Runtime PM

Upstream I/O Device Power Management

Magnus Damm
magnus.damm.zs@renesas.com

Renesas Electronics Corp.

April 2011

Outline

Outline

Outline

Introduction & Motivation

Power Management is becoming increasingly important for...

- SoC vendors
- Embedded System designers
- End users

Yet, Traditional Linux PM is not a perfect fit for Embedded Systems.

Introduction & Motivation

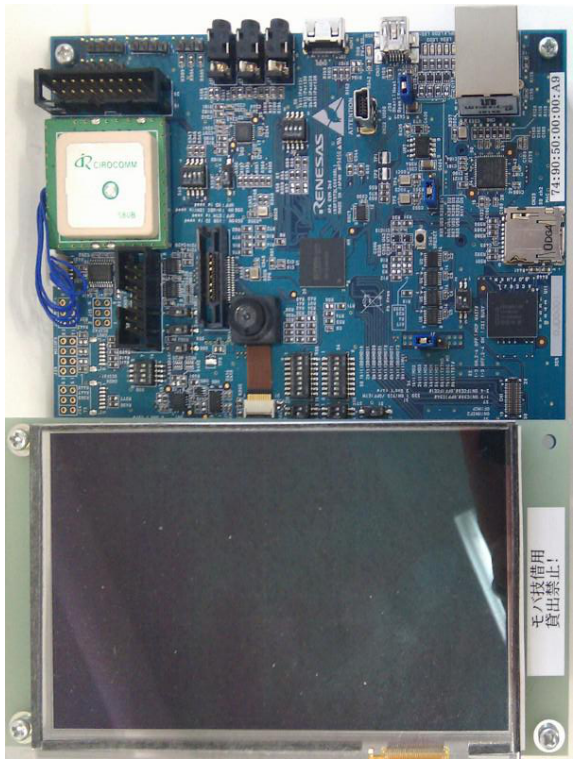
Why?

- Linux kernel PM originally designed for PC use case.
- Little feedback to community from Embedded vendors.
- Embedded vendors treating PM as “secret sauce”.

This future is bright:

- Linux kernel has Runtime PM upstream.
- The Embedded vendors are slowly improving.

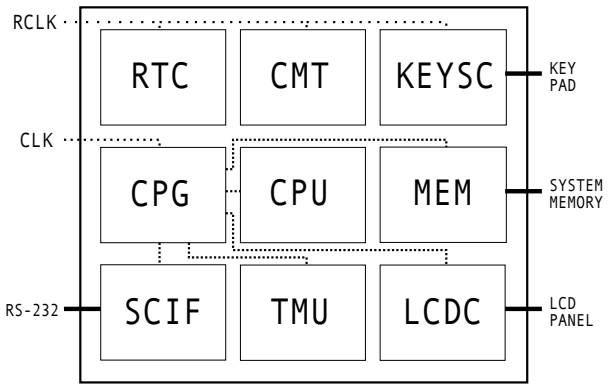
Outline

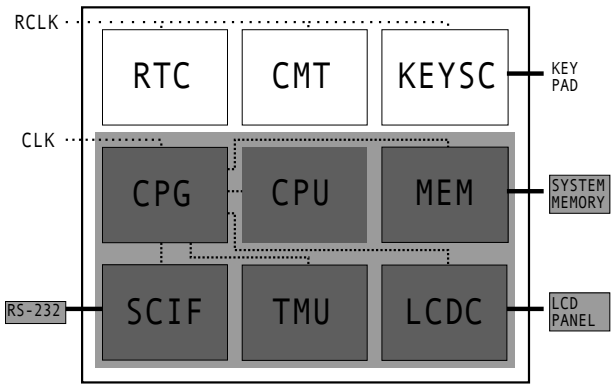


Product name	SH73720 (R8A73720)
CPU core	ARM® Cortex™-A8/NEON™ and SH4AL-DSP
Power supply voltage	Internal: 1.1 to 1.26V External: 1.65 to 1.95V or 2.7 to 3.6V
Max. operating frequency	1GHz (ARM® Cortex™-A8), 400MHz (SH4AL-DSP)
Cache memory	• Primary cache: 32 Kbytes instruction/32 Kbytes data, separate • Secondary cache: 256 Kbytes instruction/data, shared
On-chip RAM	• ILRAM: 4Kbytes • RSRAM: 2Kbytes • MERAM: 1.5Mbyte
On-chip peripheral functions	• Support for 16-megapixel camera ISP • VPU (H.264, MPEG-4, MPEG2, VC-1) • 2D/3D graphics engin Open GL® ES 1.1/2.0, Open VG™ • DMAC×30 channels • MMU • SPU (24-bit dedicated audio DSP×2) • LCD controller with 24-bit TFT color LCD panel support
Interfaces	• Dedicated interface (for connection to baseband processor, etc.) • Transport stream (TS) interface • HDMI v1.3a transmitter • Video I/O (direct interface to camera module) • I²C bus interface • Serial interface with FIFO × 12 channel • Sound interface unit × 2 channels • SD memory card interface × 3 channels • USB2.0 Host/function × 1 channels
Package	561- pin POP-compatible BGA (12×12mm, 0.4mm pin pitch)

Renesas AP4 board "`Mackerel`":

- LAN9220 Ethernet, Serial-over-USB, USB Function/Host
- 256 MiB RAM, NOR Flash, 2 x MMC/SD/SDIO, 1 x MicroSD
- WVGA LCD Panel, 8-bit YUV Camera, Audio In/Out





AP4 (sh7372) SoC Power Management properties:

- 2 CPU Cores (AMP: Cortex-A8 + SH4AL-DSP)
- ~30 Shared clocks
- ~10 Power domains

Outline

Outline

System-Wide PM Overview

System-Wide PM is trivial in theory:

- Suspend System
- Wait for Wakeup Event
- Resume System

But in practice:

- Suspend-to-RAM and Suspend-to-Disk are quite different

Suspend-to-Disk - CONFIG_HIBERNATION

`CONFIG_HIBERNATION`:

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

```
#echo disk >/sys/power/state
```

Suspend-to-Disk allows total system power down.

Suspend-to-RAM - CONFIG_SUSPEND

CONFIG_SUSPEND:

- Freezes system activity, suspends devices
- Enters sleep mode, waits for wakeup event
- Resumes devices, continues system activity

```
#echo mem >/sys/power/state
```

Wakeup latency of Suspend-to-RAM beats Suspend-to-Disk.

Type of Device - Wakeup source or not?

For Suspend-to-RAM there are two types of devices:

- Hardware device without wakeup source
- Hardware device tied to wakeup source

An actual hardware device may have a wakeup source but..

- Software support is missing/incomplete
- Signal for wakeup is not connected

Typical wakeup devices:

- Network interface, RTC, Keypad, Touchscreen

Devices without wakeup source

Simple device driver example:

`->probe()`:

- Allocates memory, Maps I/O memory, Enables clocks
- Requests IRQs, Starts hardware

`->remove()`:

- Stops hardware, Frees IRQs
- Disables clocks, Unmaps I/O memory, Frees memory

`->suspend()`:

- Stops hardware

`->resume()`:

- Starts hardware

Devices with wakeup source

Simple device driver example:

`->probe()`:

- Same as non-wakeup example plus `device_init_wakeup()`

`->suspend()`:

- Put hardware in low power mode if possible
- Checks `device_may_wakeup()`
- Notifies IRQ controller with `enable_irq_wake()`

`->resume()`:

- Put hardware in regular mode of operation
- Checks `device_may_wakeup()`
- Notifies IRQ controller with `disable_irq_wake()`

System Devices

IRQ controller software is at `suspend()` time expected to:

- Disable all non-wakeup IRQs
- Enable IRQs marked with `enable_irq_wake()`

Clock generator software is at `suspend()` time expected to:

- Disable all non-wakeup clocks

Timers:

- Are suspended late in the process.

Outline

PC vs Embedded System

Traditional PC hardware is often associated with:

- BIOS, ACPI and firmware interfaces
- Focus on CPU core, Standardized hardware busses
- Limited number of wakeup sources

In suspended state, most of the PC hardware is shutdown, but...

- At least one wakeup IRQs must be enabled
- A subset of the clocks must be turned on
- Devices with wakeup sources enabled must be kept on

Where are IRQ and clock dependencies for wakeup devices managed?

Most PC hardware is powered off during System-Wide suspend.

PC vs Embedded System

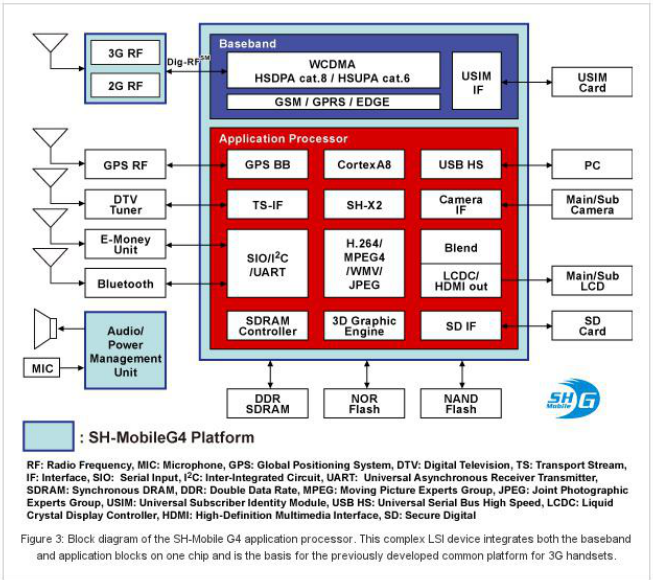
Embedded Systems are often associated with:

- Boot loaders with unreadable source code
- Device drivers programming bare metal
- Focus on I/O devices on custom busses
- Any IRQ can be a wakeup source

In suspended state, most hardware is shutdown, but...

- At least one wakeup IRQs must be enabled
- A subset of the clocks must be turned on
- Devices with wakeup sources enabled must be kept on
- Wakeup device selection limits available sleep modes

No firmware to abstract IRQ and clock dependencies.



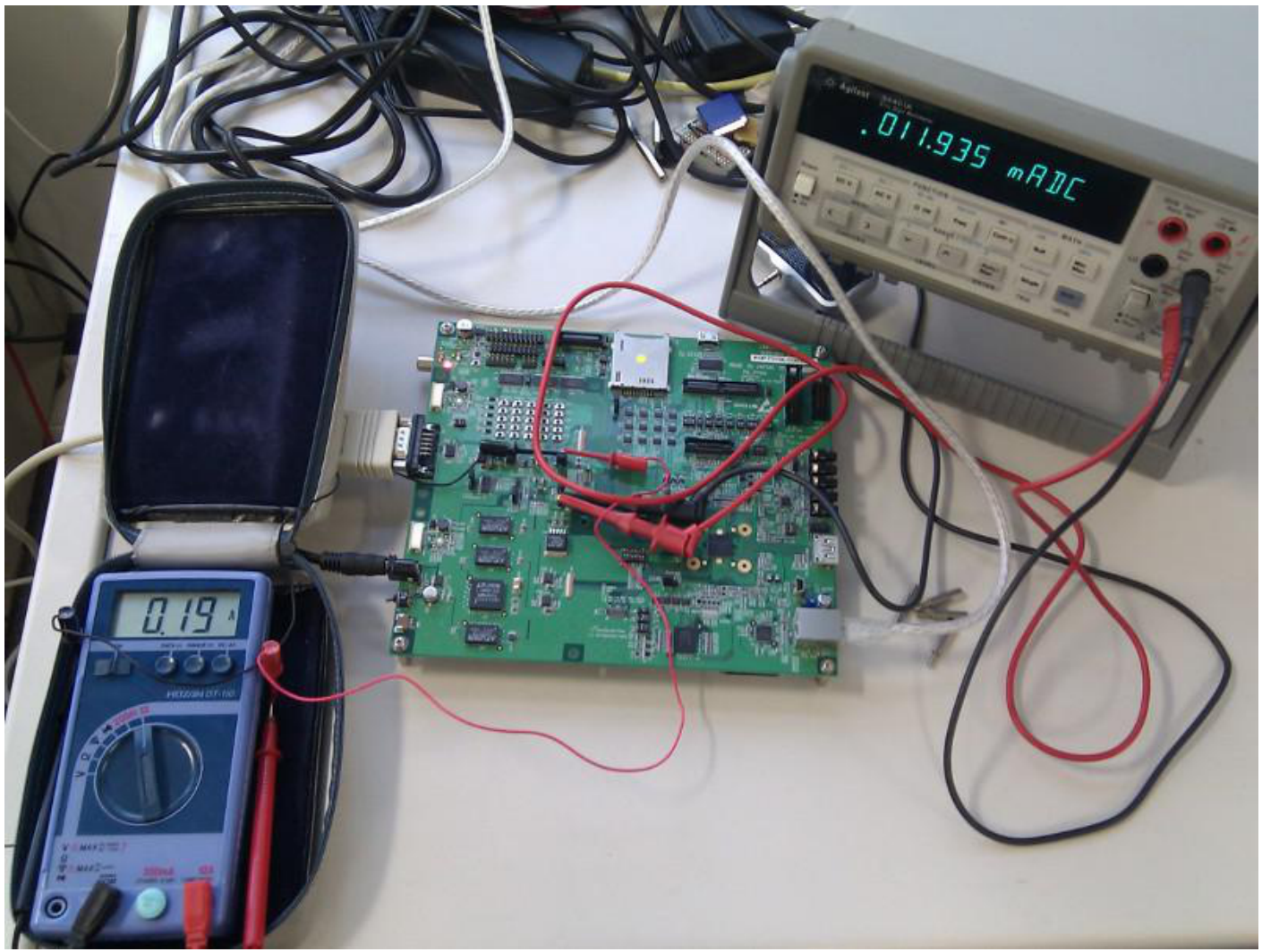
Typical Japanese Cell Phone:

- Even during standby some CPU cores need to be awake.
- Vendor-specific code deals with wakeup dependencies.

System-Wide suspend provides no dependency information.

Outline

Prototypes



Outline

Outline

Linux Idle loop & CPUIdle

CPU Core power is managed by the idle loop:

- Works well with light sleep
- However, deep sleep comes with latency costs

CPUIdle replaces the idle loop and..

- Keeps track of sleep modes and their latency
- Clock and power domain hierarchy limits availability

CPUIdle Overview

Architecture independent overview:

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

Theory: For best power savings, enter as deep mode as possible!

sh7372 CPUIdle Support

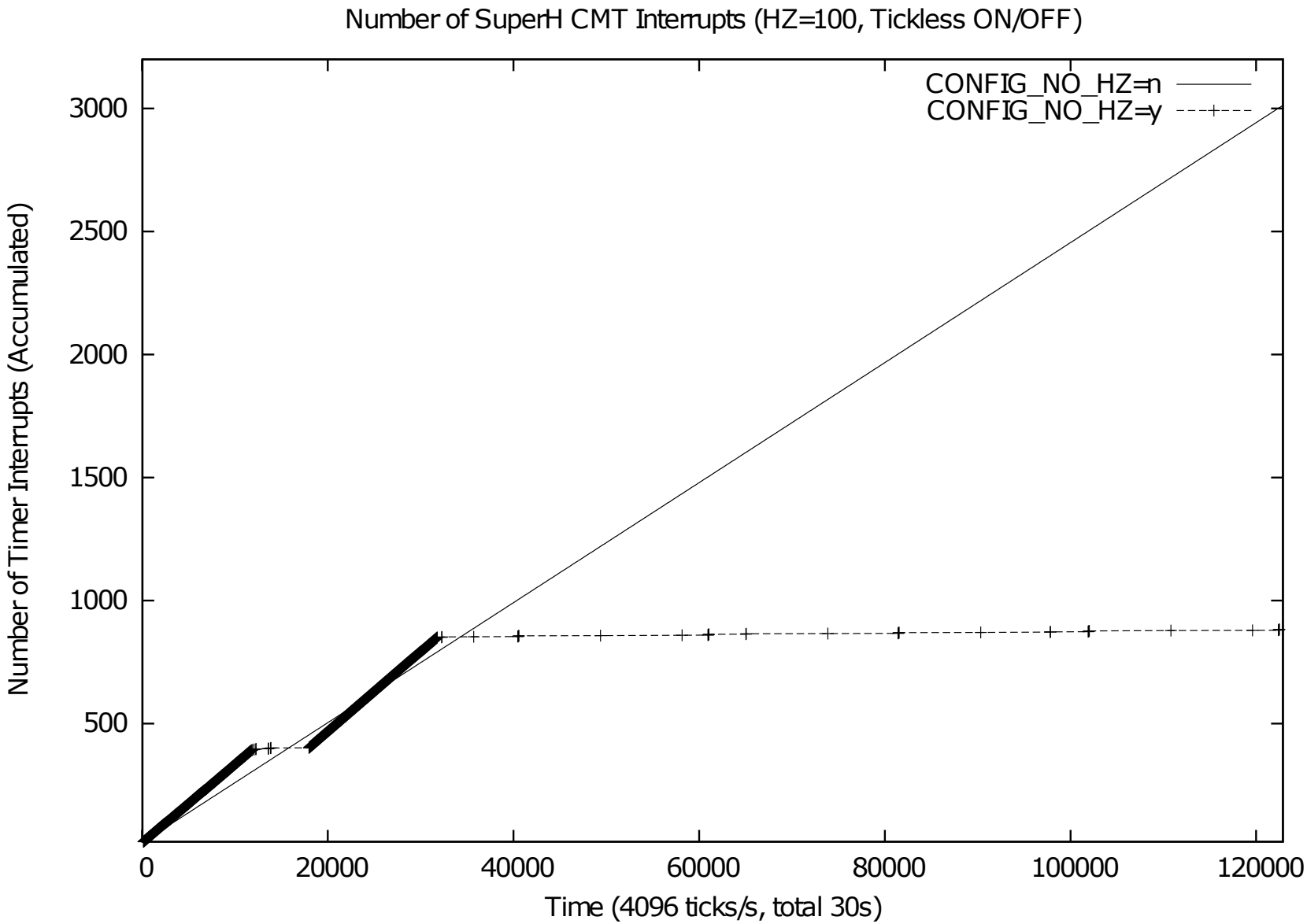
sh7372 ARM CPUIdle Overview:

- ARM WFI - Clock stopped
- Core Standby - ARM Core Power Off (L2 Cache Power On)
- A3SM - ARM Core + L2 Cache Power Off
- A4S - ARM Core + L2 Cache + I/O Devices Power Off

An ARM Core Standby prototype for sh7372 has been posted to:
`http://www.spinics.net/lists/linux-sh/msg07385.html`

Outline

Tickless CMT Timer



Outline

Outline

Runtime PM for Platform Devices:

- Give drivers a single interface for clock and power domains.
- Allows drivers to notify architecure code of device idle state.
- Provide drivers with PM callbacks for context save/restore.
- Used in drivers by Renesas, TI, Intel, Qualcomm and Samsung.

Runtime PM allows architecture code to:

- Track device driver idle state.
- Let device idle state control power domains.
- Adjust CPU sleep mode depending on idle state of devices.

Runtime PM allow drivers to export dependency information.

Runtime PM exists thanks to:

- Kevin Hilman & Paul Walmsley - Initial Runtime PM discussions
- Rafael Wysocki - Runtime PM Implementation
- CELF / Linux Foundation - For ELC and Collaboration space

Runtime PM for Platform Devices - API

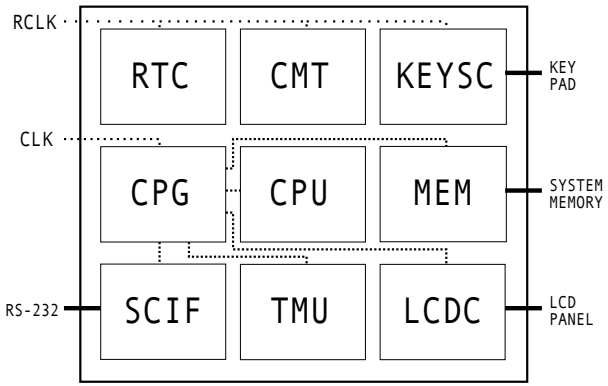
Functions from `include/linux/pm_runtime.h`:

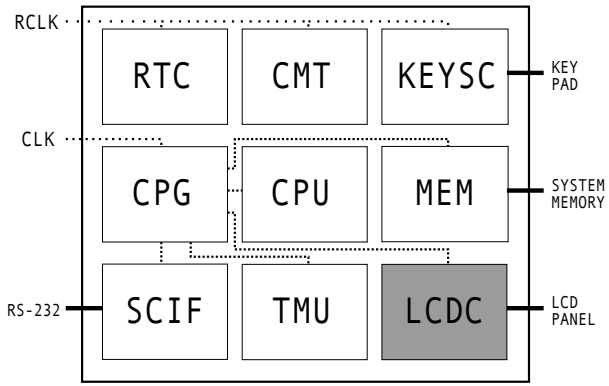
- `pm_runtime_enable(device);`
- `pm_runtime_get_sync(device);`
- `pm_runtime_put_sync(device);`
- `pm_runtime_disable(device);`

Each driver provide `struct dev_pm_ops` callbacks:

- `runtime_suspend(device);`
- `runtime_resume(device);`

Runtime PM Framework - API





Runtime PM usage in Platform Device Drivers:

- Before accessing hardware, resume device with `pm_runtime_get_sync();`
- When done with hardware, notify device idle with `pm_runtime_put_sync();`

Outline

Device Drivers - Overview

Platform Device Drivers with Runtime PM from Renesas:

- i2c-sh_mobile.c - Enabled during I2C transfer
- sh_mobile_ceu_camera.c - Enabled during Camera capture
- sh_mobile_lcdcfb.c - Enabled during refresh with SYS panels
- sh_eth.c - Enabled when network interface is up
- uio_pdrv_genirq.c - Enabled when UIO device is open()

Summary

- System-Wide suspend provides no dependency information.
- Runtime PM allow drivers to export dependency information.
- The majority of all SoC vendors start using Runtime PM.