Power Management Using PM Domains on SH7372

Rafael J. Wysocki

Renesas / Faculty of Physics U. Warsaw / SUSE Labs

October 3, 2011

SH7372 Characteristics

- System on a Chip (SoC).
- Two independent CPU cores (ARM and SuperH).
- Possibility to manipulate device clocks (enable/disable).
- Power domains that can be turned on and off through register writes.
- Hierarchy of power domains.
- No BIOS (description through board files in the kernel).

SH7372 Characteristics

- System on a Chip (SoC).
- Two independent CPU cores (ARM and SuperH).
- Possibility to manipulate device clocks (enable/disable).
- Power domains that can be turned on and off through register writes.
- Hierarchy of power domains.
- No BIOS (description through board files in the kernel).

Simple power management model

- Device on (full-power state).
- Clocks off (low-power state, level 1).
- Power removed from power domain (low-power state, level 2).
- Wakeup latency constraints determine the state to choose (run time).

Power Domains On SH7372

- C5: base (mother) domain, CPG, KEYSC, CMT, RWDT, GPIO
- A4LC: LCDC, DSI, MERAM (video)
- A4MP: SPU2, FSI (audio)
- D4: ARM debug
- A4R: SH4AL-DSP, INTCS, DMAC, IIC, TMU, MSIOF, CMT0, CEU, CSI (SH CPU core, I/O)
- A3RI: ISP (camera capture unit)
- A3RV: VPU (video encode/decode unit)
- A4S: INTCA, MFI, SBSC (interrupt and SDRAM controllers)
- A3SG: SGX (3D graphics)
- A3SP: SCIF, MSIOF, IIC, USB, SDHI, MMCIF, HDMI (I/O)
- A3SM: ARM Cortex-A8 CPU core

Hierarchy Of Power Domains

<table>
<tr><td rowspan="9">C5</td><td colspan="2">A4LC</td></tr>
<tr><td colspan="2">A4MP</td></tr>
<tr><td colspan="2">D4</td></tr>
<tr><td rowspan="3">A4R</td><td></td></tr>
<tr><td>A3RI</td></tr>
<tr><td>A3RV</td></tr>
<tr><td rowspan="3">A4S</td><td>A3SG</td></tr>
<tr><td>A3SP</td></tr>
<tr><td>A3SM</td></tr>
</table>

Hierarchy Of Power Domains

<table>
<tr><td rowspan="9">C5</td><td colspan="2">A4LC</td></tr>
<tr><td colspan="2">A4MP</td></tr>
<tr><td colspan="2">D4</td></tr>
<tr><td rowspan="3">A4R</td><td></td></tr>
<tr><td>A3RI</td></tr>
<tr><td>A3RV</td></tr>
<tr><td rowspan="3">A4S</td><td>A3SG</td></tr>
<tr><td>A3SP</td></tr>
<tr><td>A3SM</td></tr>
</table>

It turns out that A3RV depends on A4LC (because of MERAM).

Consequences Of The Design

- Every device is a direct member of one power domain.
- One power domain may have multiple masters (e.g. A3RV).
- It is desirable to turn off A4R when A3RI and A3RV are off.
- It is desirable to turn off A4S when A3SG, A3SP, A3SM are off.

Consequences Of The Design

- Every device is a direct member of one power domain.
- One power domain may have multiple masters (e.g. A3RV).
- It is desirable to turn off A4R when A3RI and A3RV are off.
- It is desirable to turn off A4S when A3SG, A3SP, A3SM are off.

However

There are devices with cross-domain dependencies (e.g. INTCA, DMAC).

Consequences Of The Design

- Every device is a direct member of one power domain.
- One power domain may have multiple masters (e.g. A3RV).
- It is desirable to turn off A4R when A3RI and A3RV are off.
- It is desirable to turn off A4S when A3SG, A3SP, A3SM are off.

However

There are devices with cross-domain dependencies (e.g. INTCA, DMAC).

The I/O runtime PM framework had to be extended so that the dependencies related to power domains could be handled.

CPUidle

Theory: Put idle CPUs into low-power states (no code execution)

- CPU scheduler knows when a CPU is idle.
- Next usage time information from clock events.
- Maximum acceptable wakeup latency from PM QoS.
- CPU low-power states (C-states) characteristics are known.
- Governors decide what state to go for.

CPUidle

Theory: Put idle CPUs into low-power states (no code execution)

- CPU scheduler knows when a CPU is idle.
- Next usage time information from clock events.
- Maximum acceptable wakeup latency from PM QoS.
- CPU low-power states (C-states) characteristics are known.
- Governors decide what state to go for.

Practice: Not so easy

- Power domain A4S is shared with I/O devices.
- Deep "C-states" may involve removing power from A4S.
- How much time is it going to take to restore power (wakeup)?
- What is the power break even time?

Runtime PM Framework

Turning devices off (when idle) and on

- The core:
  1. Handles concurrency (locking etc.).
  2. Takes care of device dependencies (parents vs children).
  3. Provides reference counting facilities (detection of idleness).
  4. Provides common helpers (e.g. `pm_runtime_suspend()`).
- Subsystems and drivers:
  1. Provide callbacks.
  2. Handle wakeup events (remote wakeup).

Runtime PM Framework

Turning devices off (when idle) and on

- The core:
  1. Handles concurrency (locking etc.).
  2. Takes care of device dependencies (parents vs children).
  3. Provides reference counting facilities (detection of idleness).
  4. Provides common helpers (e.g. `pm_runtime_suspend()`).
- Subsystems and drivers:
  1. Provide callbacks.
  2. Handle wakeup events (remote wakeup).

Power domains have to be taken into account.

Runtime PM Framework

Turning devices off (when idle) and on

- The core:
  1. Handles concurrency (locking etc.).
  2. Takes care of device dependencies (parents vs children).
  3. Provides reference counting facilities (detection of idleness).
  4. Provides common helpers (e.g. `pm_runtime_suspend()`).
- Subsystems and drivers:
  1. Provide callbacks.
  2. Handle wakeup events (remote wakeup).

Power domains have to be taken into account.

Subsystem callbacks may be overriden by power management domain callbacks (representation via `struct dev_pm_domain`).

Generic PM Domains

Simple framework for representing power domains

- Stop/start operations for member devices (may correspond to enabling/disabling clocks).
- Save state/restore state operations for member devices (represented by drivers' `.runtime_suspend()` and `.runtime_resume()` callbacks).
- Power off/power on operations for entire domains.
- Domain hierarchies.

Generic PM Domains

Simple framework for representing power domains

- Stop/start operations for member devices (may correspond to enabling/disabling clocks).
- Save state/restore state operations for member devices (represented by drivers' `.runtime_suspend()` and `.runtime_resume()` callbacks).
- Power off/power on operations for entire domains.
- Domain hierarchies.

Representation via `struct generic_pm_domain` (new in the 3.1-rc kernels, improvements on their way into the 3.2 kernel).

Device Suspend

1. Device usage counter is 0, the PM core runs `rpm_suspend(dev)`.
2. `pm_genpd_runtime_suspend(dev)` is called.
3. The "stop" operation is carried out for the device.
4. `pm_genpd_poweroff(pd)` is called for the device's domain.
5. If all devices in the domain are stopped and its subdomains are "off":
   1. The states of all devices in the domain are saved.
   2. The "power off" operation is carried out for the domain.

Device Suspend

1. Device usage counter is 0, the PM core runs `rpm_suspend(dev)`.
2. `pm_genpd_runtime_suspend(dev)` is called.
3. The "stop" operation is carried out for the device.
4. `pm_genpd_poweroff(pd)` is called for the device's domain.
5. If all devices in the domain are stopped and its subdomains are "off":
    1. The states of all devices in the domain are saved.
    2. The "power off" operation is carried out for the domain.

There are multiple conditions that have to be satisfied for this procedure to be completed (it may be aborted at any point).

Device Suspend

1. Device usage counter is 0, the PM core runs `rpm_suspend(dev)`.
2. `pm_genpd_runtime_suspend(dev)` is called.
3. The "stop" operation is carried out for the device.
4. `pm_genpd_poweroff(pd)` is called for the device's domain.
5. If all devices in the domain are stopped and its subdomains are "off":
    1. The states of all devices in the domain are saved.
    2. The "power off" operation is carried out for the domain.

There are multiple conditions that have to be satisfied for this procedure to be completed (it may be aborted at any point).

Timing (PM QoS) constraints should be taken into account when deciding whether or not to carry out "power off" (not implemented yet).

Device Resume

1. Device usage counter is incremented, the PM core runs `rpm_resume(dev)`.
2. `pm_genpd_runtime_resume(dev)` is called.
3. If necessary, the "power on" operation is carried out for the device's domain.
   - This has to abort all instances of `pm_genpd_poweroff(pd)` running for the same domain.
   - It has to be done recursively for all of the "master" domains before.
4. If necessary, the device's state is restored.
5. The "start" operation is carried out for the device.

Device Resume

1. Device usage counter is incremented, the PM core runs `rpm_resume(dev)`.
2. `pm_genpd_runtime_resume(dev)` is called.
3. If necessary, the "power on" operation is carried out for the device's domain.
   - This has to abort all instances of `pm_genpd_poweroff(pd)` running for the same domain.
   - It has to be done recursively for all of the "master" domains before.
4. If necessary, the device's state is restored.
5. The "start" operation is carried out for the device.

There are devices with cross-domain dependencies requiring special handling (interrupt controller, DMA engine). Not supported yet.

Power Management Quality Of Service (PM QoS)

Observations

1. When a device is suspended, the kernel has to decide whether or not to turn its power domain (or its master) off.
2. It takes time to turn power domains off/on and to restore devices' registers.
3. There may be requirements regarding the time devices can spend in the non-working state.
4. The device suspend code should take those requirements into account.

Power Management Quality Of Service (PM QoS)

Observations

1. When a device is suspended, the kernel has to decide whether or not to turn its power domain (or its master) off.
2. It takes time to turn power domains off/on and to restore devices' registers.
3. There may be requirements regarding the time devices can spend in the non-working state.
4. The device suspend code should take those requirements into account.

Device PM QoS

Framework allowing kernel subsystems to specify wakeup latency constraints for I/O devices (scheduled for inclusion into the 3.2 kernel).

Power Management Quality Of Service (PM QoS)

Observations

1. When a device is suspended, the kernel has to decide whether or not to turn its power domain (or its master) off.
2. It takes time to turn power domains off/on and to restore devices' registers.
3. There may be requirements regarding the time devices can spend in the non-working state.
4. The device suspend code should take those requirements into account.

Device PM QoS

Framework allowing kernel subsystems to specify wakeup latency constraints for I/O devices (scheduled for inclusion into the 3.2 kernel).

The generic PM domains code will take PM QoS constraints into account.

Sleep States

Well, no ACPI BIOS, no system sleep states, right?

Sleep States

Well, no ACPI BIOS, no system sleep states, right?

Life's more complicated than that

1. Android uses system suspend for power saving.
2. Wakeup from deep low-power states may require special signaling.

Sleep States

Well, no ACPI BIOS, no system sleep states, right?

Life's more complicated than that

1. Android uses system suspend for power saving.
2. Wakeup from deep low-power states may require special signaling.

Difference between "sleep" and "power off"

The preservation of RAM contents.

Sleep States

Well, no ACPI BIOS, no system sleep states, right?

Life's more complicated than that

1. Android uses system suspend for power saving.
2. Wakeup from deep low-power states may require special signaling.

Difference between "sleep" and "power off"

The preservation of RAM contents.

Still

SH7372 doesn't provide any "enter a sleep state" logic.

“Sleep” Vs “Low-Power”

Sleep states are special low-power states of the whole system.

"Sleep" Vs "Low-Power"

Sleep states are special low-power states of the whole system.

"Sleep" state

1. Reduced energy consumption.
2. User space is not being run.
3. Limited set of devices whose wakeup signals cause user space to be "thawed" (controlled by user space).
4. System may be instructed to go into it at any time.

"Sleep" Vs "Low-Power"

Sleep states are special low-power states of the whole system.

"Sleep" state

1. Reduced energy consumption.
2. User space is not being run.
3. Limited set of devices whose wakeup signals cause user space to be "thawed" (controlled by user space).
4. System may be instructed to go into it at any time.

There need to be special procedures leading to and from a sleep state

- System suspend and resume, respectively.

System Suspend And Power Domains

Rules

1. The state of devices in a power domain has to be saved before removing power from it.
2. Domains that were "off" before system suspend generally should stay this way.

System Suspend And Power Domains

Rules

1. The state of devices in a power domain has to be saved before removing power from it.
2. Domains that were "off" before system suspend generally should stay this way.

Wakeup device problem

1. A power domain in the "off" state contains a device whose input lines can generate wakeup events.
2. User space doesn't want to be woken up by this device.

System Suspend And Power Domains

Rules

1. The state of devices in a power domain has to be saved before removing power from it.
2. Domains that were "off" before system suspend generally should stay this way.

Wakeup device problem

1. A power domain in the "off" state contains a device whose input lines can generate wakeup events.
2. User space doesn't want to be woken up by this device.

In that case it is necessary to restore power to the domain and to reprogram the device during system suspend.

1. SoCs (like SH7372) add complexity to power management (power domains, direct control of device clocks).
2. Generic PM code has to be extended to address the added complexity.
3. That code may be modeled on the basis of the SH7372 design.
4. The fact that the new code can be tested on real hardware matching basic assumptions closely helps a lot.
5. The introduction of new generic PM code allows other platforms to use it without adding code duplication and "reinventing the wheel".
6. Discussions with developers working on other platforms resulted in better code.

References

- R. J. Wysocki, *Runtime PM vs System Sleep* (http://www.linuxplumbersconf.org/2011/ocw/system/presentations/27/original/system_sleep_vs_runtime_PM.pdf).
- R. J. Wysocki, *Runtime Power Management vs System Sleep* (http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki.pdf).
- R. J. Wysocki, *Runtime Power Management Framework for I/O Devices in the Linux Kernel* (http://events.linuxfoundation.org/slides/2010/linuxcon2010_wysocki.pdf).
- R. J. Wysocki, *Runtime Power Management Framework for I/O Devices in the Linux Kernel* (http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki2.pdf).

Documentation And Source Code

- Documentation/power/devices.txt
- Documentation/power/runtime_pm.txt
- include/linux/cpuidle.h
- include/linux/device.h
- include/linux/pm.h
- include/linux/pm_domain.h
- include/linux/pm_runtime.h
- include/linux/pm_wakeup.h
- include/linux/suspend.h
- drivers/base/power/
- drivers/cpuidle/
- drivers/cpufreq/
- kernel/power/

Thanks!

Thank you for attention!

Thanks!

Thank you for attention!

Special thanks to

Renesas Electronics Corp.

Faculty of Physics U. Warsaw