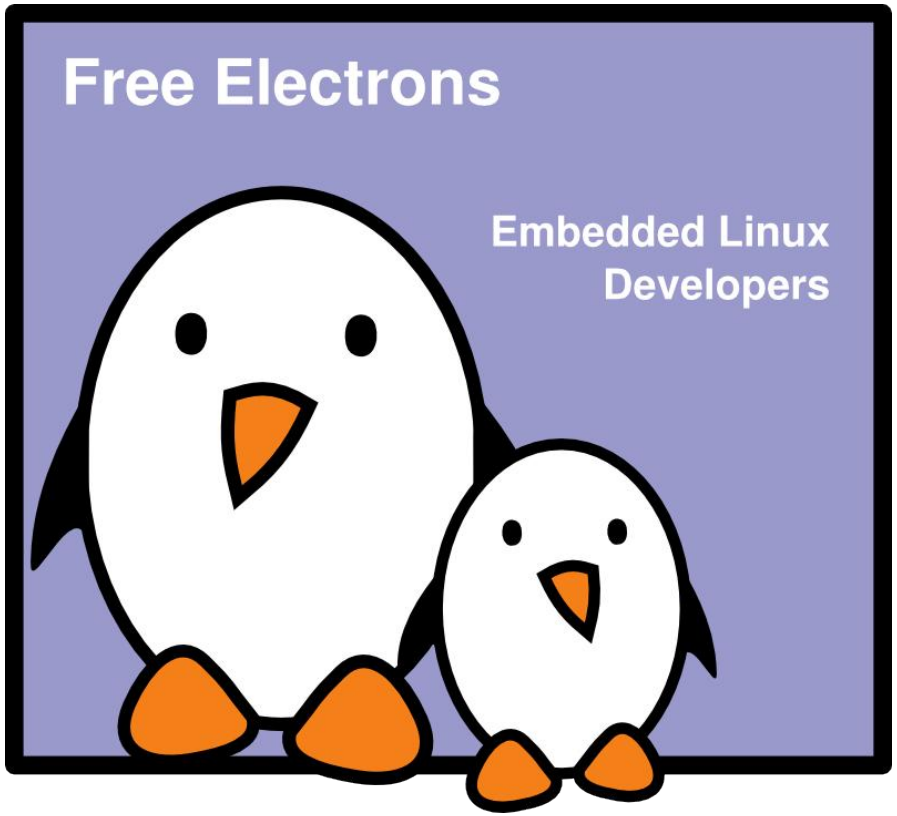
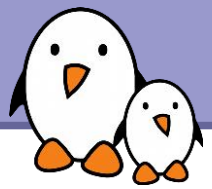
Power Management

Michael Opdenacker
Thomas Petazzoni
Free Electrons



Linux

Latest update: Feb 21, 2011,
Document sources, updates and translations:
http://free-electrons.com/docs/power
Corrections, suggestions, contributions and translations are welcome!

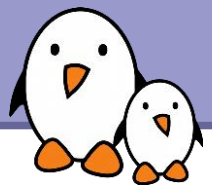


PM building blocks

Several power management « building blocks »

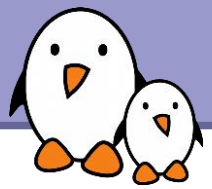
- Suspend and resume
- CPUidle
- Runtime power management
- Frequency and voltage scaling
- Applications

Independent « building blocks » that can be improved gradually during development



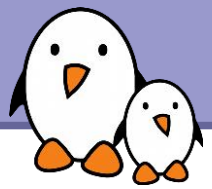
Clock framework (1)

- Generic framework to manage clocks used by devices in the system
- Allows to reference count clock users
  and to shutdown the unused clocks to save power
- Simple API described in
  http://free-electrons.com/kerneldoc/latest/DocBook/kernel-api/clk.html
  - `clk_get()` to get a reference to a clock
  - `clk_enable()` to start the clock
  - `clk_disable()` to stop the clock
  - `clk_put()` to free the clock source
  - `clk_get_rate()` to get the current rate



Clock framework (2)

- The clock framework API and the `clk` structure are usually implemented by each architecture (code duplication!)
  - See `arch/arm/mach-at91/clock.c` for an example
    This is also where all clocks are defined.
  - Clocks are identified by a name string specific to a given platform
- Drivers can then use the clock API.
  Example from `drivers/net/macb.c`:
  - `clk_get()` called from the `probe()` function, to get the definition of a clock for the current board, get its frequency, and run `clk_enable()`.
  - `clk_put()` called from the `remove()` function to release the reference to the clock, after calling `clk_disable()`.



Clock disable implementation

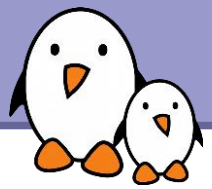
From arch/arm/mach-at91/clock.c: (2.6.36)

```
static void __clk_disable(struct clk *clk)
{
        BUG_ON(clk->users == 0);
        if (--clk->users == 0 && clk->mode)
                clk->mode(clk, 0);
        if (clk->parent)
                __clk_disable(clk->parent);
}
```

Call the hardware function switching off this clock

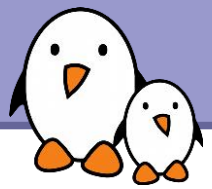
Example `mode` function (same file):

```
static void pmc_sys_mode(struct clk *clk, int is_on)
{
        if (is_on)
                at91_sys_write(AT91_PMC_SCER, clk->pmc_mask);
        else
                at91_sys_write(AT91_PMC_SCDR, clk->pmc_mask);
}
```



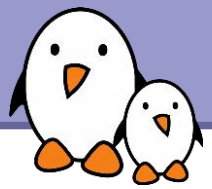
Suspend and resume

- Infrastructure in the kernel to support suspend and resume
- Platform hooks
  - `prepare()`, `enter()`, `finish()`, `valid()` in a `platform_suspend_ops` structure
  - Registered using the `suspend_set_ops()` function
  - See `arch/arm/mach-at91/pm.c`
- Device drivers
  - `suspend()` and `resume()` hooks in the `*_driver` structures (`platform_driver`, `usb_driver`, etc.)
  - See `drivers/net/macb.c`



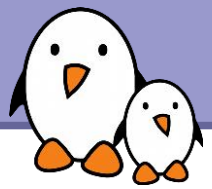
Board specific power management

- Typically takes care of battery and charging management.
- Also defines presuspend and postsuspend handlers.
- Example:
  `arch/arm/mach-pxa/spitz_pm.c`



arch/arm/mach-<cpu>/sleep.S

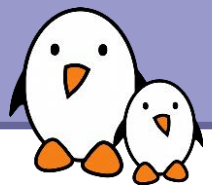
- Assembly code implementing CPU specific suspend and resume code. Note: only found on arm, just 3 other occurrences in other architectures, with other paths.
- First scenario: only a suspend function. The code goes in sleep state (after enabling DRAM self-refresh), and continues with resume code.
- Second scenario: suspend and resume functions. Resume functions called by the bootloader.
- Examples to look at:
  `arch/arm/mach-omap2/sleep24xx.S` (1st case)
  `arch/arm/mach-pxa/sleep.S` (2nd case)



Triggering suspend

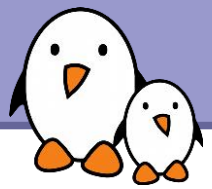
- Whatever the power management implementation, CPU specific `suspend_ops` functions are called by the `enter_state` function.
- `enter_state` also takes care of executing the suspend and resume functions for your devices.
- The execution of this function can be triggered from userspace. To suspend to RAM:
  `echo mem > /sys/power/state`
- Can also use the s2ram program from http://suspend.sourceforge.net/

Read `kernel/power/suspend.c`



Runtime power management

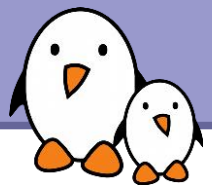
- According to the kernel configuration interface:
  Enable functionality allowing I/O devices to be put into energy-saving (low power) states at run time (or autosuspended) after a specified period of inactivity and woken up in response to a hardware-generated wake-up event or a driver's request.
- New hooks must be added to the drivers:
  `runtime_suspend()`, `runtime_resume()`, `runtime_idle()`
- API and details on
  Documentation/power/runtime_pm.txt
- See also Kevin Hilman's presentation at ELC Europe 2010:
  http://elinux.org/images/c/cd/ELC-2010-khilman-Runtime-PM.odp



Saving power in the idle loop

- The idle loop is what you run
  when there's nothing left to run in the system.
- Implemented in all architectures
  in `arch/<arch>/kernel/process.c`
- Example to read: look for `cpu_idle` in
  `arch/arm/kernel/process.c`
- Each ARM cpu defines its own `arch_idle` function.
- The CPU can run power saving HLT instructions, enter NAP mode, and even disable the timers (tickless systems).

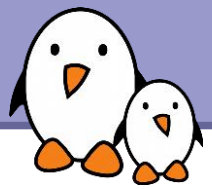
See also http://en.wikipedia.org/wiki/Idle_loop



Managing idle

Adding support for multiple idle levels

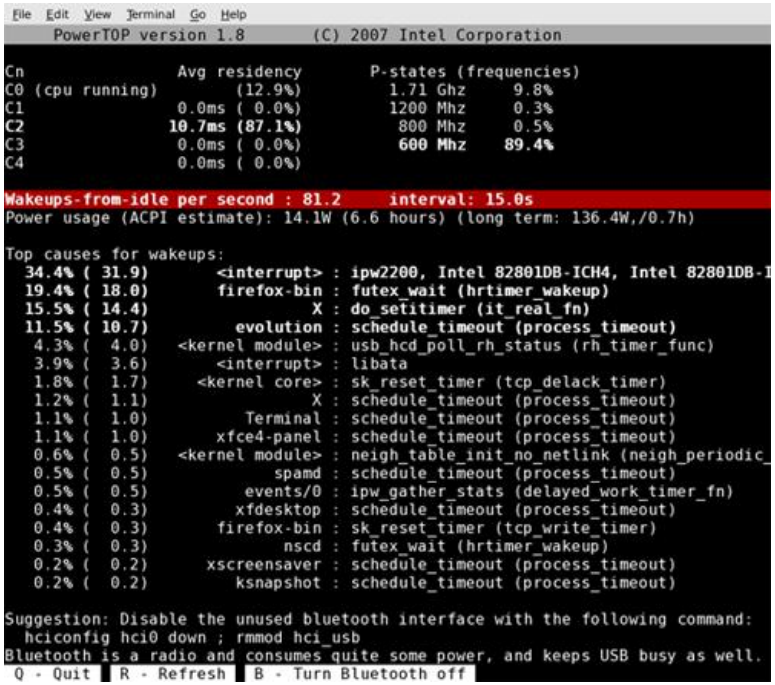
- Modern CPUs have several sleep states offering different power savings with associated wake up latencies
- Since 2.6.21, the dynamic tick feature allows to remove the periodic tick to save power, and to know when the next event is scheduled, for smarter sleeps.
- CPUidle infrastructure to change sleep states
  - Platform-specific driver defining sleep states and transition operations
  - Platform-independent governors (ladder and menu)
  - Available for x86/ACPI, not supported yet by all ARM cpus. (look for `cpuidle*` files under `arch/arm/`)
  - See `Documentation/cpuidle/` in kernel sources.

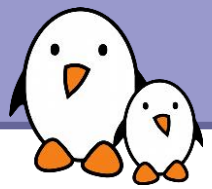


PowerTOP

http://www.lesswatts.org/projects/powertop/

- With dynamic ticks, allows to fix parts of kernel code and applications that wake up the system too often.
- PowerTOP allows to track the worst offenders
- Now available on ARM cpus implementing CPUidle
- Also gives you useful hints for reducing power.

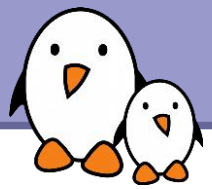




Frequency and voltage scaling (1)

Frequency and voltage scaling possible through the *cpufreq* kernel infrastructure.

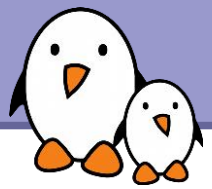
- Generic infrastructure
  `drivers/cpufreq/cpufreq.c`
  `include/linux/cpufreq.h`
- Generic governors, responsible for deciding frequency and voltage transitions
  - `performance`: maximum frequency
  - `powersave`: minimum frequency
  - `ondemand`: measures CPU consumption to adjust frequency
  - `conservative`: often better than `ondemand`.
    Only increases frequency gradually when the CPU gets loaded.
  - `userspace`: leaves the decision to an userspace daemon.
- This infrastructure can be controlled from
  `/sys/devices/system/cpu/cpu<n>/cpufreq/`



Frequency and voltage scaling (2)

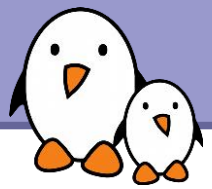
- CPU support code in architecture dependent files.
  Example to read: `arch/arm/plat-omap/cpu-omap.c`
- Must implement the operations of the `cpufreq_driver` structure and register them using `cpufreq_register_driver()`
  - `init()` for initialization
  - `exit()` for cleanup
  - `verify()` to verify the user-chosen policy
  - `setpolicy()` or `target()`
    to actually perform the frequency change

See `Documentation/cpu-freq/` for useful explanations



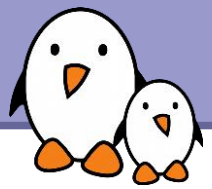
PM QoS

- PM QoS is a framework developed by Intel introduced in 2.6.25
- It allows kernel code and applications to set their requirements in terms of
  - CPU DMA latency
  - Network latency
  - Network throughput
- According to these requirements, PM QoS allows kernel drivers to adjust their power management
- See `Documentation/power/pm_qos_interface.txt` and Mark Gross' presentation at ELC 2008
- Still in very early deployment (only 4 drivers in 2.6.36).



Regulator framework

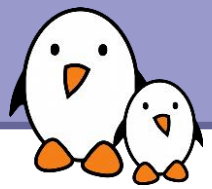
- Modern embedded hardware have hardware responsible for voltage and current regulation
- The regulator framework allows to take advantage of this hardware to save power when parts of the system are unused
  - A consumer interface for device drivers (i.e users)
  - Regulator driver interface for regulator drivers
  - Machine interface for board configuration
  - *sysfs* interface for userspace
- Merged in Linux 2.6.27.
  See `Documentation/power/regulator/` in kernel sources.
- See Liam Girdwood's presentation at ELC 2008
  http://free-electrons.com/blog/elc-2008-report#girdwood



BSP work for a new board

In case you just need to create a BSP for your board, and your CPU already has full PM support, you should just need to:

- Create clock definitions and bind your devices to them.
- Implement PM handlers (suspend, resume) in the drivers for your board specific devices.
- Implement runtime PM handlers in your drivers.
- Implement board specific power management if needed (mainly battery management)
- Implement regulator framework hooks for your board if needed.
- All other parts of the PM infrastructure should be already there: suspend / resume, cpuidle, cpu frequency and voltage scaling.



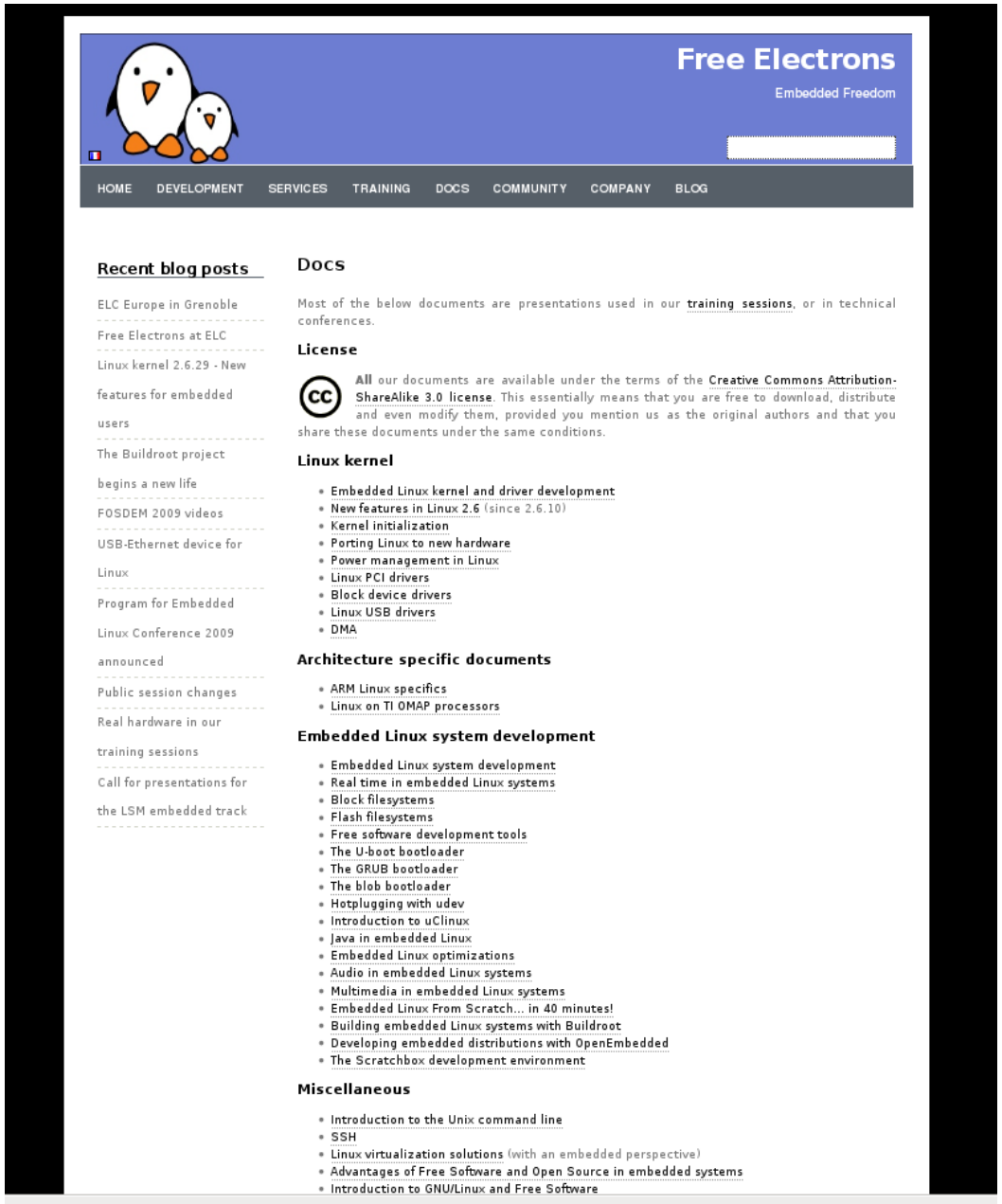
Useful resources

- `Documentation/power/` in the Linux kernel sources.
  Will give you many useful details.
- http://lesswatts.org
  Intel effort trying to create a Linux power saving community.
  Mainly targets Intel processors.
  Lots of useful resources.
- http://wiki.linaro.org/WorkingGroups/PowerManagement/
  Ongoing developments on the ARM platform.
- Tips and ideas for prolonging battery life:
  http://j.mp/fVdxKh

Practical lab – Power management

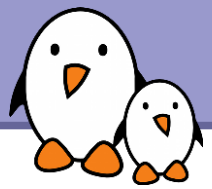
- Suspend and resume your Linux system
- Change the CPU frequency of your system

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