Last One Out, Turn Off The Lights

Embedded Linux Conference 2015

Geert Uytterhoeven
geert@linux-m68k.org

Glider bvba

Tuesday, March 24

Glider.be

Table of Contents

About Me (and Linux)

Hobbyist

1994 Linux/m68k on Amiga

1997 Linux/PPC on CHRP

1997 FBDev

Sony

2006 Linux on PS3/Cell

SONY

Glider bvba

2013 Renesas ARM-based SoCs

RENESAS

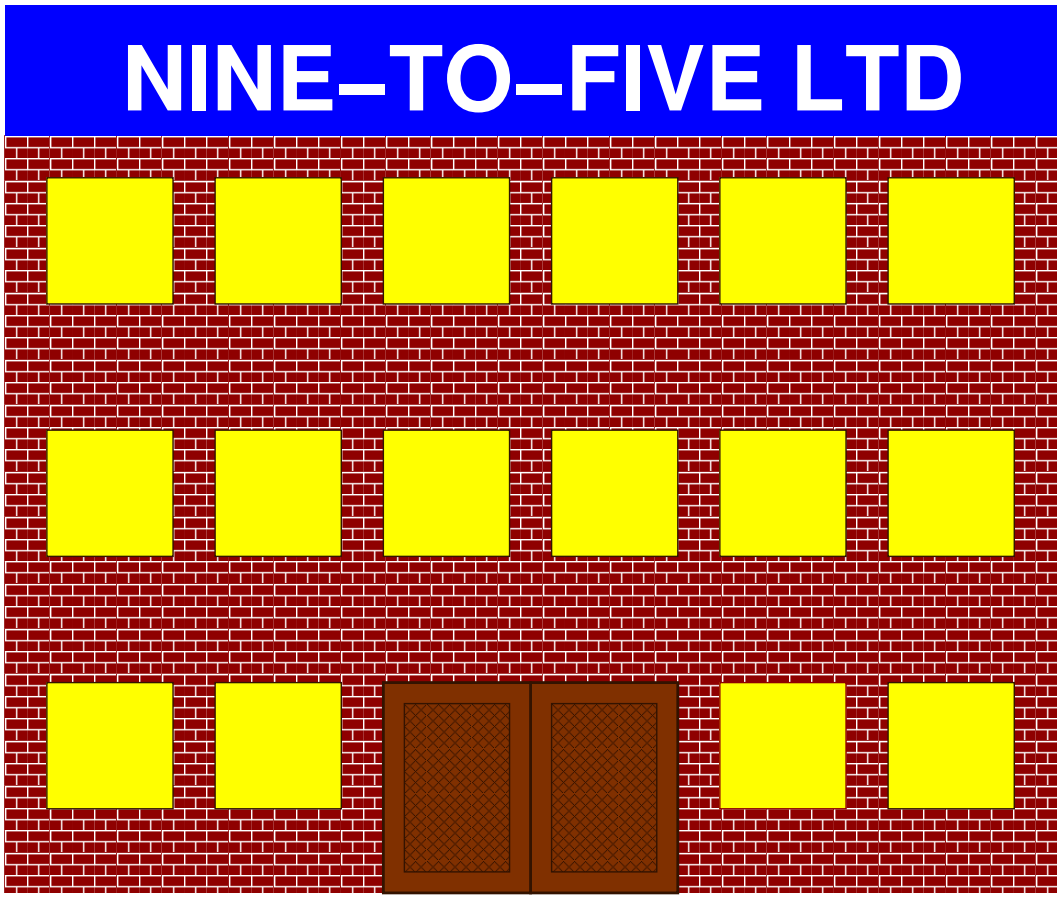
System-Centric Power Management

PC

- On
- Suspended
- Hibernated
- Off

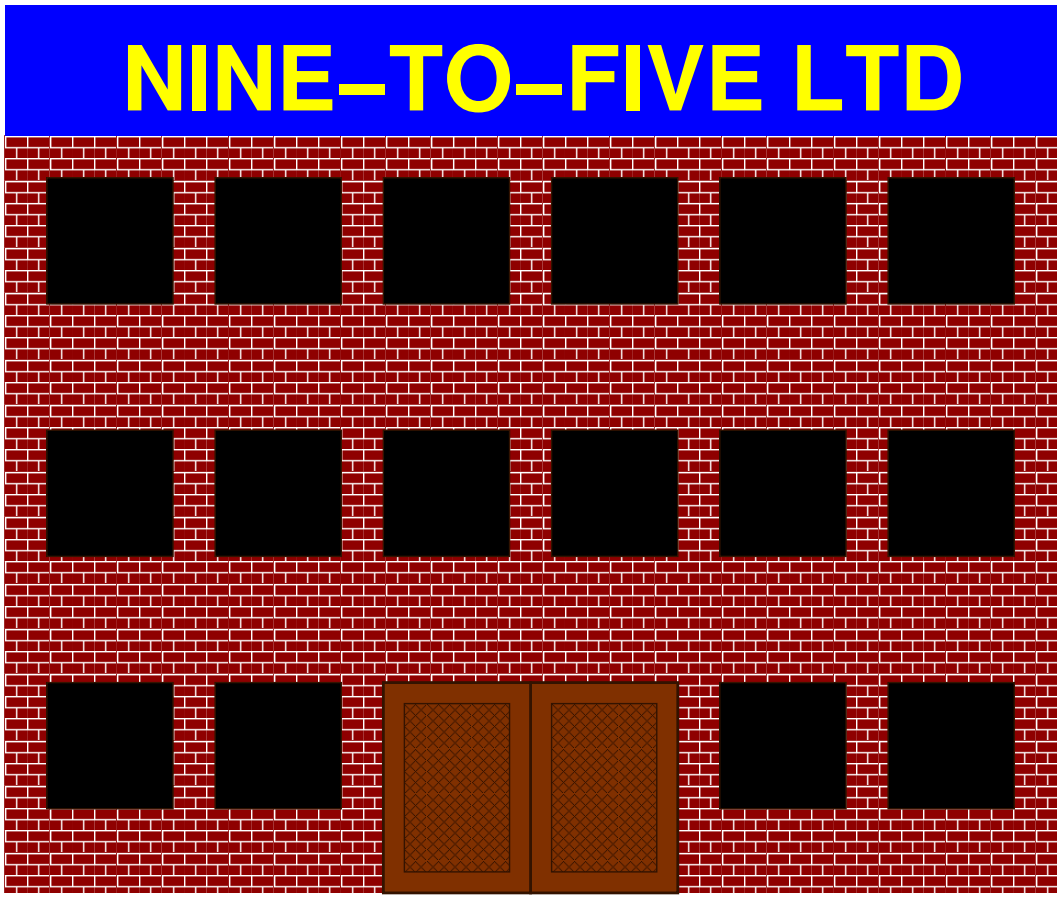
System-Centric Power Management

Building: Day Time



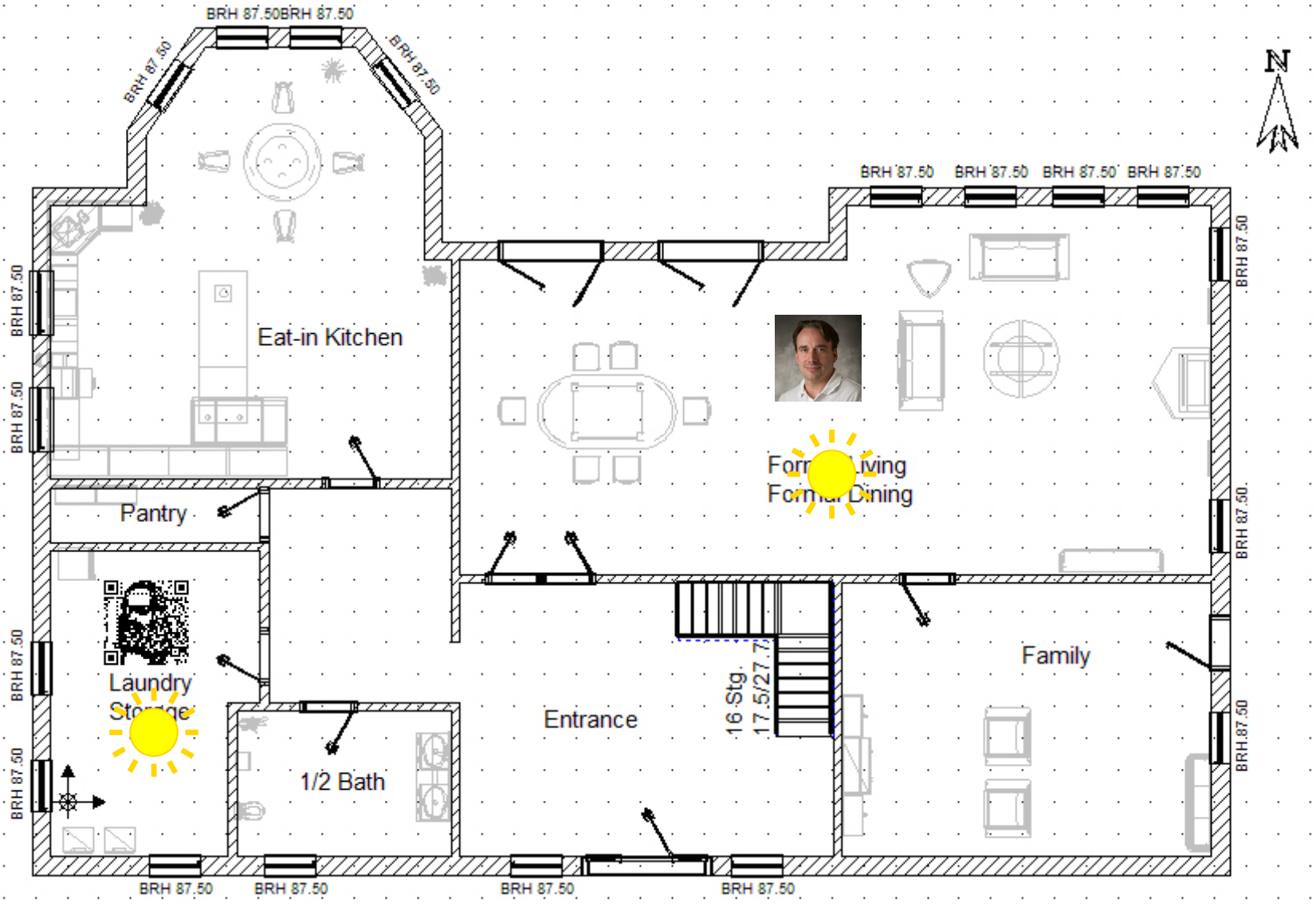
System-Centric Power Management

Building: Night Time



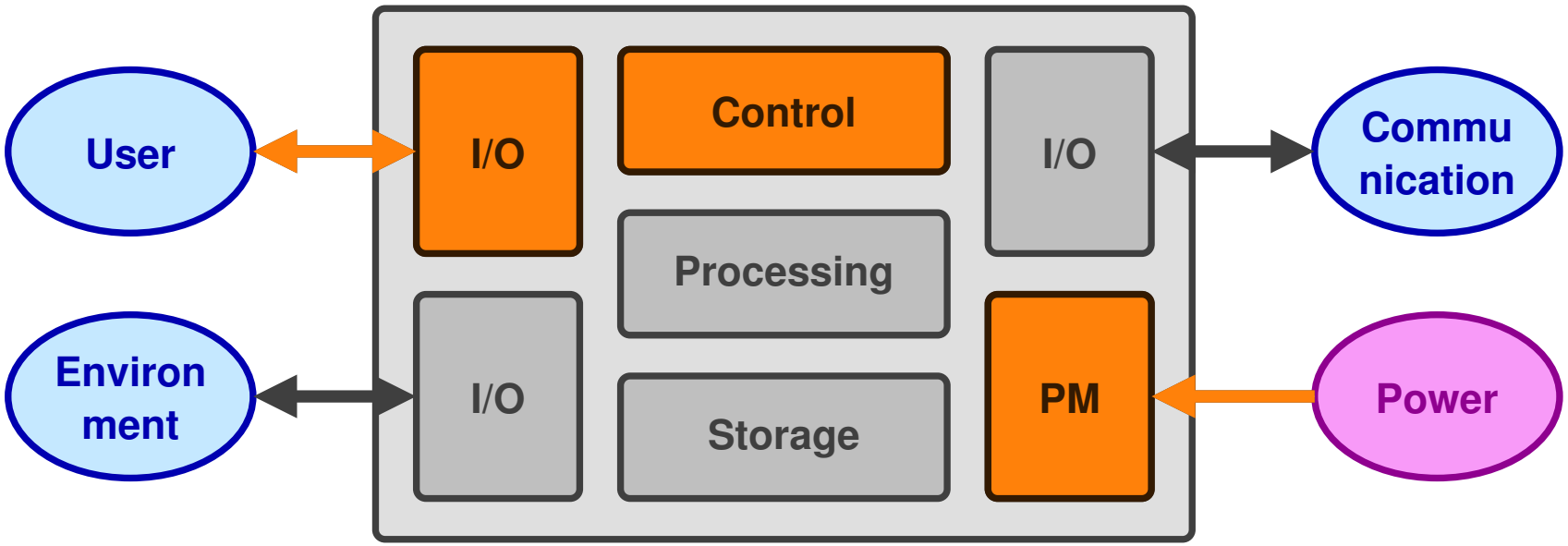
Runtime Power Management

Building



Runtime Power Management

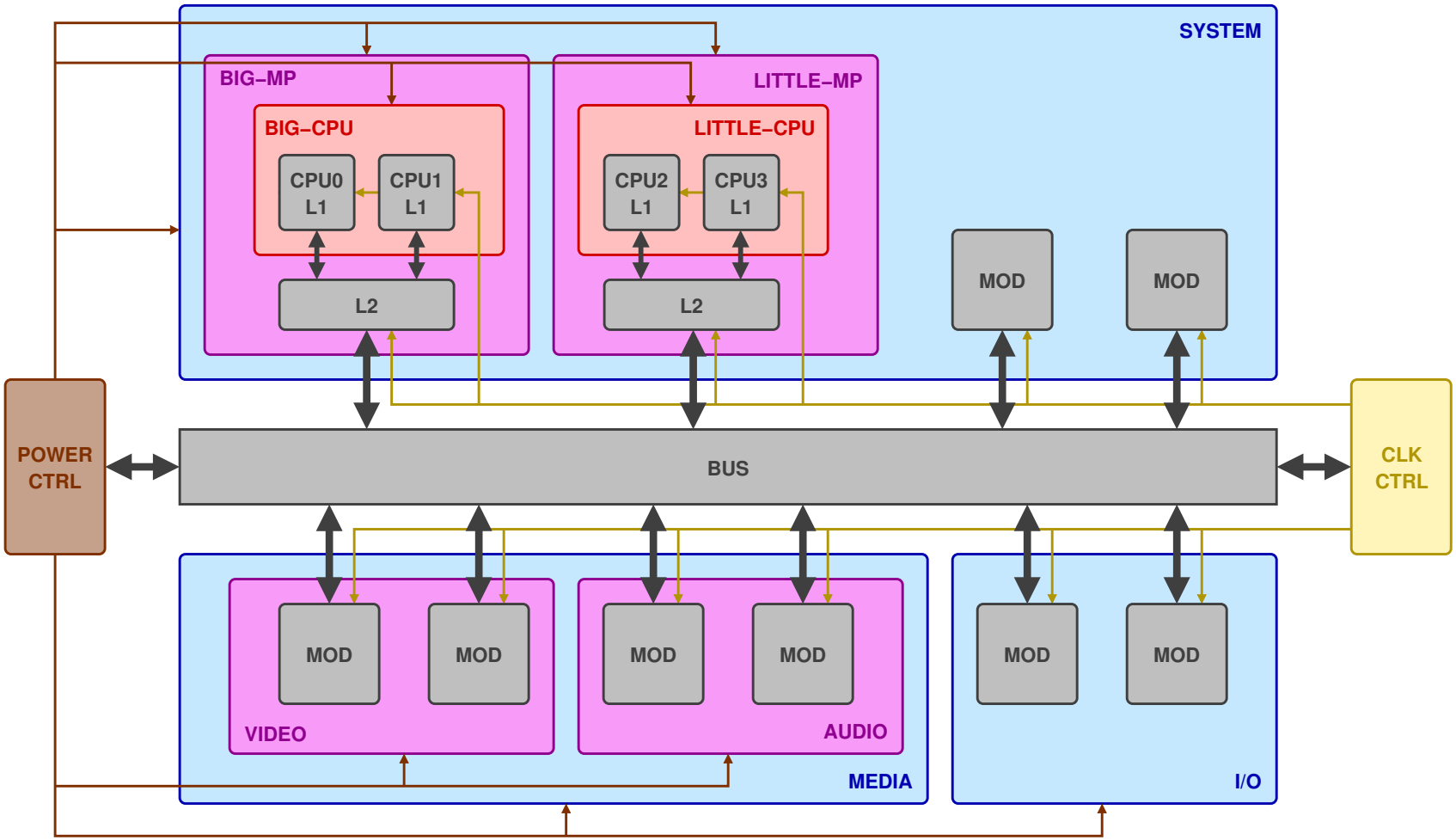
Computing Device



- Parts become active when needed
- Parts become inactive when no longer needed

Hardware / SoC

Multiple Power Areas, Clock Domain



Power Domains

- Devices can be Power Managed by Controlling Power
- Multiple Power Domains / Power Areas
- Power Controllers

Clock Domains

- Devices can be Power Managed by Controlling Clocks
  - Synchronous logic driven by clock
  - Gating the clock saves power
- Clock Controllers

Power/Clock Distribution

- Per-device
- Device groups

Power/Clock Topology

- Flat
- Tree
- Complex hierarchy

Dependencies!

- *PM Domains* **not** *Power Domains*
- Not limited to Power Domains / Power Areas

- **PM Domain =** Collection of devices treated similarly w.r.t. power management
  - One single power area
  - One clock controller for power-managing per-device clocks
  - Combination
  - Firmware (e.g. ACPI)
  - ...

- Generic I/O PM Domains (genpd)
- Generic implementations of various device PM callbacks
- Supports controlling an entire PM Domain
- Supports controlling a single device in a PM Domain
- Supports PM Subdomains
- `select PM_GENERIC_DOMAINS if PM`

Note: Other PM Domains (e.g. ACPI)

- Allows I/O devices to be put into energy-saving states
- After a specified period of inactivity
- Woken up in response to a hardware-generated wake-up event or a driver's request
- Used to have its own config symbol (`CONFIG_PM_RUNTIME`)
- Always enabled if `CONFIG_PM_SLEEP` is enabled since v3.19

More Linux PM?

- Cpuidle: Multiple CPU idle levels
- Cpufreq: CPU frequency and voltage scaling
  - Cfr. breathing, heartbeat, ...
- ...

$\Rightarrow$ *Introduction to Kernel Power Management* by Kevin Hilman

Example: r8a7740/armadillo800eva

```
$ cat /sys/kernel/debug/pm_genpd/pm_genpd_summary
```

(CONFIG_PM_DEBUG=y and CONFIG_PM_ADVANCED_DEBUG=y)

```
    domain                          status          slaves
           /device                                              runtime status
-------------------------------------------------------------------------------
a4su                                off
a3sg                                off
a3sm                                on
a3sp                                on
    /devices/platform/e6600000.pwm                               suspended
    /devices/platform/e6c50000.serial                            active
    /devices/platform/e6850000.sd                                active
    /devices/platform/e6bd0000.mmc                               active
a4s                                 on              a3sp, a3sm, a3sg
    /devices/platform/e6900000.irqpin                            active
    /devices/platform/e6900004.irqpin                            active
    /devices/platform/e6900008.irqpin                            active
    /devices/platform/e690000c.irqpin                            active
    /devices/platform/e9a00000.ethernet                          active
a3rv                                off
a4r                                 on              a3rv
    /devices/platform/fff20000.i2c                               suspended
    /devices/platform/fff80000.timer                             active
d4                                  on
a4mp                                off
    /devices/platform/fe1f0000.sound                             suspended
a4lc                                off
c5                                  on              a4lc, a4mp, d4, a4r, a4s, a4su
    /devices/platform/e6050000.pfc                               unsupported
    /devices/platform/e6138000.timer                             suspended
```

Linux API

<linux/pm.h>

```
/* struct dev_pm_ops - 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 (*runtime_suspend)(struct device *dev);
        int (*runtime_resume)(struct device *dev);
        ...
};
```

Can be:

- Bus specific (`struct bus_type.pm`)
- Device driver specific (`struct device_driver.pm`)
- Device class specific (`struct class.pm`)
- Device type specific (`struct device_type.pm`)
- **PM Domain specific (`struct dev_pm_domain.pm`)**
- Platform specific

Linux API

<linux/pm.h>

```
/*
 * Power domains provide callbacks that are executed during
 * system suspend, hibernation, system resume and during
 * runtime PM transitions along with subsystem-level and
 * driver-level callbacks.
 */
struct dev_pm_domain {
        struct dev_pm_ops ops;
        void (*detach)(struct device *dev, bool power_off);
};
```

- Used by Devices (`struct device.pm_domain`)
- Provided by:
  - Generic PM Domain (`struct generic_pm_domain.domain`)
  - Platform code, Legacy Clock Domains, VGA switcheroo
  - ACPI

Linux API

<linux/pm_domain.h>

```
struct dev_power_governor {
        bool (*power_down_ok)(struct dev_pm_domain *domain);
        bool (*stop_ok)(struct device *dev);
};
```

```
struct gpd_dev_ops {
        int (*start)(struct device *dev);
        int (*stop)(struct device *dev);
        int (*save_state)(struct device *dev);
        int (*restore_state)(struct device *dev);
        bool (*active_wakeup)(struct device *dev);
};
```

Linux API

<linux/pm_domain.h>

```
#define GENPD_FLAG_PM_CLK (1U << 0)          /* Use PM clk */

struct generic_pm_domain {
        struct dev_pm_domain domain;        /* PM domain ops */
        struct list_head gpd_list_node;     /* Global list */
        ...
        const char *name;
        ...
        enum gpd_status status;      /* Current state */
        ...
        int (*power_off)(struct generic_pm_domain *domain);
        s64 power_off_latency_ns;
        int (*power_on)(struct generic_pm_domain *domain);
        s64 power_on_latency_ns;
        struct gpd_dev_ops dev_ops;
        ...
        int (*attach_dev)(struct generic_pm_domain *domain,
                          struct device *dev);
        void (*detach_dev)(struct generic_pm_domain *domain,
                           struct device *dev);
        unsigned int flags;          /* Bit field of configs */
};
```

Linux API

Setting up PM Domains

```
void pm_genpd_init(struct generic_pm_domain *genpd,
                   struct dev_power_governor *gov,
                   bool is_off);

int pm_genpd_add_subdomain(struct generic_pm_domain *genpd,
                   struct generic_pm_domain *new_subdomain);
int pm_genpd_remove_subdomain(struct generic_pm_domain *genpd,
                   struct generic_pm_domain *target);

struct genpd_onecell_data {
        struct generic_pm_domain **domains;
        unsigned int num_domains;
};

int of_genpd_add_provider_simple(struct device_node *np,
                                 struct generic_pm_domain *genpd);
int of_genpd_add_provider_onecell(struct device_node *np,
                                 struct genpd_onecell_data *data);
void of_genpd_del_provider(struct device_node *np);
struct generic_pm_domain *of_genpd_get_from_provider(
                            struct of_phandle_args *genpdspec);
```

Linux API

Populating PM Domains

```
int __pm_genpd_add_device(struct generic_pm_domain *genpd,
                          struct device *dev,
                          struct gpd_timing_data *td);
int pm_genpd_remove_device(struct generic_pm_domain *genpd,
                           struct device *dev);

int __pm_genpd_name_add_device(const char *domain_name,
                               struct device *dev,
                               struct gpd_timing_data *td);
int pm_genpd_add_subdomain_names(const char *master_name,
                                 const char *subdomain_name);
```

$\Rightarrow$ Use DT!

Linux API

Cpuidle

```
int pm_genpd_attach_cpuidle(struct generic_pm_domain *genpd,
                            int state);
int pm_genpd_name_attach_cpuidle(const char *name, int state);
int pm_genpd_detach_cpuidle(struct generic_pm_domain *genpd);
int pm_genpd_name_detach_cpuidle(const char *name);
```

So far used only by legacy (non-DT) SH-Mobile AP4 (sh7372), which is scheduled for removal in v4.1...

Device Tree

- Preferred way to describe hardware PM Domains
  - PM Domain Providers are registered by platform code
  - PM Domain Consumers are registered by PM Domain core

- Introduced last year, not that many users yet:
  - Freescale i.MX6
  - Renesas SH-Mobile/R-Mobile
  - Samsung Exynos
  - ST-Ericsson Ux500

  However, more to come soon!

Device Tree

PM Domain Providers

- Required properties:

```
#power-domain-cells : Number of cells in a PM domain
                      specifier;
```

  - 0 for nodes representing a single PM domain
  - 1 for nodes providing multiple PM domains (power controllers)
  - can be any value as per provider DT bindings

- Example:

```
power: power-controller@12340000 {
        compatible = "foo,power-controller";
        reg = <0x12340000 0x1000>;
        #power-domain-cells = <1>;
};
```

Documentation/devicetree/bindings/power/power_domain.txt

- Required properties:

```
power-domains : A phandle and PM domain specifier as
                defined by bindings of the power
                controller specified by phandle.
```

- Example:

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

`Documentation/devicetree/bindings/power/power_domain.txt`

Device Tree

PM Domain Providers and Consumers Example

```
pd_lcd0: lcd0-power-domain@10023C80 {
        compatible = "samsung,exynos4210-pd";
        reg = <0x10023C80 0x20>;
        #power-domain-cells = <0>;
};

dsi_0: dsi@11C80000 {
        compatible = "samsung,exynos4210-mipi-dsi";
        reg = <0x11C80000 0x10000>;
        interrupts = <0 79 0>;
        power-domains = <&pd_lcd0>;
        phys = <&mipi_phy 1>;
        phy-names = "dsim";
        clocks = <&clock CLK_DSIM0>, <&clock CLK_SCLK_MIPI0>;
        clock-names = "bus_clk", "pll_clk";
        status = "disabled";
        #address-cells = <1>;
        #size-cells = <0>;
};
```

arch/arm/boot/dts/exynos4.dtsi

Device Tree

Parent/Child PM Domain Providers Example

```
parent: power-controller@12340000 {
        compatible = "foo,power-controller";
        reg = <0x12340000 0x1000>;
        #power-domain-cells = <1>;
};

child: power-controller@12341000 {
        compatible = "foo,power-controller";
        reg = <0x12341000 0x1000>;
        power-domains = <&parent 0>;
        #power-domain-cells = <1>;
};
```

`Documentation/devicetree/bindings/power/power_domain.txt`

Device Tree

Hierarchical PM Domains Example

```
sysc: system-controller@e6180000 {
        compatible = "renesas,sysc-r8a7740", "renesas,sysc-rmobile";
        reg = <0xe6180000 0x8000>, <0xe6188000 0x8000>;

        pm-domains {
                pd_c5: c5 {
                        #address-cells = <1>;
                        #size-cells = <0>;
                        #power-domain-cells = <0>;

                        pd_a4s: a4s@10 {
                                reg = <10>;
                                #address-cells = <1>;
                                #size-cells = <0>;
                                #power-domain-cells = <0>;

                                pd_a3sp: a3sp@11 {
                                        reg = <11>;
                                        #power-domain-cells = <0>;
                                };
                        };

                        pd_a4su: a4su@20 {
                                reg = <20>;
                                #power-domain-cells = <0>;
                        };
                };
        };
};
```

Documentation/devicetree/bindings/power/renesas,sysc-rmobile.txt

Platform Code

Single PM Domain Example

```
static int my_power_off(struct generic_pm_domain *genpd);
static int my_power_on(struct generic_pm_domain *genpd);

static __init int init_my_power_domain(void)
{
    struct device_node *np;

    for_each_compatible_node(np, NULL, "my-vendor,my-power") {
        struct generic_pm_domain *pd = ...;
        ...
        pd->name = np->name;
        pd->power_off = my_power_off;
        pd->power_on = my_power_on;
        pm_genpd_init(pd, NULL, false);
        of_genpd_add_provider_simple(np, pd);
    }
    return 0;
}

arch_initcall(init_my_power_domain);
```

Platform Code

Multiple PM Domains Example

```
static __init int init_my_power_controller(void)
{
    struct device_node *np;

    for_each_compatible_node(np, NULL, "my-vendor,my-power") {
        struct genpd_onecell_data *data = ...;

        data.domains = ...;
        data.num_domains = ...;
        ...
        for (i = 0; i < data.num_domains; i++)
                pm_genpd_init(data.domains[i], NULL, false);
        of_genpd_add_provider_onecell(np, data);
    }
    return 0;
}

arch_initcall(init_my_power_controller);
```

Platform Code

Clock Domain Example

```
static int my_attach_dev(struct generic_pm_domain *domain,
                         struct device *dev)
{
        pm_clk_create(dev);
        pm_clk_add(dev, ...);  /* Find and add module clock */
}

static void my_detach_dev(struct generic_pm_domain *domain,
                          struct device *dev)
{
        pm_clk_destroy(dev);
}

static __init int init_my_clock_domain(void)
{
        ...
        pd->attach_dev = my_attach_dev;
        pd->detach_dev = my_detach_dev;
        /* dev_ops.{start,stop} = pm_clk_{suspend,resume}() */
        pd->flags = GENPD_FLAG_PM_CLK;
        ...
}
```

Device Drivers

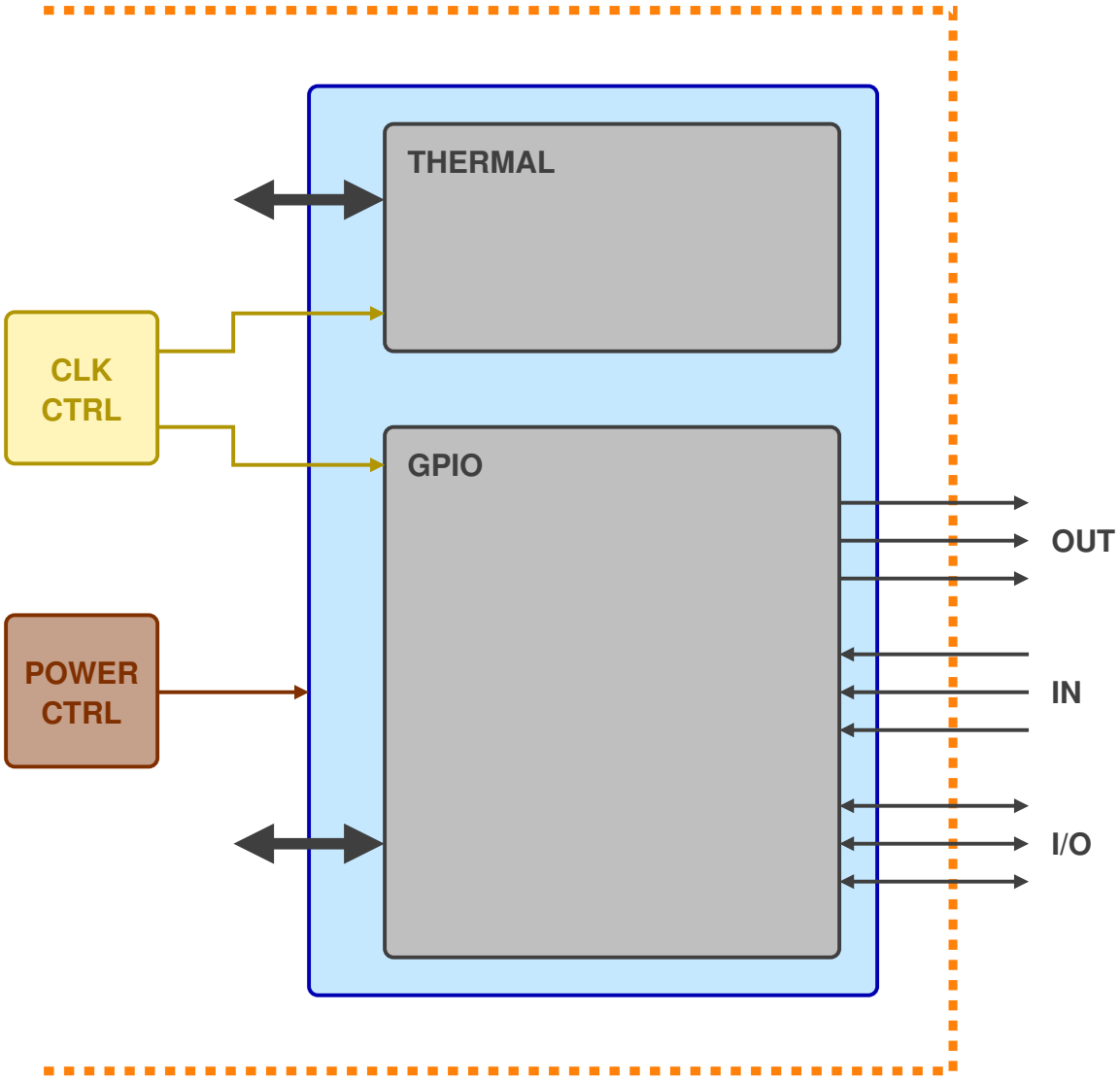
- Ideally, device drivers should not be aware of PM Domains
- Abstracted by Runtime PM

- Power Domains: Never accessed directly by drivers
  - Module needs to be powered when active
  - Automatic, Runtime PM

- Clock Domains: Who is in charge of the clocks(s)?
  - Functional clocks
  - Interface clocks
  - Clock-agnostic (hardware is *just* synchronous?)
  - Clock rate
  - ...

Device Drivers

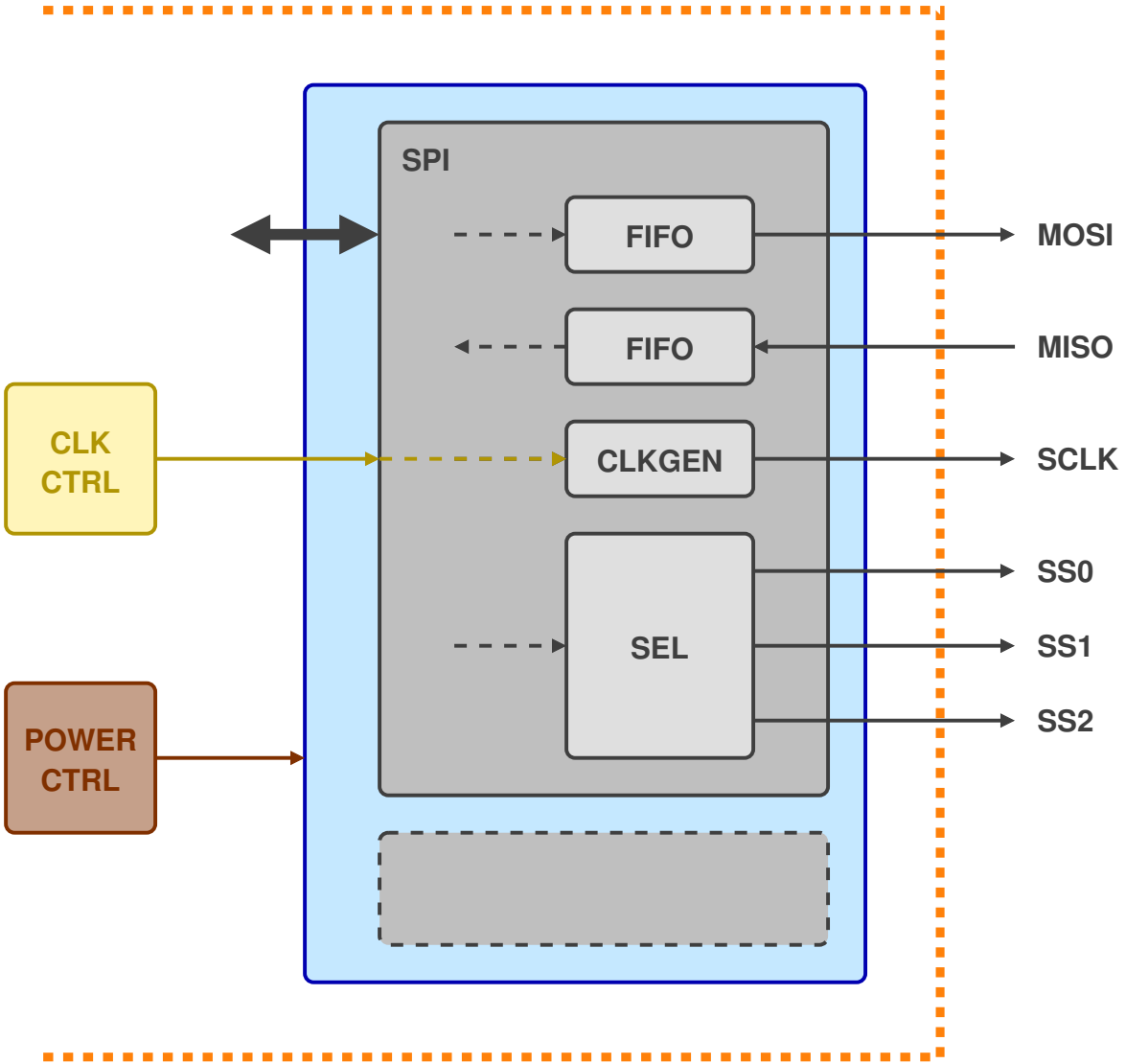
Example: Thermal and GPIO Modules



- Needs power
- Clock-agnostic
- Input may need clock

Device Drivers

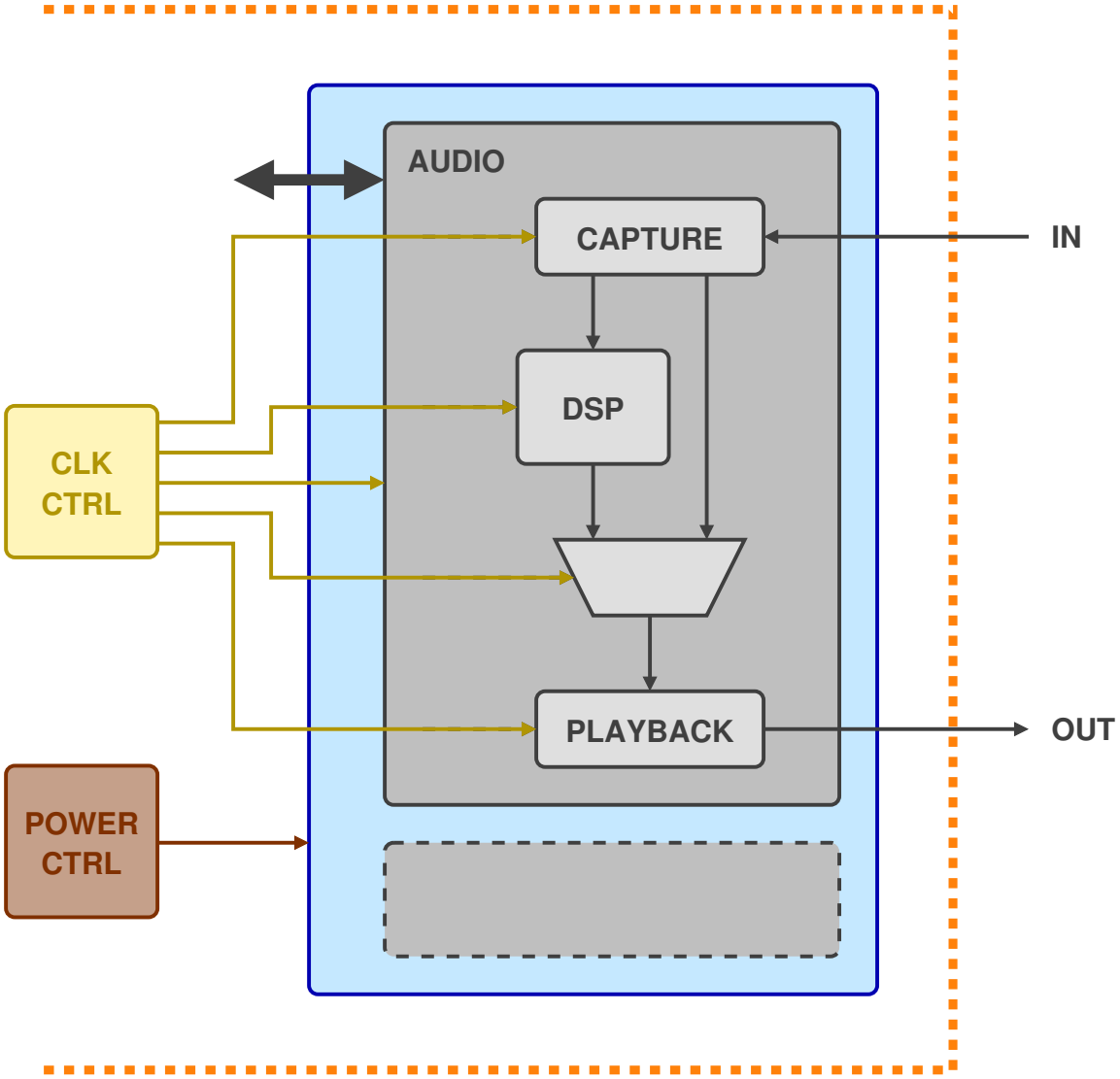
Example: SPI Master Module



- Needs power, clock
- Needs to know clock rate
- Who is in charge of the clock?

Device Drivers

Example: Audio Module



- Capture at 32 kHz
- Playback at 44.1 kHz
- Audio processing
- Needs power, clock
- Driver is in charge of the clocks

Device Drivers

Runtime PM

- Runtime PM is disabled by default
- Driver needs minimal Runtime PM:

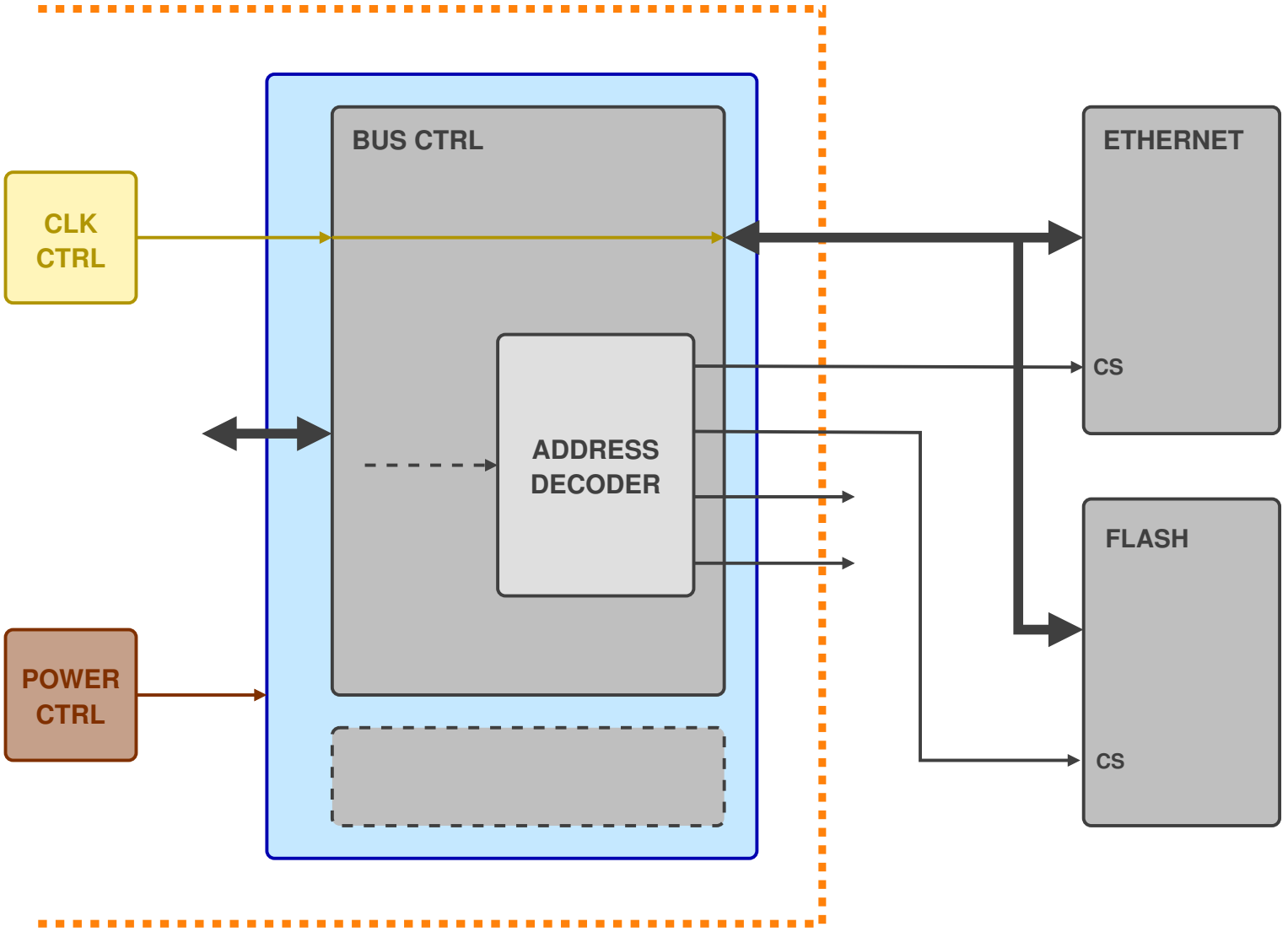
```
#include <linux/pm_runtime.h>

...

static int my_probe(struct platform_device *pdev)
{
        ...
        pm_runtime_enable(&pdev->dev);
        pm_runtime_get_sync(&pdev->dev);
        ...
}

static int my_remove(struct platform_device *pdev)
{
        ...
        pm_runtime_put(&pdev->dev);
        pm_runtime_disable(&pdev->dev);
        ...
}
```

- Better: more advanced Runtime PM
  - Call `pm_runtime_put()` after becoming inactive,
  - Call `pm_runtime_get_sync()` before becoming active.
- May be subsystem-specific
  - E.g. SPI core handles this automatically if `spi_master.auto_runtime_pm == true`
- Provide your own `struct dev_pm_ops`

Caveats

- Unused PM Domains will be powered down by the genpd core (cfr. clocks)
- If you make a mistake, something will break, eventually
  - Incorrect description in DT
  - Driver / subsystem without / with incorrect Runtime PM
  - ...
- Shared PM Domain: it may work by accident
- Wake-up

Example: Simple Power Managed Bus



Example: Simple Power Managed Bus

Original

```
bsc: bus@fec10000 {
        compatible = "simple-bus";
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0 0x20000000>;

        ethernet@10000000 {
                compatible = "smsc,lan9220", "smsc,lan9115";
                reg = <0x10000000 0x100>;
                ...
        };
};
```

- X Missing BSC clock, broke when CCF was introduced
  Bad workaround: add BSC clock to ethernet node
- X Even more broken with the advent of PM Domains

Example: Simple Power Managed Bus

Solution 1: Add Clock, PM Domain

```
bsc: bus@fec10000 {
        compatible = "simple-bus";
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0 0x20000000>;
        clocks = <&zb_clk>;
        power-domains = <&pd_a4s>;

        ethernet@10000000 {
                compatible = "smsc,lan9220", "smsc,lan9115";
                reg = <0x10000000 0x100>;
                ...
        };
};
```

- ✓ Ethernet driver now has minimal Runtime PM support
- ✓ Runtime PM takes into account parent/child relationship
- X Without a BSC driver, Runtime PM stays disabled, and the PM Domain is not powered on.

Example: Simple Power Managed Bus

Solution 2: Generic "simple-pm-bus" Bindings and Driver

```
bsc: bus@fec10000 {
        compatible = "renesas,bsc-sh73a0", "renesas,bsc",
                     "simple-pm-bus";
...
        clocks = <&zb_clk>;
        power-domains = <&pd_a4s>;

        ethernet@10000000 {
                compatible = "smsc,lan9220", "smsc,lan9115";
                ...
        };
};
```

- ✓ Ethernet driver now has minimal Runtime PM support
- ✓ Runtime PM takes into account parent/child relationship
- ✓ "simple-pm-bus" driver has minimal Runtime PM support
- ⇒ PM Domain is managed correctly

Caveats

DT describes the hardware, not the behavior

X

- *This PM Domain must not be powered down*
- `"always-on"` property in DT
- Prohibits a proper future solution

✓

- Ask yourself: why must this PM Domain not be powered down?
- Reference PM domain from (new) device node that uses it
- Have a driver or platform code that powers up the PM domain (and keeps it powered up)

Examples: Special Devices, No Runtime PM Support

- Scan DT topology to find PM Domains containing special devices
- Handle in platform code:
  - Protect against runtime suspend:

```
pm_genpd_init(..., &pm_domain_always_on_gov, ...);
```

  - Protect against system suspend:

```
static int power_off_always_busy(void)
{
        /* This domain should not be turned off */
        return -EBUSY;
}

genpd->power_off = power_off_always_busy;
```

- Hopefully a temporary solution!

Examples: Special Devices, No Runtime PM Support

CPUs

- Scan DT for device nodes under "`/cpus`"
- Avoid power down while the CPU is busy
- Optional: Handle cpuidle

Serial Console

- `no_console_suspend`
- `/sys/module/printk/parameters/console_suspend`
- `chosen/stdout-path`
- `struct device_node *of_stdout`

```
static int power_off_console_busy(void)
{
        /* Keep the PM Domain on if "no_console_suspend" is set */
        return console_suspend_enabled ? 0 : -EBUSY;
}

genpd->power_off = power_off_console;
```

Examples: Special Devices, No Runtime PM Support

ARM Coresight-ETM (Debug)

- `arch/arm/kernel/hw_breakpoint.c` accesses debug registers unconditionally
- Add minimal device node for Coresight-ETM
- Scan DT for `"arm,coresight-etm3x"`

Memory Controllers

- No driver for memory controller
- Add minimal device node for memory controller
- Scan DT for known memory controllers

Challenges

DMA and IOMMUs

- DMA mappings are typically created during device probe
- Runtime PM only knows about active devices, not about active DMA mappings
- When to suspend/resume DMA engines/channels?
- When to suspend/resume IOMMUs?

Thanks & Acknowledgements

- **Renesas Electronics Corporation**, for contracting me to do Linux kernel work,
- The **Linux Foundation**, for organizing this conference and giving me the opportunity to present here,
- The **Renesas Linux Kernel Team**, for insights and discussions,
- The **Linux Kernel Community**, for having so much fun working together towards a common goal.

Questions?