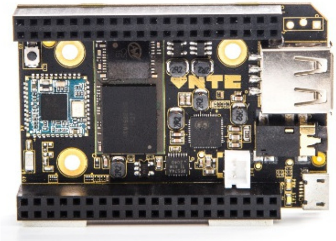
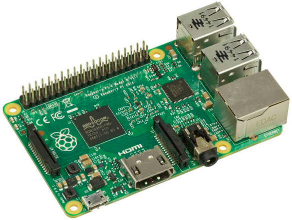
A tour of the ARM architecture and its Linux support



Thomas Petazzoni
Bootlin
thomas.petazzoni@bootlin.com

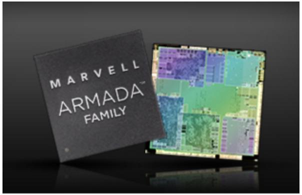


Thomas Petazzoni

- Thomas Petazzoni
- CTO and Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Linux kernel contributors, ARM SoC support, kernel maintainers

Thomas Petazzoni

- Thomas Petazzoni
- CTO and Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Linux kernel contributors, ARM SoC support, kernel maintainers
- Since 2012: Linux kernel contributor, adding support for **Marvell ARM** processors



Thomas Petazzoni

- Thomas Petazzoni
- CTO and Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Linux kernel contributors, ARM SoC support, kernel maintainers
- Since 2012: Linux kernel contributor, adding support for **Marvell ARM** processors
- Core contributor to the **Buildroot** project, an embedded Linux build system

Thomas Petazzoni

- Thomas Petazzoni
- CTO and Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Linux kernel contributors, ARM SoC support, kernel maintainers
- Since 2012: Linux kernel contributor, adding support for **Marvell ARM** processors
- Core contributor to the **Buildroot** project, an embedded Linux build system
- From **Toulouse**, France

Goal and agenda

- ARM is everywhere: in your phone, your TV, your router, your set-top box, your IoT devices, etc.

Goal and agenda

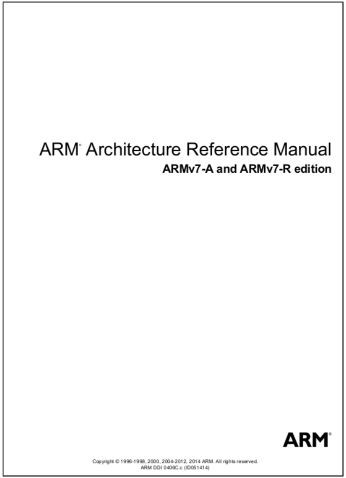
- ARM is everywhere: in your phone, your TV, your router, your set-top box, your IoT devices, etc.
- **Goal**
  - ARM is significantly different from x86
  - More and more Linux developers coming from x86 doing ARM development
  - Number of misunderstandings

Goal and agenda

- ARM is everywhere: in your phone, your TV, your router, your set-top box, your IoT devices, etc.
- **Goal**
  - ARM is significantly different from x86
  - More and more Linux developers coming from x86 doing ARM development
  - Number of misunderstandings
- **Agenda**
  - ARM: from the architecture to the board
  - Software level: bootloader and Linux kernel support

ARM: architecture specification

- *ARM Holdings plc* writes the **specification of the ARM architecture**
  - Instruction-set, including multimedia/DSP oriented instructions
  - MMU
  - Interrupt and exception handling
  - Caches
  - Virtualization
  - etc.
- Over time, improvements of the architecture, with numerous versions: ARMv4, ARMv5, ARMv6, ARMv7, ARMv8
- Takes the form of voluminous documentation, named *ARM ARM*, i.e *ARM Architecture Reference Manual*

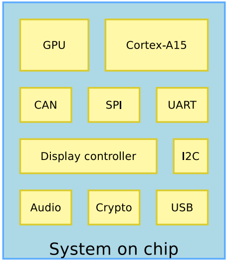

ARM® Architecture Reference Manual
ARMv7-A and ARMv7-R edition
ARM®
Copyright © 1996-1998, 2000, 2004-2012, 2014 ARM. All rights reserved.
ARM DDI 0406C.c (ID051414)

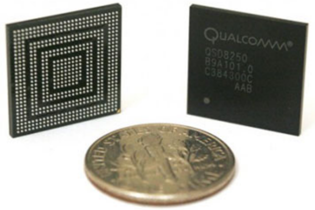
ARM cores: an actual implementation

- *ARM Holdings plc* then creates **IP cores** that implement the specification
- IP core = implementation in VHDL or Verilog of a block of hardware logic
- Examples:
  - ARM926 = implementation of ARMv5
  - ARM1176 = implementation of ARMv6
  - Cortex-A15 = implementation of ARMv7-A
  - Cortex-A53 = implementation of ARMv8-A
- Multiple possible *implementations* for the same *architecture specification*
  - Example: all of Cortex-A5,7,8,9,12,15 implement the same ARMv7-A architecture (with some additions in some cases)
  - Cortex-A5 is a low-power lower-performance implementation, Cortex-A15 is a very high-performance and more power hungry implementation.
  - Difference in internal implementation: depth of the pipeline, out-of-order execution, size of caches, etc.
- This is **NOT** hardware: ARM does **not sell silicon**

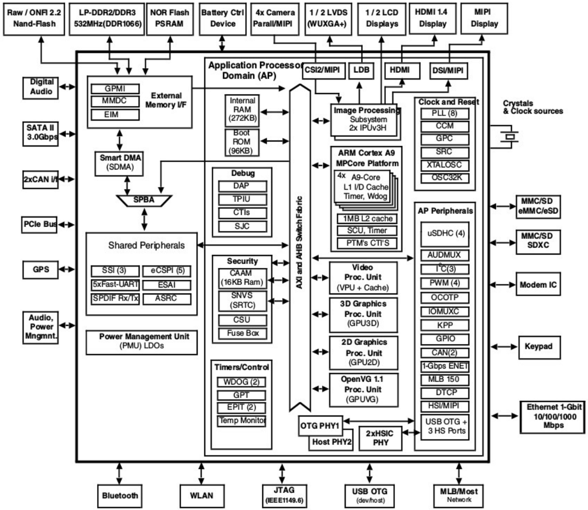
ARM System-on-Chip

- **System-on-chip**: integrated circuit that integrates all components of a computer system
  - CPU, but also peripherals: Ethernet, USB, UART, SPI, I2C, GPU, display, audio, etc.
  - Integrated in a single chip: easier to use, more cost effective
- **SoC vendors**
  - **Buy an ARM core** from ARM
  - Integrate **other IP blocks**, either designed internally, or purchased from other vendors
  - **Create and sell silicon**
- **Large** spectrum of SoCs available, addressing very different markets: automotive, mobile, industrial, low-power, set-top box, etc.



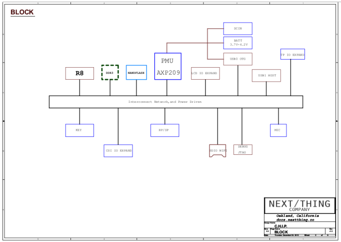


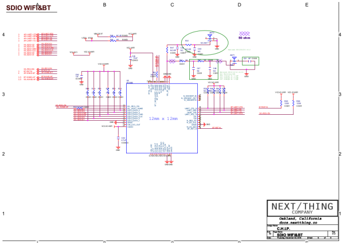
SoC example: Freescale i.MX6 block diagram



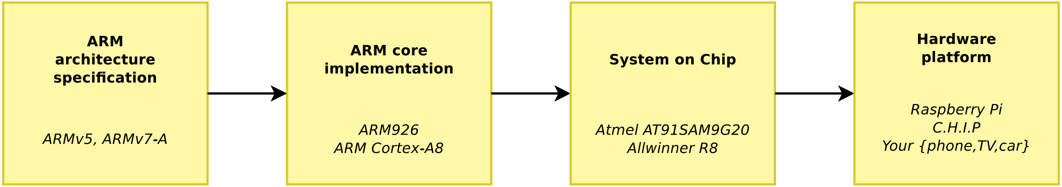
ARM hardware platform

- Even though an SoC is a full *system on a chip*, it is generally not self-sufficient
  - RAM, NAND flash or eMMC, power circuitry
  - Display panel and touchscreen
  - WiFi and Bluetooth chip
  - Ethernet PHY
  - HDMI transceiver
  - CAN transceiver
  - Connectors
- SoC connected to a wide variety of peripherals, through various **busses**
- Laid out on a PCB, with components soldered on it.



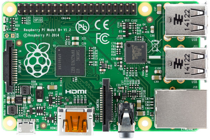


ARM: from the architecture to the board



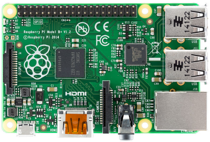
Examples of ARM boards

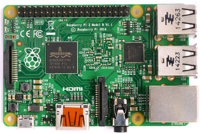
- **RaspberryPi 1**
  - SoC: Broadcom 2835
  - ARM core: ARM1176JZF (single)
  - ARM architecture: ARMv6



Examples of ARM boards

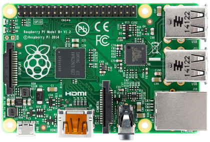
- **RaspberryPi 1**
  - SoC: Broadcom 2835
  - ARM core: ARM1176JZF (single)
  - ARM architecture: ARMv6
- **RaspberryPi 2**
  - SoC: Broadcom 2836
  - ARM core: Cortex-A7 (quad)
  - ARM architecture: ARMv7-A

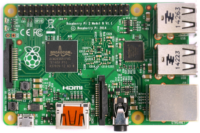


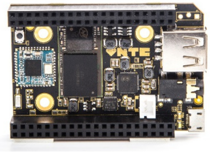


Examples of ARM boards

- **RaspberryPi 1**
  - SoC: Broadcom 2835
  - ARM core: ARM1176JZF (single)
  - ARM architecture: ARMv6
- **RaspberryPi 2**
  - SoC: Broadcom 2836
  - ARM core: Cortex-A7 (quad)
  - ARM architecture: ARMv7-A
- **C.H.I.P**
  - SoC: Allwinner R8
  - ARM core: Cortex-A8 (single)
  - ARM architecture: ARMv7-A

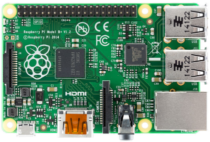


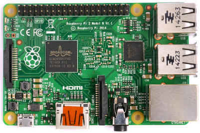


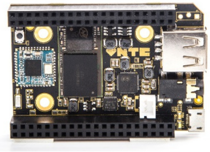


Examples of ARM boards

- **RaspberryPi 1**
  - SoC: Broadcom 2835
  - ARM core: ARM1176JZF (single)
  - ARM architecture: ARMv6
- **RaspberryPi 2**
  - SoC: Broadcom 2836
  - ARM core: Cortex-A7 (quad)
  - ARM architecture: ARMv7-A
- **C.H.I.P**
  - SoC: Allwinner R8
  - ARM core: Cortex-A8 (single)
  - ARM architecture: ARMv7-A
- **ESPRESSOBin**
  - SoC: Marvell Armada 3700
  - ARM core: Cortex-A53 (dual)
  - ARM architecture: ARMv8-A







Software support for hardware layers

- Asking *"does Linux support ARM?"* **doesn't make** a lot of sense

Software support for hardware layers

- Asking *"does Linux support ARM?"* **doesn't make** a lot of sense
- Three "levels" of hardware, three "levels" of software support
  1. The **ARM core**
  2. The **SoC**
  3. The **board**

Software support for hardware layers

- Asking *"does Linux support ARM?"* **doesn't make** a lot of sense
- Three "levels" of hardware, three "levels" of software support
  1. The **ARM core**
  2. The **SoC**
  3. The **board**
- All three levels are needed to support a given hardware platform.
- Also supporting a platform with just the serial port and Ethernet is very different from fully supporting a platform (graphics, audio, power management, etc.).

Three ARMv7 variants

1. **ARMv7-A**, where **A** stands for **Application**
    - Full-featured variant designed for complex operating systems such as Linux.
    - Has a memory management unit (MMU), caches, supports ARM and Thumb2 instruction sets, high performance, VFP and NEON instructions.
    - Cores: Cortex-A8, Cortex-A15.

Three ARMv7 variants

1. **ARMv7-A**, where **A** stands for **Application**
   - Full-featured variant designed for complex operating systems such as Linux.
   - Has a memory management unit (MMU), caches, supports ARM and Thumb2 instruction sets, high performance, VFP and NEON instructions.
   - Cores: Cortex-A8, Cortex-A15.
2. **ARMv7-M**, where **M** stands for **microcontroller**
   - Much smaller variant: **no MMU**, no caches until recently, supports only Thumb2, low performance but also low power.
   - Cores: Cortex-M3, Cortex-M4, Cortex-M7.
   - Generally runs bare metal code, or a small real-time operating system. Linux has support for them, but requires external RAM and flash.

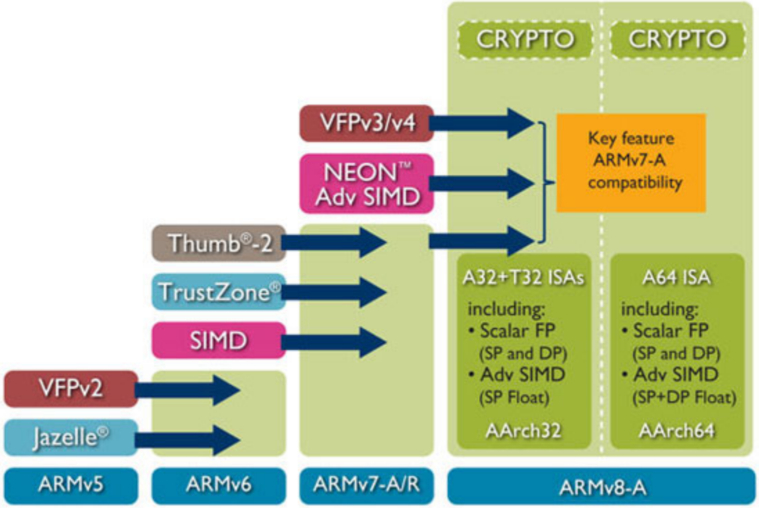
Three ARMv7 variants

1. **ARMv7-A**, where **A** stands for **Application**
   - Full-featured variant designed for complex operating systems such as Linux.
   - Has a memory management unit (MMU), caches, supports ARM and Thumb2 instruction sets, high performance, VFP and NEON instructions.
   - Cores: Cortex-A8, Cortex-A15.
2. **ARMv7-M**, where **M** stands for **microcontroller**
   - Much smaller variant: **no MMU**, no caches until recently, supports only Thumb2, low performance but also low power.
   - Cores: Cortex-M3, Cortex-M4, Cortex-M7.
   - Generally runs bare metal code, or a small real-time operating system. Linux has support for them, but requires external RAM and flash.
3. **ARMv7-R**, where **R** stands for **real-time**
   - Reduced version of the **A** profile, with focus on *deterministic response*
   - Widely used in storage devices (hard drive and SSD controllers)
   - Typically doesn't run Linux.

ARMv8

- Main feature: introduction of **AArch64**, a new instruction set, with 64 bits support
  - AArch64 support is optional: some ARMv8 cores do not support it.
- Also supports a mode called *AArch32*, which offers backward compatibility with *ARMv7-A*
- ARMv8 cores: Cortex-A32 (32 bits only), Cortex-A53, Cortex-A57, Cortex-A72, etc.

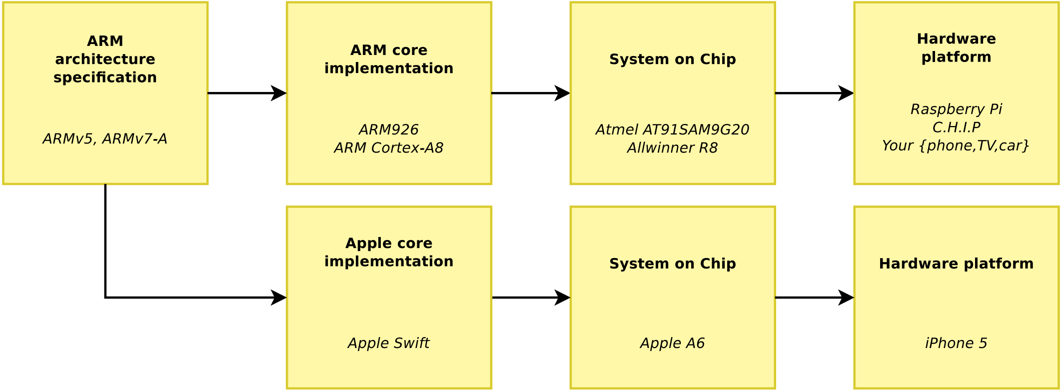
From ARMv5 to ARMv8



Architecture licensees

- Most SoC vendors buy *ARM cores* from ARM, i.e Cortex-A15 or Cortex-A57.
- A few SoC vendors however have an **architecture license**
- They pay a fee to be allowed to **create a CPU core** that implements the same CPU architecture, but do not use the *ARM cores*
- Examples:
  - Marvell Feroceon (ARMv5, used in Marvell Kirkwood), Marvell PJ4 (ARMv7, used in Marvell Armada 370/XP)
  - Qualcomm Scorpion, Qualcomm Krait (ARMv7)
  - Apple Swift (ARMv7, used in the A6), Cyclone (ARMv8, used in the A7)
  - NVidia Denver (ARMv8)
  - Cavium, Broadcom, AppliedMicro, Qualcomm, Samsung (ARMv8)

Architecture licensees: example



Lack of standardization

- ARM architecture specified: **instruction set is compatible** between all ARMv7 cores, between all ARMv8 cores
  - Can run Linux userspace code built for ARMv7 on any ARMv7 platform (provided it's not hardware related)
  - A few optional features (e.g. NEON)
  - Allows to run Ubuntu (built for ARMv7) on any ARMv7 platform
  - However, Ubuntu (built for ARMv7) will not run on RaspberryPi 1 (ARMv6)

Lack of standardization

- ARM architecture specified: **instruction set is compatible** between all ARMv7 cores, between all ARMv8 cores
  - Can run Linux userspace code built for ARMv7 on any ARMv7 platform (provided it's not hardware related)
  - A few optional features (e.g. NEON)
  - Allows to run Ubuntu (built for ARMv7) on any ARMv7 platform
  - However, Ubuntu (built for ARMv7) will not run on RaspberryPi 1 (ARMv6)
- However, **almost no standardization** for the other hardware components: inside the SoC and on the board.
  - Need specific handling at the bootloader and Linux kernel level for each SoC and board.
  - On most ARM SoCs, the hardware inside the chip is memory-mapped. No dynamic discovery/enumeration capability.

No standardization, but lot of HW re-use

- Compatibility of processor cores: they comply with ARM specifications
- For the other hardware blocks, SoC vendors very often
  - **Purchase IP blocks** from third-party vendors: ARM, Cadence, Synopsys, Mentor Graphics, Imagination Technologies, etc.
  - Extensively **re-use IP blocks** between their different SoCs
- Examples:
  - Mentor Graphics MUSB (USB gadget controller) is used in TI, Allwinner and ST SoCs, but also on Blackfin and some MIPS processors
  - The Marvell SPI controller is re-used in Marvell processors shipped over ∼15 years, from old ARMv5 Orions to modern ARMv8 processors.
- This allows to massively re-use drivers!
- Sometimes not that easy to figure out that two IP blocks in different SoCs are actually the same.

BIOS ?

- In terms of booting process, no standardized *BIOS* or *firmware* like on x86 machines.
- Each ARM SoC comes with its own **ROM code** that implements a SoC-specific boot mechanism.
- The early stages of the boot process are therefore specific to each SoC.
- In general: capable of loading a small amount of code from non-volatile storage (NAND, MMC, USB) into a SRAM internal to the processor.
  - External DRAM not initialized yet.
- Often also provides a recovery method, to *unbrick* the platform. Over USB, serial or sometimes Ethernet.
- Used to load a *first stage* bootloader into SRAM, which will itself initialize the DRAM and load/run a *second stage* into DRAM.

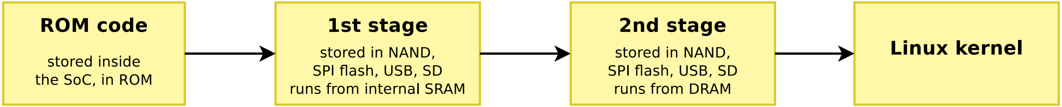
Bootloaders

- Grub(2) typically **not widely used** on ARM platforms
- **U-Boot**, the de-facto standard, found on most development boards and community platforms.
- **Barebox**, less widely used, but very interesting.
- Homemade bootloaders, especially when security/DRM are involved (phone, set-top boxes, etc.)
- Grub starts to gain some traction, especially on ARM64, for the server market
- RaspberryPi is a very special case, with some firmware executed on the GPU, and directly loading the Linux kernel.

Bootloaders (2)

- First stage bootloader provided either by:
  - A **separate project**. Example: *AT91Bootstrap* for Atmel platforms.
  - **U-Boot/Barebox** itself. Concept of *SPL*: minimal version of the bootloader that fits in the constraints of the first stage.
- Interaction with the bootloader typically over the **serial port**
  - U-Boot and Barebox offer a shell, with bootloader specific commands.
  - Sometimes screen/keyboard interaction possible, but not the norm.
  - Embedded without a serial port is weird!

Booting process diagram



Describing hardware

- On x86, most hardware can be dynamically discovered at run-time
  - PCI and USB provide dynamic enumeration capabilities
  - For the rest, ACPI provides tables describing hardware
  - Thanks to this, the kernel doesn't need to know in advance the hardware it will run on
- On ARM, no such mechanism exists at the hardware level
  - In the old days (prior to ~2011), the kernel code itself contained a description of all HW platforms it had to support
  - In ~2011, the ARM kernel developers switched to a different solution for HW description: **Device Tree**
  - Done together with an effort called **multiplatform ARM kernel**

Device Tree

- A **tree of nodes** describing non-discoverable hardware
- Providing **information** such as register addresses, interrupt lines, DMA channels, type of hardware, etc.
- Provided by the **firmware** to the **operating system**
- Operating system agnostic, **not Linux specific**
  - Can be used by bootloaders, BSDs, etc.
- Originates from the PowerPC world, where it has been in use for many more years
- Source format written by developers (`dts`), compiled into a binary format understood by operating systems (`dtb`)
  - One `.dts` for each HW platform

Device Tree example

sun5i.dtsi

```
/ {
        cpus {
                cpu0: cpu@0 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a8";
                        reg = <0x0>;
                };
        };

        soc@01c00000 {
                compatible = "simple-bus";
                ranges;

                uart1: serial@01c28400 {
                        compatible = "snps,dw-apb-uart";
                        reg = <0x01c28400 0x400>;
                        interrupts = <2>;
                        clocks = <&apb1_gates 17>;
                        status = "disabled";
                };

                uart3: serial@01c28c00 {
                        compatible = "snps,dw-apb-uart";
                        reg = <0x01c28c00 0x400>;
                        interrupts = <4>;
                        clocks = <&apb1_gates 19>;
                        status = "disabled";
                };
                [...]
        };
};
```

sun5i-r8-chip.dts

```
/ {
        model = "NextThing C.H.I.P.";
        compatible = "nextthing,chip", "allwinner,sun5i-r8",
                "allwinner,sun5i-a13";

        leds {
                compatible = "gpio-leds";

                status {
                        label = "chip:white:status";
                        gpios = <&axp_gpio 2 GPIO_ACTIVE_HIGH>;
                        default-state = "on";
                };
        };
};

[...]

&uart1 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart1_pins_b>;
        status = "okay";
};
```

Device Tree: in practice

- Used for almost **all ARM platforms** in Linux, and all ARM64 ones
- Used for a few platforms in bootloaders such as U-Boot or Barebox
- Device Tree source code **stored in the Linux kernel tree**
  - Duplicated in U-Boot/Barebox source code as needed
  - Plan for a *central* repository, but never occurred
- Supposed to be **OS-agnostic and therefore backward compatible**
  - In practice, are changed quite often to accommodate Linux kernel changes
- Loaded in memory by the bootloader, together with the Linux kernel image
- Parsed by the Linux kernel at boot time to know which hardware is available

Linux kernel

- Support for the ARM core is generally done by ARM engineers themselves
  - MMU, caches, virtualization, etc.
  - In `arch/arm` and `arch/arm64`
  - Generally in Linux upstream even before actual ARM SoCs with this core are available
- Support for the ARM SoC and HW platform is a different story
  - Requires drivers for each and every HW block, inside the SoC and on the board, in `drivers/`
  - Requires Device Tree descriptions, in `arch/arm(64)/boot/dts`
  - Sometimes supported only in vendor forks, sometimes supported in the upstream Linux kernel

Linux kernel: typical support for an SoC

- Core drivers
  - Clock controllers (`drivers/clk`), reset controller (`drivers/reset`), pin-muxing controllers (`drivers/pinctrl`), interrupt controller (`drivers/irqchip`), timers (`drivers/clocksource`), GPIO controllers (`drivers/gpio`)
- Peripheral drivers
  - Bus controllers: I2C (`drivers/i2c`), SPI (`drivers/spi`), USB (`drivers/usb`), PCI (`drivers/pci`)
  - Display controller (`drivers/gpu/drm`), camera interface (`drivers/media`), touchscreen or other input devices (`drivers/input`), Ethernet controller (`drivers/net`)
- Platform code
  - On ARM, minimal amount of platform code in `arch/arm/mach-<foo>` for power management and SMP support
  - On ARM64, no platform code at all, power management and SMP activities handled using *PSCI*

Linux kernel: from vendor to upstream

- Most vendors **fork the Linux kernel**, and add support for their SoC to their own fork
- Leads to kernel forks with sometimes **millions of added lines** for SoC support
  - Users **cannot easily change/upgrade** their kernel version
  - Generally of **poor quality**
  - Situation got somewhat worse with Android
- Some vendors engage with the **upstream** Linux kernel community, and submit patches
  - More and more vendors taking this direction
  - Mileage may vary depending on the vendor, and sometimes the SoC family
- The **community** also significantly contributes to upstream Linux kernel support for ARM SoCs
  - Example: Allwinner support is fully community-contributed, no involvement from the vendor

Linux kernel: going multiplatform

- Originally, on ARM, a compiled kernel image could only boot on a reduced set of platforms, all using the same SoC
  - Lot of compile-time conditionals
- Wish to have a behavior more similar to *x86*, with one single binary kernel that works for all platforms
- Effort started around making the ARM kernel **multiplatform**
  - Handle more things at runtime rather than at compile time
  - Part of a larger cleanup effort: switch to Device Tree, addition of numerous driver subsystems
- One can now build a single kernel for ARMv4/v5, a single kernel for ARMv6/v7, and a single kernel for ARMv8.
  - `make ARCH=arm multi_v7_defconfig`
  - And it works!

Root filesystem

- Regular desktop-style **distributions**: Debian, Ubuntu, Raspbian, Fedora, etc.

Root filesystem

- Regular desktop-style **distributions**: Debian, Ubuntu, Raspbian, Fedora, etc.
- **Specialized** systems: Android, Tizen, etc.

Root filesystem

- Regular desktop-style **distributions**: Debian, Ubuntu, Raspbian, Fedora, etc.
- **Specialized** systems: Android, Tizen, etc.
- Embedded Linux **build systems**
  - Widely used for embedded systems
  - Produce a Linux root filesystem through cross-compilation
  - Allows a much more customized and stripped down system than a full-blown distribution
  - Examples: OpenEmbedded/Yocto, Buildroot, OpenWRT, etc.

Questions? Suggestions? Comments?

Thomas Petazzoni

thomas.petazzoni@bootlin.com