Don’t waste power when idle

Ulf Hansson, Linaro, PMWG

Agenda

- SoC idling - why, what, how?
- Overview of some PM frameworks.
- Deploy PM support - what methods?
- Deploy genpd support.
- Use the runtime PM centric approach and get system PM for free!
- Latest achievements and ongoing work.
- Some other interesting news.

ENGINEERS
AND DEVICES
WORKING
TOGETHER

SoC idling - why, what, how?

The why:

- Avoid wasting power when idle!

The what:

- CPUs are important, but it's not enough.
- Lots of resources/devices can enter low power state(s).

The how:

- Solve common PM issues/corner-cases.
- Enable easy PM deployment for ARM SoCs.

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Overview of some PM frameworks

- System PM
- Runtime PM
- PM domain
- The generic PM domain (aka genpd)
- Device PM QoS

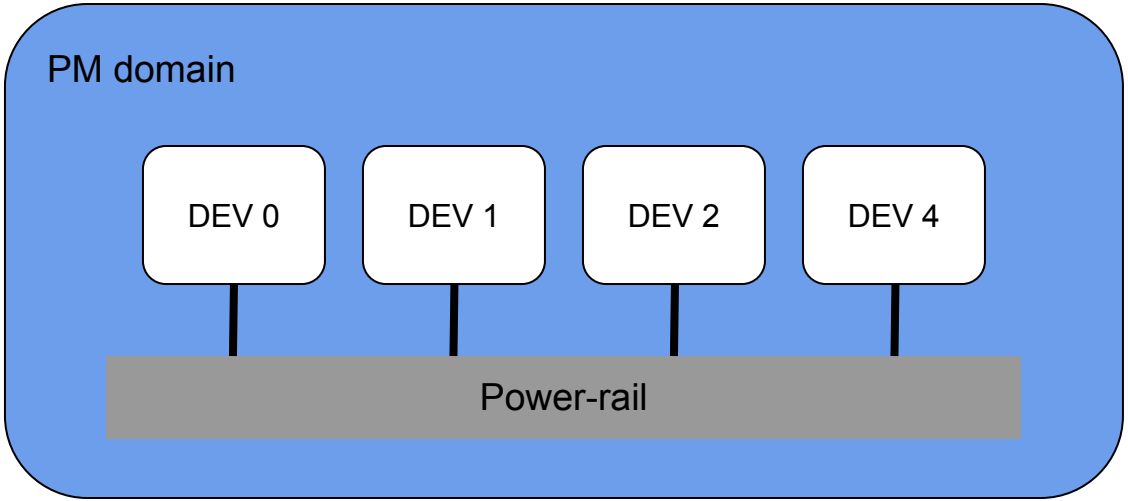
System PM vs runtime PM

- System PM - system wide with all devices.
  - Closing the lid on your laptop.
  - Pressing the power button on your Android device.
  - Autosleep - when not prevented!

- Runtime PM - fine grained for any devices.
  - At request inactivity.
  - For unused, but enabled devices.

- Historically treated as two orthogonal PM frameworks.
  - CONFIG_PM_RUNTIME was merged into the CONFIG_PM.
  - pm_runtime_force_suspend|resume() helpers was added.

PM Domains



- Devices shares common resources
- Devices needs to be managed together

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 (*freeze)(struct device *dev);
    int (*thaw)(struct device *dev);
    int (*poweroff)(struct device *dev);
    int (*restore)(struct device *dev);
    int (*suspend_late)(struct device *dev);
    int (*resume_early)(struct device *dev);
    int (*freeze_late)(struct device *dev);
    int (*thaw_early)(struct device *dev);
    int (*poweroff_late)(struct device *dev);
    int (*restore_early)(struct device *dev);
    int (*suspend_noirq)(struct device *dev);
    int (*resume_noirq)(struct device *dev);
    int (*freeze_noirq)(struct device *dev);
    int (*thaw_noirq)(struct device *dev);
    int (*poweroff_noirq)(struct device *dev);
    int (*restore_noirq)(struct device *dev);

    int (*runtime_suspend)(struct device *dev);
    int (*runtime_resume)(struct device *dev);
    int (*runtime_idle)(struct device *dev);
};
```

Sytem PM (suspend, freeze, hibernation)

Runtime PM

The hierarchy of device PM callbacks

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

ENGINEERS
AND DEVICES
WORKING
TOGETHER

The generic PM domain 1/2

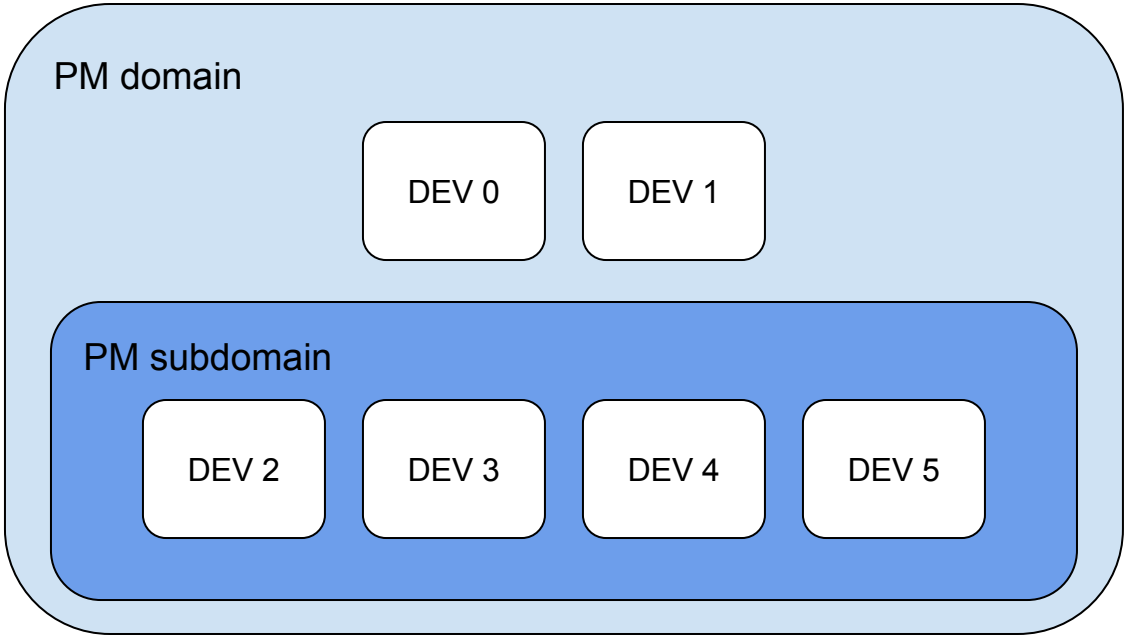
A generic solution for idle management of devices and PM domains:

- PM domain topology and the devices - described in DT.
- Grouping of devices.
- Attach/detach of device to its PM domain.
- Sub-domains and master-domains.
- SoC specific callbacks to power on/off genpd.
- SoC specific callbacks to power on/off devices.

Genpd relies on:

- Deployment of runtime PM.
- Deployment of system PM.

The generic PM domain 2/2




```
pm_domain: power-controller@12340000 {
        compatible = "foo,power-controller";
        reg = <0x12340000 0x1000>;
        #power-domain-cells = <1>;
};
pm_subdomain: power-controller@12341000 {
        compatible = "foo,power-controller";
        reg = <0x12341000 0x1000>;
        power-domains = <&pm_domain 0>;
        #power-domain-cells = <1>;
};
dev0@12350000 {
        compatible = "foo,i-leak-current";
        reg = <0x12350000 0x1000>;
        power-domains = <&pm_domain 0>;
};
...
dev2@12356000 {
        compatible = "bar,i-leak-current";
        reg = <0x12356000 0x1000>;
        power-domains = <&pm_subdomain 0>;
};
...
```

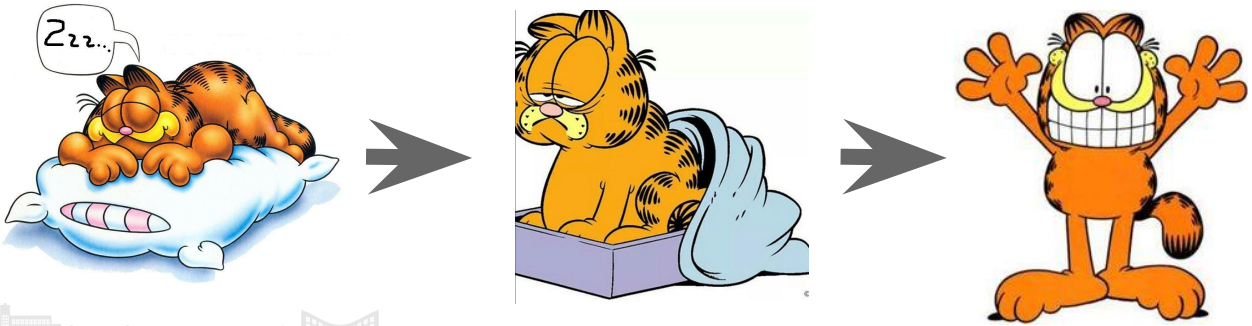
The genpd governor and dev PM QoS

Problem:

- Introduced request latency, as powering off/on a device and its PM domain, could consume a non-neglectable time.

Solution:

- Dev PM QoS to set latency constraints, which are validated by the genpd governor before power off a device.
- Or the runtime PM autosuspend option is sufficient!?



Deploy PM support - what methods?

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

1. Deploy system PM support.
2. Deploy runtime PM support.
3. Forgot to consider wake-up settings in the first steps, so adding them on top.
4. Deploy genpd support.

A better method!

1. Deploy genpd support.
2. Deploy runtime PM and use the *runtime PM centric approach* to get system PM for free. Deal with wake-up settings.

Deploy genpd support

1. DT documentation about your PM domain(s).
2. Update DTB.
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 subdomain(s) if applicable.

There are plenty of good examples!

Linaro
connect
Las Vegas 2016

ENGINEERS
AND DEVICES
WORKING
TOGETHER

The runtime PM centric approach 1/3

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

ENGINEERS
AND DEVICES
WORKING
TOGETHER

The runtime PM centric approach 2/3

1. Deploy runtime PM.
2. Deal with wake-ups.

Option 1)
Assign the system PM callbacks to pm_runtime_force_suspend|resume()

Option 2:
Call pm_runtime_force_suspend|resume() from your own system PM callbacks.

ENGINEERS
AND DEVICES
WORKING
TOGETHER

The runtime PM centric approach 3/3

mydrv.c: (Option 1)

```
...
static const struct dev_pm_ops mydrv_dev_pm_ops = {
        SET_SYSTEM_SLEEP_PM_OPS(pm_runtime_force_susp
end,
                                        pm_runtime_force_resume)
        SET_RUNTIME_PM_OPS(mydrv_ runtime_suspend,
                                mydrv_runtime_resume,
                                NULL)
};
...
```

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Latest achievements

- Genpd: Don't unnecessary power up devices during system PM *suspend*, just to shortly after power off them again.

->Decreases system PM suspend time and avoid wasting power.

- Genpd/Runtime PM: Don't unnecessary power up devices during system PM *resume*, just to shortly after power off them again.
  - Last patch in series caused a regression, so it was dropped. We are working on this.

-> Decreases system PM resume time and avoid wasting power.

- Genpd: Support to remove genpds.

Ongoing work 1/2

- Enable genpd to collect power statistics.
  - Useful when power can't be measured.
- Runtime PM deployment in the core part of subsystems.
  - Deployed: mmc, spi, block, phy, usb, etc
  - Discussed: clock, irqchip, DMA, IOMMU etc
- Enable options to describe functional dependencies between devices, from probing and system/runtime PM point of view.
  - This is *not* about child-parents relationships, but more soft dependencies between devices.
  - https://lwn.net/Articles/700930/

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Ongoing work 2/2

- A unified solution to manage idle across all kind of devices, including CPUs.
  - CPUIdle framework doesn't scale for multi-cluster SMP systems and heterogeneous systems like big.LITTLE.
  - We are extending runtime PM and genpd to also cover CPUs. Genpd already provides most of the needed building blocks to deploy this solution and we get unified solution to manage idle across all kind of devices.
  - **PMWG tree has two ARM64 SoCs being tested. We can add more!**
  - Extremely easy to deploy for ARM64. DTB updates only.
  - More discussions at LPC in November.

Some interesting news

- The usage of genpd keeps increasing.
  - v3.18: 5 callers of pm_genpd_init()
  - v4.5: 14
  - v4.8: 18
- The runtime PM centric approach is being adopted.
  - v3.18: 7 users of pm_runtime_force_suspend|resume()
  - v4.5: 16
  - v4.8: 25

Thank You!

ulf.hansson@linaro.org

#LAS16

For further information: www.linaro.org

LAS16 keynotes and videos on: connect.linaro.org