Zephyr™ Project
Developer Summit
June 8-10, 2021 • @ZephyrIoT

Zephyr Power Management - 101

FLAVIO CEOLIN
FLAVIO.CEOLIN@INTEL.COM

Zephyr™ Project
Developer Summit

"Power management is a feature that turns off the power or switches the system to a low-power state when inactive"

Agenda

- Goals
- Power Management on Zephyr
  - System Power Management
  - Device Power Management
- Enabling
  - How to use it in a target
  - How to implement or customize Power Management
- Conclusions

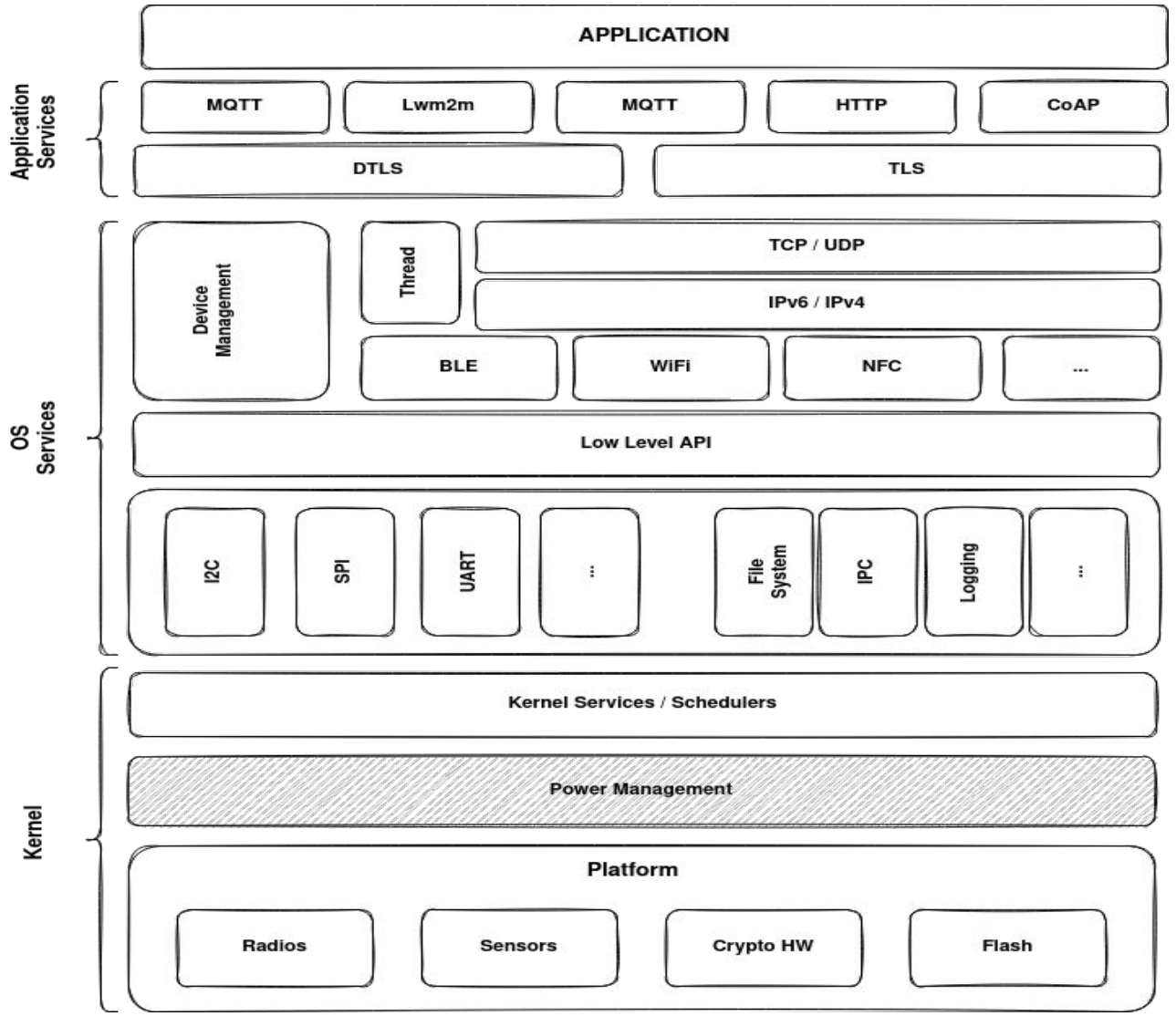
Goals

- Consume **as little power as possible** for a given **use case** and **configuration**.
  - Do not waste energy when **idle**.
- **Customizable** - provide the mechanism, not the policy.
- Cross platform (architecture / board / SoC).
- Convenient to **use** and **deploy**.

Zephyr™ Project
Developer Summit

Power Management on Zephyr

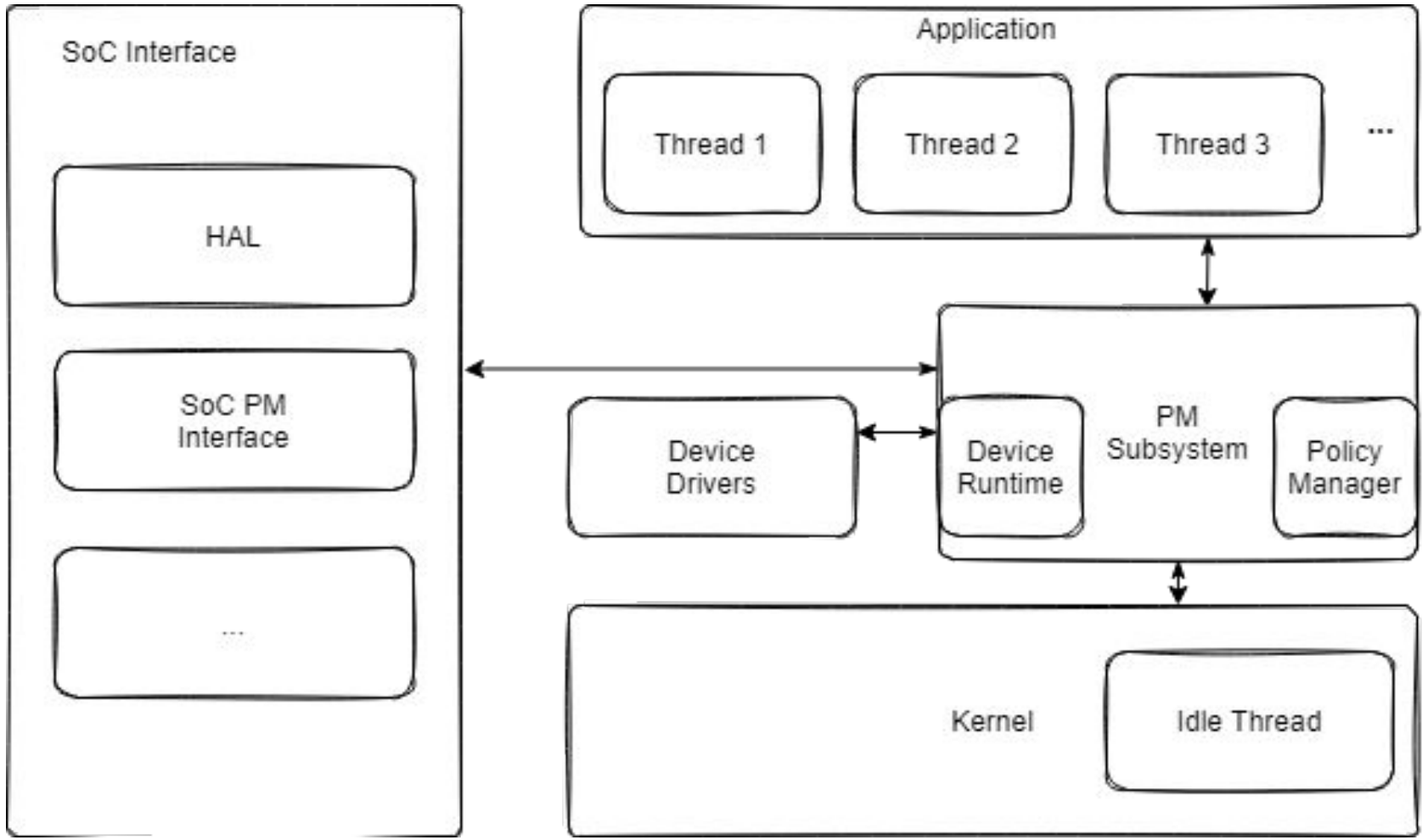
Zephyr Components



Key concepts

- Kernel **Idling**
- **System** PM - Saves energy changing the SoC power state
  - **Policy Manager**
  - Power state **constraints**
- **Device** PM – Saves energy suspending devices
  - **Runtime** PM
  - Device busy / idle

PM High-level Overview



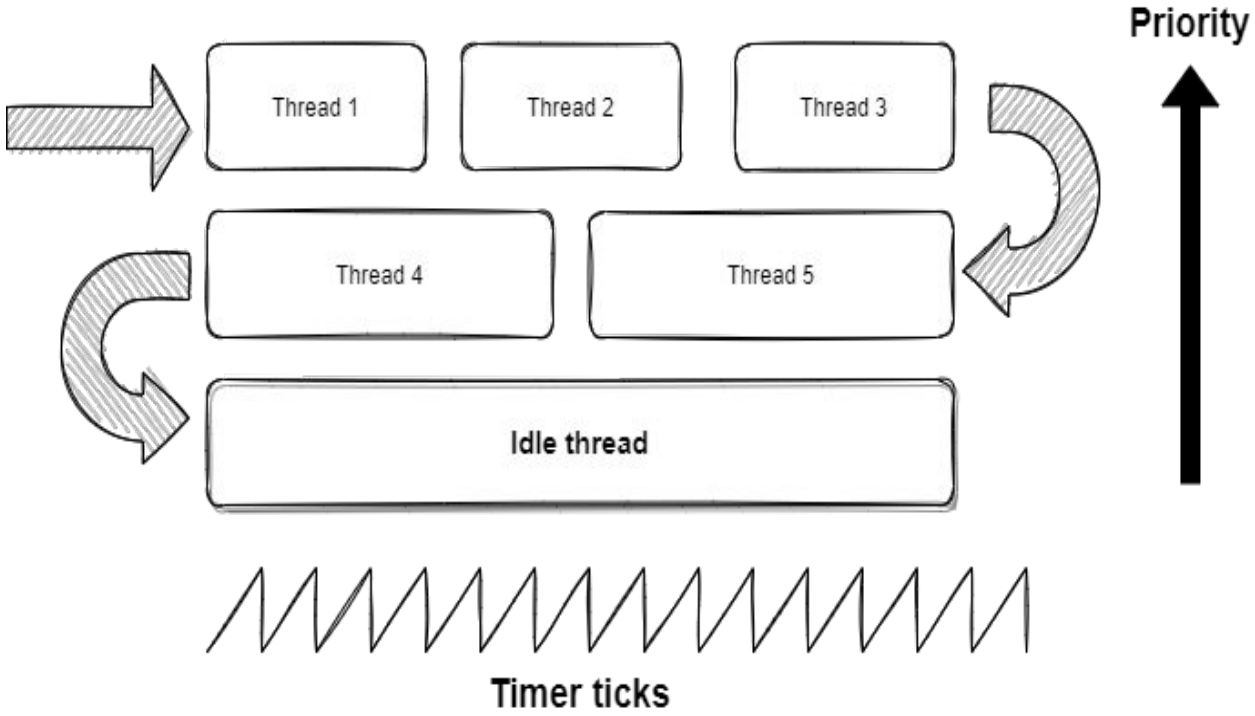
Zephyr™ Project
Developer Summit

Power Management on Zephyr

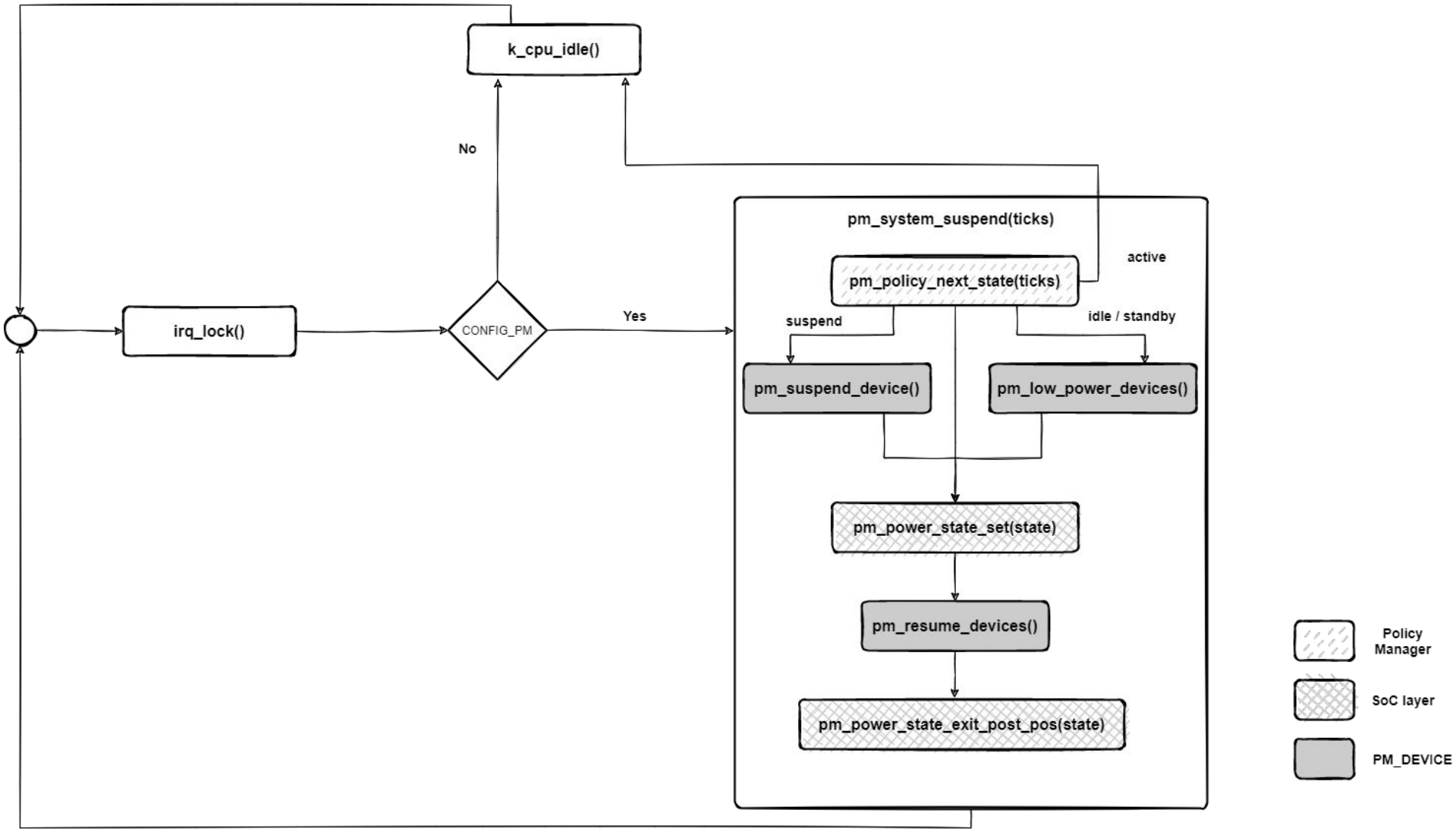
(System PM)

System IDLE

- Governed by the **idle** thread
  - Idle thread is the lowest priority thread and is executed when no other thread can run
  - Idle thread is scheduled again if no other thread is ready to run
- Tickless kernel
  - Interruptions only for registered events



Inside Idle thread



Power States

- States are **ACPI alike** and defined in devicetree
  - PM_STATE_ACTIVE
  - PM_STATE_RUNTIME_IDLE
  - PM_STATE_SUSPEND_TO_IDLE
  - PM_STATE_STANDBY
  - PM_STATE_SUSPEND_TO_RAM
  - PM_STATE_SUSPEND_TO_DISK
  - PM_STATE_SOFT_OFF
- Targets can declare only states that are meaningful for them
- It is fully customizable

```
compatible: "zephyr,power-state"

properties:
    power-state-name:
        type: string
        required: true
        description: indicates a power state
        enum:
            - "active"
            - "runtime-idle"
            - "suspend-to-idle"
            - "standby"
            - "suspend-to-ram"
            - "suspend-to-disk"
            - "soft-off"
    substate-id:
        type: int
        required: false
        description: Platform specific identification.
    min-residency-us:
        type: int
        required: false
        description: |
            Minimum residency duration in microseconds. It is the minimum time for a
            given idle state to be worthwhile energywise. It includes the time to enter
            in this state.
    exit-latency-us:
        type: int
        required: false
        description: |
            Worst case latency in microseconds required to exit the idle state.
```

Policy Manager

- Choose the best state to the system for a given idle period
- Applications can define their own policy
- Zephyr comes with residency policy as default

```
struct pm_state_info pm_policy_next_state(int32_t ticks);
```

Extra notes on System PM

- Applications can set constraints in power states to inhibit the system to go to a particular state
- Devices are suspended and resumed **sequentially and synchronous** according to their dependencies
  - Right now the **time** spent in this task **is no accounted**
- Devices set busy are not suspended
- The system accounts the time necessary to the system wakes
- Notifications about power state changes are delivered just before and after the system changes the state
- Application is responsible to set up a wake up event

Zephyr™ Project
Developer Summit

Power Management on Zephyr

(Device PM)

Key concepts

- There are two kinds of device power management:
  1. Central - The system suspends or resumes devices during system power management
  2. Runtime - Devices are suspended or resumed automatically according with their usage
- Devices are suspended and resumed according with their dependencies
- Drivers maintain power states per device
- Device driver PM interface
  - IOCTL style function implemented by drivers
    - Control codes PM_DEVICE_STATE_SET and PM_DEVICE_STATE_GET

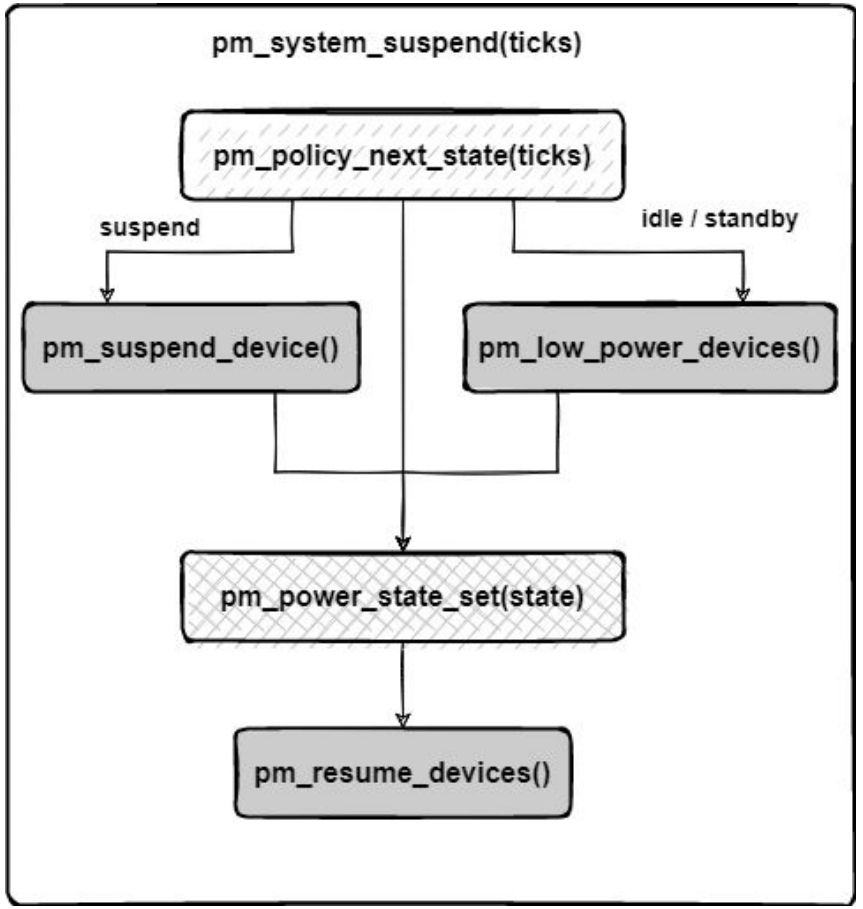
Central Device PM


```
void pm_suspend|resume_devices(void) {
    /*
     * iterate over devices and check if they
     * are not busy
     */
     ...

     /*
      * Wrapper for device driver interface.
      * device->pm_control(device, PM_DEVICE_STATE_SET ...)
      */
    pm_device_state_set (device, state, NULL, NULL);
    ...
}
```

```
/* device states */

PM_DEVICE_STATE_ACTIVE
PM_DEVICE_STATE_LOW_POWER
PM_DEVICE_STATE_SUSPEND
PM_DEVICE_STATE_FORCE_SUSPEND
PM_DEVICE_STATE_OFF
PM_DEVICE_STATE_RESUMING
PM_DEVICE_STATE_SUSPENDING
```

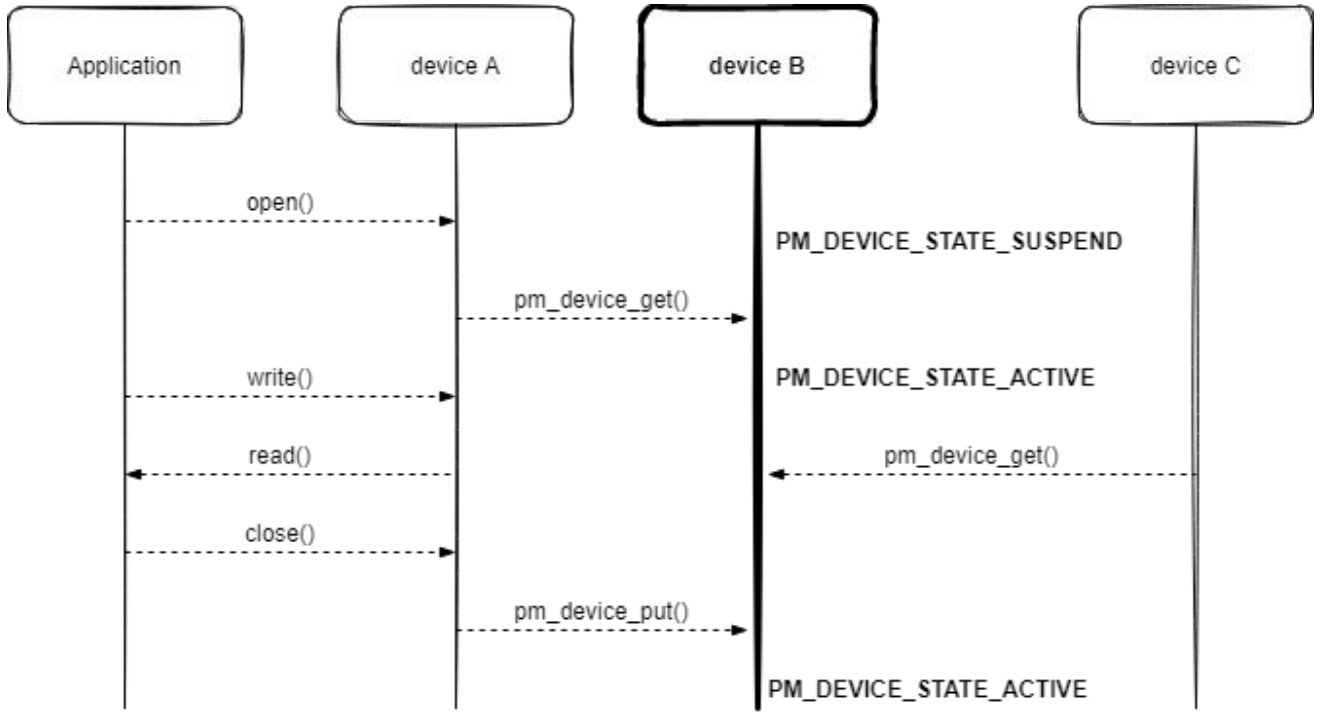


Device runtime PM

- Devices are suspend / resumed in runtime
  - Independent of the system PM
    - System does need to be IDLE !
- Synchronous and Asynchronous APIs

```
int pm_device_get_async(const struct device *dev);
int pm_device_get(const struct device *dev);

int pm_device_put(const struct device *dev);
int pm_device_put_async(const struct device *dev);
```

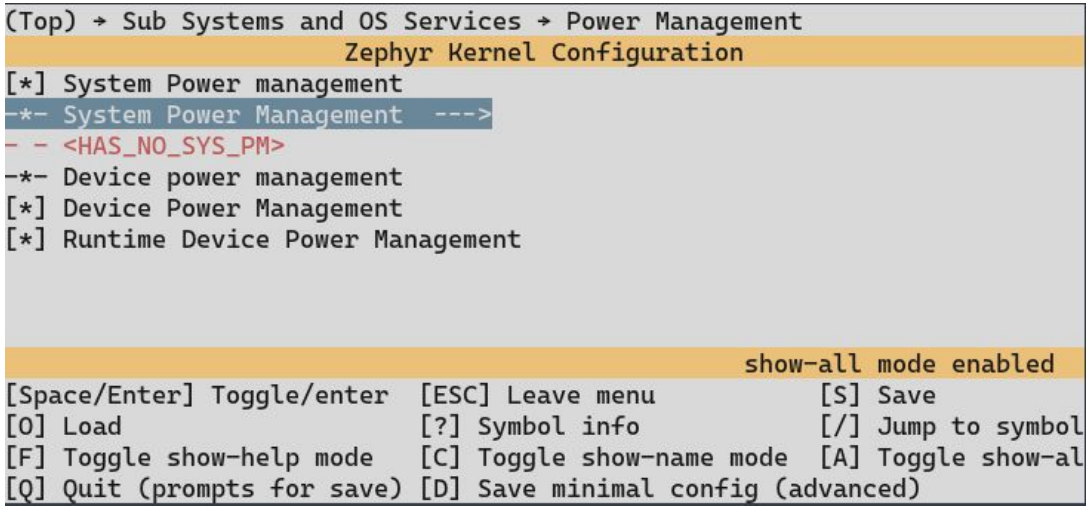


Zephyr™ Project
Developer Summit

Enabling PM

Using PM

- CONFIG_PM
  - Enable System PM.
- CONFIG_PM_DEVICE
  - Enable suspend and resume devices when the system is IDLE.
- CONFIG_PM_DEVICE_RUNTIME
  - Devices suspend / resumed in runtime



- **They can be used together !**

Implementing PM – SoC layer

- Implement SoC hooks:

```
void pm_power_state_set(struct pm_state_info info);
```

  - This function is called with interruptions locked and with the power state selected by the policy manager.
    void pm_power_state_exit_post_ops(struct pm_state_inf o info);

```
void pm_power_state_exit_post_ops(struct pm_state_info info);
```

  - It is called in the end of the IDLE thread loop. Zephyr expects interruptions to be restored after this function.

- Declare supported power states. E.g:

```
power-states {
        idle: idle {
                compatible = "zephyr,power-state";
                power-state-name = "suspend-to-idle";
                min-residency-us = <1000>;
        };

        standby: standby {
                compatible = "zephyr,power-state";
                power-state-name = "standby";
                min-residency-us = <5000>;
        };
}
```

Implementing PM - Device

- Device drivers must implement

```
int (*pm_control)(const struct device *dev, uint32_t command,
                  uint32_t *state, pm_device_cb cb, void *arg);
```

```
#define DEVICE_DEFINE(dev_name, drv_name, init_fn, pm_control_fn,    \
                      data_ptr, cfg_ptr, level, prio, api_ptr)       \
    Z_DEVICE_DEFINE(DT_INVALID_NODE, dev_name, drv_name, init_fn,    \
                    pm_control_fn,                                   \
                    data_ptr, cfg_ptr, level, prio, api_ptr)
```

```
static int dummy_device_pm_ctrl(const struct device *dev,
                                uint32_t ctrl_command,
                                uint32_t *state, pm_device_cb cb, void *arg)
{
    int ret = 0;

    switch (ctrl_command) {
    case PM_DEVICE_STATE_SET:
        if (*state == PM_DEVICE_STATE_ACTIVE) {
            ret = dummy_resume_from_suspend(dev);
        } else {
            ret = dummy_suspend(dev);
        }
        break;
    case PM_DEVICE_STATE_GET:
        *state = dummy_get_power_state(dev);
        break;
    default:
        ret = -EINVAL;

    }

    cb(dev, ret, state, arg);

    return ret;
}

DEVICE_DEFINE(dummy_driver, DUMMY_DRIVER_NAME, &dummy_init,
              dummy_device_pm_ctrl, NULL, NULL, APPLICATION,
              CONFIG_KERNEL_INIT_PRIORITY_DEFAULT, &funcs);
```

Implementing - Policy Manager

- The policy just need to implement the following function:

```
/**
 * Function to get the next PM state based on the ticks
 *
 * ticks -> the number of ticks to next scheduled event
 */
struct pm_state_info pm_policy_next_state(int32_t ticks);
```

- The state returned by the policy manager will be used by the system to figure out whether devices must be suspended and is passed to **SoC layer**
  - There are helpers to get power states declared on **Devicetree**

Zephyr™ Project
Developer Summit

Conclusions

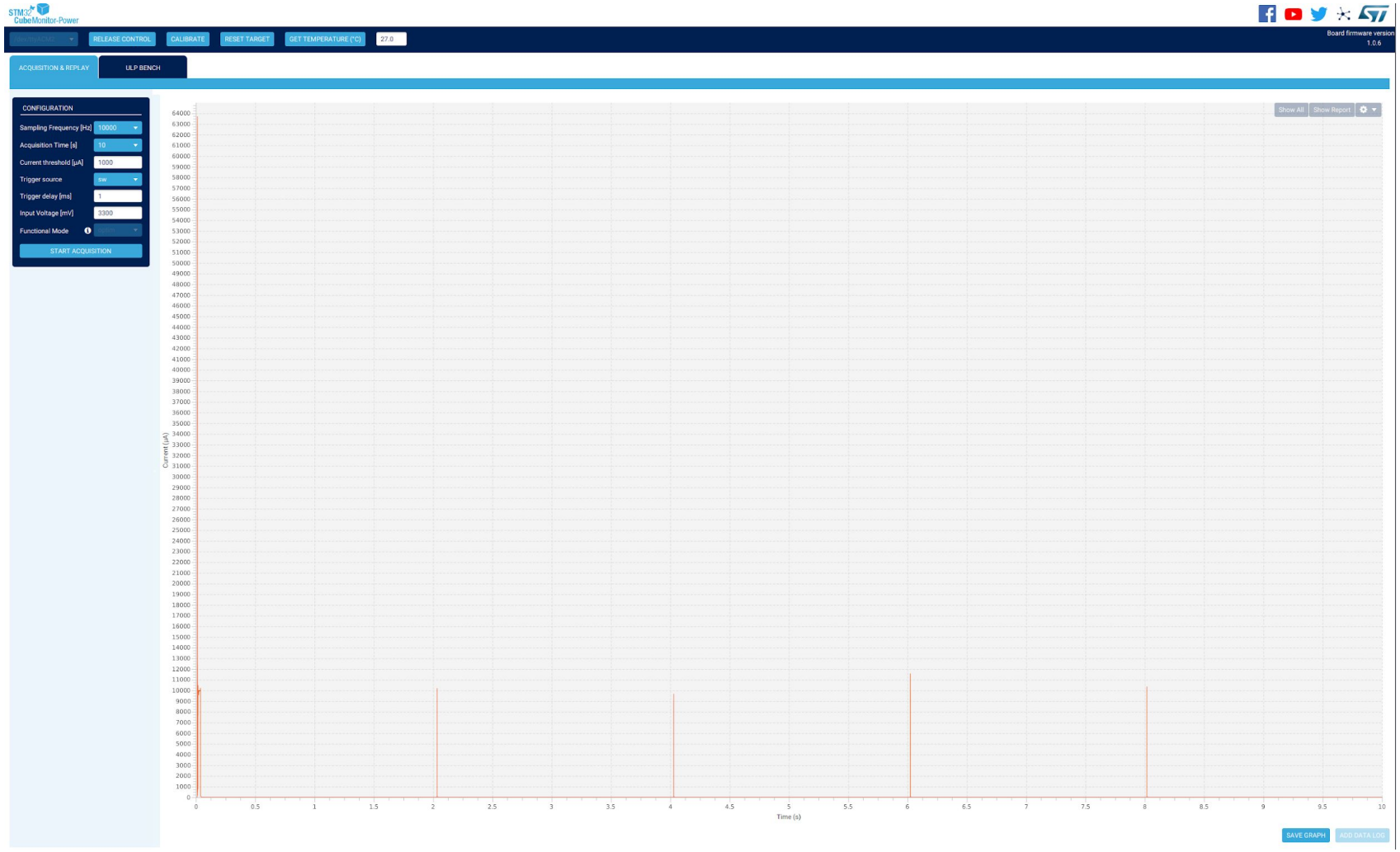
Conclusions

- Highly customizable. Applications can tune every single bit on Zephyr PM:
  1. Customize power states
  2. Redefine the policy, defining its own or through constraints usage
  3. Override SoC implementation
  4. Enable / Disable device PM
- Cross platform architecture/board/SoC → great portability
- Offers great savings when combined
  - Still f**ew devices support PM** though !

Conclusions

./samples/boards/stm32/power_mgmt/blinky

stm32l562e_dk



Zephyr™ Project
Developer Summit

Thank you !

Questions ?