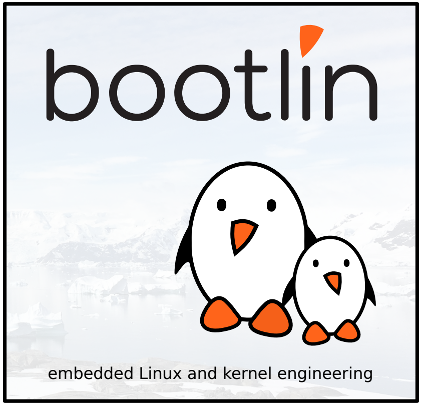
Running UBI/UBIFS on MLC NAND

Boris Brezillon, Richard Weinberger
boris@bootlin.com, richard@sigma-star.at



Running UBI/UBIFS on MLC NAND

MLC NANDs constraints: where the nightmare begins

SLC NANDs: Boring, our happy place

MLC NANDs: A nightmare

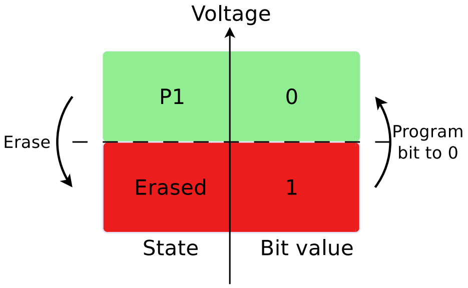
MLC NAND constraints: What we knew

- Reduced lifetime: limited number of P/E cycles (5000-10000).
- Pages within an erase block have to be programmed in ascending order.
- Data retention issues: pretty much the same problem we see on SLC NANDs, but a different order of magnitude.
- Paired pages: why the hell did they decide to assign the same cell to different NAND pages?
- Unstable bits: you'd better make sure your board has enough power to finish the current erase or program operation, and prevent future operations from happening when you are about to experience a power-cut.

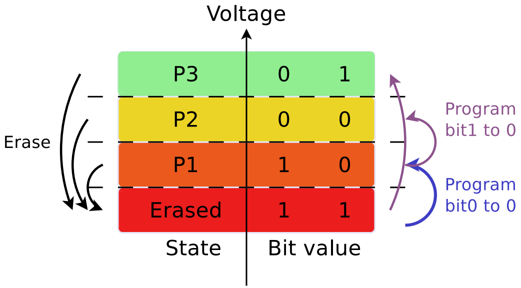
MLC NAND constraints: Reduced lifetime

- Not a big problem: we just need to make sure we correctly distribute the wear over the NAND device.
- UBI is already taking care of that.
- May require some tweaking at the UBI level: the default values are not suitable for MLC NANDs.

MLC NAND constraints: MLC cells



SLC cell



MLC cell

MLC NAND constraints: Data retention issues (1)

- A NAND cell can see its state changed (charge loss or gain): this is what we call bitflips.
- Two known sources:
  - Read/write disturbance: reading/programming a cell can change neighbor cells state
  - Inherent charge loss: NAND cells tend to lose their charge over-time. This is emphasized when NAND cells are worn out.
- Solutions:
  - Increase the ECC strength: usually, this is NAND controller dependent, and even with a strong ECC, you'll hit uncorrectable errors at some point.
  - Regularly read NAND eraseblocks to detect those that are showing a lot of bitflips, and move the data somewhere else before the ECC is unable to correct these errors.

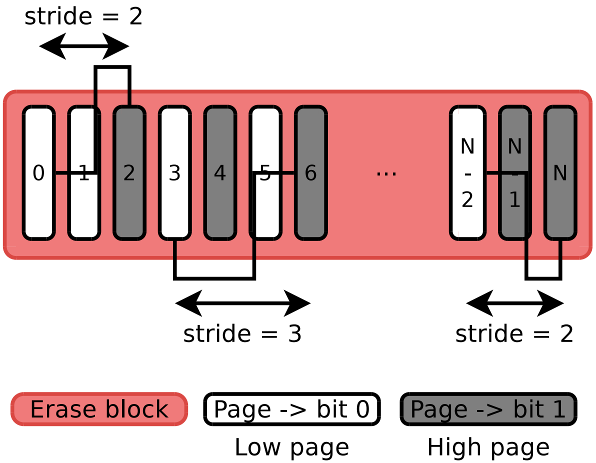
MLC NAND constraints: Data retention issues (2)

- MLC cells expose 4 different states instead of 2 states for SLC cells.
- In MLC cells, distances between states are smaller: data retention issues are emphasized.
- The smaller the process the closer the cells, which makes read/write disturbance even worse.

MLC NAND constraints: Paired pages

- This is one of the biggest problem we have with MLC NANDs.
- MLC cells expose two bits.
- Each bit is assigned to a different page.
- Pages sharing the same cells are called "Paired pages".
- Paired pages are usually not contiguous.
- Pages have to be programmed in order.
- When programming the second page of a pair, you may corrupt the first page of the pair if the operation is interrupted.

MLC NAND constraints: Paired pages



MLC NAND constraints: Unstable bits

- Problem described here:
  `http://www.linux-mtd.infradead.org/doc/ubifs.html#L_unstable_bits`
- Reported by some people on the MTD mailing list.
- We did not see it during our experimentation.
- NAND vendors do not clearly communicate on this problem.
- What is described as a black and white problem may actually be more subtle:
  - Interrupting a program or erase operation might leave the page/block in a 'valid' state but with a lot of bitflips.
  - UBI should be able to handle this case.
- We need more information from vendors to predictably reproduce the problem.

MLC NAND constraints: What we discovered (bad news)

- Modern MLC NANDs require data scrambling.
- Modern MLC NANDs show a high number of bitflips in the last set of programmed pages if the erase block is partially written.
- Both aspects seem to be related:
  - NAND vendors try to mitigate the write-disturb effect by accounting for future write-disturb when programming a page.
  - These write-disturb prediction models are assuming random data, hence the need for data scrambling/randomization.
- Let's call this problem the 'open block' issue.

MLC NAND constraints: What we discovered (good news)

- MLC NANDs can be programmed in 'SLC mode' (only write the first page of each pair)
  - This solves the paired pages problem.
  - Erase blocks written in 'SLC mode' show less bitflips.
  - Erase blocks written in 'SLC mode' are less sensitive to read/write disturbance.
- To summarize, 'SLC mode' makes MLC NAND usable.
- But this also means exposing half the storage capacity.

MLC NAND constraints: What we discovered

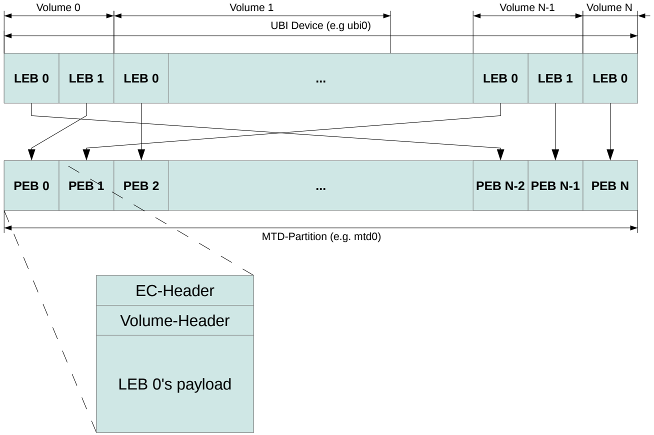
- It's a shame we had to discover those problems while developing the solution.
- More details from NAND vendors would be great.
- Yes, I'm looking at you, Samsung, Hynix, Toshiba, ...

UBI: A quick reminder

First some basics: UBI nomenclature

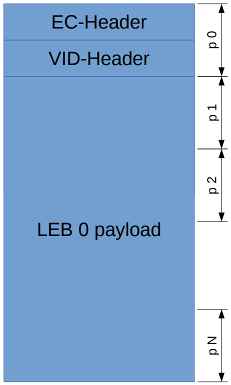
PEB	Physical erase block.
LEB	Logical erase block.
Image	UBI on your MTD partition.
Device	Runtime representation of your UBI image (i.e. **/dev/ubi0**).
Volume	A volume within the UBI image (i.e. **/dev/ubi0_0**).
Attach	Process of loading an UBI image (i.e. attach mtd0).

UBI example



Volume 0
Volume 1
Volume N-1
Volume N
UBI Device (e.g ubi0)
LEB 0
LEB 1
LEB 0
...
LEB 0
LEB 1
LEB 0
PEB 0
PEB 1
PEB 2
...
PEB N-2
PEB N-1
PEB N
MTD-Partition (e.g. mtd0)
EC-Header
Volume-Header
LEB 0's payload

UBI PEB on SLC NAND



- Erase count and volume ID header are located on the first page.
- If subpages are not supported, volume ID header will be placed on the second page.
- Other pages are used for payload.

Addressing the data retention issue

Dealing with data retention issues

- On MLC, read/write disturb is a serious problem.
- Reading back all data from time to time is needed to detect bitflips.
- ubihealthd queries read counters from userspace and triggers reads.

ubihealthd

- On SLC, and without Fastmap, a weekly **dd if=/dev/ubi0_X of=/dev/null** was enough.
- On MLC read/write disturb happens sooner.
- Instead of doing a bulk read from time to time, ubihealthd monitors read counters and triggers a read of the whole PEB. If bitflips are detected UBI will scrub the PEB.

Running UBI/UBIFS on MLC NAND

Addressing the paired pages issue

Expose page pairing schemes (the easy part)

- Various page pairing scheme found in the wild.
- Most pairing schemes follow a predictable logic.
- Can be described without a full association table.
- New interface to describe these pairing schemes.
- The NAND framework will provide an implementation for each pairing scheme.
- Pairing scheme selection should be based on NAND detection information.
- MTD provides an API to let MTD users query pairing information.

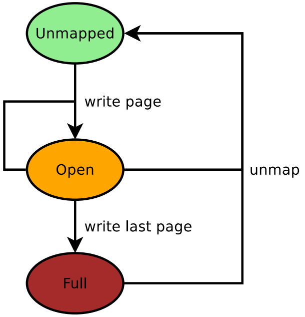
Handle the problem in UBIFS (our first attempts)

- Proposals:
  1. Write everything in 'SLC mode' and consolidate UBIFS nodes in MLC mode when we are running out of space.
  2. Write everything in 'MLC mode' (except critical data) and skip pages when we need to secure data (at sync/commit time).
- Pros:
  - UBIFS knows exactly which data are valid and dirty.
  - UBIFS knows when data protection is required (i.e. at sync and commit time).
- Cons:
  - Solution 2 is not possible because of the convoluted pairing schemes and the 'open block' issue.
  - Solution 1 is too invasive: requires exposing two modes with different LEB size.

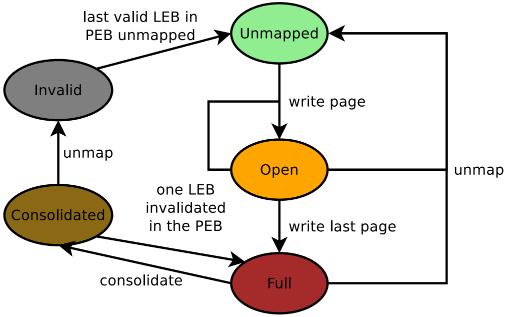
Handle the problem in UBI

- By default every LEB is written in 'SLC mode'.
- Obviously we'd waste half the storage capacity.
- As soon we run out of free erase blocks, UBI will consolidate two fully written SLC LEBs into one MLC PEB and we gain one more LEB which can be written again in 'SLC mode'.
- That way we can utilize all pages.
- Sounds pretty easy, right?

LEB consolidation: LEB states



Without LEB consolidation



With LEB consolidation

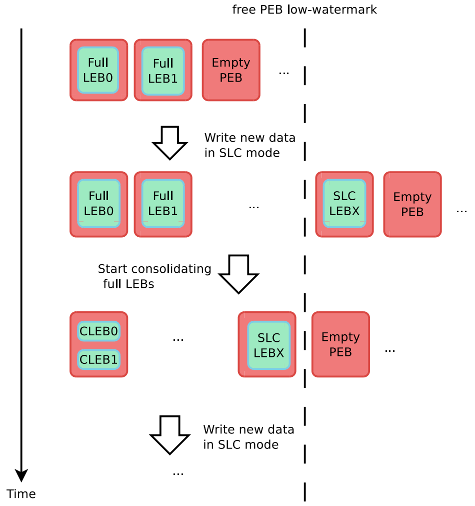
LEB consolidation: when does it take place?

- As soon as all low pages of a PEB are written the corresponding LEB is considered full and becomes a candidate for consolidation.
- Bonus question: Why has it to be full?
- To produce one free PEB, UBI selects two PEBs that host full LEBs, copies their data into a new PEB in 'MLC mode' and erases both source PEBs.
- Means that on the target PEB both low and high pages are used.
- This operation is more or less an atomic LEB change operation. So we can reuse a lot of UBI's infrastructure.
- A PEB in 'MLC mode' always hosts two LEBs, therefore right after the EC header it has two VID headers. One for each LEB. So, we have to introduce a new UBI on-flash format.

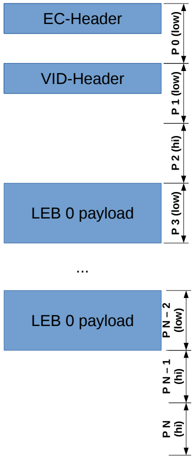
UBI extended nomenclature

PEB	Physical erase block.
LEB	Logical erase block.
SLC LEB	A LEB where data sits only on low pages.
Full LEB	A SLC LEB where the last page has been written.
CLEB	A consolidated LEB hosted on a PEB where low and high pages are used.
Image	UBI on your MTD partition.
Device	Runtime representation of your UBI image (i.e. **/dev/ubi0**).
Volume	A volume within the UBI image (i.e. **/dev/ubi0_0**).
Attach	Process of loading an UBI image (i.e. attach mtd0).

LEB consolidation: trigger

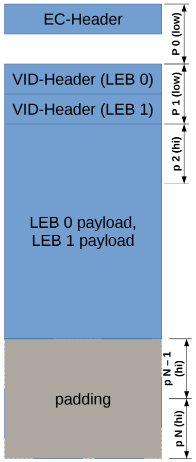


UBI PEB in SLC mode on MLC NAND



- MLC does not support subpages.
- Only low pages are used, because writes to high pages are more likely to corrupt data.
- Page 0 hosts erase count header.
- The next page, page 1 in this example, the volume ID header.
- All remaining pages are used for payload.

UBI PEB in MLC mode on MLC NAND

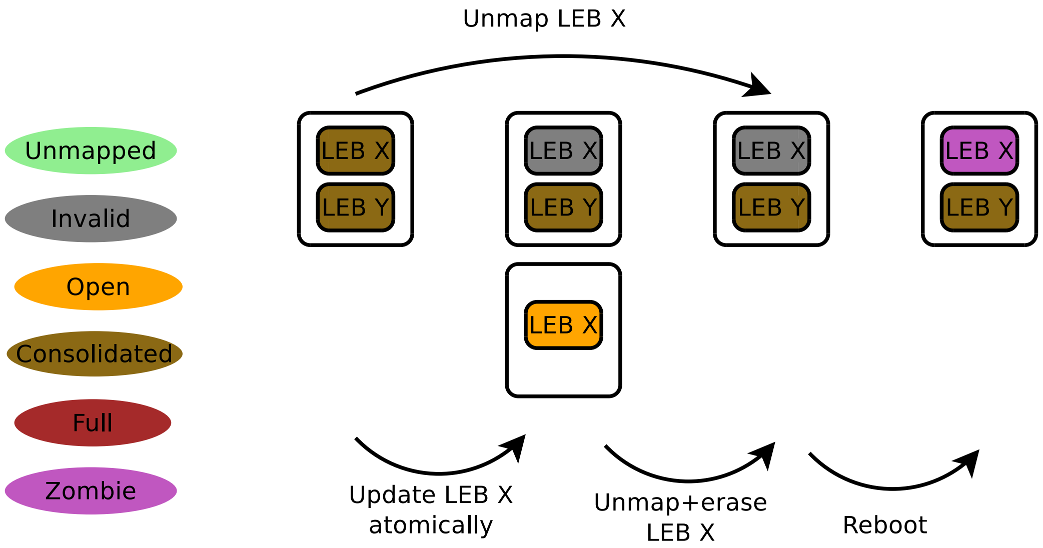


- Second page hosts two volume ID headers, one for each LEB payload.
- Both low and high pages are used.
- Stored data is immutable.

Time for some hard-hitting details

- SLC NAND: 1:1 mapping between PEBs and LEBs.
- MLC NAND: 1:2 mapping between PEBs and LEBS.
- The LEB and PEB concepts are sometime inter-mixed in UBI, leading to confusion with the 1:2 mapping.
- Space reservation logic is acting at the PEB level.
- Leads to `-ENOSPC` if we consider `nLEBS = nPEBS x 2`
- Require some guarantees from the UBI user:
  - Must fill LEBs entirely.
  - Must limit the number of LEBs opened simultaneously.
  - Must inform UBI about this number.

Fancy new corner case: zombie LEB



Fancy new corner case: zombie LEB

- Solutions:
  1. Keep track of invalidated LEBs in UBI (using a log?).
  2. Force UBI users to use atomic updates when necessary.
- We chose solution 2:
  - Patch UBIFS to use atomic update instead of unmap when the LEB is still referenced.
  - Continue to use unmap when the LEB is not referenced.
  - Unmap all unreferenced LEBs at each boot to update UBI's invalidated LEB database.
- We might decide to switch to solution 1 at some point.

Running UBI/UBIFS on MLC NAND

Conclusions

Finally, how to use UBI (and UBIFS) on MLC NAND?

- Teach MTD to know your NAND, especially the pairing.
- Also make sure you have strong ECC.
- When using ubinize, pass the type of the pairing scheme to it.
  - For SLC NAND, PEB and minimum input/output unit size are enough.
  - To produce MLC NAND images we had to teach ubinize about pairing schemes.
  - So far only two schemes are supported.
  - Schemes have to be kept in sync with the kernel.
    Better solutions?
- Run ubihealthd.

Lessons learned

- LEB consolidation is slow and source of contentions.
- Space reservation is not simple.
- LEB and PEB concepts have to be clarified/split.
- Full LEBs are not necessarily the best candidates for consolidation.
- Long story short, we have a working PoC but it still needs work.
- We are working on a second version addressing these aspects:
  - Relax locking and work with page chunks when consolidating LEBs.
  - Fix PEB reservation by moving consolidation at the volume level.
  - Rework selection of candidates for consolidation.

Questions? Suggestions? Comments?

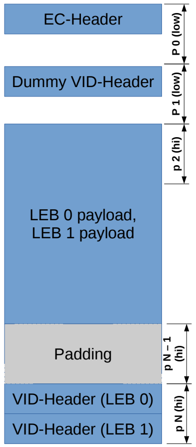
Boris Brezillon, Richard Weinberger

boris.brezillon@bootlin.com, richard@sigma-star.at

Running UBI/UBIFS on MLC NAND

Backup slides

V2 plans: How to make consolidation less invasive

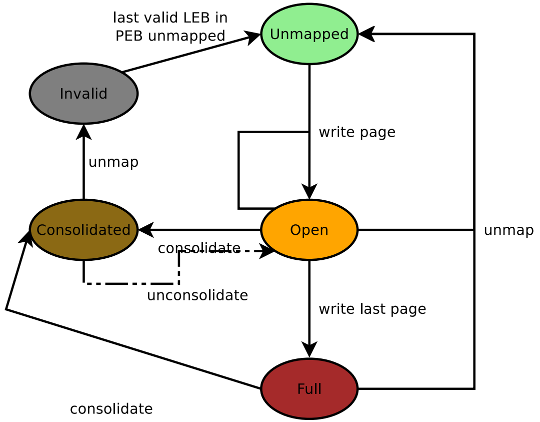


- We use the VID headers as end marker.
- If present and valid we can assume that the write process was not interrupted.
- No need to compute a checksum of all data.
- UBI attach will be faster.

V2 plans: Fix LEB/PEB reservation

- Limit consolidation scope to volumes.
- Reserve PEBs ahead of time.
- Keep a certain amount of PEBs in SLC mode (5%).
- Rework selection of candidates for consolidation: LRU (Least Recently Updated)
  - Get rid of the 'max number of open LEBs' limitation.
  - Get rid of the 'entirely fill LEBs' limitation.
  - More efficient than the 'consolidate full LEBs' approach.

V2 plans: LEB states


last valid LEB in PEB unmapped
Unmapped
Invalid
write page
unmap
Consolidated
Open
unmap
consolidate
unconsolidate
write last page
Full
consolidate