NAND/MTD support under Linux

Matthieu CASTET - www.parrot.com

12 July 2012
1 NAND Flash
Flash is everywhere

- non-volatile computer storage chip that can be electrically erased and reprogrammed
  - USB flash drives
  - Memory cards
  - Solid-state drives

Matthieu CASTET - www.parrot.com
Flash is everywhere

- smartphone
- audio player
- digital camera
- car-kit
- ...

Matthieu CASTET - www.parrot.com
## Type of flash: NOR vs NAND

<table>
<thead>
<tr>
<th>Feature</th>
<th>NOR</th>
<th>NAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>memory Bus</td>
<td>I/O</td>
</tr>
<tr>
<td>Cell Size</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>Cell Cost</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Read Time</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Program Time single Byte</td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Program Time multi Byte</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Erase Time</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Power consumption</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Can execute code</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bit twiddling</td>
<td>Yes</td>
<td>1-4 times</td>
</tr>
<tr>
<td>Bad blocks at ship time</td>
<td>No</td>
<td>Allowed</td>
</tr>
</tbody>
</table>

NAND/MTD support under Linux
1 NAND Flash
   • NAND Features
Matthieu CASTET - www.parrot.com

NAND/MTD support under Linux
**Figure 29**  
**Read timing**

C1-C2  Column address of the page to retrieve. C1 is the least significant byte.

R1-R3  Row address of the page to retrieve. R1 is the least significant byte.

Dn    Data bytes read from the addressed page.
Memory organisation

- Serial Input (x8 or x16) 30ns (max clk)
- Serial Output (x8 or x16) 30ns (max clk)
- Program: ~300µs
- Read (page load): ~25µs

NAND Memory Array
- NAND Page 2112 bytes
- 2048 Blocks (2Gb device)
- Block Erase ~2ms

NAND Block
- Data Area 2048 bytes
- Spare Area (ECC, etc.) 64 bytes
- 64 pages per block
Memory organisation

- read/write unit: page
- erase unit: block
- typical block size
  - 64 pages of 512 (32 KB)
  - 64 pages of 2,048 (128 KB)
  - 64 pages of 4,096 (256 KB)
  - 128 pages of 4,096 (512 KB)
- erase: all block set to 0xFF
- program: switch bit from 1 to 0
SLC
MLC
Bad block

- Block can be bad at ship time
- Block can become bad (Erase/Write failure)
- A marker is needed to detect them: bad block marker
Ecc

- bitflip corruption can happen: Read/Write disturb, cell wear
- need to protect data with error correcting code (ECC)
- done on host side
- for SLC
  - 1 bit per 512 B (43 nm - 350 nm)
  - 4 bits per 512 B (32 nm)
  - 8 bits per 512 B (24 nm)
  - 12? bits per 512 B (19 nm)
- for MLC
  - 4 bits per 512 B (43 nm)
- need to do bit scrubbing
  - bitflips can accumulate over time
Ecc

- bitflip corruption can happen: Read/Write disturb, cell wear
- need to protect data with error correcting code (ECC)
- done on host side
- for SLC
  - 1 bit per 512 B (43 nm - 350 nm)
  - 4 bits per 512 B (32 nm)
  - 8 bits per 512 B (24 nm)
  - 12? bits per 512 B (19 nm)
- for MLC
  - 4 bits per 512 B (43 nm)
- need to do bit scrubbing
  - bitflips can accumulate over time
Ecc

- bitflip corruption can happen: Read/Write disturb, cell wear
- need to protect data with error correcting code (ECC)
- done on host side
- for SLC
  - 1 bit per 512 B (43 nm - 350 nm)
  - 4 bits per 512 B (32 nm)
  - 8 bits per 512 B (24 nm)
  - 12? bits per 512 B (19 nm)
- for MLC
  - 4 bits per 512 B (43 nm)
- need to do bit scrubbing
  - bitflips can accumulate over time
bitflip corruption can happen: Read/Write disturb, cell wear
need to protect data with error correcting code (ECC)
done on host side
for SLC
- 1 bit per 512 B (43 nm - 350 nm)
- 4 bits per 512 B (32 nm)
- 8 bits per 512 B (24 nm)
- 12? bits per 512 B (19 nm)
for MLC
- 4 bits per 512 B (43 nm)
need to do bit scrubbing
- bitflips can accumulate over time
Ecc Algorithm

- **Hamming**
  - 1 bit correction
  - 2 bits error detection
  - requires 24 bits for 512 B

- **Reed Solomon**
  - used in cdrom
  - existing hardware IP
  - well suited to correct burst
  - example for ba315 controller
    - 4 bits correction
    - 5 bits error detection
    - requires 80 bits for 512 B
Ecc Algorithm

- **Hamming**
  - 1 bit correction
  - 2 bits error detection
  - requires 24 bits for 512 B

- **BCH (Bose, Ray-Chaudhuri and Hocquenghem)**
  - t bits correction
  - requires t*13 bits for 512 B
  - extra parity bit needed for error detection
  - more complex to compute than Hamming (crc like)
Spare area

- Also called "out of band" (OOB) area
- extra memory for each page (about 1/32 of page size)
- store BBM
- store ECC checksum
- can be used for some filesystem metadata
Spare area and ECC

- 512 B of data
- BCH for 4 bits : 52 bits
- BCH for 8 bits : 104 bits
- BCH for 12 bits : 156 bits
- we are out of $512/32 = 16$ B = 128 bits
- no space left for filesystem metadata
Write constraints

- Number of operation (NOP) : 1-4 times
- can’t rewrite ECC!
  - subpage : 4 * 512 B
  - spare area only modification
- Program pages in a block sequentially
  - can’t program $page_i$ if $page_n$ is programmed, $i < n$
  - can’t update marker in OOB area later
- Minimize partial-page programming operations
Endurance

- measured in program/erase cycle
  - 1M-100K NOR
  - 100k SLC NAND
  - 10K MLC NAND
  - 500-100 TLC NAND
Data retention

- 5-10 years
- Choose between endurance / data retention

<table>
<thead>
<tr>
<th>Retention required</th>
<th>Block Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 yr</td>
<td>10 cyc</td>
</tr>
<tr>
<td>5 yr</td>
<td>10000 cyc</td>
</tr>
<tr>
<td>1 yr</td>
<td>100000 cyc</td>
</tr>
</tbody>
</table>

- For improving data retention
  - Limit program/erase
  - Limit read (read disturb)

- Photos won't be readable for eternity on SDCard/SSD
Data retention

- 5-10 years
- Choose between endurance / data retention

<table>
<thead>
<tr>
<th>Retention required</th>
<th>Block Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 yr</td>
<td>10 cyc</td>
</tr>
<tr>
<td>5 yr</td>
<td>10000 cyc</td>
</tr>
<tr>
<td>1 yr</td>
<td>100000 cyc</td>
</tr>
</tbody>
</table>

- For improving data retention
  - Limit program/erase
  - Limit read (read disturb)

- Photos won’t be readable for eternity on SD Card/SSD
ONFI

- Open NAND Flash Interface Working Group
- standard physical interface
- standard command set
- mechanism for self-identification
- not followed by Samsung, Thoshiba, ...!
NAND Flash
NAND support in Linux

NAND Features

Booting on NAND

- does not support execute in place (XIP)
- needed hardware/software loader
  - NAND controller with boot mode
  - ROM code
- load 1 block (512 - 128 KB) in internal RAM
  - small bootloader or 2 stages (x-load/u-boot)
  - Block 0 is not bad
- difficult to support future NAND
  - identification/geometry
  - error correction
Booting on NAND: example

- **s3c2412**
  - 4 KB of data without ECC
- **ba315**
  - 512 B of data without ECC
- **omap3630**
  - ROM
  - 4 first blocks
  - 60 KB with hamming ECC
  - 32 KB with BCH ECC

Matthieu CASTET - www.parrot.com
NAND support in Linux
2 NAND support in Linux

- MTD
- UBIFS
### Flash device vs block device 1/2

<table>
<thead>
<tr>
<th>Block device</th>
<th>Flash device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consists of sectors</td>
<td>Consists of eraseblocks</td>
</tr>
<tr>
<td>Sectors are small (512, 1024 B)</td>
<td>Eraseblocks are larger (typically 128KB)</td>
</tr>
<tr>
<td>Maintains 2 main operations:</td>
<td>Maintains 3 main operations:</td>
</tr>
<tr>
<td>• read sector</td>
<td>• read from eraseblock</td>
</tr>
<tr>
<td>• write sector</td>
<td>• write to eraseblock</td>
</tr>
<tr>
<td></td>
<td>• erase eraseblock</td>
</tr>
</tbody>
</table>
## Flash device vs block device 2/2

<table>
<thead>
<tr>
<th>Block device</th>
<th>Flash device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad sectors are re-mapped and hidden by hardware (at least in modern LBA hard drives)</td>
<td>Bad eraseblocks are not hidden and should be dealt with in software</td>
</tr>
<tr>
<td>Sectors are devoid of the wear-out property</td>
<td>Eraseblocks wear-out and become bad and unusable after about $10^3$ (for MLC NAND) - $10^5$ (NOR, SLC NAND) erase cycles</td>
</tr>
</tbody>
</table>

- Flash device is more difficult to handle
FTL devices

- Flash Translation Layer
- Emulation of block interface over flash
- Firmware is a black box (Good/Bad FTL)
  - Robust to power failure?
  - Bad block management?
  - Wear leveling ok?
  - Read disturb handling?
- Optimized for FAT
- Abstraction over a standard interface
  - Does not need to handle all aspects of NAND
  - SSD: mix of SLC/MLC, I/O in parallel.
- Patented
FTL devices
MTD

- MTD stands for "Memory Technology Devices"
- Provides an abstraction of flash devices
- Hides many aspects specific to particular flash technologies
- Provides uniform API to access various types of flashes
- E.g., MTD supports NAND, NOR, ECC-ed NOR, DataFlash, OneNAND, etc
- Provides partitioning capabilities
MTD API

- In-kernel API (struct mdt_device) and user-space API (/dev/mtd0)
  - Information (device size, min. I/O unit size, etc)
  - Read from and write to eraseblocks
  - Erase an eraseblock
  - Mark an eraseblock as bad
  - Check if an eraseblock is bad
- Does not hide bad eraseblocks
- Does not do any wear-leveling
MTD NAND support

- MTD started with NOR support
- API was limited to 4GB flash (32 bits offset)
- first NAND controller were dummy/pio
  - new controllers are smarter and need a high level interface
  - dma
  - work at page/block level
MTD evolution : NAND ECC

- bitflips happen quickly
- results in new, previously untested cases/situations
- corruption in bad block marker/bad block table
  - protect bad block table with ECC
  - be robust to bit-flip in bad block marker
- corruption of fs data in OOB area
  - avoid using OOB area or protect data
- empty page not all 0xFF
  - use ECC that works on empty page or empty marker
MTD evolution: NAND ECC

- bitflips are more frequent
- need stronger (software) ECC algorithm
  - BCH
- frequent bit scrubbing
  - bitflip_threshold
  - mtd report bitflip only if nb_bitflip >= bitflip_threshold
MTD evolution: ONFI

- some new flash devices do not support legacy identification
  - add ONFI identification
  - save complex heuristic
- speed optimization: auto-detect NAND timings
Smart NAND

- support of new generation NAND is complex
  - need software modification
  - need hardware modification (software BCH ECC is slow)
- NAND with internal ECC management
  - status command report ECC status
  - optional enable/disable
  - free or used OOB area
NAND support in Linux

- MTD
- UBIFS
Linux flash filesystem

- **JFFS2**
  - Linux 2.4.10 (2001-09)
  - RAM usage
  - slow mount
  - summary

- **YAFFS/YAFFS2**
  - not mainline
  - use OOB area
  - used by android

- **UBIFS**
  - Linux 2.6.27 (2008-10)
  - next generation of the JFFS2
Linux flash filesystem

- **JFFS2**
  - Linux 2.4.10 (2001-09)
  - RAM usage
  - slow mount
  - summary

- **YAFFS/YAFFS2**
  - not mainline
  - use OOB area
  - used by android

- **UBIFS**
  - Linux 2.6.27 (2008-10)
  - next generation of the JFFS2
Linux flash filesystem

- JFFS2
  - Linux 2.4.10 (2001-09)
  - RAM usage
  - slow mount
  - summary
- YAFFS/YAFFS2
  - not mainline
  - use OOB area
  - used by android
- UBIFFS
  - Linux 2.6.27 (2008-10)
  - next generation of the JFFS2
UBI/UBIFS stack

UBIFS

UBI

MTD

NAND  NOR  OneNAND  etc

Flash hardware (NAND, NOR, etc)

NAND/MTD support under Linux
UBI

- Stands for "Unsorted Block Images"
- Provides an abstraction of "UBI volume"
- Has kernel API (include/mtd/ubi-user.h) and user space API (/dev/ubi0)
- Provides wear-leveling
- Hides bad eraseblocks
- Allows run-time volume creation, deletion, and re-size
- Is somewhat similar to LVM, but for MTD devices
## UBI volume vs MTD device

<table>
<thead>
<tr>
<th>MTD device</th>
<th>UBI device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consists of physical eraseblocks (PEB), typically 128 KB</td>
<td>Consists of logical eraseblocks (LEB), slightly smaller than PEB (e.g. 126/124 KB)</td>
</tr>
<tr>
<td>Has 3 main operations</td>
<td>Has 3 main operations</td>
</tr>
<tr>
<td>✔️ read from PEB</td>
<td>✔️ read from LEB</td>
</tr>
<tr>
<td>✔️ write to PEB</td>
<td>✔️ write to LEB</td>
</tr>
<tr>
<td>✔️ erase PEB</td>
<td>✔️ erase LEB</td>
</tr>
<tr>
<td>May have bad PEBs</td>
<td>Does not have bad LEB (handle a certain amount of bad PEB)</td>
</tr>
<tr>
<td>PEBs wear out</td>
<td>LEBs do not wear out - UBI spread the I/O load across the whole flash</td>
</tr>
<tr>
<td>MTD devices are static</td>
<td>UBI volumes are dynamic</td>
</tr>
</tbody>
</table>
Main idea behind UBI

- Maps LEBs to PEBs
- Any LEB may be mapped to any PEB
- Eraseblock headers store mapping information and erase count
- Handle bit-flips by moving data to a different PEB
- Configurable wear-leveling threshold
- Atomic LEB change
- Volume update/rename operation
- Suitable for MLC NAND
- Performs operations in background
- Works on NAND, NOR and other flash types
- Tolerant to power cuts
- Simple and robust design
- easy support in bootloader
UBIFS relies on UBI

- UBIFS does not care about bad eraseblocks and relies on UBI
- UBIFS does not care about wear-leveling and relies on UBI
- UBIFS exploits the atomic LEB change feature
Requirements

- Good scalability
  - Data structures are trees
  - Only journal has to be replayed
- High performance
  - Write-back
  - Background commit
  - Read/write is allowed during commit
  - Multi-head journal minimizes amount of GC
  - TNC makes look-ups fast
  - LPT caches make LEB searches fast
Requirements

- On-the-flight compression
- Power-cut tolerance
  - All updates are out-of-place
  - Was extensively tested
- High reliability
  - All data is protected by CRC32 checksum
  - Checksum may be disabled for data
- Recoverability
  - All nodes have a header which fully describes the node
  - The index may be fully re-constructed from the headers
  - No tool to rebuild it at the moment
UBI scales linearly
  • Read all eraseblock headers on attach
  • 1 or 2 page per eraseblock

UBI fastmap patch in progress
UBIFS do not handle MLC
  - need background flash scanning for bitflips
  - paired pages problem
- UBI/UBIFS do not handle unstable bits
- power cut just before the end of page program
- power cut just after the start of page erase
  - page can be read, but randomly fails after a few times
- power cut just after the start of page program
- power cut just before the end of page erase
  - not a full-erased page. Contains 0xFF, but data written to this page randomly fails
- need to track program/erase operation and restart interrupted one
Questions ?

- Merci pour votre attention !
- Thanks for your attention!
- Questions?