Discover and understand Embedded Linux

Thomas Petazzoni
thomas.petazzoni@bootlin.com

bootlin
embedded Linux and kernel engineering

Who is speaking?

Thomas Petazzoni

- Linux user since ≈1998
- Embedded Linux engineer since 2008
- CEO at Bootlin
- Co-maintainer of Buildroot
- 900+ patches in the Linux kernel
- Co-founder of Toulibre and Capitole du Libre

Bootlin

- Embedded Linux expertise
- Engineering and training
- 80% of business outside of France
- 20 people
- 8000+ patches in the Linux kernel
- Strong open-source culture
- Freely available training materials

Why this talk?

- Most Linux users/developers are aware of Linux on desktops/servers
- But there's another field where Linux is **very widely** used: embedded systems!
- Goal of the talks
  - Show examples of embedded systems where Linux is used
  - HW architecture of Linux-based embedded systems
  - SW architecture of Linux-based embedded systems
- Ready?

Wikipedia to the rescue

An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system

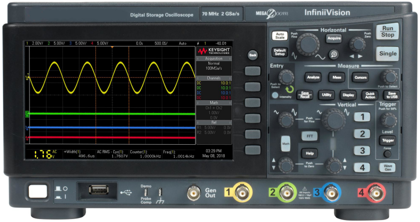
Trend in embedded systems

- Increasing complexity / feature set
  1. Purely analog electronics
  2. Micro-controllers with bare-metal applications (no operating system) or relatively simple real-time operating systems
  3. Micro-processors to run a rich operating system (Linux!) to provide multimedia features, connectivity, security, etc.
- Think about the evolution of TVs

9R/9RT/9RX Tractor comfort and convenience packages
SETTLE IN AND ENJOY THE RIDE
JOHN DEERE
If you're in need of a big-time tractor, there's a good chance you'll be spending big-time hours in the cab. That's why we've pumped up the comfort and convenience across the entire 9 Series lineup.
You may feel like you're cruising in a luxury car, but the results in the field will confirm you're working hard. It's also quiet, thanks to the laminated glass and front console barrier. Enjoy a cooler cab and better visibility to operator screens with rear window tint on the CommandView™ 4 cab glass. And don't forget to stretch those legs. Just because you can.
Amenities are abundant and they vary by package.
Select
Cloth seat with mechanical controls and mechanical lumbar support; 8° left-hand and 40° right-hand swivel
AM/FM/WX radio with auxiliary and Bluetooth inputs; 4 speakers
Business-band ready
4 USB ports and 12V outlet
Dual-tilt steering column
Foot rests
Premium
Cloth seat with electronic controls and pneumatic lumbar support; 25° left-hand and 40° right-hand swivel
6.5-inch (16.5 cm) touchscreen radio, SiriusXM satellite and smartphone ready; 6 speakers with subwoofer
Business-band ready
4 USB ports, 12V outlet
Dual-tilt steering column
Foot rests
Refrigerator
Right-hand accessory rail
110/220V outlet
Ultimate
Leather seat with electronic controls, massage and pneumatic lumbar support; 25° left-hand and 40° right-hand swivel; heated and ventilated; adjustable bolsters
6.5-inch (16.5 cm) touchscreen radio, SiriusXM satellite and smartphone ready; 6 speakers with subwoofer
Business-band ready
4 USB ports, 12V outlet
Dual-tilt steering column
Foot rests
Refrigerator
Right-hand accessory rail
Carpeted floor mat
Leather-wrapped steering wheel
110/220V outlet
Ultimate Comfort and Convenience package shown.

Sports

Electronic equipment



Coffee machine

Counting points at bridge games

Healthcare

Structural monitoring

Digital signage → advertising crap

Consumer devices: multimedia, home automation

Gas filling station

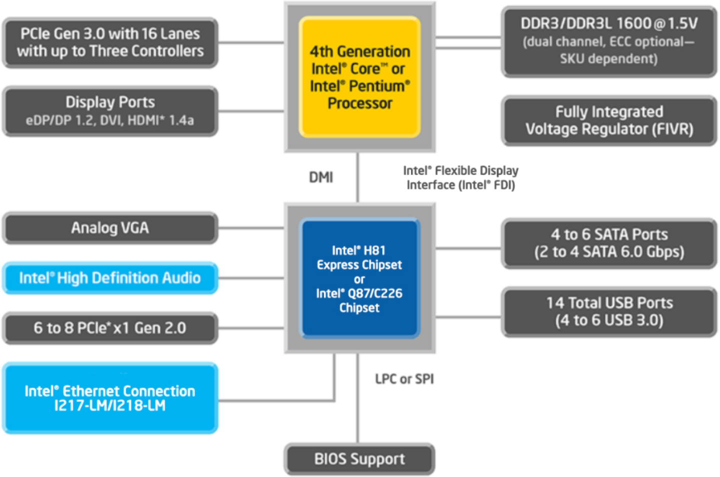
Transportation

Space

Falcon 9 image from NASA, under public domain

Hardware

Intel PC architecture



CPUs in embedded systems

8-bit micro controllers	16-bit micro controllers	32-bit micro controllers	32-bit micro processors	64-bit micro processors
Microchip PIC Microchip AVR ST STM8 ~tens of Mhz ~tens of KB of RAM/flash	Microchip PIC24	ARM Cortex-M RPi RP2040 up to ~100-200 Mhz 100 KB to ~MBs of RAM/flash	ARMv5 ARMv7-A RISC-V 32 x86 500 Mhz to > 1 Ghz many MBs to GB of RAM/flash	ARMv8-A RISC-V 64 x86-64 500 Mhz to > 1 Ghz many MBs to GB of RAM/flash

Internal RAM & flash
No MMU

External RAM and storage
MMU

Bare-metal applications

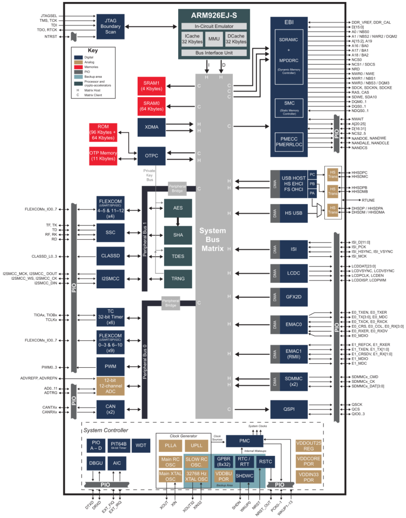
Small real-time OS
RIOT, FreeRTOS, Zephyr, etc.

Linux

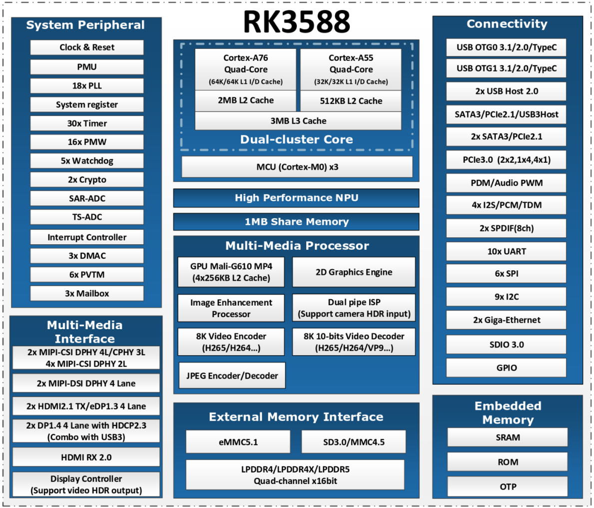
Concept of System-on-chip

- Integrate many HW features in a single chip: *system on chip*
  - One or several CPU cores
  - Many HW interfaces
    - Slow: GPIO, UART, I2C, SPI
    - High-speed: MMC/SDIO, PCIe, USB
    - Multimedia: parallel in/out, DSI, CSI, LVDS, HDMI, I2S
  - Accelerators: crypto, video encoding/decoding, GPU, NPU
- *Silicon vendors* design chips by combining hardware blocks
  - Purchased from IP vendors
  - Designed internally
- IP vendors: ARM, SiFive, Cadence, Synopsys, etc.
- Silicon vendors: NXP, TI, Microchip, Marvell, Rockchip, Qualcomm, etc.
- **Massive** number of SoCs available, to address very different markets

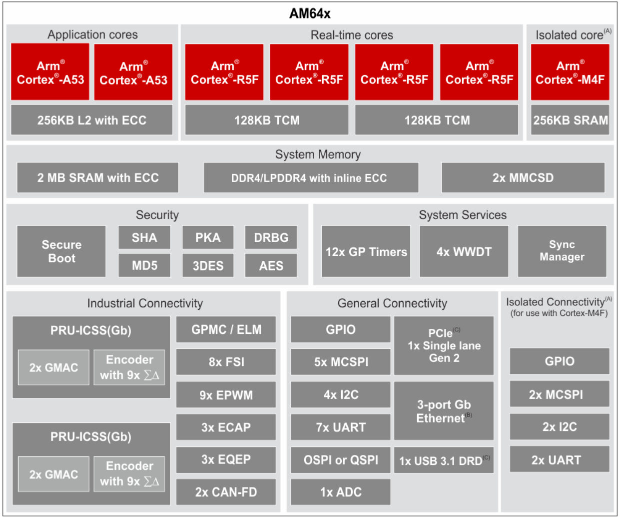
SoC example: Microchip SAM9x60



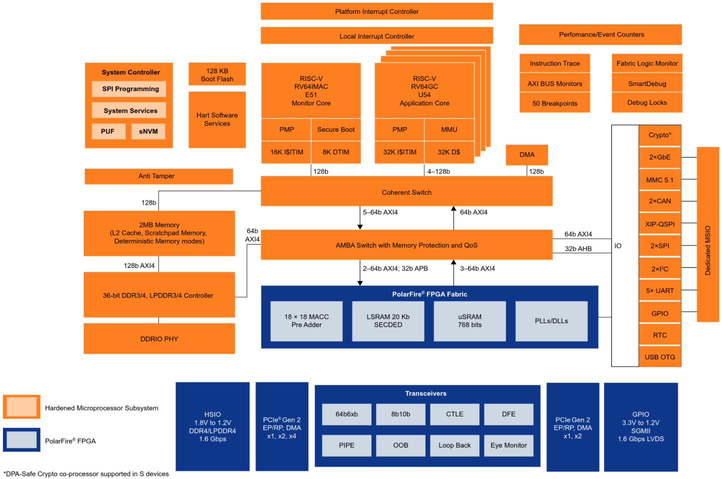
SoC example: Rockchip RK3588



SoC example: Texas Instruments AM64



SoC example: Microchip Polarfire

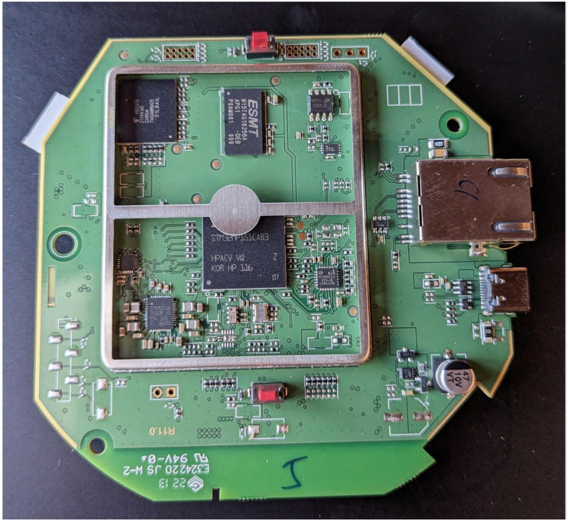


*DPA-Safe Crypto co-processor supported in S devices
**SECDED supported on all MSS memories

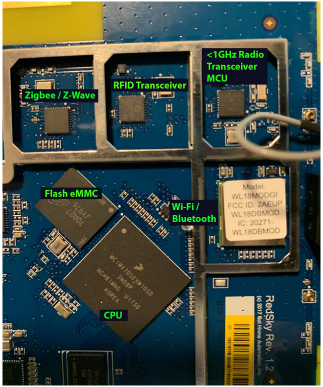
Typical design of embedded hardware

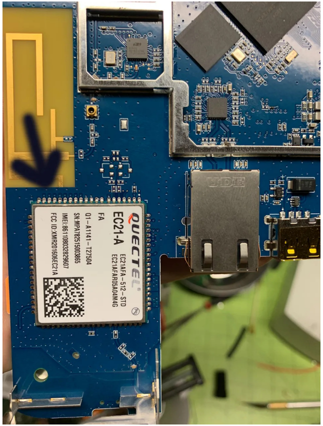
- A custom *board* is typically designed for each embedded product
- A system-on-chip at its core
- RAM: DDR, LPDDR
- Storage: eMMC, SD, flash
- Display panel, sometimes a display bridge, a touchscreen controller
- Audio codec
- Camera sensor
- Ethernet PHY, Ethernet switch
- WiFi, Bluetooth chips, or other radio interfaces
- Power supplies, protections
- Connectors

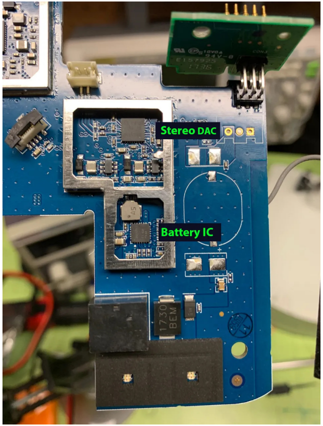
Board example: Ikea smart home hub



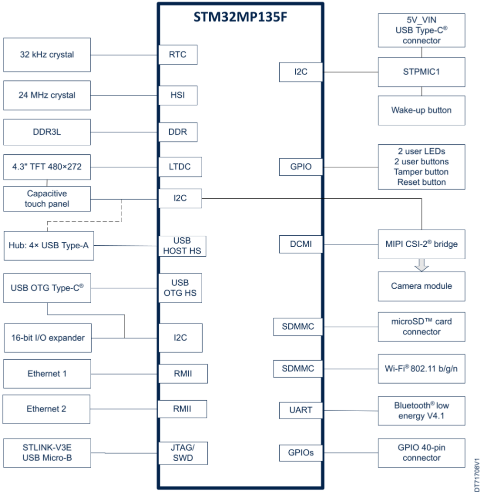
Board example: Amazon Ring







Board block diagram: STM32MP135 evaluation board

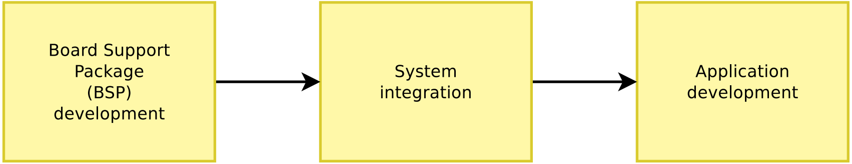


FPGA and micro-controllers

- In Embedded Linux systems, Linux runs on *application processors*
- Some systems have special requirements
  - Very tight real-time requirements
  - Safety requirements
  - Support of custom hardware interfaces
- Integration of
  - Micro-controllers: to run bare-metal code / small real-time OS
  - FPGA: to program custom hardware logic
- … either …
  - Inside the *system-on-chip*
  - or on the *board*

Software

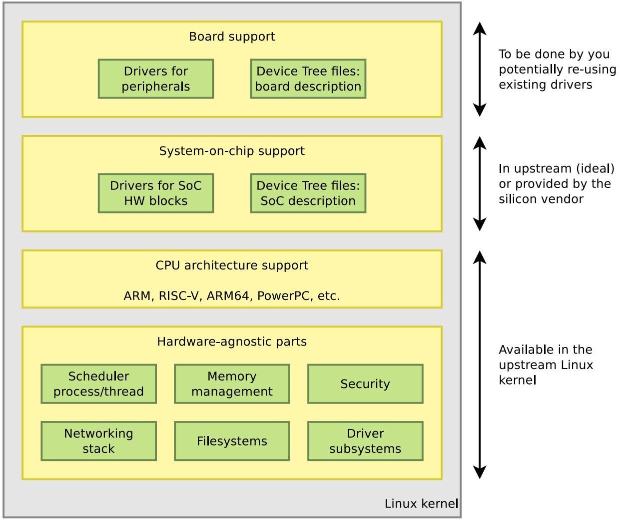
Main development steps



Board Support Package development

- Making sure that the *hardware* is supported by the *software*
- Mostly affects the *bootloader* and *Linux kernel*
- Goal is to have all required hardware interfaces/features supported
- Requires
  - Porting/adaptation
  - Sometimes new drivers
- The hardware in each system-on-chip is often different: different drivers are required
- The hardware in each board is often different: different drivers are required
- There is no such thing as *"a kernel that works on ARM"*

Board Support Package development



BSP development: starting point

- Ideally: the upstream Linux kernel, and an upstream bootloader has support for your SoC, and one or several evaluation boards
- Less ideally: your *silicon vendor* provides a *fork* of Linux + bootloader that includes support for your SoC, and one or several evaluation boards
  - Less ideal because: generally not maintained over the long run, poor quality, non-standard interfaces
- Even less ideally: what your *silicon vendor* provides is so crappy that you can't use it
  - More work/effort for you

BSP development: your work

- Mainly: what is specific to the hardware on your board
- Your hardware often derived from a reference design/evaluation board
- Bootloader level
  - DDR controller configuration, potentially a tricky part
  - UART
  - Storage: eMMC, SD, NAND flash
  - Sometimes networking, or other peripherals depending on the use cases
  - Always: Device Tree (description of HW) + configuration
  - Sometimes: new drivers needed for on-board peripherals
- Linux kernel level
  - All hardware features that are needed
  - Always: Device Tree (description of HW) + configuration
  - Sometimes: new drivers needed for on-board peripherals

Boot flow

- On x86
  - UEFI firmware → GRUB bootloader → Linux kernel
- Most ARM 32-bit platforms
  - ROM code → U-Boot bootloader → Linux kernel
- ARM 64-bit platforms (simplified)
  - ROM code → Trusted Firmware → U-Boot bootloader → Linux kernel
  - Sometimes a trusted operating system, such as OP-TEE
  - Sometimes additional firmware, running on co-processors
- Bootloader is generally *U-Boot*, not *Grub*
- Exact boot flow is specific to the system-on-chip

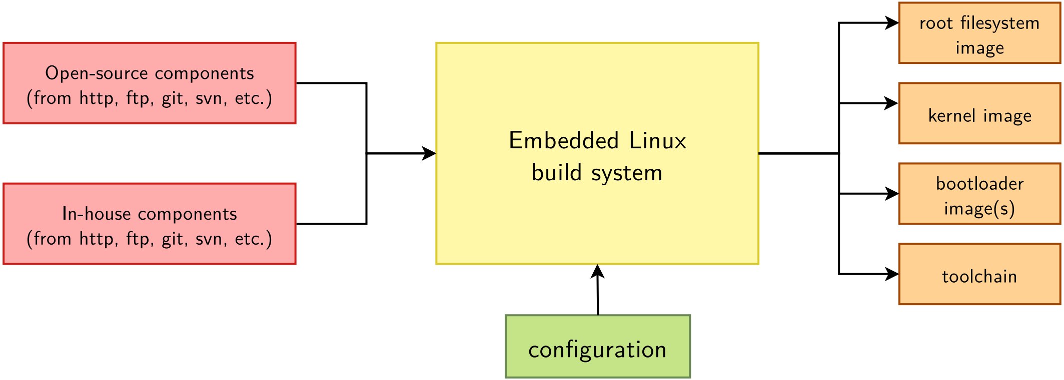
System integration

- How do you integrate all the software components together?
  - Bootloader
  - Linux kernel
  - Open-source components: systemd? Wayland? GStreamer? NetworkManager? Python? NodeJS? Qt? Gtk? OpenCV? etc.
  - In-house components: your libraries/services/applications
- Varying levels of complexity
  - Some embedded Linux systems have a very simple software stack
  - Some need very complex software stacks
  - And everything in-between
- Two main options
  - Binary distributions
  - Embedded Linux build systems

Binary distribution

- What you use on your desktop/server: Debian, Ubuntu, Fedora, RaspberryPi OS
- Provides
  - Pre-compiled packages of many open-source software components
  - Installer
  - Configuration/integration
- Good:
  - Easy to install, easy to install packages, well known
  - Regular updates, including security
- Less good for embedded:
  - Biased towards *installation* rather than *factory install a pre-configured image*
  - Biased towards *native compilation*, and *development on the target*
  - Difficult to have a reproducible image creation process
  - Large, not easy to tweak/optimize, lots of mandatory dependencies

Embedded Linux build system



Embedded Linux build system

- Most popular build systems:
  - *Yocto*, based on *OpenEmbedded*: very powerful, flexible, smart, but complex, steep learning curve
  - *Buildroot*: simpler, but less flexible and somewhat "dumb"
- Everything is rebuilt from source
- Good:
  - Flexibility in the tuning/optimization
  - Reproducibility
  - Smaller footprint: less packages, reduced dependencies
  - Generates an image to flash, not an installer
  - Cross-compilation: fast build machine, no development files on target
- Less good:
  - Need to learn (but fun!)
  - Build time
  - Security updates good, but perhaps not as strong as Debian

Factory flashing and OTA

- Factory flashing
  - Linux on embedded devices is not installed by users
  - Devices are flashed at the factory with an image of the Linux system
  - Image must be fully functional with all software installed and configured
  - Provisioning of MAC address, serial number, keys, etc.
  - The user doesn't even notice there's Linux underneath
- Over-the-air updates
  - Old trend: ship device with a firmware, never touch it again
  - Now, devices must be updated
  - Fix bugs, security issues, deploy new features
  - Generally: image-based update, not package-based
  - Two read-only copies of the system + one data partition
  - Popular OTA solutions: RAUC, swupdate, Mender

Security

- *Cyber-security* regulations more and more strict
- Defensive security
  - Secure boot: authenticate all software running on the device, only boot/run software signed by the manufacturer
  - Encryption: protect the OS/application and/or the user data/configuration
  - Containers, virtualization
  - Mandatory access control: SELinux, AppArmor, etc.
- Updates to fix vulnerabilities
  - Monitor CVEs
  - Long-term support in Linux kernel, Yocto, Buildroot
  - Deploy updates in the field
  - How often? Impact on testing, validation, certification?

Application development

- An embedded Linux system is just a normal Linux system, with usually a smaller selection of components
- In terms of application development, developing on embedded Linux is exactly the same as developing on a desktop Linux system
- All existing skills can be re-used, without any particular adaptation
- All languages, frameworks, libraries, can be integrated into the embedded Linux system
  - Python, Rust, C++, Go, NodeJS, etc.
  - Beware of the limits of your embedded hardware in terms of performance, storage and memory

Want to get started?

- Buy a hardware platform
  - NOT a RaspberryPi!
  - BeagleBoard: BeaglePlay (ARM64), Beagle-V (RISC-V)
  - STM32MP1 discovery kits
  - Plenty of inexpensive boards based on Allwinner or Rockchip processors
- Build your embedded Linux system
  - Configure/build your bootloader
  - Configure/build your kernel
  - Build a custom system with Yocto/Buildroot
  - Explore hardware interfaces
- Don't use vendor solutions/tools: use upstream Linux, upstream U-Boot, upstream Yocto/Buildroot
- Bootlin training materials available free of charge, including practical labs!

Questions?

We're hiring:

- Internships on embedded Linux topics
- Embedded Linux/Linux kernel engineers

Toulouse, Lyon, remote

Thomas Petazzoni
thomas@bootlin.com
https://bootlin.com