

Device Power Management and Idle

Ulf Hansson, Linaro, PMWG
ulf.hansson@linaro.org



Linaro
connect
Bangkok 2019

Goal: Don't waste energy when idle!

- Interfaces and frameworks
- Use cases and deployment
- Good practices and some useful tricks

Feel free to ask questions!

System wide sleep - overview

System wide sleep for all devices/resources.

- Triggered from userspace.
 - Closing the lid on your laptop.
- Triggered internally by the kernel.
 - Autosleep - used by Android.
- Sleep may be prevented by wakeup sources (wake locks).
 - Kernel: `pm_stay_awake()`, `pm_wakeup_event(timeout)`.
 - Userspace: `/sys/power/wake_lock|unlock`.

System wide sleep - low power states

- Suspend-to-idle - always supported.
- Suspend-to-standby.
- Suspend-to-RAM.
- Suspend-to-disk (aka hibernation).

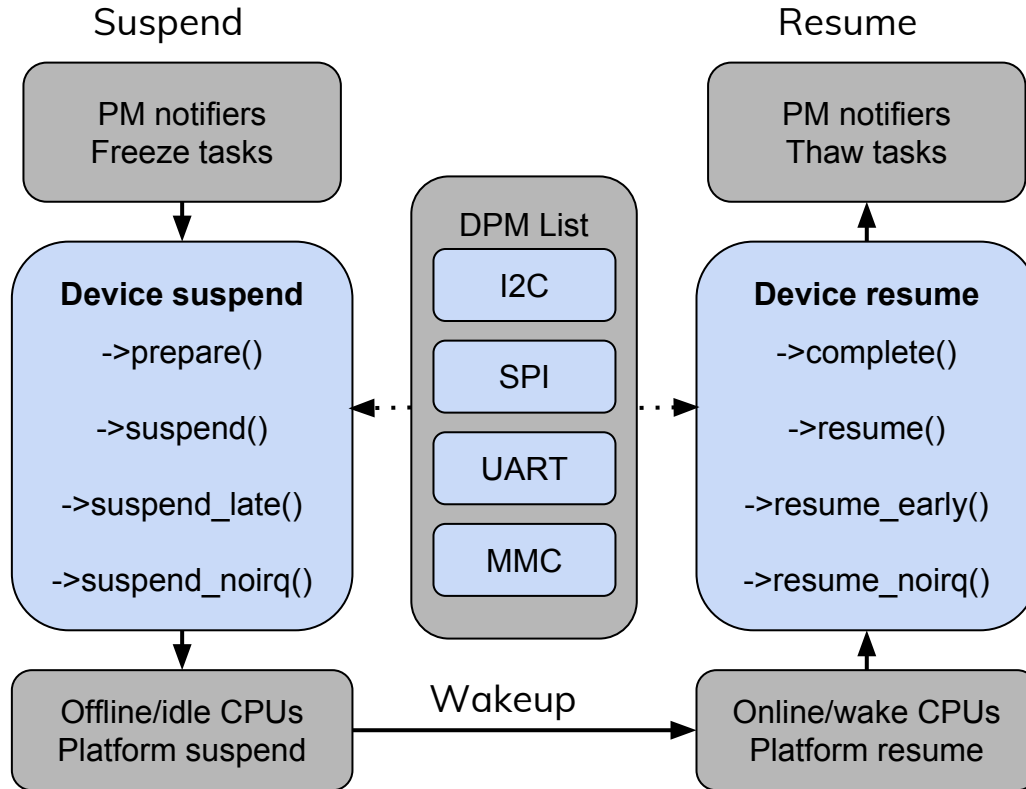
System wide sleep - sysfs

- `echo [state] > /sys/power/state`
 - **“freeze”** == suspend-to-idle == ACPI S0 (always supported)
 - **“standby”** == ACPI S1 (platform specific)
 - **“mem”** == depends on `/sys/power/mem_sleep` (platform specific)
 - **“disk”** == hibernation

- `echo [state] > /sys/power/mem_sleep`
 - **“s2idle”** == suspend-to-idle
 - **“shallow”** == “standby”
 - **“deep”** == suspend-to-RAM

- `echo [state] /sys/power/autosleep`
 - State available in `/sys/power/state`
 - **“off”** == disable autosleep

System suspend/resume



Runtime PM - overview

Avoid wasting energy for idle resources on a running system

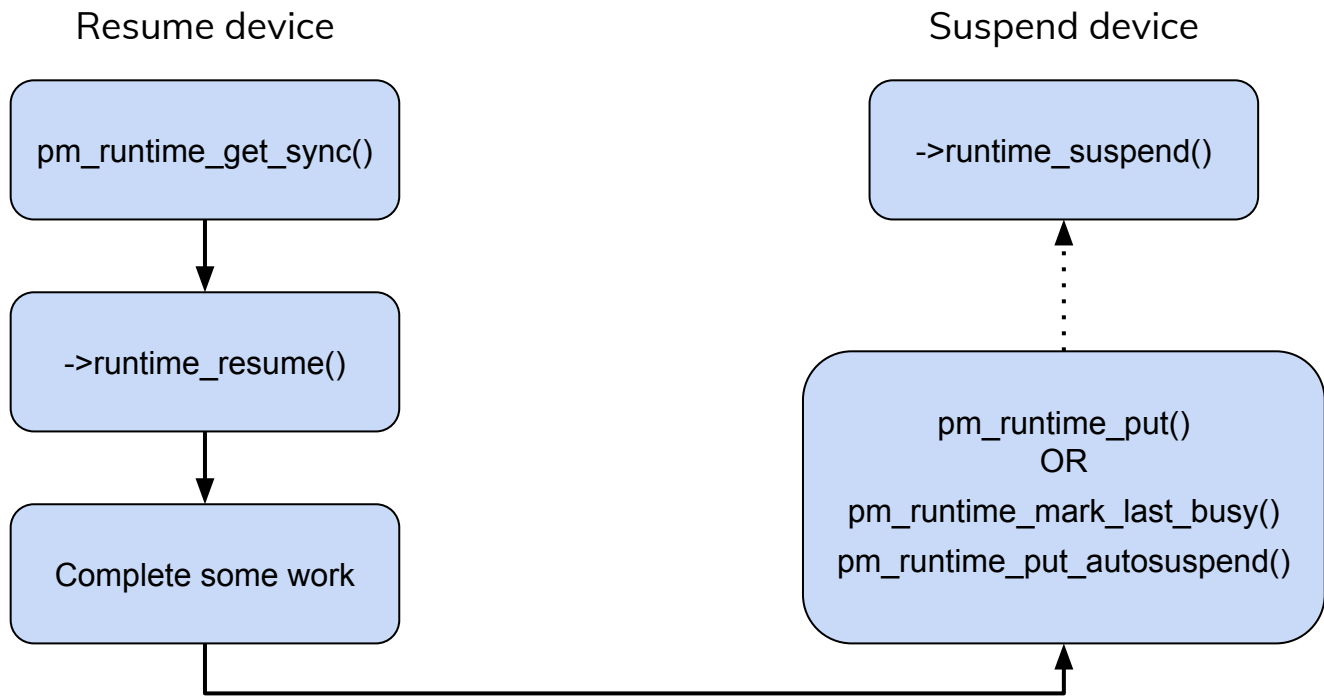
- **At request inactivity and for unused devices.**
- Deployment needed by the subsystem/driver - per device.
 - `pm_runtime_enable()`.
- Suspend and resumed state.
 - `pm_runtime_resume()`.
 - `pm_runtime_suspend()`.
- Allows reference counting.
 - `pm_runtime_get_sync()` - synchronous.
 - `pm_runtime_put()` - asynchronous.
- Defer suspend to after a period of idle.
 - `pm_runtime_set_autosuspend_delay()`.
 - `pm_runtime_use_autosuspend()`.
 - `pm_runtime_mark_last_busy()`.

Runtime PM - sysfs

- `echo [on|auto] > /sys/devices/*/*/power/control`
 - Force a device to become and stay resumed.

- `cat /sys/devices/*/*/power/runtime_*`
 - State and stats.
 - `CONFIG_PM_ADVANCED_DEBUG`.

Runtime PM suspend/resume



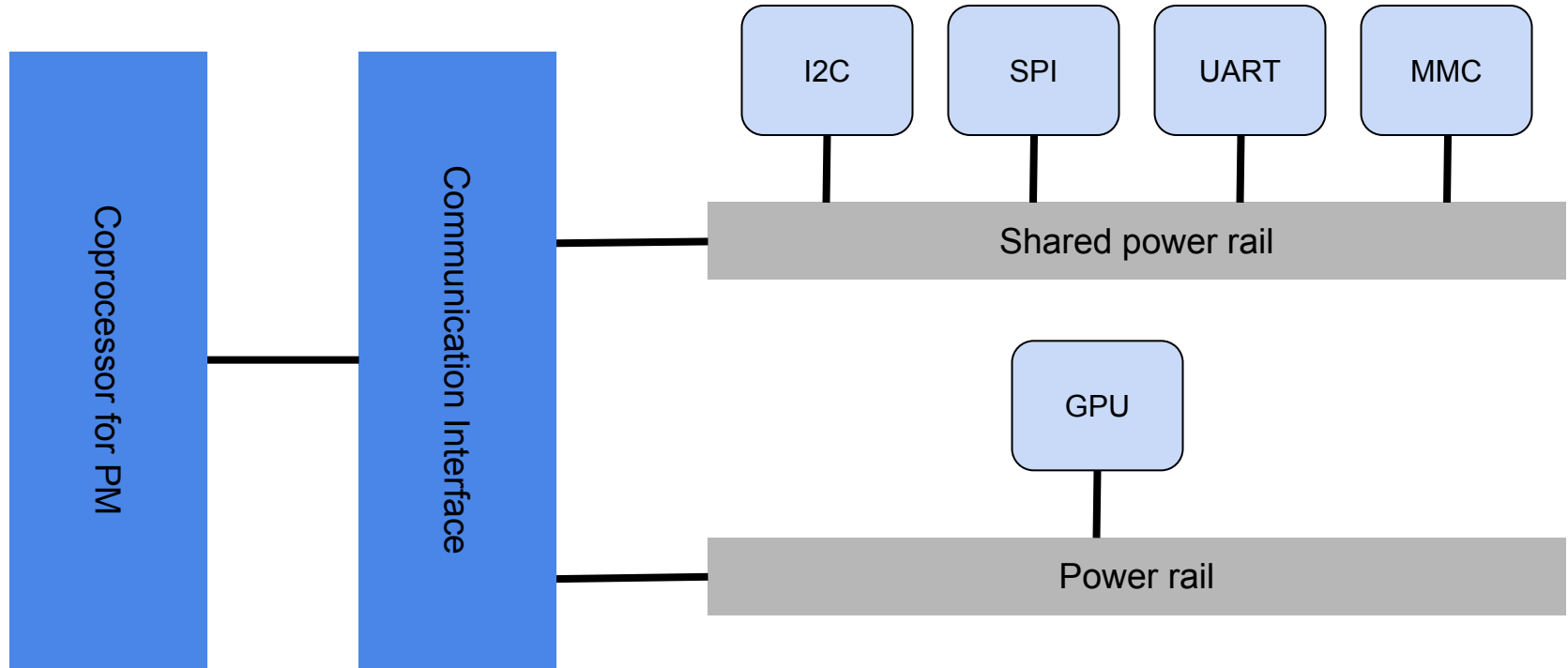
The device PM callbacks

```
struct dev_pm_ops {  
    int (*prepare)(struct device *dev);  
    void (*complete)(struct device *dev);  
    int (*suspend)(struct device *dev);  
    int (*resume)(struct device *dev);  
    int (*suspend_late)(struct device *dev);  
    int (*resume_early)(struct device *dev);  
    int (*suspend_noirq)(struct device *dev);  
    int (*resume_noirq)(struct device *dev);  
    ...  
    int (*runtime_suspend)(struct device *dev);  
    int (*runtime_resume)(struct device *dev);  
    int (*runtime_idle)(struct device *dev);  
};
```

System wide suspend/resume

Runtime PM suspend/resume

SoC PM Topology - PM domains



Hierarchy of device PM callbacks

```
struct device {  
    ...  
    struct device          *parent;  
    ...  
    struct dev_pm_domain   *pm_domain;  
    struct bus_type        *bus;  
    struct device_driver   *driver;  
    ...  
};
```

The ACPI PM domain

- Used on x86 and ARM servers.
 - `drivers/acpi/device_pm.c` (etc)
- Centralized power management for devices.
 - All in FW - PM topology, clocks, power-rails, etc.
 - Knows about wakeups.
- Supports system wide sleep and runtime PM.

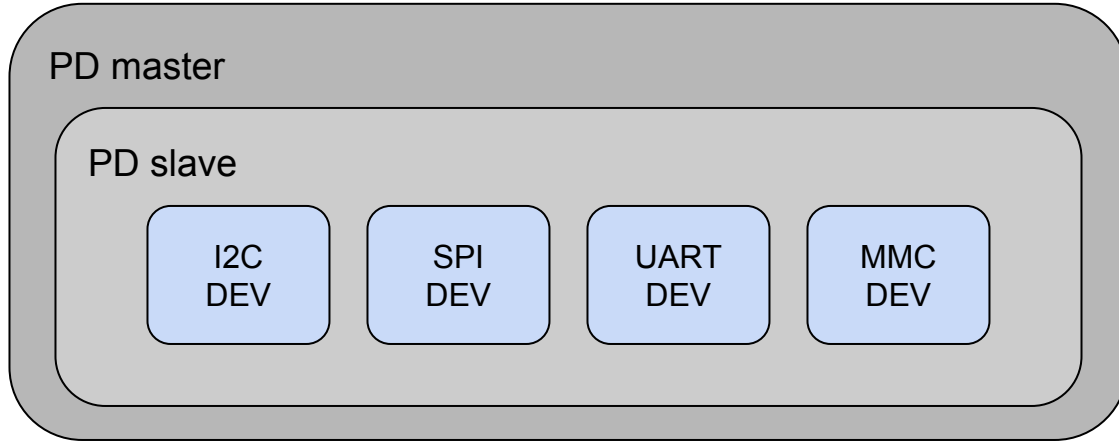
More details another time...

The generic PM domain (genpd) - overview

A generic/flexible solution for idle management of PM domains and devices.

- Widely used on ARM SoCs.
 - `drivers/base/power/domain*`, `include/linux/pm_domain.h`.
- Supports system wide sleep and runtime PM.
- PM topology described in DT, including device-nodes.
 - Supports master/subdomains.
- Interface to register a genpd provider.
 - Callbacks to power on/off a genpd.
 - Callbacks to power on/off devices.
 - Multiple idle states per genpd.
 - Configurations flags.
- Deals with attach/detach of devices - even multiple per device.
- Genpd “performance states” - another time...

Genpd - topology in DT



Documentation/devicetree/bindings/power/power_domain.txt

```
pm_domains {
    compatible = "foo,power-controller";
    pd_master: pd_master {
        #power-domain-cells = <0>;
    };
    pd_slave: pd_slave {
        power-domains = <&pd_master>;
        #power-domain-cells = <0>;
    };
};

i2c@12350000 {
    compatible = "foo,i-leak-current";
    reg = <0x12350000 0x1000>;
    power-domains = <&pd_slave>;
};

spi@12356000 {
    compatible = "bar,i-leak-current";
    reg = <0x12356000 0x1000>;
    power-domains = <&pd_slave>;
};
```

The genpd governor

Problem: When is low latency more important than wasting energy?

Option 1:

- Use runtime PM autosuspend.

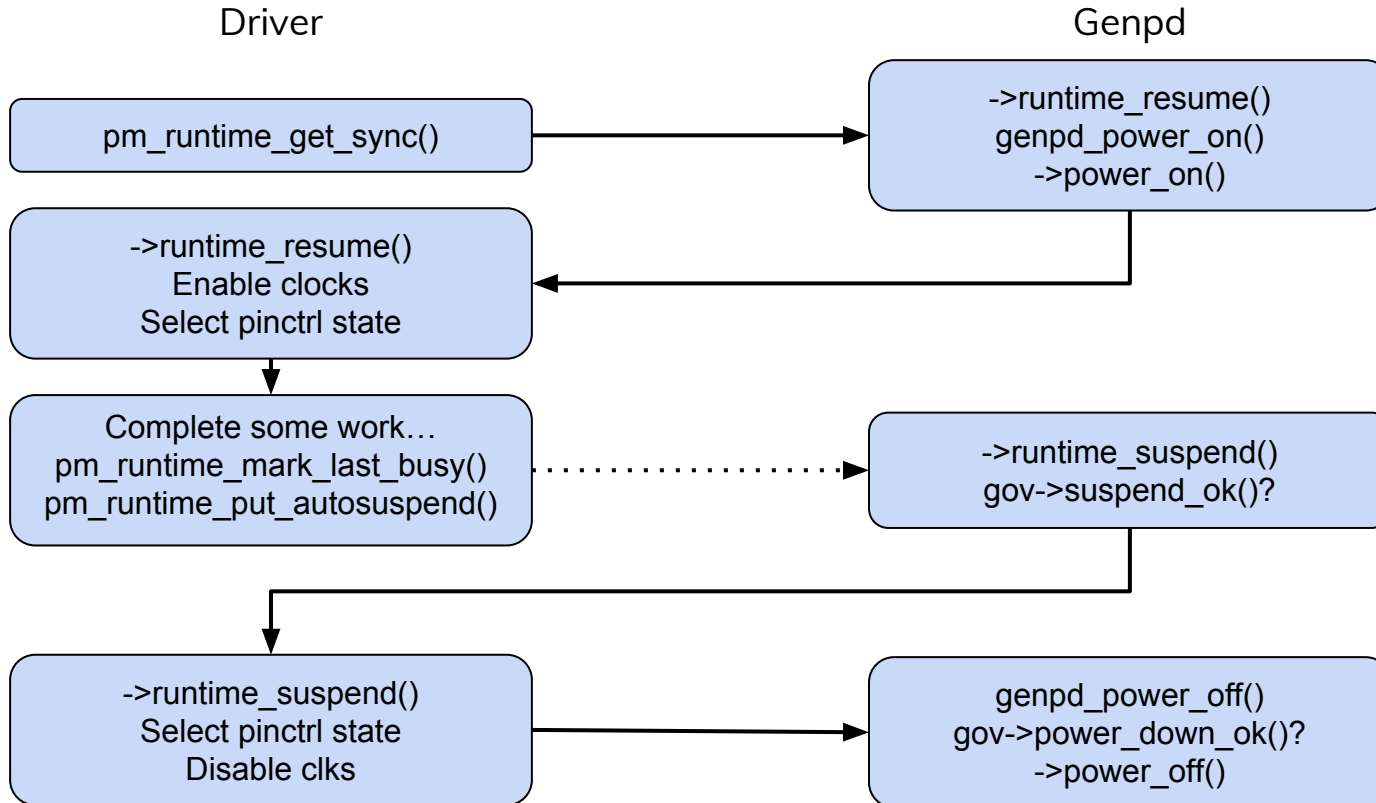
Option 2:

- Keep the device resumed.

Option 3:

- Use the genpd governor to obey to dev PM QoS constraints.

Genpd and runtime PM



Remote wakeups - for runtime PM

- Enable if supported.
 - UART - wake on console. SDIO IRQs - WiFi.
- At `->runtime_suspend()`:
 - Enable wakeup IRQ and configure the logic.
 - Disable device IRQ.
- At `->runtime_resume()`:
 - Disable wakeup IRQ and reconfigure the logic.
 - Re-enable device IRQs.

If not supported - keep device runtime resumed!

Simplified by helpers:

- `dev_pm_set_dedicated_wake_irq()` - setup the IRQ.
- The runtime PM core enables/disables the IRQ!

System wakeups - for system suspend/resume

- Enable if supported and wanted by userspace.
 - GPIO power button. UART - wake on console. SDIO IRQs - WakeOnLan.
- At `->probe()`:
 - **`device_init_wakeup()`** - to announce wakeup support:
- At `->suspend()`:
 - **`device_may_wakeup()`** - enable wakeup IRQ and configure the logic - **`enable_irq_wake()`**.
 - Disable device IRQ.
- At `->resume()`:
 - **`device_may_wakeup()`** - disable wakeup IRQ and reconfigure the logic - **`disable_irq_wake()`**.
 - Re-enable device IRQs.

Simplified by helpers:

- **`dev_pm_set_dedicated_wake_irq()`** - setup the IRQ.
- The PM core enables/disables wakeup IRQs at the “noirq” phase.

Genpd and system wakeups

Problem: Shall genpd remain powered on as to serve wakeup IRQs?

- In PM core after called `->suspend()`:
 - `dev->power.wakeup_path = device_may_wakeup()`.
 - Propagated to parent devices.
- **GENPD_FLAG_ACTIVE_WAKEUP** - a per genpd configuration flag.
 - Stay powered on to serve wakeup IRQs!
- At genpd's `->suspend_noirq()`:
 - **GENPD_FLAG_ACTIVE_WAKEUP && dev->power.wakeup_path** -> stay powered on!

Limitations: May not be fine grained enough.

- Compare out-band vs in-band wakeup logic.

Deploy PM support - what methods?

An observed method. It works, but could be tricky:

- Deploy system wide PM support.
- Deploy runtime PM support.
- Add support for wakeup settings.
- Deploy genpd support.

A smother method:

- Deploy genpd support.
- Deploy runtime PM support.
- Use the *runtime PM centric approach* - get system wide PM support for “free”.
- Deal with wake-up settings.

Deploy genpd support

1. DT documentation about your PM domain(s).
2. Update DTB and add device nodes.
3. Implement the SoC specific parts for the PM domain(s).
4. Initialize a genpd via `pm_genpd_init()`.
5. Register an genpd OF provider via `of_genpd_add_provider_simple|onecell()`.
6. Add subdomains if applicable, via `of_genpd_add_subdomain()`.

There are plenty of good examples! (32 callers of `pm_genpd_init()`)

The runtime PM centric approach - why?

Observation 1:

Operations to put a device into low power state, may be very similar during system suspend as runtime PM suspend and vice versa for resume.

- Minimize open coding.

Observation 2:

Don't runtime resume the device during system suspend and system resume, unless it's going to be used.

- Avoid wasting energy.
- Decrease system suspend time and system resume time.

The runtime PM centric approach - how?

- Re-use runtime PM callbacks for system suspend/resume
 - Call `pm_runtime_force_suspend()` from the `->suspend()` callback (or assign it).
 - Call `pm_runtime_force_resume()` from the `->resume()` callback (or assign it).

mydrv.c:

```
static const struct dev_pm_ops mydrv_dev_pm_ops = {  
    SET_SYSTEM_SLEEP_PM_OPS(pm_runtime_force_suspend, pm_runtime_force_resume)  
    SET_RUNTIME_PM_OPS(mydrv_runtime_suspend, mydrv_runtime_resume, NULL)  
};
```

There are plenty of good examples! (~120 users).

Thank you

Join Linaro to accelerate deployment of your
Arm-based solutions through collaboration

contactus@linaro.org



9Boards is a range of specifications with boards and peripherals offering different performance levels and features in a standard footprint.



**Linaro
connect**

Bangkok 2019