Getting started on low power optimization on Zephyr

Tobias Meyer, Konux GmbH

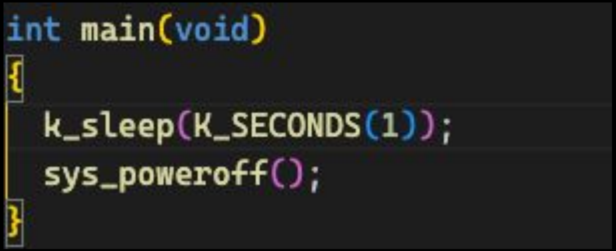
Tobias Meyer

- Engineering Manager @konux
- Firmware Developer over 15 years of experience
- Using zephyr nearly 3 years

Battery powered devices

Imagine ... you prepare the whole business logic and now the device is dead instant?

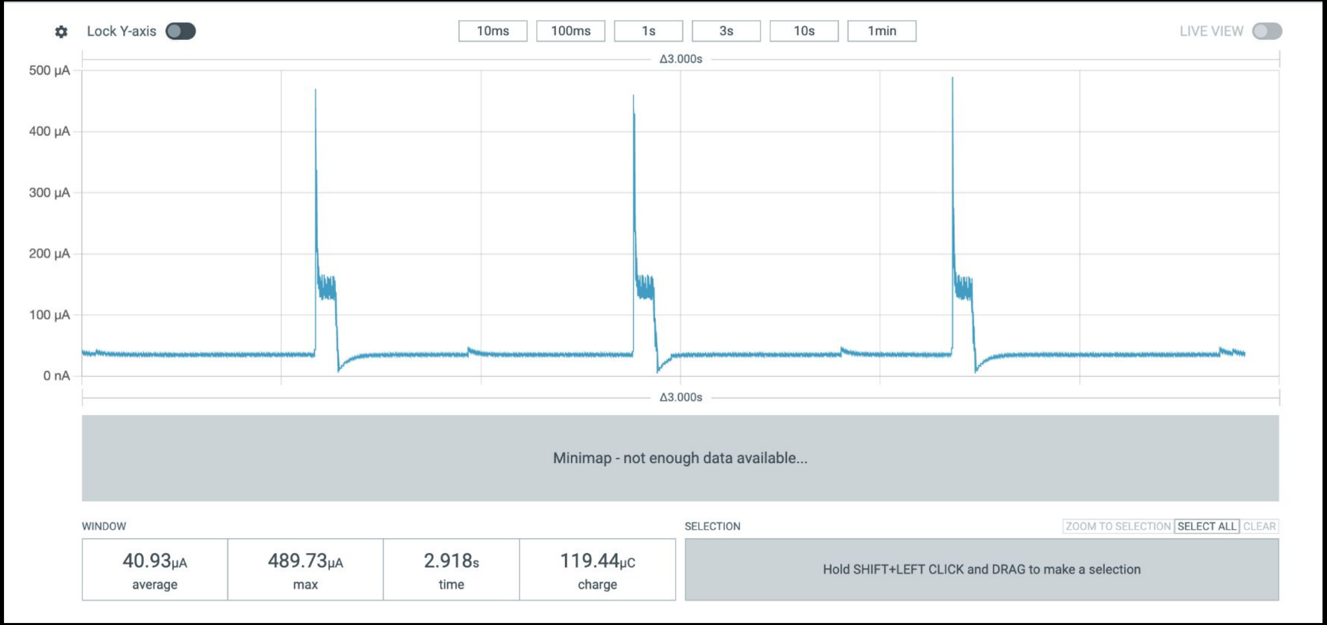
First step always check hardware and default logic behaves as it should ... (DTS & power off)



```c
int main(void)
{
  k_sleep(K_SECONDS(1));
  sys_poweroff();
}
```

`CONFIG_POWEROFF` needs to be enabled to use this API.

`CONFIG_PM` to check if low power sleep works



WINDOW

40.93µA	489.73µA	2.918s	119.44µC
average	max	time	charge

SELECTION

Hold SHIFT+LEFT CLICK and DRAG to make a selection

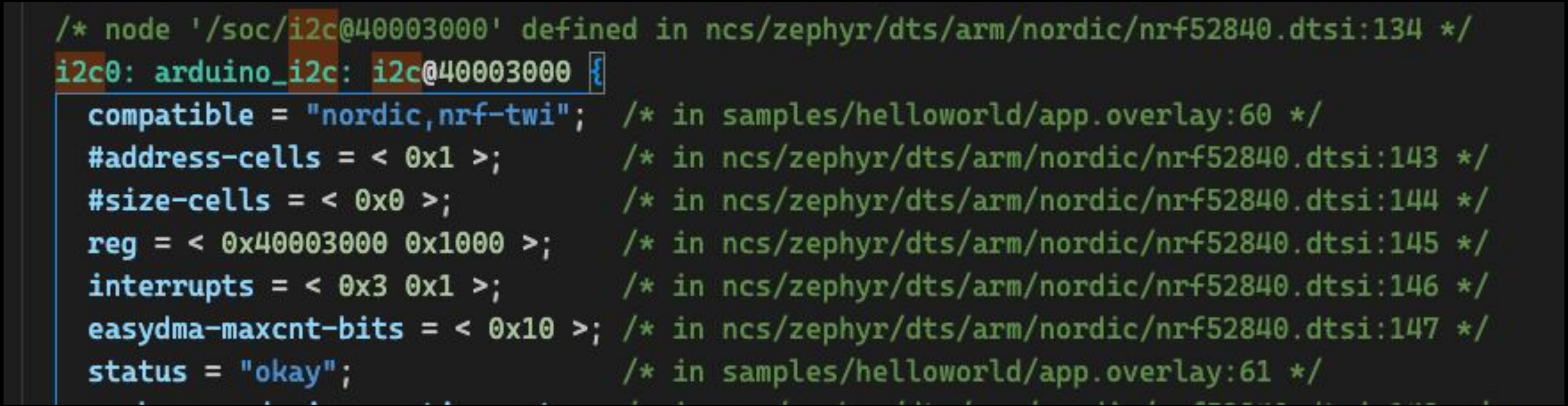
Strange peaks?

Issue was e.g. here CONFIG_SENSOR on & temperature sensor enabled

Issue was e.g. here CONFIG_SENSOR on & temperature sensor enabled

- Check ./{build}//zephyr/include/generated/zephyr/autoconf.h
  disable in your prj.conf e.g. CONFIG_SENSOR,CONFIG_LOG,CONFIG_UART (..)

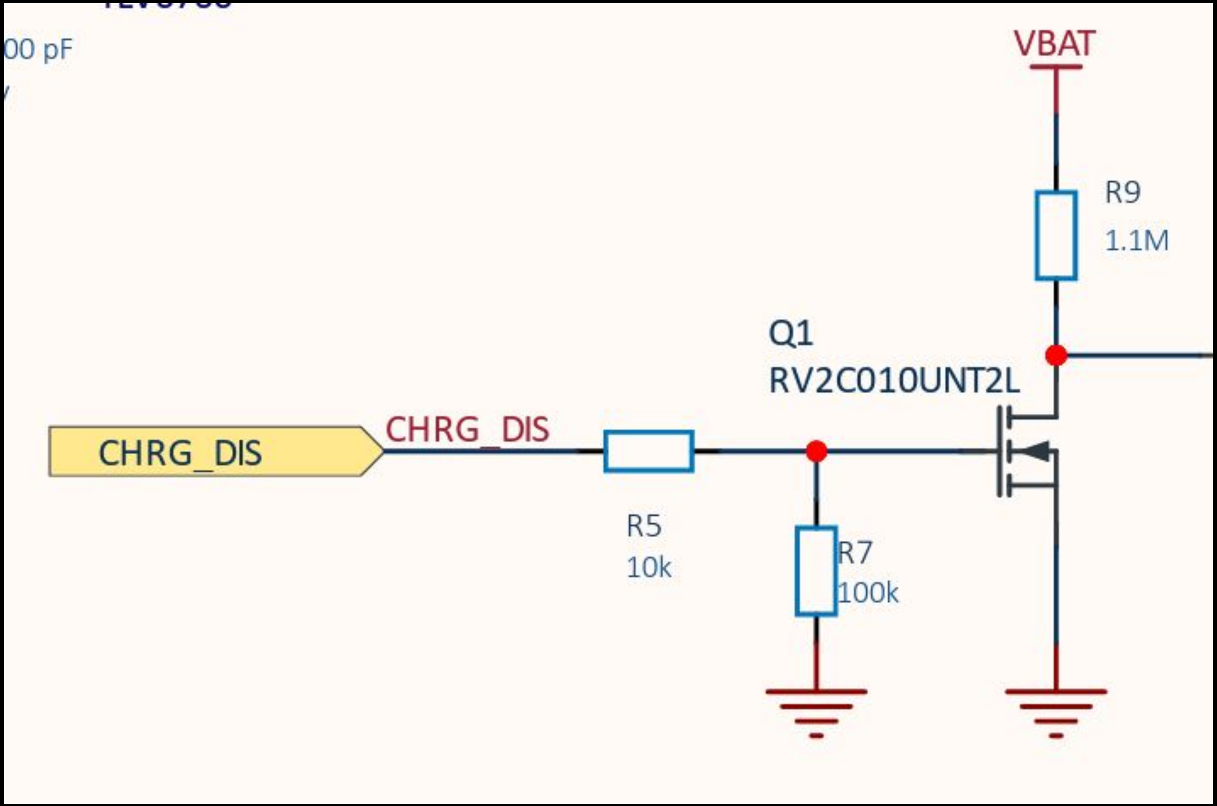
- Check ./{build}/zephyr/zephyr.dts



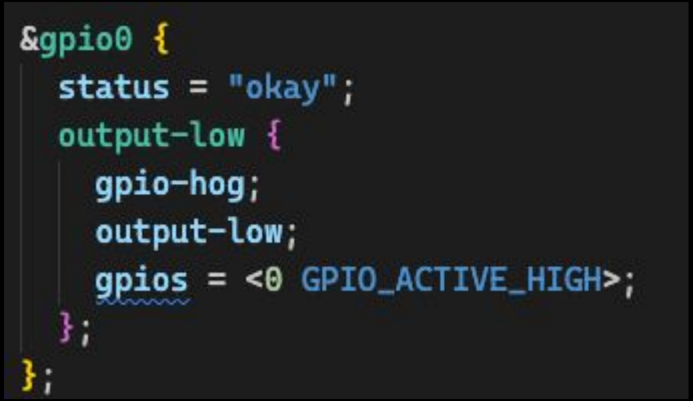
```
/* node '/soc/i2c@40003000' defined in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:134 */
i2c0: arduino_i2c: i2c@40003000 {
  compatible = "nordic,nrf-twi";  /* in samples/helloworld/app.overlay:60 */
  #address-cells = < 0x1 >;       /* in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:143 */
  #size-cells = < 0x0 >;          /* in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:144 */
  reg = < 0x40003000 0x1000 >;    /* in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:145 */
  interrupts = < 0x3 0x1 >;       /* in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:146 */
  easydma-maxcnt-bits = < 0x10 >; /* in ncs/zephyr/dts/arm/nordic/nrf52840.dtsi:147 */
  status = "okay";                /* in samples/helloworld/app.overlay:61 */
```

- Disable all peripheral devices

- GPIO config ensure Pullup & pulldown, input output as needed by your application



- GPIO config ensure Pullup & pulldown, input output as needed by your application (E.g. use hogs already for early start configs)



```
&gpio0 {
  status = "okay";
  output-low {
    gpio-hog;
    output-low;
    gpios = <0 GPIO_ACTIVE_HIGH>;
  };
};
```

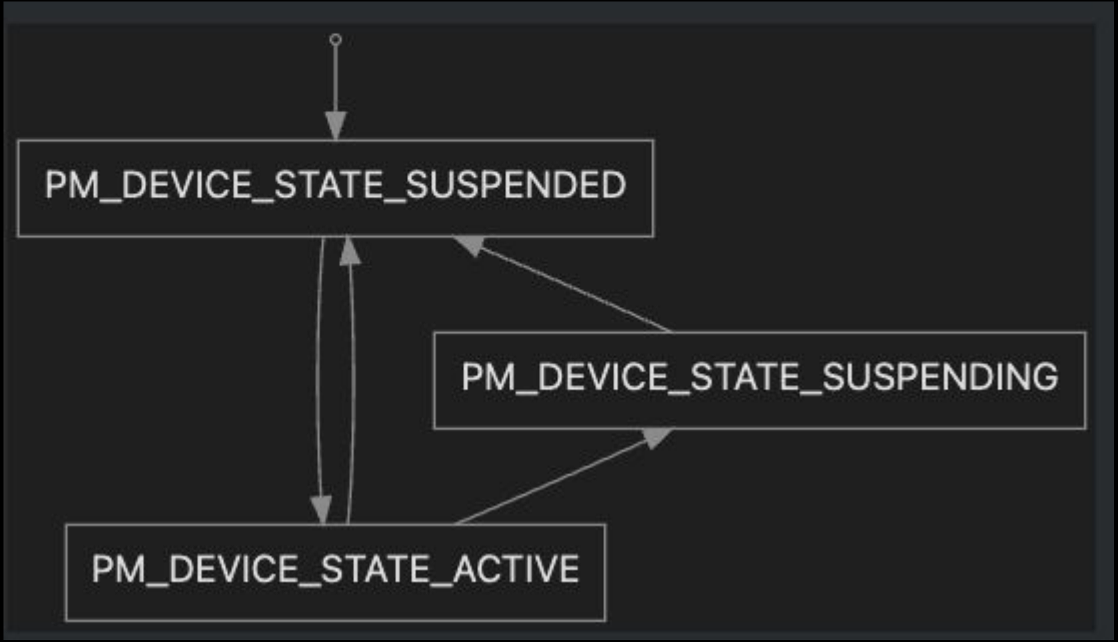
Device Runtime Power Management

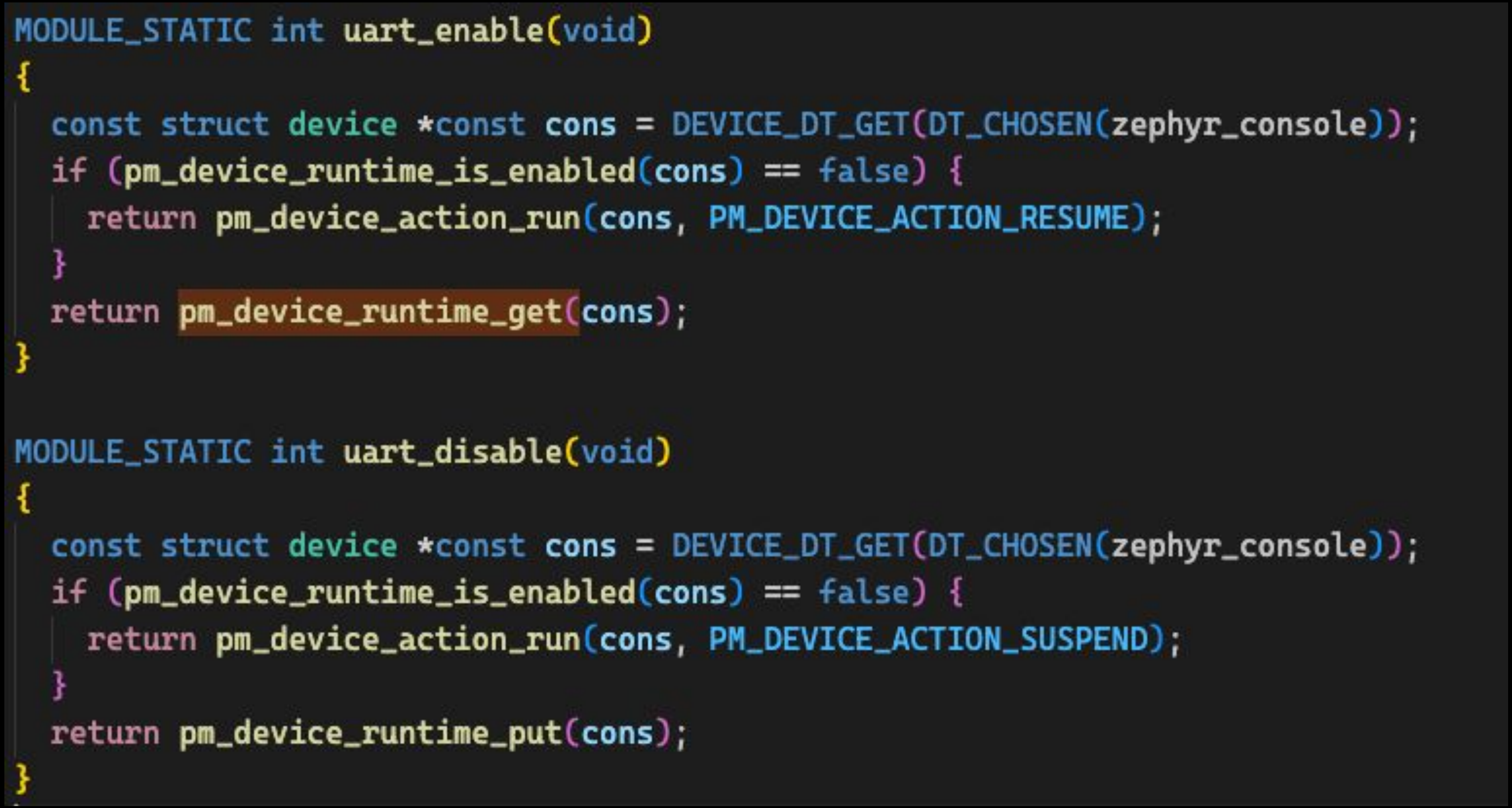
- Fine-grained, usage-based power control.
- Devices suspend/resume individually depending on real-time usage.
- Independent of CPU sleep states — saves power during normal operation.

System-Managed Device Power Management

- Devices are suspended/resumed as a group when the CPU enters/exits a low-power state.
- Simpler to use, but coarse-grained and less optimal during runtime.
- Device suspend timing limited by idle-thread context; may block low-power entry if busy.

- Enable CONFIG_PM_DEVICE_RUNTIME
- Add property `zephyr,pm-device-runtime-auto` or config `CONFIG_PM_DEVICE_RUNTIME_DEFAULT_ENABLE`
- Use **`pm_device_runtime_get()`** , **`pm_device_runtime_put()`**

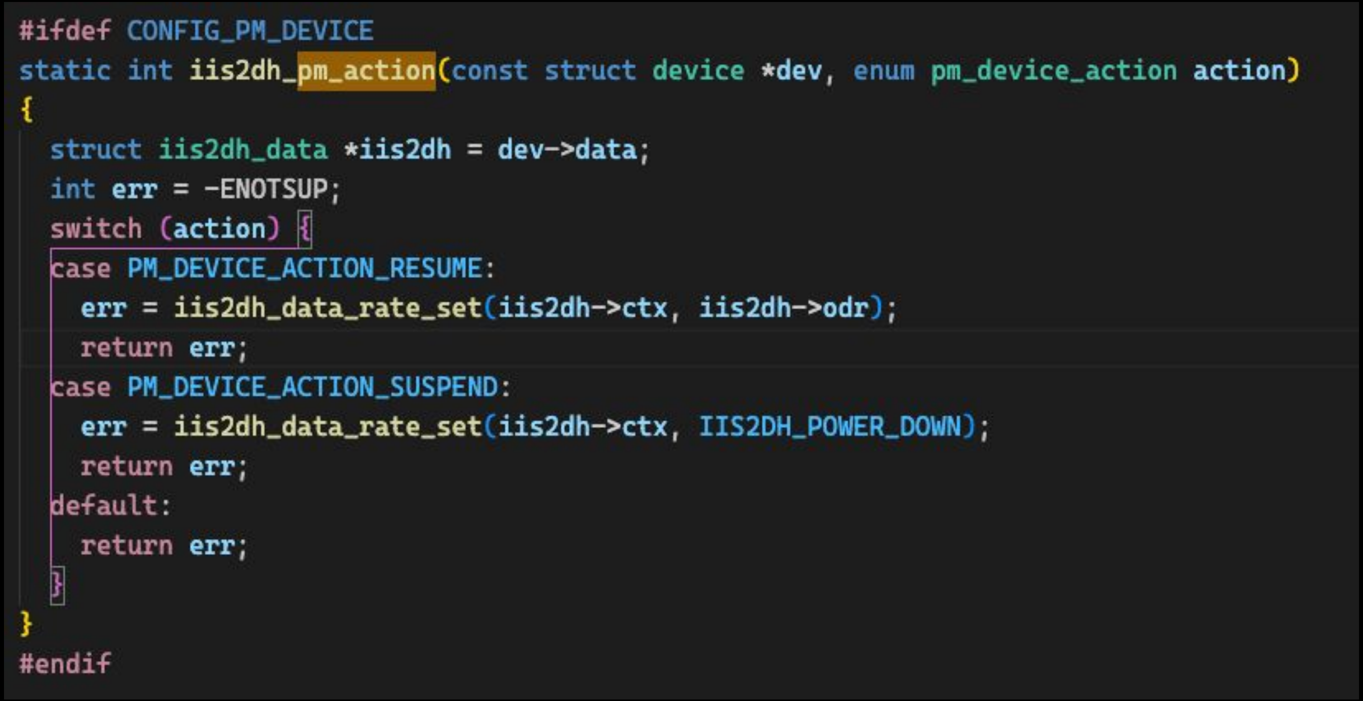




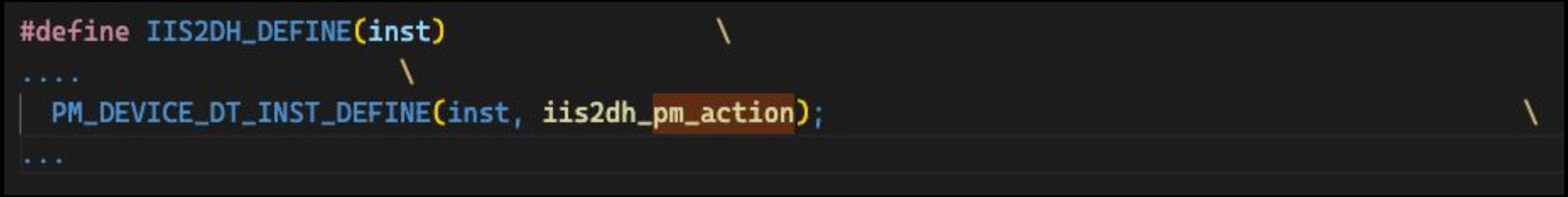
```c
MODULE_STATIC int uart_enable(void)
{
  const struct device *const cons = DEVICE_DT_GET(DT_CHOSEN(zephyr_console));
  if (pm_device_runtime_is_enabled(cons) == false) {
    return pm_device_action_run(cons, PM_DEVICE_ACTION_RESUME);
  }
  return pm_device_runtime_get(cons);
}

MODULE_STATIC int uart_disable(void)
{
  const struct device *const cons = DEVICE_DT_GET(DT_CHOSEN(zephyr_console));
  if (pm_device_runtime_is_enabled(cons) == false) {
    return pm_device_action_run(cons, PM_DEVICE_ACTION_SUSPEND);
  }
  return pm_device_runtime_put(cons);
}
```

Extend your driver



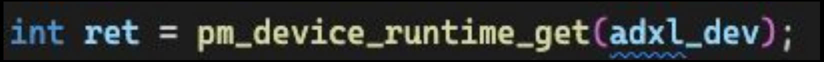
```c
#ifdef CONFIG_PM_DEVICE
static int iis2dh_pm_action(const struct device *dev, enum pm_device_action action)
{
  struct iis2dh_data *iis2dh = dev->data;
  int err = -ENOTSUP;
  switch (action) {
  case PM_DEVICE_ACTION_RESUME:
    err = iis2dh_data_rate_set(iis2dh->ctx, iis2dh->odr);
    return err;
  case PM_DEVICE_ACTION_SUSPEND:
    err = iis2dh_data_rate_set(iis2dh->ctx, IIS2DH_POWER_DOWN);
    return err;
  default:
    return err;
  }
}
#endif
```



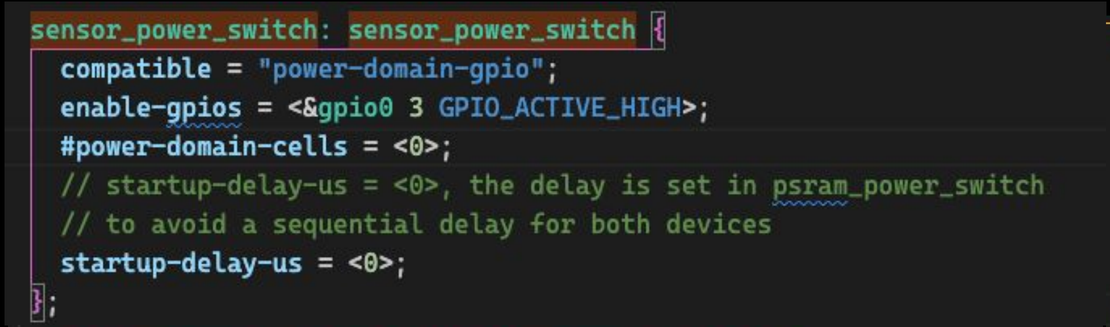
```c
#define IIS2DH_DEFINE(inst)                  \
....                        \
  PM_DEVICE_DT_INST_DEFINE(inst, iis2dh_pm_action);                              \
...
```

Power domains for dependency of power e.g. sensors behind a mosfet `CONFIG_PM_DEVICE_POWER_DOMAIN`

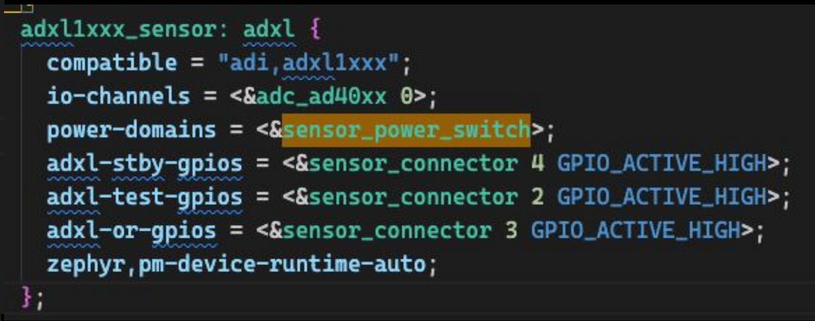
- You can even create dependency if power domains depend on each other



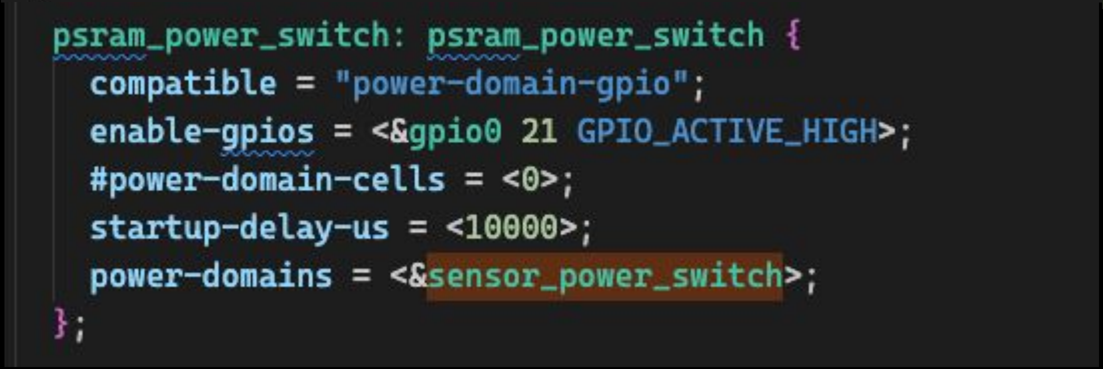
```c
int ret = pm_device_runtime_get(adxl_dev);
```



```
sensor_power_switch: sensor_power_switch {
  compatible = "power-domain-gpio";
  enable-gpios = <&gpio0 3 GPIO_ACTIVE_HIGH>;
  #power-domain-cells = <0>;
  // startup-delay-us = <0>, the delay is set in psram_power_switch
  // to avoid a sequential delay for both devices
  startup-delay-us = <0>;
};
```



```
adxl1xxx_sensor: adxl {
  compatible = "adi,adxl1xxx";
  io-channels = <&adc_ad40xx 0>;
  power-domains = <&sensor_power_switch>;
  adxl-stby-gpios = <&sensor_connector 4 GPIO_ACTIVE_HIGH>;
  adxl-test-gpios = <&sensor_connector 2 GPIO_ACTIVE_HIGH>;
  adxl-or-gpios = <&sensor_connector 3 GPIO_ACTIVE_HIGH>;
  zephyr,pm-device-runtime-auto;
};
```



```
psram_power_switch: psram_power_switch {
  compatible = "power-domain-gpio";
  enable-gpios = <&gpio0 21 GPIO_ACTIVE_HIGH>;
  #power-domain-cells = <0>;
  startup-delay-us = <10000>;
  power-domains = <&sensor_power_switch>;
};
```

Some easy summary

- Confirm with “off” that your DTS matches lowest power board config
- Try to keep device in idle mode as much as possible
- Use **PM_DEVICE** , **PM_DEVICE_RUNTIME** and **PM_DEVICE_POWER_DOMAIN** to configure system for low power
- Recommend use **pm_device_runtime_get** / put to enable/disable peripheral based on needs
- Use power domains if you need to use mosfet to disable e.g. a bunch of sensors
- Pro tip use a power profiler in your CI testing and check values regularly

Any questions?

https://www.linkedin.com/in/trmk/

https://www.trmk.de
https://github.com/tobiwan88

Contact