



Understand your NAND and drive it within Linux

Miquèl Raynal
miquel@bootlin.com

© Copyright 2004-2018, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!





- ▶ Embedded Linux engineer at ~~Free Electrons~~ → 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
 - ▶ **Active contributor to the NAND subsystem**
 - ▶ **Kernel support for various ARM SoCs**
- ▶ Living in **Toulouse**, south west of France



What is this talk about?

- ▶ Introduction to the basics of NAND flash memory
- ▶ How they are driven by the NAND controller
- ▶ Overview of the Linux memory stack, especially the new interface to drive NAND controllers: `->exec_op()`




Disclaimer

- ▶ I am not a NAND expert, more the NAND maintainer slave
- ▶ I will probably oversimplify some aspects
- ▶ This presentation is not about history nor NOR technology
- ▶ Focus on SLC NAND (Single Level Cell) to simplify explanations, logic is similar with MLC/TLC NAND (Multi/Triple Level Cell)



The commercial minute

- ▶ Main purpose: replace hard disks drives
- ▶ Main goal: lowest cost per bit
- ▶ Widely used in many consumer devices, embedded systems...
- ▶ Flavors:
 - ▶ **Raw NAND / parallel NAND** 
 - ▶ Serial NAND (mostly over SPI)
 - ▶ Managed NAND with FTL (Flash Translation Layer)
 - ▶ SD cards
 - ▶ USB sticks
 - ▶ SSD
 - ▶ etc



Understanding the NAND memory cell

Miquèl Raynal
miquel@bootlin.com

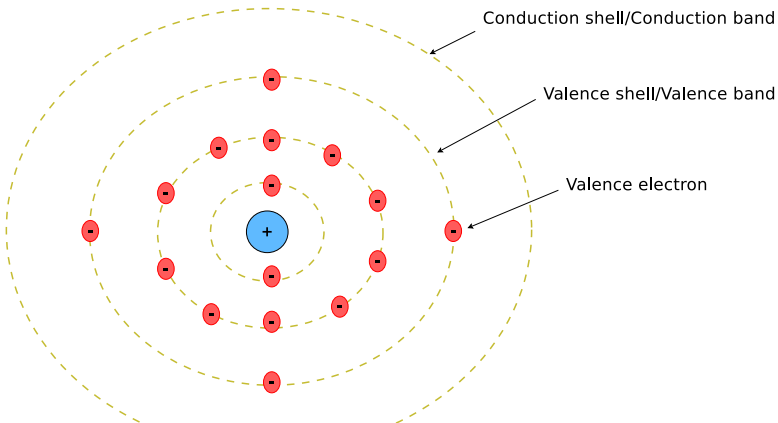
© Copyright 2004-2018, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!





Back to school: Silicon

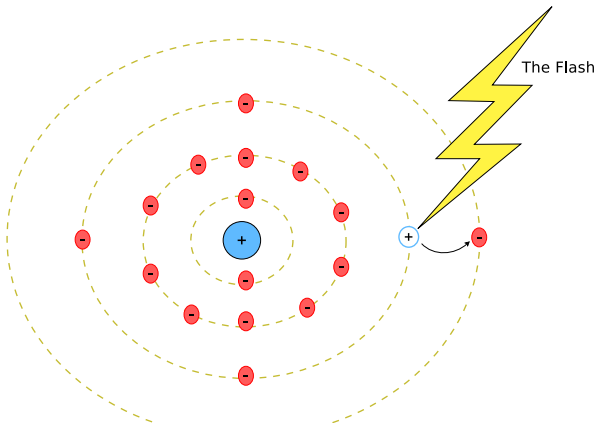
- ▶ Silicon, Si
 - ▶ Electrically balanced (neutral)
 - ▶ 14 electrons spread in 3 orbits
 - ▶ 4 electrons in the valence shell → easy bonding with other Silicon atoms (crystal)





Back to school: electricity

- ▶ Electricity \implies free electrons
 - ▶ Silicon is almost an insulator
 - ▶ Valence electron stroke by light \rightarrow absorbs energy \rightarrow jumps to the conduction band
 - ▶ Free electrons drift randomly unless a voltage is applied \rightarrow attracted to the $+$ side





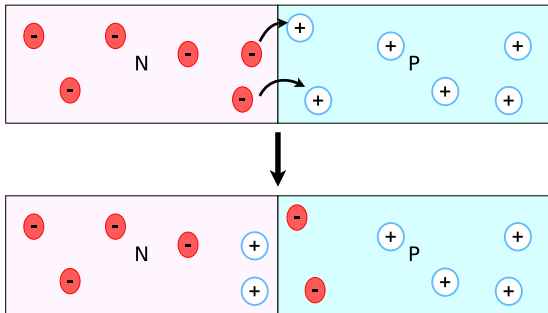
Back to school: doping

- ▶ Nothing to do with cycling
- ▶ Purpose of doping: enhance conductivity
 - ▶ Add impurities (atoms with more or less valence electrons than Si)
 - ▶ Once bound with 4 Si atoms:
 - ▶ 1 free electron \leftarrow N-doping
 - ▶ 1 hole \leftarrow P-doping
 - ▶ Still electrically neutral



P-N junction: the diode

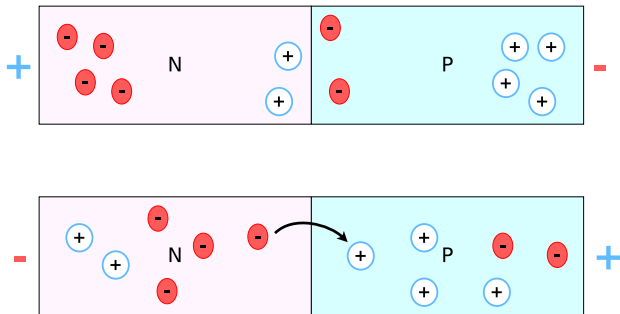
- ▶ Electrons close to the junction will jump to recombine with the closest hole
- ▶ Creation of a barrier of potential: a non-crossable electric field
- ▶ Depletion region thickness is modular





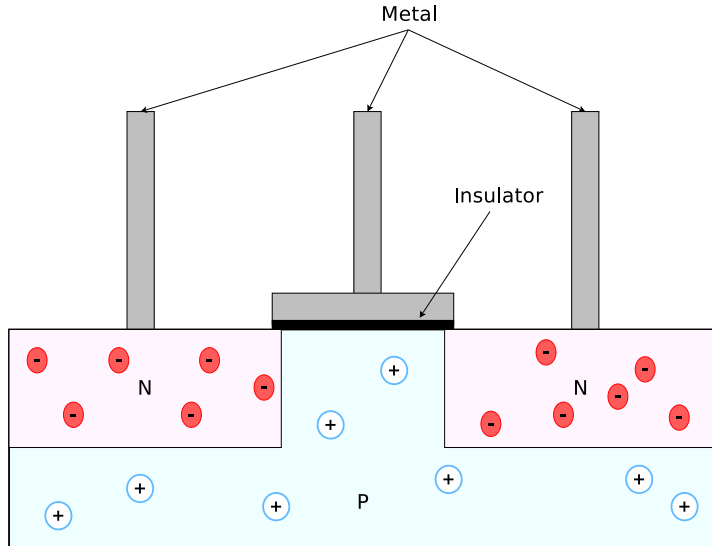
P-N junction: the diode

- ▶ Electrons close to the junction will jump to recombine with the closest hole
- ▶ Creation of a barrier of potential: a non-crossable electric field
- ▶ Depletion region thickness is modular



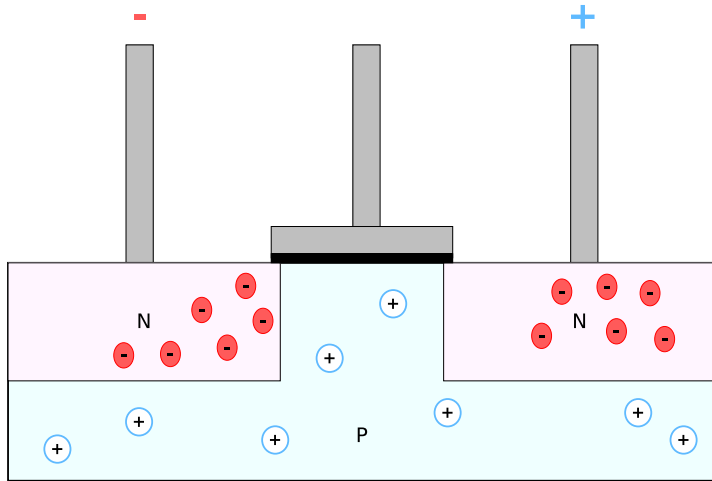


Metal-Oxide-Semiconductor Field-Effect Transistor



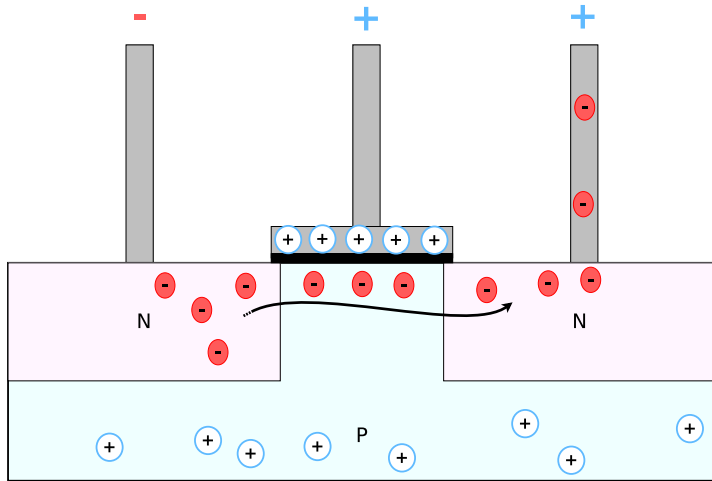


Metal-Oxide-Semiconductor Field-Effect Transistor



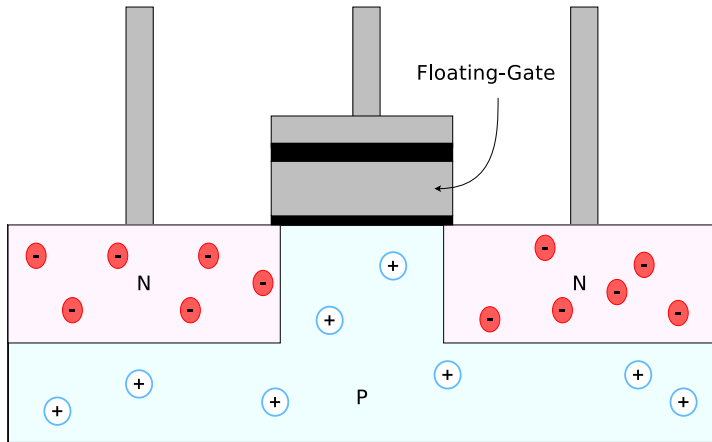


Metal-Oxide-Semiconductor Field-Effect Transistor



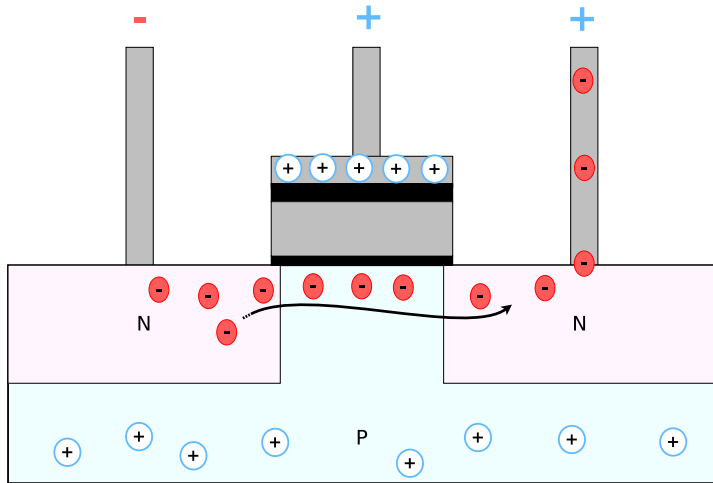


Floating-gate transistor



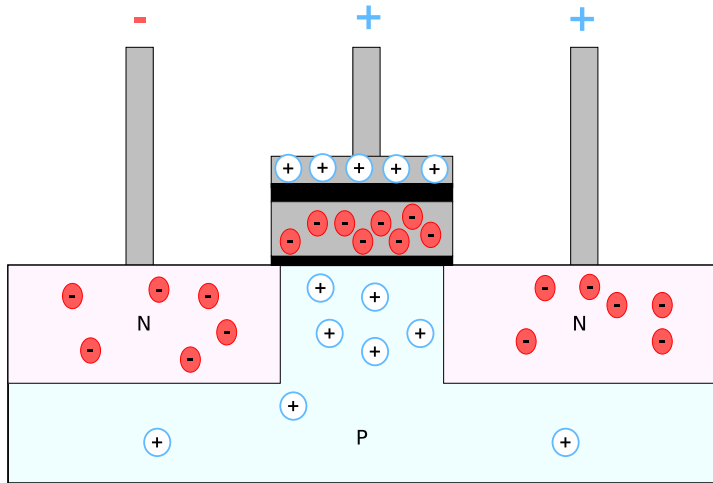


Floating-gate transistor: reading a one





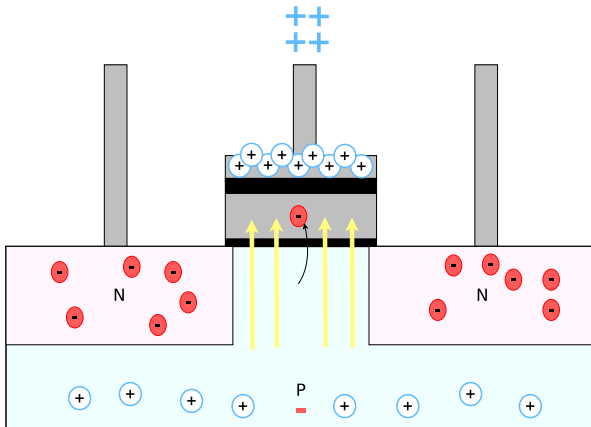
Floating-gate transistor: reading a zero





Programming a cell to a 0 state

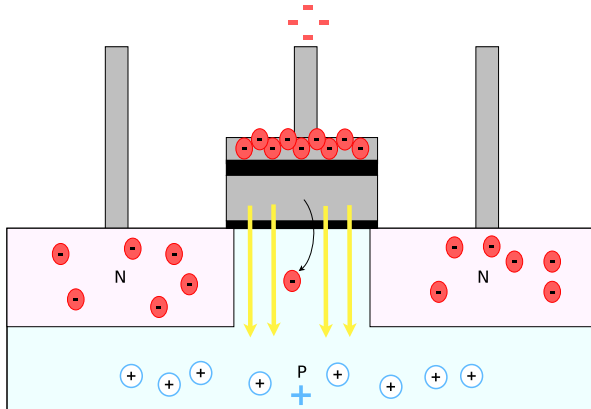
- ▶ Change the charge of the floating-gate
- ▶ No electrical contact → Fowler-Nordheim tunneling





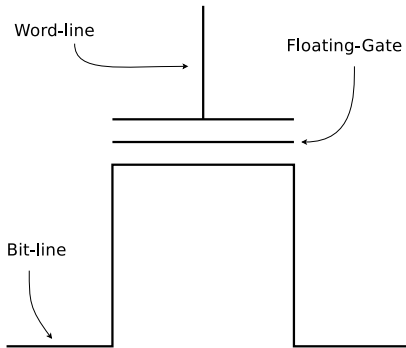
Erasing a cell to a 1 state

- ▶ Reverse the electric field
- ▶ Done by applying a high negative voltage on the control gate



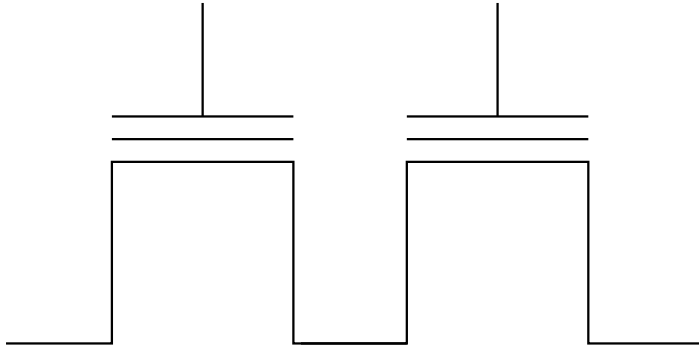


Memory cell



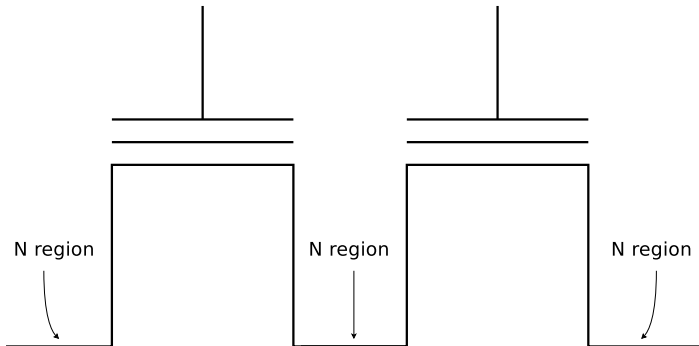


NAND gate



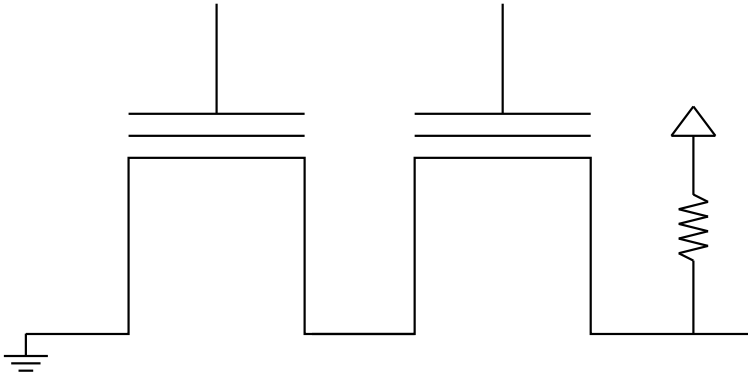


NAND gate



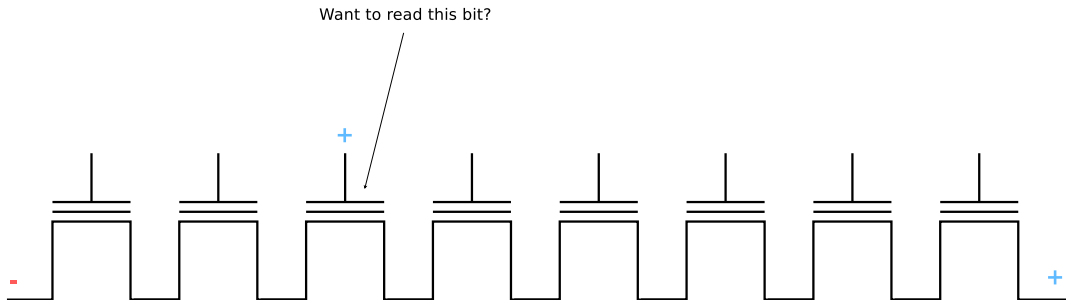


NAND gate



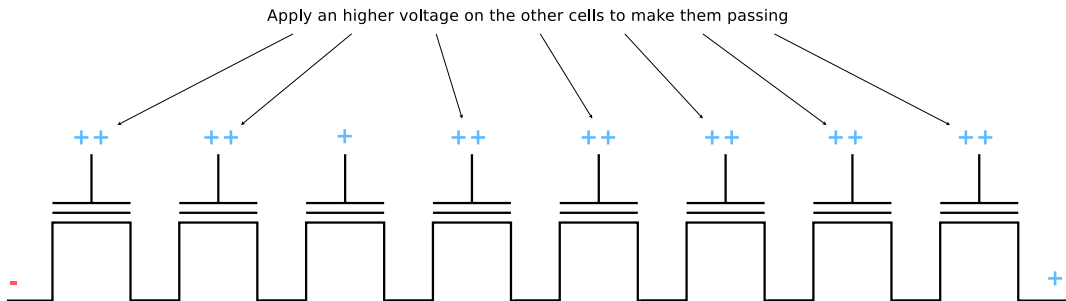


Memory string



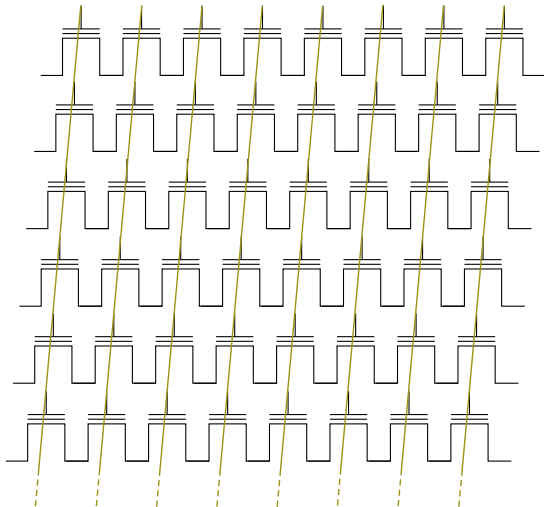


Memory string



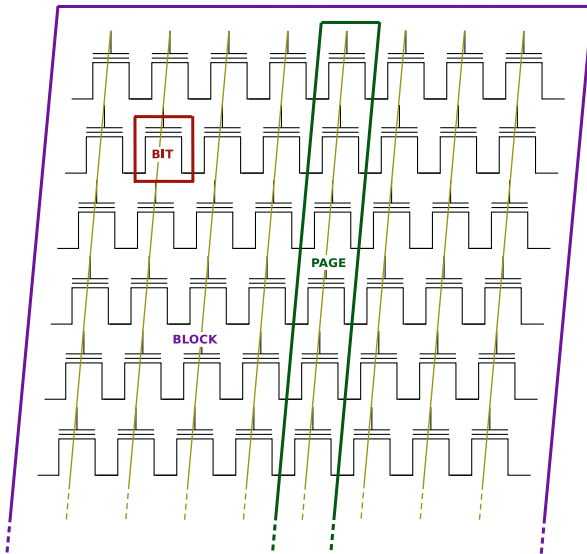


Memory array





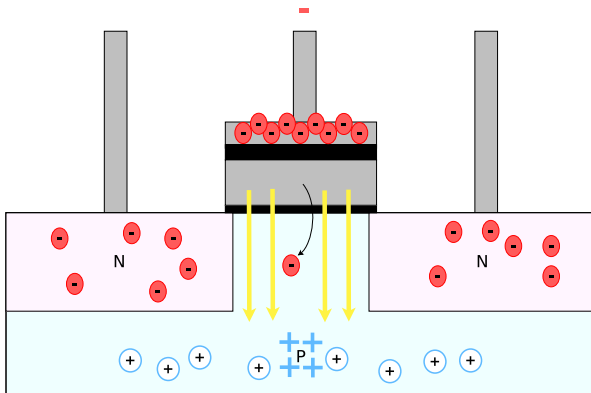
Memory array





Erasing

- ▶ High negative voltage → not that easy to produce
- ▶ Bulk is the same for all cells → “eraseblock”



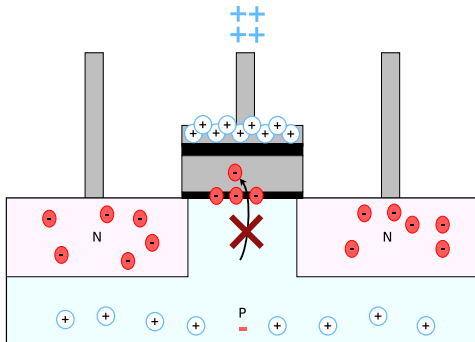


- ▶ **Always** erase before programming
 - ▶ “Clean” erased state is only 1111... everywhere because floating gates are not electrically charged.
 - ▶ Writing is a one-way operation that brings more electrons in the floating-gate.
 - ▶ This is “programming a 0”.
 - ▶ There is no “programming a 1” action.



Main design flaw: bitflips

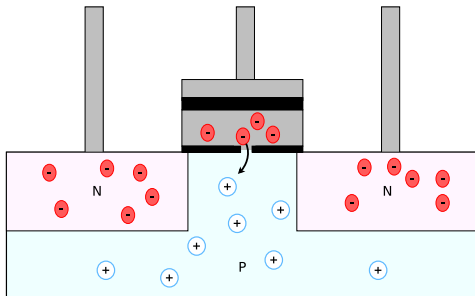
- ▶ Cells may not be fully erased/programmed
 - ▶ Electrons without enough energy might get trapped, creating a depletion region
 - ▶ Oxide becomes negative, preventing tunneling of the electrons if the barrier gets too high





Main design flaw: bitflips

- ▶ Cells may not be fully erased/programmed
 - ▶ Electrons without enough energy might get trapped, creating a depletion region
 - ▶ Oxide becomes negative, preventing tunneling of the electrons if the barrier gets too high
- ▶ Data retention issue
 - ▶ Writing/erasing moves electrons through the oxide layer
 - ▶ Electrons will dissipate their energy colliding with the material, damaging it
→ possible charge loss





Main design flaw: bitflips

- ▶ Cells may not be fully erased/programmed
 - ▶ Electrons without enough energy might get trapped, creating a depletion region
 - ▶ Oxide becomes negative, preventing tunneling of the electrons if the barrier gets too high
- ▶ Data retention issue
 - ▶ Writing/erasing moves electrons through the oxide layer
 - ▶ Electrons will dissipate their energy colliding with the material, damaging it
 - possible charge loss
- ▶ Read/write disturbances
- ▶ ~100k program/erase cycles with SLC NAND



Driving a NAND chip: the NAND controller

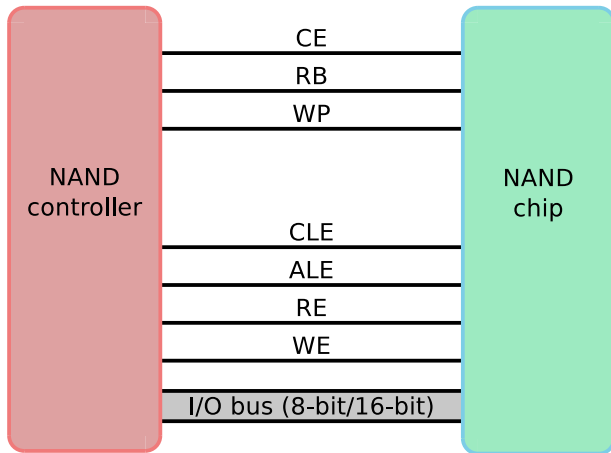
Miquèl Raynal
miquel@bootlin.com

© Copyright 2004-2018, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!



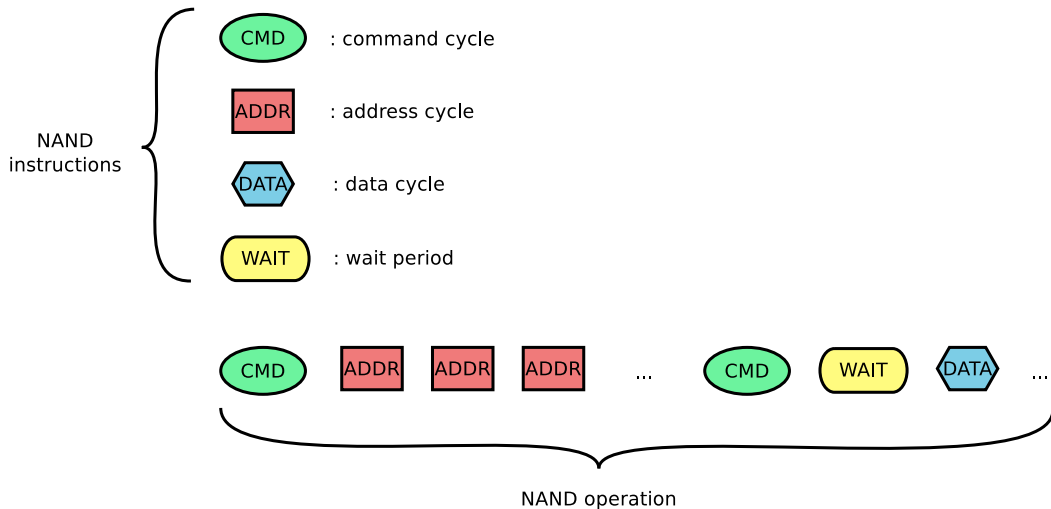


NAND bus



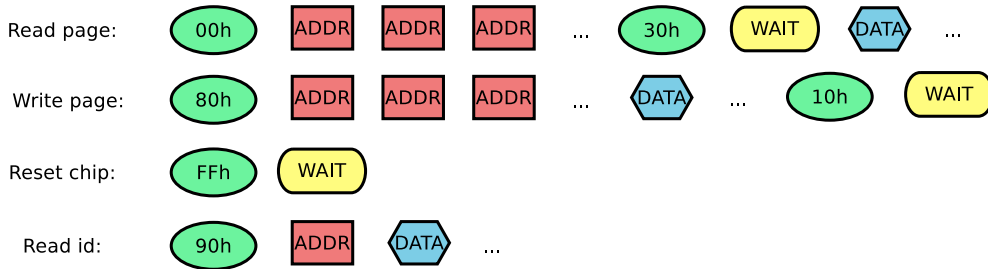


NAND protocol





NAND protocol (examples)





NAND controller

- ▶ Controllers are often embedded in SoCs
- ▶ Diverse implementations, from the most simplest to highly sophisticated ones
- ▶ Controller job: communicate with the NAND chip
 - ▶ Can embed an ECC engine to handle bitflips
 - ▶ Can embed advanced logic to optimize throughput
 - ▶ Sequential accesses
 - ▶ Parallel die accesses



Dealing with NAND from Linux

Miquèl Raynal
miquel@bootlin.com

© Copyright 2004-2018, Bootlin.

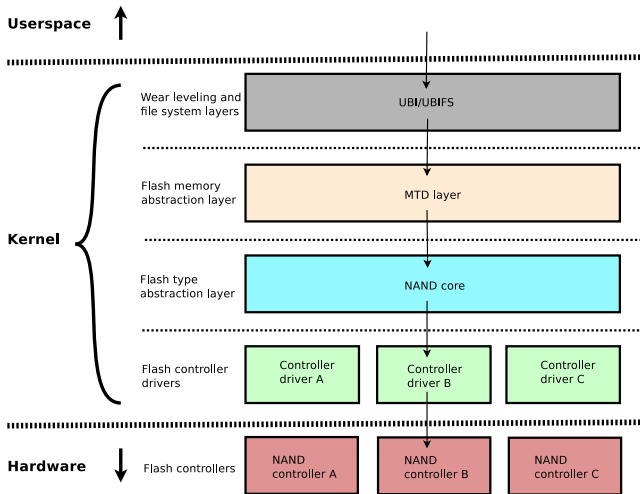
Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!



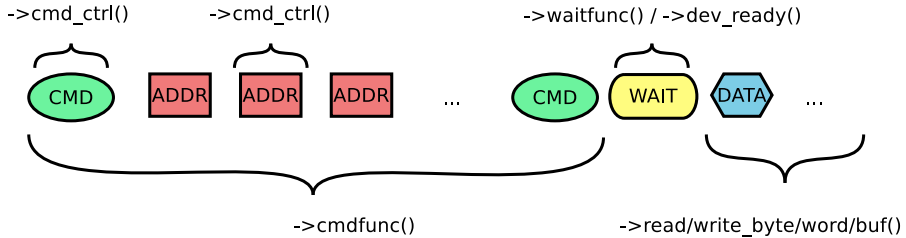


Linux Memory Technology Device stack (MTD)



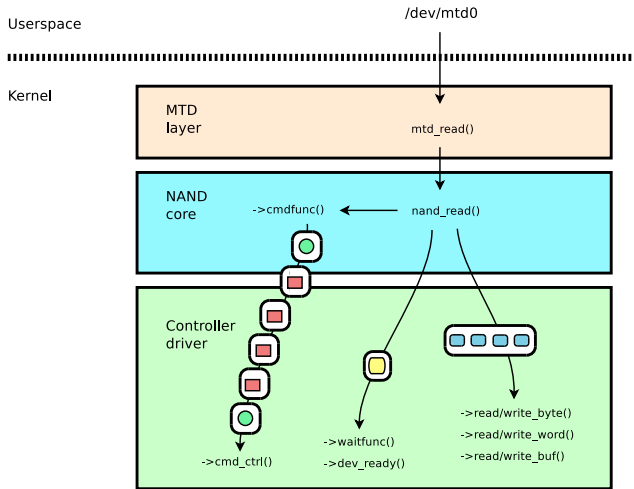


When there were still dinosaurs





NAND legacy stack





Limitations of the old methods

- ▶ NAND controllers have become more complex
 - ▶ Can handle higher-level operations
 - ▶ higher performances?
 - ▶ May provide support for operations that would do all command/address/data cycles in one-go
 - ▶ Some controllers are not able anymore to do basic operations (single cycles)
 - ▶ They cannot send a single command, address or data cycle!
- ▶ Workaround: overload `->cmdfunc()`



Drawbacks of overloading `->cmdfunc()`

- ▶ NAND controller drivers have to re-implement everything
 - ▶ Encourages people to implement a minimal set of commands
- ▶ Logic changes from driver to driver
 - ▶ NAND operations evolve over the time → new vendor specific operations
 - ▶ Hard to maintain as support across the NAND controllers is not uniform
 - ▶ Need to patch all the drivers for additions supposedly simple in the core
 - ▶ According to the NAND maintainer, vendors are still creative

“Why are they so mean to us?!” – Boris Brezillon, 04/01/2018

- ▶ Some controllers need the length of the data transfer
 - ▶ Not available in `->cmdfunc()`
 - ▶ Drivers started predicting what the core “next move” would be
- ▶ Clear symptoms that the framework was not fitting the user needs anymore



Addressing these limitations: `->exec_op()`

- ▶ New interface that asks to execute the whole operation
- ▶ Just a translation in NAND operations of the MTD layer orders
 - ▶ Doesn't try to be smart, logic should be in the NAND framework
- ▶ Calls the controller `->exec_op()` hook and passes it an array of instructions to execute
- ▶ Should fit most NAND controllers we already know about
- ▶ Introduced in Linux v4.16
- ▶ Marvell's NAND controller driver migrated
- ▶ More to come: FSMC, Sunxi, VF610, Arasan, MXC, Atmel...

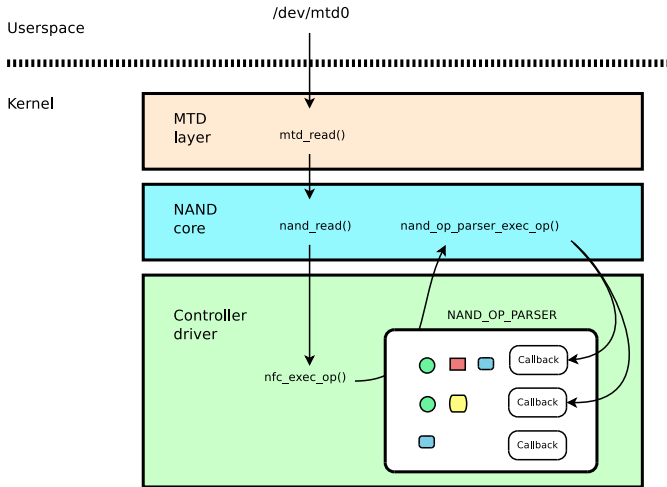


->exec_op() controller's implementation

- ▶ When receiving an array of sequential instructions:
 - ▶ Parses the sequence
 - ▶ Splits in as much sub-operations as needed to perform the task
 - ▶ Declares if the overall operation can be handled
 - ▶ Otherwise returns -ENOTSUPP
- ▶ Simple controllers → trivial logic
- ▶ More complex controllers → use the core's parser

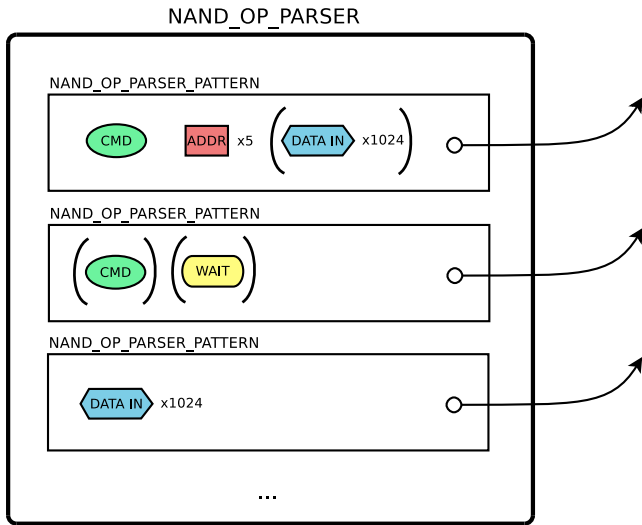


Parser





Patterns





Swipe right to match

Reset



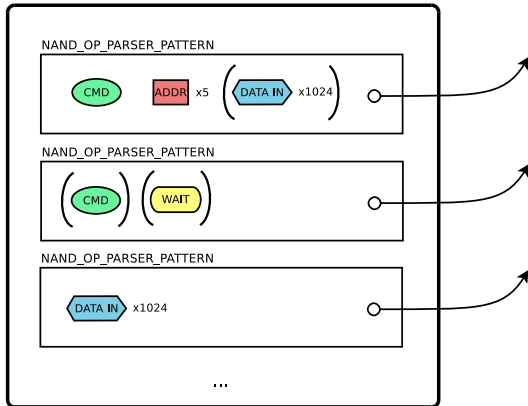
Read ID



Change read column



NAND_OP_PARSER





NAND hooks

- ▶ Various hooks should be implemented by the controller driver
 - ▶ `->exec_op()` is one tool to do “low-level” operations
 - ▶ `->setup_data_interface()` to manage controller timings
 - ▶ `->select_chip()` to select a NAND chip die



Good habits when you hack a NAND controller driver

- ▶ Test with the userspace tools through the `/dev/mtd*` devices
mtd-utils: `nandbiterrs`, `nandreadpage`, `flash_speed`, `flash_erase`,
`nanddump`, `nandwrite`, etc



Good habits when you hack a NAND controller driver

- ▶ Test with the userspace tools through the `/dev/mtd*` devices
mtd-utils: `nandbiterrs`, `nandreadpage`, `flash_speed`, `flash_erase`,
`nanddump`, `nandwrite`, etc
- ▶ Get the NAND documentation
`dd if=/dev/zero of=nand.txt`



Good habits when you hack a NAND controller driver

- ▶ Test with the userspace tools through the `/dev/mtd*` devices
mtd-utils: `nandbiterrs`, `nandreadpage`, `flash_speed`, `flash_erase`,
`nanddump`, `nandwrite`, etc
- ▶ Get the NAND documentation
`dd if=/dev/zero of=nand.txt`
- ▶ Ping the MTD community early on the public mailing-list



Good habits when you hack a NAND controller driver

- ▶ Test with the userspace tools through the `/dev/mtd*` devices
mtd-utils: `nandbiterrs`, `nandreadpage`, `flash_speed`, `flash_erase`,
`nanddump`, `nandwrite`, etc
- ▶ Get the NAND documentation
`dd if=/dev/zero of=nand.txt`
- ▶ Ping the MTD community early on the public mailing-list
- ▶ Do not forget to add the maintainer(s) in copy, it puts them in a bad mood





Sources/Links

- ▶ Presentation by Boris Brezillon (Free Electrons/Bootlin) at ELCE 2016 in Berlin:
“Modernizing the NAND framework, the big picture”
<https://www.youtube.com/watch?v=vhEb0fgk71M>
https://events.linuxfoundation.org/sites/events/files/slides/brezillon-nand-framework_0.pdf
- ▶ Presentation by Arnout Vandecappelle (Essensium/Mind) at ELCE 2016 in Berlin:
“Why NAND flash breaks down”
<https://www.youtube.com/watch?v=VajB8vCsZ3s>
https://sched.ws/hosted_files/openiotelceurope2016/36/Flash-technology-ELCE16.pdf
- ▶ YouTube channel “Learn engineering” that democratizes physical concepts
<https://www.youtube.com/watch?v=7ukDKVHnac4>
- ▶ SlideShare by Nur Baya Binti Mohd Hashim (UNIMAP) about semiconductors
<http://slideplayer.com/slide/10946788>

Questions? Suggestions? Comments?

Miquèl Raynal

`miquel@bootlin.com`

Slides under CC-BY-SA 3.0

<https://bootlin.com/pub/conferences/2018/fosdem/raynal-exec-op/>

Support our crowdfunding campaign to develop
an upstream Linux kernel driver for Allwinner VPU

<https://bootlin.com/blog/allwinner-vpu-crowdfunding/>





- ▶ For throughput or compatibility purpose, a controller driver may overload the following functions defined by the core to bypass `->exec_op()` and talk directly to the NAND controller
 - ▶ `->read/write_page()`
 - ▶ `->read/write_oob()`
 - ▶ Bitflips should be corrected and reported by the controller driver
 - ▶ Let the NAND core handle the rest and report to upper layers
- ▶ It is also mandatory to fill their “raw” counterpart in order to be able to test and debug all the functionalities of the driver
 - ▶ `->read/write_page_raw()`
 - ▶ `->read/write_oob_raw()`