MLC/TLC NAND support: (new ?) challenges for the MTD/NAND subsystem

Boris Brezillon

- Embedded Linux engineer and trainer at Free Electrons
  - Embedded Linux and Android **development**: kernel and driver development, system integration, boot time and power consumption optimization, consulting, etc.
  - Embedded Linux, Linux driver development, Android system and Yocto/OpenEmbedded **training courses**, with materials freely available under a Creative Commons license.
  - `http://free-electrons.com`
- Contributions
  - **Kernel support for the AT91 SoCs** ARM SoCs from Atmel
  - **Kernel support for the sunXi SoCs** ARM SoCs from Allwinner
- Living in **Toulouse**, south west of France

Agenda

Context description
- What is this talk about ?
- NAND Flash technology
- Flash memory handling in Linux

MLC Constraints
- Paired pages
- Unpredictable voltage level
- Data retention problems
- Power-cut related problems

Proposed Solutions
- Paired pages
- Unpredictable voltage level
- Data retention problems

Conclusion

Context: What is this talk about ?

- Explaining the constraints induced by MLC chips and comparing them to SLC chips
- Detailing the current Linux Flash handling stack and pointing missing stuff to properly handle MLC chips
- Going through main MLC constraints and describing existing solutions or proposing new solutions to address them
- Be careful: most of this talk is describing hypothetical changes

Context: Short description of the NAND technology

- Encode bits with Voltage levels
- Start with all bits set to 1
- Programming implies changing some bits from 1 to 0
- Restoring bits to 1 is done via the ERASE operation
- Programming and erasing is not done on a per bit or per byte basis
- Organization
  - Page: minimum unit for PROGRAM operation
  - Block: minimum unit for ERASE operation

Context: NAND Flash organization

Serial Input
(x8 or x16)
30ns (max clk)

Register
2112 bytes

Serial Output
(x8 or x16)
30ns (max clk)

Program:
~300µs

NAND Memory Array
NAND Page 2112 bytes

Read (page load):
~25µs

2048 Blocks
(2Gb device)

64 pages
per block

NAND Block

Block Erase
-2ms

Data Area
2048 bytes

Spare Area
(ECC, etc.)
64 bytes

Context: What are MLC NAND chips ?

- Standard NAND chips are SLC (Single-Level Cells) chips
- MLC stands for Multi-Level Cells
  - Multi is kind of misleading here, we're talking about 4 level cells: b00, b01, b10, b11
  - One cell contains 2 bits
- Bigger than SLC chips, but also less reliable
- Requires more precautions when accessing the chip (true for both read and write accesses)

Context: MLC vs SLC Cell

Voltage

bit 0
0
1

SLC Cell

bit 0	bit 1
0	0
0	1
1	0
1	1

MLC Cell

Context: Flash related layers in the Linux kernel

Layer			
File System Layer	...	JFFS2	UBIFS
Wear Leveling Layer			UBI
Flash Memory Abstraction Layer	MTD		
Flash Type Abstraction Layer	...	...	NAND
Flash Drivers	... Drivers	Drivers	NAND Drivers

Context: MTD

- Provide an abstraction layer to expose all kind of memory devices (RAM, ROM, NOR, NAND, DATAFLASH, ...)
- Does not care about how memory device is accessed: that's MTD driver responsibility
- Expose methods to access the memory device (read/write/erase)
- Expose memory layout information
  - `erasesize`: minimum erase size unit
  - `writesize`: minimum write size unit
  - `oobsize`: extra size to store metadata or ECC data
  - `size`: device size
  - `flags`: information about device type and capabilities
- MTD drivers should fill layout information and access methods in `mtd_info` and then register the device

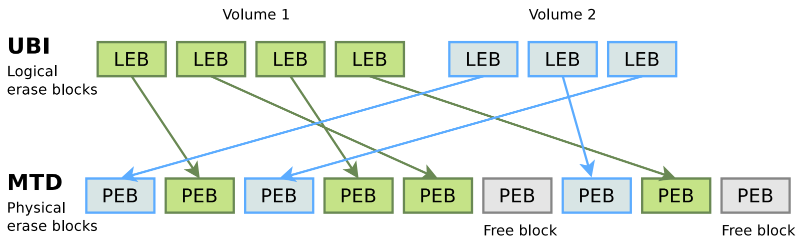
Context: NAND and NAND driver

- Provide an abstraction layer for raw NAND devices
- Take care of registering NAND chips to the MTD layer
- Expose an interface for NAND controllers to register their NAND chips: `struct nand_chip`
- Implement the glue between NAND and MTD logics
- Provide a lot of interfaces for other NAND related stuff:
  - ECC controller: `struct nand_ecc_ctrl`
  - Bad Block handling: `struct nand_bbt_descr`
  - etc

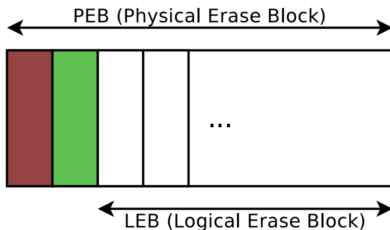
Context: UBI

- Stands for Unsorted Block Interface
- Deal with wear leveling
  - Distribute erase block wear over the whole flash
  - Take care of moving data from unreliable blocks to reliable ones
  - Take care of marking bad blocks (after torturing them)
- Provides a volume abstraction layer
  - Volume are not composed of physically contiguous blocks
  - Volume are not attached specific erase blocks
  - Can be dynamically created, removed, resized or renamed
- Makes use of the MTD abstraction to access memory devices

Context: UBI



context: ubi metadata



Context: UBIFS

- Stands for UBI File System
- Rely on the UBI layer for the wear leveling part
- Journalized file system created to address JFFS2 scalability problems
- I won't detail UBIFS architecture here:
  - It would take too long
  - I'm not qualified enough to describe it

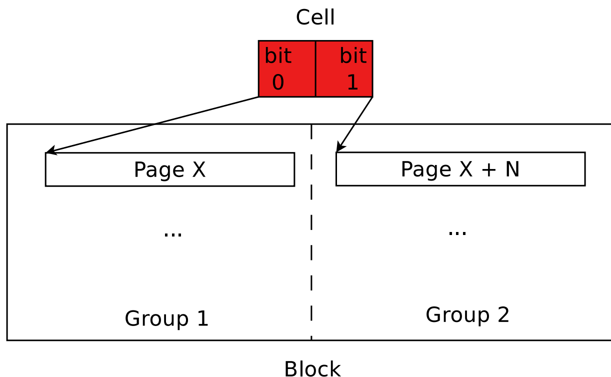
MLC Constraints

- Paired pages impose care when programming a page
- Voltage thresholds delimiting each level might change with wear
- More prone to bit-flips
- Sensitive to systematic data pattern

MLC Constraints: paired pages

- MLC embed 2 bits in each cell
- Why are NAND vendors so mean to us poor software developers ?
  - One bit assigned to one page and the other one to another page
  - TLC cells embed 3 bits: same problem except pages are paired by 3
  - Changing the cell level is a risky operation, which, if interrupted, can lead to undefined voltage level in this cell
  - Since the same cell is shared by several pages, programming one page might corrupt the page(s) it is paired with
  - Each NAND vendor has its own scheme for page pairing, this forces us to provide a vendor specific (if not chip specific) function to get which pages are paired

MLC Constraints: paired pages



MLC Constraints: adapting voltage thresholds

- Voltage level stored when programming a cell might change with wear
- Becomes problematic when the level cross the voltage threshold used by the internal logic to determine values stored in cells
- Can be fixed by ECCs if the number of impacted cells stays low
- Requires a solution when the number of impacted cells is too important
- Solution: move voltage thresholds to deal with this situation

MLC Constraints: adapting voltage threshold

MLC Constraints: data retention

- NAND cells are not indefinitely maintaining their state
- External environment (like temperature) can reduce data retention
- First source of data retention problems are read/write disturbance

MLC Constraints: read and program disturbance

- This problem is seen on all NAND chips (including SLC) but happen more frequently on MLC/TLC NANDs
- Read disturbance:
  - Is caused by a read command
  - Might impact the page currently being read or other pages in the same block
- Program disturbance:
  - Is caused by a program command
  - Might impact other pages in the same block
- The most problematic disturbance are those appearing on other pages than the one being accessed
- Requires scanning all pages (or at least those rarely read) in background to detect those where the number of bit-flips exceed the bit-flips threshold

MLC Constraints: read and program disturbance

Erase Block

Read Data Here

Disturb Data Here

Write Data Here

Disturb Data Here

Write Page

MLC Constraints: avoiding systematic data patterns

- Some MLC chips are sensitive to systematic data patterns
- Scramble data to avoid writing such pattern
- Require a descrambling phase when reading data from the NAND

MLC Constraints: unstable bits

- Not an MLC problem per se (also happens on SLC chips)
- Interrupted PROGRAM/ERASE operations might lead to unstable bits
  - Cells can store the correct value for some time
  - Suddenly return erroneous values
- Fully described here: `http://www.linux-mtd.infradead.org/doc/ubifs.html#L_unstable_bits`

Paired pages handling: First proposal

- Only write on one of the paired pages
- Pros:
  - Simple to implement
  - Can be handle at the NAND layer only
  - Some chips provide an SLC mode (even simpler to implement)
- Cons:
  - You loose half the NAND capacity (even more in case of TLC chips)
- Implementation details:
  - Declare the chip as having half (or one-third in case of TLC) the effective size
  - Use the SLC mode if it exists
  - Or only write on the pages that are assigned the first bit of each cell
  - In any case hide the logic to the upper layers

Paired pages handling: Second proposal

- Differentiate 'safe' and 'unsafe' LEBs
- Safe LEBs: only use one bit of each cell
  - UBI deals with paired pages and expose a linear view to users
  - Users have to take safe LEB size into account
  - Put safe LEB in a pool first time it is unmapped
  - Use pages from the 2nd group when mapped again
  - Erase it the second time it is unmapped
- Unsafe LEBs expose all LEB capacity
  - Users have to deal with paired pages themselves
  - Or accept to loose some data
  - Or atomically program/update LEBs

Paired pages handling: Second proposal

- Pros:
  - Reduce wear (safe LEBs are reused twice before being erased)
  - Provides fine grained control over which operations are sensible and which one are not
- Cons:
  - Still can't use the whole flash capacity
  - More complicated to implement than 1st proposal
  - Impact all layers up to UBIFS
- Usage:
  - Safe LEB: file system journal where each entry should be consistent
  - Unsafe LEBs: atomic LEB update where a CRC is used to ensure whole LEB consistency

Paired pages handling: Second proposal

- Implementation details:
  - NAND and MTD layers are exposing paired pages information
  - UBI should never use pages paired with the EC and VID headers
  - UBI provides a way to declare safe and unsafe LEBs
    - Safe LEBs: only using half (or one-third) of the block capacity so that all writes are safe
    - Safe LEB marker in `ubi_vid_hdr`
  - UBIFS makes use of the unsafe/safe LEB capabilities depending on each operation and the associated required reliability (log update, garbage collection, etc)

Paired pages handling: Third proposal

- Yet to be proposed ;-)
- Give more control to UBIFS ?
- Solution proposed here: `http://www.linux-mtd.infradead.org/doc/ubifs.html#L_ubifs_mlc`
  - Let UBIFS decide when a LEB should be safe (pages paired to the already programmed ones should not be touched)
  - Should be done when committing changes (FS sync) ?
- My knowledge of the UBIFS infrastructure is quite limited
- Should be discussed with the UBIFS Maintainer: Artem Bityutskiy
- UBI should hide pages paired with VID and EC headers
- Pros:
  - Better use of the overall NAND capacity ?
- Cons:
  - Far more complicated to implement: UBIFS has to directly deal with paired pages
  - Only UBIFS will benefit from the paired pages handling (but are there other RW UBI users anyway ?)

Paired pages handling: Third proposal

PEB (Physical Erase Block)

...

...

LEB (Logical Erase Block)

EC header

VID header

Page paired with EC header

Page paired with VID header

Payload

Paired pages handling: Third proposal

LEB

Step1: map LEB

...

Step2: write data

...

Step2: securize data

...

Step2: write new data

...

UBIFS unreliable data

UBIFS reliable data

Paired pages skipped for reliability concern

Empty pages

Write pointer

Handling unpredictable voltage threshold

- NAND vendors provide a way to tweak the cell level threshold, but ...
  - There is no standard way to do that
  - Each vendor implement it differently
  - This might differ even with NAND chips from the same manufacturer
  - While mandatory, this feature is not (or poorly) documented
- Detecting the appropriate threshold is not that simple and this value is only valid for a given block
- It depends on block wear, but there is no paper describing how we should choose it (depends on the number of erase/program cycles, but how ?)
- Iterating over modes implies a performance penalty, since the page has to be read several times

Handling unpredictable voltage threshold

- Micron implementation is already supported in mainline
- But, existing core code ...
  - stops searching for the best read-retry mode as soon as a page is successfully read (even if the number of bit-flips exceed the `bitflips_threshold` value)
  - does not save the last valid read-retry mode: performance penalty at each read
- What's missing ?
  - A way for vendor specific code to be registered (assign the `setup_read_retry` callback)
  - Some fixes to the existing implementation to find the best read-retry mode
  - Optional: store best read-retry mode in memory
  - Optional: guess best read-retry mode from erase counter

Preventing uncorrectable bit-flips: First Proposal

- Regularly read all pages to detect pages/blocks where the bit-flips threshold is raised
- Problem: a page read might generate read disturbance and corrupt other pages in the same block
- Better read a full block
- Solution proposed (and developed) by Richard Weinberger
  - At UBI level
  - Creation of a new user-space interface (sysfs) to trigger a full volume scan
  - Scan done in background (in the UBI thread, or an independent one)

Preventing uncorrectable bit-flips: First Proposal

- Pros:
  - Rather simple implementation
  - Pretty easy to use
  - Let user-space decide when the scan is necessary
- Cons:
  - Force user-space to store information on the last scan and logic about when to scan next time
  - Launching a full scan might be ineffective in some cases (some blocks are read quite often and do not need to be scanned)
  - Performance penalty when reading/programming while a scan is in progress (the operation might have to wait for the page read to finish)

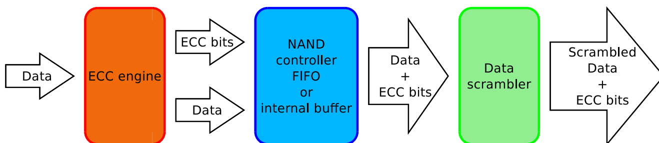
Preventing uncorrectable bit-flips: Second Proposal

- UBI layer can store useful information/statistics about
  - read and write accesses
  - number of corrected bit-flips
- UBI can make use of these statistics to decide when to read each page/block
- Pros:
  - All the complexity is hidden to user-space
  - More efficient in term of useful page/block reads
- Cons:
  - Far more complicated to implement
  - Increase memory footprint
  - Still require one full scan at boot (to restore the database)
  - Performance penalty when reading/programming while a bit-flip detection is in progress

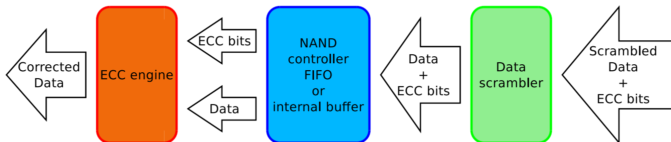
Avoiding systematic data pattern: data scramblers

- Should be handled in the NAND layer
- Better use a hardware scrambler, but software implementation is possible
- Same approach as for ECC handling

Avoiding systematic data pattern: data scramblers



Write page workflow



Read page workflow

Data scrambler: implementation details

- Data scrambling can be hidden in NAND controller driver's implementation, but
  - You'll have to use your own read/write implementations
  - If we ever decide to add a mode to disable the scrambler when accessing the NAND, you'll have to implement more functions
  - Factorizing common operations in default helper functions is always a good thing
  - Trying to match a common model always makes you think twice before coding dirty hacks ;-)
- The proposed interface is trying to be as much generic as possible, but was designed with 2 implementations in mind
  - The sunxi NAND controller one
  - A software based implementation using the LFSR algorithm
- Please let me know if your scrambler does not fit in the model proposed here

Data scrambler: implementation details

```
enum nand_scrambler_action {
  NAND_SCRAMBLER_DISABLE,
  NAND_SCRAMBLER_READ,
  NAND_SCRAMBLER_WRITE,
};

struct nand_scrambler_ops {
  int (*config)(struct mtd_info *mtd, int page, int column,
                enum nand_scrambler_action action);
  void (*write_buf)(struct mtd_info *mtd, const uint8_t *buf, int len);
  void (*read_buf)(struct mtd_info *mtd, uint8_t *buf, int len);
};

struct nand_scrambler_layout {
  int nranges;
  struct nand_rndfree ranges[0];
};

struct nand_scrambler_ctrl {
  struct nand_scrambler_layout *layout;
  struct nand_scrambler_ops *ops;
};

[...]

struct nand_chip {
[...]
  struct nand_scrambler_ctrl *scrambler;
[...]
};
```

Data scrambler: implementation details

- Scrambler layout (`struct nand_scrambler_layout`)
  - Describes area that should not be scrambled
  - Particularly useful for Bad Block Markers
  - Not mandatory but highly recommended if feasible
- Scrambler operations (`struct nand_scrambler_ops`)
  - `config`
    - configure the scrambler block for a `READ` or `WRITE` operation, or disable it
    - `page` and `column` arguments are necessary to setup the appropriate key or seed value in the scrambler block
  - `read_buf` and `write_buf`
    - wrapper functions responsible for enabling the scrambler block before calling NAND controller `read_buf` or `write_buf` and disabling it after the operation is done
    - Not mandatory if you do not rely on default helpers

Data scrambler: implementation details

- Proposed an implementation a year ago: `https://lkml.org/lkml/2014/4/30/721`
- Proof of concept available here: `https://github.com/bbrezillon/linux-sunxi/tree/sunxi-nand-next`

Unstable bits handling

- Part of a solution described here: `http://www.linux-mtd.infradead.org/doc/ubifs.html#L_unstable_bits`
- That's a topic I haven't thought about yet
- Any proposal is welcome

We need NAND chip vendors help

- Most solution proposed in this talk are based on experiments and not facts or statistics
- NAND chip vendors could help us by
  - Documenting undocumented or (poorly documented) parts
    - How to change voltage threshold
    - Impacts of systematic data pattern
    - Impacts of power-cut failures on data reliability (unstable bits issue)
  - Providing statistics on
    - Cells wear evolution
    - Impacts of wear on voltage level
    - Impacts of read/write disturbance (to determine how often a block should be scanned)
  - Proposing new approaches to deal with MLC constraints

What's next ?

- Most of the solution proposed here are either untested ones or just proof of concepts
  - Need to discuss them with MTD, UBI and UBIFS maintainers
  - Provide MLC chips constraints emulation in order to test UBI/UBIFS MLC related stuff with checkfs
  - Provide implementations and iterate till they are accepted
- Doing that on my spare time: don't expect to see things coming quickly
- Any kind of help is welcome: new ideas, implementations, tests, reviews, ...

Questions?

Boris Brezillon

boris.brezillon@free-electrons.com

Slides under CC-BY-SA 3.0

http://free-electrons.com/pub/conferences/2014/elce/brezillon-drm-kms/