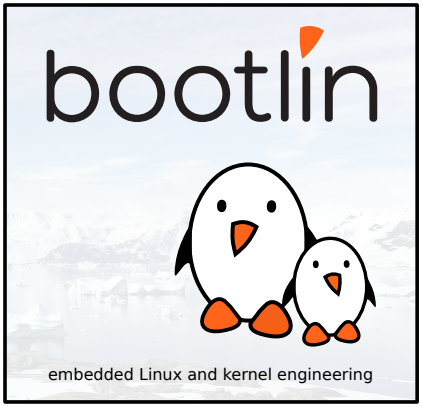
SPI Memory support in Linux and U-Boot

Miquèl Raynal
miquel@bootlin.com

Boris Brezillon
boris@bootlin.com



Miquèl Raynal

- Embedded Linux engineer at Bootlin
  - Embedded Linux **development**: kernel and driver development, system integration, boot time and power consumption optimization, consulting, etc.
  - Embedded Linux, Linux driver development, Yocto Project / OpenEmbedded and Buildroot **training courses**, with materials freely available under a Creative Commons license.
  - `https://bootlin.com`
- Contributions
  - **Maintainer of the NAND subsystem**
  - **Kernel support for various ARM SoCs**
- Living in **Toulouse**, south west of France

How we feel when talking about MTD

Children's drawing center, Michal Wimmer

MTD

Van Gogh's "Starry Night" painting (Wikimedia Commons CC-BY-SA 3.0)

Other subsystems

What is this talk about?

- Understanding what SPI memories are and what protocol they use
- Looking at the Linux (and U-Boot) SPI memory stack (both past and present)
- Have a glimpse of future `spi-mem` framework evolutions
- Getting feedback from developers/users (if any in this room)

SPI bus evolutions: let's start small

- The SPI protocol started as a simple 4-wires protocol
  - CS: Chip Select
  - SCK: Serial Clock
  - MISO: Master In Slave Out
  - MOSI: Master Out Slave In
- Relatively high frequency (usually above 10MHz)
- Full-duplex by nature
- Master-Slave approach:
  - Only one master in control
  - Each slave has its own CS line

Jeremy Clarkson entering a Peel P50, Top Gear, BBC

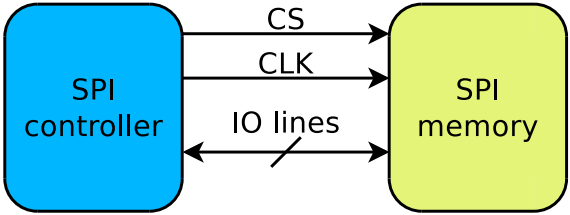
SPI bus evolutions: we need more juice!

- SPI is good, but not fast enough for some use cases, like storage
- Solutions to address this limitation
  - Increase SCK frequency: some devices now support speed above 100MHz
  - Increase the I/O bus width: Dual SPI, Quad SPI and now Octo SPI
  - DDR mode: data are sampled on both SCK edges
- All these solutions come with extra cost:
  - More complex to implement
  - Quad and Octo modes require more pins

Photography: Vladislav Maschl. Quote: Mario Andretti, 1978 F1 world champion

Dual/Quad/Octo SPI: physical layer

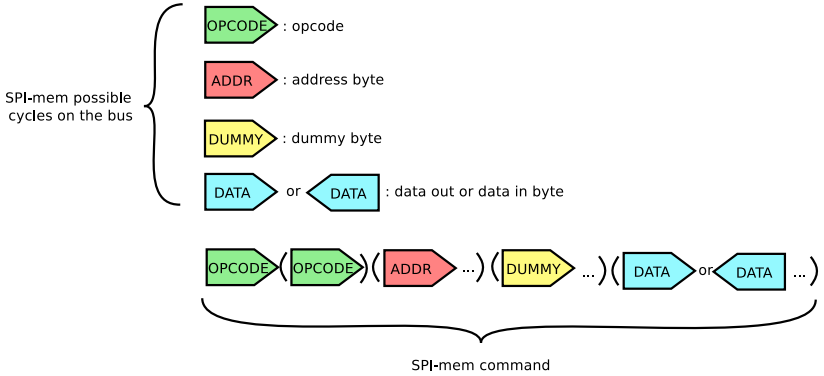
- Half-duplex
- I/O lines are bi-directional
- Number of I/O lines is device-specific
- The slave and master must agree on that (can be negotiated or hardcoded)



- Controllers might use the CS lines as I/O lines ⇒ only 1 device on the bus

SPI memories: a pseudo-standard protocol 1/2

- Standardizes how to communicate with a device
- Most of the time a memory device but not necessarily
- Every access is done through a SPI memory operation formed of:



SPI memories: a pseudo-standard protocol 2/2

- The opcode determines
  - The number of address and dummy bytes
  - The direction of the data transfer (if any)
  - The number of I/O lines used for each element
- Command set is device specific

SPI memories: standard command sets

- There are currently two distinct standard command sets
  - SPI NAND
  - SPI NOR
- Standardizes the following operations:
  - Read/Write accesses
  - Erase operations
  - Device identification
  - Accesses to internal registers
- Also standardizes some registers and their contents:
  - STATUS
  - CONFIGURATION
- Vendor specific operations/registers can be added on top

SPI memories: NOR vs. NAND command set

- Example: Read operation

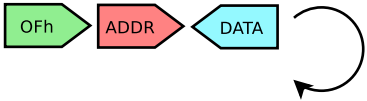
NOR command set

READ DATA BYTES

NAND command set

PAGE READ to cache

GET FEATURES to read the status

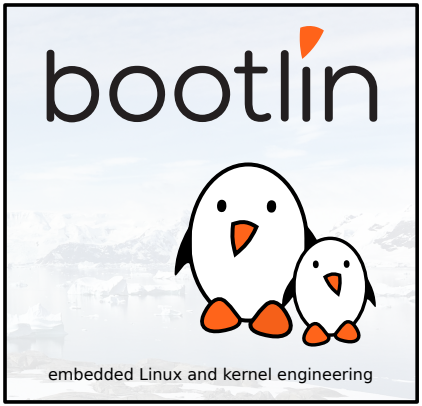


RANDOM DATA READ from cache

SPI memories support in Linux

Miquèl Raynal
miquel@bootlin.com

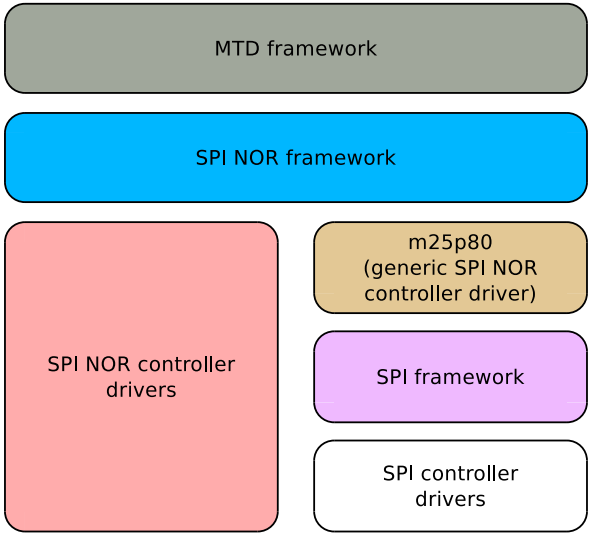
Boris Brezillon
boris@bootlin.com



SPI memories support in Linux: a bit of history

- Initially supported as simple SPI device drivers
- Most of the time placed in `drivers/mtd/devices/`
- Drivers were manually building SPI memory operations using `spi_message`s made of several `spi_transfer`s
- Apparition of SPI NORs and advanced SPI controllers forced us to reconsider this approach
  - Creation of a `spi-nor` subsystem to deal with the SPI NOR command set
  - Creation of a `spi_nor` interface to be implemented by advanced SPI controller drivers
  - Generic SPI NOR controller driver used to interface with generic SPI controllers (`drivers/mtd/devices/m25p80.c`)

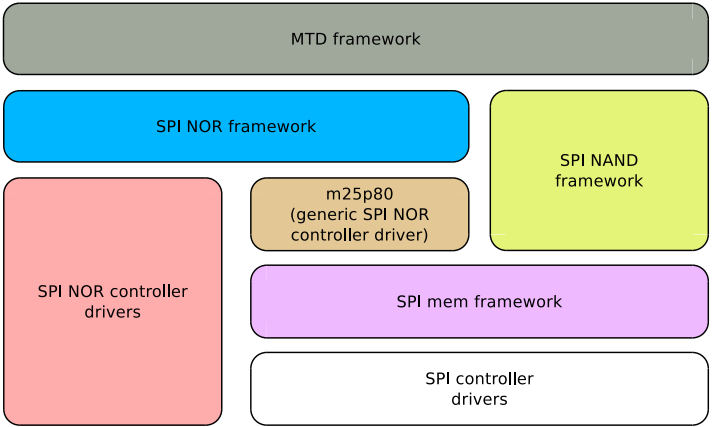
The SPI NOR stack



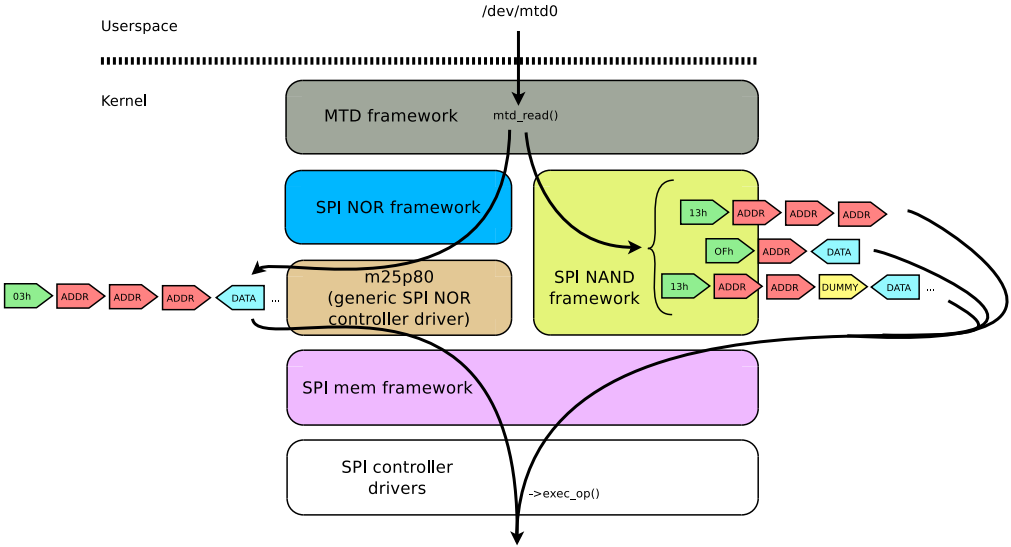
SPI memories support in Linux: recent changes

- The approach taken to support SPI NORs worked fine until people decided to support SPI NANDs
- Most SPI controllers are memory agnostic and can thus be interfaced with any kind of device (NOR, NAND, SRAM, and even regular SPI devices)
- Problems:
  - NOR and NAND command sets are totally different
  - NOR and NAND devices have different constraints and can't be handled the same way
  - We want to have the same SPI controller driver, no matter the device it's interfaced with
  - We don't want to create a custom interface per-memory type
- Solution:
  - Move the SPI memory protocol bits to the SPI subsystem
  - Let the SPI NOR and SPI NAND layers interface with this SPI memory layer

The SPI memory stack



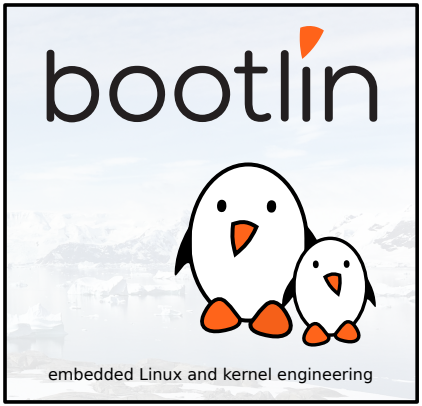
The SPI memory stack: read example



SPI memories support in U-Boot

Miquèl Raynal
miquel@bootlin.com

Boris Brezillon
boris@bootlin.com



U-Boot: Almost the same framework

- Port of the `spi-mem`/`spi-nand` framework
- Internal rework to use most of the MTD stack instead of the internal glue that has been added over the releases
- Cleaner partition handling not even in Linux yet!
- Merged in v2018.11-rc2

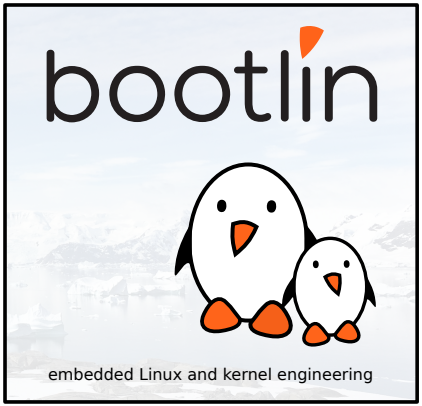
U-Boot: the `mtd` command

- Existing MTD devices commands: `sf`, `nand`, `onenand`
- But also: `mtdparts`
  - Shall we add a `spinand` one?
- MTD already abstracts the type of device for the user
- Creation of a generic command: `mtd`
  - Similar operations than before
  - U-Boot Driver-Model compliant
  - `help mtd`
  - The above commands should be deprecated (on the long run)
    - `mtdparts`/`mtdids` variables still useful!
    - Any `mtd` command will check for a change in these variables, in this case, MTD partitions will be updated

SPI memories: future development

Miquèl Raynal
miquel@bootlin.com

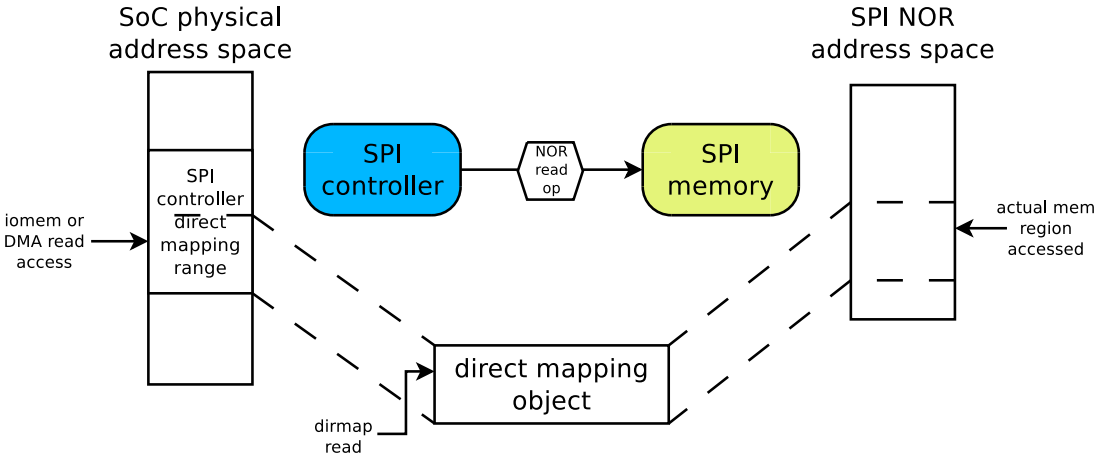
Boris Brezillon
boris@bootlin.com



SPI memories support in Linux: the next steps

- Support direct mapping
  - Supported by most advanced SPI controllers
  - Optimizes I/Os
  - An interface has been proposed here `http://lists.infradead.org/pipermail/linux-mtd/2018-June/081460.html`
- Convert all SPI NOR controller drivers to SPI controller drivers implementing the SPI memory interface
- Try not to reproduce our previous mistakes
  - Extend the SPI memory interface with extra care
  - Try to stay memory-agnostic
- Extra features
  - XIP?
  - Other optimizations?

SPI memories: a few words about the dirmap API



SPI memories: a few words about the dirmap API

- A direct mapping instance has 3 properties:
  - The memory device offset it's pointing it
  - The size of the mapping
  - A `spi_mem_op` template to execute when the dirmap is accessed
- Implementation is controller specific
- Four methods to implement:
  - `->create_dirmap()`: create a direct mapping
  - `->destroy_dirmap()`: destroy a direct mapping
  - `->dirmap_read()`: do a read access on the dirmap object
  - `->dirmap_write()`: do a read access on the dirmap object
- All methods are optional, when unimplemented the framework falls back to regular `->exec_op()` operations
- SPI mem users can create, destroy and do read/write accesses on dirmap using the `spi_mem_dirmap_{create,destroy,read,write}()` functions

SPI NOR: what's in the pipe?

- Add support for non-uniform erase sizes (Tudor Ambarus, merged in 4.20)
- Convert Atmel/Microchip and Freescale SPI NOR controller drivers to the SPI mem interface (Piotr Bugalski and Frieder Schrempf)
- Use the SPI mem direct mapping API to get better performance
- Finally move the m25p80 driver in `drivers/mtd/spi-nor/` and rename it *mtd: spi-nor: Move m25p80 code in spi-nor.c*

SPI NAND: what's in the pipe?

- Implement generic support for on-flash bad block table parsing/update
- Parse the ONFI parameter table when available?
- Define a generic ECC engine interface so that SPI NANDs without on-die ECC can be used with SoCs providing such an ECC engine (or with the software ECC implementation)
- Use the SPI mem direct mapping API to get better performance
- Add support for more chips
  mtd: spinand: winbond: Add support for W25N01GV
- The SPI NAND staging driver is going to be removed in the next release
  staging: Remove the mt29f_spinand driver

Questions? Suggestions? Comments?

Miquèl Raynal and Boris Brezillon

miquel@bootlin.com and boris@bootlin.com

https://bootlin.com/pub/conferences/2018/elce/raynal-spi-memories