# Memory Technology Devices: what's new?

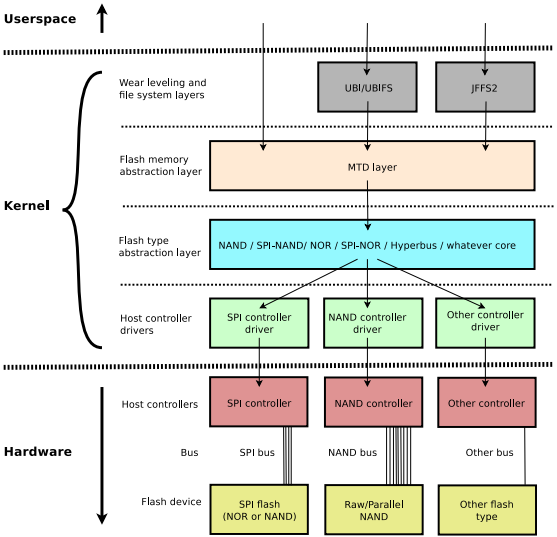
Miquèl Raynal
*miquel@bootlin.com*

Richard Weinberger
*richard@sigma-star.at*

bootlin

sigma star

# What exactly are we talking about?



# Administration

- Lot's of changes in the maintainers team

– Adrian (UBIFS)
– Artem (UBI/UBIFS)
– Boris (MTD/NAND)
– Brian (MTD)
– Cyrille (SPI-NOR)
– David (MTD)
– Marek (MTD)

• Richard (MTD/NAND/UBI/UBIFS/JFFS2), aka the pillar

\+ Frieder (MTD)
\+ Miquèl (MTD/NAND)
\+ Tudor (SPI-NOR)
\+ Vignesh (MTD/HyperBus)

# Where is the latest code?

- Migration to a kernel.org repository:
  ~~https://git.infradead.org/linux-mtd.git~~
  https://git.kernel.org/pub/scm/linux/kernel/git/mtd/linux.git
  - mtd/fixes (all fixes)
  - mtd/next (MTD + parallel NORs + HyperBus)
  - nand/next (all NAND flavors)
  - spi-nor/next (SPI-NORs only)

- UBI/UBIFS development is now also on kernel.org:
  https://git.kernel.org/pub/scm/linux/kernel/git/rw/ubifs.git

# MTD layer

# Panic write flag

- MTD supports panic write since ever
- Useful to dump kernel crashes to flash
- During panic we must not do complex stuff
- Before v5.3 drivers had no way to detect panic write
- Now drivers can check the new `oops_panic_write` flag

# Flash type layers

# NAND: Keep it tidy!

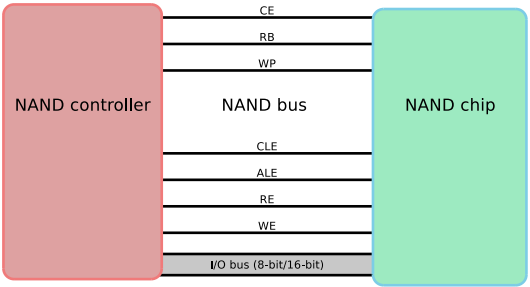
- `mv items/ where-they-belong/`
  - Creation of a `raw/` subdirectory
  - Onenand drivers moved into the `nand/` subdirectory
  - Code reorganization:
    - Timings
    - ONFI
    - JEDEC
    - ...

# NAND: Generic NAND layer

- Common to all flavors of NAND chips
- Abstracts the type of bus (ops)
- Describes the memory organization
  - targets
  - luns
  - planes
  - eraseblocks
  - pages
  - ...
- Shape the I/O requests
  - in-band data offset
  - in-band data length
  - out-of-band data offset
  - out-of-band data length
  - buffers
  - ...
- Share Bad Block Table handling and Bad Blocks logic
  - init
  - cleanup
  - read
  - update

# Raw NAND: Jumbo cleanup 1/2

- Make a distinction between the controller and the flash chip



# Raw NAND: Jumbo cleanup 2/2

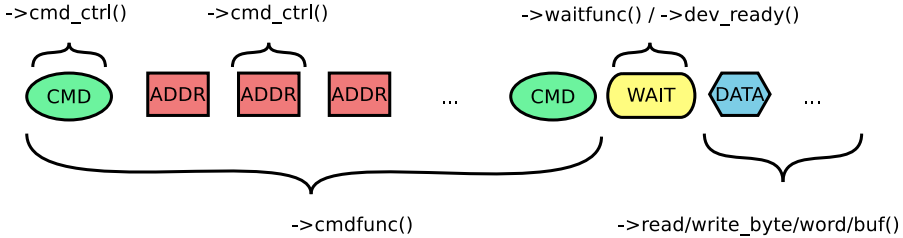
- Deprecation of several hooks, creation of a legacy structure

```
drivers/mtd/nand/raw/nand_base.c   | 618 +-------------------------
drivers/mtd/nand/raw/nand_legacy.c | 642 ++++++++++++++++++++++++++++
```

- No more MTD objects in the NAND world

```
- * @mtd: MTD device structure
+ * @chip: NAND chip object
[...]
-static int nand_block_markbad_lowlevel(struct mtd_info *mtd, ...)
+static int nand_block_markbad_lowlevel(struct nand_chip *chip, ...)
 {
-       struct nand_chip *chip = mtd_to_nand(mtd);
+       struct mtd_info *mtd = nand_to_mtd(chip);
```

# Raw NAND: `->exec_op()`: origins

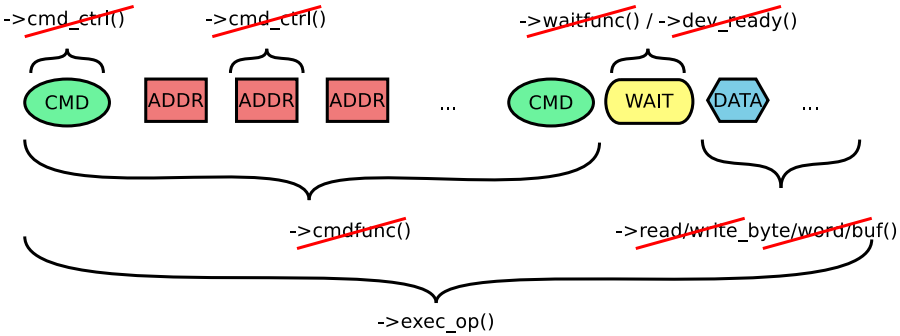



- Controllers can usually handle more complex/higher-level operations (sometimes they cannot even send a single cycle)
- The API is incomplete

- Workaround for driver developers: overload `->cmdfunc()`
  → re-implementing everything?
  - Hard to maintain
  - Incomplete implementations

# Raw NAND: `->exec_op()`: one hook to rule them all

- Asks to execute the whole operation
- Just a translation in NAND operations of the MTD layer orders
- An array of instructions is passed
- The controller's implementation will parse the sequence, split it in as many sub-operations as needed/supported and execute



# Raw NAND: A journey supporting Micron's hardware

- a.k.a: why we tend not to believe hardware manufacturers?

# Raw NAND: A journey supporting Micron's hardware 1/3

Plot What is the on-die ECC engine capability of a Micron NAND chip, between: unsupported, supported and mandatory?

# Raw NAND: A journey supporting Micron's hardware 1/3

**Plot** What is the on-die ECC engine capability of a Micron NAND chip, between: unsupported, supported and mandatory?

**Solution** Discover these information statically: build an ID $\leftrightarrow$ capabilities table.

# Raw NAND: A journey supporting Micron's hardware 1/3

**Plot** What is the on-die ECC engine capability of a Micron NAND chip, between: unsupported, supported and mandatory?

**Solution** Discover these information statically: build an ID $\leftrightarrow$ capabilities table.

**Issue 1** It seems that Micron produces many chips with the same ID and blow internal fuses to prevent the advertisement and the use of on-die ECC on outgoing products depending if their on-die ECC tests passed or not. So only the ID, as-is, is unusable.

# Raw NAND: A journey supporting Micron's hardware 1/3

**Plot** What is the on-die ECC engine capability of a Micron NAND chip, between: unsupported, supported and mandatory?

**Solution** Discover these information statically: build an ID $\leftrightarrow$ capabilities table.

**Issue 1** It seems that Micron produces many chips with the same ID and blow internal fuses to prevent the advertisement and the use of on-die ECC on outgoing products depending if their on-die ECC tests passed or not. So only the ID, as-is, is unusable.

**Fix 1** Discover these information dynamically: enable the engine with `->set_features()` then read back the ECC engine status with `->get_features()`. With this information we know if the engine is supported. Disable it with `->set_features()`. If `->get_features()` report the engine is still on, on-die ECC is mandatory.

# Raw NAND: A journey supporting Micron's hardware 2/3

Issue 2 On certain chips where on-die ECC is supposedly mandatory, `->get_features()` returns that it is disabled.

# Raw NAND: A journey supporting Micron's hardware 2/3

**Issue 2** On certain chips where on-die ECC is supposedly mandatory, `->get_features()` returns that it is disabled.

**Fix 2** Use the `ECC_STATUS` bit of the 5th ID byte as discriminant, our observations shows that it is dynamically changed and it reflects what `->get_features()` fails to retrieve.

# Raw NAND: A journey supporting Micron's hardware 2/3

**Issue 2** On certain chips where on-die ECC is supposedly mandatory, `->get_features()` returns that it is disabled.

**Fix 2** Use the `ECC_STATUS` bit of the 5th ID byte as discriminant, our observations shows that it is dynamically changed and it reflects what `->get_features()` fails to retrieve.

**Issue 3** Has explained above, a chip without on-die ECC does not mean there is no ECC engine on it, it means it is at least unavailable. But, there are buggy chips where enabling the engine with a `->set_feature()` will work despite the fact that the engine is not supposed to be available and will probably not work reliably!

# Raw NAND: A journey supporting Micron's hardware 2/3

**Issue 2** On certain chips where on-die ECC is supposedly mandatory, `->get_features()` returns that it is disabled.

**Fix 2** Use the `ECC_STATUS` bit of the 5th ID byte as discriminant, our observations shows that it is dynamically changed and it reflects what `->get_features()` fails to retrieve.

**Issue 3** Has explained above, a chip without on-die ECC does not mean there is no ECC engine on it, it means it is at least unavailable. But, there are buggy chips where enabling the engine with a `->set_feature()` will work despite the fact that the engine is not supposed to be available and will probably not work reliably!

**Fix 3** Let's check **first** the `ECC_STATUS` bit, and activate the engine only if the default state reflects an available engine. The engine will still be activated and deactivated if it appears to be present and usable, in order to know if it is mandatory.

# Raw NAND: A journey supporting Micron's hardware 3/3

Issue 4 We found chips that do not update the `ECC_STATUS` bit when the engine is enabled.

# Raw NAND: A journey supporting Micron's hardware 3/3

Issue 4 We found chips that do not update the `ECC_STATUS` bit when the engine is enabled.

Fix 4 Micron advises to rely on `->get_features()`. Oh, wait...

- Back to endless part number tables. Maybe they did mess that too?

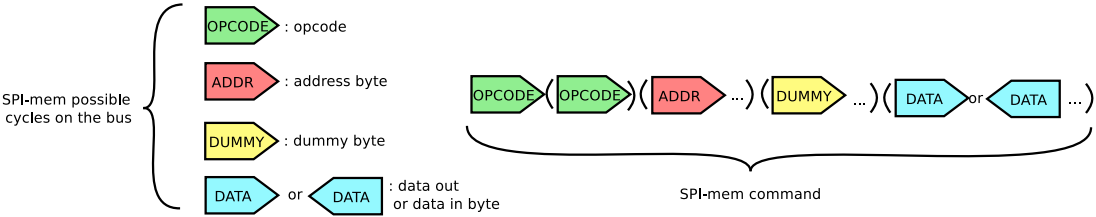
# Raw NAND: Supporting more and more hardware

- Raw NANDs should have died by now
- New controllers (or IP flavors):
  - MTK, Tegra, STM32 FMC2, Amlogic, Brcmnand, Macronix
- Cleaned, updated, migrated to use the new hooks:
  - Marvell, OMAP2, Sunxi, GPMI, Qualcomm, Denali, VF610, FSMC, Ams-Delta

- Not dead enough from my perspective :)
- What does my crystal ball says?

# SPI memories

- SPI controllers are memory agnostic
- Introduction of the `spi-mem` layer
- Standardizes how to communicate with a device




- Command set is device specific (2 known at this time)
- Supports direct-mapping!
- Generic SPI controller drivers can implement the `spi_mem_ops` interface

# SPI-NAND

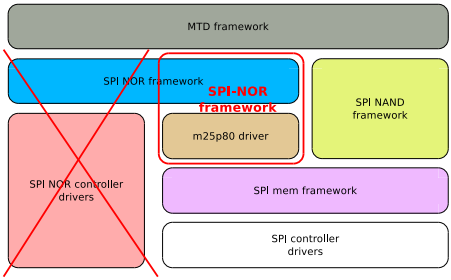
- Introduction of a real SPI-NAND framework, built on top of `spi-mem`
- Similarly to `->exec_op()`, translates the upper layer I/O requests into SPI cycles thanks to the `spi-mem` abstraction layer

- Very active field, could replace raw memories in the near future due to the parallelization of data sampling (dual-spi, quad-spi, octo-spi) abstracted by the `spi-mem` layer.
- Plenty of new chips already supported, keeps increasing!
  - Winbond, Gigadevices, Toshiba, Micron, Macronix, Paragon

- Staging `mt29f_spinand` driver removed

# SPI-NOR: Fit the `spi-mem` layer

- Much older than the SPI-NAND framework
- Re-worked to fit the `spi-mem` approach
  - Temporary use of the m28p80 driver as intermediate layer between SPI-NOR and `spi-mem`
  - Structural changes to move generic code to the SPI-NOR (`spi-nor.c`) layer
  - Get rid of manufacturer specific code in the generic bits




- Goal is to get rid of the SPI-NOR controller drivers entirely

# SPI-NOR: Improvements

- DMA'able buffers mandatory now (assumed by the `spi-mem` layer)
- Lock/unlock logic reworked
- Notable new features:
  - System power management support
  - Non-uniform erase size
  - JEDEC SFDP 4-byte Address Instruction Table parser
  - `->default_init()` and `->post_sfdp()` hooks to tweak the flash parameters
  - Octal mode
  - ...

# HyperBus framework

- 29/10/2019 @ Lyon: Vignesh talk about HyperBus support in Linux

- Physically similar to Octo-SPI
- Double data rate
- Part of xSPI standard JEDEC 251 (standardizes all SPI-NOR flashes)
- Different protocol than what traditional SPI NOR chips use

- HyperFlash
  - NOR based memory with HyberBus interface
  - Follows CFI Parallel NOR command set
    `drivers/mtd/chips/cfi_cmdset_0002.c`
  - Command set encapsulated by the HyperBus protocol

# Wear leveling and file-system layers

# UBI: Dealing with read disturb

- Reading the same page over and over can corrupt nearby pages
- Big issue on MLC NAND but also possible on SLC NAND
- Problem: Seldom read pages might corrupt faster than you can fix them

# UBI: Dealing with read disturb

- Old solution: Read whole NAND once in a while
- . . . reading on what layer?
- UBI does wear leveling for us, so we need to read at UBI level
- Once a week: `dd if=/dev/ubiX_Y of=/dev/null`
- Done?

# UBI: Dealing with read disturb

- Problem 1: Hurts performance
- Problem 2: Will only catch pages with UBI user data, no meta data
- Solution for problem 2: Reboot also once in a while
- Upon UBI attach UBI reads all meta data and can detect errors

# UBI: Dealing with read disturb

- Fastmap makes things interesting, as always
- At attach time most meta data is not read
- UBI meta data can bitrot and corrupt badly

# UBI: Bitrot interface

- Simple UBI ioctl interface to userspace
- Allows userspace to initiate checks
- PEBs can be in many different state, userspace has to restart checks
- Simple but handy userspace daemon: ubihealthd

# UBI: ubihealthd

- Part of mtd-utils package
- Stateless, no config file, no state file needed
- Shuffles list of PEBs and checks every block one by one
- One PEB per 120 seconds, default value

# UBIFS: fstests

- fstests (former xfstests) can now test UBIFS
- Useful for regression testing
- Some tests can fail, e.g. atime tests

# UBIFS: Authentication

- Beside of encryption you also want your data authenticated
- UBIFS authentication can be combined with encryption
- UBIFS encryption via fscrypt covers only user data
- UBIFS authentication covers user and meta data
- e.g. switching inodes of `/bin/login` and `/bin/true` will be detected

# UBIFS: Bug fixes

- xattrs saw major rework
- `O_TMPFILE` needed multiple fixes, sadly

# What next?

- Definitely coming
  - NAND
    - External ECC engines
    - MLC support (in pseudo SLC mode)
  - HyperBus
    - HyperRAM support
    - Use of spi-mem layer
- Keep on dreaming
  - ONFI for SPI-NANDs? Maybe, maybe not
  - DMA-able buffers everywhere in MTD (implies new I/O requests structures)
  - An MTD I/O scheduler, parallelized access
    - Running upstream Linux in SSDs?

# Questions? Suggestions? Comments?

Miquèl Raynal & Richard Weinberger

miquel@bootlin.com and richard@sigma-star.at



https://bootlin.com/pub/conferences/2019/elce/raynal-weinberger-what-s-new