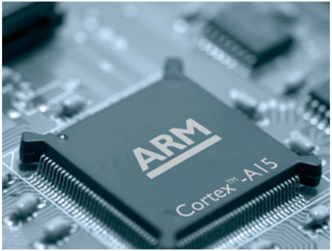
Linux kernel: consolidation in the ARM architecture support



Thomas Petazzoni
Bootlin
thomas.petazzoni@bootlin.com

Thomas Petazzoni

- **Embedded Linux engineer and trainer at Bootlin** since 2008
  - **Embedded Linux development**: kernel and driver development, system integration, boot time and power consumption optimization, consulting, etc.
  - **Embedded Linux training**, Linux driver development training and Android system development training, with materials freely available under a Creative Commons license.
  - `https://www.bootlin.com`
- **Major contributor to Buildroot**, an open-source, simple and fast embedded Linux build system
- Working on **mainlining support for the Marvell Armada 370 and Armada XP ARM SoCs**
- **Speaker** at Embedded Linux Conference, Embedded Linux Conference Europe, FOSDEM, Libre Software Meeting, etc.
- Living in Toulouse, south west of France

Agenda

- The initial complaint
- The ARM "*problem(s)*"
- The solutions
  - Maintainer
  - Device Tree
  - Clock framework
  - Pinctrl subsystem

*Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.*

*Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.*

Linus Torvalds – 17 March 2011

Somebody in the ARM community really needs to step up and tell people to stop dicking around.

ARM Holdings

ARM Holdings is a British company that designs processor cores.

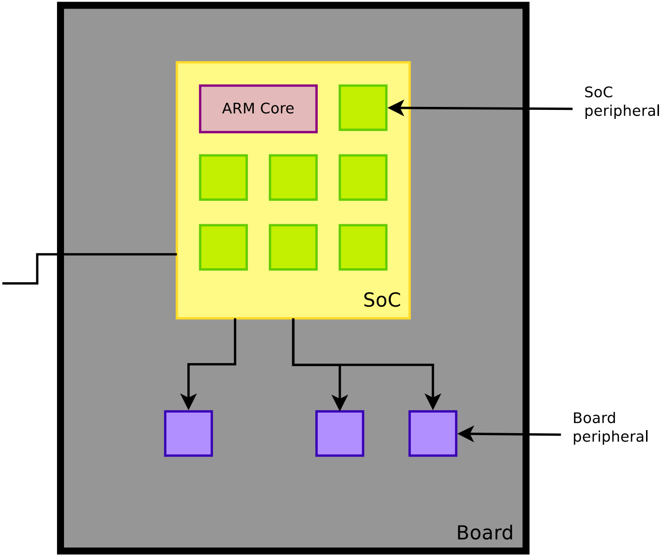
- They define instruction sets, and design cores implementing those instruction sets, memory management units, caches, etc.
- Examples of cores: ARM926EJ-S, Cortex-A8, Cortex-A9, Cortex-M3, etc.
- *ARM* does not produce processors that people can buy

Silicon vendors

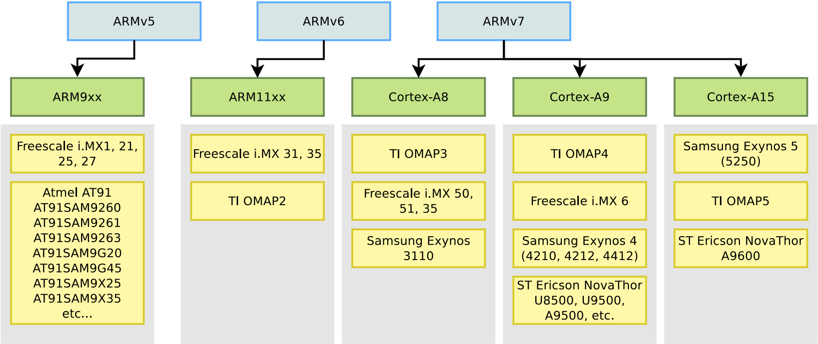
A number of *silicon-vendors* buy those designs, and create *System-on-chip* combining a ARM core and a number of peripherals

- Peripherals are typically UARTs, bus controllers (USB, SPI, I2C, PCI, etc.), Ethernet controllers, ADCs, CAN controllers, video/audio encoding/decoding, graphics (2D, 3D), etc.
- Silicon vendors combine different sets of peripherals to address different markets (industrial, automotive, consumer, etc.)
- Examples: Texas Instruments OMAP4 (dual Cortex-A9), Atmel AT91SAM9M10 (926EJ-S), Freescale i.MX 6 (from 1 to 4 Cortex-A9), etc.

High level overview of an ARM board



Wide variety of ARM platforms



and this is only a very, very partial list. ARM has more than 150 licensees.

Maintenance problem

- This **huge variety of SoCs**, each getting more and more complicated, leads to a **large quantity of code** to support them
- The historical ARM maintainer, Russell King, through which all ARM code was initially going, got **overflowed by the amount of code**
- Code from sub-architectures (SoC families) started to go directly to Linus
- Focus of sub-architecture maintainers on their sub-architecture, **no vision of the other sub-architectures**
- Consequence: a lot of **code duplication**, **no common infrastructures** designed for common problems, similar problems solved differently, etc.

Single kernel image

- Until now, ARM platforms mainly used in the industrial embedded space, or un-modifiable consumer products
  - A given binary kernel image was configured and built **specifically for each platform**
  - All the existing code makes the assumption that the kernel is built for one single platform
- ARM now used in modifiable consumer products (tablets, phones, etc.)
  - Desire for distributions to provide a system for such platforms
  - On x86, easy because one kernel image supports all platforms. On ARM, each platform currently needs one specific kernel image.
  - In the future, **wish to support multiple platforms inside a single kernel image**.

Solutions

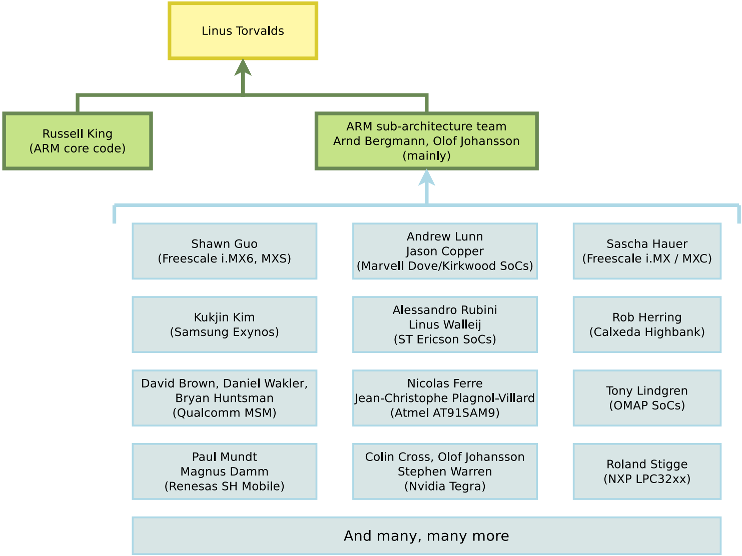
- Sub-architecture maintainer
- Device Tree
- Clock framework
- Pinctrl subsystem

Sub-architecture maintainer

ARM sub-architecture maintainer

- The lack of a cross sub-architecture vision was filled by creating a **team of ARM sub-architecture maintainers**, in May 2011
- Initially with Arnd Bergmann (Linaro/IBM), Nicolas Pitre (Linaro), Marc Zyngier (ARM), later joined by Olof Johansson (Tegra maintainer)
- This team takes care of **reviewing and consolidating the code** of the different sub-architectures, and **sending it to Linus Torvalds**
- Russell King continues to be the maintainer for the ARM core part (memory management, CPU support code, etc.)
- **Important role of Linaro** in this new maintainer team
  - Linaro is a not-for-profit engineering organization consolidating and optimizing open source Linux software and tools for the ARM architecture.

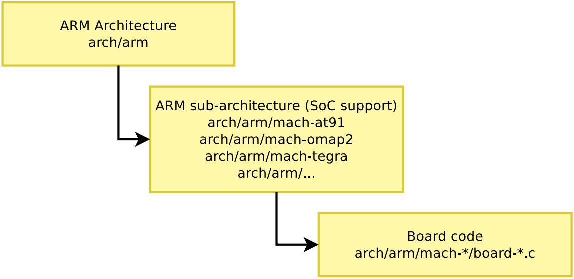
Flow of code


Linus Torvalds
Russell King
(ARM core code)
ARM sub-architecture team
Arnd Bergmann, Olof Johansson
(mainly)
Shawn Guo
(Freescale i.MX6, MXS)
Andrew Lunn
Jason Copper
(Marvell Dove/Kirkwood SoCs)
Sascha Hauer
(Freescale i.MX / MXC)
Kukjin Kim
(Samsung Exynos)
Alessandro Rubini
Linus Walleij
(ST Ericson SoCs)
Rob Herring
(Calxeda Highbank)
David Brown, Daniel Wakler,
Bryan Huntsman
(Qualcomm MSM)
Nicolas Ferre
Jean-Christophe Plagnol-Villard
(Atmel AT91SAM9)
Tony Lindgren
(OMAP SoCs)
Paul Mundt
Magnus Damm
(Renesas SH Mobile)
Colin Cross, Olof Johansson
Stephen Warren
(Nvidia Tegra)
Roland Stigge
(NXP LPC32xx)
And many, many more

Device tree

Definition of platform details

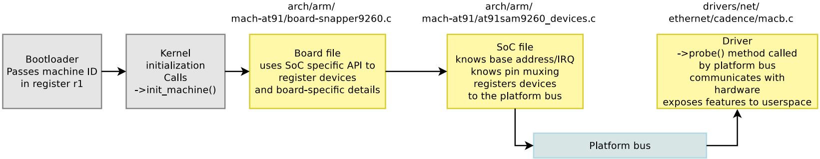
- All the board-specific details and SoC-specific details require specific C code to support new platforms
- A lot of very similar C code to support each and every board, to list all the peripherals, their configuration, etc.
- Code organized in a hierarchy



List of files in `arch/arm/mach-at91`

```
at91rm9200.c
at91rm9200_devices.c
at91rm9200_time.c
at91sam9260.c
at91sam9260_devices.c
at91sam9261.c
at91sam9261_devices.c
at91sam9263.c
at91sam9263_devices.c
at91sam926x_time.c
at91sam9_alt_reset.S
at91sam9g45.c
at91sam9g45_devices.c
at91sam9g45_reset.S
at91sam9n12.c
at91sam9rl.c
at91sam9rl_devices.c
at91sam9x5.c
at91x40.c
at91x40_time.c
board-1arm.c
board-afeb-9260v1.c
board-cam60.c
board-carmeva.c
board-cpu9krea.c
board-cpuat91.c
board-csb337.c
board-csb637.c
board-dt.c
board-eb01.c
board-eb9200.c
board-ecbat91.c
board-eco920.c
board-flexibity.c
board-foxg20.c
board-gsia18s.c
board-kafa.c
board-kb9202.c
board-neocore926.c
board-pcontrol-g20.c
board-picotux200.c
board-qil-a9260.c
board-rm9200dk.c
board-rm9200ek.c
board-rsi-ews.c
board-sam9260ek.c
board-sam9261ek.c
board-sam9263ek.c
board-sam9g20ek.c
board-sam9-l9260.c
board-sam9m10g45ek.c
board-sam9rlek.c
board-snapper9260.c
board-stamp9g20.c
board-usb-a926x.c
board-yl-9200.c
clock.c
clock.h
cpuidle.c
generic.h
gpio.c
include
irq.c
Kconfig
leds.c
Makefile
Makefile.boot
pm.c
pm.h
pm_slowclock.S
sam9_smc.c
sam9_smc.h
setup.c
soc.h
```

Old probing mechanism: overview



The board file

```
static struct macb_platform_data snapper9260_macb_data = {
        .phy_irq_pin    = -EINVAL,
        .is_rmii        = 1,
};

static struct i2c_board_info __initdata snapper9260_i2c_devices[] = {
        {        I2C_BOARD_INFO("max7312", 0x28),
                .platform_data = &snapper9260_io_expander_data, },
        {       I2C_BOARD_INFO("tlv320aic23", 0x1a),            },
};

static void __init snapper9260_board_init(void)
{
        at91_add_device_i2c(snapper9260_i2c_devices,
                            ARRAY_SIZE(snapper9260_i2c_devices));
        at91_register_uart(0, 0, 0);
        at91_register_uart(AT91SAM9260_ID_US0, 1,
                           ATMEL_UART_CTS | ATMEL_UART_RTS);
        at91_add_device_eth(&snapper9260_macb_data);
        [...]
}

MACHINE_START(SNAPPER_9260, "Bluewater Systems Snapper 9260/9G20 module")
        [...]
        .init_machine   = snapper9260_board_init,
MACHINE_END
```

The board file registers devices (from `arch/arm/mach-at91/board-snapper9260.c`)

The SoC file: device definition

```
static struct macb_platform_data eth_data;

static struct resource eth_resources[] = {
        [0] = {
                .start  = AT91SAM9260_BASE_EMAC,
                .end    = AT91SAM9260_BASE_EMAC + SZ_16K - 1,
                .flags  = IORESOURCE_MEM,
        },
        [1] = {
                .start  = AT91SAM9260_ID_EMAC,
                .end    = AT91SAM9260_ID_EMAC,
                .flags  = IORESOURCE_IRQ,
        },
};

static struct platform_device at91sam9260_eth_device = {
        .name           = "macb",
        .id             = -1,
        .dev            = {
                                .dma_mask               = &eth_dmamask,
                                .coherent_dma_mask      = DMA_BIT_MASK(32),
                                .platform_data          = &eth_data,
        },
        .resource       = eth_resources,
        .num_resources  = ARRAY_SIZE(eth_resources),
};
```

The SoC file: device registration

```
void __init at91_add_device_eth(struct macb_platform_data *data)
{
        [...]
        if (gpio_is_valid(data->phy_irq_pin)) {
                at91_set_gpio_input(data->phy_irq_pin, 0);
                at91_set_deglitch(data->phy_irq_pin, 1);
        }

        /* Pins used for MII and RMII */
        at91_set_A_periph(AT91_PIN_PA19, 0);    /* ETXCK_EREFCK */
        at91_set_A_periph(AT91_PIN_PA17, 0);    /* ERXDV */
        at91_set_A_periph(AT91_PIN_PA14, 0);    /* ERX0 */
        at91_set_A_periph(AT91_PIN_PA15, 0);    /* ERX1 */
        at91_set_A_periph(AT91_PIN_PA18, 0);    /* ERXER */
        at91_set_A_periph(AT91_PIN_PA16, 0);    /* ETXEN */
        at91_set_A_periph(AT91_PIN_PA12, 0);    /* ETX0 */
        at91_set_A_periph(AT91_PIN_PA13, 0);    /* ETX1 */
        at91_set_A_periph(AT91_PIN_PA21, 0);    /* EMDIO */
        at91_set_A_periph(AT91_PIN_PA20, 0);    /* EMDC */

        if (!data->is_rmii) {
              [...]
        }

        eth_data = *data;
        platform_device_register(&at91sam9260_eth_device);
}
```

The driver part

```
static int __init macb_probe(struct platform_device *pdev)
{
    [...]
}

static int __exit macb_remove(struct platform_device *pdev)
{
    [...]
}

static struct platform_driver macb_driver = {
        .remove         = __exit_p(macb_remove),
        .driver         = {
                .name   = "macb",
                .owner  = THIS_MODULE,
        },
};

static int __init macb_init(void)
{
        return platform_driver_probe(&macb_driver, macb_probe);
}

static void __exit macb_exit(void)
{
        platform_driver_unregister(&macb_driver);
}

module_init(macb_init);
module_exit(macb_exit);
```

Device Tree: principle

- The general idea of the *Device Tree* is to **separate a large part of the hardware description from the kernel sources**
- The Device Tree is a **tree of nodes**, describing the different hardware components of a system and their characteristics
- Written in a **specialized language**, the *Device Tree Source* is compiled into a *Device Tree Blob* by the *Device Tree Compiler*
- This mechanism takes its roots from *OpenFirmware* used on some PowerPC platforms, and has been used on all PowerPC platforms for a long time.
- It is now also being used in other architectures in the Linux kernel such as Microblaze, OpenRISC and C6X.
- Inheritance mechanism: `.dts` files for boards, `.dtsi` for include files

Device Tree: tegra20.dtsi

```
/include/ "skeleton.dtsi"

/ {
    compatible = "nvidia,tegra20";
    interrupt-parent = <&intc>;

    intc: interrupt-controller {
        compatible = "arm,cortex-a9-gic";
        reg = <0x50041000 0x1000
               0x50040100 0x0100>;
        interrupt-controller;
        #interrupt-cells = <3>;
    };

    serial@70006000 {
        compatible = "nvidia,tegra20-uart";
        reg = <0x70006000 0x40>;
        reg-shift = <2>;
        interrupts = <0 36 0x04>;
        status = "disable";
    };

    serial@70006040 {
        compatible = "nvidia,tegra20-uart";
        reg = <0x70006040 0x40>;
        reg-shift = <2>;
        interrupts = <0 37 0x04>;
        status = "disable";
    };

    i2c@7000c000 {
        compatible = "nvidia,tegra20-i2c";
        reg = <0x7000c000 0x100>;
        interrupts = <0 38 0x04>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "disable";
    };

    i2c@7000c400 {
        compatible = "nvidia,tegra20-i2c";
        reg = <0x7000c400 0x100>;
        interrupts = <0 84 0x04>;
        #address-cells = <1>;
        #size-cells = <0>;
        status = "disable";
    };

    usb@c5004000 {
        compatible = "nvidia,tegra20-ehci","usb-ehci";
        reg = <0xc5004000 0x4000>;
        interrupts = <0 21 0x04>;
        phy_type = "ulpi";
        status = "disable";
    };

    [...]
};
```

Device Tree: tegra-harmony.dts

```
/dts-v1/;

/include/ "tegra20.dtsi"

/ {
    model = "NVIDIA Tegra2 Harmony evaluation board";
    compatible = "nvidia,harmony", "nvidia,tegra20";

    memory {
        reg = <0x00000000 0x40000000>;
    };

    serial@70006300 {
        status = "okay";
        clock-frequency = <216000000>;
    };

    i2c@7000c400 {
        status = "okay";
        clock-frequency = <400000>;
    };

    i2c@7000c000 {
        status = "okay";
        clock-frequency = <400000>;

        wm8903: wm8903@1a {
            compatible = "wlf,wm8903";
            reg = <0x1a>;
            interrupt-parent = <&gpio>;
            interrupts = <187 0x04>;

            gpio-controller;
            #gpio-cells = <2>;
            [...]
        };
    };

    usb@c5004000 {
        status = "okay";
        nvidia,phy-reset-gpio = <&gpio 169 0>;
    };

    [...]
};
```

Device Tree usage

- When the `.dts` file is in `arch/arm/boot/dts`, as simple as:
  `make ARCH=arm foobar.dtb`
- Then, on ARM, two cases:
  1. **Your bootloader has DT support**.
     You need to load both your kernel image and DT image, and start the kernel with both addresses. The DTB address is passed to the kernel in register `r2`, instead of the *ATAG* address. With U-Boot:
     `bootm kerneladdr - dtbaddr`
  2. **Your bootloader does not have DT support**.
     ARM has a special `CONFIG_ARM_APPENDED_DTB` option that allows to append the `zImage` directly with the `dtb`. This is provided for debugging only, bootloaders are expected to provide DT support.

Device Tree in the ARM code

- Only one “board” file is needed per SoC
- Uses `DT_MACHINE_START` instead of `MACHINE_START`
- Provides a `dt_compat` table to list the platforms compatible with this definition
- `of_platform_populate` will instantiate the devices
- From `arch/arm/mach-tegra/board-dt-tegra20.c`:

```
static void __init tegra_dt_init(void)
{
        [...]
        of_platform_populate(NULL, tegra_dt_match_table,
                                tegra20_auxdata_lookup, NULL);
}

static const char *tegra20_dt_board_compat[] = {
        "nvidia,tegra20",
        NULL
};

DT_MACHINE_START(TEGRA_DT, "nVidia Tegra20 (Flattened Device Tree)")
        .init_machine   = tegra_dt_init,
        .dt_compat      = tegra20_dt_board_compat,
        [...]
MACHINE_END
```

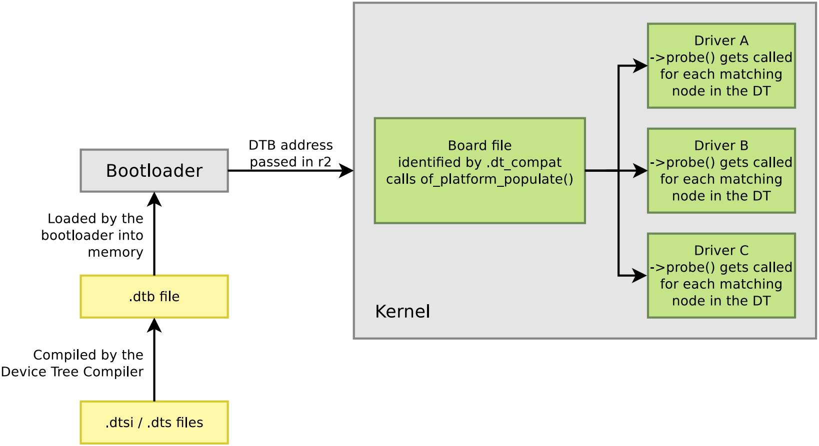
Device tree in drivers: i2c-tegra.c

```
static const struct of_device_id tegra_i2c_of_match[] __devinitconst = {
        { .compatible = "nvidia,tegra20-i2c", },
        { .compatible = "nvidia,tegra20-i2c-dvc", },
        {},
};
MODULE_DEVICE_TABLE(of, tegra_i2c_of_match);

static struct platform_driver tegra_i2c_driver = {
        .probe   = tegra_i2c_probe,
        .remove  = __devexit_p(tegra_i2c_remove),
        .driver  = {
                .name  = "tegra-i2c",
                .owner = THIS_MODULE,
                .of_match_table = tegra_i2c_of_match,
        },
};

static int __init tegra_i2c_init_driver(void)
{
        return platform_driver_register(&tegra_i2c_driver);
}

static void __exit tegra_i2c_exit_driver(void)
{
        platform_driver_unregister(&tegra_i2c_driver);
}

subsys_initcall(tegra_i2c_init_driver);
module_exit(tegra_i2c_exit_driver);
```

Device Tree binding

- With the Device Tree, each driver defines:
  - its *compatible* string, which uniquely identifies the driver, and allows devices to be bound to the corresponding driver
  - the properties of each device in the device tree
- This definition is called a **device tree binding**
- All *device tree bindings* are normally documented in `Documentation/devicetree/bindings`.

Device Tree: summary



Clocks in SoCs

- The different hardware parts of an SoC are driven by different clocks, operating at different frequencies
- Most of those clocks are part of a complex clock tree, where parents clocks are inputs to children clocks
- Many of those clocks are software configurable (on/off, multiple frequencies, etc.) and must be manipulated at runtime for power management reasons
- Due to the parent/child relationship, one must ensure that
  1. The parent clock is enabled when a child clock needs to be enabled
  2. The parent clock is disabled once all children have been disabled

The old clock management infrastructure

- Clocks need to be manipulated by *device drivers*: they know when to enable/disable the needed clocks
- Clocks are listed and controlled by *SoC code*
- Since some time, a common clock API has been defined in `<linux/clk.h>`, defining
  1. An opaque `struct clk` structure, that drivers could manipulate
  2. A simple `clk_get`, `clk_put`, `clk_enable`, `clk_disable`, `clk_get_rate` API
- Each ARM sub-architecture had to have its own definition of `struct clk` and its own implementation of the API
  - A lot of code duplication
  - No common facilities, even though most clocks are relatively similar between SoC

The new clock framework

- A proper *clock framework* has been added in kernel 3.4, released in May 2012
- Initially from Jeremy Kerr (Canonical), finally implemented and merged by Mike Turquette (Texas Instruments)
- This framework:
  - Implements the `clk_get`, `clk_put`, `clk_prepare`, `clk_unprepare`, `clk_enable`, `clk_disable`, `clk_get_rate`, etc. **API for usage by device drivers**
  - Provides data structures (`struct clk_hw` and `struct clkops`) for **SoC code to define its clocks**, and a `clk_register`, `clk_unregister` API to register them, and `clk_register_clkdevs` to associate clocks to device names
  - Implements **some basic clock types** (fixed rate, gatable, divider, fixed factor, etc.)
  - Provides a *debugfs* representation of the clock tree
  - Is implemented in `drivers/clk`

Clock framework, the driver side

From `drivers/serial/tty/mxs-auart.c`, the UART driver for i.MX23/28 SoCs.

```
static int mxs_auart_startup(struct uart_port *u)
{
        [...]
        clk_prepare_enable(s->clk);
        [...]
}

static void mxs_auart_shutdown(struct uart_port *u)
{
         [...]
         clk_disable_unprepare(s->clk);
}

static int __devinit mxs_auart_probe(struct platform_device *pdev)
{
        [...]
        s->clk = clk_get(&pdev->dev, NULL);
        [...]
        s->port.uartclk = clk_get_rate(s->clk);
        [...]
}

static int __devexit mxs_auart_remove(struct platform_device *pdev)
{
        [...]
        clk_put(s->clk);
        [...]
}
```

Clock framework, SoC side

From drivers/clk/mxs/clk-imx28.c

```
static struct clk_lookup uart_lookups[] __initdata = {
        { .dev_id = "duart", },
        { .dev_id = "mxs-auart.0", },
        [...]
        { .dev_id = "8006a000.serial", },
        [...]
};

static struct clk *clks[clk_max];

int __init mx28_clocks_init(void)
{
        [...]
        clks[ref_xtal] = mxs_clk_fixed("ref_xtal", 24000000);
        clks[pll0] = mxs_clk_pll("pll0", "ref_xtal", PLL0CTRL0, 17, 480000000);
        [...]
        clks[ref_cpu] = mxs_clk_ref("ref_cpu", "pll0", FRAC0, 0);
        clks[ref_emi] = mxs_clk_ref("ref_emi", "pll0", FRAC0, 1);
        [...]
        clks[gpmi_sel] = mxs_clk_mux("gpmi_sel", CLKSEQ, 2, 1, sel_gpmi, ARRAY_SIZE(sel_gpmi));
        clks[ssp0_sel] = mxs_clk_mux("ssp0_sel", CLKSEQ, 3, 1, sel_io0, ARRAY_SIZE(sel_io0));
        clks[ssp1_sel] = mxs_clk_mux("ssp1_sel", CLKSEQ, 4, 1, sel_io0, ARRAY_SIZE(sel_io0));
        clks[ssp2_sel] = mxs_clk_mux("ssp2_sel", CLKSEQ, 5, 1, sel_io1, ARRAY_SIZE(sel_io1));
        [...]
        clks[uart] = mxs_clk_gate("uart", "ref_xtal", XTAL, 31);
        [...]
        clk_register_clkdevs(clks[uart], uart_lookups, ARRAY_SIZE(uart_lookups));
        [...]
}
```

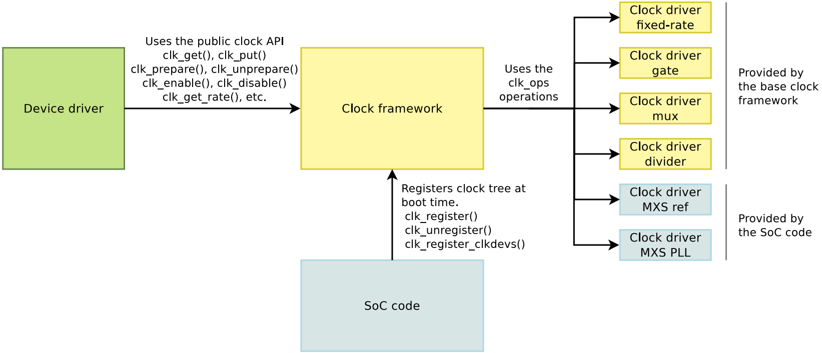
Clock framework, SoC side

- `mxs_clk_fixed()`
  Registers a fixed-rate clock, using the `clk-fixed` clock type provided by the base clock framework in `drivers/clk/clk-fixed.c`
- `mxs_clk_ref()`
  Registers a reference clock, using the `clk-ref` clock type specific to i.MX, implemented in `drivers/clk/mxs/clk-ref.c`
- `mxs_clk_pll()`
  Registers a PLL clock, using the `clk-pll` clock type specific to i.MX, implemented in `drivers/clk/mxs/clk-pll.c`
- `mxs_clk_mux()`
  Registers a muxed clock, using the `clk-mux` clock type provided by the base clock framework in `drivers/clk/clk-mux.c`

Clock tree in *debugfs*

```
# cd /sys/kernel/debug/clk
# find
./ref_xtal
./ref_xtal/pll0
./ref_xtal/pll0/ref_io1
./ref_xtal/pll0/ref_io1/ssp2_sel
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2/clk_notifier_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2/clk_enable_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2/clk_prepare_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2/clk_flags
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/ssp2/clk_rate
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/clk_notifier_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/clk_enable_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/clk_prepare_count
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/clk_flags
./ref_xtal/pll0/ref_io1/ssp2_sel/ssp2_div/clk_rate
[...]
```

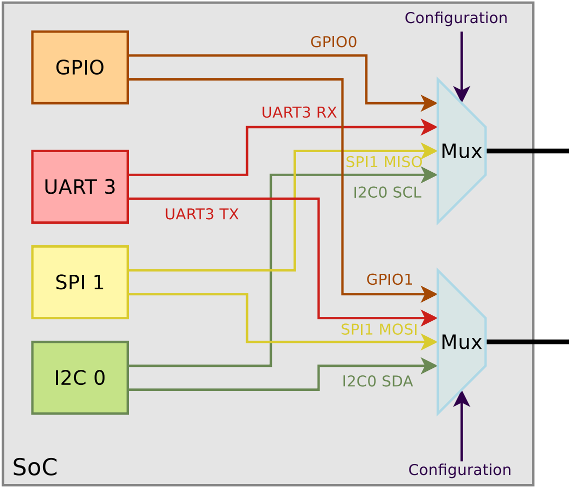
Clock framework: summary



Introduction to pin muxing

- SoCs integrate many more peripherals than the number of available pins allows to expose.
- Many of those pins are therefore **multiplexed**: they can either be used as function A, *or* function B, *or* function C, *or* a GPIO
- Example of functions are:
  - parallel LCD lines
  - SDA/SCL lines for I2C busses
  - MISO/MOSI/CLK lines for SPI
  - RX/TX/CTS/DTS lines for UARTs
- This muxing is **software-configurable**, and depends on **how the SoC is used on each particular board**

Pin muxing: principle



Pin muxing: example

8.4.2 PIO Controller B Multiplexing

Table 8-3. Multiplexing on PIO Controller B (PIOB)

I/O Line	Peripheral A	Peripheral B	Reset State	Power Supply	Function	Comments
PB0	SPI0_MISO		I/O	VDDIOP0		
PB1	SPI0_MOSI		I/O	VDDIOP0		
PB2	SPI0_SPCK		I/O	VDDIOP0		
PB3	SPI0_NPCS0		I/O	VDDIOP0		
PB4	TXD1		I/O	VDDIOP0		
PB5	RXD1		I/O	VDDIOP0		
PB6	TXD2		I/O	VDDIOP0		
PB7	RXD2		I/O	VDDIOP0		
PB8	TXD3	ISI_D8	I/O	VDDIOP2		
PB9	RXD3	ISI_D9	I/O	VDDIOP2		
PB10	TWD1	ISI_D10	I/O	VDDIOP2		
PB11	TWCK1	ISI_D11	I/O	VDDIOP2		
PB12	DRXD		I/O	VDDIOP0		
PB13	DTXD		I/O	VDDIOP0		

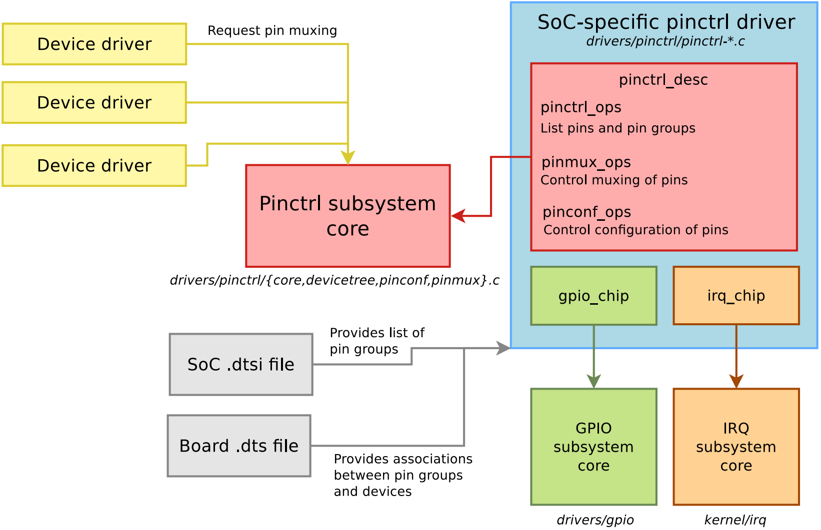
The old pin-muxing code

- Each ARM sub-architecture had its own pin-muxing code
- The API was specific to each sub-architecture
- Lot of similar functionality implemented in different ways
- The pin-muxing had to be done at the SoC level, and couldn't be requested by device drivers

The new pin-muxing subsystem

- The new **pinctrl** subsystem aims at solving those problems
- Mainly developed and maintained by Linus Walleij, from Linaro/ST-Ericsson
- Implemented in `drivers/pinctrl`
- Provides:
  - An API to register *pinctrl driver*, i.e entities knowing the list of pins, their functions, and how to configure them. Used by SoC-specific drivers to expose pin-muxing capabilities.
  - An API for *device drivers* to request the muxing of a certain set of pins
  - An interaction with the *GPIO* framework

The new pin-muxing subsystem: diagram



Declaring pin groups in the SoC dtsi

- From arch/arm/boot/dts/imx28.dtsi
- Declares the *pinctrl* device and various pin groups

```
pinctrl@80018000 {
   #address-cells = <1>;
   #size-cells = <0>;
   compatible = "fsl,imx28-pinctrl", "simple-bus";
   reg = <0x80018000 2000>;

   duart_pins_a: duart@0 {
           reg = <0>;
           fsl,pinmux-ids = <0x3102 0x3112>;
           fsl,drive-strength = <0>;
           fsl,voltage = <1>;
           fsl,pull-up = <0>;
   };
   duart_pins_b: duart@1 {
           reg = <1>;
           fsl,pinmux-ids = <0x3022 0x3032>;
           fsl,drive-strength = <0>;
           fsl,voltage = <1>;
           fsl,pull-up = <0>;
   };
```

```
   mmc0_8bit_pins_a: mmc0-8bit@0 {
           reg = <0>;
           fsl,pinmux-ids = <0x2000 0x2010 0x2020
                   0x2030 0x2040 0x2050 0x2060
                   0x2070 0x2080 0x2090 0x20a0>;
           fsl,drive-strength = <1>;
           fsl,voltage = <1>;
           fsl,pull-up = <1>;
   };
   mmc0_4bit_pins_a: mmc0-4bit@0 {
           reg = <0>;
           fsl,pinmux-ids = <0x2000 0x2010 0x2020
                   0x2030 0x2080 0x2090 0x20a0>;
           fsl,drive-strength = <1>;
           fsl,voltage = <1>;
           fsl,pull-up = <1>;
   };
   mmc0_cd_cfg: mmc0-cd-cfg {
           fsl,pinmux-ids = <0x2090>;
           fsl,pull-up = <0>;
   };
   mmc0_sck_cfg: mmc0-sck-cfg {
           fsl,pinmux-ids = <0x20a0>;
           fsl,drive-strength = <2>;
           fsl,pull-up = <0>;
   };
};
```

Associating devices with pin groups, board dts

- From arch/arm/boot/dts/cfa10036.dts

```
apb@80000000 {
  apbh@80000000 {
     ssp0: ssp@80010000 {
         compatible = "fsl,imx28-mmc";
         pinctrl-names = "default";
         pinctrl-0 = <&mmc0_4bit_pins_a
                      &mmc0_cd_cfg &mmc0_sck_cfg>;
         bus-width = <4>;
         status = "okay";
     };
  };

  apbx@80040000 {
     duart: serial@80074000 {
         pinctrl-names = "default";
         pinctrl-0 = <&duart_pins_b>;
         status = "okay";
     };
  };
};
```

Device drivers requesting pin muxing

- From drivers/mmc/host/mxs-mmc.c

```
static int mxs_mmc_probe(struct platform_device *pdev)
{
    [...]
    pinctrl = devm_pinctrl_get_select_default(&pdev->dev);
    if (IS_ERR(pinctrl)) {
        ret = PTR_ERR(pinctrl);
        goto out_mmc_free;
    }
    [...]
}
```

References

- About ARM maintenance
  - *ARM Subarchitecture Status*, Arnd Bergmann, ELC 2012
- About the Device Tree
  - Official Wiki, `http://devicetree.org`
  - *Experiences With Device Tree Support Development For ARM-Based SOC's*, Thomas P. Abraham, ELC 2012
  - *Device Tree Status Report*, Grant Likely, ELCE 2011.
- About the clock framework
  - `Documentation/clk.txt` in the kernel sources
  - `http://lwn.net/Articles/472998/`
  - *Common Clock Framework*, Mike Turquette, ELC2012
- About the pinctrl subsystem
  - `Documentation/pinctrl.txt` in the kernel sources
  - `http://lwn.net/Articles/468759/`
  - *Pin Control Subsystem Overview*, Linus Walleij, ELC2012
- Slides and video at
  - `https://bootlin.com/blog/elc-2012-videos/` for Embedded Linux Conference 2012
  - `https://bootlin.com/blog/elce-2011-videos/` for Embedded Linux Conference Europe 2011

Conclusion

- The *pinctrl* and *clk* subsystems now provide **generic abstractions** to manage pin muxing and the clocks of an SoC
- The *device tree* provides a **better way of representing the hardware**, requiring less code to describe new platforms
- The usage of these new infrastructures is **mandatory for new platforms**
- **Conversion of existing platforms** that are widely used is in process
- The ARM community has gained better code infrastructures, a better organization, and has become **even more dynamic than it was**.

Questions?

Thomas Petazzoni

thomas.petazzoni@bootlin.com

PDF and sources will be available on https://bootlin.com/pub/conferences/2012/lsm/