ps2251-03.html)
31 - **MICRON** (http://www.ece.umd.edu/~blj/CS-590.26/micron-tn2919.pdf)
32 – **ELNEC** (https://www.elnec.com/sw/an_elnec_nand_flash.pdf)
33 – **Types of ECC Used on Flash** (http://www.spansion.com/Support/Application%20Notes/Types_of_ECC_Used_on_Flash_AN.pdf)
34 – **SEGGER** (https://www.segger.com/emfile_driver_nand_flash.html)
35 – **Rusolut - Chris Sampson, TRC Data Recovery** (http://rusolut.com/xor-scrambler-key-extraction/)

Igor Sestanj

NAND Flash Data Recovery Cookbook

Publisher: Igor Sestanj

Cover: Aleksandar Mihajlovic

Belgrade, Serbia - 2016

Pages: 69

Format: PDF (http://www.adreca.net/NAND-Flash-Data-Recovery-Cookbook.pdf)

ISBN: 978-86-919771-0-8


ISBN 978-86-919771-0-8
9 788691 977108