Zephyr® Project
Developer Summit

intel®

Exploring Zephyr Power Management

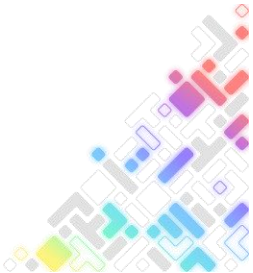
Overview and Implementation Guide

THE LINUX FOUNDATION

#EmbeddedOSSummit @ceolin

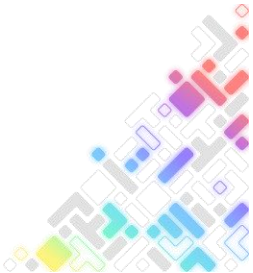
Why ?

- Extended battery Life
- Cost Savings
- Environmental Impact
- Regulatory Compliance
- User Experience
- …



Agenda

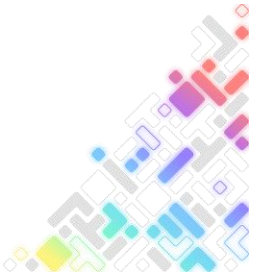
- Design and objectives
- Overview
- System Power Management
- Device Power Management
  - System Managed Device PM
  - Device Runtime PM
  - Power Domain
- Enabling and using Power Management



Design and objectives

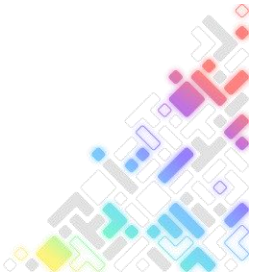
- **Reduce power consumption**, consume as little power as possible for a given use case and configuration.
- **Flexible configuration**, provide flexibility to suit specific requirements of an application and hardware platform
- **Maintain responsiveness**, the system must remain responsive to events and interrupts
- **Scalability**, scale from simple low-power devices to complex systems

- It is based on a collaborative model where different components like device drivers, subsystems and applications work together to manage power consumption



Key Concepts

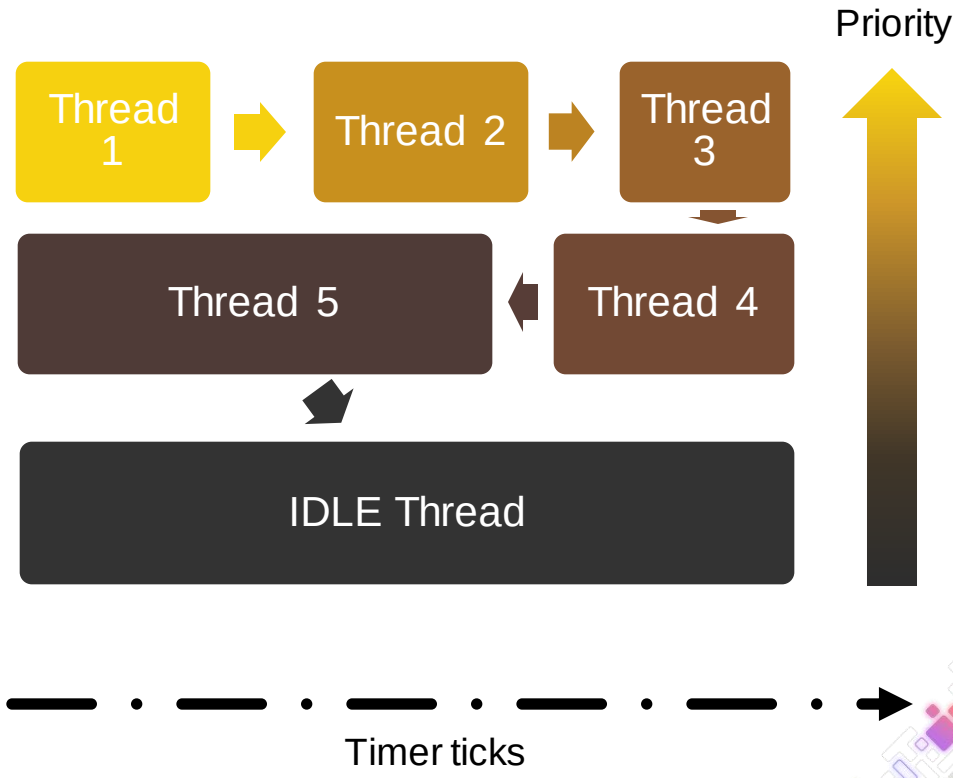
- Kernel Idling
  - Tickless Kernel
- System PM
  - Power States (idle / sleep states)
  - Policy Manager
- Device PM
  - System-Managed Device Power Management
  - Device Runtime Power Management



System Power Management

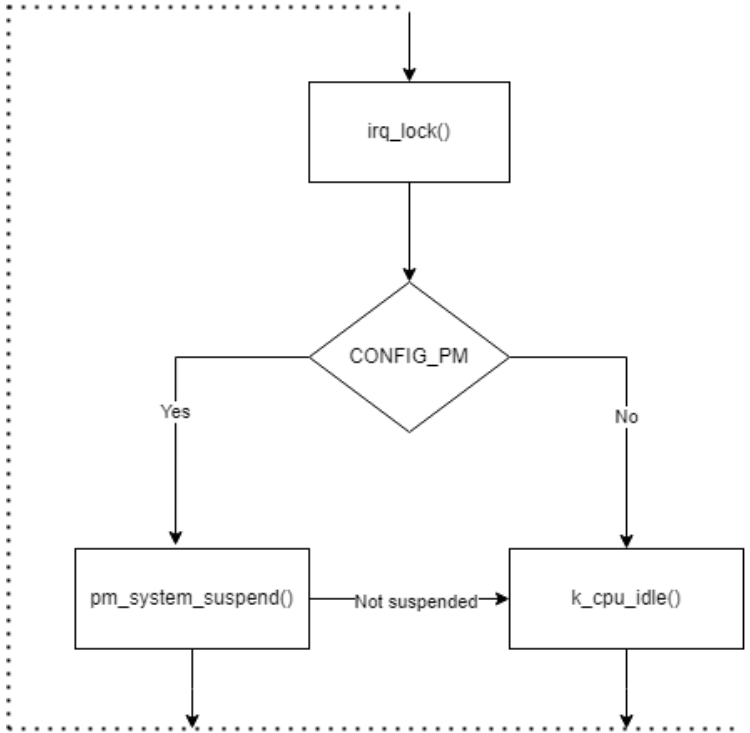
System PM

- 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



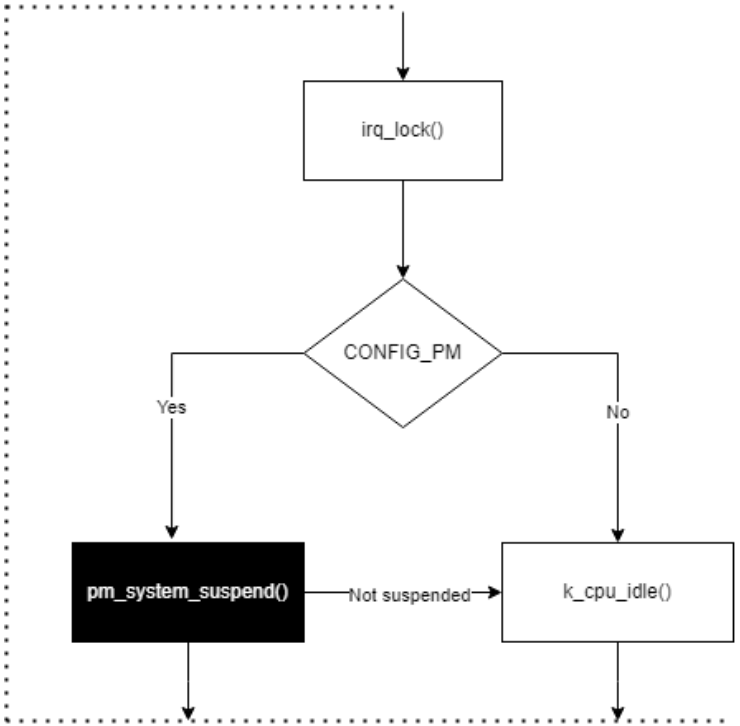
Inside IDLE thread

- IDLE thread loop
- **pm_system_suspend()** and **k_cpu_idle()** are called with local interruptions masked. These functions are responsible to re-enable interruption before return



Inside IDLE thread

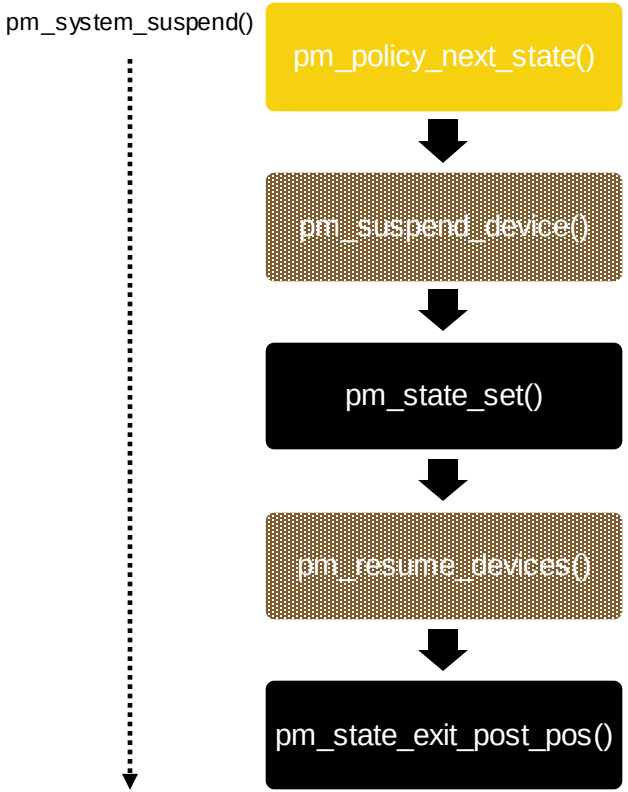
- IDLE thread loop
- **pm_system_suspend()** and **k_cpu_idle()** are called with local interruptions masked. These functions are responsible to re-enable interruption before return



PM subsystem

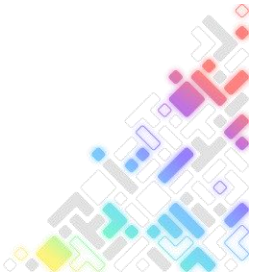
System Power Management

- Checks with the policy manager which power state should be used based on number of ticks for the next scheduled event
- Based on the power state selected may suspend devices
- Calls the SoC PM hook (**pm_state_set()**) with the selected state
  - The hook is called by the CPU that is being suspended



Power States

- They are declared per cpu in DT
- The policy manager is responsible to choose the most appropriated state based on the idle time until the next scheduled event
- The selected state is provided to the SoC hook that is responsible to take the correspondent action
- It is up to the target to implement which states its support
- High-level map to ACPI power state S0…S5



Power States

```
enum pm_state {
    /**
     * ...
     * @note This state is correlated with ACPI G0/S0 state
     */
    PM_STATE_ACTIVE,
    /**
     * ...
     * @note This state is correlated with ACPI S0ix state
     */
    PM_STATE_RUNTIME_IDLE,
    /**
     * ...
     * @note This state is correlated with ACPI S1 state
     */
    PM_STATE_SUSPEND_TO_IDLE,
    /**
     * ...
     * @note This state is correlated with ACPI S2 state
     */
    PM_STATE_STANDBY,
    /**
     * ...
     *
     * @note This state is correlated with ACPI S3 state
     */
    PM_STATE_SUSPEND_TO_RAM,
    /**
     * ...
     * @note This state is correlated with ACPI S4 state
     */
    PM_STATE_SUSPEND_TO_DISK,
    /**
     * ...
     * @note This state is correlated with ACPI G2/S5 state
     */
    PM_STATE_SOFT_OFF,
};
```

```
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
    description: Platform specific identification.
  min-residency-us:
    type: int
    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
    description: |
        Worst case latency in microseconds required to exit the idle state.
```

Policy Manager

- The default policy is based on residency time declared in power states
- Accounts constraints imposed by the application and drivers
- Honors latencies requested

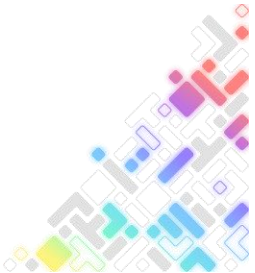
- Applications can define their own policy
  - **CONFIG_PM_POLICY_CUSTOM=y**

- **bool pm_state_force();** /* Bypass the policy decision */

Device Power Management

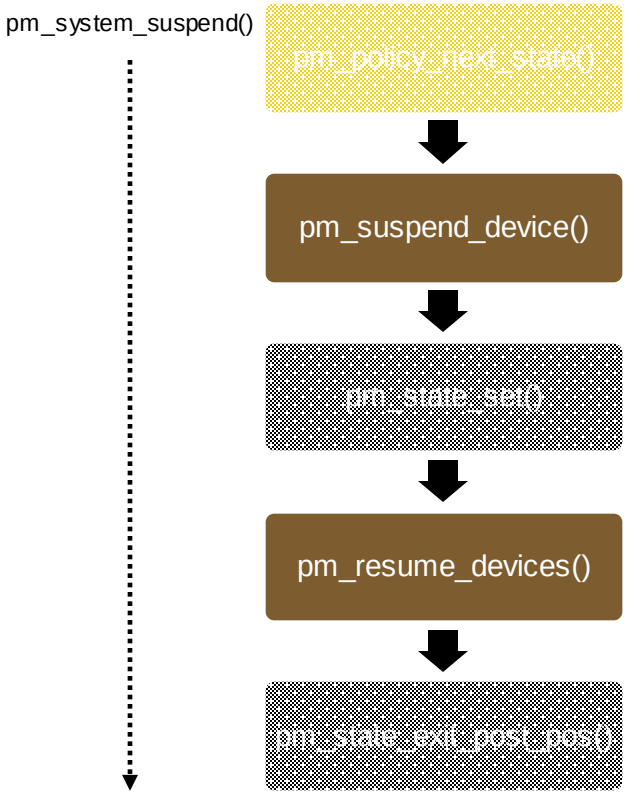
Device Power Management

- Enabled with **CONFIG_DEVICE_PM=y**
- Drivers that support PM just need to implement a callback
  - It is also used to be informed when a power domain it belongs is suspend / resumed
- Static / Dynamic Power Management
  - Devices are suspended when the system sleeps (System Managed Device PM)
  - Devices are actively suspended when are not used (Device Runtime PM)



System Managed Device PM

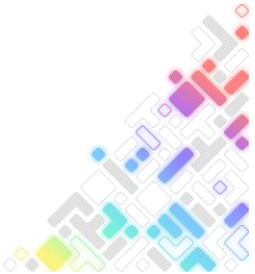
- Devices are suspended and resumed according with their dependencies
- Devices with runtime power management enabled are not touched



System Managed Device PM

```
/*
 * Ignore uninitialized devices, busy devices, wake up sources, and
 * devices with runtime PM enabled.
 */
if (!device_is_ready(dev) || pm_device_is_busy(dev) ||
    pm_device_wakeup_is_enabled(dev) ||
    pm_device_runtime_is_enabled(dev)) {
    continue;
}
```

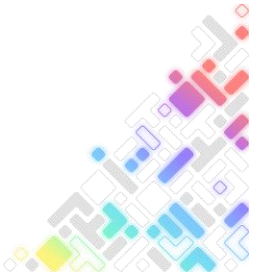
- **CONFIG_PM_NEED_ALL_DEVICES_IDLE=y**
  - Keeps the system active if any device is busy



Device Runtime Power Management

- Enabled with **CONFIG_PM_DEVICE_RUNTIME=y**
- Reduces the overall system power consumption by suspending devices that are not being used
  - System does not need to be IDLE
- Speed up system power management
- Synchronous and asynchronous operations

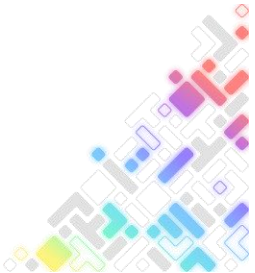
- Collaborative effort involving **device drivers**, **subsystems** and **applications**.



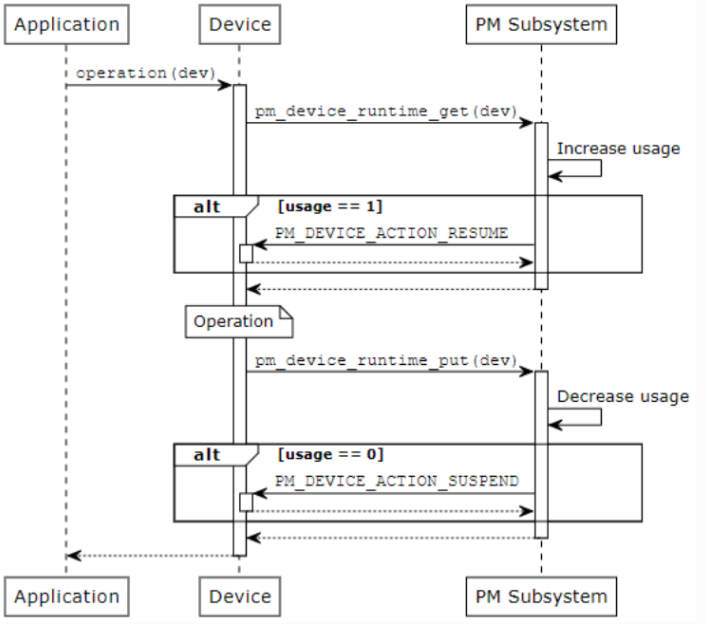
Device Runtime Power Management

- Enabled with **CONFIG_PM_DEVICE_RUNTIME=y**
- Reduces the overall system power consumption by suspending devices that are not being used
  - System does not need to be IDLE
- Speed up system power management
- Synchronous and asynchronous operations

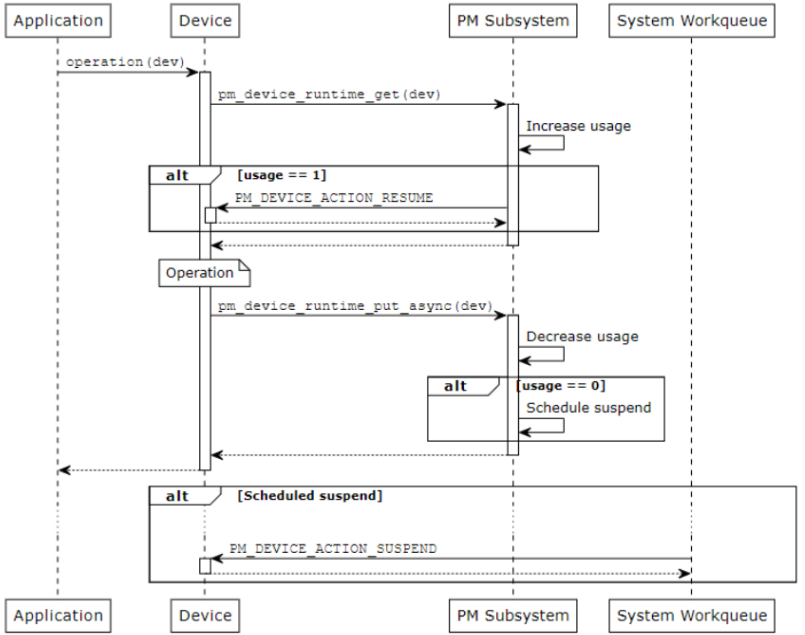
- Collaborative effort involving **device drivers**, **subsystems** and **applications**.



Device Runtime Power Management



sync

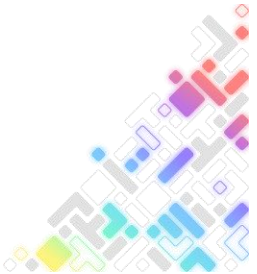


async

Power Domain

- Enabled with **CONFIG_PM_DEVICE_POWER_DOMAIN=y**
- Power domains are a special kind of device
  - They must be declared compatible with "power-domain" in DT or initialized with the flag **PM_DEVICE_FLAG_PD**
- They are responsible to notify their children when they are suspended and resumed

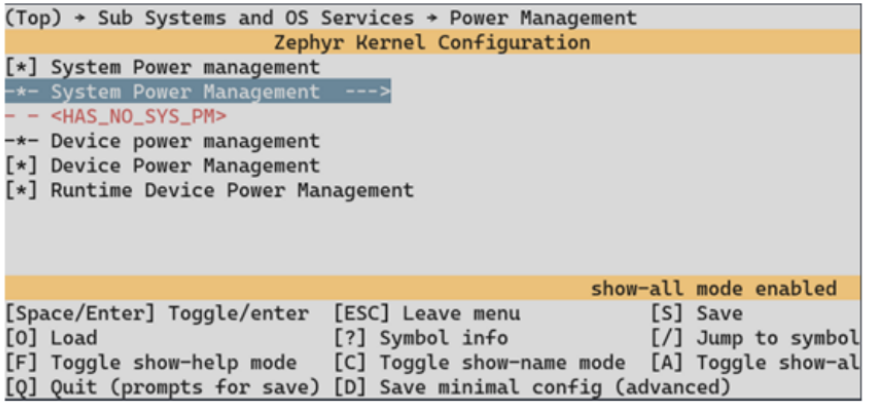
- Device runtime power management takes care to resume a power domain if a children device is resumed



Using and enabling PM

Using Power Management

- **CONFIG_PM**
  - Enable system power management
- **CONFIG_PM_DEVICE**
  - Enable System Managed Device PM
- **CONFIG_DEVICE_RUNTIME_PM**
  - Automatically suspend devices when they are not in use

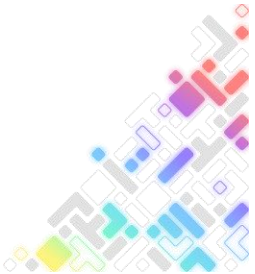


These options can be combined !

Implementing PM for a SoC

- Implement SoC APIs
  - **pm_state_set**(enum pm_state state, uint8_t substate_id);
  - **pm_state_exit_post_ops**(enum pm_state state, uint8_t substate_id);

- These functions are called by the IDLE thread with interruptions masked
- The power state is selected by the policy manager
- Interruptions must be restored in **pm_state_exit_post_ops()**



Implementing PM for a SoC

- Declare supported power states in DT

```
power-states {
        idle: idle {
                compatible = "zephyr,power-state";
                power-state-name = "suspend-to-idle";
                min-residency-us = <1000000>;
        };
        suspend_to_ram: suspend_to_ram {
                compatible = "zephyr,power-state";
                power-state-name = "suspend-to-ram";
                min-residency-us = <2000000>;
        };
};
cpu0: cpu@0 {
        device_type = "cpu";
        compatible = "arm,cortex-m4";
        reg = <0>;
        cpu-power-states = <&idle &suspend_to_ram>;
};
```

Implementing a custom Policy

CONFIG_PM_POLICY_CUSTOM=y

```
const struct pm_state_info *
    pm_policy_next_state(uint8_t cpu, int32_t ticks);
```

- Constraints, latency and events must be accounted by the policy
- States returned by the policy are given to **pm_state_set()**

```
const struct pm_state_info *pm_policy_next_state(uint8_t cpu, int32_t ticks)
{
    int64_t cyc = -1;
    uint8_t num_cpu_states;
    const struct pm_state_info *cpu_states;

#ifdef CONFIG_PM_NEED_ALL_DEVICES_IDLE
    if (pm_device_is_any_busy()) {
        return NULL;
    }
#endif

    if (ticks != K_TICKS_FOREVER) {
        cyc = k_ticks_to_cyc_ceil32(ticks);
    }

    num_cpu_states = pm_state_cpu_get_all(cpu, &cpu_states);

    if (next_event_cyc >= 0) {
        uint32_t cyc_curr = k_cycle_get_32();
        int64_t cyc_evt = next_event_cyc - cyc_curr;

        /* event happening after cycle counter max value, pad */
        if (next_event_cyc <= cyc_curr) {
            cyc_evt += UINT32_MAX;
        }

        if (cyc_evt > 0) {
            /* if there's no system wakeup event always wins,
             * otherwise, who comes earlier wins
             */
            if (cyc < 0) {
                cyc = cyc_evt;
            } else {
                cyc = MIN(cyc, cyc_evt);
            }
        }
    }

    for (int16_t i = (int16_t)num_cpu_states - 1; i >= 0; i--) {
        const struct pm_state_info *state = &cpu_states[i];
        uint32_t min_residency_cyc, exit_latency_cyc;

        /* check if there is a lock on state + substate */
        if (pm_policy_state_lock_is_active(state->state, state->substate_id)) {
            continue;
        }

        min_residency_cyc = k_us_to_cyc_ceil32(state->min_residency_us);
        exit_latency_cyc = k_us_to_cyc_ceil32(state->exit_latency_us);

        /* skip state if it brings too much latency */
        if ((max_latency_cyc >= 0) &&
            (exit_latency_cyc >= max_latency_cyc)) {
            continue;
        }

        if ((cyc < 0) ||
            (cyc >= (min_residency_cyc + exit_latency_cyc))) {
            return state;
        }
    }

    return NULL;
}
```

Implementing PM in a device

CONFIG_PM_DEVICE=y

```
PM_DEVICE_DEFINE(dummy_driver, dummy_device_pm_action);

DEVICE_DEFINE(dummy_driver, DUMMY_DRIVER_NAME, &dummy_init,
            PM_DEVICE_GET(dummy_driver), NULL, NULL, POST_KERNEL,
            CONFIG_KERNEL_INIT_PRIORITY_DEVICE, &funcs);
```

If device belongs to a power domain:

```
int pm_device_power_domain_add(const struct device *dev,
                const struct device *domain)
```

```
static int dummy_device_pm_action(const struct device *dev,
                    enum pm_device_action action)
{
    switch (action) {
    case PM_DEVICE_ACTION_RESUME:
        /* resume device */
        break;
    case PM_DEVICE_ACTION_SUSPEND:
        /* suspend device */
        break;
    case PM_DEVICE_ACTION_TURN_OFF:
        /* Power domain this device is in  was suspended */
        break;
    case PM_DEVICE_ACTION_TURN_ON:
        /* Power domain this device is in  was resumed */
        break;
    default:
        return -ENOTSUP;
    }

    return 0;
}
```

Device Runtime PM

CONFIG_PM_DEVICE_RUNTIME=y

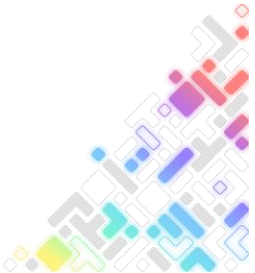
```
int pm_device_runtime_enable(const struct device *dev);

int pm_device_runtime_disable(const struct device *dev);

int pm_device_runtime_get(const struct device *dev);

int pm_device_runtime_put(const struct device *dev);

int pm_device_runtime_put_async(const struct device *dev,
        k_timeout_t delay);
```

Implementing a Power Domain

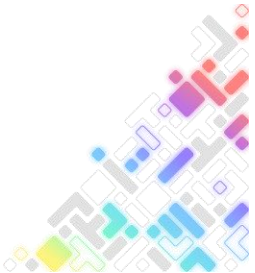
CONFIG_PM_DEVICE_POWER_DOMAIN=y

```
static int domain_pm_action(const struct device *dev,
    enum pm_device_action action)
{
    int rc = 0;
    switch (action) {
    case PM_DEVICE_ACTION_RESUME:
        /* Switch power on */
        pm_device_children_action_run(dev, PM_DEVICE_ACTION_TURN_ON, NULL);
        break;
    case PM_DEVICE_ACTION_SUSPEND:
        pm_device_children_action_run(dev, PM_DEVICE_ACTION_TURN_OFF, NULL);
        break;
    case PM_DEVICE_ACTION_TURN_ON:
        __fallthrough;
    case PM_DEVICE_ACTION_TURN_OFF:
        break;
    default:
        rc = -ENOTSUP;
    }
    return rc;
}
```

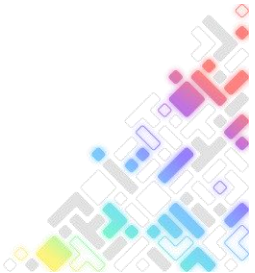
```
test_domain: test_domain {
        compatible = "power-domain";
        status = "okay";
};
```

Much more …

- Device PM
  - Set device busy
    - **CONFIG_PM_NEED_ALL_DEVICES_IDLE**
  - Set initial device state
  - Wake-up source
- Tunning pm policy
  - Add / remove latency requirements
  - Add / remove events
- …

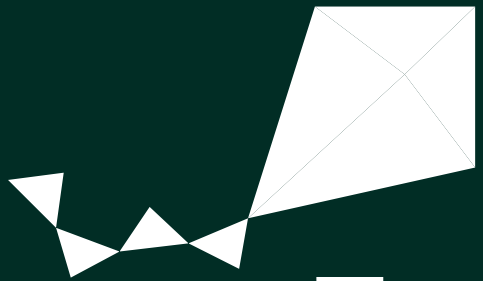


Questions ?



Thank you !

Flavio Ceolin
flavio.ceolin@intel.com



Zephyr® Project

Developer Summit