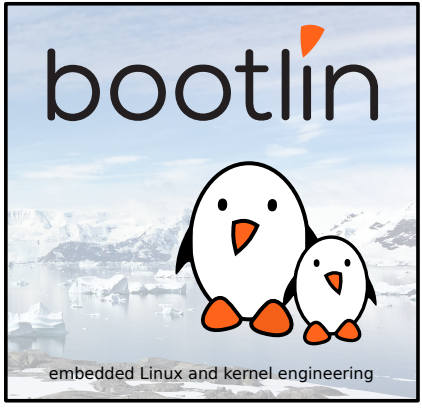
# Linux Power Management features, their relationships and interactions

Théo Lebrun
*theo.lebrun@bootlin.com*



# Théo Lebrun

- Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- Linux kernel device driver developer
  - Suspend-to-RAM support for a TI SoC
  - Upstreaming of Mobileye SoCs
- Open-source contributor (kernel, PipeWire ecosystem, etc.)
- Current maintainer of https://elixir.bootlin.com
- Living in **Lyon**, France
- theo.lebrun@bootlin.com

https://bootlin.com/company/staff/theo-lebrun/

# Table of Contents

# System-wide suspend states

# System-wide suspend

- First, stop the world by freezing all tasks.
- Then suspend individual devices in four steps:
  1. Prepare,
  2. Suspend,
  3. Late suspend,
  4. No-IRQ suspend.
- Finally, go into *« suspend »*.

# System-wide suspend

- First, stop the world by freezing all tasks.
- Then suspend individual devices in four steps:
  1. Prepare,
  2. Suspend,
  3. Late suspend,
  4. No-IRQ suspend.
- Finally, go into *« suspend »*.
- Things to think about:
  - Desired **suspend type**: suspend-to-idle, standby, suspend-to-RAM, hibernation;
  - **Wakeup source**;
  - Targeted individual **device states** during suspend;
  - Entry & exit **latency goals**.
- Doc: `admin-guide/pm/sleep-states` & `admin-guide/pm/suspend-flows`
- Code: `kernel/power/suspend.c`

# System-wide suspend: entering suspend

```
$ cat /sys/power/mem_sleep
[s2idle] deep
$ echo deep > /sys/power/mem_sleep
$ echo mem > /sys/power/state

Unable to handle kernel paging request at virtual address ...
(if you are lucky)
```

# System-wide suspend: entering suspend

```
$ cat /sys/power/mem_sleep
[s2idle] deep
$ echo deep > /sys/power/mem_sleep
$ echo mem > /sys/power/state

Unable to handle kernel paging request at virtual address ...
(if you are lucky)
```

```
$ # Need debugging help?
$ echo 0 > /sys/module/printk/parameters/console_suspend
$ echo 8 > /proc/sys/kernel/printk
$ echo 1 > /sys/power/pm_print_times     # ifdef CONFIG_PM_SLEEP_DEBUG
$ echo 1 > /sys/power/pm_debug_messages  # same
```

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.

include/linux/pm.h

```
struct dev_pm_ops {
    int (*prepare)(struct device *dev);
    int (*suspend)(struct device *dev);
    int (*suspend_late)(struct device *dev);
    int (*suspend_noirq)(struct device *dev);

    int (*resume_noirq)(struct device *dev);
    int (*resume_early)(struct device *dev);
    int (*resume)(struct device *dev);
    void (*complete)(struct device *dev);

    /* and more (hibernation and runtime PM)... */
};
```

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.

include/linux/pm.h

```
struct dev_pm_ops {
    int (*prepare)(struct device *dev);
    int (*suspend)(struct device *dev);
    int (*suspend_late)(struct device *dev);
    int (*suspend_noirq)(struct device *dev);

    int (*resume_noirq)(struct device *dev);
    int (*resume_early)(struct device *dev);
    int (*resume)(struct device *dev);
    void (*complete)(struct device *dev);

    /* and more (hibernation and runtime PM)... */
};
```

```
# Pseudocode (ie Python).

# Each function is called,
# one after the other.
for dev in devices_topdown:
    prepare(dev)
for dev in devices_downtop:
    suspend(dev)
for dev in devices_downtop:
    suspend_late(dev)
for dev in devices_downtop:
    suspend_noirq(dev)
```

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.
- `->prepare()`: do not register new children devices.

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.
- `->prepare()`: do not register new children devices.
- `->suspend()`: please stop doing I/O.

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.
- `->prepare()`: do not register new children devices.
- `->suspend()`: please stop doing I/O.
- `->suspend_late()`: please stop.

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.
- `->prepare()`: do not register new children devices.
- `->suspend()`: please stop doing I/O.
- `->suspend_late()`: please stop.
- `->suspend_noirq()`: please.
  - Additional guarantee: IRQ handlers will not be called.

# System-wide suspend: device PM callbacks

- See `struct dev_pm_ops` & doc `driver-api/pm/devices`.
- `->prepare()`: do not register new children devices.
- `->suspend()`: please stop doing I/O.
- `->suspend_late()`: please stop.
- `->suspend_noirq()`: please.
  - Additional guarantee: IRQ handlers will not be called.
- Any of the `->suspend*()` callbacks *can/might/should/must*, depending on subsystem:
  1. save device state, for later restore, and,
  2. put individual device into low-power state.
- Summary: **behavior is device specific**.
  No guarantees about device states are provided, and no information is *exported* (apart from potential error codes).

# System-wide suspend: device PM callbacks

- Two GPIO controllers with implementations at different stages.
- Implication: `pinctrl-nomadik` pins must be configured at `->suspend()` or before.

drivers/pinctrl/renesas/pinctrl-rzg2l.c

```
static const struct dev_pm_ops rzg2l_pinctrl_pm_ops = {
    NOIRQ_SYSTEM_SLEEP_PM_OPS(rzg2l_pinctrl_suspend_noirq,
                              rzg2l_pinctrl_resume_noirq)
};
```

drivers/pinctrl/nomadik/pinctrl-nomadik.c

```
static SIMPLE_DEV_PM_OPS(nmk_pinctrl_pm_ops,
                         nmk_pinctrl_suspend,
                         nmk_pinctrl_resume);
```

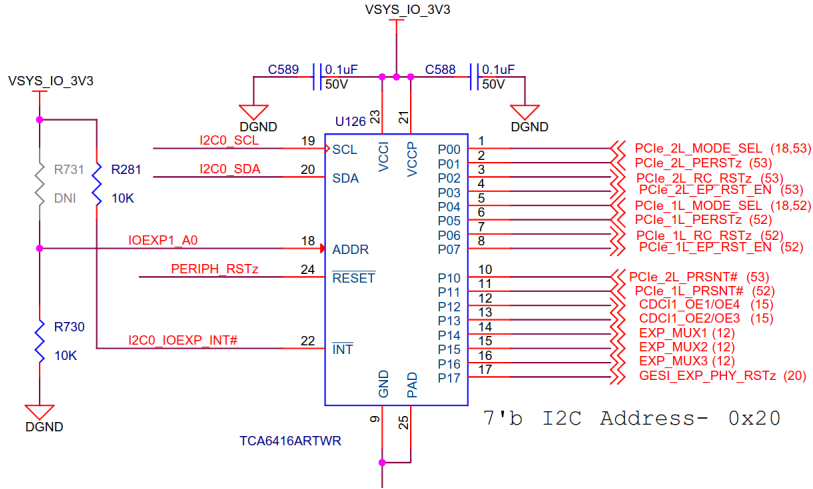
# System-wide suspend: device PM callbacks

- Careful! Moving everything to `->suspend_noirq()` is **not** the solution.

# System-wide suspend: device PM callbacks

- Careful! Moving everything to `->suspend_noirq()` is **not** the solution.
- Do you need interrupts for your suspend process?
- Do the actions you provide, eg `pinctrl_select_state()`, require interrupts to work?
- Goto 1: recursively think about your dependencies.
  They must work as long as you do.

## I2C GPIO EXPANDER1



# System-wide suspend: suspend-to-idle mode

- Always available, if `CONFIG_SUSPEND=y`.
- Piggyback on platforms' idle loop support.

# System-wide suspend: suspend-to-idle mode

- Always available, if `CONFIG_SUSPEND=y`.
- Piggyback on platforms' idle loop support.

`kernel/power/suspend.c`

```
static void s2idle_enter(void) /* abbreviated! */
{
    s2idle_state = S2IDLE_STATE_ENTER;
    /* Push all the CPUs into the idle loop. */
    wake_up_all_idle_cpus();
    /* Put current CPU in idle as well, waiting for wakeup event. */
    swait_event_exclusive(s2idle_wait_head, s2idle_state == S2IDLE_STATE_WAKE);
    /* Wake up all CPUs for them to restore their state. */
    wake_up_all_idle_cpus();
    s2idle_state = S2IDLE_STATE_NONE;
}
```

- `s2idle_state` is set to `S2IDLE_STATE_WAKE` inside interrupt handlers.

# System-wide suspend: platform-provided modes (standby, S2R)

- Next states are calling into platform code.
- States might not be supported!

kernel/power/suspend.c

```
typedef int __bitwise suspend_state_t;
#define PM_SUSPEND_ON       ((__force suspend_state_t) 0)
#define PM_SUSPEND_TO_IDLE  ((__force suspend_state_t) 1)
#define PM_SUSPEND_STANDBY  ((__force suspend_state_t) 2) /* Standby */
#define PM_SUSPEND_MEM      ((__force suspend_state_t) 3) /* Suspend-to-RAM */
#define PM_SUSPEND_MIN      PM_SUSPEND_TO_IDLE
#define PM_SUSPEND_MAX      ((__force suspend_state_t) 4)

struct platform_suspend_ops {
    int (*valid)(suspend_state_t state);
    int (*enter)(suspend_state_t state);
    /* ... */
};
extern void suspend_set_ops(const struct platform_suspend_ops *ops);
```

# System-wide suspend: platform-provided modes (standby, S2R)

- Expected behavior of standby and suspend-to-RAM? *No one knows.*
- S2R *should* lower the memory frequency and put it in self-refresh.

# System-wide suspend: platform-provided modes (standby, S2R)

- Expected behavior of standby and suspend-to-RAM? *No one knows.*
- S2R *should* lower the memory frequency and put it in self-refresh.
- Else?
  - Standby *could* be implemented using an idle loop (`cpu_do_idle()`, WFI).
  - Whole SoC *could* be turned off.
  - None/some/all CPU caches *could* be stopped.
  - None/some clocks *could* be stopped.
  - Few drivers customize their behavior using the `pm_suspend_target_state` global.
  - The regulator subsystem exposes OF properties for picking suspend state: `regulator-state-*`.
- Summary: **behavior is platform specific**.

# System-wide suspend: platform-provided modes examples

- S2R often implies code running from SRAM. See arm/mach-mvebu or arm/mach-at91 for examples fully handled inside Linux.

# System-wide suspend: platform-provided modes examples

- S2R often implies code running from SRAM. See arm/mach-mvebu or arm/mach-at91 for examples fully handled inside Linux.
- PSCI: look into `drivers/firmware/psci/psci.c`. No standby support, only S2R. Offloaded to firmware with a `PSCI_1_0_FN64_SYSTEM_SUSPEND` call.

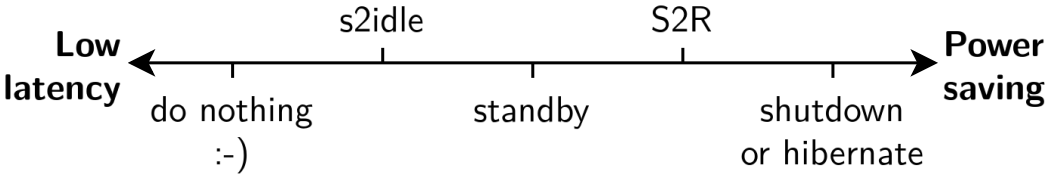
# System-wide suspend: platform-provided modes examples

- S2R often implies code running from SRAM. See arm/mach-mvebu or arm/mach-at91 for examples fully handled inside Linux.
- PSCI: look into `drivers/firmware/psci/psci.c`. No standby support, only S2R. Offloaded to firmware with a `PSCI_1_0_FN64_SYSTEM_SUSPEND` call.
- arm/mach-at91 has 5 different suspend modes:
  from `AT91_PM_STANDBY` (WFI + reduce DRAM power)
  to `AT91_PM_BACKUP` (SoC off + DDR self-refresh + many clocks disabled).
  Standby & S2R can be configured to any of those using a module parameter:
  `atmel.pm_modes=ulp0,backup`.

# System-wide suspend: hibernation

- The most *efficient* mode: shutdown!
- Beforehand, save all state to disk.
- Hibernation is not shutdown:
  - Some peripherals might be configured as wakeup sources.
  - It is useful if userspace takes a long time to initialize at boot.

# System-wide suspend: tradeoff



# System-wide suspend: s2idle & `CLOCK_MONOTONIC` quirk

- What is the expected behavior of `clock_gettime(CLOCK_MONOTONIC, tp)` across suspend?
  1. It should continue ticking,
  2. It should be stopped.
  3. It depends on the suspend type.

# System-wide suspend: s2idle & CLOCK_MONOTONIC quirk

```
$ man clock_gettime.2
...
 CLOCK_MONOTONIC
    A nonsettable system-wide clock that represents monotonic time since—
    as described by POSIX—"some unspecified point in the past". On
    Linux, that point corresponds to the number of seconds that the sys-
    tem has been running since it was booted.

    The CLOCK_MONOTONIC clock is not affected by discontinuous jumps in
    the system time (e.g., if the system administrator manually changes
    the clock), but is affected by frequency adjustments. This clock
    does not count time that the system is suspended. All CLOCK_MONOTO-
    NIC variants guarantee that the time returned by consecutive calls
    will not go backwards, but successive calls may—depending on the ar-
    chitecture—return identical (not-increased) time values.
...
```

# System-wide suspend: s2idle & CLOCK_MONOTONIC quirk

- For a working s2idle, interrupts must be kept enabled (as wakeup source).
- Interrupt handlers make one assumption: the timekeeping subsystem is running.
- CLOCK_MONOTONIC is driven by the timekeeping subsystem.
- $\Longrightarrow$ In s2idle, the kernel breaks its promise.

# System-wide suspend: s2idle & CLOCK_MONOTONIC quirk

- Tip: use tiny tool clkdump to dump clocks.
- It prints all clocks then `sleep(1)`, and loop.
- Guess when the s2idle happened:

```
$ ./clkdump | grep 'MONOTONIC\s'  # output is abbreviated
Sat, 1 Jan 2000 00:35:10 GMT    CLOCK_MONOTONIC    1.000084 s
Sat, 1 Jan 2000 00:35:11 GMT    CLOCK_MONOTONIC    1.000083 s
Sat, 1 Jan 2000 00:35:12 GMT    CLOCK_MONOTONIC    1.000080 s
Sat, 1 Jan 2000 00:35:55 GMT    CLOCK_MONOTONIC   43.095237 s
Sat, 1 Jan 2000 00:35:56 GMT    CLOCK_MONOTONIC    1.000138 s
Sat, 1 Jan 2000 00:35:57 GMT    CLOCK_MONOTONIC    1.000097 s
Sat, 1 Jan 2000 00:35:58 GMT    CLOCK_MONOTONIC    1.000128 s
```

# System-wide suspend: s2idle & CLOCK_MONOTONIC quirk

- **Except** if the kernel offloads the idle loop to a cpuidle device, that can enter s2idle with interrupts disabled. In that case, timekeeping is suspended then resumed and CLOCK_MONOTONIC behaves as expected.
- The code path is completely different inbetween s2idle and s2idle + cpuidle.
- **Most platforms are safe.** Try disabling your cpuidle driver!

# Runtime Power Management
# *(pm_runtime)*

# Runtime PM

- Individual device suspend and resume
- Doc: power/runtime_pm

include/linux/pm.h

```
struct dev_pm_ops {
    /* Device is active but not needed anymore. */
    int (*runtime_suspend)(struct device *dev);

    /* Device is suspended but needed. */
    int (*runtime_resume)(struct device *dev);

    /* ... */
};
```

# Runtime PM

- Devices don't suspend & resume themselves manually.
- Think of the device model as a tree of devices.
- Device users touch a usage reference counter.

```
/* Pseudocode for mental model! */

void pm_runtime_get(struct device *dev) {
    dev.power.usage_count++;
    if (dev->parent)
        pm_runtime_get(dev->parent);
    if (dev->power.usage_count == 1)
        runtime_resume(dev);
}

void pm_runtime_put(struct device *dev) {
    dev.power.usage_count--;
    if (dev.power.usage_count == 0)
        runtime_suspend(dev);
    if (dev->parent)
        pm_runtime_put(dev->parent);
}
```

# Runtime PM: features

- **Kind of!** This code is *slightly* oversimplified.

# Runtime PM: features

- **Kind of!** This code is *slightly* oversimplified.
- What happens when calling `pm_runtime_get|put()` inside an IRQ?
  - `->runtime_suspend|resume()` is marked IRQ safe or,
  - The call **will be done** async (put request into workqueue).
  - See `pm_runtime_irq_safe()` and `RPM_ASYNC`.

# Runtime PM: features

- **Kind of!** This code is *slightly* oversimplified.
- What happens when calling `pm_runtime_get|put()` inside an IRQ?
  - `->runtime_suspend|resume()` is marked IRQ safe or,
  - The call **will be done** async (put request into workqueue).
  - See `pm_runtime_irq_safe()` and `RPM_ASYNC`.
- Devices can be disabled.
  - `dev->power.disable_depth`, yet another refcount.
  - Devices' default state is disabled (refcount=1).
  - Disabling a device runtime PM does not force suspend it!
    You can disable a device while it is active, and it will stay put.
  - See `pm_runtime_enable()` and `pm_runtime_disable()`.

# Runtime PM: features (bis)

# Runtime PM: features (bis)

- Devices can be allowed/forbidden from runtime PM.
  - `dev->power.runtime_auto` boolean.
  - This is different from disabling!
  - Default state is `true`.
  - See `pm_runtime_allow()` and `pm_runtime_forbid()`.

# Runtime PM: features (bis)

- Devices can be allowed/forbidden from runtime PM.
  - `dev->power.runtime_auto` boolean.
  - This is different from disabling!
  - Default state is `true`.
  - See `pm_runtime_allow()` and `pm_runtime_forbid()`.
- A device can ask to ignore its `->runtime_suspend|resume()` callbacks.
  - You are a minor, ie your parent (device) handles PM for you.
  - See `pm_runtime_no_callbacks()`.

# Runtime PM: features (bis)

- Devices can be allowed/forbidden from runtime PM.
  - `dev->power.runtime_auto` boolean.
  - This is different from disabling!
  - Default state is `true`.
  - See `pm_runtime_allow()` and `pm_runtime_forbid()`.
- A device can ask to ignore its `->runtime_suspend|resume()` callbacks.
  - You are a minor, ie your parent (device) handles PM for you.
  - See `pm_runtime_no_callbacks()`.
- **Autosuspend!**
  - Don't suspend as soon as `dev->power.usage_count == 0`, wait a bit.
  - Think storage device that you do not want to toggle on/off all the time.
  - See `pm_runtime_use_autosuspend()` and `pm_runtime_set_autosuspend_delay()`.
  - Userspace might play with it: `/sys/devices/.../power/autosuspend_delay_ms`.

# Runtime PM vs system-wide suspend

- Implicit `pm_runtime_disable()` before suspend-late.
- *Almost*. Standard `pm_runtime_disable()` wakes up the device if there is a resume request pending.

# Runtime PM vs system-wide suspend

- Implicit `pm_runtime_disable()` before suspend-late.
- *Almost*. Standard `pm_runtime_disable()` wakes up the device if there is a resume request pending.
- Implicit `pm_runtime_enable()` after resume-early.

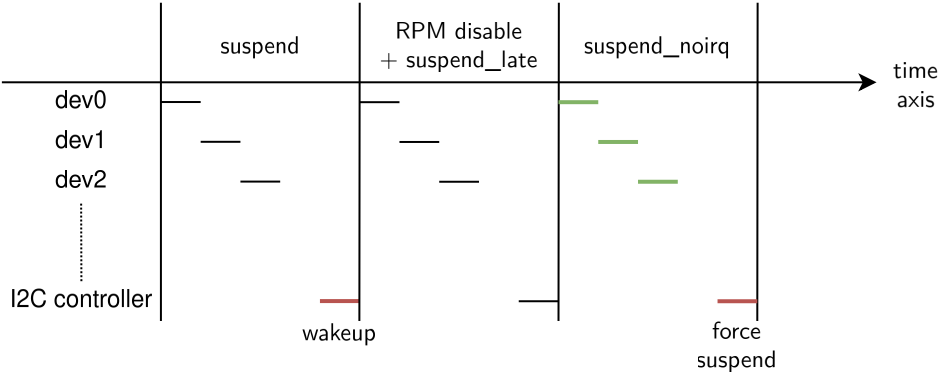
# Runtime PM vs system-wide suspend

- Implicit `pm_runtime_disable()` before suspend-late.
- *Almost.* Standard `pm_runtime_disable()` wakes up the device if there is a resume request pending.
- Implicit `pm_runtime_enable()` after resume-early.
- Each driver must take explicit action!
  1. Do nothing, the default;
  2. `pm_runtime_force_suspend()` & `pm_runtime_force_resume()`;
  3. Custom behavior otherwise.

# Runtime PM vs system-wide suspend: example issue

```
commit 7da7fd7e66ac9b0d4287aefba516795145f3c722
Author: Thomas Richard <thomas.richard@bootlin.com>
Date:   Thu Jun 13 15:13:28 2024 +0200

i2c: omap: wakeup the controller during suspend() callback

A device may need the controller up during suspend_noirq() or resume_noirq().
But if the controller is autosuspended, there is no way to wake it up during
suspend_noirq() or resume_noirq() because runtime PM is disabled.

The suspend() callback wakes up the controller, so it is available until
its suspend_noirq() callback (pm_runtime_force_suspend()). During the resume,
it is restored by resume_noirq() callback (pm_runtime_force_resume()).
Then resume() callback enables autosuspend.

So the controller is up during a little time slot in suspend and resume
sequences even if it is not used.
```

# Runtime PM vs system-wide suspend: example issue



# Summary

- **Behavior is device and platform specific.**
  *« X is suspended »* does not tell you much.

# Summary

- **Behavior is device and platform specific.**
  *« X is suspended »* does not tell you much.
- Issues arise when subsystems, each with their suspend assumptions, come in contact.

# Summary

- **Behavior is device and platform specific.**
  *« X is suspended »* does not tell you much.
- Issues arise when subsystems, each with their suspend assumptions, come in contact.
- Beware of code paths that differ from one suspend type to another.

# Summary

- **Behavior is device and platform specific.**
  *« X is suspended »* does not tell you much.
- Issues arise when subsystems, each with their suspend assumptions, come in contact.
- Beware of code paths that differ from one suspend type to another.

- **To be continued...**
  genpd, QoS, wakeup sources, and more (?).

# Questions? Suggestions? Comments?

Théo Lebrun
*theo.lebrun@bootlin.com*