Runtime Power Management on SuperH Mobile
Upstream Implementation and Status

Magnus Damm
damm@igel.co.jp

Renesas Technology

April 2009
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Outline

Hardware Overview
- Introduction
- SuperH Mobile Hardware

Linux Kernel Power Management
- Clock Framework
- Device Drivers
- Sleep Modes
- Hibernation and Suspend
- Timers
- Idle loop

Performance and Upstream Status
- Performance
- Upstream Status and Future Work
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Hardware - Introduction

Magnus Damm (damm@igel.co.jp)  Runtime Power Management on SuperH Mobile  April 2009
SuperH Mobile Migo-R board with Aglient 34401A multimeter
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
SuperH Mobile Hardware - SH7723

SH7723 Block Diagram

- NOR Flash
- DDR1
- NAND
- PAL/NTSC Encoder
- TV-Out NTSC/PAL
- Touch-Panel VF
- LCD Controller
- VPU5F MPEG4/H.264
- VC1 Processing Unit
- 2DG 2D Graphic Accelerator
- MeRAM 128KB
- VIO5 5M-Pixel Camera VF, Image & Blend Function
- SIU2 Sound Accelerator, DSP Function, Equalizer, Filter
- ATAPI I/F
- I/O Port
- Sound CODEC
- DVD VF
- ISDB-T or DVB-H DMB Tuner Module
- Camera Module (Max 5M-Pixel)
- Audio Player
- SD Card
- WiFi
SuperH Mobile Hardware - Simplified Version

Simplified version of SH7343, SH7722, SH7366, SH7723
SuperH Mobile Hardware - Sleep modes

The `sleep` instruction controls system standby modes:

- “Sleep Mode” (CPU Core clock disabled only)
- “Software Standby Mode” (All system clocks off except RCLK)
- “R-Standby Mode” (Core power off)
- “U-Standby Mode” (Core power off, resume from reset vector)

Some processors support R-Standby only, some U-Standby only.
Clock Pulse Generator:

- **Simplified Version**: 5 Clocks, 7 Module Stop Bits
- **SH7723**: 12 Clocks, 48 Module Stop Bits
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Clock Framework - API

The architecture independent Clock Framework API:

From `include/linux/clk.h`:

- clk = clk_get(device, string);
- clk_enable(clk);
- clk_get_rate(clk);
- clk_disable(clk);
- clk_put(clk);

Note: `clk_get_rate()` only valid after `clk_enable()`!
From `include/linux/clk.h`:

- `clk_enable(clk);`
- `clk_disable(clk);`
Outline

Hardware Overview
Introduction
SuperH Mobile Hardware

Linux Kernel Power Management
Clock Framework
Device Drivers
Sleep Modes
Hibernation and Suspend
Timers
Idle loop

Performance and Upstream Status
Performance
Upstream Status and Future Work
Device Drivers - Overview

SuperH Mobile Device Driver Examples:
- i2c-sh_mobile.c - Clock enabled during I2C transfer
- sh_mobile_ceu_camera.c - Clock enabled during Camera capture
- sh_mobile_lcdcfb.c - Use deferred io with SYS panels
- sh_sci.c - Serial port receive needs clock enabled
- uio_pdrv_genirq.c - UIO device needs clock framework feature!
Device Drivers - PM Suggestions

For Energy Efficient Device Drivers...

- Use clock framework for System Clocks / Module Stop Bits
- Disable clocks when hardware is unused
- Avoid software polling
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Sleep Modes - Overview

Architecture independent overview:

- Light: Low latency - Few dependencies - Basic Power Savings
- ...
- ...
- Deep: High latency - Many dependencies - Good Power Savings

Theory: For best power savings, enter as deep mode as possible!
Self-Refresh...
  ▶ is a must when System RAM clocks are stopped
  ▶ means System RAM may not be accessed
  ▶ requires Sleep code in cache or on-chip RAM

SuperH Mobile Sleep code is...
  ▶ Implemented in relocatable assembly code
  ▶ Relocated to on-chip RAM and executed there

In the future sleep code may use a simple in-kernel code generator
SuperH Mobile Software Sleep Modes:

- “Sleep Mode”
- “Sleep Mode” + System RAM in Self-Refresh
- “Software Standby Mode” + System RAM in Self-Refresh
- “U/R-Standby Mode” + System RAM in Self-Refresh
SuperH Mobile “Sleep Mode” properties:

- Stops CPU Core clock
- Wakeup from interrupt or reset
- Low latency, few hardware dependencies
- Basic power savings
“Sleep Mode” may be combined with System RAM Self-Refresh:

- Special Linux-only software sleep mode
- Saves System RAM power
- May only be used if System RAM is inactive
- Must check bus master activity
SuperH Mobile “Software Standby Mode” properties:

- Stops all system clocks except RCLK
- System RAM must be put in Self-Refresh mode
- Wakeup from interrupt or reset
- Around 2 ms latency, many hardware dependencies
SuperH Mobile “U/R-Standby Mode” properties:

- Powers down SoC Core area
- Wakeup from interrupt or reset
- “U-Standby Mode” resumes from reset vector
- Most hardware blocks need re-init on resume
Sleep Modes - Upstream Status

Queued for 2.6.30:
- “Sleep Mode”
- “Sleep Mode” + System RAM in Self-Refresh
- “Software Standby Mode” + System RAM in Self-Refresh

Future work:
- “U/R-Standby Mode” + System RAM in Self-Refresh
Outline

Hardware Overview
  Introduction
  SuperH Mobile Hardware

Linux Kernel Power Management
  Clock Framework
  Device Drivers
  Sleep Modes
  Hibernation and Suspend
  Timers
  Idle loop

Performance and Upstream Status
  Performance
  Upstream Status and Future Work
Hibernation and Suspend - Overview

Easy theory:
- Suspend System
- Wait for Wakeup Event
- Resume System

But in practice:
- Suspend and Hibernation are quite different
- Per-device `struct dev_pm_ops` contains 14 callbacks
- Few in-tree architecture implementations
Suspend-to-Disk (CONFIG_HIBERNATION):

- Freezes system activity, suspends drivers
- Saves image to swap, turns power off
- Power on, boots kernel, loads image from swap
- Resumes drivers, continues system activity

Generic SuperH implementation, tested on a sh7785lcr board
Hibernation and Suspend - Suspend

Suspend-to-Ram (CONFIG_SUSPEND):
- Freezes system activity, suspends drivers
- Enters Sleep mode, waits for wakeup event
- Resumes drivers, continues system activity

On SuperH Mobile..
- Sleep mode translates to “Software Standby Mode”
- RTC and KEYSC drivers support `device_may_wakeup()`
- CONFIG_PM_TEST_SUSPEND is known to be working
Hibernation and Suspend - CONFIG_KEXEC_JUMP

- Hybrid approach using kexec
-Suspends and resumes drivers like CONFIG_HIBERNATION
-Enters secondary kernel
-Can be used for suspend to disk or sleep mode.
Hibernation and Suspend - Upstream Status

- Hibernation and Standby are queued for upstream 2.6.30
- Kexec Jump is queued for upstream 2.6.30 as well
- Latest kexec-tools git has SuperH kexec jump support
- “U/R-Standby” and wakeup dependencies need more work
Outline

Hardware Overview
   Introduction
   SuperH Mobile Hardware

Linux Kernel Power Management
   Clock Framework
   Device Drivers
   Sleep Modes
   Hibernation and Suspend
   Timers
   Idle loop

Performance and Upstream Status
   Performance
   Upstream Status and Future Work
Timers - Overview

The Linux kernel manages time using:

- Clockevents (Schedules next timer event)
- Clocksources (Keeps track of elapsed time)

Available SuperH Mobile Timers:

- CMT (Driven by RCLK, can be used to wake up from deep sleep)
- TMU (High resolution, stopped in “Software Standby Mode”)
Timers - Tickless

Classic jiffy-based system: (CONFIG_NO_HZ=n)
- Clockevent: CLOCK_EVT_MODE_PERIODIC mode
- Clocksource: Optional

Tickless system: (CONFIG_NO_HZ=y)
- Clockevent: CLOCK_EVT_MODE_ONESHOT mode.
- Clocksource: Required (!)

Check wakeup with CONFIG_TIMER_STATS + /proc/timer_stats
Timers - SuperH CMT

Problem: Only one CMT Timer available in deep sleep

Solution: SuperH CMT driver...

- Clockevent and Clocksource with a single timer channel
- Timer hardware counting up, wraps to 0 on match
- Match register reprogrammed, software accumulates time
- Provides a low resolution clocksource
- Supports \texttt{CLOCK_EVT\_MODE\_ONESHOT} clockevent mode.
- Allows tickless operation with a single timer!
Timers - Tickless SuperH CMT

Number of SuperH CMT Interrupts (HZ=100, Tickless ON/OFF)

- CONFIG_NO_HZ=n
- CONFIG_NO_HZ=y

Time (4096 ticks/s, total 30s)
Timers - Upstream Status

Timer changes queued for 2.6.30:
- SuperH CMT driver (clockevent only)
- Clockevent set_mode() delta patch
- setup_irq() and remove_irq() patches

Timer changes submitted upstream:
- Clocksource patches (power management, callback args)
- Early Platform Driver patches
- SuperH CMT driver updates (clocksource/early platform)
- SuperH CMT early platform data
Outline

Hardware Overview
   Introduction
   SuperH Mobile Hardware

Linux Kernel Power Management
   Clock Framework
   Device Drivers
   Sleep Modes
   Hibernation and Suspend
   Timers
   Idle loop

Performance and Upstream Status
   Performance
   Upstream Status and Future Work
Idle loop - CPU Core Power Management

CPU Core power is managed by the idle loop:
- Works well with light sleep
- However, deep sleep comes with latency costs

Cpuidle:
- Keeps track of sleep modes and their latency
- Makes use of SuperH Mobile Software Sleep Modes
- Tries to enter as deep sleep mode as possible
Idle loop - Upstream Status

2.6.30 status:
- Regular SuperH idle loop is using “Sleep Mode” already

Cpuidle status:
- Prototype supports “Software Standby Mode”
- Needs clock framework for dependencies
- Tickless timers needed for good performance
Outline

Hardware Overview
- Introduction
- SuperH Mobile Hardware

Linux Kernel Power Management
- Clock Framework
- Device Drivers
- Sleep Modes
- Hibernation and Suspend
- Timers
- Idle loop

Performance and Upstream Status
- Performance
- Upstream Status and Future Work
Performance and Upstream Status

Outline

Hardware Overview
- Introduction
- SuperH Mobile Hardware

Linux Kernel Power Management
- Clock Framework
- Device Drivers
- Sleep Modes
- Hibernation and Suspend
- Timers
- Idle loop

Performance and Upstream Status
- Performance
- Upstream Status and Future Work
Performance

Typical Processor Power Consumption [SH7722 @ 266/66/133 MHz]:
- No Power Management: ~0.6W
- “Sleep Mode”: ~0.2W
- “Software Standby Mode”: ~0.01W

Migo-R System Power Consumption (Without LCD board):
- Loaded CPU or “nohlt” idle: ~0.6W
- “Sleep Mode” when idle: ~0.4W
- “Software Standby Mode” when idle: ~0.2W
- External Ethernet chip consumes ~0.6W (!)
Outline

Hardware Overview
   Introduction
   SuperH Mobile Hardware

Linux Kernel Power Management
   Clock Framework
   Device Drivers
   Sleep Modes
   Hibernation and Suspend
   Timers
   Idle loop

Performance and Upstream Status

   Performance
   Upstream Status and Future Work
2.6.30:

- Timer patches
- “Sleep Mode” and “Software Standby Mode”
- System RAM in Self-Refresh
- Suspend-to-Disk (CONFIG_HIBERNATION)
- Kexec Jump
- Suspend-to-Ram (CONFIG_SUSPEND)
- Driver wakeup updates (RTC/KEYSC/Touchscreen)
Future Work:

- Improved clock framework support
- Cpuidle integration
- Frequency scaling
- “R-Standby” support
- Driver updates
- UIO Clock framework support