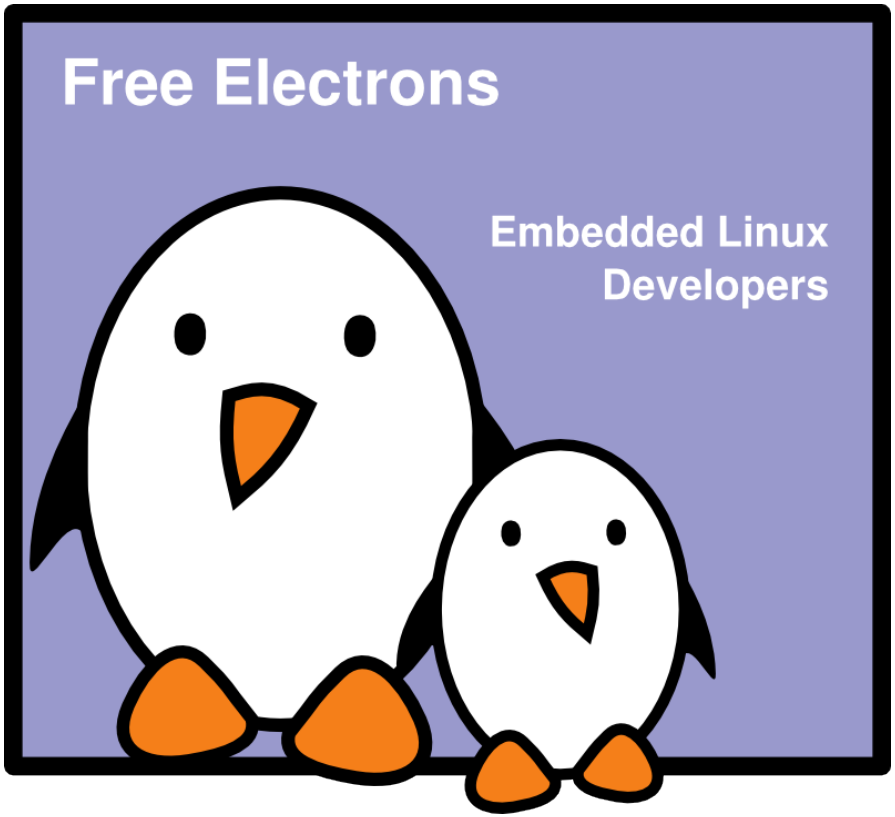
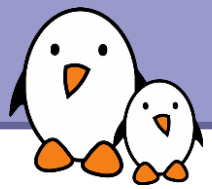
Porting the Linux kernel to an ARM board

Thomas Petazzoni
Free Electrons

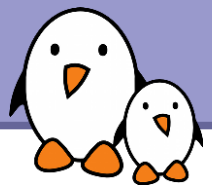


Latest update: Feb 17, 2010,
Document sources, updates and translations:
http://free-electrons.com/docs/kernel-porting
Corrections, suggestions, contributions and translations are welcome!



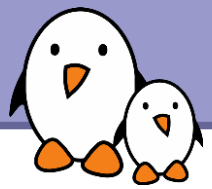
Porting the Linux kernel

- The Linux kernel supports a lot of different CPU architectures
- Each of them is maintained by a different group of contributors
  - See the `MAINTAINERS` file for details
- The organization of the source code and the methods to port the Linux kernel to a new board are therefore very architecture-dependent
- For example, PowerPC and ARM are very different
  - PowerPC relies on device trees to describe hardware details
  - ARM relies on source code only
- This presentation is focused on the ARM architecture only



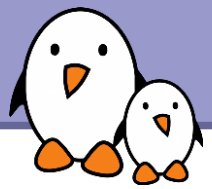
Architecture, CPU and machine

- In the source tree, each architecture has its own directory `arch/arm` for the ARM architecture
- This directory contains generic ARM code
  - `boot, common, configs, kernel, lib, mm, nwfpe, vfp, oprofile, tools`
- And many directories for different CPU families
  - `mach-*` directories : `mach-pxa` for PXA CPUs, `mach-imx` for Freescale iMX CPUs, etc.
  - Each of these directories contain
    - Support for the CPU
    - Support for several boards using this CPU
- Some CPU types share some code, in an entity called a *platform*
  - `plat-omap` contains common code from `mach-omap1` and `mach-omap2`



Source code for Calao USB A9263

- Taking the case of the Calao USB A9263 board, which uses a AT91SAM9263 CPU.
- `arch/`
  - `arm/`
    - `mach-at91/`
      - AT91 generic code
        `clock.c, leds.c, irq.c, pm.c`
      - CPU-specific code for the AT91SAM9263
        `at91sam9263.c, at91sam926x_time.c, at91sam9263_devices.c`
      - Board specific code
        `board-usb-a9263.c`
- For the rest of this presentation, we will focus on board support only

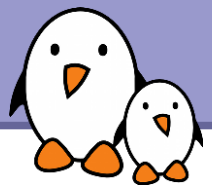


Configuration

- A configuration option must be defined for the board, in `arch/arm/mach-at91/Kconfig`

```
config MACH_USB_A9263
        bool "CALAO USB-A9263"
        depends on ARCH_AT91SAM9263
        help
          Select this if you are using a Calao Systems USB-A9263.
          <http://www.calao-systems.com>
```

- This option must depend on the CPU type option corresponding to the CPU used in the board
  - Here the option is `ARCH_AT91SAM9263`, defined in the same file
- A default configuration file for the board can optionally be stored in `arch/arm/configs/`. For our board, it's `usb-a9263_defconfig`



Compilation

- The source files corresponding to the board support must be associated with the configuration option of the board
- This is done in `arch/arm/mach-at91/Makefile`

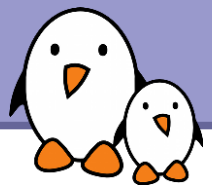
```
obj-$(CONFIG_MACH_USB_A9263)    += board-usb-a9263.o
```

- The Makefile also tells which files are compiled for every AT91 CPU

```
obj-y                := irq.o gpio.o
obj-$(CONFIG_AT91_PMC_UNIT)     += clock.o
obj-y                           += leds.o
obj-$(CONFIG_PM)                += pm.o
obj-$(CONFIG_AT91_SLOW_CLOCK)   += pm_slowclock.o
```

- And which files for our particular CPU, the AT91SAM9263

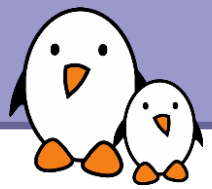
```
obj-$(CONFIG_ARCH_AT91SAM9263)  += at91sam9263.o at91sam926x_time.o
at91sam9263_devices.o sam9_smc.o
```



Machine structure

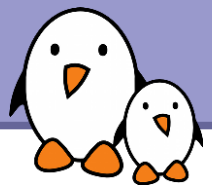
- Each board is defined by a machine structure
  - The word « machine » is quite confusing since every `mach-*` directory contains several machine definitions, one for each board using a given CPU type
- For the Calao board, at the end of `arch/arm/mach-at91/board-usb-a9263.c`

```
MACHINE_START(USB_A9263, "CALAO USB_A9263")
        /* Maintainer: calao-systems */
        .phys_io        = AT91_BASE_SYS,
        .io_pg_offst    = (AT91_VA_BASE_SYS >> 18) & 0xfffc,
        .boot_params    = AT91_SDRAM_BASE + 0x100,
        .timer          = &at91sam926x_timer,
        .map_io         = ek_map_io,
        .init_irq       = ek_init_irq,
        .init_machine   = ek_board_init,
MACHINE_END
```



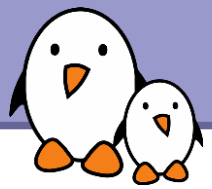
Machine structure macros

- `MACHINE_START` and `MACHINE_END`
  - Macros defined in `arch/arm/include/asm/mach/arch.h`
  - They are helpers to define a `struct machine_desc` structure stored in a specific ELF section
  - Several `machine_desc` structures can be defined in a kernel, which means that the kernel can support several boards.
  - The right structure is chosen at boot time



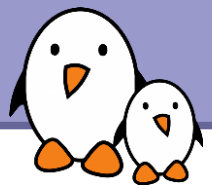
Machine type number

- In the ARM architecture, each board type is identified by a machine type number
- The latest machine type numbers list can be found at http://www.arm.linux.org.uk/developer/machines/download.php
- A copy of it exists in the kernel tree in `arch/arm/tools/mach-types`
  - For the Calao board
    ```
    usb_a9263  MACH_USB_A9263 USB_A9263 1710
    ```
- At compile time, this file is processed to generate a header file, `include/asm-arm/mach-types.h`
  - For the Calao board
    ```
    #define MACH_TYPE_USB_A9263 1710
    ```
  - And a few other macros in the same file



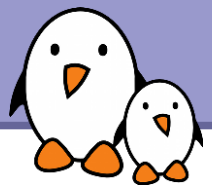
Machine type number

- The machine type number is set in the `MACHINE_START()` definition
  `MACHINE_START(USB_A9263, "CALAO USB_A9263")`
- At run time, the machine type number of the board on which the kernel is running is passed by the bootloader in register *r1*
- Very early in the boot process (`arch/arm/kernel/head.S`), the kernel calls `__lookup_machine_type` in `arch/arm/kernel/head-common.S`
- `__lookup_machine_type` looks at all the `machine_desc` structures of the special ELF section
  - If it doesn't find the requested number, prints a message and stops
  - If found, it knows the machine descriptions and continues the boot process



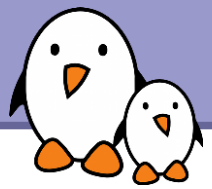
Early debugging and boot parameters

- Early debugging
  - `phys_io` is the physical address of the I/O space
  - `io_pg_offset` is the offset in the page table to remap the I/O space
  - These are used when `CONFIG_DEBUG_LL` is enabled to provide very early debugging messages on the serial port
- Boot parameters
  - `boot_params` is the location where the bootloader has left the boot parameters (the kernel command line)
  - The bootloader can override this address in register *r2*
  - See also `Documentation/arm/Booting` for the details of the environment expected by the kernel when booted



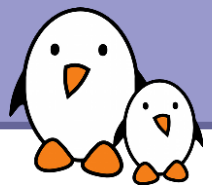
System timer

- The timer field point to a `struct sys_timer` structure, that describes the system timer
  - Used to generate the periodic tick at HZ frequency to call the scheduler periodically
- On the Calao board, the system timer is defined by the `at91sam926x_timer` structure in `at91sam926x_time.c`
- It contains the interrupt handler called at HZ frequency
- It is integrated with the `clockevents` and the `clocksource` infrastructures
  - See `include/linux/clocksource.h` and `include/linux/clockchips.h` for details



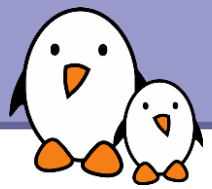
map_io()

- The `map_io()` function points to `ek_map_io()`, which
  - Initializes the CPU using `at91sam9263_initialize()`
    - Map I/O space
    - Register and initialize the clocks
  - Configures the debug serial port and set the console to be on this serial port
  - Called at the very beginning of the C code execution
    - `init/main.c: start_kernel()`
    - `arch/arm/kernel/setup.c: setup_arch()`
    - `arch/arm/mm/mmu.c: paging_init()`
    - `arch/arm/mm/mmu.c: devicemaps_init()`
      - `mdesc->map_io()`



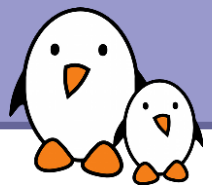
init_irq()

- `init_irq()` to initialize the IRQ hardware specific details
- Implemented by `ek_init_irq()`, which calls `at91sam9263_init_interrupts()` in `at91sam9263.c`, which mainly calls `at91_aic_init()` in `irq.c`
  - Initialize the interrupt controller, assign the priorities
  - Register the IRQ chip (`irq_chip` structure) to the kernel generic IRQ infrastructure, so that the kernel knows how to ack, mask, unmask the IRQs
- Called a little bit later than `map_io()`
  - `init/main.c: start_kernel()`
  - `arch/arm/kernel/irq.c: init_IRQ()`
  - `init_arch_irq()` (equal to `mdesc->init_irq`)



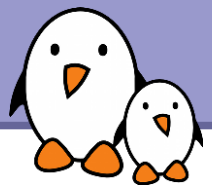
init_machine()

- `init_machine()` completes the initialization of the board by registering all platform devices
- Called by `customize_machines()` in `arch/arm/kernel/setup.c`
- This function is an `arch_initcall` (list of functions whose address is stored in a specific ELF section, by levels)
- At the end of kernel initialization, just before running the first userspace program init:
  - `init/main.c: kernel_init()`
  - `init/main.c: do_basic_setup()`
  - `init/main.c: do_initcalls()`
  - Calls all initcalls, level by level



init_machine() for Calao

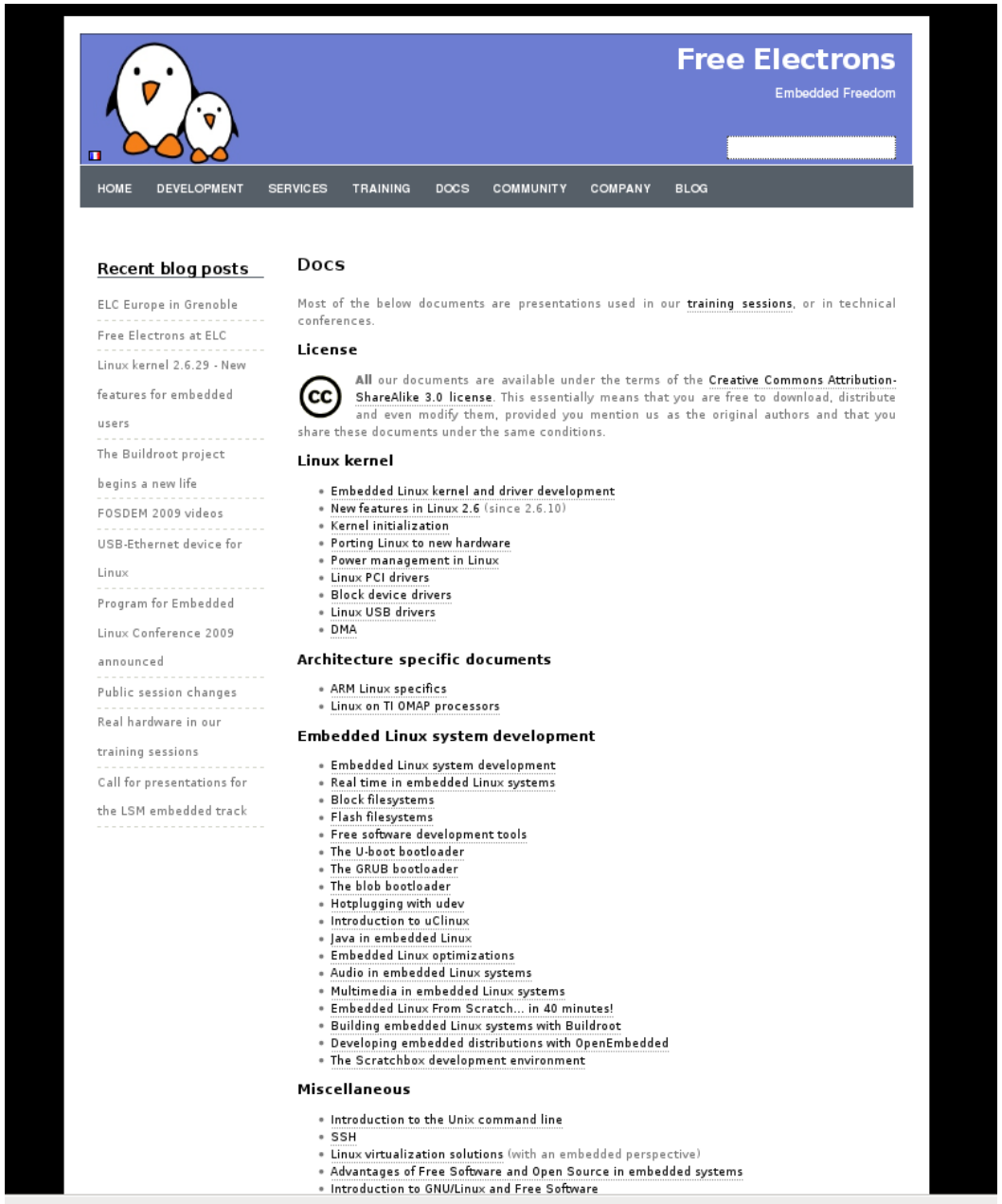
- For the Calao board, implement in `ek_board_init()`
  - Registers serial ports, USB host, USB device, SPI, Ethernet, NAND flash, 2IC, buttons and LEDs
  - Uses `at91_add_device_*()` helpers, defined in `at91sam9263_devices.c`
  - These helpers call `platform_device_register()` to register the different `platform_device` structures defined in the same file
  - For some devices, the board specific code does the registration itself (buttons) or passes board-specific data to the registration helper (USB host and device, NAND, Ethernet, etc.)



Drivers

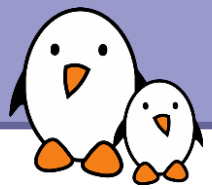
- The `at91sam9263_devices.c` file doesn't implement the drivers for the platform devices
- The drivers are implemented at different places of the kernel tree
- For the Calao board
  - USB host, driver `at91_ohci`, `drivers/usb/host/ohci-at91.c`
  - USB device, driver `at91_udc`, `drivers/usb/gadget/at91_udc.c`
  - Ethernet, driver `macb`, `drivers/net/macb.c`
  - NAND, driver `atmel_nand`, `drivers/mtd/nand/atmel_nand.c`
  - I2C on GPIO, driver `i2c-gpio`, `drivers/i2c/busses/i2c-gpio.c`
  - SPI, driver `atmel_spi`, `drivers/spi/atmel_spi.c`
  - Buttons, driver `gpio-keys`, `drivers/input/keyboard/gpio_keys.c`
- All these drivers are selected by the ready-made configuration file

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs