From raw to refined: The evolution of raw flash support in Linux

Miquèl Raynal
miquel.raynal@bootlin.com

Embedded Linux Conference Europe 2025

Miquèl Raynal

- Embedded Linux engineer at **Bootlin**
  - Development, consulting and training about embedded Linux
  - Strong open-source focus
- Linux kernel contributor and maintainer
  - Maintainer of the **NAND subsystem**
  - Co-maintainer of the **MTD subsystem**
  - Co-maintainer of the IEEE 802.15.4 subsystem
- User and sporadic contributor of U-Boot, Zephyr, Buildroot...
- Living in Toulouse, France

Evolution of the flash market

- Parallel NORs
  - Block erasable
  - Random accesses, can be used for code execution
  - Rapidly cheaper than original EEPROMs
  - `drivers/mtd/{chips,devices,maps}/`?
  - -ETOOYOUNG
- Parallel NANDs
  - Cheaper as density is higher
  - Page reads/writes
  - Less stable (need error correction)
  - ONFI and JEDEC specification
  - `drivers/mtd/nand/raw/`
- SPI NORs
  - Easier to route
  - Known bus, controllers already supported
  - JEDEC specification
  - `drivers/spi/spi-mem.c` & `drivers/mtd/spi-nor/`
- SPI NANDs
  - "Best of all worlds"?
  - Yet, no actual industry standard?
  - `drivers/spi/spi-mem.c` & `drivers/mtd/nand/spi/`

Storage improvements

- One Time Programmable areas
- (Problems with) high densities

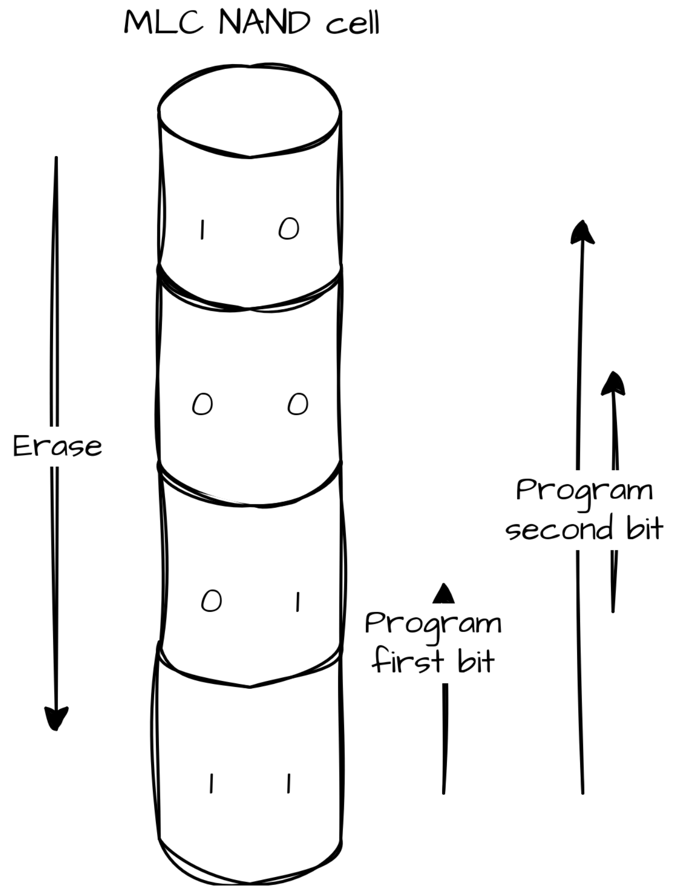
OTP support

- Factory OTP is not writeable
  - Typically stores unique factory identifiers
- User OTP are writeable and can be locked (more than one time programmable...)
  - MAC addresses?
  - Device serial numbers?
- MTD features various OTP callbacks:
  (`get_infos`, `erase`, `write`, `read` and `lock`)
  - SPI NOR has support for user OTP areas since v5.13 (no factory OTP support)
  - Macronix raw NAND chip driver has received OTP support in v6.5
  - SPI NAND has gotten support for user and factory (read-only) OTP in v6.15

NAND: SLC, MLC, TLC

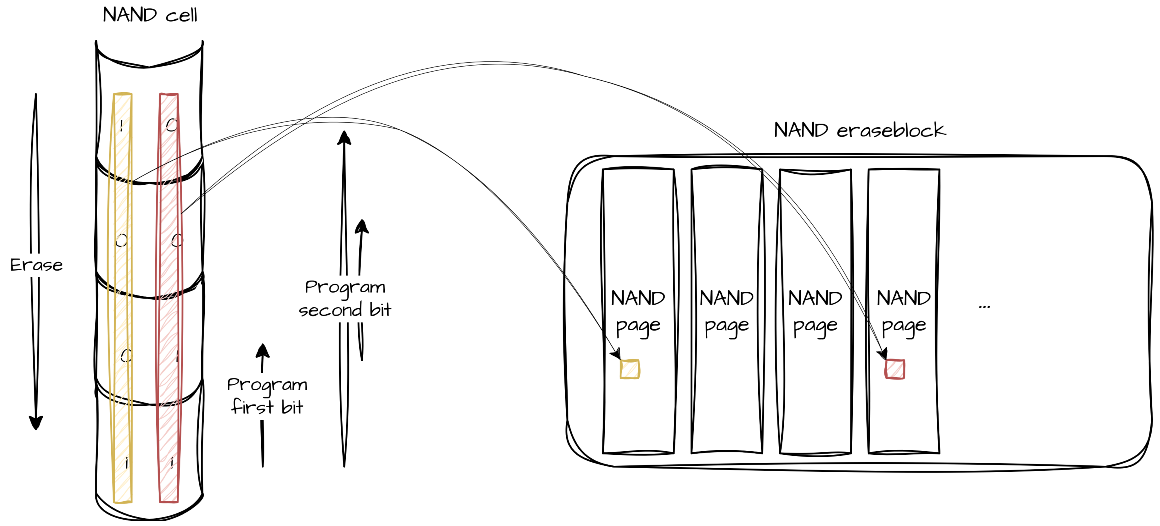
A NAND cell may store:

- 1 bit of data (2 states): Single Level Cells (SLC)
  - Supported since the beginning
- 2 bits of data (4 states): Multi-Level Cells (MLC)
  - Partially supported! (see next slide)
- 3+ bits per cell (TLC, QLC) and other technologies
  - No support, typically only available through a vendor FTL



NAND: MLC support issues

- Deep Knowledge of the chip is required
  - Each cell stores one bit from 2 non contiguous pages, pairing scheme must be known
  - Very sensitive to power cuts or write abortions
  - High risk of disturbances, requires stronger corrections
  - Overall, an order of magnitude less stable than SLC
- Reported as “a nightmare” by the former NAND maintainer

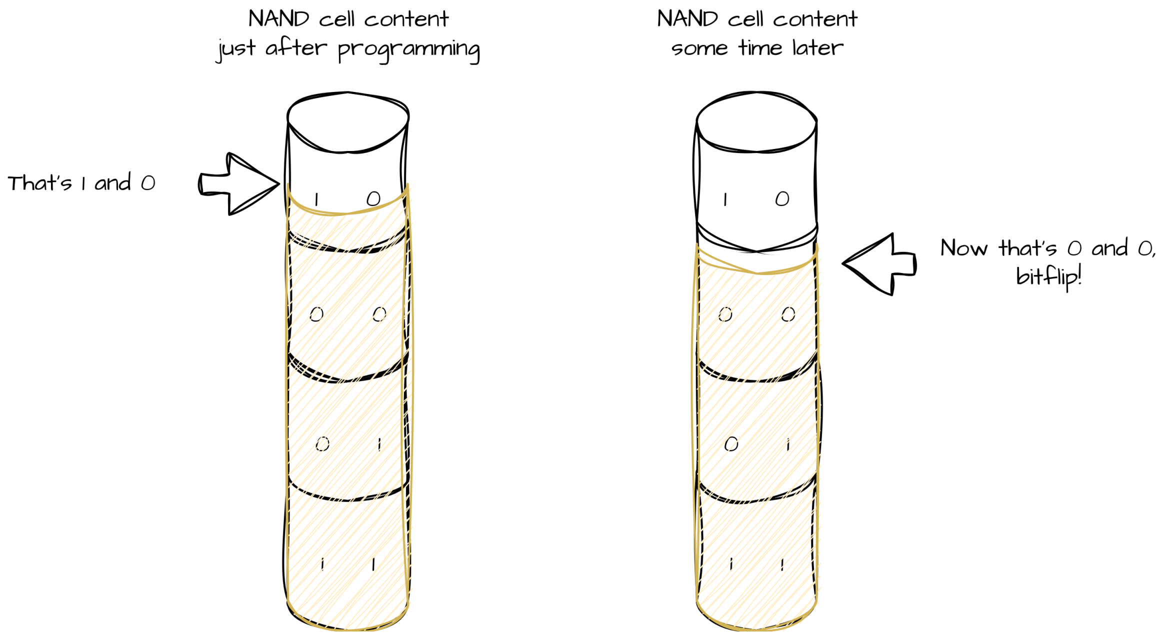


NAND: MLC support in Linux

- Adopted solution was to enable a pseudo SLC mode
  - Halves the capacity of the chip
  - Gets rid of the biggest stability issues
    - `slc-mode` DT property
    - Pairing scheme must be known by Linux and `ubinize`
    - Run `ubihealthd`
- Has been an out of tree patch set for quite some time, eventually merged in v5.8

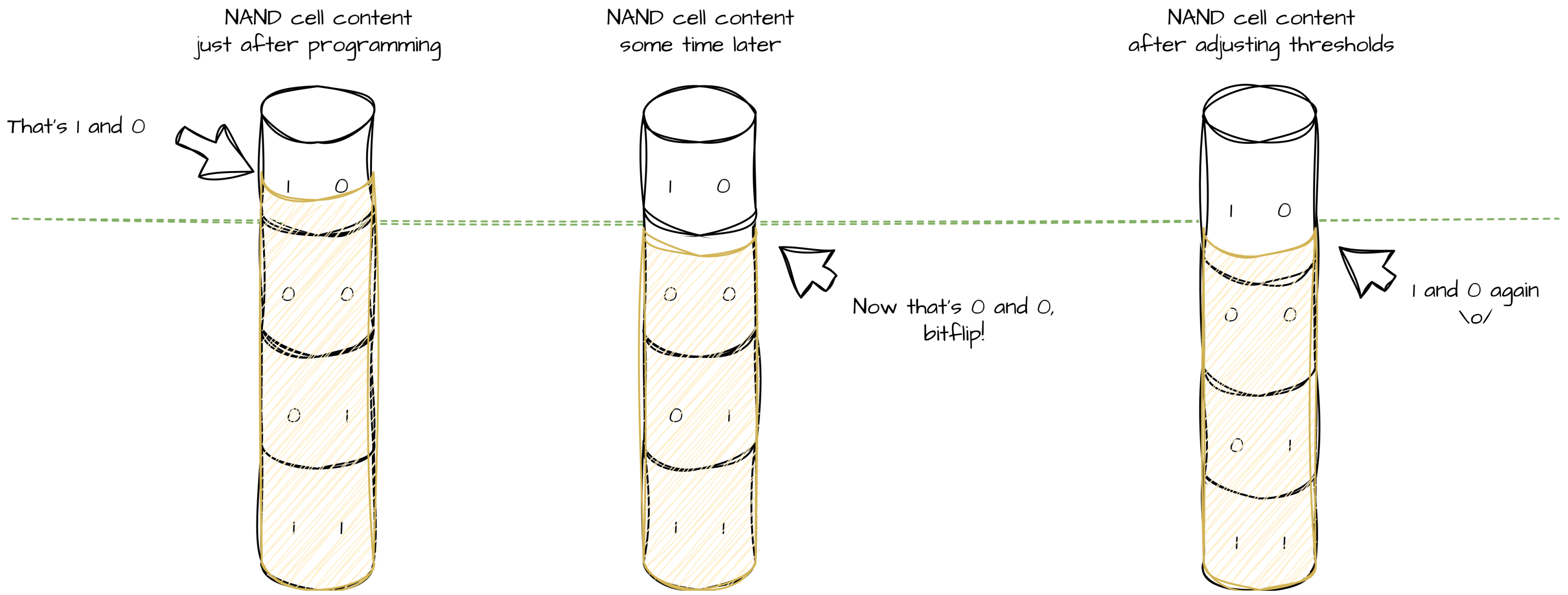
NAND: Retention issues?

- Reading a NAND page typically returns a small amount of bitflips (or none, ideally)
- Data retention issues may sometimes lead to more charge loss than ECC can correct
  - Typical with MLC NANDs
  - Also possible with SLC NANDs
    - “Bad” MLC batches may be rebranded as SLC as well


NAND cell content just after programming
NAND cell content some time later
That's 1 and 0
1 0
0 0
0 1
1 1
1 0
0 0
0 1
1 1
Now that's 0 and 0, bitflip!

NAND: Read retry!

- Need for a retry mechanism, where the sensing circuitry adapts its thresholds



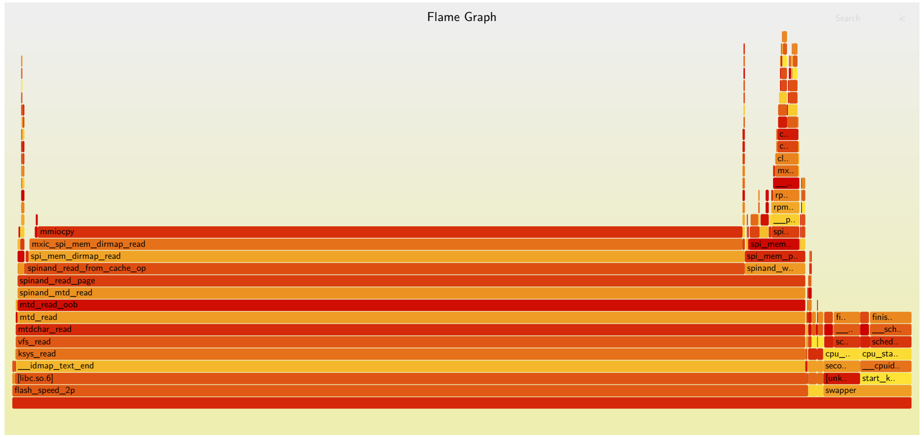
- Read retry support for raw NANDs since v3.14
- Available in SPI NAND since v6.15! (Macronix chips)

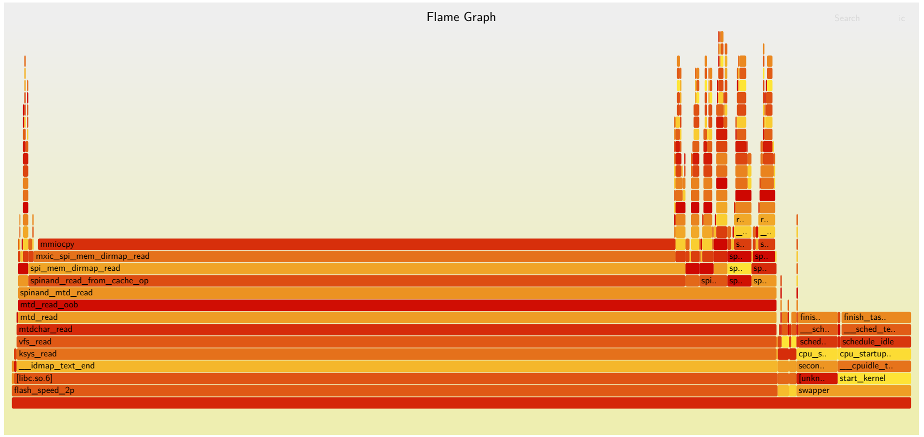
Speed improvements

- Host side optimizations
- Flash side optimizations
- Bus optimizations

MTD speed benchmarks and pitfalls

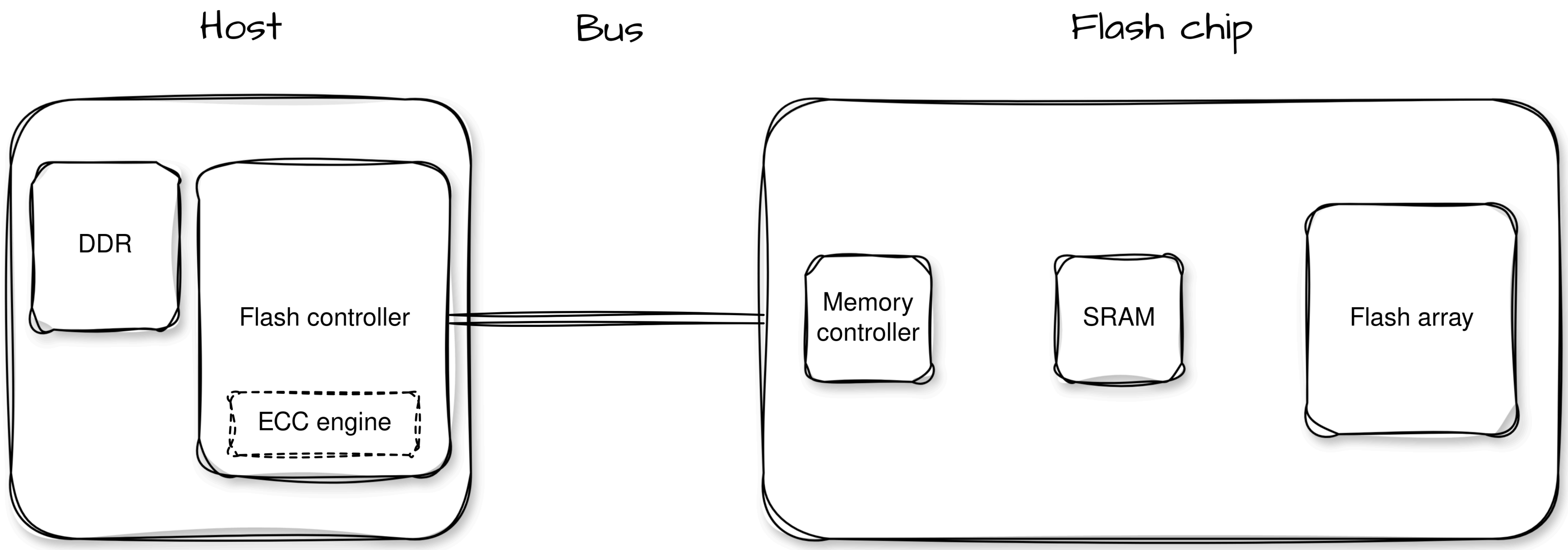
- `flash_speed -c10 /dev/mtdX`
  - Reads the clock, does I/O, reads the clock again
  - Sensitive to system load
    - System must be idle
    - Task priority must be tuned for better repeatability (`chrt -f 50 <benchmark>`)
    - CPU power management must be disabled for allowing meaningful comparisons (no `cpufreq`/`cpuidle`)



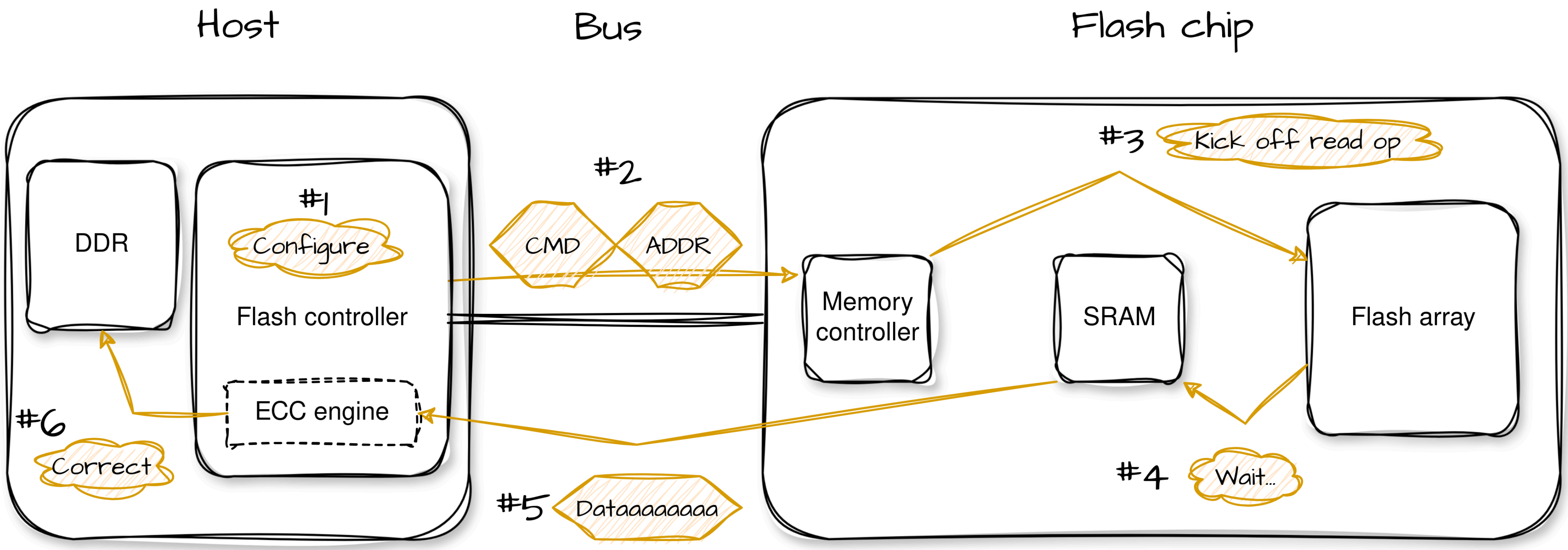


- Amount of data transferred matters a lot (values coming next are based on page reads, unless stated differently)

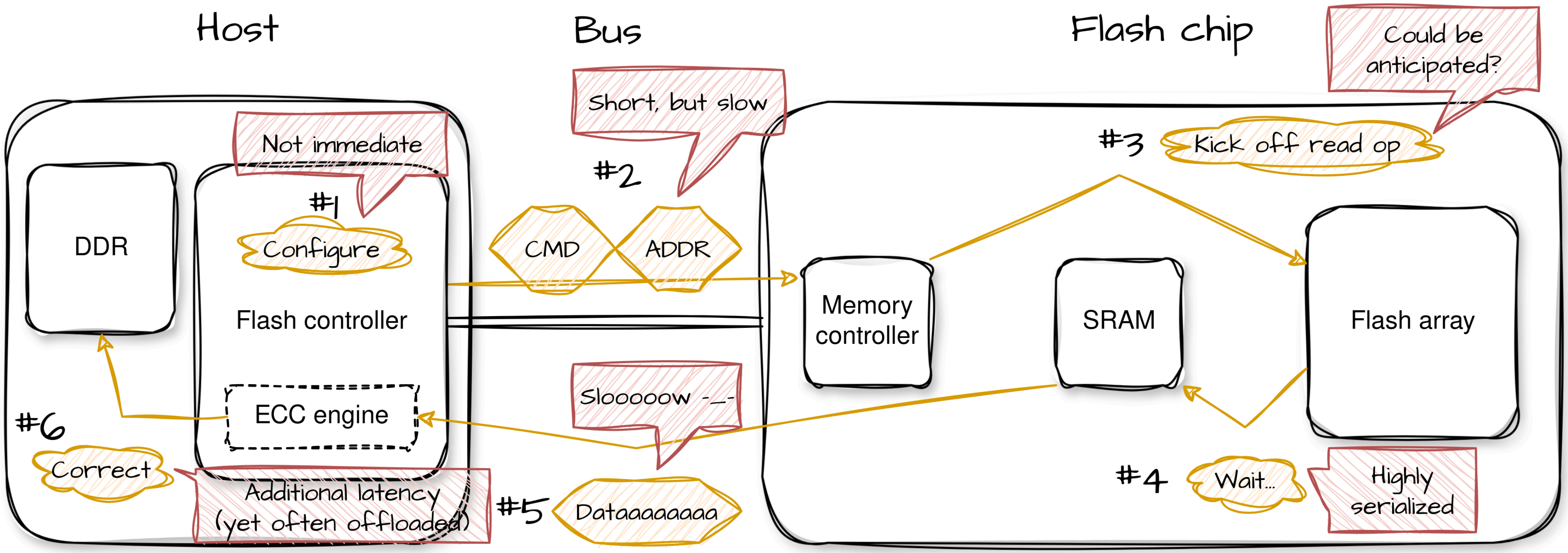
Hardware connections



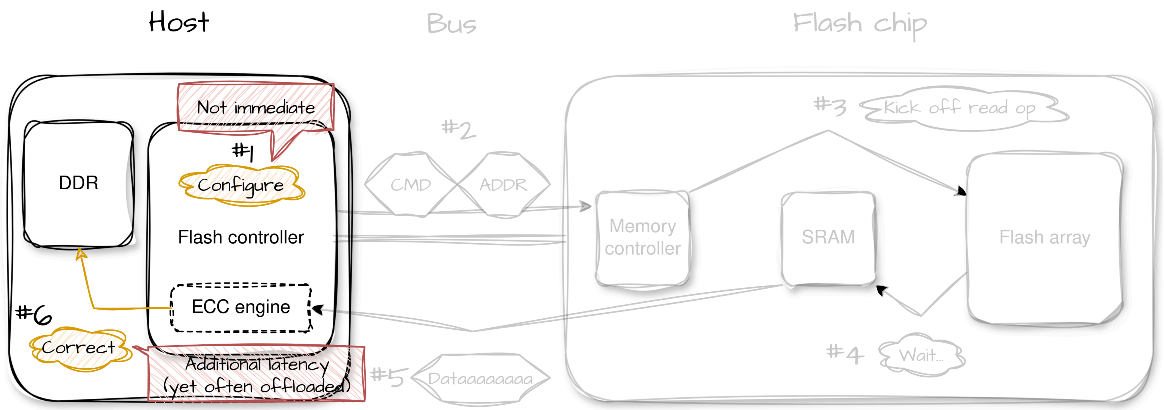
Reading from flash



Bottlenecks behind a flash read



Host side optimizations



SPI memories: Direct mappings

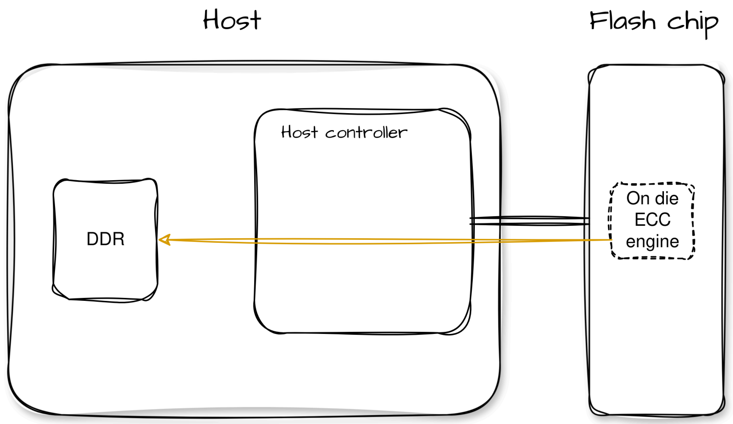
- Modern SPI controllers can map portions of SPI memories in the CPU address space
- May bring performance improvements
- Offloading to hardware the creation and sending of the SPI messages
- All SPI memory operations use `dirmaps`
  - There is actually a fallback on the classical `spi_mem_exec_op()` helper if `dirmaps` are not supported
- Available since Linux v5.0

Parallel NANDs: ECC engines

- Raw NAND controllers commonly feature pipelined ECC engines
  - By far the default ECC engine pick
- Support for Hamming and BCH software ECC engines is also available
  - DT properties: `nand-use-soft-ecc-engine`, `nand-ecc-algo`, `nand-ecc-strength`/`nand-ecc-step-size`
  - Fundamentally slower, but may be helpful in some situations:
    - Workaround hardware limitations (see Arasan NAND controller driver page read implementation)
    - Enable the use of chips with stronger ECC needs than the controllers are capable of
    - Useful in extreme environments (high temperature or space)

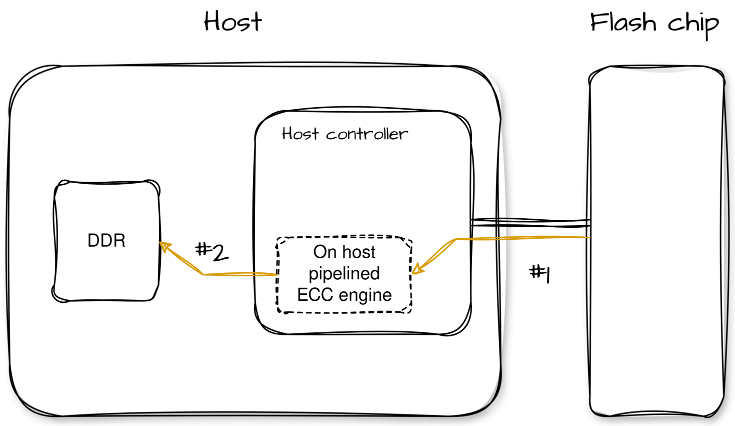
SPI NANDs: Mostly on die ECC engines

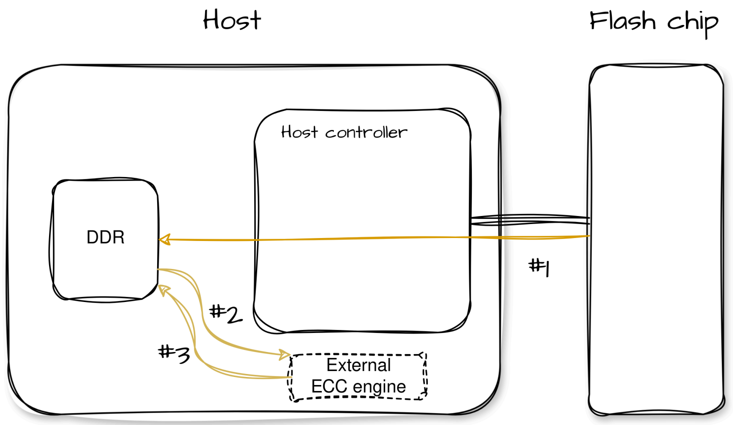
- SPI controllers historically do not feature any correction capability
- Most SPI NAND chips feature an on-die ECC engine
  - A bit slow
  - Barely configurable
  - Makes the chip more expensive
  - Wastes silicon on the flash chip



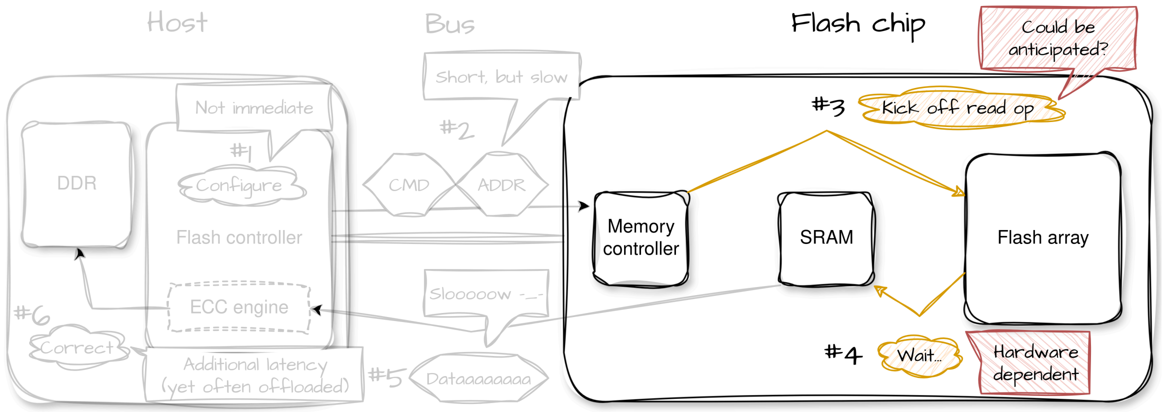
SPI NANDs: Offloading external ECC engines

- On host hardware ECC engines are now supported!
  - Sometimes the same engine is shared between the parallel NAND controller and the SPI controller
  - Can either be pipelined (faster) or external
- Independent ECC engine drivers have been contributed! (`drivers/mtd/nand/ecc-*.c`)
  - Simple API: `->init_ctx()`, `->cleanup_ctx()`, `->prepare_io_req()`, `->finish_io_req()`
  - Macronix in v5.18, then Mediatek and Qualcomm, soon Realtek!

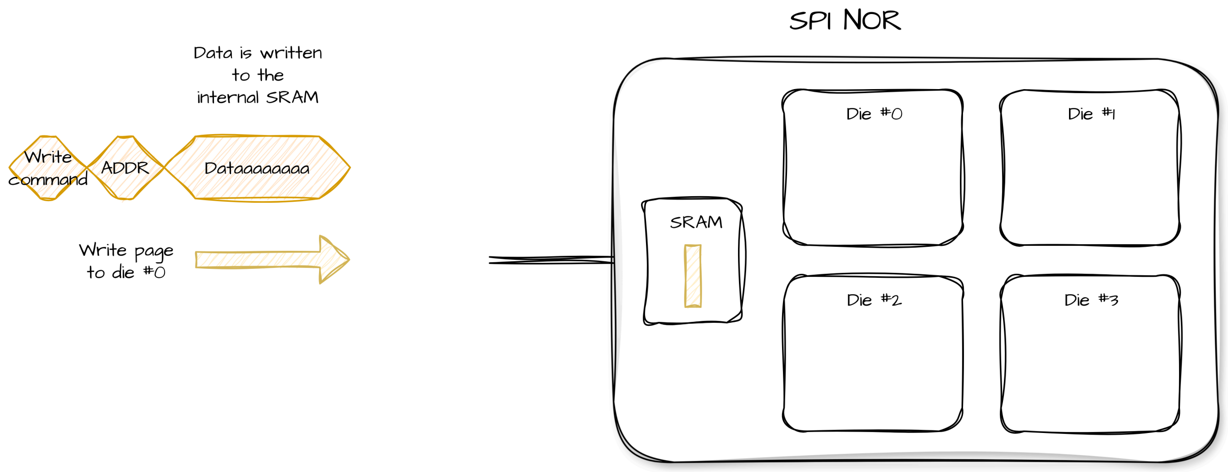




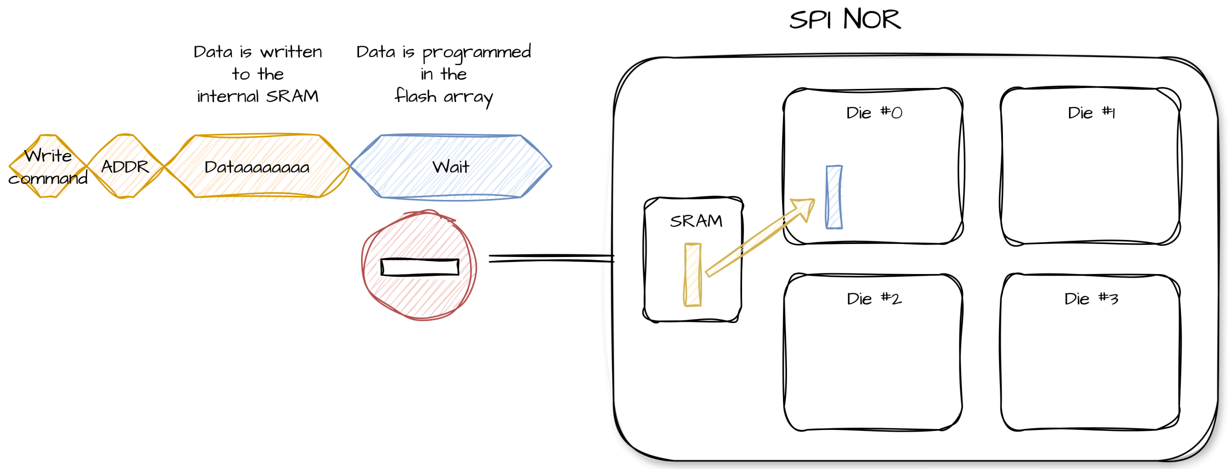
Flash side optimizations



SPI NOR: Full serialization is slow



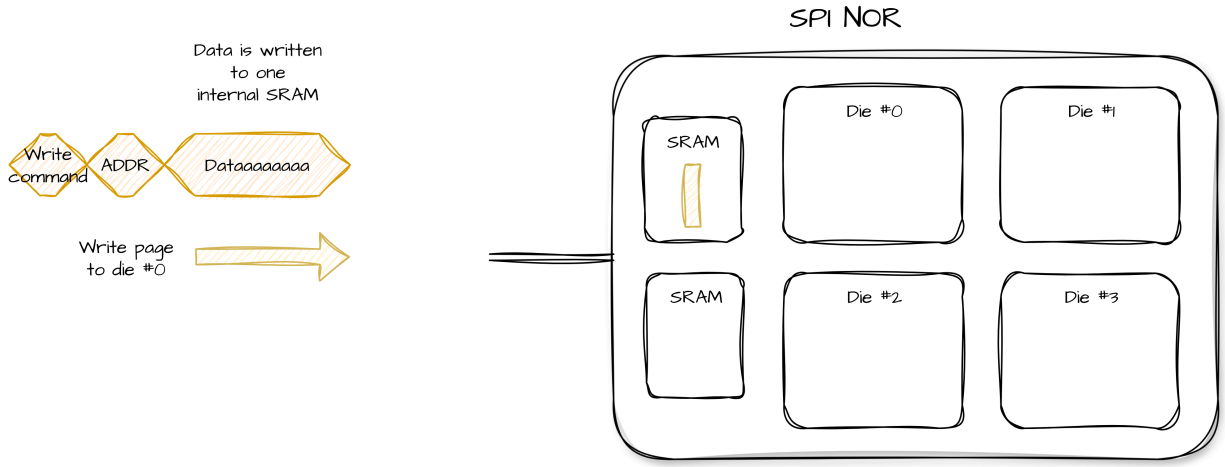
- Lack of responsiveness as writes are slow
  - page write speed:
    - 247 KiB/s
  - page read speed:
    - 902 KiB/s
- All further operations blocked until the flash is ready again

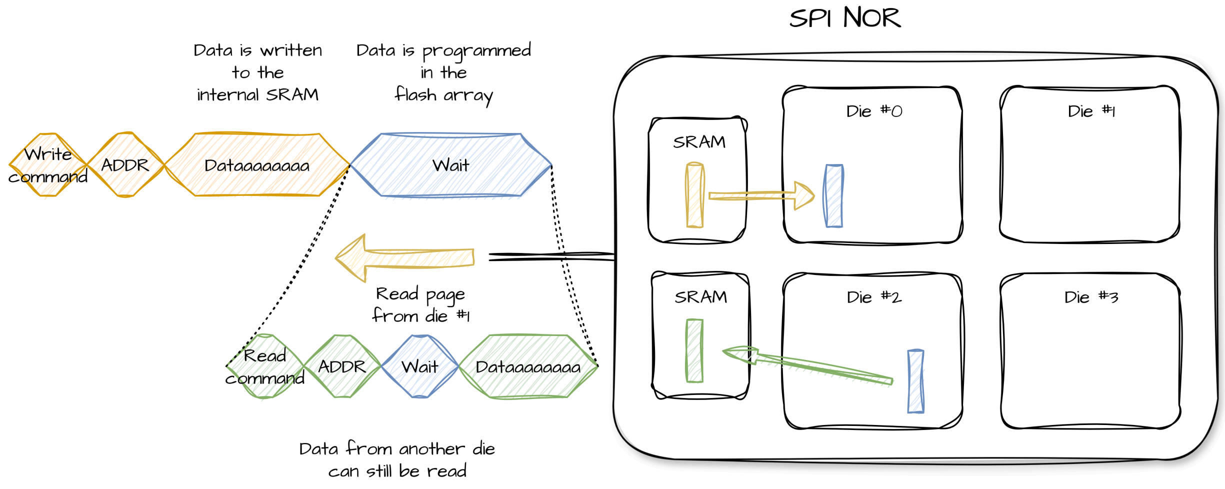


SPI NOR
Data is written to the internal SRAM
Data is programmed in the flash array
Write command
ADDR
Dataaaaaaaa
Wait
SRAM
Die #0
Die #1
Die #2
Die #3

SPI NOR: Read while write (RWW)

- Read While Write feature introduced in v6.4
- Supported on Macronix chips
  - May also be useful during updates (downloading while keeping the rootfs available)





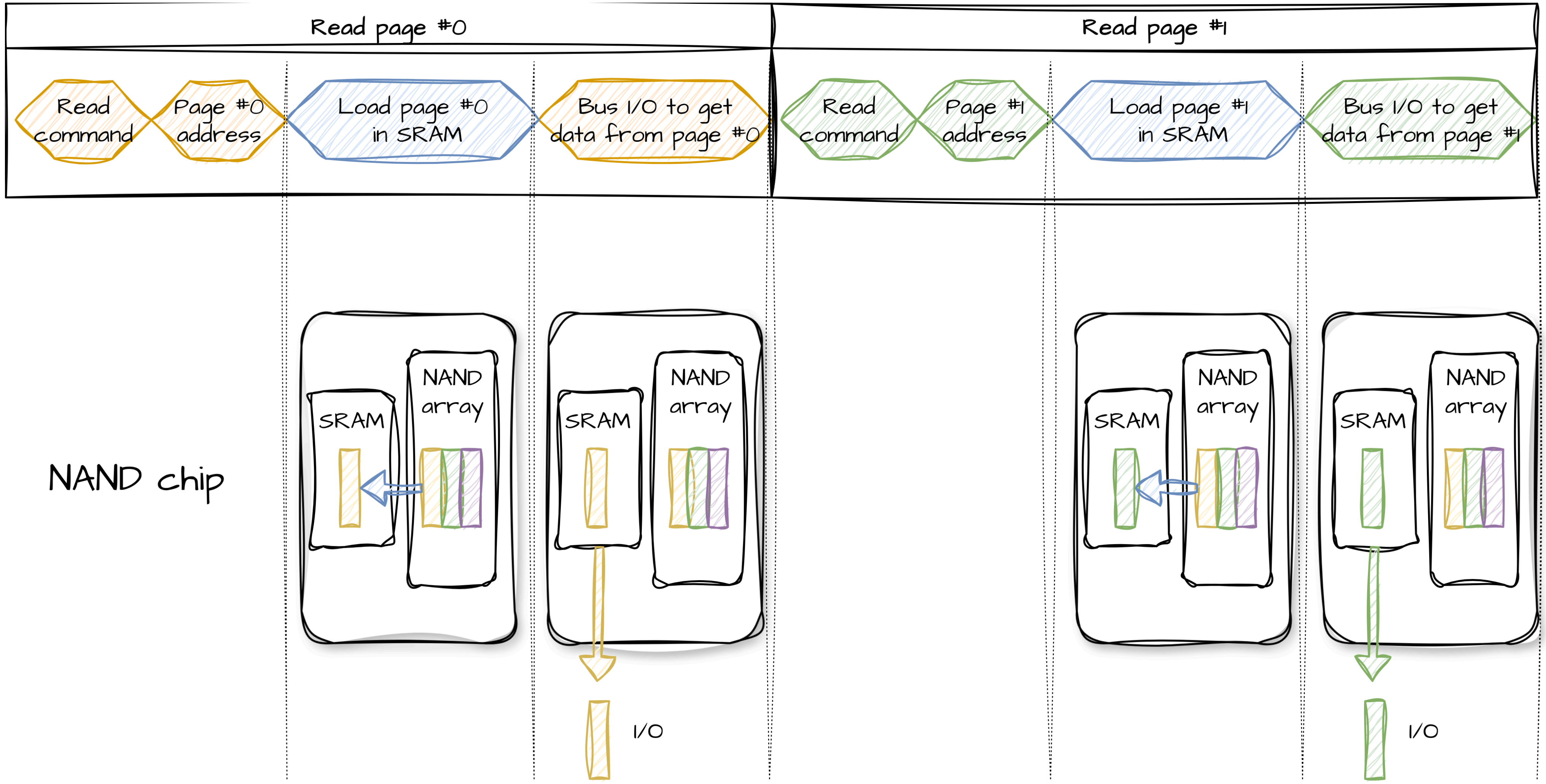
SPI NOR: Read while write benchmark

- Additional option added to `flash_speed`:
  - `-k, --sec-peb <num>`
    `Start of secondary block to measure RWW latency (requires -d)`

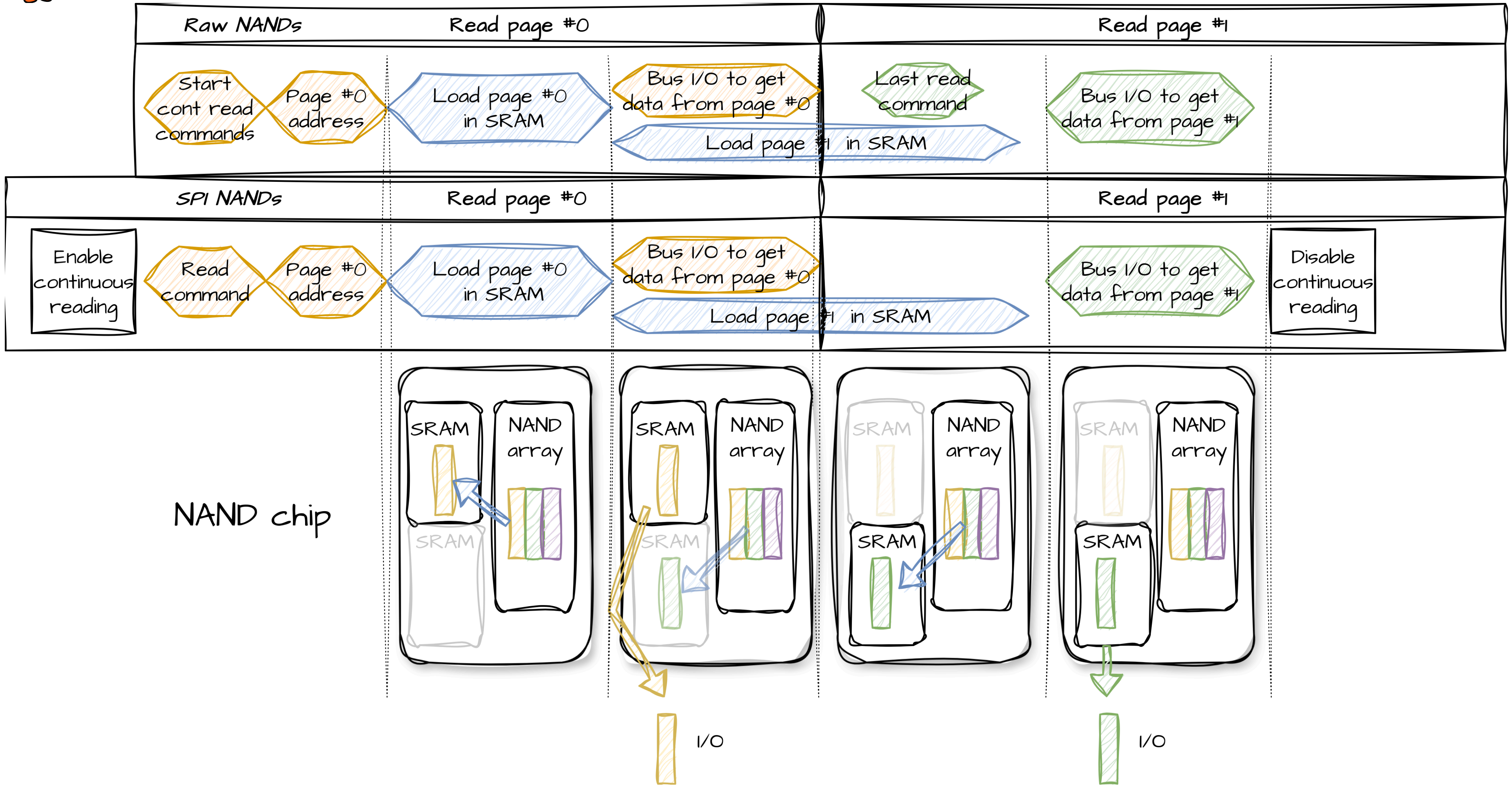
```
$ flash_speed -b0 -k0 -c1 -d /dev/mtd0 # Same die
[...]
testing read while write latency
read while write took 7ms, read ended after 7ms

$ flash_speed -b0 -k4096 -c1 -d /dev/mtd0 # Different dies
[...]
testing read while write latency
read while write took 7ms, read ended after 5ms
```

NAND: Sequential accesses?



NAND: Sequential cached reads/continuous reads!



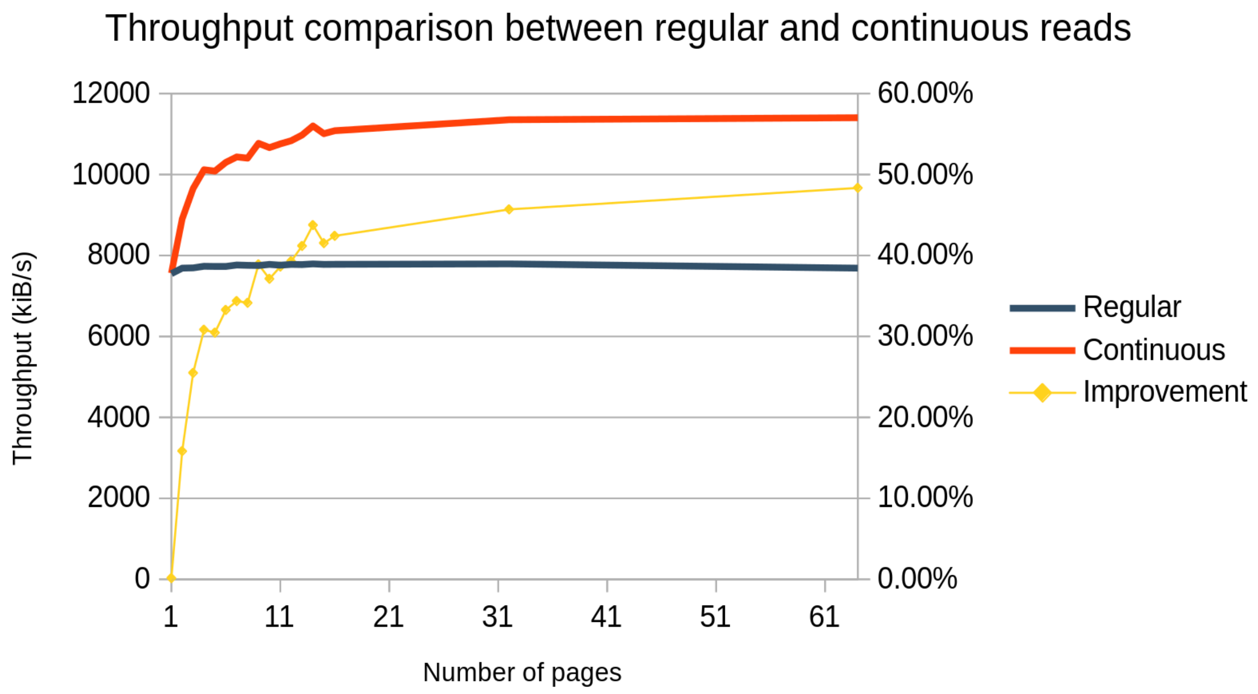
Raw NAND: Sequential cached reads/continuous reads: results

- Raw NAND support for sequential cached reads: v6.3
  - Received several fixes until v6.8
  - Read speed comparison on an i.MX6 with a Macronix MX30LF2G (2kiB pages) NAND:

kiB/s	Regular read	Continuous read
1 page	15515	15875
2 pages	15398	16253
64 pages	15633	18285

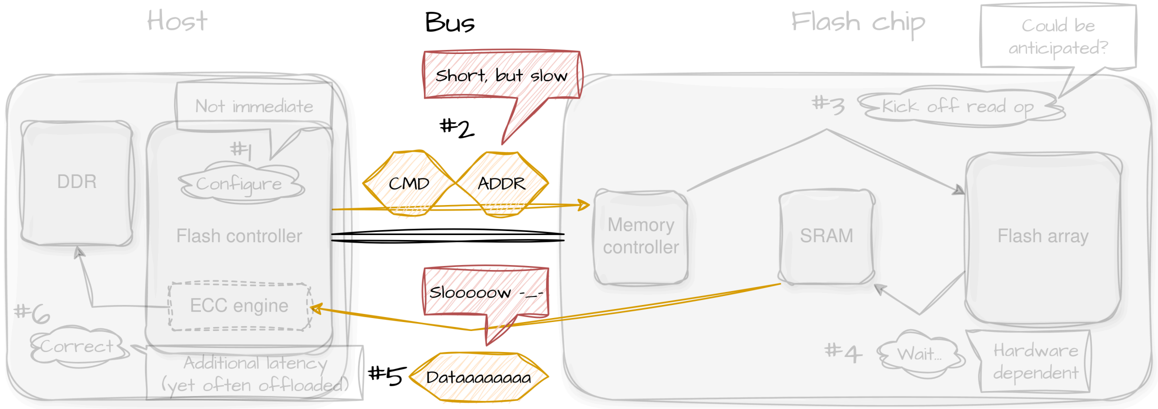
SPI NAND: Sequential cached reads/continuous reads: results

- Speed benchmark on an i.MX6 with a Macronix MX35LF2G (2kiB pages) NAND



- Feature merged in v6.12

Bus optimizations



Parallel NANDs: Data interface (past)

Support for the timing modes has been there forever:

- “speed” mode ranging from 0 (all chips start in mode 0) up to mode 5
- Chips advertise the supported modes in their parameter page
- Controllers must be configured to comply with the picked mode and acknowledge the switch
- Chips normally support changing timing mode with `GET_FEATURE` and `SET_FEATURE` commands
    - Some chips do not support these commands
    - Some chips support the commands but do not advertise them in their parameter page
    - Some chips support `SET_FEATURE` but not `GET_FEATURE`, leaving us in the dark regarding the actual timing mode on the chip side, so we currently assume the switch was successful

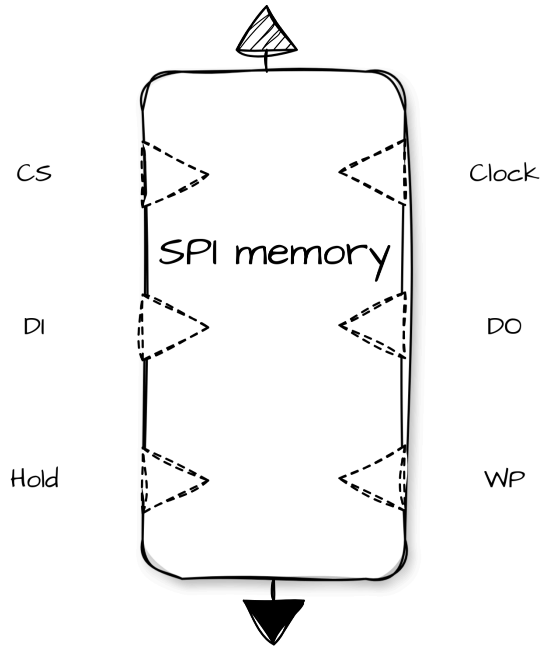
Parallel NANDs: Data interface (present)

- Management of the whole data interface instead of just the timing mode
- New callback called `->choose_interface_config()` (Linux v5.10)
    - Allows to pick the best commonly supported configuration between SDR and NV-DDR timing modes
        - NV-DDR modes also range from 0 to 5
        - Synchronous modes (as opposed to SDR)
- NV-DDR timings introduced in Linux v5.14, with support for the Arasan NAND controller
    - Speed benchmark reading one 16kiB NAND page:

kiB/s	SDR mode 5	NV-DDR mode 5
Read	8094	16062
Write	7013	24824

SPI memories: Single SPI just works

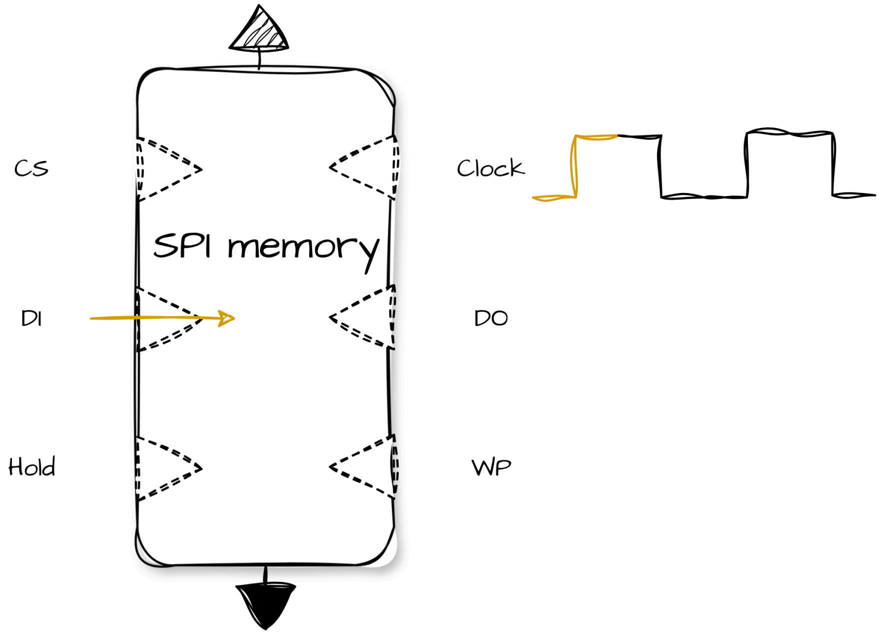
- Single SPI is slow! don't tell Mark
  - In Linux we represent regular single SPI transaction with: 1-1-1
    - # of data lines for the command opcode
    - # of data lines for the address byte(s)
    - # of data lines for the data cycles

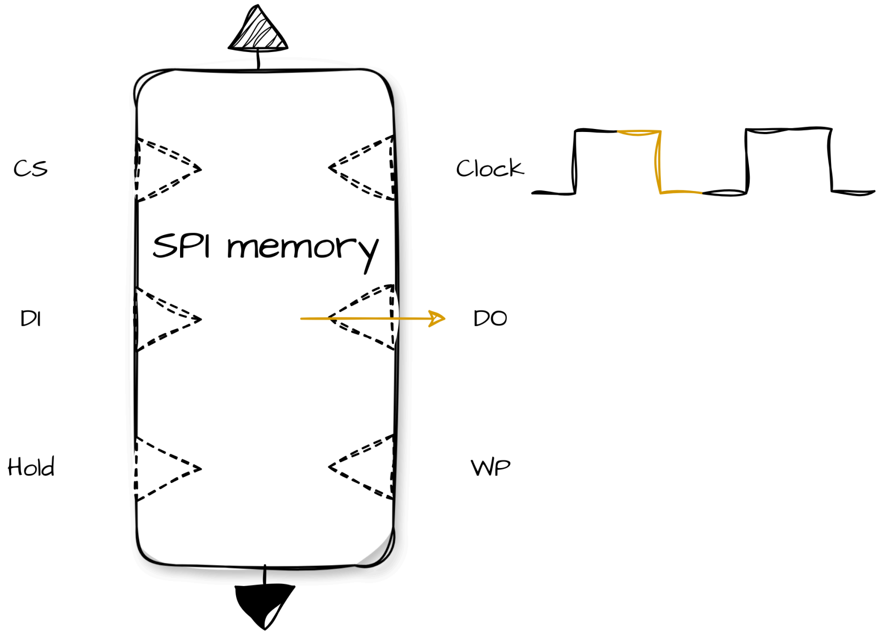


,

SPI memories: Parallelism again?

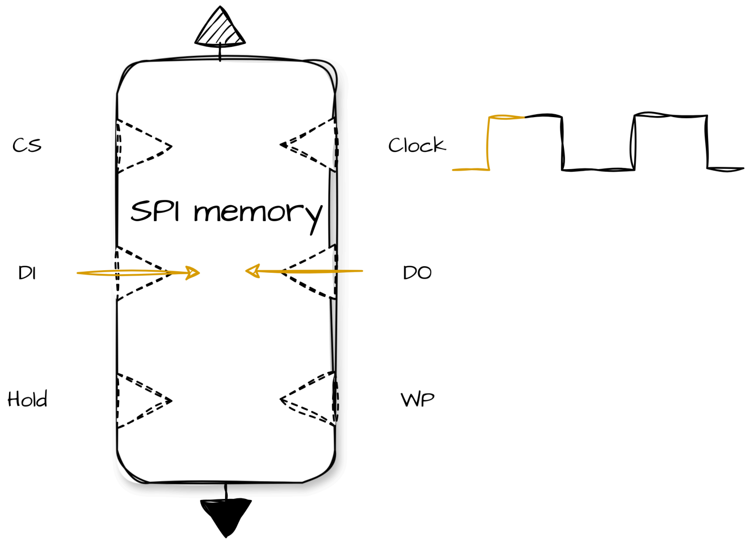
- Memories typically
  - read data from the DI pin on the clock rising edge
  - write data (or status) on the DO pin on the clock falling edge

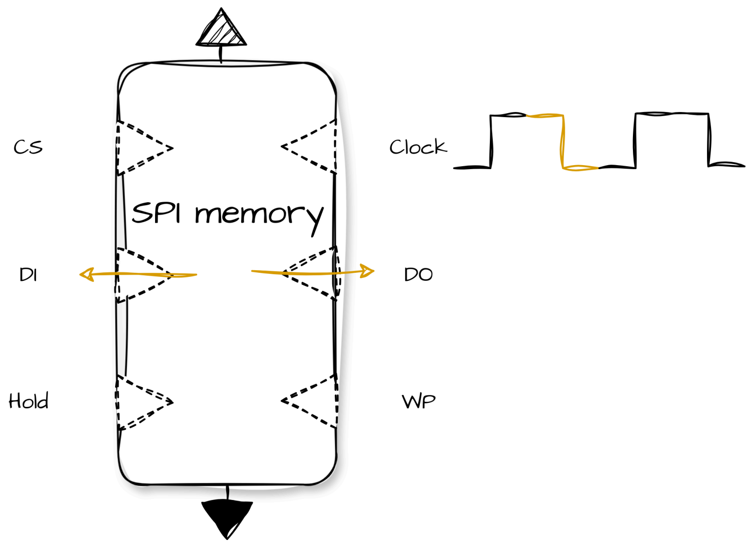




SPI memories: Dual SPI is better!

- No hardware implications, just re-use a temporarily unused pin!
  - 1-1-2, 1-2-2 (Command opcode is always transferred in single SPI mode)

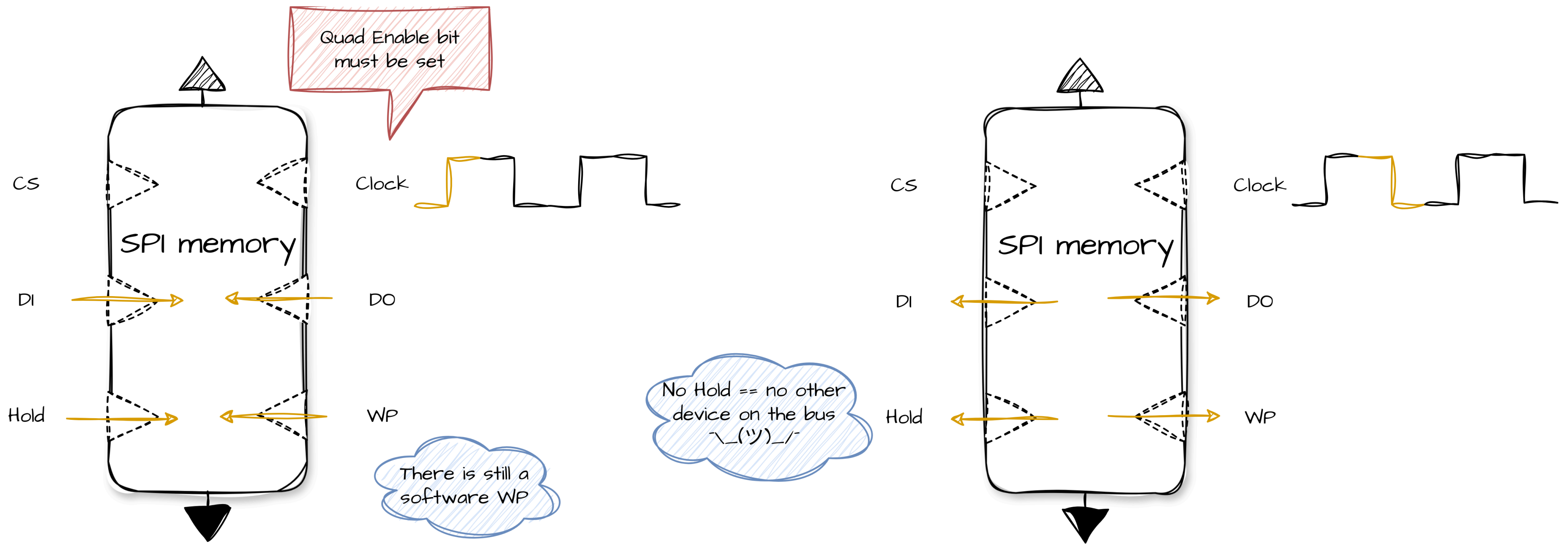




- Has been supported forever in SPI NOR, like ~v3.12
  - Way before the introduction of `spi-mem` in v4.18
- Support in SPI NAND is more recent, this was present at subsystem creation (v4.19)

SPI memories: Quad is faster!

- "What are WP and HOLD for again? What about moar speed instead?"
  - 1-1-4, 1-4-4


Quad Enable bit must be set
CS
Clock
SPI memory
DI
DO
Hold
WP
There is still a software WP
No Hold == no other device on the bus ¯_(ツ)_/¯
CS
Clock
SPI memory
DI
DO
Hold
WP

- Dual and Quad support were introduced at the same time as they are very similar

SPI memories: Single/Dual/Quad comparison

Speed comparison on Nuvoton MA35D1 dev kit with a Winbond W25N02JW NAND:

kiB/s	1-1-1	1-2-2	1-4-4
Read	1097	1229	1433
Write	2012	-	2253

SPI memories: But what about Octal?

- This time our hardware colleagues have more work
  - 8 data lines
  - usually a RESET pin to make sure we start on a common ground
- Introduced in v5.1 in SPI NOR
- Supported in v6.17 in SPI NAND (Winbond only so far)!

Speed comparison on TI AM62A LP starter kit with a Winbond W35N01JW NAND:

kiB/s	1-1-1	1-1-8	1-8-8
Read	2342	10711	10711
Write	2028	-	7293

Lack of difference in read speed between the two octal modes comes from the fact that the 7 saved clock cycles when sending the address only represent an extra delay of 0.24us at 30MHz, once per page.

SPI memories: Double Transfer Rates

- Use both sides of the clock
- Many SPI controllers have DTR support
  - A new soup of possible variants (S: Single Data Rate, D: Double Transfer Rate)
    - 1S-1D-1D, 1S-1S-1D
    - 1S-1S-2D, 1S-1D-2D, 1S-2S-2D, 1S-2D-2D
    - 1S-1S-4D, 1S-1D-4D, 1S-4S-4D, 1S-4D-4D
- SPI NOR has support for DTR since v5.11
  - Mixed modes (S-S-D or S-D-D) are not supported
- SPI NAND support introduced in v6.14 (Winbond quad and octal chips)
  - DTR stateful modes (D-D-D) are not supported
- Read comparison on the Nuvoton platform with a quad capable NAND chip:

kiB/s	Single	Dual	Quad
SDR (1S-XS-XS)	1097	1229	1433
DTR (1S-XD-XD)	1568	1681	1810

SPI memories: What about Octal DTR (ODDR) modes?

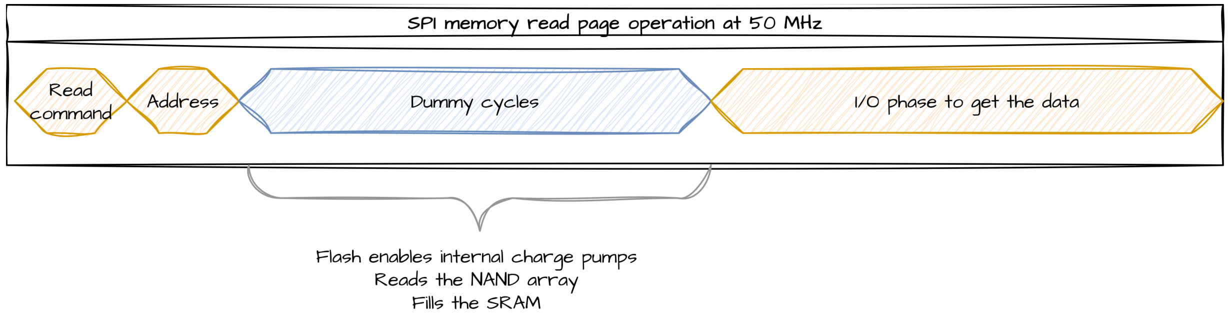
- SPI NOR supports it, part of the package since v5.11
  - Stateful support refused on memories without a proper reset pin
  - SPI NOR benchmark on Xilinx Zynq ZC702 @ 16.6MHz with a Macronix MX25UW12845G:

kiB/s	1S-1S-1S	8D-8D-8D
Read	1040	1287
Write	547	664

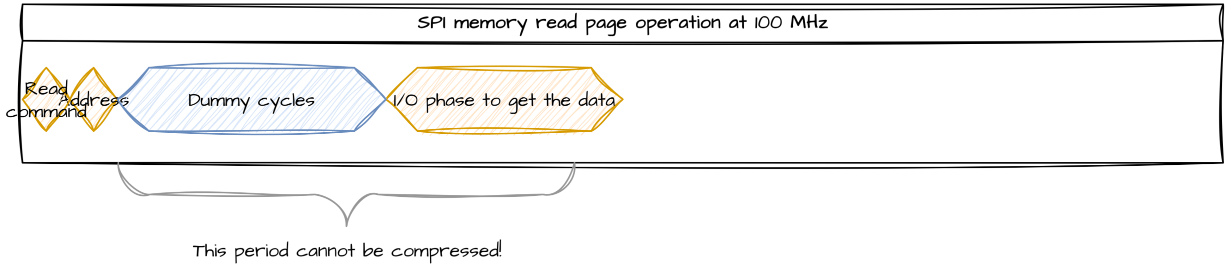
- The SPI NAND core is not ready for stateful modes
  - Important rework required
  - In the pipe, RFC hopefully for end of 2025

SPI memories: Increasing the bus speed

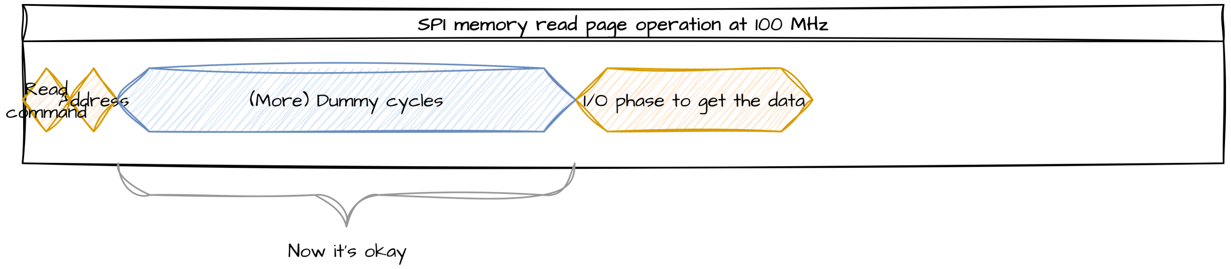
- Increasing the bus speed has limitations
- Flashes require a minimum amount of time to internally process the host requests
  - No extra Read/Busy bin like in the parallel world
  - Dummy cycles to the rescue



SPI memory read page operation at 50 MHz
Read command
Address
Dummy cycles
I/O phase to get the data
Flash enables internal charge pumps
Reads the NAND array
Fills the SRAM



SPI memory read page operation at 100 MHz
Read command
Address
Dummy cycles
I/O phase to get the data
This period cannot be compressed!



SPI memory read page operation at 100 MHz
Read command
Address
(More) Dummy cycles
I/O phase to get the data
Now it's okay

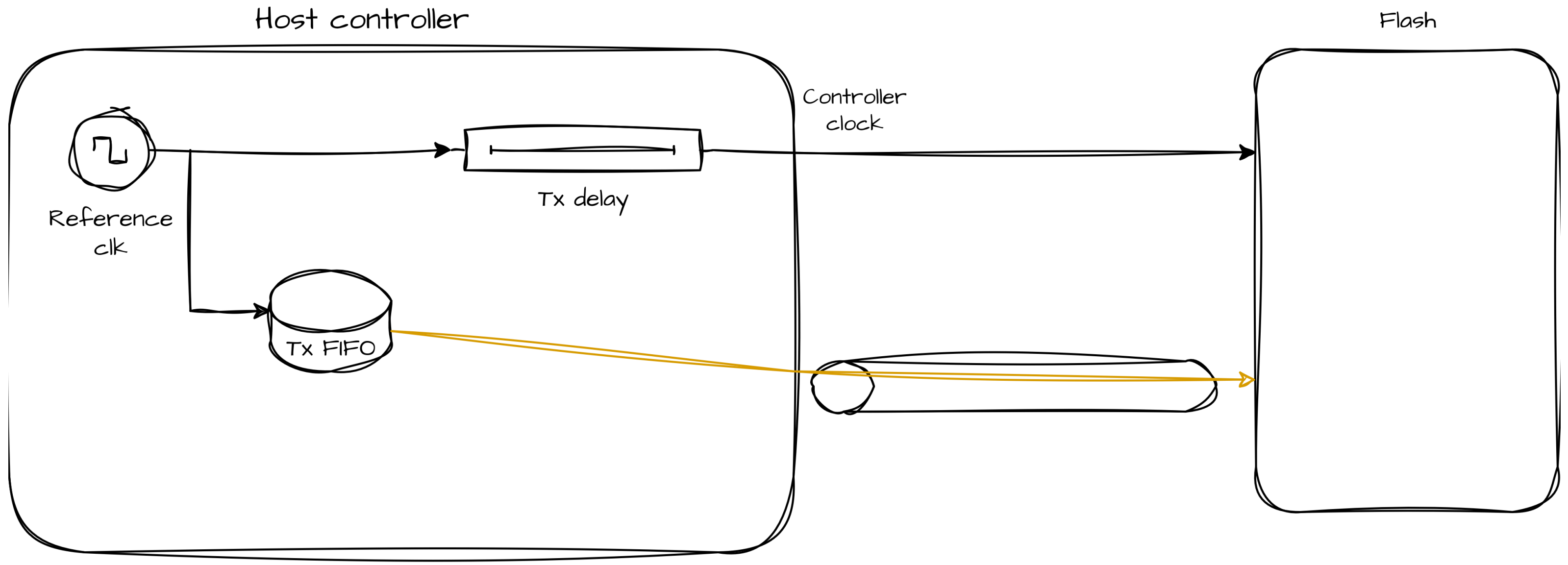
SPI memories: Flexible dummy cycles

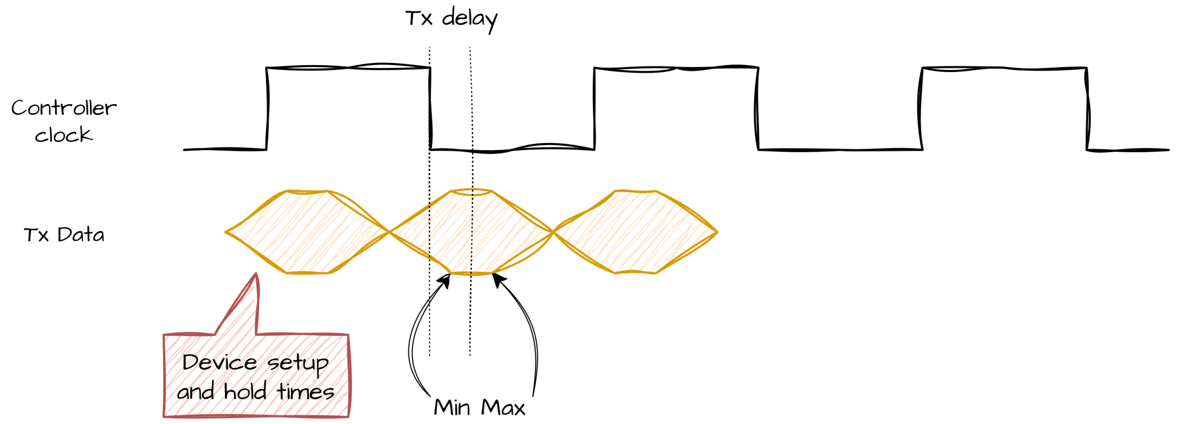
- Flash drivers may provide several variants for the same operation with varying dummy cycles and maximum frequencies
- The theoretical benefit is derived in the `spi-mem` layer when picking the best available variant:
  - Page read in 1S-8S-8S mode @ 86MHz with 8 dummy cycles: 49us
    - Empirically, 11377 kiB/s
  - Page read in 1S-8S-8S mode @ 166MHz with 20 dummy cycles: 25us
    - Empirically, 9516 kiB/s, because bus is at its maximum frequency already
- Support for flexible dummy cycles in v6.17
  - Already used in SPI NAND, soon in SPI NOR probably

SPI memories: Push the car accelerator!

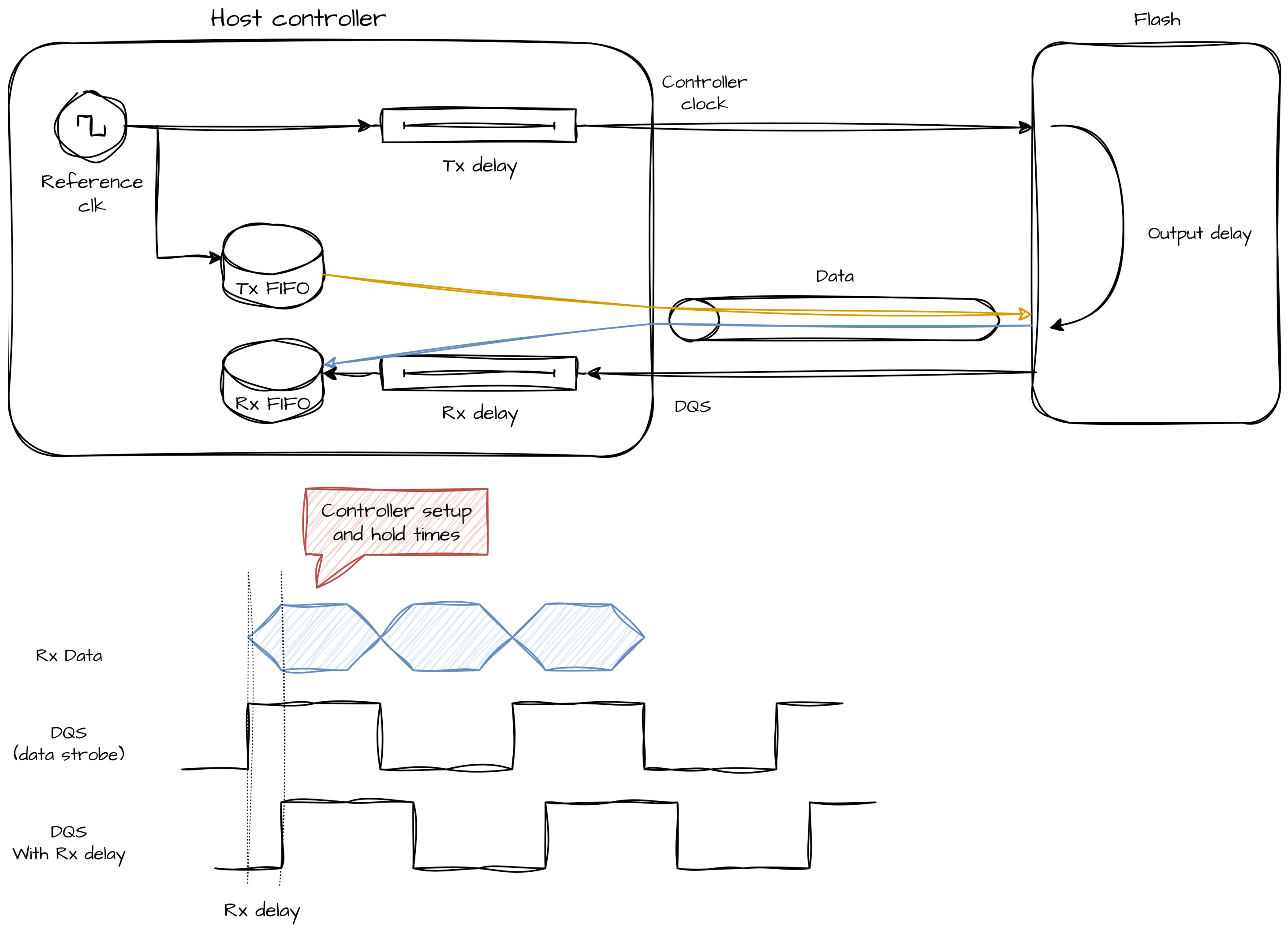
- Many designs accept frequencies in the 20-60MHz range
  - Certain chips support way more (eg. Winbond: 166MHz)
- What about higher frequencies?
- Need to fine tune timings for the data to remain reliably transferred in both directions

SPI memories: High frequencies, problem statement, Tx

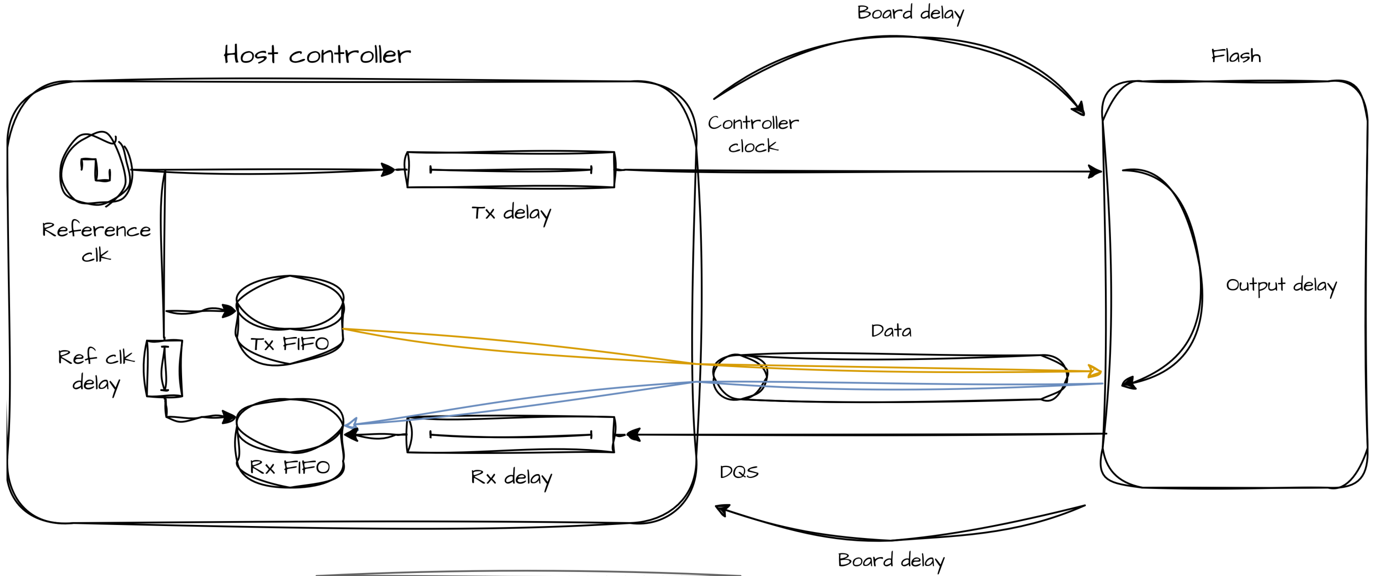


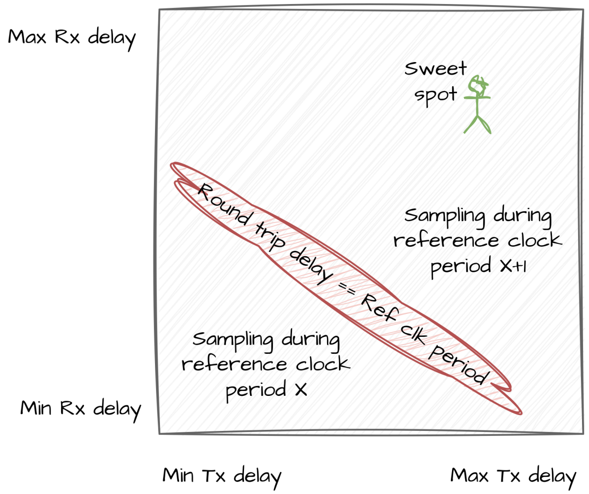


SPI memories: High frequencies, problem statement, Rx



SPI memories: High frequencies, problem statement, Reference clock

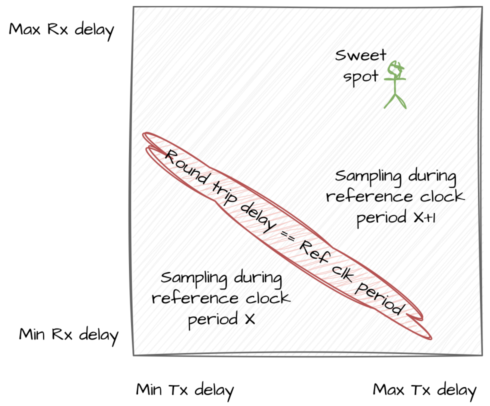




SPI memories: PHY calibration

- Second RFC sent by Santhosh Kumar from TI mid-August for the TI/Cadence controller:
  - `https://lore.kernel.org/linux-spi/20250811193219.731851-1-s-k6@ti.com/`
  - Currently only in the read path, write path to come
- Speed benchmark on the TI platform in octal mode:

kiB/s	Single, No calibration	Octal, No calibration	Octal, With calibration
Read	2342	10711	34133


Max Rx delay
Sweet spot
Round trip delay == Ref clk period
Sampling during reference clock period X+1
Sampling during reference clock period X
Min Rx delay
Min Tx delay
Max Tx delay

Future

- Imagine a world full of serial NAND chips (that's already great)...
  - Supporting continuous reads
  - and octal DTR stateful mode
  - on a PHY calibration enabled platform

Let's call this, 2026?

Thank you!

Questions?

Miquèl Raynal

miquel.raynal@bootlin.com